aws

开发人员指南

Amazon Lex V1

Amazon Lex V1: 开发人员指南

Table of Contents

终止支持通知：2025年9月15日， AWS 我们将停止对Amazon Lex V1的支持。2025 年 9 月 15 日之后，您将无法再访问亚马逊 Lex V1 主机或 Amazon Lex V1 资源。如果您使用的是 Amazon Lex V2，请改为参阅 Amazon Lex V2 指南。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。

什么是 Amazon Lex？

Amazon Lex 是一项 AWS 服务，用于使用语音和文本来为应用程序构建对话界面。借助 Amazon Lex，为 Amazon Alexa 提供技术支持的同一对话引擎现可供任何开发人员使用，从而使您能够在新的和现有的应用程序中构建高级的自然语言聊天机器人。Amazon Lex 具备自然语言理解 (NLU) 和自动语音识别 (ASR) 的深度功能性和灵活性，使您能够通过生动的对话互动构建高度参与的用户体验并创建新的产品类别。

借助 Amazon Lex，任何开发人员都能快速构建对话聊天机器人。借助 Amazon Lex，无需深度学习专业知识，您只需在 Amazon Lex 控制台中指定基本对话流程即可创建机器人。Amazon Lex 管理对话并在对话中动态调整响应。借助此控制台，您可构建、测试和发布您的文本或语音聊天自动程序。随后，您可将对话接口添加到移动设备、Web 应用程序和聊天平台 (例如，Facebook Messenger) 上的自动程序。

Amazon Lex 提供了与 AWS 平台上的许多其他服务的预建集成 AWS Lambda，包括亚马逊 Cognito、Amazon 和 Amazon Dyn AWS Mobile Hub amoDB，您可以轻松地与 AWS 平台上的许多其他服务集成。 CloudWatch与 Lambda 的集成使机器人能够访问预先构建的无服务器企业连接器，以链接到 Salesforce 或 Marketo 等 SaaS 应用程序中的数据。 HubSpot

使用 Amazon Lex 的一些好处包括：

- 简易性 — Amazon Lex 指导您使用控制台在几分钟内创建您自己的聊天机器人。您只需提供几个示例短语，Amazon Lex 即可构建完整的自然语言模型，机器人可通过此模型使用语音和文本进行交互来提问、回答和完成复杂的任务。

- 大众化的深度学习技术 — 由与 Alexa 相同的技术提供技术支持，Amazon Lex 提供了 ASR 和 NLU 技术来创建语音语言理解 (SLU) 系统。借助 SLU，Amazon Lex 采用自然语言语音和文本输入，理解输入背后的意图，并通过调用相应的业务功能来实现用户意图。

  语音识别和自然语言理解是在计算机科学领域内要解决的部分最具挑战性的难题，这需要基于大量数据和基础设施来展开尖端的深度学习算法训练。Amazon Lex 推出了所有开发人员都能接触到的深度学习技术，由与 Alexa 相同的技术提供支持。Amazon Lex 聊天机器人将传入语音转换为文本并理解用户意图以生成智能响应，以便您能够集中精力为您的客户构建具有差异化增值的机器人，从而定义可通过对话接口生成的全新的产品类别。

- 无缝部署和扩展 — 借助 Amazon Lex，您可以直接从 Amazon Lex 控制台构建、测试和部署机器人。Amazon Lex 使您能够轻松发布要在移动设备、Web 应用程序和聊天服务（如 Facebook Messenger）上使用的语音或文本聊天机器人。Amazon Lex 将自动扩展，您无需担心预置硬件和管理基础设施来支持您的机器人体验。

- 与 AWS 平台的内置集成 — Amazon Lex 与其他 AWS 服务具有本机互操作性，例如 Amazon Cognito、、 AWS Lambda Amazon 和 CloudWatch。 AWS Mobile Hub您可借助 AWS 平台来实施安全性、监控、用户身份验证、业务逻辑、存储和移动应用程序开发。

- 经济高效 — 通过使用 Amazon Lex，无前期成本或最低费用。您只需为发出的文本或语音请求付费。定 pay-as-you-go价和每个请求的低成本使该服务成为构建对话界面的经济实惠的方式。通过 Amazon Lex 免费套餐，您可轻松试用 Amazon Lex，无需任何初期投资。

您是否是首次接触 Amazon Lex 的用户？

如果您是首次接触 Amazon Lex 的用户，我们建议您按顺序阅读以下内容：

1. Amazon Lex 入门 — 在本节中，您将设置账户并测试 Amazon Lex。
2. API 参考 — 本节提供了您可用于探索 Amazon Lex 的更多示例。

Amazon Lex：工作原理

借助 Amazon Lex，您能够使用由支持 Amazon Alexa 的相同技术提供支持的语音或文本界面构建应用程序。下面是您在使用 Amazon Lex 时执行的典型步骤：

1. 创建自动程序，并为其配置您想要支持的一个或多个目的。配置自动程序，使其了解用户的目标 (目的)，与用户进行对话以引出信息，并实现用户的目的。
2. 测试自动程序。您可以使用由 Amazon Lex 控制台提供的测试窗口客户端。
3. 发布版本和创建别名
4. 部署机器人。您可以在移动应用程序或消息收发等平台 (如 Facebook Messenger) 上部署自动程序。

开始之前，请熟悉以下 Amazon Lex 核心概念和术语：

- 机器人 — 机器人执行自动化任务，如订购披萨、预定酒店、订花等。Amazon Lex 机器人由自动语音识别 (ASR) 和自然语言理解 (NLU) 功能提供支持。在您的账户中，每个机器人都应具有唯一的名称。

  Amazon Lex 机器人可理解通过文本或语音提供的用户输入并支持自然语言交流。您可以创建 Lambda 函数，然后将其作为代码挂钩添加到您的意图配置中，从而执行用户数据验证以及完成任务。

- 意图 — 意图表示用户要执行的操作。您创建机器人以支持一个或多个相关意图。例如，您可以创建一个用于订购披萨和饮料的自动程序。对于每个目的，您需要提供以下必要信息：

  - 意图名称 — 意图的描述性名称。例如，**`OrderPizza`**。意图名称在您的账户内必须是唯一的。
  - 示例言语 — 用户表达意图的可能方式。例如，用户可能会说“我能订购披萨吗”和“我想订购披萨”。
  - 如何完成意图 — 在用户提供必要的信息后，您希望如何履行意图（例如，向当地披萨店下达订单）。我们建议您创建一个 Lambda 函数来履行意图。

您可以选择对意图进行配置，使 Amazon Lex 只将履行意图所需的必要信息返回客户端应用程序。

除了订购披萨等自定义意图之外，Amazon Lex 还提供内置意图来快速设置您的机器人。有关更多信息，请参阅 内置目的和槽类型。

- 插槽 — 一个意图可能需要零个或零个以上的插槽（参数）。您可以添加槽，作为意图配置的一部分。在运行时，Amazon Lex 提示用户提供特定的插槽值。用户必须为所有必需插槽提供值，然后 Amazon Lex 才能履行意图。

  例如，`OrderPizza` 目的需要诸如披萨尺寸、饼皮类型和披萨数量等槽。在目的配置中，您将添加这些槽。对于每个插槽，您需要提供插槽类型和提示，以便 Amazon Lex 发送到客户端来从用户那里引发数据。用户可以回复包含额外词语的插槽值，如“请来一张大号披萨”或“我还是吃小号的吧”。Amazon Lex 仍然可以理解预期的插槽值。

- 插槽类型 – 每个插槽都具有一种类型。您可创建自定义槽类型或使用内置槽类型。在您的账户中，每个插槽类型都应具有唯一的名称。例如，您可针对 `OrderPizza` 目的创建并使用以下槽类型：

  - 大小：使用枚举值 `Small`、`Medium` 和 `Large`。
  - 馅饼皮：使用枚举值 `Thick` 和 `Thin`。

  Amazon Lex 还提供了内置插槽类型。例如，`AMAZON.NUMBER` 是可用于订购披萨数量的内置槽类型。有关更多信息，请参阅 内置目的和槽类型。

有关提供了 Amazon Lex 的 AWS 区域的列表，请参阅《Amazon Web Services 一般参考》中的 AWS 区域和端点。

下列主题提供了其他信息。我们建议您按顺序检查它们，然后探索 Amazon Lex 入门 练习。

主题

Amazon Lex 支持的语言

Amazon Lex V1 支持多种语言和区域设置。下表列出了支持的语言和支持这些语言的功能。

Amazon Lex V2 支持额外的语言。请参阅 Amazon Lex V2 支持的语言

支持的语言和区域设置

Amazon Lex V1 支持以下语言和区域设置。

代码	语言和区域设置
de-DE	德语（德国）
en-AU	英语（澳大利亚）
en-GB	英语（英国）
en-IN	英语（印度）

代码	语言和区域设置
en-US	英语（美国）
es-419	西班牙语（拉丁美洲）
es-ES	西班牙语（西班牙）
es-US	西班牙语（美国）
fr-CA	法语（加拿大）
fr-FR	法语（法国）
it-IT	意大利语（意大利）
ja-JP	日语（日本）
ko-KR	韩语（韩国）

Amazon Lex 功能支持的语言和区域设置

除了下表中列出的语言和区域设置之外，所有 Amazon Lex 功能在所有语言和区域设置中均支持。

功能	支持的语言和区域设置
设置意图上下文	英语（美国）(en-US)

编程模型

机器人 是 Amazon Lex 中的主要资源类型。Amazon Lex 中的其他资源类型包括意图、插槽类型、别名 和机器人通道关联。

您可以使用 Amazon Lex 控制台或建模 API 创建机器人。该控制台提供了图形用户界面，您可以通过它为您的应用构建生产就绪自动程序。如果你愿意，你可以通过 AWS CLI 或你自己的自定义程序使用模型构建 API 来创建机器人。

创建自动程序后，您需要将其部署到一个支持的平台上或将其集成到您自己的应用程序中。当用户与该机器人交互时，客户端应用程序会使用 Amazon Lex 运行时 API 向机器人发送请求。例如，当用户

说：“我想订披萨”时，您的客户端会使用某个运行时 API 操作将此输入发送到 Amazon Lex。用户可以语音或文本形式提供输入。

您还可以创建 Lambda 函数并在意图中使用它们。使用这些 Lambda 函数代码挂钩可执行诸如初始化、验证用户输入和完成意图等运行时活动。以下各节提供了更多信息。

主题

建模 API 操作

要以编程方式创建自动程序、目的和槽类型，请使用建模 API 操作。您也可以使用建模 API 管理、更新和删除自动程序的资源。建模 API 操作包括：

- PutBot、PutBotAlias、PutIntent 和 PutSlotType，分别用于创建和更新自动程序、自动程序别名、目的和槽类型。
- CreateBotVersion、CreateIntentVersion 和 CreateSlotTypeVersion，分别用于创建和发布自动程序、目的和槽类型版本。
- GetBot 和 GetBots，分别用于获取您创建的特定自动程序或自动程序列表。
- GetIntent 和 GetIntents，分别用于获取您创建的特定目的或目的列表。
- GetSlotType 和 GetSlotTypes，分别用于获取您创建的特定槽类型或槽类型列表。
- GetBuiltinIntent、GetBuiltinIntents 和 GetBuiltinSlotTypes，分别用于获取您可以在机器人中使用的 Amazon Lex 内置意图、Amazon Lex 内置意图列表或内置插槽类型列表。
- GetBotChannelAssociation 和 GetBotChannelAssociations，分别用于获取您的自动程序与消息收发平台之间的关联，或您的自动程序与消息收发平台之间的关联列表。
- DeleteBot、DeleteBotAlias、DeleteBotChannelAssociation、DeleteIntent 和 DeleteSlotType，用于删除您账户中不需要的资源。

您可以使用建模 API 创建自定义工具来管理您的 Amazon Lex 资源。例如，自动程序、目的和槽类型各自都有 100 个版本的限制。您可以使用建模 API 构建一个工具，当自动程序接近版本数限制时自动删除旧版本。

为确保一次只有一个操作更新资源，Amazon Lex 使用了校验和。当您使用 Put API 操作 PutBot、PutBotAlias、PutIntent 或 PutSlotType 更新某个资源时，必须在请求中传递该资源的当前

校验和。如果两个工具同时尝试更新一个资源，它们会提供相同的当前校验和。第一个到达 Amazon Lex 的请求与该资源的当前校验和相符。当第二个请求达到时，校验和已经不同了。第二个工具会收到 `PreconditionFailedException` 异常，更新终止。

`Get` 操作 GetBot、GetIntent 和 GetSlotType 具有最终一致性。如果您在使用某个 `Put` 操作创建或修改资源后立即使用 `Get` 操作，可能不会返回更改。在 `Get` 操作返回最近的更新之后，它始终返回这一更新的资源，直到资源再次修改。您可以通过查看校验和确定是否返回了更新的资源。

运行时 API 操作

客户端应用程序使用以下运行时 API 操作与 Amazon Lex 通信：

- PostContent — 采用语音或文本输入并返回意图信息和要传达给用户的文本或语音消息。目前，Amazon Lex 支持以下音频格式：

  输入音频格式 – LPCM 和 Opus

  输出音频格式 – MPEG、OGG 和 PCM

  `PostContent` 操作支持 8kHz 和 16kHz 的音频输入。最终用户通过电话与 Amazon Lex 交流的应用程序（如自动呼叫中心）可以直接传递 8kHz 音频。

- PostText – 采用文本作为输入并返回目的信息和要传达给用户的文本消息。

您的客户端应用程序使用运行时 API 调用特定 Amazon Lex 机器人来处理言语（用户文本或语音输入）。例如，假设用户说“我想订披萨”。客户端会使用其中一个 Amazon Lex 运行时 API 操作将此用户输入发送给机器人。Amazon Lex 从用户输入识别出用户请求针对的是机器人中定义的 `OrderPizza` 意图。Amazon Lex 与用户进行对话以收集所需信息（即插槽数据），如披萨尺寸、配料和披萨数量等。在用户提供所有必要的插槽数据后，Amazon Lex 根据意图的配置方式调用 Lambda 函数代码挂钩来履行意图或将意图数据返回给客户端。

当您的自动程序使用语音输入时，请使用 PostContent 操作。例如，自动呼叫中心应用程序可以将语音发送给 Amazon Lex 机器人而非代理来解决客户查询。您可以使用 8kHz 音频格式将音频直接从电话发送给 Amazon Lex。

Amazon Lex 控制台中的测试窗口使用 PostContent API 向 Amazon Lex 发送文本和语音请求。您可以在 Amazon Lex 入门 练习中使用此测试窗口。

Lambda 函数作为代码挂钩

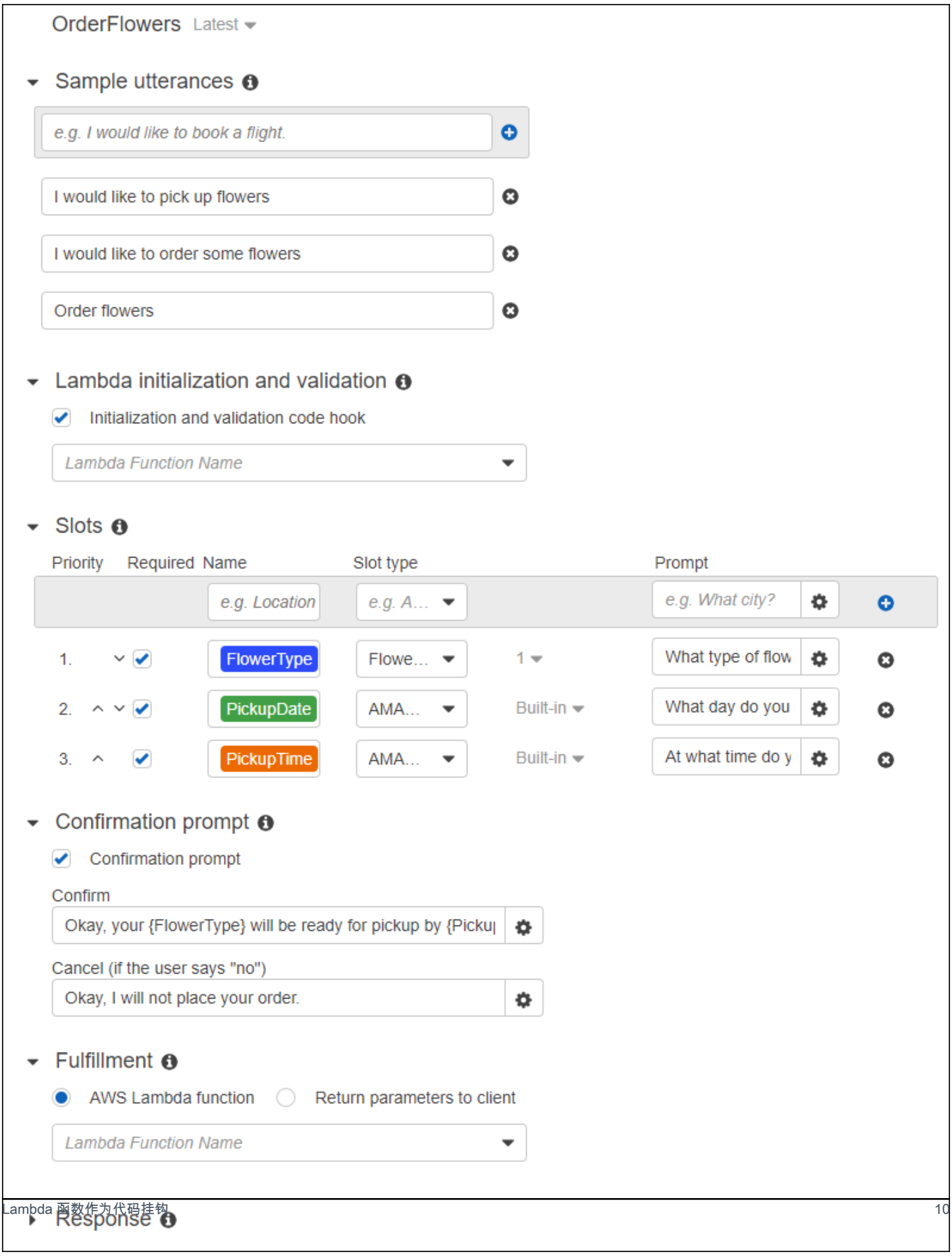
您可以将 Amazon Lex 机器人配置为调用 Lambda 函数作为代码挂钩。代码挂钩有多个用途：

- 自定义用户交互 — 例如，当 Joe 询问有什么披萨配料时，您可以使用之前对 Joe 的选择的了解来显示一组配料。
- 验证用户的输入 — 假设 Jen 想在几小时后取花。您可以验证 Jen 输入的时间，并发送适当的响应。
- 履行用户意图 — 在 Joe 提供所有披萨订购信息后，Amazon Lex 可以调用 Lambda 函数向附近披萨店下达订单。

在您配置意图时，您可以在以下位置指定 Lambda 函数作为代码挂钩：

- 用于初始化和验证的对话代码挂钩 — 对于每个用户输入，都将调用此 Lambda 函数（假设 Amazon Lex 理解用户意图）。
- 履行代码挂钩 — 在用户提供所有需要的插槽数据后将调用此 Lambda 函数来履行意图。

您可以在 Amazon Lex 控制台中选择意图并设置这些代码挂钩，如下面的屏幕截图所示：


OrderFlowers
Latest
Sample utterances
e.g. I would like to book a flight.
I would like to pick up flowers
I would like to order some flowers
Order flowers
Lambda initialization and validation
Initialization and validation code hook
Lambda Function Name
Slots
Priority
Required
Name
Slot type
Prompt
e.g. Location
e.g. A...
e.g. What city?
1.
FlowerType
Flowe...
1
What type of flow
2.
PickupDate
AMA...
Built-in
What day do you
3.
PickupTime
AMA...
Built-in
At what time do y
Confirmation prompt
Confirmation prompt
Confirm
Okay, your {FlowerType} will be ready for pickup by {Picku
Cancel (if the user says "no")
Okay, I will not place your order.
Fulfillment
AWS Lambda function
Return parameters to client
Lambda Function Name
Response

您也可以在 PutIntent 操作中使用 dialogCodeHook 和 fulfillmentActivity 字段设置代码挂钩。

一个 Lambda 函数就可以执行初始化、验证和履行操作。Lambda 函数收到的事件数据有一个标识调用方为对话或履行代码挂钩的字段。您可以使用此信息运行相应部分的代码。

您可以使用 Lambda 函数构建能够引领复杂对话的机器人。可以在 Lambda 函数响应中使用 dialogAction 字段指示 Amazon Lex 采取特定操作。例如，您可以使用 ElicitSlot 对话操作指示 Amazon Lex 向用户询问非必需的插槽值。如果您定义了澄清提示，则可以使用 ElicitIntent 对话操作在用户完成上一目的时引出一个新目的。

有关更多信息，请参阅 使用 Lambda 函数。

管理消息

主题

- 消息类型
- 配置消息的上下文
- 受支持的消息格式
- 消息组
- 响应卡

当您创建自动程序时，您可以配置您希望自动程序发送给客户端的说明性或信息性消息。考虑以下示例：

- 您可以为自动程序配置以下说明提示：

  ```
  I don't understand. What would you like to do?
  ```

  Amazon Lex 在不理解用户意图时会将此消息发送到客户端。

- 假设您要创建自动程序来支持名为 OrderPizza 的目的。对于披萨订单，您希望用户提供信息，如披萨尺寸、配料和馅饼皮类型。您可以配置以下提示：

  ```
  What size pizza do you want?
  ```

```
What toppings do you want?
Do you want thick or thin crust?
```

在 Amazon Lex 确定用户订购披萨的意图后，它会向客户端发送这些消息，从而从用户那里获取信息。

本节介绍如何在自动程序配置中设计用户交互。

消息类型

消息可以是提示或声明。

- 提示通常是一个问题，并且需要用户响应。
- 声明是信息性的。它不需要响应。

消息可以包括对槽、会话属性和请求属性的引用。在运行时，Amazon Lex 将这些引用替换为实际值。

要引用已设置的槽值，请使用以下语法：

```
{SlotName}
```

要引用会话属性，请使用以下语法：

```
[SessionAttributeName]
```

要引用请求属性，请使用以下语法：

```
((RequestAttributeName))
```

消息可以同时包括槽值、会话属性和请求属性。

例如，假设您根据机器人的 OrderPizza 意图配置了以下消息：

```
"Hey [FirstName], your {PizzaTopping} pizza will arrive in [DeliveryTime] minutes."
```

此消息同时引用槽 (PizzaTopping) 和会话属性 (FirstName 和 DeliveryTime)。在运行时，Amazon Lex 将这些占位符替换为值，并且向客户端返回以下消息：

```
"Hey John, your cheese pizza will arrive in 30 minutes."
```

要在消息中包括方括号 ([]) 或大括号 ({})，请使用反斜杠 (\) 转义字符。例如，以下消息包括大括号和方括号：

```
\{Text\} \[Text\]
```

返回到客户端应用程序的文本如下所示：

```
{Text} [Text]
```

有关会话属性的信息，请参阅运行时 API 操作 PostText 和 PostContent。有关示例，请参阅预订旅程。

Lambda 函数还可以生成消息并将它们返回到 Amazon Lex 以发送给用户。如果您在配置意图时添加 Lambda 函数，则可以动态创建消息。通过在配置机器人时提供消息，您无需在 Lambda 函数中构造提示。

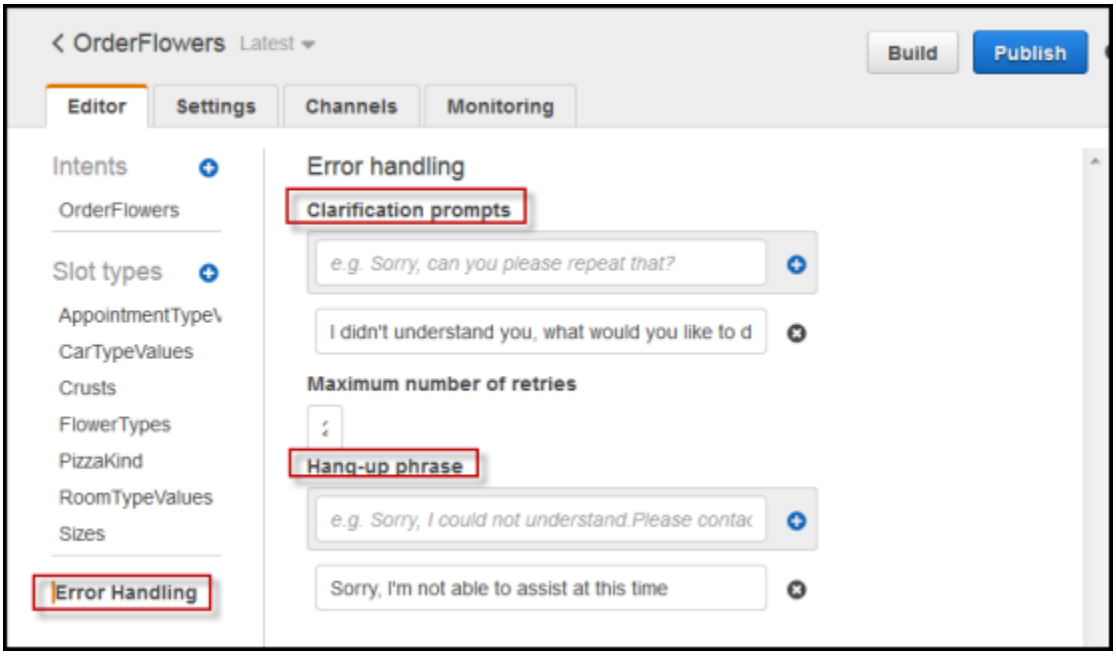
配置消息的上下文

在创建机器人时，您可以在不同的上下文中创建消息，如机器人中的说明提示、有关插槽值的提示以及来自意图的消息。Amazon Lex 在每个上下文中选择适当的消息来返回给您的用户。您可以为每个上下文提供一组消息。如果您这样做，Amazon Lex 会从组中随机选择一条消息。您也可以指定消息的格式或将消息组合在一起。有关更多信息，请参阅 受支持的消息格式。

如果您具有与意图关联的 Lambda 函数，则可以覆盖您在构建时配置的任何消息。但使用其中任何消息无需 Lambda 函数。

自动程序消息

您可以使用说明提示和会话结束消息配置您的机器人。在运行时，如果 Amazon Lex 不了解用户的意图，则会使用说明提示 。您可以配置 Amazon Lex 在发送会话结束消息之前请求澄清的次数。您可以在 Amazon Lex 控制台的错误处理部分配置机器人级别消息，如下图所示：



借助 API，您可以通过设置 PutBot 操作中的 `clarificationPrompt` 和 `abortStatement` 字段来配置消息。

如果您将 Lambda 函数与意图结合使用，则 Lambda 函数可能会返回指示 Amazon Lex 询问用户意图的响应。如果 Lambda 函数没有提供此类消息，则 Amazon Lex 会使用说明提示。

槽提示

您必须为目的中所需的每个槽指定至少一条提示消息。在运行时，Amazon Lex 将使用其中一条消息来提示用户为此插槽提供值。例如，对于 `cityName` 槽，下面是有效的提示：

```
Which city would you like to fly to?
```

您可以使用控制台为每个槽设置一个或多个提示。您还可以使用 PutIntent 操作创建多组提示。有关更多信息，请参阅 消息组。

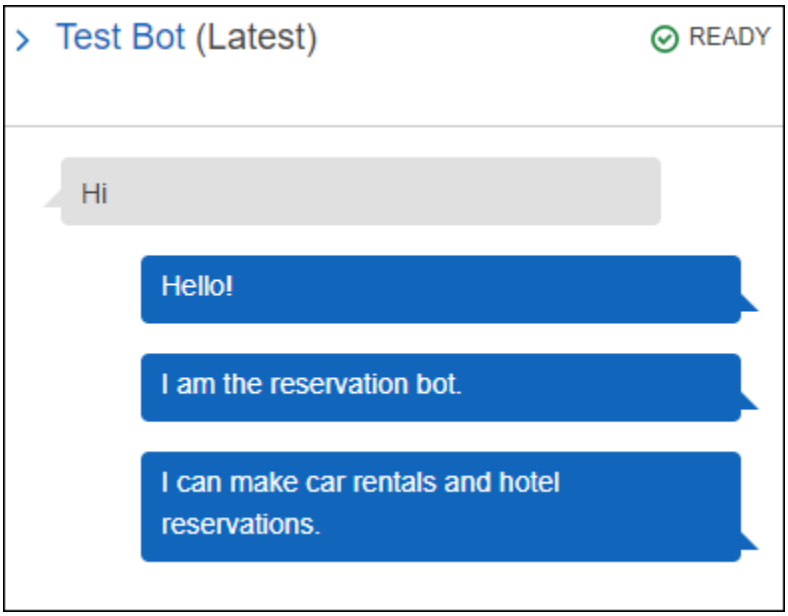
响应

在控制台中，使用 Responses (响应) 部分为您的自动程序构建令人愉快的动态对话。您可以为响应创建一个或多个消息组。在运行时，Amazon Lex 通过从每个消息组选择一条消息来构建响应。有关消息组的更多信息，请参阅消息组。

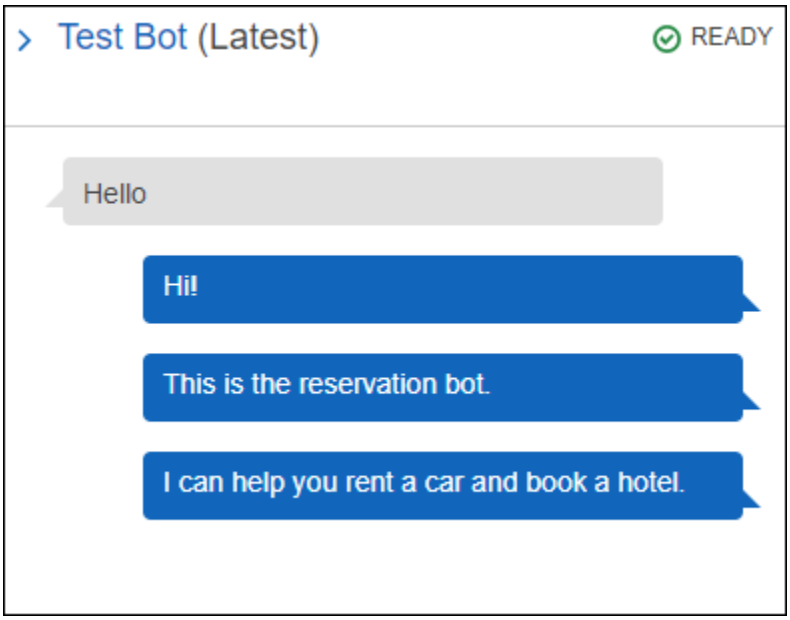
例如，您的第一个消息组可能包含不同的问候语：“Hello”(您好)、“Hi”(嗨) 和“Greetings”(问候)。第二个消息组可能包含不同形式的介绍：“I am the reservation bot”(我是预订自动程序) 和“This is the reservation bot”(这是预订自动程序)。第三个消息组可能传达自动程序的功能：“I can help with car

rentals and hotel reservations”(我可以帮忙租车和预订宾馆)、“You can make car rentals and hotel reservations”(您可以租车和预订宾馆) 和“I can help you rent a car and book a hotel”(我可以帮您租车和预订宾馆)。

Lex 使用每个消息组中的消息在对话中动态构建响应。例如，一种交互可能如下所示：



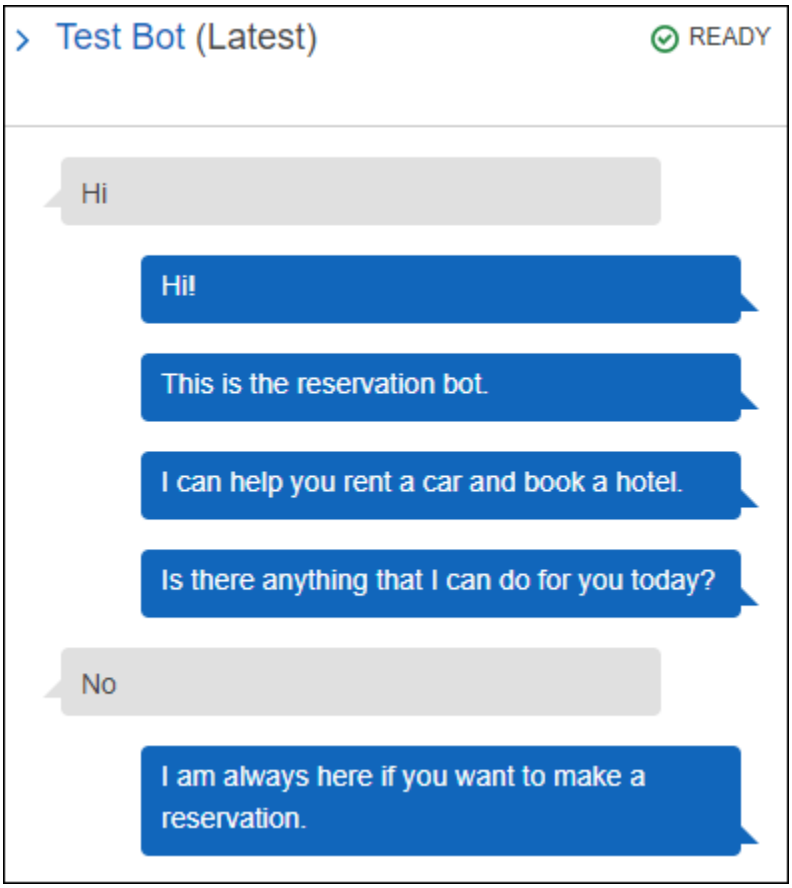
另一种交互可能如下所示：



在任一情况下，用户都可以使用新目的进行响应，如 `BookCar` 或 `BookHotel` 目的。

您可以将自动程序设置为在响应中询问后续问题。例如，对于前面的交互，您可以使用以下问题创建第四个消息组：“Can I help with a car or a hotel?”(我可以帮忙租车或预订宾馆吗？)、“Would you like to

make a reservation now?”(您是否要立即预订？) 和“Is there anything that I can do for you?”(我能为您做些什么吗？)。对于包括“No”(不) 作为响应的消息，您可以创建跟进提示。下图提供了一个示例。



要创建跟进提示，请选择 Wait for user reply (等待用户回复)。然后键入在用户说“No”(不) 时要发送的一条或多条消息。当您创建响应以用作跟进提示时，还必须指定对语句的回答是“No”(不) 时的适当语句。有关示例，请参阅下图：

要使用 API 将响应添加到目的，请使用 PutIntent 操作。要指定响应，请设置 PutIntent 请求中的 conclusionStatement 字段。要设置跟进提示，请设置 followUpPrompt 字段并包括要在用户说“No”(不) 时发送的语句。您不能对同一目的同时设置 conclusionStatement 字段和 followUpPrompt 字段。

受支持的消息格式

当您使用 PostText 操作时，或当您将 PostContent 操作与设置为 text/plain;charset=utf8 的 Accept 标头结合使用时，Amazon Lex 支持以下格式的消息：

- PlainText — 消息包含 UTF-8 纯文本。
- SSML — 消息包含为语音输出设置格式的文本。
- CustomPayload — 消息包含为您的客户端创建的自定义格式。您可以定义负载以满足应用程序的需要。
- Composite — 消息是来自每个消息组的消息的集合。有关消息组的更多信息，请参阅消息组。

默认情况下，Amazon Lex 返回为特定提示定义的任一个消息。例如，如果您定义五个消息来引发插槽值，则 Amazon Lex 会随机选择其中一个消息并将其返回给客户端。

如果您希望 Amazon Lex 在运行时请求中向客户端返回特定类型的消息，请设置 x-amzn-lex:accept-content-types 请求参数。响应局限于所请求的一种或多种类型。如果有多个指定类型的消息，则 Amazon Lex 会随机返回一个消息。有关 x-amz-lex:accept-content-types 标头的更多信息，请参阅设置响应类型。

消息组

消息组 是一组对特定提示的合适响应。当您希望自动程序在对话中动态构建响应时，请使用消息组。当 Amazon Lex 向客户端应用程序返回响应时，它会从每个组中随机选择一个消息。您最多可以为每个响应创建五个消息组。每个组最多可以包含五个消息。有关在控制台中创建消息组的示例，请参阅响应。

要创建消息组，您可以使用控制台，也可以使用 PutBot、PutIntent 或 PutSlotType 操作来将组号分配给消息。如果您未创建消息组，或者仅创建了一个消息组，则 Amazon Lex 在 Message 字段中发送单个消息。仅当您在控制台中创建了多个消息组时，或当您使用 PutIntent 操作创建或更新目的时创建了多个消息组时，客户端应用程序才会在响应中获取多个消息。

在 Amazon Lex 发送组中的消息时，响应的 Message 字段包含一个包含消息的转义的 JSON 对象。以下示例显示包含多个消息的 Message 字段的内容。

> **Note**
>
> 为方便阅读，为此示例设置了格式。响应不包含回车符 (CR)。

```
{\"messages\":[
   {\"type\":\"PlainText\",\"group\":0,\"value\":\"Plain text\"},
   {\"type\":\"SSML\",\"group\":1,\"value\":\"SSML text\"},
   {\"type\":\"CustomPayload\",\"group\":2,\"value\":\"Custom payload\"}
]}
```

您可以设置消息的格式。格式可以是以下格式之一：

- PlainText— 消息采用纯文本 UTF-8 文本。
- SSML – 消息是语音合成标记语言 (SSML)。
- CustomPayload— 消息采用您指定的自定义格式。

要控制 PostContent 和 PostText 操作在 Message 字段中返回的消息的格式，请设置 x-amz-lex:accept-content-types 请求属性。例如，如果您将标头设置为以下内容，您将在响应中仅收到纯文本和 SSML 消息：

```
x-amz-lex:accept-content-types: PlainText,SSML
```

如果您请求特定消息格式，但消息组没有包含该格式的消息，则会出现 NoUsableMessageException 异常。当您使用消息组按类型对消息进行分组时，请勿使用 x-amz-lex:accept-content-types 标头。

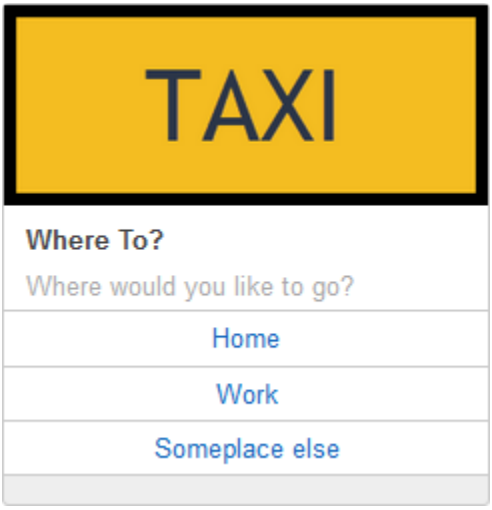
有关 x-amz-lex:accept-content-types 标头的更多信息，请参阅设置响应类型。

响应卡

> **Note**
>
> 响应卡不适用于 Amazon Connect 聊天。但是，有关类似功能，请参阅在聊天中添加交互式消息。

响应卡 包含一组对提示的适当响应。使用响应卡可为您的用户简化交互，并通过减少文本交互中的拼写错误来提高自动程序的准确度。您可以为 Amazon Lex 向客户端应用程序发送的每个提示发送一张响应卡。响应卡可以与 Facebook Messenger、Slack、Twilio 以及您自己的客户端应用程序结合使用。

例如，在出租车应用程序中，您可以在响应卡中配置一个“家”选项，并将值设置为用户的家庭地址。当用户选择此选项时，Amazon Lex 会收到输入文本形式的完整地址。参见下图：



您可以为以下提示定义响应卡：

- 结论语句
- 确认提示
- 跟进提示
- 拒绝语句
- 槽类型表达

您只能为每个提示定义一张响应卡。

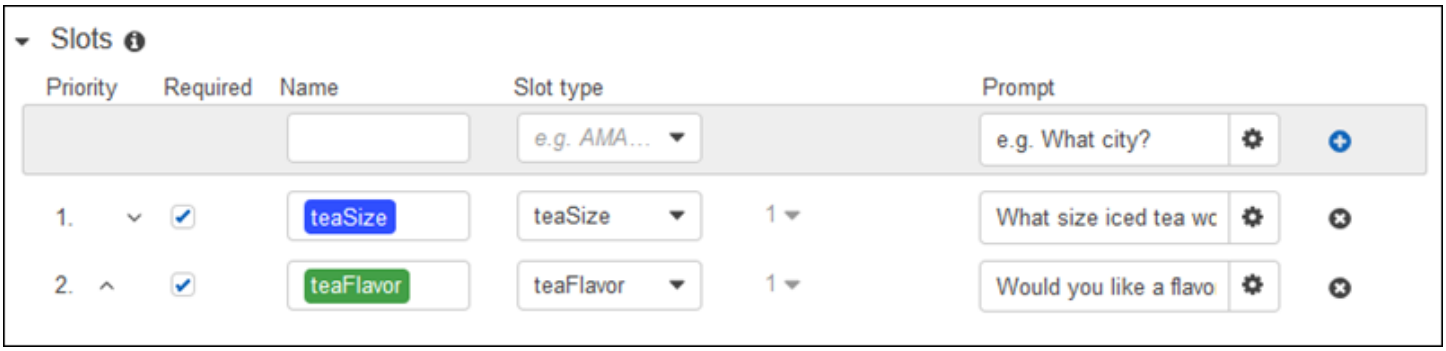
可以在创建目的时配置响应卡，可以在构建时使用控制台或 PutIntent 操作定义静态响应卡。或者，也可以在运行时在 Lambda 函数中定义动态响应卡。如果同时定义了静态和动态响应卡，动态响应卡优先。

Amazon Lex 以客户端可以理解的格式发送响应卡。它可以针对 Facebook Messenger、Slack 和 Twilio 转换响应卡。对于其他客户端，Amazon Lex 会在 PostText 响应中发送一个 JSON 结构。例如，如果客户端是 Facebook Messenger，Amazon Lex 会将响应卡转换为通用模板。有关 Facebook Messenger 通用模板的更多信息，请参阅 Facebook 网站上的 Generic Template。有关 JSON 结构的示例，请参阅动态生成响应卡。

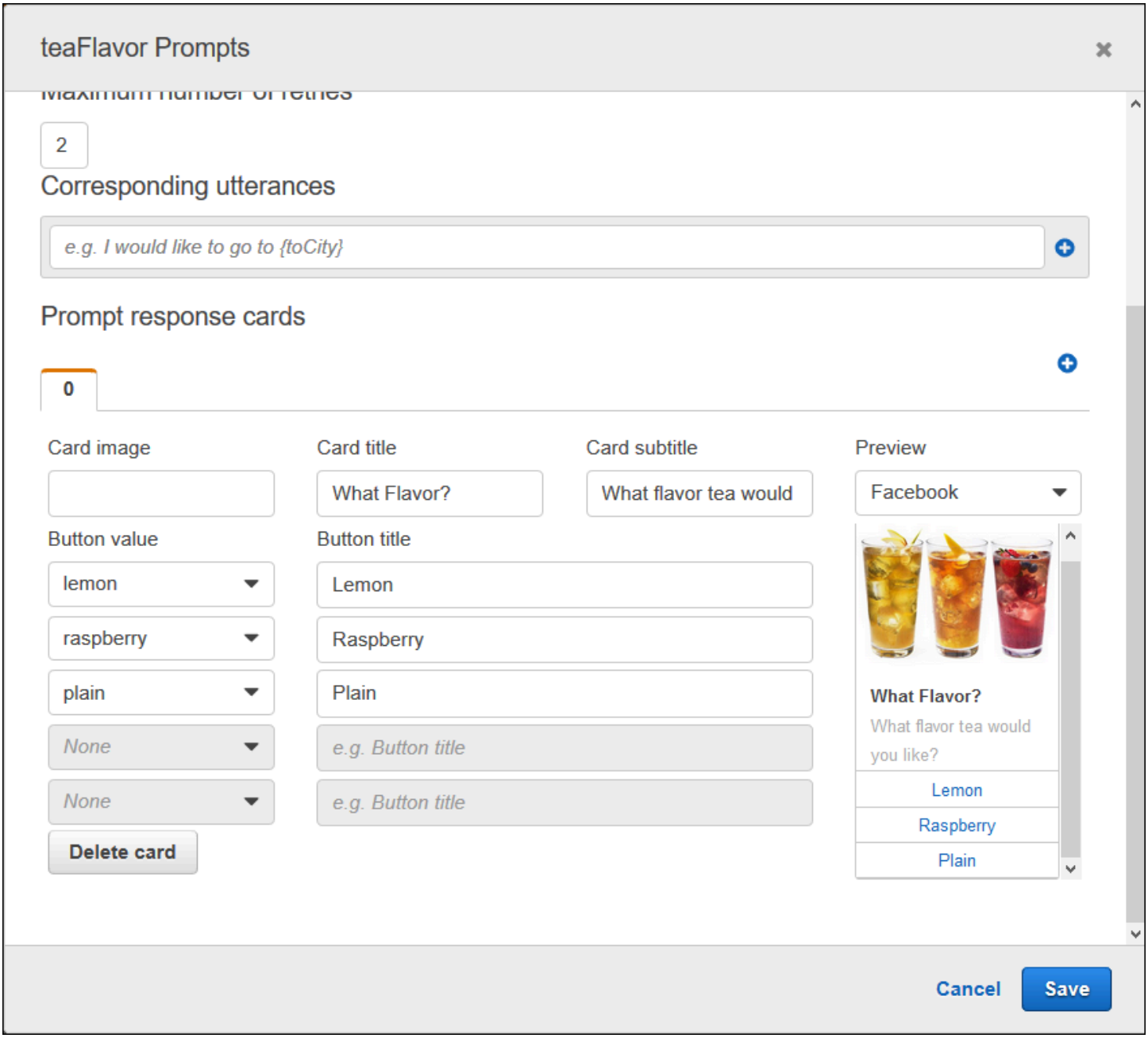
响应卡只能配合 PostText 操作使用，不适用于 PostContent 操作。

定义静态响应卡

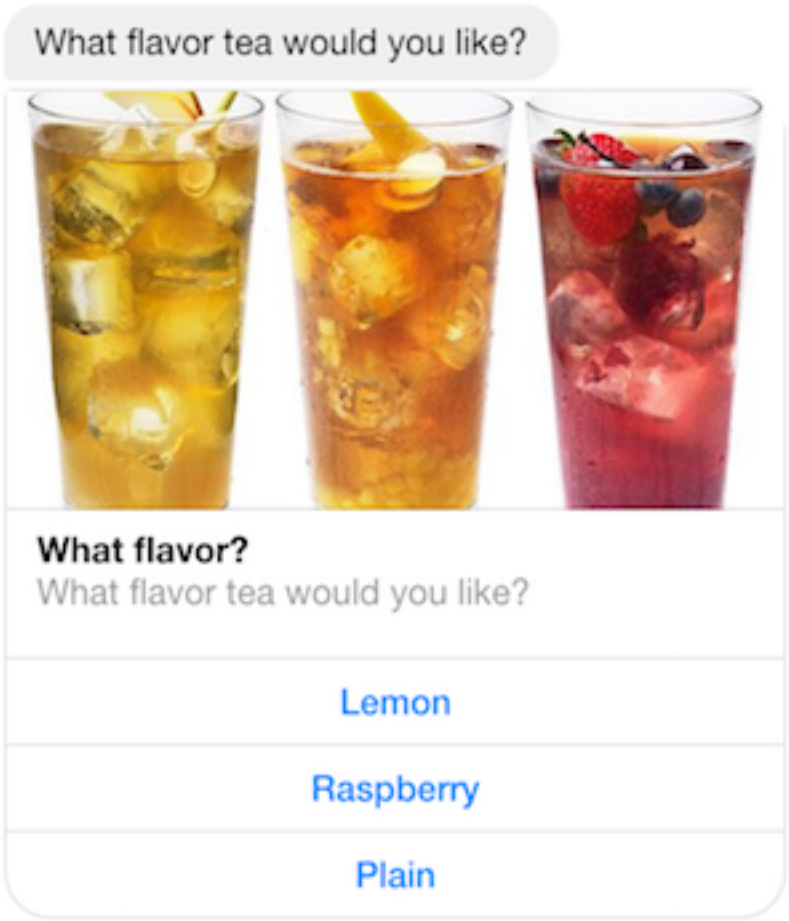
您可以在创建意图时使用 PutBot 操作或 Amazon Lex 控制台定义静态响应卡。静态响应卡与目的同时定义。当响应固定不变时，可以使用静态响应卡。假设您正在创建一个自动程序，它的一个目的包含与口味对应的槽。在定义这个口味槽时，您指定了提示，如下面的控制台屏幕截图所示：



在定义提示时，您可以选择关联一张响应卡并用 PutBot 操作或在 Amazon Lex 控制台中定义详细信息，如下例所示：



现在假设您已将自动程序与 Facebook Messenger 集成。用户可以单击按钮来选择口味，如下图所示：



要自定义响应卡的内容，您可以引用会话属性。在运行时，Amazon Lex 将这些引用替换为会话属性中合适的值。有关更多信息，请参阅 设置会话属性。有关示例，请参阅使用响应卡。

动态生成响应卡

要在运行时动态生成响应卡，请使用针对意图的初始化和验证 Lambda 函数。当响应是在运行时在 Lambda 函数中确定时，可以使用动态响应卡。响应用户输入时，Lambda 函数会生成一张响应卡并在响应的 `dialogAction` 部分返回它。有关更多信息，请参阅 响应格式。

下面是 Lambda 函数中显示 `responseCard` 元素的部分响应。它会生成类似上一部分示例所示的用户体验。

```
responseCard: {
  "version": 1,
  "contentType": "application/vnd.amazonaws.card.generic",
  "genericAttachments": [
    {
```

```
      "title": "What Flavor?",
      "subtitle": "What flavor do you want?",
      "imageUrl": "Link to image",
      "attachmentLinkUrl": "Link to attachment",
      "buttons": [
        {
          "text": "Lemon",
          "value": "lemon"
        },
        {
          "text": "Raspberry",
          "value": "raspberry"
        },
        {
          "text": "Plain",
          "value": "plain"
        }
      ]
    }
  ]
}
```

有关示例，请参阅安排预约。

管理对话上下文

对话上下文是用户、您的应用程序或 Lambda 函数向 Amazon Lex 机器人提供的用于履行意图的信息。对话上下文包括用户提供的插槽数据、客户端应用程序设置的请求属性以及客户端应用程序和 Lambda 函数创建的会话属性。

主题

- 设置意图上下文
- 使用默认插槽值
- 设置会话属性
- 设置请求属性
- 设置超时会话
- 在目的之间共享信息
- 设置复杂属性

设置意图上下文

您可以让 Amazon Lex 根据上下文触发意图。上下文是一个在定义机器人时与意图相关联的状态变量。

通过控制台或 PutIntent 操作创建意图时，您可以为该意图配置上下文。您只能在英语（美国）(en-US) 区域设置中使用上下文，并且前提是您在使用 PutBot 操作创建机器人时将 `enableModelImprovements` 参数设置为 `true`。

上下文有两种类型的关系，即输出上下文和输入上下文。当履行关联的意图时，输出上下文就会变为活动状态。在 PostText 或 PostContent 操作的响应中，会将输出上下文返回到您的应用程序，并为当前会话设置上下文。上下文被激活后，会保持活动状态，直到定义上下文时配置的回合数或时间限制。

输入上下文指定了可以识别意图的条件。只有当对话的所有输入上下文都处于活动状态时，才能识别出意图。没有输入上下文的意图始终可以被识别。

Amazon Lex 自动管理通过使用输出上下文履行意图而激活的上下文的生命周期。您还可以在调用 `PostContent` 或 `PostText` 操作时设置活动上下文。

您也可以通过 Lambda 函数为意图设置对话上下文。将 Amazon Lex 的输出上下文发送到 Lambda 函数输入事件。Lambda 函数可以在其响应中发送上下文。有关更多信息，请参阅 Lambda 函数输入事件和响应格式。

例如，假设您打算预订一辆配置返回名为“book_car_fulfilled”的输出上下文的租车。履行意图时，Amazon Lex 会将输出上下文变量设置为“book_car_fulfilled”。由于“book_car_fulfilled”上下文处于活动状态，因此，只要用户的言语被识别为试图引发该意图，就可以考虑将“book_car_fulfilled”上下文设置为输入上下文的意图进行识别。您可以将其用于只有在预订车辆后才有意义的意图，例如，通过电子邮件发送收据或修改预订。

输出上下文

履行意图时，Amazon Lex 会激活意图的输出上下文。您可以通过输出上下文来控制哪些意图符合跟随当前意图的条件。

每个上下文都有一个在会话中维护的参数列表。这些参数是已履行意图的插槽值。您可以使用这些参数为其他意图预填充插槽值。有关更多信息，请参阅使用默认插槽值。

当您通过控制台或 PutIntent 操作创建意图时，您可以配置输出上下文。您可以为一个意图配置多个输出上下文。履行意图时，所有输出上下文都将被激活，并在 PostText 或 PostContent 响应中返回。

以下是使用控制台为意图分配输出上下文的示例。

在定义输出上下文时，还要定义其生存时间以及该上下文包含在 Amazon Lex 的响应中的时长或回合数。一个回合是指从您的应用程序向 Amazon Lex 发出的一个请求。一旦回合数或时间到期，上下文将不再处于活动状态。

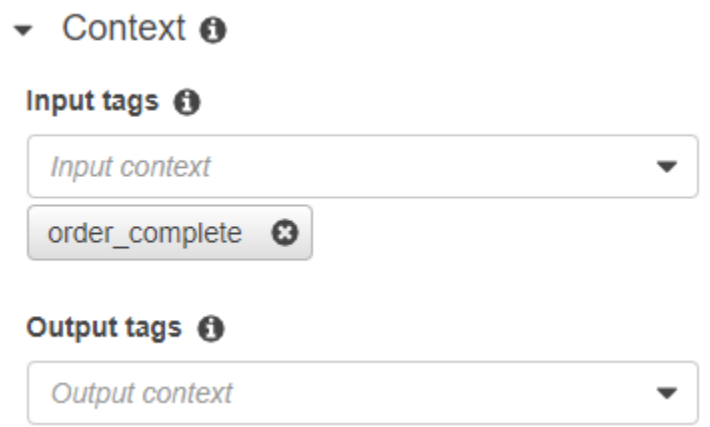
您的应用程序可以根据需要使用输出上下文。例如，您的应用程序可以通过输出上下文来：

- 根据上下文更改应用程序的行为。例如，旅行应用程序对上下文“book_car_fulfilled”的操作可能与对“rental_hotel_fulfilled”的操作不同。
- 将输出上下文作为下一句话的输入上下文返回给 Amazon Lex。如果 Amazon Lex 将言语识别为尝试引发意图，则会通过上下文将可以返回的意图限制为具有指定上下文的意图。

输入上下文

您可以设置输入上下文来限制对话中识别意图的点。没有输入上下文的意图始终可以被识别。

您可以通过控制台或 `PutIntent` 操作设置意图响应的输入上下文。一个意图可以包含多个输入上下文。以下是使用控制台为意图分配输入上下文的示例。



对于具有多个输入上下文的意图，所有上下文都必须处于活动状态才能触发该意图。您可以在调用 PostText、PostContent 或 PutSession 操作时设置输入上下文。

您可以将意图中的插槽配置为采用当前活动上下文中的默认值。当 Amazon Lex 识别出新的意图但未收到插槽值时，使用默认值。定义插槽时，可以在表单 `#context-name.parameter-name` 中指定上下文名称和插槽名称。有关更多信息，请参阅 使用默认插槽值。

使用默认插槽值

使用默认值时，如果用户输入中未提供插槽，您可以指定一个源用于为新意图填充的插槽值。此来源可以是之前的对话、请求或会话属性，也可以是您在构建时设置的固定值。

您可以将以下内容作为默认值来源。

- 之前的对话（上下文）— #context-name.parameter-name
- 会话属性 — [attribute-name]
- 请求属性 — <attribute-name>
- 固定值 — 任何与先前值不匹配的值

通过 PutIntent 操作向意图添加插槽时，可以添加默认值列表。将按这些默认值列出的顺序对其进行排列。例如，假设您有一个带插槽的意图，其插槽定义如下：

```
"slots": [
    {
        "name": "reservation-start-date",
        "defaultValueSpec": {
            "defaultValueList": [
                {
                    "defaultValue": "#book-car-fulfilled.startDate"
                },
                {
                    "defaultValue": "[reservationStartDate]"
                }
            ]
        },
        Other slot configuration settings
    }
]
```

识别出意图后，名为 “reservation-start-date” 的插槽的值将设置为以下值之一。

1. 如果 “book-car-fulfilled” 上下文处于活动状态，则使用 “startDate” 参数的值作为默认值。
2. 如果 “book-car-fulfilled” 上下文未处于活动状态，或者未设置 “startDate” 参数，则使用 “reservationStartDate” 会话属性的值作为默认值。
3. 如果前两个默认值均未使用，则该插槽没有默认值，Amazon Lex 将照常引发一个值。

如果使用插槽的默认值，则即使需要该插槽，也不会引发该插槽。

设置会话属性

会话属性包含特定于应用程序的信息，于会话期间在机器人与客户端应用程序之间传递。Amazon Lex 向为机器人配置的所有 Lambda 函数传递会话属性。如果 Lambda 函数添加或更新会话属性，Amazon Lex 会将新信息返回给客户端应用程序。例如：

- 在练习 1：使用蓝图创建 Amazon Lex 机器人（控制台）中，示例自动程序使用 `price` 会话属性维护鲜花价格。Lambda 函数根据所订购鲜花的类型设置此属性。有关更多信息，请参阅 步骤 5 (可选)：查看信息流的详细信息 (控制台)。
- 在预订旅程中，示例自动程序使用 `currentReservation` 会话属性，在预订酒店或预订租车的对话中维护槽类型数据的副本。有关更多信息，请参阅 信息流的详细信息。

在 Lambda 函数中使用会话属性来初始化机器人并自定义提示和响应卡。例如：

- 初始化 — 在订购披萨机器人中，客户端应用程序在第一次调用 PostContent 或 PostText 操作时传递用户位置作为会话属性。例如，`"Location": "111 Maple Street"`。Lambda 函数根据该信息查找最近的披萨店下订单。
- 个性化提示 — 配置提示和响应卡，以引用会话属性。例如，“嘿 [FirstName]，你想要什么浇头？” 如果您传递该用户的名字作为会话属性 (`{"FirstName": "Jo"}`)，Amazon Lex 将用该名字替换占位符。然后，它会向该用户发送个性化提示，“Jo 您好，您想要什么配料？”

会话属性在会话持续时间内一直存在。Amazon Lex 将其存储在加密数据存储中，直到会话结束。客户端可以通过调用 PostContent 或 PostText 操作同时为 `sessionAttributes` 字段设置值来创建会话属性。Lambda 函数可以在响应中创建会话属性。在客户端或 Lambda 函数创建会话属性后，每当客户端应用程序在发给 Amazon Lex 的请求中不包括 `sessionAttribute` 字段时，都会使用存储的属性值。

例如，假设您有两个会话属性 `{"x": "1", "y": "2"}`。如果客户端调用 `PostContent` 或 `PostText` 操作而未指定 `sessionAttributes` 字段，Amazon Lex 通过存储的会话属性 (`{"x":`

1, "y": 2}) 调用 Lambda 函数。如果 Lambda 函数未返回会话属性，Amazon Lex 会将存储的会话属性返回给客户端应用程序。

如果客户端应用程序或 Lambda 函数传递会话属性，Amazon Lex 将更新存储的会话属性。传递现有值 (例如 {"x": 2}) 将更新存储的值。如果您传递一组新的会话属性（如 {"z": 3}），将删除现有值而只保留新值。当传递空映射 {} 时，将擦除存储的值。

要向 Amazon Lex 发送会话属性，您需要创建属性 string-to-string映射。下面显示了如何映射会话属性：

```
{
   "attributeName": "attributeValue",
   "attributeName": "attributeValue"
}
```

对于 PostText 操作，您可以使用 sessionAttributes 字段将映射插入请求正文中，如下所示：

```
"sessionAttributes": {
   "attributeName": "attributeValue",
   "attributeName": "attributeValue"
}
```

对于 PostContent 操作，您对映射进行 base64 编码，然后将其作为 x-amz-lex-session-attributes 标头发送。

如果要在会话属性中发送二进制或结构化数据，则必须先将该数据转换为简单字符串。有关更多信息，请参阅 设置复杂属性。

设置请求属性

请求属性包含请求特定的信息，并仅应用于当前请求。客户端应用程序会将此信息发送给 Amazon Lex。可以使用请求属性传递不需要在整个会话中保留的信息。您可以创建自己的请求属性，也可以使用预定义属性。要发送请求属性，请在 the section called “PostText” 请求的 the section called “PostContent” 或 requestAttributes 字段中使用 x-amz-lex-request-attributes 标头。由于请求属性不像会话属性那样在不同请求间保留，因此不会在 PostContent 或 PostText 响应中返回。

> **Note**
>
> 要发送在请求间保留的信息，请使用会话属性。

命名空间 x-amz-lex: 是为预定义请求属性预留的。请勿创建带有 x-amz-lex: 前缀的请求属性。

设置预定义请求属性

Amazon Lex 提供预定义请求属性，用于管理它对发送至机器人的信息的处理方式。这些属性不会在整个会话中保留，因此必须在每个请求中发送预定义属性。所有预定义属性都在 x-amz-lex: 命名空间中。

除了以下预定义属性之外，Amazon Lex 还提供消息收发平台的预定义属性。有关这些属性的列表，请参阅在消息收发平台上部署 Amazon Lex 机器人。

设置响应类型

如果您有两个具有不同功能的客户端应用程序，则可能需要限制响应中的消息的格式。例如，您可能希望将发送到 Web 客户端的消息限制为纯文本，但使移动客户端能够同时使用纯文本和语音合成标记语言 (SSML)。要设置 PostContent 和 PostText 操作返回的消息的格式，请使用 x-amz-lex:accept-content-types" 请求属性。

您可以将此属性设置为以下消息类型的任意组合：

- PlainText — 消息包含 UTF-8 纯文本。
- SSML — 消息包含为语音输出设置格式的文本。
- CustomPayload — 消息包含为您的客户端创建的自定义格式。您可以定义负载以满足应用程序的需要。

Amazon Lex 仅返回具有响应的 Message 字段中指定的类型的消息。您可以设置多个值，以逗号分隔这些值。如果您使用消息组，则每个消息组必须至少包含一个指定类型的消息。否则，您将收到 NoUsableMessageException 错误。有关更多信息，请参阅 消息组。

> **Note**
>
> x-amz-lex:accept-content-types 请求属性不会影响 HTML 正文的内容。PostText 操作响应的内容始终为纯 UTF-8 文本。PostContent 操作响应的正文包含请求中 Accept 标头中设置的格式的数据。

设置首选时区

要设置用于解析日期的时区，以使其与用户的时区相关，请使用 x-amz-lex:time-zone 请求属性。如果您未在 x-amz-lex:time-zone 属性中指定时区，则默认值取决于您用于自动程序的区域。

区域	默认时区
美国东部（弗吉尼亚州北部）	America/New_York
美国西部（俄勒冈州）	America/Los_Angeles
亚太地区（新加坡）	Asia/Singapore
亚太地区（悉尼）	Australia/Sydney
亚太地区（东京）	Asia/Tokyo
欧洲地区（法兰克福）	Europe/Berlin
欧洲地区（爱尔兰）	Europe/Dublin
欧洲地区（伦敦）	Europe/London

例如，如果用户在对“您希望您的包裹哪天送达？”提示的响应中回复 tomorrow，则包裹送达的实际日期依据用户的时区确定。例如，纽约时间 9 月 16 日 01:00 时相当于洛杉矶时间 9 月 15 日 22:00。如果您的服务在美国东部（弗吉尼亚州北部）区域中运行，洛杉矶的一位用户使用默认时区要求包裹于“明天”送达，那么该包裹将于 17 日送达，而不是 16 日。但是，如果您将 x-amz-lex:time-zone 请求属性设置为 America/Los_Angeles，该包裹将于 16 日送达。

您可以将该属性设置为任何互联网编号分配机构 (IANA) 时区名称。有关时区名称的列表，请参阅维基百科上的 tz 数据库时区的列表。

设置用户定义的请求属性

用户定义的请求属性是您在每个请求中发送给自动程序的数据。可在 PostContent 请求的 amz-lex-request-attributes 标头或 PostText 请求的 requestAttributes 字段中发送信息。

要向 Amazon Lex 发送请求属性，您需要创建属性 string-to-string映射。下面显示了如何映射请求属性：

```
{
   "attributeName": "attributeValue",
   "attributeName": "attributeValue"
}
```

对于 PostText 操作，您可以使用 requestAttributes 字段将映射插入请求正文中，如下所示：

```
"requestAttributes": {
   "attributeName": "attributeValue",
   "attributeName": "attributeValue"
}
```

对于 PostContent 操作，您对映射进行 base64 编码，然后将其作为 x-amz-lex-request-attributes 标头发送。

如果您要在请求属性中发送二进制或结构化数据，必须先将该数据转换为简单字符串。有关更多信息，请参阅 设置复杂属性。

设置超时会话

Amazon Lex 保留上下文信息（插槽数据和会话属性），直到对话会话结束。要控制会话在自动程序中的持续时间，请设置会话超时。默认情况下，会话持续时间为 5 分钟，但您可以指定介于 0 到 1440 分钟 (24 小时) 之间的任何持续时间。

例如，假设您创建一个支持 OrderShoes 和 GetOrderStatus 等意图的 ShoeOrdering 机器人。当 Amazon Lex 检测到用户的意图是订购鞋子时，它会要求提供插槽数据。例如，要求提供鞋子尺码、颜色、品牌等。如果用户提供了一些插槽数据，但未完成鞋子的购买，Amazon Lex 将在整个会话中记住所有插槽数据和会话属性。如果用户在会话到期之前返回到会话，该用户可以提供其余槽数据并完成购买。

在 Amazon Lex 控制台中，您应在创建机器人时设置会话超时。使用 AWS 命令行界面 (AWS CLI) 或 API，您可以通过设置 IdleS ession Seconds 字段来设置使用该PutBot操作创建机器人或更新机器人时的超时TTLIn时间。

在目的之间共享信息

Amazon Lex 支持在意图之间共享信息。要在目的之间共享信息，请使用会话属性。

例如，ShoeOrdering 自动程序的用户从订购鞋子开始。该自动程序将与用户进行对话，收集槽数据，如鞋子尺寸、颜色和品牌。当用户下单时，履行订单的 Lambda 函数将设置 orderNumber 会话属性，其中包含订单号。要获取订单状态，用户可使用 GetOrderStatus 目的。自动程序可向用户询问槽数据，如订单号和订购日期。自动程序在获得所需的信息以后将返回订单状态。

如果您认为您的用户可能会在同一会话期间改换目的，则可将自动程序设计为返回最新订单的状态。您不必再向用户询问订单信息，而是使用 orderNumber 会话属性在不同目的间共享信息并实现 GetOrderStatus 目的。自动程序通过返回用户最后所下订单的状态完成此操作。

有关跨目的信息共享的示例，请参阅预订旅程。

设置复杂属性

会话和请求属性是属性和值的 string-to-string映射。在许多情况下，您可以使用字符串映射在客户端应用程序与自动程序之间传输属性值。但在某些情况下，您可能需要传输无法轻易转换为字符串映射的二进制数据或复杂结构。例如，以下 JSON 对象表示由美国人口最多的三个城市组成的数组：

```
{
   "cities": [
      {
         "city": {
            "name": "New York",
            "state": "New York",
            "pop": "8537673"
         }
      },
      {
         "city": {
            "name": "Los Angeles",
            "state": "California",
            "pop": "3976322"
         }
      },
      {
         "city": {
            "name": "Chicago",
            "state": "Illinois",
            "pop": "2704958"
         }
      }
   ]
}
```

这个数据数组不能很好地转换为 string-to-string地图。在这种情况下，您可以将对象转换为一个简单字符串，以便通过 PostContent 和 PostText 操作将其发送给自动程序。

例如，如果您正在使用 JavaScript，则可以使用`JSON.stringify`操作将对象转换为 JSON，使用`JSON.parse`操作将 JSON 文本转换为 JavaScript 对象：

```
// To convert an object to a string.
```

```
var jsonString = JSON.stringify(object, null, 2);
// To convert a string to an object.
var obj = JSON.parse(JSON string);
```

要通过PostContent操作发送会话属性，在将属性添加到请求标头之前，必须对这些属性进行 base64 编码，如以下 JavaScript代码所示：

```
var encodedAttributes = new Buffer(attributeString).toString("base64");
```

您可以通过先将二进制数据转换为 base64 编码字符串、然后将该字符串作为会话属性中的值发送，来向 PostContent 和 PostText 操作发送二进制数据：

```
"sessionAttributes" : {
   "binaryData": "base64 encoded data"
}
```

使用置信度分数

当用户说话时，Amazon Lex 会使用自然语言理解 (NLU) 来理解用户的请求并返回正确的意图。默认情况下，Amazon Lex 会返回您的机器人定义的最可能的意图。

在某些情况下，Amazon Lex 可能很难确定最可能的意图。例如，用户可能会说出模棱两可的言语，或者可能有两个相似的意图。为了帮助确定正确的意图，您可以将自己的领域知识与替代意图列表的置信度分数相结合。置信度分数是 Amazon Lex 提供的评级，它表明其对意图是正确意图的信心程度。

要确定两个替代意图之间的差异，您可以比较它们的置信度分数。例如，如果一个意图的置信度分数为 0.95，而另一个意图的置信度分数为 0.65，则第一个意图可能是正确的。但是，如果一个意图的分数为 0.75，而另一个意图的分数为 0.72，则这两个意图之间存在歧义，您可以在应用程序中使用领域知识来区分。

您还可以使用置信度分数来创建测试应用程序，以确定对意图言语的更改是否会影响机器人的行为。例如，您可以使用一组言语来获取机器人意图的置信度分数，然后用新的言语更新意图。然后，您可以查看置信度分数，以确定是否有所改善。

Amazon Lex 返回的置信度分数是比较值。请避免将其视作绝对分数进行信任。根据对 Amazon Lex 的改进，这些值可能会发生变化。

当您使用置信度分数时，Amazon Lex 会在每个响应中返回最有可能的意图和最多 4 个替代意图及其相关分数。如果所有置信度分数都低于阈值，Amazon Lex 将包括 AMAZON.FallbackIntent 和/或

AMAZON.KendraSearchIntent（如果您已配置它们）。您可以使用默认阈值，也可以设置您自己的阈值。

以下 JSON 代码显示了 PostText 操作的响应中的 alternativeIntents 字段。

```
"alternativeIntents": [
   {
      "intentName": "string",
      "nluIntentConfidence": {
         "score": number
      },
      "slots": {
         "string" : "string"
      }
   }
],
```

设置您创建或更新机器人时的阈值。您可以使用 API 或 Amazon Lex 控制台。对于下面列出的区域，您需要选择加入才能提高准确性和置信度分数。在控制台中，在高级选项部分选择置信度分数。使用 API 时，在调用 PutBot 操作时设置 enableModelImprovements 参数。

- 美国东部（弗吉尼亚州北部）(us-east-1)
- 美国西部（俄勒冈州）(us-west-2)
- 亚太地区（悉尼）(ap-southeast-2)
- 欧洲地区（爱尔兰）(eu-west-1)

在所有其他区域中，默认情况下都提供精度改进和置信度分数支持。

要更改置信度阈值，请在控制台中或使用 PutBot 操作进行设置。阈值必须是介于 1.00 和 0.00 之间的数字。

要使用控制台，请在创建或更新机器人时设置置信度阈值。

在创建机器人时设置置信度阈值（控制台）

- 在创建您的机器人上，在置信度分数阈值字段中输入一个值。

更新置信度阈值（控制台）

1. 从机器人列表中，选择要更新的机器人。

2. 选择设置选项卡。
3. 在左侧导航窗格中，选择常规。
4. 更新置信度分数阈值字段中的值。

设置或更新置信度阈值 (SDK)

- 设置 PutBot 操作的 `nluIntentConfidenceThreshold` 参数。以下 JSON 代码显示了正在设置的参数。

```
"nluIntentConfidenceThreshold": 0.75,
```

会话管理

要更改 Amazon Lex 在与用户对话中使用的意图，您可以使用对话代码挂钩 Lambda 函数中的响应，也可以在自定义应用程序 APIs中使用会话管理。

使用 Lambda 函数 URL

当您使用 Lambda 函数时，Amazon Lex 会使用包含该函数输入的 JSON 结构对其进行调用。JSON 结构包含一个名为 `currentIntent` 的字段，该字段包含 Amazon Lex 已确定为最有可能的用户言语意图的意图。JSON 结构还包括一个 `alternativeIntents` 字段，该字段包含最多四个可能满足用户意图的额外意图。每个意图都包含一个名为 `nluIntentConfidenceScore` 的字段，其中包含 Amazon Lex 为该意图分配的置信度分数。

要使用替代意图，您可以在 Lambda 函数的 `ConfirmIntent` 或 `ElicitSlot` 对话操作中指定该意图。

有关更多信息，请参阅 使用 Lambda 函数。

使用会话管理 API

要使用与当前意图不同的意图，请使用 PutSession 操作。例如，如果您认为第一个替代意图比 Amazon Lex 选择的意图更可取，则可以使用 `PutSession` 操作来更改意图，以便用户与之交互的下一个意图就是您选择的意图。

有关更多信息，请参阅 使用 Amazon Lex API 管理会话。

对话日志

您可以启用对话日志 来存储自动程序的交互。您可以使用这些日志查看自动程序的性能，并解决与对话相关的问题。您可以记录 PostText 操作的文本。您可以记录 PostContent 操作的文本和音频。通过启用对话日志，您可以获得用户与自动程序所进行对话的详细视图。

例如，与您的自动程序开展的会话具有会话 ID。您可以使用此 ID 获取对话的记录，包括用户话语和相应的自动程序响应。您还可以获取元数据，例如话语的意图名称和槽位值。

> **Note**
>
> 您不能将对话日志用于受儿童在线隐私保护法案 (COPPA) 约束的自动程序。

对话日志是为别名配置的。每个别名的文本日志和音频日志都可以有不同的设置。您可以为每个别名启用文本日志和/或音频日志。文本日志在 CloudWatch 日志中存储文本输入、音频输入脚本和相关元数据。音频日志将音频输入存储在 Amazon S3 中。您可以使用 AWS KMS 客户管理功能启用文本和音频日志的加密 CMKs。

要配置日志记录，请使用控制台或 PutBotAlias 操作。对于您的机器人或在 Amazon Lex 控制台中提供的测试机器人，您无法记录 $LATEST 别名的对话。为别名启用对话日志PostContent或对该别名的PostText操作后，将在配置的日志 CloudWatch 日志组或 S3 存储桶中记录文本或音频语句。

主题

- 用于对话日志的 IAM 策略
- 配置对话日志
- 加密对话日志
- 在 Amazon 日志中查看文本 CloudWatch 日志
- 在 Amazon S3 中访问音频日志
- 使用 CloudWatch 指标监控对话日志状态

用于对话日志的 IAM 策略

根据您选择的日志类型，Amazon Lex 需要权限才能使用亚马逊 CloudWatch 日志和亚马逊简单存储服务 (S3) 存储桶来存储您的日志。您必须创建 AWS Identity and Access Management 角色和权限才能让 Amazon Lex 访问这些资源。

为对话日志创建 IAM 角色和策略

要启用对话日志，您必须授予 CloudWatch 日志和 Amazon S3 的写入权限。如果您为 S3 对象启用对象加密，则需要向用于加密对象的 AWS KMS 密钥授予访问权限。

您可以使用 IAM AWS Management Console、IAM API 或 AWS Command Line Interface 来创建角色和策略。这些说明使用 AWS CLI 来创建角色和策略。有关使用控制台创建策略的更多信息，请参阅《AWS Identity and Access Management 用户指南》中的在 JSON 选项卡上创建策略。

> **Note**
>
> 以下代码针对 Linux 和 macOS 编排了格式。对于 Windows，将 Linux 行继续符 (\) 替换为脱字号 (^)。

为对话日志创建 IAM 角色

1. 在名为 **LexConversationLogsAssumeRolePolicyDocument.json** 的当前目录中创建一个文档，向其中添加以下代码并保存。此策略文档将 Amazon Lex 作为受信任实体添加到角色中。这允许 Lex 代入将日志传送到为对话日志配置的资源的角色。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lex.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. 在中 AWS CLI，运行以下命令为对话日志创建 IAM 角色。

```
aws iam create-role \
    --role-name role-name \
    --assume-role-policy-document file://
LexConversationLogsAssumeRolePolicyDocument.json
```

接下来，创建策略并将其附加到该角色以允许 Amazon Lex 写入 CloudWatch 日志。

创建用于将对话文本记录到 Log CloudWatch s 的 IAM 策略

1. 在名为 **LexConversationLogsCloudWatchLogsPolicy.json** 的当前目录中创建一个文档，向其中添加 IAM 策略并保存。

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
              "logs:CreateLogStream",
              "logs:PutLogEvents"
          ],
          "Resource": "arn:aws:logs:region:account-id:log-group:log-group-name:*"
      }
  ]
}
```

2. 在中 AWS CLI，创建向 CloudWatch 日志组授予写入权限的 IAM 策略。

```
aws iam create-policy \
    --policy-name cloudwatch-policy-name \
    --policy-document file://LexConversationLogsCloudWatchLogsPolicy.json
```

3. 将该策略附加到您为对话日志创建的 IAM 角色中。

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::account-id:policy/cloudwatch-policy-name \
    --role-name role-name
```

如果要将音频日志记录到 S3 存储桶，请创建允许 Amazon Lex 写入存储桶的策略。

创建用于将音频日志记录到 S3 存储桶中的 IAM 策略

1. 在名为 **LexConversationLogsS3Policy.json** 的当前目录中创建一个文档，向其中添加以下策略并保存。

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
              "s3:PutObject"
          ],
          "Resource": "arn:aws:s3:::bucket-name/*"
      }
  ]
}
```

2. 在中 AWS CLI，创建授予您的 S3 存储桶写入权限的 IAM 策略。

```
aws iam create-policy \
    --policy-name s3-policy-name \
    --policy-document file://LexConversationLogsS3Policy.json
```

3. 将该策略附加到您为对话日志创建的角色。

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::account-id:policy/s3-policy-name \
    --role-name role-name
```

授予传递 IAM 角色的权限

当您使用控制台 AWS Command Line Interface、或 AWS 软件开发工具包指定用于对话日志的 IAM 角色时，指定对话日志 IAM 角色的用户必须有权将该角色传递给 Amazon Lex。要允许用户将角色传递给 Amazon Lex，您必须向用户、角色或组授予 `PassRole` 权限。

以下策略定义要授予用户、角色或组的权限。您可以使用 `iam:AssociatedResourceArn` 和 `iam:PassedToService` 条件键来限制权限的范围。有关更多信息，请参阅用户指南中的授予用户将角色传递给 AWS 服务的权限以及 IAM 和 AWS STS 条件上下文密钥。AWS Identity and Access Management

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
```

```
            "Resource": "arn:aws:iam::account-id:role/role-name",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "lex.amazonaws.com"
                },
                "StringLike": {
                    "iam:AssociatedResourceARN": "arn:aws:lex:region:account-
id:bot:bot-name:bot-alias"
                }
            }
        }
    ]
}
```

配置对话日志

您可以使用控制台或 PutBotAlias 操作的 conversationLogs 字段来启用和禁用对话日志。您可以启用或禁用音频日志和/或文本日志。日志记录将在新自动程序会话上启动。对日志设置的更改不会体现在活动会话中。

要存储文本日志，请在您的 AWS 账户中使用一个 Amazon Log CloudWatch s 日志组。您可以使用任何有效的日志组。日志组必须与 Amazon Lex 机器人位于同一区域中。有关创建 CloudWatch 日志组的更多信息，请参阅 Amazon Logs 用户指南中的使用日志组和 CloudWatch 日志流。

要存储音频日志，请在您的 AWS 账户中使用 Amazon S3 存储桶。您可以使用任何有效的 S3 存储桶。该存储桶必须与 Amazon Lex 存储桶位于同一区域。有关创建 S3 存储桶的更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的创建存储桶。

您必须为 IAM 角色提供策略，使 Amazon Lex 能够写入配置的日志组或存储桶。有关更多信息，请参阅 为对话日志创建 IAM 角色和策略。

如果您使用创建服务相关角色 AWS Command Line Interface，则必须使用custom-suffix选项为角色添加自定义后缀，如下所示：

```
aws iam create-service-linked-role \
    --aws-service-name lex.amazon.aws.com \
    --custom-suffix suffix
```

您用于启用对话日志的 IAM 角色必须具有 iam:PassRole 权限。应将以下策略附加到角色。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::account:role/role"
        }
    ]
}
```

启用对话日志

使用控制台启用日志

1. 打开 Amazon Lex 控制台 https://console.aws.amazon.com/lex。
2. 从列表中，选择一个机器人。
3. 选择 Settings (设置) 选项卡，然后从左侧菜单中选择 Conversation logs (对话日志)。
4. 在别名列表中，选择要配置对话日志的别名的设置图标。
5. 选择是记录文本、音频还是两者。
6. 要进行文本记录，请输入 Amazon Log CloudWatch s 日志组名称。
7. 对于音频日志记录，请输入 S3 存储桶信息。
8. 可选。要加密音频日志，请选择用于加密的密 AWS KMS 钥。
9. 选择具有所需权限的 IAM 角色。
10. 选择 Save (保存) 以开始记录对话。

使用 API 启用文本日志

1. 使用 `conversationLogs` 字段中 `logSettings` 成员的条目调用 PutBotAlias 操作
   - 将 `destination` 成员设置为 `CLOUDWATCH_LOGS`
   - 将 `logType` 成员设置为 `TEXT`
   - 将`resourceArn`成员设置为日志日志组的 Amazon 资源名称 (ARN)，该组是 CloudWatch 日志的目标
2. 将 `conversationLogs` 字段的 `iamRoleArn` 成员设置为 IAM 角色的 Amazon 资源名称 (ARN)，该角色应具有在指定资源上启用对话日志所需的权限。

使用 API 启用音频日志

1. 使用 conversationLogs 字段中 logSettings 成员的条目调用 PutBotAlias 操作
   - 将 destination 成员设置为 S3
   - 将 logType 成员设置为 AUDIO
   - 将 resourceArn 成员设置为存储音频日志的 Amazon S3 存储桶的 ARN
   - 可选。要使用特定 AWS KMS 密钥加密音频日志，请设置用于加密的密钥的 ARN kmsKeyArn 成员。
2. 将 conversationLogs 字段的 iamRoleArn 成员设置为 IAM 角色的 Amazon 资源名称 (ARN)，该角色应具有在指定资源上启用对话日志所需的权限。

禁用对话日志

使用控制台禁用日志

1. 打开 Amazon Lex 控制台 https://console.aws.amazon.com/lex。
2. 从列表中，选择一个机器人。
3. 选择 Settings (设置) 选项卡，然后从左侧菜单中选择 Conversation logs (对话日志)。
4. 在别名列表中，选择要配置对话日志的别名的设置图标。
5. 清除文本日志和/或音频日志的复选框以禁用日志记录。
6. 选择 Save (保存) 以停止记录对话。

使用 API 禁用日志

- 不带 conversationLogs 字段调用 PutBotAlias 操作。

使用 API 禁用文本日志

-
  - 正在记录音频时
    - 仅对于 AUDIO，使用 logSettings 条目调用 PutBotAlias 操作。
    - 对 PutBotAlias 操作的调用不得包括针对 TEXT 的条目 logSettings。
  - 没有记录音频时
    - 不带 conversationLogs 字段调用 PutBotAlias 操作。

使用 API 禁用音频日志

-
  - 正在记录文本时
    - 仅对于 TEXT，使用 logSettings 条目调用 PutBotAlias 操作。
    - 对 PutBotAlias 操作的调用不得包括针对 AUDIO 的条目 logSettings。
  - 没有记录文本时
    - 不带 conversationLogs 字段调用 PutBotAlias 操作。

加密对话日志

您可以使用加密来帮助保护对话日志的内容。对于文本和音频日志，您可以使用 AWS KMS 客户管理 CMKs 来加密 CloudWatch 日志组和 S3 存储桶中的数据。

> **Note**
>
> Amazon Lex 仅支持对称 CMKs。不能使用非对称 CMK 来加密数据。

您可以使用 Amazon Lex 用于文本 CloudWatch 日志的日志组上的 AWS KMS 密钥启用加密。您无法在日志设置中提供 AWS KMS 密钥来启用对日志组的 AWS KMS 加密。有关更多信息，请参阅 Amazon Lo CloudWatch gs 用户指南 AWS KMS中的使用加密 CloudWatch 日志数据。

对于音频日志，您可以在 S3 存储桶上使用默认加密或指定 AWS KMS 密钥来加密您的音频对象。即使您的 S3 存储桶使用默认加密，您仍然可以指定不同的 AWS KMS 密钥来加密您的音频对象。有关更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的 Amazon Lex 默认 S3 存储桶加密。

如果您选择加密音频日志，Amazon Lex 需要 AWS KMS 权限。您需要将其他策略附加到用于对话日志的 IAM 角色。如果您对 S3 存储桶使用默认加密，则您的策略必须授予对该存储桶配置的 AWS KMS 密钥的访问权限。如果您在音频日志设置中指定了 AWS KMS 密钥，则必须授予对该密钥的访问权限。

如果您还没有为对话日志创建角色，请参阅用于对话日志的 IAM 策略。

创建用于使用 AWS KMS 密钥加密音频日志的 IAM 策略

1. 在名为 **LexConversationLogsKMSPolicy.json** 的当前目录中创建一个文档，向其中添加以下策略并保存。

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
              "kms:GenerateDataKey"
          ],
          "Resource": "kms-key-arn"
      }
  ]
}
```

2. 在中 AWS CLI，创建授予使用 AWS KMS 密钥加密音频日志的权限的 IAM 策略。

```
aws iam create-policy \
    --policy-name kms-policy-name \
    --policy-document file://LexConversationLogsKMSPolicy.json
```

3. 将该策略附加到您为对话日志创建的角色。

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::account-id:policy/kms-policy-name \
    --role-name role-name
```

在 Amazon 日志中查看文本 CloudWatch 日志

Amazon Lex 将您的对话文本日志存储在亚马逊 CloudWatch 日志中。要查看日志，您可以使用 CloudWatch 日志控制台或 API。有关更多信息，请参阅 Amazon Logs 用户指南中的使用筛选模式搜索CloudWatch 日志数据和 CloudWatch 日志见解查询语法。

使用 Amazon Lex 控制台查看日志

1. 打开 Amazon Lex 控制台 https://console.aws.amazon.com/lex。
2. 从列表中，选择一个机器人。
3. 选择 Settings (设置) 选项卡，然后从左侧菜单中选择 Conversation logs (对话日志)。
4. 选择 “文本日志” 下的链接，在 CloudWatch 控制台中查看别名的日志。

您也可以使用 CloudWatch 控制台或 API 来查看日志条目。要查找日志条目，请导航到为该别名配置的日志组。您可以在 Amazon Lex 控制台中或使用 GetBotAlias 操作查找日志的日志流前缀。

用户话语的日志条目位于多个日志流中。对话中的一个话语在一个日志流中具有一个带指定前缀的条目。日志流中的条目包含以下信息。

```
{
   "messageVersion": "1.0",
   "botName": "bot name",
   "botAlias": "bot alias",
   "botVersion": "bot version",
   "inputTranscript": "text used to process the request",
   "botResponse": "response from the bot",
   "intent": "matched intent",
   "nluIntentConfidence": "number",
   "slots": {
       "slot name": "slot value",
       "slot name": null,
       "slot name": "slot value"
       ...
   },
   "alternativeIntents": [
       {
           "name": "intent name",
           "nluIntentConfidence": "number",
           "slots": {
               "slot name": slot value,
               "slot name": null,
               "slot name": slot value
               ...
           }
       },
       {
           "name": "intent name",
           "nluIntentConfidence": number,
           "slots": {}
       }
   ],
   "developerOverride": "true" | "false",
   "missedUtterance": true | false,
   "inputDialogMode": "Text" | "Speech",
   "requestId": "request ID",
   "s3PathForAudio": "S3 path to audio file",
```

```
  "userId": "user ID",
  "sessionId": "session ID",
  "sentimentResponse": {
      "sentimentScore": "{Positive: number, Negative: number, Neutral: number,
Mixed: number}",
      "sentimentLabel": "Positive" | "Negative" | "Neutral" | "Mixed"
  },
  "slotToElicit": "slot name",
  "dialogState": "ElicitIntent" | "ConfirmIntent" | "ElicitSlot" | "Fulfilled" |
"ReadyForFulfillment" | "Failed",
  "responseCard": {
      "genericAttachments": [
          ...
      ],
      "contentType": "application/vnd.amazonaws.card.generic",
      "version": 1
   },
  "locale": "locale",
  "timestamp": "ISO 8601 UTC timestamp",
  "kendraResponse": {
     "totalNumberOfResults": number,
     "resultItems": [
         {
             "id": "query ID",
             "type": "DOCUMENT" | "QUESTION_ANSWER" | "ANSWER",
             "additionalAttributes": [
                 {
                    ...
                 }
             ],
             "documentId": "document ID",
             "documentTitle": {
                 "text": "title",
                 "highlights": null
             },
             "documentExcerpt": {
                 "text": "text",
                 "highlights": [
                     {
                         "beginOffset": number,
                         "endOffset": number,
                         "topAnswer": true | false
                     }
                 ]
```

```
                },
                "documentURI": "URI",
                "documentAttributes": []
            }
        ],
        "facetResults": [],
        "sdkResponseMetadata": {
            "requestId": "request ID"
        },
        "sdkHttpMetadata": {
            "httpHeaders": {
                "Content-Length": "number",
                "Content-Type": "application/x-amz-json-1.1",
                "Date": "date and time",
                "x-amzn-RequestId": "request ID"
            },
            "httpStatusCode": 200
        },
        "queryId": "query ID"
    },
    "sessionAttributes": {
        "attribute name": "attribute value"
        ...
     },
    "requestAttributes": {
        "attribute name": "attribute value"
        ...
     }
}
```

日志条目的内容取决于事务的结果以及自动程序和请求的配置。

- 如果 missedUtterance 字段为 true，则 intent、slots 和 slotToElicit 字段不会显示在条目中。
- 如果音频日志已禁用或者 inputDialogMode 字段为 Text，则不显示 s3PathForAudio 字段。
- 仅当您为自动程序定义了响应卡时，才会显示 responseCard 字段。
- 仅当您在请求中指定了请求属性时，才会显示 requestAttributes 映射。
- 只有在 AMAZON.KendraSearchIntent 请求搜索 Amazon Kendra 索引时，kendraResponse 字段才会出现。
- 当在机器人的 Lambda 函数中指定了替代意图时，developerOverride 字段的值为 True。

- 仅当在请求中指定了会话属性时，才会显示 `sessionAttributes` 映射。
- 仅当将自动程序配置为返回情绪值时，才会显示 `sentimentResponse` 映射。

> **Note**
>
> 输入格式可以更改，但不必对 `messageVersion` 做出相应更改。您的代码不应在有新字段时引发错误。

您必须设置角色和策略才能允许 Amazon Lex 写入 CloudWatch 日志。有关更多信息，请参阅用于对话日志的 IAM 策略。

在 Amazon S3 中访问音频日志

Amazon Lex 将对话的音频日志存储在 S3 存储桶中。

使用控制台访问音频日志

1. 打开 Amazon Lex 控制台 https://console.aws.amazon.com/lex。
2. 从列表中，选择一个机器人。
3. 选择 Settings (设置) 选项卡，然后从左侧菜单中选择 Conversation logs (对话日志)。
4. 选择音频日志下的链接以在 Amazon S3 控制台中访问该别名的日志。

您也可以使用 Amazon S3 控制台或 API 来访问音频日志。您可以在 Amazon Lex 控制台或 `GetBotAlias` 操作响应的 `resourcePrefix` 字段中查看音频文件的 S3 对象键前缀。

使用 CloudWatch 指标监控对话日志状态

使用 Amazon CloudWatch 监控您的对话日志的交付指标。您可以在指标上设置警报，以便在日志记录发生问题时获取通知。

Amazon Lex 在 `AWS/Lex` 命名空间中为对话日志提供了四个指标：

- `ConversationLogsAudioDeliverySuccess`
- `ConversationLogsAudioDeliveryFailure`
- `ConversationLogsTextDeliverySuccess`

- `ConversationLogsTextDeliveryFailure`

有关更多信息，请参阅 CloudWatch 对话日志的指标。

成功指标表明 Amazon Lex 已成功将音频或文本日志写入其目标。

失败指标表明 Amazon Lex 无法将音频或文本日志传输到指定目标。这通常是配置错误。当您的失败指标大于零时，请检查以下内容：

- 确保 Amazon Lex 是 IAM 角色的可信实体。
- 对于文本记录，请确保 CloudWatch 日志组存在。对于音频日志记录，请确保 S3 存储桶存在。
- 确保 Amazon Lex 用于访问 CloudWatch 日志日志组或 S3 存储桶的 IAM 角色具有日志组或存储桶的写入权限。
- 确保 S3 存储桶与 Amazon Lex 机器人位于相同的区域，并且属于您的账户。
- 如果您使用 AWS KMS 密钥进行 S3 加密，请确保没有任何策略阻止 Amazon Lex 使用您的密钥，并确保您提供的 IAM 角色具有必要的 AWS KMS 权限。有关更多信息，请参阅 用于对话日志的 IAM 策略。

使用 Amazon Lex API 管理会话

用户启动与您的机器人的对话时，Amazon Lex 会创建一个会话。您的应用程序与 Amazon Lex 之间交换的信息组成了对话的会话状态。当您发出请求时，由您指定的自动程序名称与用户标识符构成的组合会识别会话。有关用户标识符的更多信息，请参阅 PostContent 或 PostText 操作中的 `userId` 字段。

来自会话操作的响应包括一个唯一的会话标识符，用于标识用户的特定会话。您可在测试期间或者在帮助对自动程序进行故障排查时使用此标识符。

您可以修改在应用程序和自动程序之间发送的会话状态。例如，您可以创建和修改其中包含会话自定义信息的会话属性，也可以通过设置对话上下文来更改对话流，以解释下一个表达。

您可以通过两种方式来更新会话状态。第一种方式是使用在每轮对话之后调用的 Lambda 函数和 `PostContent` 或 `PostText` 操作。有关更多信息，请参阅 使用 Lambda 函数。另一种方式是在应用程序中使用 Amazon Lex 运行时 API 来更改会话状态。

Amazon Lex 运行时 API 提供了若干操作，使您可以管理机器人对话的会话信息。这些操作包括 PutSession 操作、GetSession 操作和 DeleteSession 操作。使用这些操作，您可以获取自动程序与用户会话的状态信息，还可以对状态执行更精细的控制。

如果您希望获取会话的当前状态，请使用 GetSession 操作。此操作会返回会话的当前状态，包括与用户进行的对话的状态、已经设置的任何会话属性，以及用户与之交互的最近三个目的的槽值。

PutSession 操作可让您直接操作当前会话状态。您可以设置自动程序接下来将执行的对话操作的类型。这让您可以控制与自动程序的对话流。将对话操作 type 字段设置为 Delegate，可以让 Amazon Lex 确定机器人的下一个操作。

您可以使用 PutSession 操作创建与自动程序的新会话，并设置自动程序在开始时应使用的目的。您还可以使用 PutSession 操作将一个目的更改为另一个。创建会话或更改目的时，您还可以设置会话状态，例如槽值和会话属性。新目的完成后，您可以选择重启之前的目的。您可以使用 GetSession 操作，从 Amazon Lex 获取以前意图的对话状态，然后使用该信息来设置意图的对话状态。

来自 PutSession 操作的响应包含与 PostContent 操作相同的信息。您可以使用此信息向用户提示信息的下一部分，就像您对待 PostContent 操作的响应一样。

使用 DeleteSession 操作以删除现有会话并使用新会话重新开始。例如，在您测试自动程序时，可以使用 DeleteSession 操作从自动程序删除测试会话。

会话操作可用于履行 Lambda 函数。例如，如果您的 Lambda 函数返回 Failed 作为履行状态，您可以通过 PutSession 操作将对话操作类型设置为 close，将 fulfillmentState 设置为 ReadyForFulfillment 以重试履行步骤。

您可以对会话操作采取下列措施：

- 让自动程序启动对话而不是等待用户启动。
- 在对话期间切换目的。
- 返回到以前的目的。
- 在交互过程中启动或重新启动对话。
- 验证槽值并让自动程序重新提示无效的值。

下文分别详细介绍了这些操作。

切换目的

您可以使用 PutSession 操作将一个目的切换为另一个。您还可以使用它切换回以前的目的。您可以使用 PutSession 操作为新目的设置会话属性值或槽值。

- 调用 PutSession 操作。将目的名称设置为新目的的名称，并将对话操作设置为 Delegate。您还可以为新目的设置所需的任意槽值和会话属性。

- Amazon Lex 使用新意图启动与用户的对话。

恢复以前的目的

要恢复以前的目的，可以使用 GetSession 操作获取目的的摘要，然后使用 PutSession 操作将目的设置回其之前的对话状态。

- 调用 GetSession 操作。来自操作的响应包含用户与之交互的最近三个目的的对话状态摘要。
- 使用目的摘要中的信息调用 PutSession 操作。这会将用户返回到对话中相同位置的上一个目的。

在某些情况下，可能需要恢复用户与自动程序的对话。例如，假设您创建了一个客户服务自动程序。您的应用程序确定用户需要与客户服务代表交谈。在与用户交谈之后，客户服务代表可以使用所收集的信息将对话引导回自动程序。

要恢复会话，请使用类似于下文的步骤：

- 您的应用程序确定用户需要与客户服务代表交谈。
- 使用 GetSession 操作获取目的的当前对话状态。
- 客户服务代表与用户交谈并解决问题。
- 使用 PutSession 操作设置目的的对话状态。这可能包括设置槽值、设置会话属性或者更改目的。
- 自动程序恢复与用户的对话。

您可以使用 PutSession 操作 checkpointLabel 参数标注意图，以便稍后找到它。例如，要求客户提供信息的机器人可能会在客户收集信息时进入 Waiting 意图。机器人为当前意图创建检查点标签，然后启动 Waiting 意图。当客户返回时，机器人可以使用检查点标签找到以前的意图并切换回来。

意图必须存在于由 GetSession 操作返回的 recentIntentSummaryView 结构中。如果您在 GetSession 操作请求中指定了检查点标签，则最多将返回三个具有该检查点标签的意图。

- 使用 GetSession 操作获取会话的当前状态。
- 使用 PutSession 操作将检查点标签添加到最后一个意图。如有必要，您可以使用此 PutSession 调用切换到不同的意图。
- 当要切换回标记的意图时，调用 GetSession 操作以返回最近的意图列表。您可以使用 checkpointLabelFilter 参数，以便 Amazon Lex 仅返回具有指定检查点标签的意图。

启动新会话

如果您希望自动程序开始与用户的对话，可以使用 PutSession 操作。

- 创建没有槽的欢迎目的，并创建一条总结性消息向用户提示阐明目的。例如，“您希望订购什么？ 您可以说‘订一杯饮料’或‘订一个披萨’。”
- 调用 PutSession 操作。将目的名称设置为欢迎目的的名称，并将对话操作设置为 Delegate。
- Amazon Lex 将使用您的欢迎意图中的提示做出响应，以启动与用户的对话。

验证槽值

您可以使用客户端应用程序验证对自动程序的响应。如果响应无效，您可以使用 PutSession 操作获取用户的新响应。例如，假设您的鲜花订购自动程序只能卖郁金香、玫瑰和百合。如果用户订购康乃馨，您的应用程序可以执行以下操作：

- 检查 PostText 或 PostContent 响应返回的槽值。
- 如果槽值无效，则调用 PutSession 操作。您的应用程序应清除槽值，设置 slotToElicit 字段，并将 dialogAction.type 值设置为 elicitSlot。（可选）如果您希望更改 Amazon Lex 用于引入插槽值的消息，可以设置 message 和 messageFormat 字段。

自动程序部署选项

目前，Amazon Lex 提供以下机器人部署选项：

- AWS 移动软件开发工具包 — 您可以使用 AWS 移动设备构建与 Amazon Lex 通信的移动应用程序 SDKs。
- Facebook Messenger — 您可将 Facebook Messenger 页面与 Amazon Lex 机器人集成，以便 Facebook 上的最终用户能够与机器人通信。在当前实现中，此集成仅支持文本输入消息。
- Slack — 您可以将 Amazon Lex 机器人与 Slack 消息收发应用程序集成。
- Twilio — 您可以将 Amazon Lex 机器人与 Twilio Simple Messaging Service (SMS) 集成。

有关示例，请参阅 部署 Amazon Lex 机器人。

内置目的和槽类型

为了更方便地创建机器人，Amazon Lex 支持使用标准内置意图和插槽类型。

主题

内置目的

对于常见操作，您可以使用标准内置意图库。要基于内置意图创建意图，请在控制台中选择一个内置意图，为其指定新名称。新意图将具有基础意图的配置，例如示例言语。

在当前实现中，不能执行以下操作：

- 向基础意图添加示例言语，或从基础意图中删除示例言语
- 为内置意图配置槽

向机器人添加内置意图

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 选择要向其添加内置意图的机器人。
3. 在导航窗格中，选择目的旁边的加号 (+)。
4. 对于添加目的，选择搜索现有目的。
5. 在搜索意图框中，键入要添加到您的机器人的内置意图的名称。
6. 对于复制内置意图，为意图提供名称，然后选择添加。
7. 根据需要为您的机器人配置意图。

主题

- AMAZON.StartOverIntent
- AMAZON.StopIntent

> **Note**
>
> 对于英语（美国）(en-US) 区域设置，Amazon Lex 支持 Alexa 标准内置意图中的意图。有关内置目的的列表，请参阅 Alexa Skills Kit 中的标准内置目的。
> Amazon Lex 不支持以下意图：
>
> - `AMAZON.YesIntent`
> - `AMAZON.NoIntent`
> - Alexa Skills Kit 中的内置目的库中的目的

AMAZON.CancelIntent

对表示用户要取消当前交互的词语和短语的响应。在结束与用户的交互之前，您的应用程序可以使用此意图来删除插槽类型值和其他属性。

常见言语：

- 取消
- 没关系
- 算了

AMAZON.FallbackIntent

当意图的用户输入与机器人的预期不符时，您可以配置 Amazon Lex 以调用回退意图。例如，如果用户输入“我想要订购糖果”，与 `OrderFlowers` 机器人中的意图不匹配，Amazon Lex 会调用回退意图来处理该响应。

您可以通过向自动程序添加内置 `AMAZON.FallbackIntent` 目的类型来添加回退目的。您可以使用 PutBot 操作指定目的，也可以从控制台的内置目的列表中选择目的。

调用回退意图分两步。在第一步中，基于用户的输入来匹配回退意图。匹配回退意图时，自动程序的行为方式取决于为提示配置的重试次数。例如，如果确定目的的最大尝试次数为 2，则自动程序会在调用回退目的之前两次返回自动程序的澄清提示。

在以下情况下，Amazon Lex 与回退意图匹配：

- 用户输入到意图的内容不符合自动程序的预期
- 音频输入为噪声，或文本输入未被识别为单词。
- 用户的输入不明确且 Amazon Lex 无法确定要调用的意图。

在以下情况下调用回退意图：

- 对话开始后，经过配置的澄清尝试次数后，自动程序无法将用户输入识别为目的。
- 经过配置的尝试次数后，意图无法将用户输入识别为槽位值。
- 经过配置的尝试次数后，意图无法将用户输入识别为对确认提示的响应。

您可以将以下各项与回退目的结合使用：

- 履行 Lambda 函数
- 结论语句
- 跟进提示

您不能将以下内容添加到回退意图：

- 言语
- 槽值
- 初始化和验证 Lambda 函数
- 确认提示

如果您同时为机器人配置了取消语句和回退意图，Amazon Lex 将使用回退意图。如果您需要机器人具有取消语句，则可以将履行功能用于回退意图，以提供与取消语句相同的行为。有关更多信息，请参阅 PutBot 操作的 `abortStatement` 参数。

使用澄清提示

如果您向自动程序提供了一个澄清提示，则该提示将用于向用户征求有效目的。该澄清提示将重复您配置的次数。此后，将调用回退目的。

如果您在创建机器人时未设置澄清提示，并且用户未使用有效意图开始对话，则 Amazon Lex 会立即调用您的回退意图。

当您在没有澄清提示的情况下使用回退意图时，Amazon Lex 在以下情况下不会调用回退意图：

- 当用户响应跟进提示但不提供目的时。例如，在响应后续提示（“您今天还想要其他东西吗？”）时，用户说“是”。由于 Amazon Lex 没有发送给用户以获取用户意图的澄清提示，因此它会返回“400 错误请求”异常。
- 使用 AWS Lambda 函数时，会返回`ElicitIntent`对话框类型。由于 Amazon Lex 没有用于获取用户意图的澄清提示，因此它会返回“400 错误请求”异常。
- 当使用 `PutSession` 操作时，您发送一个 `ElicitIntent` 对话类型。由于 Amazon Lex 没有用于获取用户意图的澄清提示，因此它会返回“400 错误请求”异常。

使用具有回退意图的 Lambda 函数

调用回退目的时，响应取决于 PutIntent 操作的 `fulfillmentActivity` 参数设置。自动程序执行下列操作之一：

- 将意图信息返回给客户端应用程序。
- 调用履行 Lambda 函数。它通过为会话设置的会话变量调用该函数。

有关设置在调用回退目的时的响应的更多信息，请参见 PutIntent 操作的 `fulfillmentActivity` 参数。

如果在回退意图中使用履行 Lambda 函数，则可以使用此函数来调用另一个意图或与用户进行某种形式的通信，例如收集回叫号码或与客户服务代表进行会话。

您可以在回退意图 Lambda 函数中执行任何您可以在任何其他意图的履行函数中执行的操作。有关使用创建发货功能的更多信息 AWS Lambda，请参阅使用 Lambda 函数。

回退意图可以在同一会话中多次调用。例如，假设您的 Lambda 函数使用 `ElicitIntent` 对话操作来提示用户输入一个不同的意图。如果 Amazon Lex 在配置的尝试次数后无法推断用户的意图，将再次调用回退意图。当用户在配置的尝试次数后未使用有效的槽位值进行响应时，它也会调用回退意图。

您可以配置一个 Lambda 函数来跟踪使用会话变量调用回退意图的次数。如果该意图的调用次数超过了 Lambda 函数中设置的阈值，则该函数可以采取不同的操作。有关会话变量的更多信息，请参阅设置会话属性。

AMAZON.HelpIntent

对表示用户在与机器人交互时需要帮助的词语或短语的响应。调用该意图时，您可以对您的 Lambda 函数或应用程序进行配置，以便提供有关机器人功能的信息、询问有关需要帮助的方面的后续问题，或者将交互移交给人工客服。

常见言语：

- 帮助
- 我需要帮助
- 可以帮帮我吗

AMAZON.KendraSearchIntent

要搜索已使用 Amazon Kendra 编制索引的文档，请使用 AMAZON.KendraSearchIntent 意图。在与用户进行的对话中，如果 Amazon Lex 无法确定下一个操作，则会触发搜索意图。

AMAZON.KendraSearchIntent 仅适用于英语（美国）(en-US) 区域设置以及美国东部（弗吉尼亚州北部）、美国西部（俄勒冈州）和欧洲地区（爱尔兰）区域。

Amazon Kendra 是一项 machine-learning-based搜索服务，用于索引 PDF 文档或微软 Word 文件等自然语言文档。它可以搜索已编制索引的文档并为问题返回以下类型的响应：

- 答案
- 可能是问题答案的常见问题解答条目
- 与问题相关的文档

有关使用 AMAZON.KendraSearchIntent 的示例，请参阅示例：为 Amazon Kendra 索引创建常见问题机器人。

如果您为机器人配置了 AMAZON.KendraSearchIntent 意图，则 Amazon Lex 在无法确定插槽或意图的用户言语时，会调用该意图。例如，如果您的机器人引发名为“披萨配料”的插槽类型的响应，而用户问的是“什么是披萨？”，Amazon Lex 将调用 AMAZON.KendraSearchIntent 来处理此问题。如果没有来自 Amazon Kendra 的响应，则对话将按照机器人中的配置继续进行。

当您在同一机器人中同时使用 AMAZON.KendraSearchIntent 和 AMAZON.FallbackIntent 时，Amazon Lex 将按照以下方式使用意图：

1. Amazon Lex 调用 AMAZON.KendraSearchIntent。该意图调用 Amazon Kendra Query 操作。
2. 如果 Amazon Kendra 返回响应，则 Amazon Lex 向用户显示该结果。
3. 如果没有来自 Amazon Kendra 的响应，则 Amazon Lex 将重新提示用户。下一个操作取决于用户的响应。
   - 如果用户的响应包含 Amazon Lex 可识别的言语（例如填充插槽值或确认意图），则与用户的对话将按照机器人的配置继续进行。
   - 如果用户的响应未包含 Amazon Lex 可识别的言语，则 Amazon Lex 将再次调用 Query 操作。
4. 如果在配置的重试次数之后没有响应，Amazon Lex 将调用 AMAZON.FallbackIntent 并结束与用户的对话。

有三种方法可以使用 AMAZON.KendraSearchIntent 向 Amazon Kendra 发出请求：

- 让搜索意图为您提出请求。Amazon Lex 以用户的言语作为搜索字符串调用 Amazon Kendra。在创建意图时，您可以定义限制 Amazon Kendra 返回的响应数的查询筛选条件字符串。Amazon Lex 使用查询请求中的筛选条件。
- 使用您的对话 Lambda 函数向请求添加其他查询参数以缩小搜索结果范围。您将包含 Amazon Kendra 查询参数的 kendraQueryFilterString 字段添加到 delegate 对话操作。使用 Lambda 函数向请求添加查询参数时，这些查询参数将优先于您在创建意图时定义的查询筛选条件。
- 使用对话 Lambda 函数创建新查询。您可以创建 Amazon Lex 发送的完整 Amazon Kendra 查询请求。您可以在 delegate 对话操作的 kendraQueryRequestPayload 字段中指定查询。kendraQueryRequestPayload 字段优先于 kendraQueryFilterString 字段。

要在创建机器人时指定 queryFilterString 参数，或者在对话 Lambda 函数中调用 delegate 操作时指定 kendraQueryFilterString 字段，请指定用作 Amazon Kendra 查询的属性筛选条件的字符串。如果字符串不是有效的属性筛选器，您将在运行时收到 InvalidBotConfigException 异常。有关属性筛选条件的更多信息，请参阅《Amazon Kendra 开发人员指南》中的使用文档属性筛选查询。

要控制 Amazon Lex 发送到 Amazon Kendra 的查询，您可以在对话 Lambda 函数的 kendraQueryRequestPayload 字段中指定查询。如果查询无效，则 Amazon Lex 返回 InvalidLambdaResponseException 异常。有关更多信息，请参阅《Amazon Kendra 开发人员指南》中的查询操作。

有关如何使用 AMAZON.KendraSearchIntent 的示例，请参阅示例：为 Amazon Kendra 索引创建常见问题机器人。

Amazon Kendra 搜索的 IAM 策略

要使用该AMAZON.KendraSearchIntent意图，您必须使用一个提供 AWS Identity and Access Management (IAM) 策略的角色，这些策略允许 Amazon Lex 担任有权调用 Amazon Kendra Query 意图的运行时角色。您使用的 IAM 设置取决于您是使用 Amazon Lex 控制台创建，还是使用 AWS 开发工具包或 AWS Command Line Interface (AWS CLI) 创建。AMAZON.KendraSearchIntent使用控制台时，您可以选择向 Amazon Lex 服务相关角色添加调用 Amazon Kendra 的权限，或使用专门用于调用 Amazon Kendra Query 操作的角色。使用 AWS CLI 或 SDK 创建 Intent 时，必须使用专门用于调用Query操作的角色。

附加权限

您可以使用控制台将访问 Amazon Kendra Query 操作的权限附加到默认 Amazon Lex 服务相关角色。当您将权限附加到服务相关角色时，无需专门创建和管理运行时角色即可连接到 Amazon Kendra 索引。

用于访问 Amazon Lex 控制台的用户、角色或组必须有权限管理角色策略。将以下 IAM 策略附加到控制台访问角色。当您授予这些权限时，角色将有权更改现有服务相关角色策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy",
                "iam:GetRolePolicy"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
        },
        {
            "Effect": "Allow",
            "Action": "iam:ListRoles",
            "Resource": "*"
        }
    ]
}
```

指定角色

您可以使用控制台 AWS CLI、或 API 来指定在调用 Amazon Kendra Query 操作时要使用的运行时角色。

用于指定运行时角色的用户、角色或组必须具有 iam:PassRole 权限。以下策略定义权限。您可以使用 iam:AssociatedResourceArn 和 iam:PassedToService 条件上下文键进一步限制权限的范围。有关更多信息，请参阅AWS Identity and Access Management 用户指南中的 IAM 和 AWS STS 条件上下文密钥。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::account:role/role"
        }
    ]
}
```

Amazon Lex 调用 Amazon Kendra 时所需使用的运行时角色必须具有 kendra:Query 权限。当您使用现有 IAM 角色获取调用 Amazon Kendra Query 操作的权限时，该角色必须附加以下策略。

您可以使用 IAM 控制台、IAM API 或 AWS CLI 创建策略并将其附加到角色。这些说明使用 AWS CLI 创建角色和策略。

> **Note**
>
> 以下代码针对 Linux 和 macOS 编排了格式。对于 Windows，将 Linux 行继续符 (\) 替换为脱字号 (^)。

向角色添加 Query 操作权限

1. 在当前目录中创建一个名为 **KendraQueryPolicy.json** 的文档，向其中添加以下代码并保存

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "kendra:Query"
            ],
            "Resource": [
                "arn:aws:kendra:region:account:index/index ID"
            ]
        }
    ]
}
```

2. 在中 AWS CLI，运行以下命令来创建用于运行 Amazon Kendra Query 操作的 IAM 策略。

```
aws iam create-policy \
    --policy-name query-policy-name \
    --policy-document file://KendraQueryPolicy.json
```

3. 将该策略附加到用于调用 Query 操作的 IAM 角色。

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::account-id:policy/query-policy-name
    --role-name role-name
```

您可以选择更新 Amazon Lex 服务相关角色或使用您在为机器人创建 AMAZON.KendraSearchIntent 时所创建的角色。以下过程演示如何选择要使用的 IAM 角色。

为亚马逊指定运行时角色。 KendraSearchIntent

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 选择要向其添加 AMAZON.KendraSearchIntent 的自动程序。
3. 选择意图旁边的加号（+）。
4. 在添加意图中，选择搜索现有意图。
5. 在搜索意图中，输入 **AMAZON.KendraSearchIntent**，然后选择添加。
6. 在复制内置意图中，输入意图的名称，如 **KendraSearchIntent**，然后选择添加。
7. 打开 Amazon Kendra 查询部分。
8. 对于 IAM 角色，选择下列选项之一：

- 要更新 Amazon Lex 服务相关角色以便机器人能够查询 Amazon Kendra 索引，请选择添加 Amazon Kendra 权限。
- 要使用有权调用 Amazon Kendra Query 操作的角色，请选择使用现有角色。

使用请求和会话属性作为筛选器

要从来自 Amazon Kendra 的响应中筛选出与当前对话相关的项目，请在创建机器人时添加 queryFilterString 参数以使用会话和请求属性作为筛选条件。您可以在创建意图时指定属性的占位符，以便 Amazon Lex V2 在调用 Amazon Kendra 之前将其替换为值。有关请求属性的更多信息，请参阅设置请求属性。有关会话属性的更多信息，请参阅 设置会话属性。

以下是使用字符串来筛选 Amazon Kendra 查询的 queryFilterString 参数示例。

```
"{"equalsTo": {"key": "City", "value": {"stringValue": "Seattle"}}}"
```

以下是使用名为 "SourceURI" 的会话属性来筛选 Amazon Kendra 查询的 queryFilterString 参数示例。

```
"{"equalsTo": {"key": "SourceURI","value": {"stringValue": "[FileURL]"}}}"
```

以下是使用名为 "DepartmentName" 的请求属性来筛选 Amazon Kendra 查询的 queryFilterString 参数示例。

```
"{"equalsTo": {"key": "Department","value": {"stringValue": "((DepartmentName))"}}}"
```

AMAZON.KendraSearchInteng 筛选条件使用的格式与 Amazon Kendra 搜索筛选条件的格式相同。有关更多信息，请参阅《Amazon Kendra 开发人员指南》中的使用文档属性筛选搜索结果。

与 AMAZON.KendraSearchIntent 一起使用的查询筛选条件字符串必须确保每个筛选条件的首字母为小写字母。例如，以下是 AMAZON.KendraSearchIntent 的有效查询筛选条件。

```
{
    "andAllFilters": [
        {
            "equalsTo": {
                "key": "City",
                "value": {
                    "stringValue": "Seattle"
```

```
                }
            }
        },
        {
            "equalsTo": {
                "key": "State",
                "value": {
                    "stringValue": "Washington"
                }
            }
        }
    ]
}
```

使用搜索响应

Amazon Kendra 在意图的 conclusion 语句中返回搜索的响应。除非履行 Lambda 函数生成结论消息，否则意图必须具有 conclusion 语句。

Amazon Kendra 有四种类型的响应。

- x-amz-lex:kendra-search-response-question_answer-question-<N> — 与搜索匹配的常见问题中的问题。
- x-amz-lex:kendra-search-response-question_answer-answer-<N> — 与搜索匹配的常见问题中的答案。
- x-amz-lex:kendra-search-response-document-<N> — 索引中与言语文本相关的文档摘录。
- x-amz-lex:kendra-search-response-document-link-<N> — 索引中与言语文本相关的文档的 URL。
- x-amz-lex:kendra-search-response-answer-<N> — 索引中能作为问题答案的文档摘录。

在 request 属性中返回响应。每个属性最多可以具有五个响应，编号为 1 到 5。有关响应的更多信息，请参阅《Amazon Kendra 开发人员指南》中的响应类型。

conclusion 语句必须具有一个或多个消息组。每个消息组都包含一条或多条消息。每条消息均可以包含一个或多个占位符变量，这些变量替换为来自 Amazon Kendra 的响应中的请求属性。消息组中必须至少有一条消息的所有变量都由运行时响应中的请求属性值替换，或者组中必须有一条没有占位符变量的消息。请求属性使用双括号 (“((“”))”) 进行设置。以下消息组消息与来自 Amazon Kendra 的任何响应匹配：

- “我为你找到了一个常见问题解答问题：((x-amz-lex: kendra-search-response-question _answer-question-1))，答案是 ((: _answer-answer-answer-1))” x-amz-lex kendra-search-response-question
- “我找到了一份有用的文档的摘录：((x-amz-lex: kendra-search-response-document -1))”
- “我想你的问题的答案是 ((x-amz-lex: kendra-search-response-answer -1))”

使用 Lambda 函数管理请求和响应

AMAZON.KendraSearchIntent 意图可以使用您的对话代码挂钩和履行代码挂钩来管理对 Amazon Kendra 的请求和响应。当您想要修改发送到 Amazon Kendra 的查询时，请使用对话代码挂钩 Lambda 函数；当您想要修改响应时，请使用履行代码挂钩 Lambda 函数。

使用对话代码挂钩创建查询

您可以使用对话代码挂钩创建要发送到 Amazon Kendra 的查询。使用对话代码挂钩是可选的。如果您未指定对话代码挂钩，则 Amazon Lex 会根据用户言语来构造查询并使用您在配置意图时提供的 queryFilterString（如有）。

您可以在对话代码挂钩响应中使用两个字段来修改对 Amazon Kendra 的请求：

- kendraQueryFilterString — 使用此字符串为 Amazon Kendra 请求指定属性筛选条件。您可以使用索引中定义的任何索引字段筛选查询。有关筛选字符串的结构，请参阅《Amazon Kendra 开发人员指南》中的使用文档属性筛选查询。如果指定的筛选器字符串无效，则会出现 InvalidLambdaResponseException 异常。kendraQueryFilterString 字符串将覆盖为意图配置的 queryFilterString 中指定的任何查询字符串。
- kendraQueryRequestPayload — 使用此字符串指定 Amazon Kendra 查询。您的查询可以使用 Amazon Kendra 的任何功能。如果您没有指定有效的查询，则会出现 InvalidLambdaResponseException 异常。有关更多信息，请参阅《Amazon Kendra 开发人员指南》中的查询。

创建筛选条件或查询字符串后，将响应发送到 Amazon Lex，并将响应的 dialogAction 字段设置为 delegate。Amazon Lex 将查询发送到 Amazon Kendra，然后将查询响应返回给履行代码挂钩。

对响应使用实现代码挂钩

在 Amazon Lex 将查询发送到 Amazon Kendra 后，查询响应将返回到 AMAZON.KendraSearchIntent 履行 Lambda 函数。代码挂钩的输入事件包含来自 Amazon Kendra 的完整响应。查询数据与 Amazon Kendra Query 操作返回的结构相同。有关更多信息，请参阅《Amazon Kendra 开发人员指南》中的查询响应语法。

实现代码挂钩是可选的。如果代码挂钩不存在，或者代码挂钩未在响应中返回消息，则 Amazon Lex 将对响应使用 `conclusion` 语句。

示例：为 Amazon Kendra 索引创建常见问题机器人

此示例创建一个使用 Amazon Kendra 索引为用户的问题提供答案的 Amazon Lex 机器人。常见问题解答自动程序为用户管理对话。它使用 `AMAZON.KendraSearchIntent` 意图查询索引并向用户提供响应。创建自动程序：

1. 创建一个自动程序，您的客户将与其交互以从其获取答案。
2. 创建自定义意图。您的自动程序至少需要一个至少具有一种表达的目的。此意图使您能够构建自动程序，但不用于其他方面。
3. 将 `KendraSearchIntent` 意图添加到机器人中，并将其配置为与 Amazon Kendra 索引配合使用。
4. 通过询问可用存储在 Amazon Kendra 索引中的文档回答的问题来测试机器人。

在使用此示例之前，您需要创建 Amazon Kendra 索引。有关更多信息，请参阅《Amazon Kendra 开发人员指南》中的 Getting started with an S3 bucket (console)。

创建常见问题解答自动程序

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 在导航窗格中，选择自动程序。
3. 选择创建。
4. 选择 Custom bot (自定义自动工具)。按下面所示配置自动程序：
   - 机器人名称 — 为机器人提供一个指示其用途的名称，例如 **`KendraTestBot`**。
   - 输出语音 — 选择无。
   - 会话超时 — 输入 **5**。
   - 情绪分析 — 选择否。
   - COPPA — 选择否。
   - 用户言语存储 — 选择不存储。
5. 选择创建。

要成功构建自动程序，您必须至少创建一个至少具有一种示例表达的目的。该意图是构建 Amazon Lex 机器人所必需的，但不用于常见问题响应。目的的表达不得应用于客户询问的任何问题。

创建所需的目的

1. 在自动程序入门页面上，选择创建目的。
2. 对于添加目的，选择创建目的。
3. 在创建目的对话框中，为该目的提供一个名称，例如 **RequiredIntent**。
4. 对于示例表达，键入一种表达，例如 **Required utterance**。
5. 选择保存意图。

现在，创建搜索 Amazon Kendra 索引的意图以及它应返回的响应消息。

创建亚马逊。 KendraSearchIntent 意图和响应消息

1. 在导航窗格中，选择目的旁边的加号 (+)。
2. 对于添加目的，选择搜索现有目的。
3. 在搜索意图框中，输入 **AMAZON.KendraSearchIntent**，然后从列表中选择它。
4. 对于复制内置目的，为目的提供名称，如 **KendraSearchIntent**，然后选择添加。
5. 在目的编辑器中，选择 Amazon Kendra 查询以打开查询选项。
6. 从 Amazon Kendra 索引菜单中，选择您希望目的搜索的索引。
7. 在响应部分中，添加以下三条消息：

```
I found a FAQ question for you: ((x-amz-lex:kendra-search-response-question_answer-
question-1)) and the answer is ((x-amz-lex:kendra-search-response-question_answer-
answer-1)).
I found an excerpt from a helpful document: ((x-amz-lex:kendra-search-response-
document-1)).
I think the answer to your questions is ((x-amz-lex:kendra-search-response-
answer-1)).
```

8. 选择保存意图，然后选择构建以构建自动程序。

最后，使用控制台测试窗口来测试来自自动程序的响应。您的问题应位于索引支持的域中。

测试常见问题解答自动程序

1. 在控制台测试窗口中，为您的索引键入一个问题。
2. 验证测试窗口的响应部分中的答案。
3. 要为其他问题重置测试窗口，请选择清除聊天历史记录。

AMAZON.PauseIntent

对允许用户暂停与机器人的交互以便稍后返回到该交互的词语和短语的响应。您的 Lambda 函数或应用程序需要将意图数据保存在会话变量中，或者在恢复当前意图时，您需要使用 GetSession 操作来检索意图数据。

常见言语：

- 暂停
- 暂停一下

AMAZON.RepeatIntent

对允许用户重复上一条消息的词语和短语的响应。您的应用程序需要使用 Lambda 函数将之前的意图信息保存在会话变量中，或者您需要使用 GetSession 操作来获取之前的意图信息。

常见言语：

- 重复
- 再说一遍
- 重复一遍

AMAZON.ResumeIntent

对允许用户恢复上一个已暂停意图的词语和短语的响应。您的 Lambda 函数或应用程序必须管理恢复上一个意图所需的信息。

常见言语：

- 继续
- 继续
- 继续下去

AMAZON.StartOverIntent

对允许用户停止处理当前意图并从头开始的词语和短语的响应。您可以使用 Lambda 函数或 `PutSession` 操作再次引发第一个插槽。

常见言语：

- 从头开始
- 重新开始
- 再次开始

AMAZON.StopIntent

对表示用户想要停止处理当前意图并结束与机器人的交互的词语和短语的响应。您的 Lambda 函数或应用程序应清除所有现有属性和插槽类型值，然后结束该交互。

常见言语：

- stop
- off
- 不要说了

内置槽类型

Amazon Lex 支持内置插槽类型，用于定义如何识别和处理插槽中的数据。您可以在您的意图中创建这些类型的槽。因此，无需为常用槽数据 (如日期、时间和位置) 创建枚举值。内置槽类型没有版本。

槽类型	简短描述	支持的区域设置	
AMAZON.Airport	识别代表机场的词语。	所有区域设置	
亚马逊。 AlphaNumeric	识别由字母和数字组成的单词。	除韩语 (ko-KR) 之外的所有区域设置	
AMAZON.City	识别代表城市的词语。	所有区域设置	

槽类型	简短描述	支持的区域设置
AMAZON.Country	识别代表国家/地区的词语。	所有区域设置
AMAZON.DATE	识别代表日期的词语并将其转换为标准格式。	所有区域设置
AMAZON.DURATION	识别代表持续时间的词语并将其转换为标准格式。	所有区域设置
亚马逊。 EmailAddress	识别表示电子邮件地址的词语并将其转换为标准电子邮件地址。	所有区域设置
亚马逊。 FirstName	识别代表名字的词语。	所有区域设置
亚马逊。 LastName	识别代表姓氏的词语。	所有区域设置
AMAZON.NUMBER	识别数字词语并将其转换为数字。	所有区域设置
AMAZON.Percentage	识别表示百分比的词语并将其转换为一个数字和一个百分比符号 (%)。	所有区域设置
亚马逊。 PhoneNumber	识别表示电话号码的词语并将其转换为数字字符串。	所有区域设置
亚马逊。 SpeedUnit	识别表示速度单位的词语并将其转换为标准缩写。	英语（美国）(en-US)

槽类型	简短描述	支持的区域设置	
AMAZON.State	识别代表省/市/自治区的词语。	所有区域设置	
亚马逊。 StreetName	识别代表街道名称的词语。	除英语（美国）(en-US) 之外的所有区域设置	
AMAZON.TIME	识别指示时间的词语并将其转换为时间格式。	所有区域设置	
亚马逊。 WeightUnit	识别表示重量单位的词语并将其转换为标准缩写	英语（美国）(en-US)	

> **Note**
>
> 对于英语（美国）(en-US) 区域设置，Amazon Lex 支持 Alexa Skills Kit 中的插槽类型。有关可用内置槽类型的列表，请参阅 Alexa Skills Kit 文档中的槽类型参考。
>
> - Amazon Lex 不支持 `AMAZON.LITERAL` 或 `AMAZON.SearchQuery` 内置插槽类型。

AMAZON.Airport

提供机场列表。示例包括：

- 约翰·菲茨杰拉德·肯尼迪国际机场
- 墨尔本机场

亚马逊。 AlphaNumeric

识别由字母和数字组成的字符串，例如 **APQ123**。

此插槽类型在韩语 (ko-KR) 区域设置中不可用。

您可以对包含以下内容的字符串使用 AMAZON.AlphaNumeric 槽类型：

- 字母字符，例如 **ABC**
- 数字字符，例如 **123**
- 字母数字字符的组合，例如 **ABC123**

您可以向 AMAZON.AlphaNumeric 槽类型添加正则表达式以验证为槽输入的值。例如，您可以使用正则表达式来验证：

- 英国或加拿大邮政编码
- 驾照编号
- 车辆识别号码

使用标准正则表达式。Amazon Lex 支持在正则表达式中使用以下字符：

- A-Z, a-z
- 0-9

Amazon Lex 在正则表达式中也支持 Unicode 字符。格式为 \u*Unicode*。使用四位数表示 Unicode 字符。例如，[\u0041-\u005A] 等同于 [A-Z]。

不支持以下正则表达式运算符：

- 无限重复符：*、+ 或 {x,}，无上限。
- 通配符（.）

正则表达式的最大长度为 300 个字符。存储在 AMAZON 中的字符串的最大长度。 AlphaNumeric 使用正则表达式的插槽类型为 30 个字符。

以下是一些示例正则表达式。

- 字母数字字符串，例如 **APQ123** 或 **APQ1**：[A-Z]{3}[0-9]{1,3} 或更受约束的 [A-DP-T]{3}[1-5]{1,3}
- 美国邮政服务优先邮件国际格式，例如 **CP123456789US**：CP[0-9]{9}US
- 银行汇款路径号码，例如 **123456789**：[0-9]{9}

要为槽类型设置正则表达式，请使用控制台或 PutSlotType 操作。保存槽类型时验证正则表达式。如果表达式无效，Amazon Lex 将返回错误消息。

在插槽类型中使用正则表达式时，Amazon Lex 会根据正则表达式检查该类型的插槽的输入。如果输入与表达式匹配，则接受槽的值。如果输入不匹配，Amazon Lex 提示用户重复输入。

AMAZON.City

提供本地和世界城市列表。插槽类型可以识别城市名称的常见变体。Amazon Lex 不会从变体转换为官方名称。

示例：

- New York
- 雷克雅未克
- 东京
- 凡尔赛

AMAZON.Country

世界各个国家/地区的名字。示例：

- 澳大利亚
- 德国
- 日本
- 美国
- 乌拉圭

AMAZON.DATE

将表示日期的词语转换为日期格式。

日期以 ISO-8601 日期格式提供给您的意图。您的意图在插槽中收到的日期可能会有所不同，具体取决于用户说出的特定短语。

- 映射到特定日期的言语（例如“今天”、“现在”或“11 月 25 日”）会转换为完整的日期：2020-11-25。默认为当前日期当天或之后的日期。

- 映射到特定星期的言语（例如“本周”或“下周”）会转换为一周的第一天的日期。在 ISO-8601 格式中，一周从星期一开始，到星期日结束。例如，如果今天是 2020 年 11 月 25 日，则“下周”会转换为 2020-11-30。
- 映射到月但不是特定日期（例如“下个月”）的言语会转换为该月的最后一天。例如，如果今天是 2020 年 11 月 25 日，则“下个月”会转换为 2020-12-31。
- 映射到年但不是特定月份或日期（例如“下一年”）的言语会转换为下一年的最后一天。例如，如果今天是 2020 年 11 月 25 日，则“下一年”会转换为 2021-12-31。

AMAZON.DURATION

将表示持续时间的词语转换为数字持续时间。

持续时间被解析为基于 ISO-8601 持续时间格式的格式 PnYnMnWnDTnHnMnS。P 表示这是持续时间，n 是数值，n 后面的大写字母是特定的日期或时间元素。例如，P3D 表示 3 天。T 用于表示其余值代表时间元素而不是日期元素。

示例：

- “十分钟”：PT10M
- “五个小时”：PT5H
- “三天”：P3D
- “四十五秒”：PT45S
- “八周”：P8W
- “七年”：P7Y
- “五小时十分钟”：PT5H10M
- “两年三小时十分钟”：P2YT3H10M

亚马逊。 EmailAddress

识别表示以 username@domain 形式提供的电子邮件地址的单词。地址中的用户名可以包括下列特殊字符：下划线 (_)、连字符 (-)、句点 (.) 和加号 (+)。

亚马逊。 FirstName

常用的名字。这种插槽类型可以识别正式名字和非正式昵称。发送给您的意图的名字是用户发送的值。Amazon Lex 不会将昵称转换为正式名字。

对于听起来相似但拼写不同的名字，Amazon Lex 会向您的意图发送一个通用格式。

在英语（美国）(en-US) 区域设置中，使用插槽名称 AMAZON.US_First_Name。

示例：

- Emily
- John
- Sophie

亚马逊。 LastName

常用的姓氏。对于听起来相似但拼写不同的姓氏，Amazon Lex 会向您的意图发送一份通用格式。

在英语（美国）(en-US) 区域设置中，使用插槽名称 AMAZON.US_Last_Name。

示例：

- 布罗斯基
- 达舍
- 埃弗斯
- 帕雷斯
- 威尔特

AMAZON.NUMBER

将表示数值的词语或数字转换为数字，包括十进制数字。下表介绍 AMAZON.NUMBER 槽类型如何捕获数字词汇。

输入	响应
一百二十三点四五	123.45
一二三点四五	123.45
零点四二	0.42
点四二	0.42

输入	响应
232.998	232.998
50	50

AMAZON.Percentage

将表示百分比的词汇和符号转换为一个带百分比符号 (%) 的数字值。

如果用户输入不带百分号或“百分之”这个词的数字，则槽值将设置为该数字。下表介绍 `AMAZON.Percentage` 槽类型如何捕获百分比。

输入	响应
百分之 50	50%
百分之 0.4	0.4%
23.5%	23.5%
百分之二十五	25%

亚马逊。 PhoneNumber

如下将表示电话号码的数字或词转换为不含标点符号的字符串格式。

类型	描述	输入	结果
前面带加号 (+) 的国际号码	前面带加号 (+) 的 11 位数字。	+61 7 4445 1061 +1 (509) 555-1212	+61744431061 +15095551212
前面不带加号 (+) 的国际号码	前面不带加号 (+) 的 11 位数字	1 (509) 555-1212 61 7 4445 1061	15095551212 61744451061
国家/地区代码	不带国际代码的 10 位数字	(03) 5115 4444	0351154444

类型	描述	输入	结果
		(509) 555-1212	5095551212
本地号码	不带国际代码或区号的 7 位数电话号码	555-1212	5551212

亚马逊。 SpeedUnit

将表示速度单位的词转换为相应的缩写。例如，“每小时英里数”将转换为 mph。

此插槽类型仅在英语（美国）(en-US) 区域设置中可用。

以下示例说明 AMAZON.SpeedUnit 槽类型如何捕获速度单位。

速度单位	缩写
miles per hour、mph、MPH、m/h	mph
kilometers per hour、km per hour、kmph、KMPH、km/h	kmph
meters per second、mps、MPS、m/s	mps
nautical miles per hour、knots、knot	knot

AMAZON.State

国家/地区内部的地理和政治区域的名称。

示例：

- 巴伐利亚
- 福岛县
- 太平洋西北地区
- 昆士兰
- 威尔士

亚马逊。 StreetName

典型街道地址内的街道名称。这只包括街道名称，不包括门牌号码。

此插槽类型在英语（美国）(en-US) 区域设置中不可用。

示例：

- 堪培拉大道
- 前街
- 市场路

AMAZON.TIME

将表示时间的词转换为时间值。包括针对不确定的时间的决议。当用户输入不明确的时间时，Amazon Lex 使用 Lambda 事件的 `slotDetails` 属性将不确定的时间的解析传递给 Lambda 函数。例如，如果自动程序提示用户输入交付时间，用户可能以“10 点钟”作为响应。这样的时间是不明确的。它可能指上午 10:00 或晚上 10:00。这种情况下，`slots` 图中值为 `null`，而 `slotDetails` 实体包含对该时间的两种可能的解析。Amazon Lex 将以下内容输入 Lambda 函数：

```
"slots": {
   "deliveryTime": null
},
"slotDetails": {
   "deliveryTime": {
      "resolutions": [
         {
            "value": "10:00"
         },
         {
            "value": "22:00"
         }
      ]
   }
}
```

当用户以不明确的时间作为响应时，Amazon Lex 将该时间发送到 Lambda 事件的 `slots` 属性中的 Lambda 函数，且 `slotDetails` 属性为空。例如，如果用户在收到输入交付时间的提示时，以“晚上 10:00”作为响应，则 Amazon Lex 将以下内容输入 Lambda 函数：

```
"slots": {
   "deliveryTime": "22:00"
}
```

有关 Amazon Lex 发送到 Lambda 函数的数据的更多信息，请参阅输入事件格式。

亚马逊。 WeightUnit

将表示重量单位的词转换为相应的缩写。例如，“kilogram”将转换为 kg。

此插槽类型仅在英语（美国）(en-US) 区域设置中可用。

以下示例说明 AMAZON.WeightUnit 槽类型如何捕获重量单位：

重量单位	缩写
kilograms、kilos、kgs、KGS	kg
grams、gms、gm、GMS、g	g
milligrams、mg、mgs	mg
pounds、lbs、LBS	lbs
ounces、oz、OZ	oz
tonne、ton、t	t
kiloton、kt	kt

自定义槽位类型

对于每个目的，您都可以指定参数来指示目的要完成用户请求所需的信息。这些参数，或者说槽，都有一个类型。一个插槽类型就是一个值列表，Amazon Lex 用它来训练机器学习模型识别插槽的值。例如，您可以定义一个名为“Genres.”的槽类型，该槽类型中的每个值都是一个体裁的名称：“comedy”、“adventure”、“documentary”等等。您可以为槽类型值定义同义词。例如，可以为“comedy”值定义同义词“funny”和“humorous”。

您可以通过配置槽类型来限制对槽值的解析。槽值将用作枚举，仅当用户输入的值与一个槽值或同义词相同时才会解析为槽值。同义词会解析为相应的槽值。例如，如果用户输入“funny”，会解析为槽值“comedy”。

或者，您可以通过配置槽类型来扩展值。槽值将用作训练数据，当用户提供的值与槽值和同义词相似时，该槽将解析为用户提供的值。这是默认行为。

Amazon Lex 为插槽维护一个可能解析值的列表。列表中的每个条目都提供一个解析值，Amazon Lex 将其识别为插槽的更多可能值。解析值是与槽值的尽可能匹配。该列表最多包含五个值。

当用户输入的值是同义词时，解析值列表中的第一个条目就是槽类型值。例如，如果用户输入“funny”，则 `slots` 字段包含“funny”，而 `slotDetails` 字段中的第一个条目是“comedy”。您可以在使用 PutSlotType 操作创建或更新槽类型时配置 `valueSelectionStrategy`，以便使用解析列表中的第一个值填充槽值。

如果您使用 Lambda 函数，该函数的输入事件中会包含一个名为 `slotDetails` 的解析列表。以下示例显示了 Lambda 函数的输入的插槽和插槽详细信息部分：

```
"slots": {
   "MovieGenre": "funny";
},
"slotDetails": {
   "Movie": {
      "resolutions": [
         "value": "comedy"
      ]
   }
}
```

对于每个槽类型，最多可定义 10000 个值和同义词。每个自动程序的槽类型值和同义词的总数最多为 50000。例如，您可以拥有 5 种插槽类型，每种类型包含 5,000 个值和 5,000 个同义词，或者您可以拥有 10 种插槽类型，每种类型包含 2,500 个值和 2,500 个同义词。如果超过这些限制，在调用 PutBot 操作时，您将获得一个 `LimitExceededException`。

槽混淆处理

Amazon Lex 能够混淆处理（即隐藏）插槽的内容，使其内容不可见。为了保护作为插槽值捕获的敏感数据，可以启用插槽混淆处理，为对话记录掩蔽这些值。

当您选择混淆处理插槽值时，Amazon Lex 在对话日志中将插槽的值替换为插槽的名称。对于名为 full_name 的槽，该槽的值将被模糊处理，如下所示：

```
Before obfuscation:
    My name is John Stiles
After obfuscation:
    My name is {full_name}
```

如果言语中包含括号字符 ({})，Amazon Lex 将用两个反斜杠 (\\) 转义括号字符。例如，文本 {John Stiles} 的模糊处理如下所示：

```
Before obfuscation:
    My name is {John Stiles}
After obfuscation:
    My name is \\{{full_name}\\}
```

对话日志中的槽值会被模糊处理。槽位值在 PostContent 和 PostText 操作的响应中仍然可用，并且槽位值可用于验证和实现 Lambda 函数。如果您在提示或响应中使用槽值，则对话日志中不会对这些槽值进行模糊处理。

在第一轮对话中，如果 Amazon Lex 能够识别出言语中的插槽和插槽值，则会对插槽值进行混淆处理。如果没有识别出插槽值，Amazon Lex 不对言语进行混淆处理。

在第二轮和接下来的轮次中，Amazon Lex 知晓要引发的插槽以及是否需要对插槽值进行混淆处理。如果 Amazon Lex 识别到该插槽值，则会对该值进行混淆处理。如果 Amazon Lex 未识别出值，则对整个言语进行混淆处理。无法理解的话语中的任何槽值都不会被模糊处理。

Amazon Lex 也不会对您存储在请求或会话属性中的插槽值进行混淆处理。如果您将应模糊处理的槽值作为属性存储，则必须加密该值，或者以其他方式对该值进行模糊处理。

Amazon Lex 不会对音频中的插槽值进行混淆处理。它会对音频转录中的槽值进行模糊处理。

您不需要在自动程序中对所有槽进行混淆处理。您可以使用控制台或使用 Amazon Lex API 选择需要对哪些插槽进行混淆处理。在控制台中，在槽的设置中选择 Slot obfuscation (槽模糊处理)。如果您使用的是 API，请在调用 PutIntent 操作时，将槽的 obfuscationSetting 字段设置为 DEFAULT_OBFUSCATION。

情绪分析

您可以使用情绪分析来确定用户语句中表达的情绪。通过情绪信息，您可以管理对话流或执行呼叫后分析。例如，如果用户情绪是消极的，您可以创建一个流，将对话交给人工代理。

Amazon Lex 与 Amazon Comprehend 集成以检测用户情绪。来自 Amazon Comprehend 的响应可表示文本的整体情绪是积极、中性、消极还是混杂。响应包含用户语句最可能传达的情绪以及每个情绪类别的分数。分数表示正确检测到情绪的可能性。

您可以使用控制台或使用 Amazon Lex API 为机器人启用情绪分析。在 Amazon Lex 控制台上，选择机器人的设置选项卡，然后将情绪分析选项设置为是。如果您使用的是 API，请将 `detectSentiment` 字段设置为 `true`，然后调用 PutBot 操作。

启用情绪分析后，来自 PostContent 和 PostText 操作的响应将在自动程序响应中返回 `sentimentResponse` 字段和其他元数据。`sentimentResponse` 字段具有 `SentimentLabel` 和 `SentimentScore` 两个字段，包含情绪分析的结果。如果您使用的是 Lambda 函数，则 `sentimentResponse` 字段将包含在发送到函数的事件数据中。

以下是 `sentimentResponse` 字段作为 `PostText` 或 `PostContent` 响应的一部分返回的示例。`SentimentScore` 字段是包含响应分数的字符串。

```
{
    "SentimentScore":
        "{
        Mixed: 0.030585512690246105,
        Positive: 0.94992071056365967,
        Neutral: 0.0141543131828308,
        Negative: 0.00893945890665054
        }",
    "SentimentLabel": "POSITIVE"
}
```

Amazon Lex 代表您调用 Amazon Comprehend，以确定机器人处理的每个语句中的情绪。启用情绪分析即表示您同意 Amazon Comprehend 的服务条款和协议。有关 Amazon Comprehend 定价的更多信息，请参阅 Amazon Comprehend 定价。

有关 Amazon Comprehend 情绪分析工作原理的更多信息，请参阅《Amazon Comprehend 开发人员指南》中的确定情绪。

为您的 Amazon Lex 资源添加标签

为了帮助您管理 Amazon Lex 机器人、机器人别名和机器人通道，您可以将元数据作为标签分配给每个资源。标签是您分配给 AWS 资源的标签。每个标签均包含一个键和一个值。

标签可让您按各种标准（例如用途、所有者或应用程序）对 AWS 资源进行分类。标签帮助您：

- 识别和整理您的 AWS 资源。许多 AWS 资源都支持标记，因此您可以为不同服务中的资源分配相同的标签，以表明这些资源是相关的。例如，您可以使用相同标签标记机器人及其使用的 Lambda 函数。
- 分配成本。您可以在 AWS 账单与成本管理 控制面板上激活标签。 AWS 使用标签对您的成本进行分类，并向您提供每月成本分配报告。对于 Amazon Lex，您可以使用别名特定的标签（$LATEST 别名除外）为每个别名分配成本。您可以通过对 Amazon Lex 机器人使用标签来为 $LATEST 别名分配成本。有关更多信息，请参阅《AWS 账单与成本管理 用户指南》中的使用成本分配标签。
- 控制对资源的访问。您可以对 Amazon Lex 使用标签来创建策略以控制对 Amazon Lex 资源的访问。这些策略可以附加到 IAM 角色或用户，以启用基于标签的访问控制。有关更多信息，请参阅 ABAC 与 Amazon Lex。要查看基于身份的策略（用于根据资源上的标签来限制对该资源的访问）的示例，请参阅 使用标签访问资源。

您可以使用 AWS Management Console、或 Amazon Lex API 来处理标签。 AWS Command Line Interface

为资源添加标签

如果您使用的是 Amazon Lex 控制台，则可以在创建资源时标记资源，也可以稍后添加标记。您还可以使用控制台来更新或删除现有标签。

如果您使用的是 AWS CLI 或 Amazon Lex API，则可以使用以下操作来管理资源的标签：

- ListTagsForResource — 查看与资源关联的标签。
- PutBot 和 PutBotAlias — 在创建机器人或机器人别名时应用标签。
- TagResource — 添加和修改现有资源上的标签。
- UntagResource — 从资源中删除标签。

Amazon Lex 中支持贴标签的资源如下：

- 自动程序 - 使用如下所示的 Amazon 资源名称 (ARN)：
  - `arn:${partition}:lex:${region}:${account}:bot:${bot-name}`
- 自动程序别名 - 使用如下所示的 ARN：
  - `arn:${partition}:lex:${region}:${account}:bot:${bot-name}:${bot-alias}`
- 自动程序通道 - 使用如下所示的 ARN：
  - `arn:${partition}:lex:${region}:${account}:bot-channel:${bot-name}:${bot-alias}:${channel-name}`

标签限制

以下基本限制适用于 Amazon Lex 资源上的标签：

- 最大标签数量为 50
- 最大键长度为 128 个字符
- 最大值长度为 256 个字符
- 键和值的有效字符包括 a–z、A–Z、0–9、空格和以下字符：_ . : / = + - 和 @
- 键和值区分大小写。
- 不要将 `aws:` 用作键的前缀；这是留给 AWS 使用的。

标记资源（控制台）

您可以使用控制台管理自动程序、自动程序别名或自动程序通道资源上的标签。您可以在创建资源时添加标签，也可以从现有资源中添加、修改或删除标签。

在创建机器人时添加标签

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 选择 Create (创建) 以创建新的自动程序。
3. 在 Create your bot (创建您的自动程序) 页面的底部，选择 Tags (标签)。
4. 选择 Add tag (添加标签) 并向自动程序添加一个或多个标签。最多可以添加 50 个标签。

在创建机器人别名时添加标签

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 选择想要为其添加机器人别名的机器人。
3. 选择设置。
4. 添加别名，选择自动程序版本，然后选择 Add tags (添加标签)。
5. 选择 Add tag (添加标签) 并向自动程序别名添加一个或多个标签。最多可以添加 50 个标签。

在创建自动程序通道时添加标签

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 选择要向其添加自动程序通道的自动程序。
3. 选择 Channels (通道)，然后选择要添加的通道。
4. 添加自动程序通道的详细信息，然后选择 Tags (标签)。
5. 选择 Add tag (添加标签) 并向自动程序通道添加一个或多个标签。最多可以添加 50 个标签。

在导入自动程序时添加标签

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 选择 Actions (操作)，然后选择 Import (导入)。
3. 选择用于导入自动程序的 zip 文件。
4. 选择 Tags (标签)，然后选择 Add tag (添加标签) 以向自动程序添加一个或多个标签。最多可以添加 50 个标签。

添加、删除或修改现有机器人上的标签

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 从左侧菜单中，选择 Bots (自动程序)，然后选择要修改的自动程序。
3. 选择 Settings (设置)，然后从左侧菜单中选择 General (常规)。
4. 选择 Tags (标签)，然后添加、修改或删除自动程序的标签。

添加、删除或修改自动程序别名上的标签

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 从左侧菜单中，选择 Bots (自动程序)，然后选择要修改的自动程序。
3. 选择 Settings (设置)，然后从左侧菜单中选择 Aliases (别名)。
4. 选择要修改的别名的 Manage tags (管理标签)，然后添加、修改或删除自动程序别名的标签。

添加、删除或修改现有自动程序通道上的标签

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 从左侧菜单中，选择 Bots (自动程序)，然后选择要修改的自动程序。
3. 选择 Channels。
4. 选择 Tags (标签) ，然后添加、修改或删除自动程序通道的标签。

标记资源 (AWS CLI)

您可以使用 AWS CLI 来管理机器人、机器人别名或机器人频道资源上的标签。您可以在创建自动程序或自动程序别名时添加标签，也可以从自动程序、自动程序别名或自动程序通道中添加、修改或删除标签。

所有示例都是针对 Linux 和 macOS 进行格式设置的。要在 Windows 中使用该命令，请将 Linux 继续符 (\) 替换为脱字号 (^)。

在创建机器人时添加标签

- 以下缩写`put-bot` AWS CLI 命令显示了在创建机器人时添加标签时必须使用的参数。要实际创建自动程序，您必须提供其他参数。有关更多信息，请参阅 步骤 4：入门 (AWS CLI)。

```
aws lex-models put-bot \
    --tags '[{"key": "key1", "value": "value1"}, \
             {"key": "key2", "value": "value2"}]'
```

在创建机器人别名时添加标签

- 以下缩写put-bot-alias AWS CLI 命令显示了在创建机器人别名时添加标签时必须使用的参数。要实际创建自动程序别名，您必须提供其他参数。有关更多信息，请参阅 练习 5：创建别名 (AWS CLI)。

```
aws lex-models put-bot \
    --tags '[{"key": "key1", "value": "value1"}, \
             {"key": "key2", "value": "value2"}]"
```

列出资源上的标签

- 使用list-tags-for-resource AWS CLI 命令显示与机器人、机器人别名、机器人频道关联的资源。

```
aws lex-models list-tags-for-resource \
    --resource-arn bot, bot alias, or bot channel ARN
```

添加或修改资源上的标签

- 使用tag-resource AWS CLI 命令添加或修改机器人、机器人别名或机器人频道。

```
aws lex-models tag-resource \
    --resource-arn bot, bot alias, or bot channel ARN \
    --tags '[{"key": "key1", "value": "value1"}, \
             {"key": "key2", "value": "value2"}]'
```

从资源中删除标签

- 使用untag-resource AWS CLI 命令从机器人、机器人别名或机器人频道中移除标签。

```
aws lex-models untag-resource \
    --resource-arn bot, bot alias, or bot channel ARN \
    --tag-keys '["key1", "key2"]'
```

Amazon Lex 入门

Amazon Lex 提供 API 操作，您可以轻松地将这些操作与现有应用程序集成。有关支持的操作的列表，请参阅 API 参考。您可以使用以下任意选项：

- AWS 软件开发工具包 — 使用时，系统会使用 SDKs 您提供的凭证自动对您向 Amazon Lex 发出的请求进行签名和身份验证。这是用于构建应用程序的推荐选择。
- AWS CLI — 您无需编写任何代码 AWS CLI 即可使用任何 Amazon Lex 功能。
- AWS 管理控制台 — 该控制台是开始测试和使用 Amazon Lex 的最简单方法。

如果您是首次使用 Amazon Lex，建议先阅读 Amazon Lex：工作原理。

主题

步骤 1：设置 AWS 账户并创建管理员用户

首次使用 Amazon Lex 前，请完成以下任务：

1. 报名参加 AWS
2. 创建用户

报名参加 AWS

如果您已经有一个 AWS 帐户，请跳过此任务。

当您注册亚马逊 Web Services (AWS) 时，您的 AWS 账户将自动注册所有服务 AWS，包括 Amazon Lex。您只需为使用的服务付费。

使用 Amazon Lex 时，您仅需为实际使用的资源付费。如果您是 AWS 新客户，还可以免费试用 Amazon Lex。有关更多信息，请参阅 AWS 免费使用套餐。

如果您已经有一个 AWS 帐户，请跳到下一个任务。如果您还没有 AWS 账户，请按照以下步骤创建。

创建 AWS 账户

1. 打开https://portal.aws.amazon.com/billing/注册。
2. 按照屏幕上的说明操作。

   注册过程的一部分涉及接听电话或短信，并在电话键盘上输入验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

记下您的 AWS 账户 ID，因为下个任务需要用到它。

创建用户

中的 AWS服务（例如 Amazon Lex）要求您在访问时提供凭证，以便服务可以确定您是否有权访问该服务所拥有的资源。控制台要求您的密码。但是，我们不建议您 AWS 使用 AWS 账户凭证进行访问。而是建议您：

- 使用 AWS Identity and Access Management (IAM) 创建用户
- 将用户添加到具有管理权限的 IAM 组中。
- 向您创建的 用户授予管理权限。

然后，您可以使用特殊的 URL 和用户的凭据 AWS 进行访问。

本指南中的入门练习假定您拥有具有管理权限的用户 (adminuser)。请按照以下过程在您的账户中创建 adminuser。

创建管理员用户和登录控制台

1. 在您的 AWS 账户中创建名为 adminuser 的管理员用户。有关说明，请参阅《IAM 用户指南》中的创建您的第一个用户和管理员组。
2. 作为用户，您可以使用特殊的 URL 登录。 AWS Management Console 有关更多信息，请参阅《IAM 用户指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/getting-started_how-users-sign-in.html中的用户如何登录您的账户。

有关 IAM 的更多信息，请参阅以下文档：

- AWS Identity and Access Management (IAM)
- 入门
- IAM 用户指南

下一个步骤

第 2 步：设置 AWS Command Line Interface

第 2 步：设置 AWS Command Line Interface

如果您更喜欢将 Amazon Lex 与 AWS Command Line Interface (AWS CLI) 一起使用，请下载并对其进行配置。

> **Important**
>
> 您无需使用 AWS CLI 即可执行入门练习中的步骤。但是，本指南中后面的某些练习会用到 AWS CLI。如果您更愿意使用控制台开始工作，请跳过此步骤并转至步骤 3：入门 (控制台)。稍后，当你需要时 AWS CLI，请返回此处进行设置。

要设置 AWS CLI

1. 下载并配置 AWS CLI。有关说明，请参阅《AWS Command Line Interface 用户指南》中的以下主题：
   - 开始使用进行设置 AWS Command Line Interface
   - 配置 AWS Command Line Interface
2. 将管理员用户的命名配置文件添加到 AWS CLI 配置文件的末尾。执行 AWS CLI 命令时使用此配置文件。有关命名配置文件的更多信息，请参阅 AWS Command Line Interface 用户指南中的命名配置文件。

   ```
   [profile adminuser]
   aws_access_key_id = adminuser access key ID
   aws_secret_access_key = adminuser secret access key
   region = aws-region
   ```

   有关可用 AWS 区域的列表，请参阅中的区域和终端节点Amazon Web Services 一般参考。

3. 在命令提示符处键入 Help 命令验证设置：

```
aws help
```

步骤 3：入门 (控制台)

步骤 3：入门 (控制台)

学习如何使用 Amazon Lex 的最简单方法是使用控制台。为了便于您入门，我们创建了以下练习，所有练习都使用控制台进行：

- 练习 1 — 使用蓝图创建 Amazon Lex 机器人，蓝图是预定义的机器人，提供了所有必要的机器人配置。您只需做最少的工作即可测试 end-to-end设置。

  此外，您还可以使用提供的 Lambda 函数蓝图来 AWS Lambda创建 Lambda 函数。该函数是一个代码挂钩，它使用与您的自动程序兼容的预定义代码。
- 练习 2 — 通过手动创建和配置机器人来创建自定义机器人。您还可以创建 Lambda 函数作为代码挂钩。我们提供了示例代码。
- 练习 3 — 发布机器人，然后创建它的新版本。作为本练习的一部分，您将创建一个指向自动程序版本的别名。

主题

练习 1：使用蓝图创建 Amazon Lex 机器人（控制台）

在本练习中，您将执行以下操作：

- 在 Amazon Lex 控制台中创建您的首个 Amazon Lex 机器人并进行测试。

  在本练习中，您将使用OrderFlowers蓝图。有关蓝图的信息，请参阅 Amazon Lex 和 AWS Lambda 蓝图。

- 创建 AWS Lambda 函数并在 Lambda 控制台中对其进行测试。在处理请求时，机器人调用此 Lambda 函数。在本练习中，您将使用 AWS Lambda 控制台中提供的 Lambda 蓝图 (lex-order-flowers-python) 来创建您的 Lambda 函数。此蓝图代码说明如何使用该 Lambda 函数执行初始化和验证以及履行 `OrderFlowers` 意图。

- 更新机器人以将 Lambda 函数添加为代码挂钩，从而履行此意图。测试 end-to-end体验。

下面几节介绍蓝图的作用。

Amazon Lex 机器人：蓝图概览

您可以使用OrderFlowers蓝图创建 Amazon Lex 机器人。有关机器人结构的更多信息，请参阅。Amazon Lex：工作原理此自动程序将按如下方式进行预配置：

- 意图 — OrderFlowers
- 槽类型：一个称为 `FlowerTypes` 的自定义槽类型，具有枚举值：`roses`、`lilies` 和 `tulips`。
- 槽：在机器人实现此意图之前，意图需要以下信息（即槽）。
  - `PickupTime`（AMAZON.TIME 内置类型）
  - `FlowerType`（FlowerTypes 自定义类型）
  - `PickupDate`（AMAZON.DATE 内置类型）
- 表达：以下示例表达表示用户的意图：
  - “我想要取花。”
  - “我想要订些花。”
- 提示：在机器人确定此意图后，它会使用以下提示来填充槽：
  - 用于 `FlowerType` 槽的提示：“您想要订哪种类型的花？”
  - 提示进入时`PickupDate`段 — “你想在哪一天拿起 {FlowerType}？”
  - 提示输入插`PickupTime`槽 — “你想在什么时候拿起 {FlowerType}？”
  - 确认声明 — “好的，您的 {FlowerType} 将在 {} 上准备好在 {PickupTime} 上取货。PickupDate这样可以吗？”

AWS Lambda 功能：蓝图摘要

此练习中的 Lambda 函数执行初始化和验证以及履行任务。因此，创建 Lambda 函数后，通过将该 Lambda 函数指定为代码挂钩，可以处理初始化和验证以及履行任务，从而更新意图配置。

- 作为初始化和验证代码挂钩，Lambda 函数执行基本验证。例如，如果用户提供正常营业时间范围之外的某个提取时间，则 Lambda 函数会指示 Amazon Lex 重新提示用户指定时间。
- 作为履行代码挂钩的一部分，Lambda 函数会返回一条摘要消息，指示鲜花订单已下单（即，意图已履行）。

下一个步骤

步骤 1：创建 Amazon Lex 机器人（控制台）

步骤 1：创建 Amazon Lex 机器人（控制台）

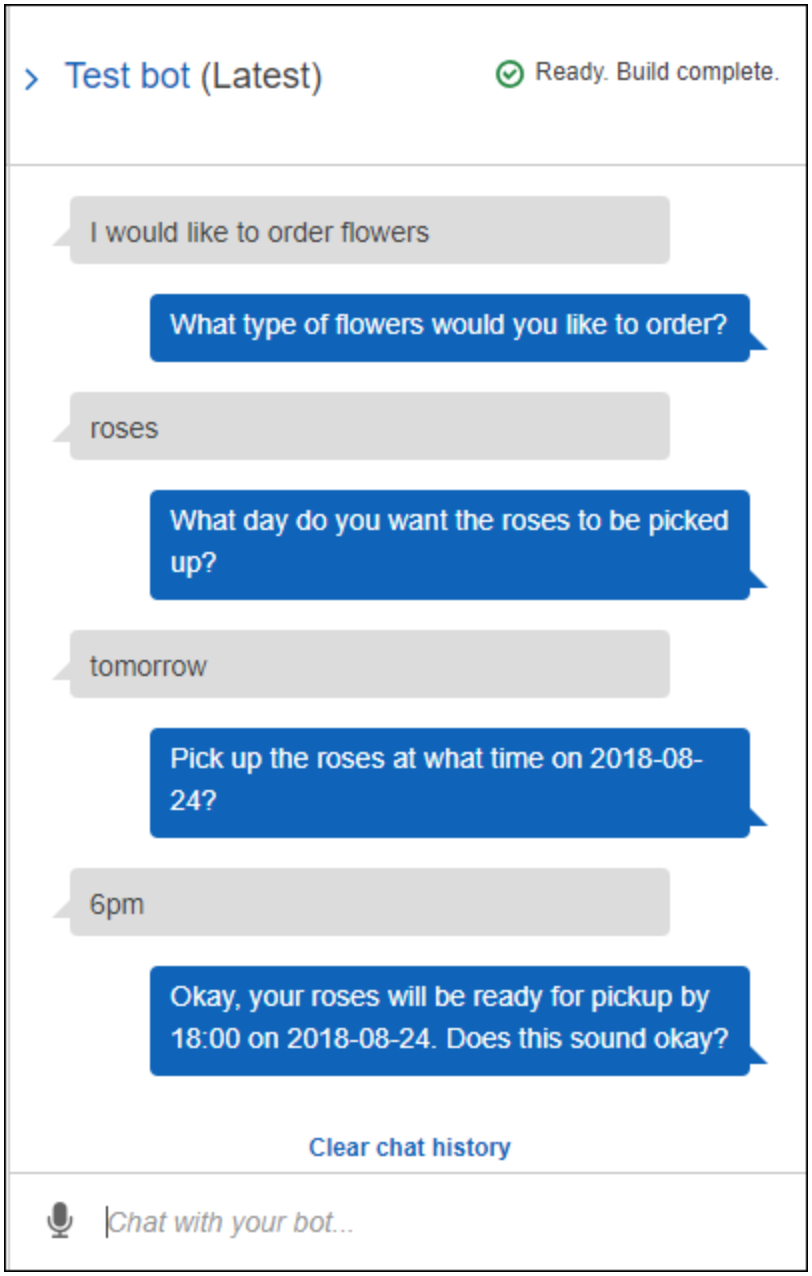
在本练习中，创建一个用于订购鲜花的机器人，名为 OrderFlowersBot。

创建 Amazon Lex 机器人（控制台）

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 如果这是您的第一个自动程序，请选择 Get Started (开始)；否则，在 Bots (自动程序) 页面上，选择 Create (创建)。
3. 在 Create your Lex bot 页面上，提供以下信息，然后选择 Create。
   - 选择 OrderFlowers 蓝图。
   - 保留默认的机器人名称 (OrderFlowers)。
   - 对于 COPPA，选择 **No**。
   - 对于用户言语存储，选择相应的响应。
4. 选择创建。控制台可向 Amazon Lex 提出必要的请求，以保存配置。然后，控制台显示自动程序编辑器窗口。
5. 等待自动程序的构建确认。
6. 测试自动程序。

> **Note**
>
> 通过在测试窗口中键入文本可以测试自动程序，或者对于兼容的浏览器，可通过选择测试窗口中的麦克风按钮并说话这种方法。

使用以下示例文本与自动程序进行对话来订花：



自动程序通过此输入推断槽数据的 OrderFlowers 目的和提示。在您提供所有必要的槽数据后，自动程序将所有这类信息返回到客户端应用程序 (这里指控制台) 即可实现目的 (OrderFlowers)。控制台在测试窗口中显示这类信息。

具体来说：

- 在语句“What day do you want the roses to be picked up?”中出现了词语“roses”，这是因为已使用替换项 {FlowerType} 来配置 pickupDate 槽的提示。在控制台中对此进行验证。
- “Okay, your roses will be ready...”语句是您配置的确认提示。
- 最后一个语句 (“FlowerType:roses...”) 是返回到客户端 (这里指测试窗口) 的槽数据。在下一个练习中，将使用 Lambda 函数履行此意图，在这种情况下，您将收到一条指示订单已履行的消息。

下一个步骤

步骤 2 (可选)：查看信息流的详细信息 (控制台)

步骤 2 (可选)：查看信息流的详细信息 (控制台)

本节介绍示例对话中每个用户输入在客户端与 Amazon Lex 之间的信息流。

该示例使用控制台测试窗口与机器人进行对话。

打开 Amazon Lex 测试窗口

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 选择要测试的机器人。
3. 在控制台的右侧，选择测试聊天机器人。

要查看语音或所键入内容的信息流，请选择相应的主题。

主题

- 步骤 2a (可选)：查看语音信息流的详细信息 (控制台)
- 步骤 2b (可选)：查看所键入信息流的详细信息 (控制台)

步骤 2a (可选)：查看语音信息流的详细信息 (控制台)

本节介绍在客户使用语音发送请求时客户端与 Amazon Lex 之间的信息流。有关更多信息，请参阅 PostContent。

1. 用户说：我想要订些花。

   a. 客户端 (控制台) 将向 Amazon Lex 发送以下 PostContent 请求：

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/
content HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "audio/x-l16; sample-rate=16000; channel-count=1"
Accept: "audio/mpeg"


Request body
input stream
```

   请求 URI 和正文都将向 Amazon Lex 提供信息：

   - 请求 URI — 提供机器人名称 (OrderFlowers)、机器人别名 ($LATEST) 以及用户名称（用于识别用户的随机字符串）。content 指示这是 PostContent API 请求（而不是 PostText 请求）。
   - Request headers（请求标头）
     - x-amz-lex-session-attributes — base64 编码值表示“{}”。在客户端发出第一个请求时，没有会话属性。
     - Content-Type – 反映音频格式。
   - 请求正文 – 用户输入音频流 (“我想要订些花。”)。

   > **Note**
   >
   > 如果用户选择向 PostContent API 发送文本 (“我想要订些花”) 而非使用语音，则请求正文为用户输入。Content-Type 标头会相应地进行设置：
   >
   > ```
   > POST /bot/OrderFlowers/alias/$LATEST/
   > user/4o9wwdhx6nlheferh6a73fujd3118f5w/content HTTP/1.1
   > x-amz-lex-session-attributes: "e30="
   > Content-Type: "text/plain; charset=utf-8"
   > ```

```
Accept: accept

Request body
input stream
```

b. 从输入流中，Amazon Lex 可检测意图 (OrderFlowers)。随后，它会选择其中一个目的槽 (本例中为 FlowerType) 和其中一个值引出提示，然后发送具有以下标头的响应：

```
x-amz-lex-dialog-state:ElicitSlot
x-amz-lex-input-transcript:I would like to order some flowers.
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-message:What type of flowers would you like to order?
x-amz-lex-session-attributes:e30=
x-amz-lex-slot-to-elicit:FlowerType
x-amz-lex-
slots:eyJQaWNrdXBUaW1lIjpudWxsLCJGbG93ZXJUeXBlIjpudWxsLCJQaWNrdXBEYXRlIjpudWxsfQ==
```

标头值提供以下信息：

- x-amz-lex-input-transcript – 提供来自请求的音频脚本 (用户输入)
- x-amz-lex-message — 提供在响应中返回的音频 Amazon Lex 的脚本
- x-amz-lex-slots – 槽和值的 base64 编码版本：

```
{"PickupTime":null,"FlowerType":null,"PickupDate":null}
```

- x-amz-lex-session-attributes – 会话属性 ({}) 的 base64 编码版本

客户端将播放响应正文中的音频。

2. 用户说：玫瑰

a. 客户端 (控制台) 将向 Amazon Lex 发送以下 PostContent 请求：

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/
content HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "audio/x-l16; sample-rate=16000; channel-count=1"
Accept: "audio/mpeg"
```

```
Request body
input stream ("roses")
```

请求正文是用户输入音频流 (玫瑰)。sessionAttributes 将保留为空。

b. Amazon Lex 将解释当前意图的上下文中的输入流（它会记住它已经向此用户询问了关于 FlowerType 插槽的信息）。Amazon Lex 将首先更新当前意图的插槽值。然后，它选择另一个槽 (PickupDate)，以及它的一个提示消息 (When do you want to pick up the roses?)，并返回包含以下标头的响应：

```
x-amz-lex-dialog-state:ElicitSlot
x-amz-lex-input-transcript:roses
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-message:When do you want to pick up the roses?
x-amz-lex-session-attributes:e30=
x-amz-lex-slot-to-elicit:PickupDate
x-amz-lex-
slots:eyJQaWNrdXBUaW1lIjpudWxsLCJGbG93ZXJUeXBlIjoicm9zaSdzIiwiUGlja3VwRGF0ZSI6bnVsbH0=
```

标头值提供以下信息：

- x-amz-lex-slots – 槽和值的 base64 编码版本：

  ```
  {"PickupTime":null,"FlowerType":"roses","PickupDate":null}
  ```

- x-amz-lex-session-attributes – 会话属性 ({}) 的 base64 编码版本

客户端将播放响应正文中的音频。

3. 用户说：明天

a. 客户端 (控制台) 将向 Amazon Lex 发送以下 PostContent 请求：

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/
content HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "audio/x-l16; sample-rate=16000; channel-count=1"
Accept: "audio/mpeg"


Request body
```

```
input stream ("tomorrow")
```

请求正文是用户输入音频流 (“明天”)。sessionAttributes 将保留为空。

b. Amazon Lex 将解释当前意图的上下文中的输入流（它会记住它已经向此用户询问了关于 PickupDate 插槽的信息）。Amazon Lex 将更新当前意图的插槽 (PickupDate) 值。然后，它将选择另一个槽来引出 (PickupTime) 的值，并选择其中一个值引出提示 (您想在 2017 年 3 月 18 日的什么时间取玫瑰？)，然后返回带有以下标头的响应：

```
x-amz-lex-dialog-state:ElicitSlot
x-amz-lex-input-transcript:tomorrow
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-message:When do you want to pick up the roses on 2017-03-18?
x-amz-lex-session-attributes:e30=
x-amz-lex-slot-to-elicit:PickupTime
x-amz-lex-
slots:eyJQaWNrdXBUaW1lIjpudWxsLCJGbG93ZXJUeXBlIjoicm9zaSdzIiwiUGlja3VwRGF0ZSI6IjIwMTctM
x-amzn-RequestId:3a205b70-0b69-11e7-b447-eb69face3e6f
```

标头值提供以下信息：

- x-amz-lex-slots – 槽和值的 base64 编码版本：

  ```
  {"PickupTime":null,"FlowerType":"roses","PickupDate":"2017-03-18"}
  ```

- x-amz-lex-session-attributes – 会话属性 ({}) 的 base64 编码版本

客户端将播放响应正文中的音频。

4. 用户说：下午 6 点

a. 客户端 (控制台) 将向 Amazon Lex 发送以下 PostContent 请求：

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/
content HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "text/plain; charset=utf-8"
Accept: "audio/mpeg"


Request body
```

```
input stream ("6 pm")
```

请求正文是用户输入音频流 (“下午 6 点”)。sessionAttributes 将保留为空。

b. Amazon Lex 将解释当前意图的上下文中的输入流（它会记住它已经向此用户询问了关于 PickupTime 插槽的信息）。它将首先更新当前目的的槽值。

现在，Amazon Lex 将检测到它拥有所有插槽的信息。但是，OrderFlowers 目的配置了确认消息。因此，Amazon Lex 需要先获得用户的显式确认，然后才能继续履行意图。在订花之前，它将发送具有以下请求确认的标头的响应：

```
x-amz-lex-dialog-state:ConfirmIntent
x-amz-lex-input-transcript:six p. m.
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-message:Okay, your roses will be ready for pickup by 18:00 on
 2017-03-18.  Does this sound okay?
x-amz-lex-session-attributes:e30=
x-amz-lex-
slots:eyJQaWNrdXBUaW1lIjoiMTg6MDAiLCJGbG93ZXJUeXBlIjoicm9zaSdzIiwiUGlja3VwRGF0ZSI6IjIwM
x-amzn-RequestId:083ca360-0b6a-11e7-b447-eb69face3e6f
```

标头值提供以下信息：

- x-amz-lex-slots – 槽和值的 base64 编码版本：

  ```
  {"PickupTime":"18:00","FlowerType":"roses","PickupDate":"2017-03-18"}
  ```

- x-amz-lex-session-attributes – 会话属性 ({}) 的 base64 编码版本

客户端将播放响应正文中的音频。

5. 用户回答“是”

a. 客户端 (控制台) 将向 Amazon Lex 发送以下 PostContent 请求：

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/
content HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "audio/x-l16; sample-rate=16000; channel-count=1"
Accept: "audio/mpeg"
```

```
Request body
input stream ("Yes")
```

请求正文是用户输入音频流 (“是”)。sessionAttributes 将保留为空。

b. Amazon Lex 将解释输入流，并了解用户希望继续完成订单。OrderFlowers 目的配置有 ReturnIntent 作为完成活动。这将指示 Amazon Lex 向客户端返回所有意图数据。Amazon Lex 将返回带有以下内容的响应:

```
x-amz-lex-dialog-state:ReadyForFulfillment
x-amz-lex-input-transcript:yes
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-session-attributes:e30=
x-amz-lex-
slots:eyJQaWNrdXBUaW1lIjoiMTg6MDAiLCJGbG93ZXJUeXBlIjoicm9zaSdzIiwiUGlja3VwRGF0ZSI6IjIwM
```

x-amz-lex-dialog-state 响应标头设置为 ReadyForFulfillment。然后，客户端可以完成目的。

6. 现在，重新测试自动程序。要建立新的 (用户) 上下文，请在控制台中选择 Clear 链接。为 OrderFlowers 目的提供数据，包括一些无效数据。例如：

   - “茉莉花”作为鲜花类型 (它不是一种受支持的鲜花类型)
   - “昨天”作为您想要取花的日期

   请注意，自动程序将接受这些值，因为您没有任何代码来初始化和验证用户数据。在下一节中，您将添加 Lambda 函数来完成此操作。请注意有关 Lambda 函数的以下内容：

   - 它将在每个用户输入后验证槽数据。它将在结束时完成目的。即自动程序将处理鲜花订单，然后向用户返回一条消息，而不是只向客户端返回槽数据。有关更多信息，请参阅 使用 Lambda 函数。
   - 此外，它还将设置会话属性。有关会话属性的更多信息，请参阅 PostText。

     完成“入门”部分后，您可以做其他练习 (其他示例：创建 Amazon Lex 机器人)。预订旅程 使用会话属性来共享各个目的的信息，以与用户进行动态对话。

下一个步骤

步骤 3：创建 Lambda 函数（控制台）

步骤 2b (可选)：查看所键入信息流的详细信息 (控制台)

本部分介绍了客户端和 Amazon Lex (客户端在其中使用 PostText API 来发送请求) 之间的信息流。有关更多信息，请参阅 PostText。

1. 用户类型：我想要订些花

   a. 客户端 (控制台) 将向 Amazon Lex 发送以下 PostText 请求：

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "I would like to order some flowers",
    "sessionAttributes": {}
}
```

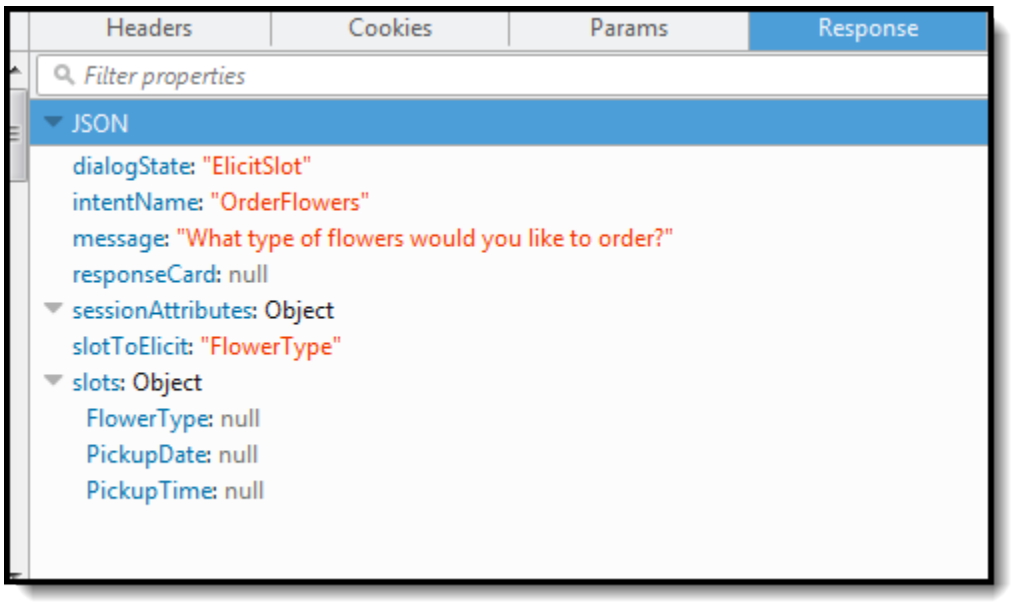
   请求 URI 和正文都将向 Amazon Lex 提供信息：

   - 请求 URI — 提供机器人名称 (OrderFlowers)、机器人别名 ($LATEST) 和用户名称（用于标识用户的随机字符串）。后面的 text 表示它是一个 PostText API 请求 (而不是 PostContent)。

   - 请求正文 – 包括用户输入 (inputText) 和空 sessionAttributes。在客户端发出第一个请求时，没有会话属性。稍后，Lambda 函数会启动这些属性。

   b. 在 inputText 中，Amazon Lex 可检测意图 (OrderFlowers)。此意图没有任何代码挂钩（即 Lambda 函数）用于初始化和验证用户输入或履行。

   Amazon Lex 将选择其中一个意图插槽 (FlowerType) 来引发值。此外，它还会选择槽的其中一个值引出提示 (目的配置的所有部分)，然后将以下响应发送回客户端。控制台向用户显示响应中的消息。



客户端将显示响应中的消息。

2. 用户类型：玫瑰

   a. 客户端 (控制台) 将向 Amazon Lex 发送以下 PostText 请求：

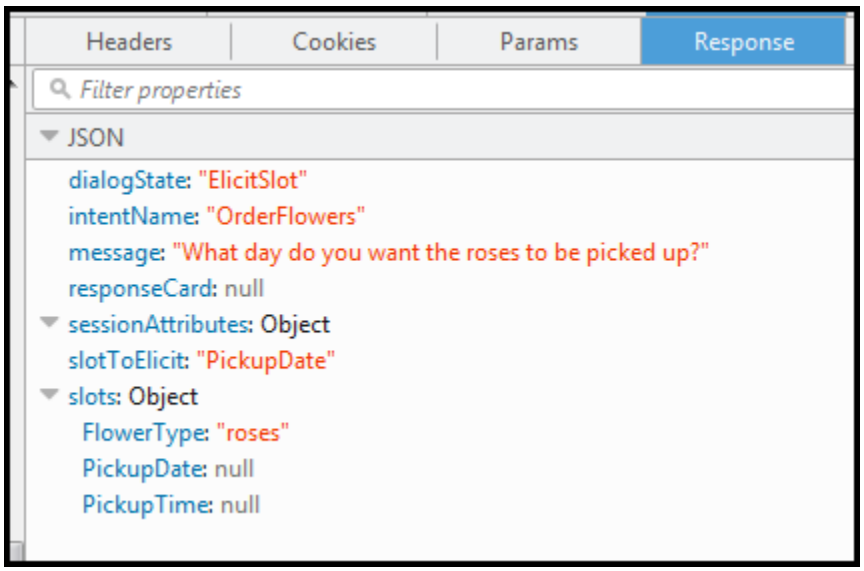
```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "roses",
    "sessionAttributes": {}
}
```

   请求正文中的 `inputText` 将提供用户输入。`sessionAttributes` 将保留为空。

   b. Amazon Lex 首先解读当前意图的上下文中的 `inputText`（服务记得它已经向特定用户询问了关于 `FlowerType` 插槽的信息）。Amazon Lex 首先更新当前意图的插槽值，然后选择另一个插槽 (`PickupDate`) 以及该插槽 的其中一个提示消息（“您想在哪天取玫瑰？”）。

   然后，Amazon Lex 将返回以下响应：



客户端将显示响应中的消息。

3. 用户类型：明天

   a. 客户端 (控制台) 将向 Amazon Lex 发送以下 PostText 请求：

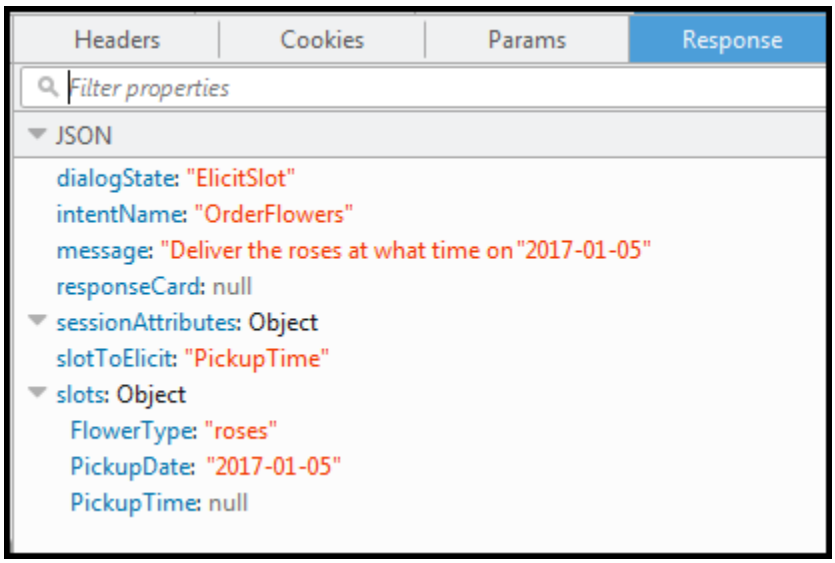
```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "tomorrow",
    "sessionAttributes": {}
}
```

   请求正文中的 `inputText` 将提供用户输入。`sessionAttributes` 将保留为空。

   b. Amazon Lex 首先解读当前意图的上下文中的 `inputText`（服务记得它已经向特定用户询问了关于 `PickupDate` 插槽的信息）。Amazon Lex 将更新当前意图的插槽 (`PickupDate`) 值。它将选择另一个槽来引出 (`PickupTime`) 的值。它向客户端返回其中一个值引发提示（在 2017 年 1 月 5 日的什么时间交付玫瑰？）。

   Amazon Lex 将返回以下响应：



客户端将显示响应中的消息。

4. 用户类型：下午 6 点

   a. 客户端 (控制台) 将向 Amazon Lex 发送以下 PostText 请求：

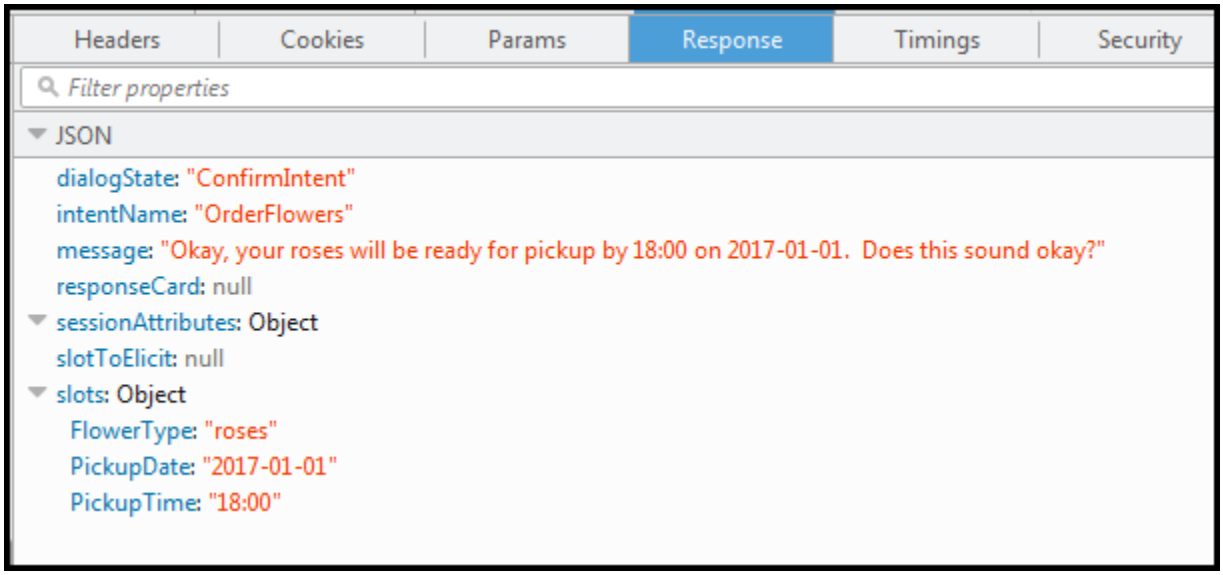
```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "6 pm",
    "sessionAttributes": {}
}
```

   请求正文中的 `inputText` 将提供用户输入。`sessionAttributes` 将保留为空。

   b. Amazon Lex 首先解读当前意图的上下文中的 `inputText`（服务记得它已经向特定用户询问了关于 `PickupTime` 插槽的信息）。Amazon Lex 将首先更新当前意图的插槽值。现在，Amazon Lex 将检测到它拥有所有插槽的信息。

   `OrderFlowers` 目的配置了确认消息。因此，Amazon Lex 需要先获得用户的显式确认，然后才能继续履行意图。在订花之前，Amazon Lex 将向请求确认的客户端发送以下消息：



客户端将显示响应中的消息。

5. 用户类型：是

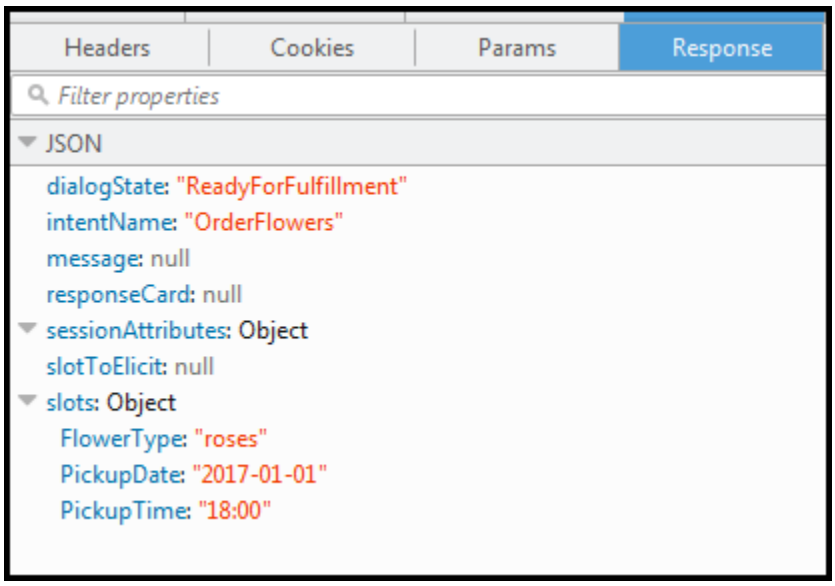
   a. 客户端 (控制台) 将向 Amazon Lex 发送以下 PostText 请求：

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "Yes",
    "sessionAttributes": {}
}
```

   请求正文中的 `inputText` 将提供用户输入。`sessionAttributes` 将保留为空。

   b. Amazon Lex 将解释确认当前意图的上下文中的 `inputText`。它将了解用户希望继续完成订单。`OrderFlowers` 意图配置有 `ReturnIntent` 作为履行活动（没有 Lambda 函数用来履行意图）。因此，Amazon Lex 向客户端返回插槽数据。



Amazon Lex 将 `dialogState` 设置为 `ReadyForFulfillment`。然后，客户端可以完成目的。

6. 现在，再次测试自动程序。为此，您必须在控制台中选择 Clear 链接以建立新的 (用户) 上下文。现在，在您提供订花目的的数据时，请尝试提供无效数据。例如：

   - “茉莉花”作为鲜花类型 (它不是一种受支持的鲜花类型)。
   - “昨天”作为您想要取花的日期。

   请注意，自动程序将接受这些值，因为您没有任何代码来初始化/验证用户数据。在下一节中，您将添加 Lambda 函数来完成此操作。请注意有关 Lambda 函数的以下内容：

   - Lambda 函数将在每个用户输入后验证插槽数据。它将在结束时完成目的。即自动程序将处理鲜花订单，然后向用户返回一条消息，而不是只向客户端返回槽数据。有关更多信息，请参阅 使用 Lambda 函数。
   - Lambda 函数还将设置会话属性。有关会话属性的更多信息，请参阅 PostText。

     完成“入门”部分后，您可以做其他练习 (其他示例：创建 Amazon Lex 机器人)。预订旅程 使用会话属性来共享各个目的的信息，以与用户进行动态对话。

下一个步骤

步骤 3：创建 Lambda 函数（控制台）

步骤 3：创建 Lambda 函数（控制台）

创建 Lambda 函数（使用lex-order-flowers-python蓝图），并使用控制台中的示例事件数据执行测试调用。 AWS Lambda

您返回 Amazon Lex 控制台，然后添加 Lambda 函数作为代码挂钩以履行 `OrderFlowers` 意图（在上一节中创建的 `OrderFlowersBot` 中）。

创建 Lambda 函数 (控制台)

1. 登录 AWS Management Console 并打开 AWS Lambda 控制台，网址为https://console.aws.amazon.com/lambda/。
2. 选择 Create function（创建函数）。
3. 在 Create function（创建函数）页面中，选择 Use a blueprint（使用蓝图）。在筛选条件文本框中键入 **`lex-`** 然后按 `Enter` 查找蓝图，选择 `lex-order-flowers-python` 蓝图。

   Node.js 和 Python 中均提供 Lambda 函数蓝图。对于此练习，请使用基于 Python 的蓝图。
4. 在 Basic information (基本信息) 页面上，执行以下操作：
   - 键入 Lambda 函数名称 (`OrderFlowersCodeHook`)。
   - 对于执行角色，选择创建具有基本 Lambda 权限的新角色。
   - 保留其他默认值。
5. 选择 Create function (创建函数)。
6. 如果您使用的是英语 (美国) (en-US) 以外的区域设置，请按照更新特定区域设置的蓝图中所述更新意图名称。
7. 测试 Lambda 函数
   a. 选择 Select a test events (选择测试事件)、Configure test event (配置测试事件)。
   b. 从 Event template (事件模板) 列表中选择 Amazon Lex Order Flowers。此示例事件与 Amazon Lex 请求/响应模型相匹配（请参阅使用 Lambda 函数）。为测试事件提供一个名称 (`LexOrderFlowersTest`)。
   c. 选择创建。
   d. 选择 Test (测试) 以测试代码挂钩。
   e. 验证 Lambda 函数已成功执行。在此情况下，此响应与 Amazon Lex 响应模型相匹配。

下一个步骤

步骤 4：将 Lambda 函数添加为代码挂钩（控制台）

步骤 4：将 Lambda 函数添加为代码挂钩（控制台）

在本节中，您将按如下方式更新使用 Lambda 函数的 OrderFlowers意图的配置：

- 首先使用 Lambda 函数作为代码挂钩以履行 `OrderFlowers` 意图。您将测试机器人并验证您收到了来自 Lambda 函数的履行消息。Amazon Lex 仅在您提供订花所需的所有插槽的数据后才会调用 Lambda 函数。
- 配置同一 Lambda 函数作为代码挂钩以执行初始化和验证。您将测试和验证 Lambda 函数是否会执行验证（当您提供插槽数据时）。

将 Lambda 函数添加为代码挂钩（控制台）

1. 在 Amazon Lex 控制台中，选择该OrderFlowers机器人。控制台显示OrderFlowers意图。确保目的版本设置为 `$LATEST`，因为这是我们可以修改的唯一版本。
2. 添加 Lambda 函数作为履行代码挂钩并对其进行测试。

   a. 在编辑器中，为履行选择 AWS Lambda 函数，然后选择上一步骤中创建的 Lambda 函数 (`OrderFlowersCodeHook`)。选择确定以授予 Amazon Lex 调用 Lambda 函数的权限。

      您正在配置此 Lambda 函数作为代码挂钩以履行意图。Amazon Lex 仅在具有用户提供的用于履行意图的所有必需的插槽数据后，才会调用此函数。

   b. 指定 Goodbye message。

   c. 选择构建。

   d. 使用之前的对话测试此自动程序。

   最后一个语句“谢谢，您订购的玫瑰.....”是配置为代码挂钩的 Lambda 函数的响应。在上一节，没有 Lambda 函数。现在，使用 Lambda 函数实际履行 `OrderFlowers` 意图。
3. 添加 Lambda 函数作为初始化和验证代码挂钩，然后进行测试。

   您使用的示例 Lambda 函数代码可同时执行用户输入验证和履行。Lambda 函数接收的输入事件有一个字段 (`invocationSource`)，代码使用该字段来确定要执行的代码的部分。有关更多信息，请参阅 Lambda 函数输入事件和响应格式。

   a. 选择 `OrderFlowers` 目的的 $LATEST 版本。这是您可更新的唯一版本。

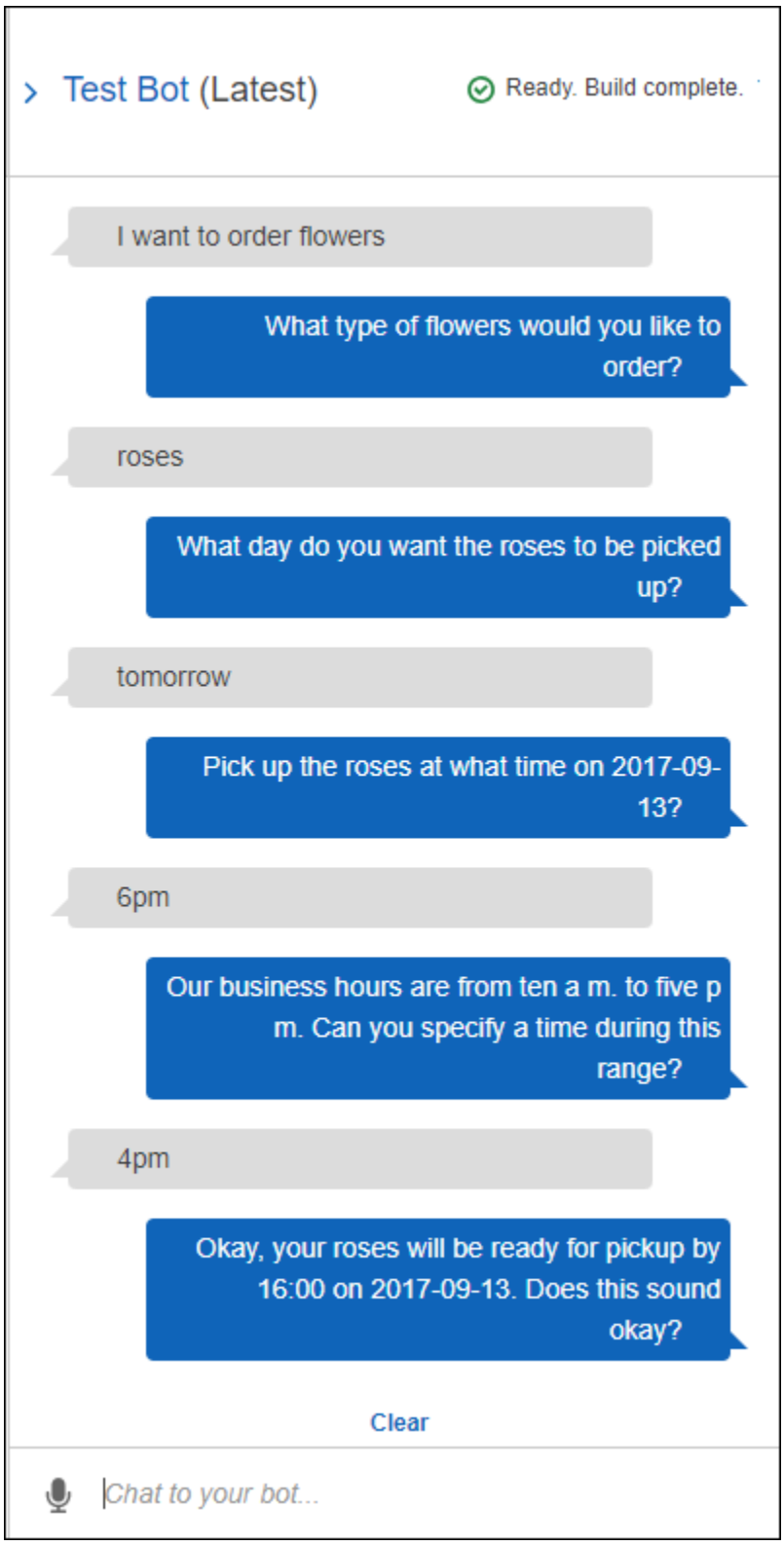
b. 在编辑器中，选择 Options 中的 Initialization and validation。

c. 同样，选择同一 Lambda 函数。

d. 选择构建。

e. 测试自动程序。

您现在已准备与 Amazon Lex 进行对话，如下图所示。要测试验证部分，请选择下午 6 点这个时间，然后您的 Lambda 函数会返回一个响应（“我们的营业时间是上午 10 点到下午 5 点。”），并再次向您发出提示。在您提供所有有效的插槽数据后，Lambda 函数会履行此订单。


Test Bot (Latest)
Ready. Build complete.
I want to order flowers
What type of flowers would you like to order?
roses
What day do you want the roses to be picked up?
tomorrow
Pick up the roses at what time on 2017-09-13?
6pm
Our business hours are from ten a m. to five p m. Can you specify a time during this range?
4pm
Okay, your roses will be ready for pickup by 16:00 on 2017-09-13. Does this sound okay?
Clear
Chat to your bot...

下一个步骤

步骤 5 (可选)：查看信息流的详细信息 (控制台)

步骤 5 (可选)：查看信息流的详细信息 (控制台)

本节介绍每个用户输入的客户端与 Amazon Lex 之间的信息流，包括 Lambda 函数的集成。

> **Note**
>
> 本节假定客户端使用 PostText 运行时 API 向 Amazon Lex 发送请求，并显示相应的请求和响应详细信息。有关客户端和 Amazon Lex（客户端在其中使用 PostContent API）之间的信息流示例，请参阅步骤 2a (可选)：查看语音信息流的详细信息 (控制台)。

如需关于 PostText 运行时 API 的更多信息以及关于以下步骤中所示的请求和响应的更多详情，请参阅 PostText。

1. 用户：我想要订些花。

   a. 客户端 (控制台) 将向 Amazon Lex 发送以下 PostText 请求：

   ```
   POST /bot/OrderFlowers/alias/$LATEST/user/ignw84y6seypre4xly5rimopuri2xwnd/text
   "Content-Type":"application/json"
   "Content-Encoding":"amz-1.0"

   {
       "inputText": "I would like to order some flowers",
       "sessionAttributes": {}
   }
   ```

   请求 URI 和正文都将向 Amazon Lex 提供信息：

   - 请求 URI — 提供机器人名称 (OrderFlowers)、机器人别名 ($LATEST) 和用户名称（用于标识用户的随机字符串）。后面的 text 表示它是一个 PostText API 请求 (而不是 PostContent)。
   - 请求正文 – 包括用户输入 (inputText) 和空 sessionAttributes。在客户端发出第一个请求时，没有会话属性。稍后，Lambda 函数会启动这些属性。

   b. 在 inputText 中，Amazon Lex 可检测意图 (OrderFlowers)。此意图配置有 Lambda 函数作为代码挂钩，以用于用户数据初始化和验证。因此，Amazon Lex 将通过传递以下信息作为事件数据来调用该 Lambda 函数：

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook",
    "userId": "ignw84y6seypre4xly5rimopuri2xwnd",
    "sessionAttributes": {},
    "bot": {
        "name": "OrderFlowers",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "OrderFlowers",
        "slots": {
            "PickupTime": null,
            "FlowerType": null,
            "PickupDate": null
        },
        "confirmationStatus": "None"
    }
}
```

有关更多信息，请参阅 输入事件格式。

除了客户端发送的信息以外，Amazon Lex 还包含以下额外数据：

- `messageVersion` — Amazon Lex 当前只支持 1.0 版。
- `invocationSource` — 表明 Lambda 函数调用的目的。在本例中，它将执行用户数据初始化和验证。此时，Amazon Lex 知道用户没有提供所有插槽数据以履行意图。
- 将所有槽值设置为空值的 `currentIntent` 信息。

c. 此时，所有槽值均为空值。Lambda 函数没有任何内容需要验证。Lambda 函数会向 Amazon Lex 返回以下响应：

```
{
    "sessionAttributes": {},
    "dialogAction": {
        "type": "Delegate",
        "slots": {
            "PickupTime": null,
            "FlowerType": null,
```

```
            "PickupDate": null
        }
    }
}
```

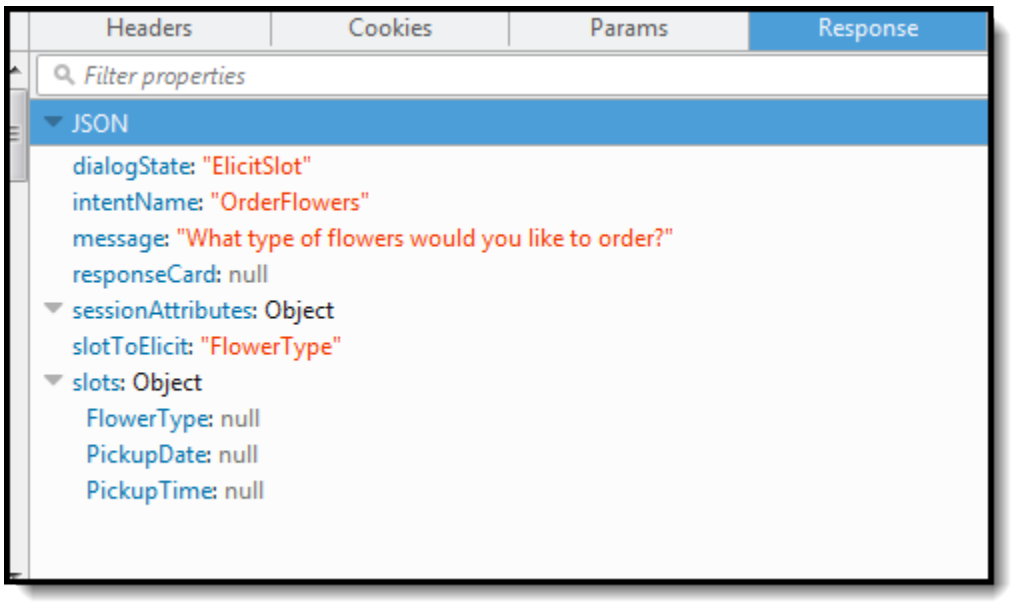
有关响应格式的信息，请参阅 响应格式。

请注意以下几点：

- `dialogAction.type` — 通过将此值设置为 `Delegate`，Lambda 函数会将决定下一个操作的责任委派给 Amazon Lex。

> **Note**
>
> 如果 Lambda 函数在用户数据验证中检测到任何内容，它会指示 Amazon Lex 下一步操作，如接下来的几个步骤所示。

d. 根据 `dialogAction.type`，Amazon Lex 将决定下一步操作。由于没有一个槽得到填充，它将决定引出 `FlowerType` 槽的值。它将选择此槽的其中一个值引出提示（“您想要订哪种类型的花？”），并将以下响应发送回客户端：



客户端将显示响应中的消息。

2. 用户：玫瑰

a. 客户端将向 Amazon Lex 发送以下 PostText 请求：

```
POST /bot/OrderFlowers/alias/$LATEST/user/ignw84y6seypre4xly5rimopuri2xwnd/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "roses",
    "sessionAttributes": {}
}
```

在请求正文中，inputText 将提供用户输入。sessionAttributes 将保留为空。

b. Amazon Lex 解读当前意图的上下文中的 inputText。服务会记住它已经向特定用户询问了关于 FlowerType 槽的信息。它将更新当前意图中的插槽值，并调用具有以下事件数据的 Lambda 函数：

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook",
    "userId": "ignw84y6seypre4xly5rimopuri2xwnd",
    "sessionAttributes": {},
    "bot": {
        "name": "OrderFlowers",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "OrderFlowers",
        "slots": {
            "PickupTime": null,
            "FlowerType": "roses",
            "PickupDate": null
        },
        "confirmationStatus": "None"
    }
}
```

请注意以下几点：

- invocationSource – 继续为 DialogCodeHook (我们只验证用户数据)。
- currentIntent.slots — Amazon Lex 已将 FlowerType 插槽更新为“玫瑰”。

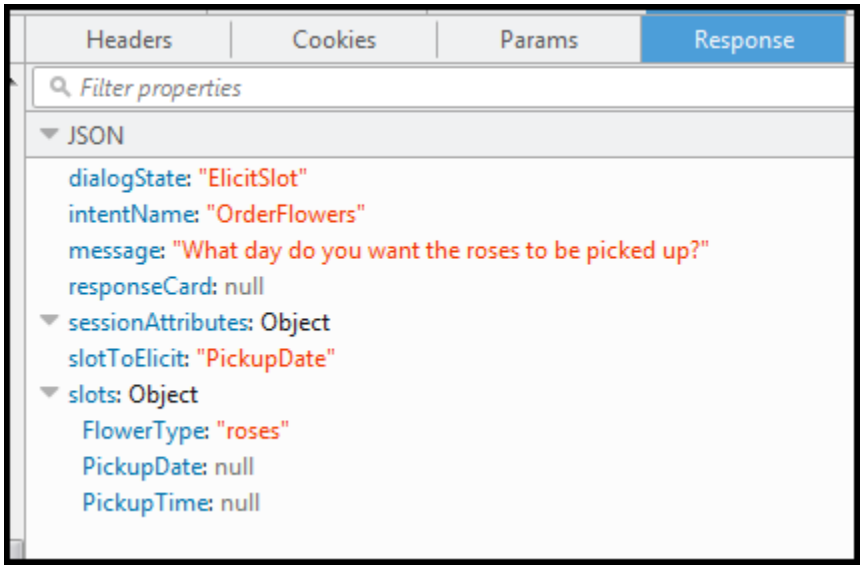
c. 根据 DialogCodeHook 的 invocationSource 值，Lambda 函数会执行用户数据验证。它会将 roses 识别为有效的插槽值（并将 Price 设置为会话属性），然后向 Amazon Lex 返回以下响应。

```
{
    "sessionAttributes": {
        "Price": 25
    },
    "dialogAction": {
        "type": "Delegate",
        "slots": {
            "PickupTime": null,
            "FlowerType": "roses",
            "PickupDate": null
        }
    }
}
```

请注意以下几点：

- sessionAttributes — Lambda 函数已经将（玫瑰的）Price 添加为会话属性。
- dialogAction.type – 设置为 Delegate。用户数据是有效的，因此 Lambda 函数会指示 Amazon Lex 选择下一步操作。

d. 根据 dialogAction.type，Amazon Lex 选择下一步操作。Amazon Lex 知道它需要更多插槽数据，因此它将根据意图配置选择下一个具有最高优先级的未填充插槽 (PickupDate)。Amazon Lex 会根据意图配置，针对这个插槽选择一个值引发提示消息（“您想要在哪天收到玫瑰？”），然后向客户端发送回以下响应：



客户端在响应中显示消息 –“What day do you want the roses to be picked up?”。

3. 用户：明天

   a. 客户端将向 Amazon Lex 发送以下 PostText 请求：

```
POST /bot/OrderFlowers/alias/$LATEST/user/ignw84y6seypre4xly5rimopuri2xwnd/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "tomorrow",
    "sessionAttributes": {
        "Price": "25"
    }
}
```

   在请求正文中，inputText 将提供用户输入，然后客户端会将会话属性传递回服务。

   b. Amazon Lex 会记住上下文（它正引发 PickupDate 插槽的数据）。在这个上下文中，它知道 inputText 值用于 PickupDate 槽。然后，Amazon Lex 将通过发送以下事件调用 Lambda 函数：

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook",
    "userId": "ignw84y6seypre4xly5rimopuri2xwnd",
    "sessionAttributes": {
```

```
        "Price": "25"
    },
    "bot": {
        "name": "OrderFlowersCustomWithRespCard",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "OrderFlowers",
        "slots": {
            "PickupTime": null,
            "FlowerType": "roses",
            "PickupDate": "2017-01-05"
        },
        "confirmationStatus": "None"
    }
}
```

Amazon Lex 已经通过设置 PickupDate 值更新了 currentIntent.slots。另请注意，该服务会将 sessionAttributes 传递到 Lambda 函数。

c. 根据 DialogCodeHook 的 invocationSource 值，Lambda 函数将执行用户数据验证。它会将 PickupDate 插槽值识别为有效值，并向 Amazon Lex 返回以下响应：

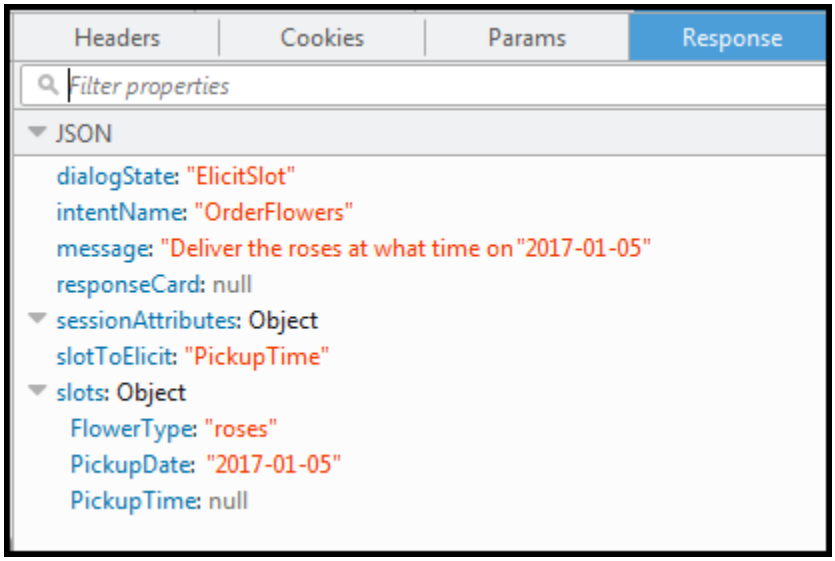
```
{
    "sessionAttributes": {
        "Price": 25
    },
    "dialogAction": {
        "type": "Delegate",
        "slots": {
            "PickupTime": null,
            "FlowerType": "roses",
            "PickupDate": "2017-01-05"
        }
    }
}
```

请注意以下几点：

- sessionAttributes – 无更改。

- `dialogAction.type` – 设置为 `Delegate`。用户数据是有效的，因此 Lambda 函数会指示 Amazon Lex 选择下一步操作。

d. 根据 `dialogAction.type`，Amazon Lex 选择下一步操作。Amazon Lex 知道它需要更多插槽数据，因此它将根据意图配置选择下一个具有最高优先级的未填充插槽 (`PickupTime`)。Amazon Lex 会根据意图配置，针对这个插槽选择一项提示消息（“在 2017 年 1 月 5 日的什么时间配送玫瑰？”），并将以下响应发送回客户端：



客户端会显示响应中的消息（“在 2017 年 1 月 5 日的什么时间配送玫瑰？”）

4. 用户：下午 4 点

a. 客户端将向 Amazon Lex 发送以下 PostText 请求：

```
POST /bot/OrderFlowers/alias/$LATEST/user/ignw84y6seypre4xly5rimopuri2xwnd/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "4 pm",
    "sessionAttributes": {
        "Price": "25"
    }
}
```

在请求正文中，`inputText` 将提供用户输入。客户端将在请求中传递 `sessionAttributes`。

b. Amazon Lex 了解上下文。它了解它正在引出 PickupTime 槽的数据。在此上下文中，它知道 inputText 值用于 PickupTime 插槽。然后，Amazon Lex 将通过发送以下事件调用 Lambda 函数：

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook",
    "userId": "ignw84y6seypre4xly5rimopuri2xwnd",
    "sessionAttributes": {
        "Price": "25"
    },
    "bot": {
        "name": "OrderFlowersCustomWithRespCard",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "OrderFlowers",
        "slots": {
            "PickupTime": "16:00",
            "FlowerType": "roses",
            "PickupDate": "2017-01-05"
        },
        "confirmationStatus": "None"
    }
}
```

Amazon Lex 已经通过设置 PickupTime 值更新了 currentIntent.slots。

c. 根据 DialogCodeHook 的 invocationSource 值，Lambda 函数会执行用户数据验证。它会将 PickupDate 插槽值识别为有效值，并向 Amazon Lex 返回以下响应。

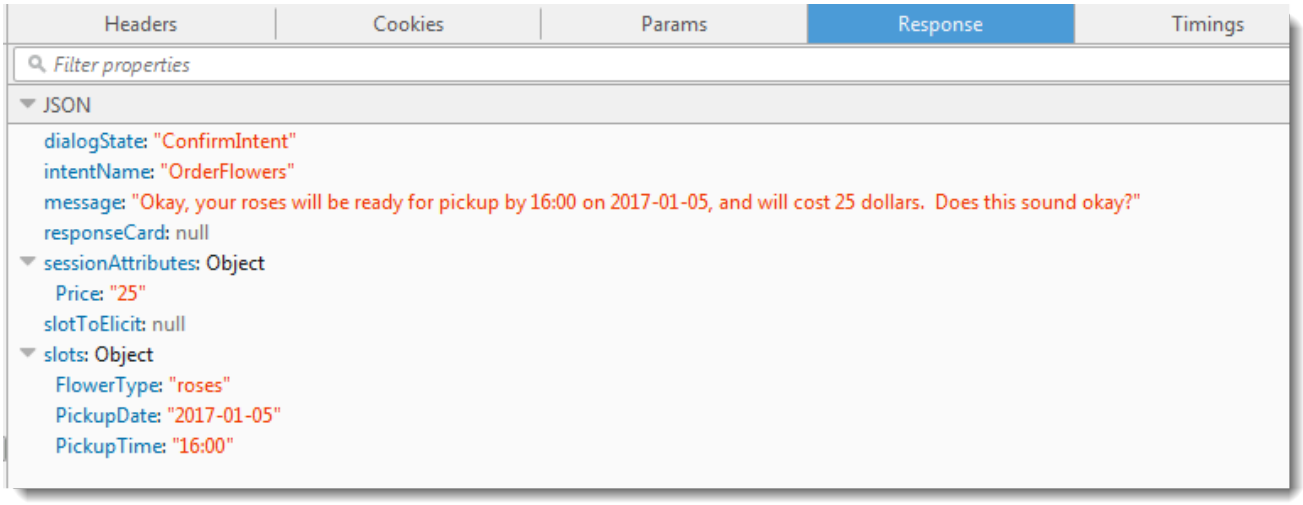
```
{
    "sessionAttributes": {
        "Price": 25
    },
    "dialogAction": {
        "type": "Delegate",
        "slots": {
            "PickupTime": "16:00",
            "FlowerType": "roses",
```

```
            "PickupDate": "2017-01-05"
        }
    }
}
```

请注意以下几点：

- `sessionAttributes` – 会话属性无更改。
- `dialogAction.type` – 设置为 `Delegate`。用户数据是有效的，因此 Lambda 函数会指示 Amazon Lex 选择下一步操作。

d. 此时，Amazon Lex 知道它具有所有插槽数据。此目的配置有确认提示。因此，在履行意图之前，Amazon Lex 将向用户发送以下响应来请求确认：



客户端只显示响应中的消息并等待用户响应。

5. 用户：是

a. 客户端将向 Amazon Lex 发送以下 PostText 请求：

```
POST /bot/OrderFlowers/alias/$LATEST/user/ignw84y6seypre4xly5rimopuri2xwnd/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
    "inputText": "yes",
    "sessionAttributes": {
        "Price": "25"
    }
```

```
}
```

b. Amazon Lex 将解释确认当前意图的上下文中的 inputText。Amazon Lex 了解用户希望继续完成订单。此时，Amazon Lex 将通过发送以下事件来调用 Lambda 函数以履行意图，这会在它发送给 Lambda 函数的事件中将 invocationSource 设置为 FulfillmentCodeHook。Amazon Lex 也将 confirmationStatus 设置为 Confirmed。

```
{
    "messageVersion": "1.0",
    "invocationSource": "FulfillmentCodeHook",
    "userId": "ignw84y6seypre4xly5rimopuri2xwnd",
    "sessionAttributes": {
        "Price": "25"
    },
    "bot": {
        "name": "OrderFlowersCustomWithRespCard",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "OrderFlowers",
        "slots": {
            "PickupTime": "16:00",
            "FlowerType": "roses",
            "PickupDate": "2017-01-05"
        },
        "confirmationStatus": "Confirmed"
    }
}
```

请注意以下几点：

- invocationSource — 此时，Amazon Lex 将此值设置为 FulfillmentCodeHook，指示 Lambda 函数履行意图。
- confirmationStatus – 设置为 Confirmed。

c. 此时，Lambda 函数履行 OrderFlowers 意图，并返回以下响应：

```
{
    "sessionAttributes": {
```

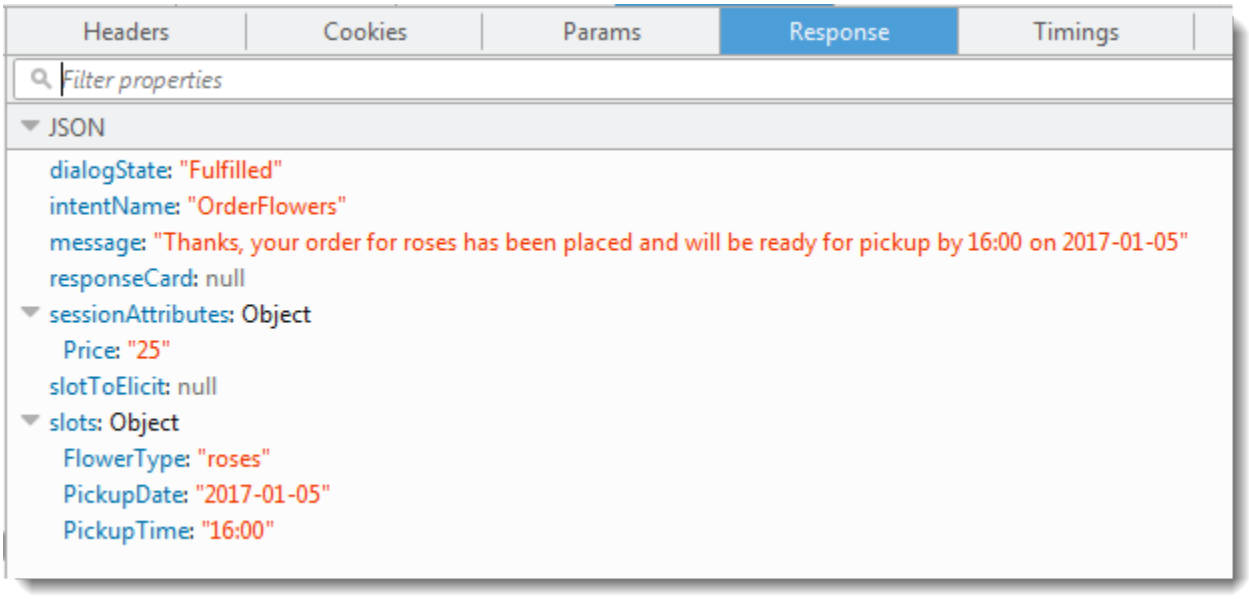
```
        "Price": "25"
    },
    "dialogAction": {
        "type": "Close",
        "fulfillmentState": "Fulfilled",
        "message": {
            "contentType": "PlainText",
            "content": "Thanks, your order for roses has been placed and will
 be ready for pickup by 16:00 on 2017-01-05"
        }
    }
}
```

请注意以下几点：

- `dialogAction.type` – Lambda 函数将该值设置为 `Close`，指示 Amazon Lex 不要期待用户响应。
- `dialogAction.fulfillmentState` – 设置为“已完成”，并且包含要传达给用户的适当 `message`。

d. Amazon Lex 将查看 `fulfillmentState`，并将以下响应发送回客户端。

然后，Amazon Lex 将向客户端返回以下信息：



请注意：

- `dialogState` — Amazon Lex 将此值设置为 `fulfilled`。

- `message` — 与 Lambda 函数提供的消息相同。

客户端将显示消息。

6. 现在，再次测试自动程序。要建立新的 (用户) 上下文，请选择测试窗口中的 Clear 链接。现在提供 `OrderFlowers` 目的的无效槽数据。此时，Lambda 函数将执行数据验证，将无效的插槽数据值重置为空值，并要求 Amazon Lex 提示用户提供有效数据。例如，请尝试以下操作：

   - “茉莉花”作为鲜花类型 (它不是一种受支持的鲜花类型)。
   - “昨天”作为您想要取花的日期。
   - 下单后，输入另一种鲜花类型，而不是回复“yes”来确认订单。作为响应，Lambda 函数将更新会话属性中的 `Price`，以维护鲜花订单的滚动合计。

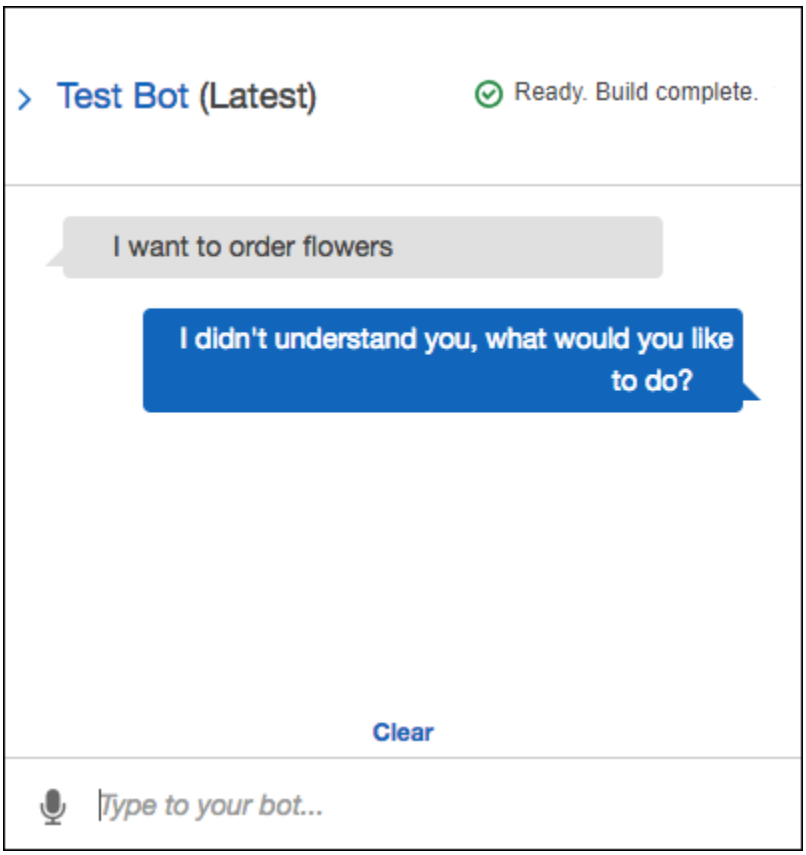
   Lambda 函数还将执行履行活动。

下一个步骤

步骤 6：更新目的配置以添加表达 (控制台)

步骤 6：更新目的配置以添加表达 (控制台)

`OrderFlowers` 自动程序只配置两个表达。这对 Amazon Lex 提供的信息有限，而它要依赖这些信息来构建能够识别并响应用户意图的机器学习模型。尝试键入“我想订购鲜花”，如下面的测试窗口中所示。Amazon Lex 无法识别该文本，并响应“我不明白，请问您想做什么呢？” 您可以通过添加更多表达来改进机器学习模型。

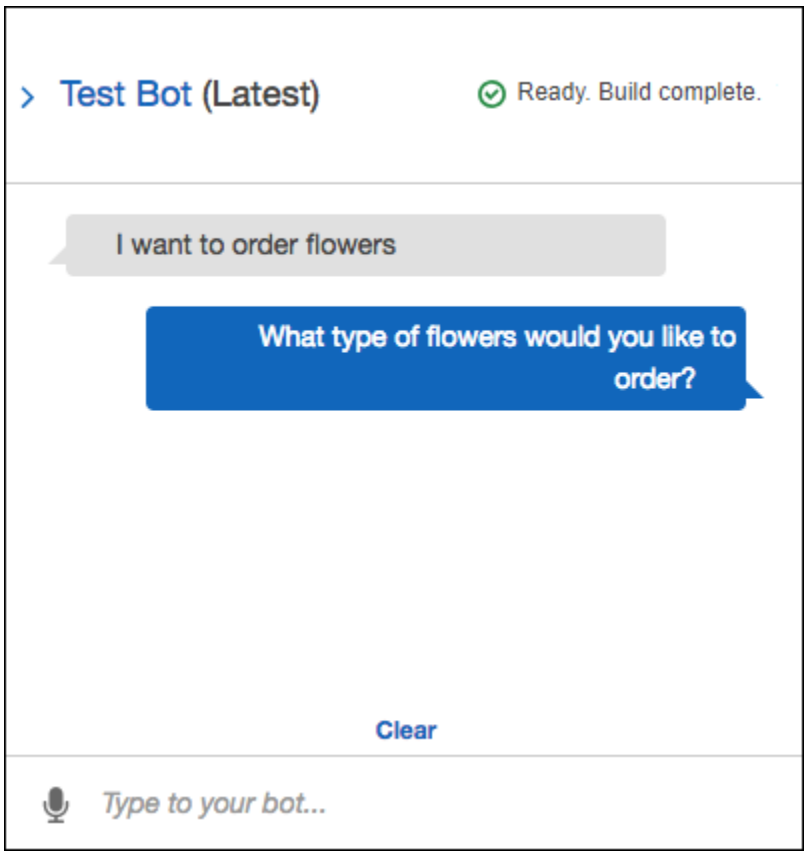


您添加的每句话都为 Amazon Lex 提供了更多信息来学习如何响应用户。您无需添加准确的表达，Amazon Lex 会基于您提供的示例形成相应的概念，以识别精确的匹配和类似的输入。

添加表达 (控制台)

1. 在意图编辑器的示例言语部分中键入言语“我想订购鲜花”，然后单击该新言语旁边的加号图标，将其添加到意图。

2. 将自动程序构建为能够接受更改。选择 Build，然后再次选择 Build。
3. 测试自动程序，以确认它识别出了新表达。在测试窗口中，如下图所示，键入“我想订购鲜花”。Amazon Lex 会识别出该短语，并提供响应“您想要订购什么花？”。



下一个步骤

步骤 7 (可选)：清除 (控制台)

步骤 7 (可选)：清除 (控制台)

现在，删除您创建的资源并清除账户。

您只能删除未使用的资源。一般来说，您应该按照以下顺序删除资源：

- 删除自动程序以释放目的资源。
- 删除目的以释放槽类型资源。
- 最后删除槽类型。

清除账户 (控制台)

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。

2. 从机器人列表中，选中旁边的复选框OrderFlowers。
3. 要删除自动程序，请选择 Delete，然后在确认对话框中选择 Continue。
4. 在左侧窗格中，选择 Intents。
5. 在目的列表中，选择 OrderFlowersIntent。
6. 要删除目的，请选择 Delete，然后在确认对话框中选择 Continue。
7. 在左侧窗格中，选择 Slot types。
8. 在槽类型列表中，选择 Flowers。
9. 要删除槽类型，请选择 Delete，然后在确认对话框中选择 Continue。

您已经删除了创建的 Amazon Lex 所有资源并清理了账户。如果需要，您可以使用 Lambda 控制台删除本练习中使用的 Lambda 函数。

练习 2：创建自定义 Amazon Lex 机器人

在本练习中，您将使用 Amazon Lex 控制台创建一个用于订购披萨的自定义机器人 (`OrderPizzaBot`)。您需要通过以下操作配置该自动程序：添加自定义目的 (`OrderPizza`)、定义自定义槽类型以及定义要完成披萨订单所需的槽 (披萨饼皮、尺寸等)。有关槽类型和槽的更多信息，请参阅 Amazon Lex：工作原理。

主题

- 步骤 1：创建 Lambda 函数
- 步骤 2：创建自动程序
- 步骤 3：构建和测试自动程序
- 步骤 4 (可选)：清理

步骤 1：创建 Lambda 函数

首先，创建一个用于履行披萨订单的 Lambda 函数。您将在下一节中创建的 Amazon Lex 机器人中指定此函数。

创建 Lambda 函数

1. 登录 AWS Management Console 并打开 AWS Lambda 控制台，网址为https://console.aws.amazon.com/lambda/。

2. 选择 Create function (创建函数)。
3. 在创建函数页面上，选择从 Scratch 开始创作。

   因为您使用本练习中提供的自定义代码创建 Lambda 函数，因此请选择从头创建该函数。

   执行以下操作：

   a. 键入名称 (PizzaOrderProcessor)。
   b. 对于 Runtime，选择 Node.js 的最新版本。
   c. 对于 Role，选择 Create new role from template(s)。
   d. 输入新的角色名称 (PizzaOrderProcessorRole)。
   e. 选择 Create function (创建函数)。
4. 在函数页面上，执行以下操作：

   在 Function code 部分中，选择 Edit code inline，然后复制以下 Node.js 函数代码并将其粘贴到窗口中。

```
'use strict';

// Close dialog with the customer, reporting fulfillmentState of Failed or
 Fulfilled ("Thanks, your pizza will arrive in 20 minutes")
function close(sessionAttributes, fulfillmentState, message) {
    return {
        sessionAttributes,
        dialogAction: {
            type: 'Close',
            fulfillmentState,
            message,
        },
    };
}

// --------------- Events -----------------------

function dispatch(intentRequest, callback) {
    console.log(`request received for userId=${intentRequest.userId}, intentName=
${intentRequest.currentIntent.name}`);
    const sessionAttributes = intentRequest.sessionAttributes;
    const slots = intentRequest.currentIntent.slots;
    const crust = slots.crust;
    const size = slots.size;
```

```
    const pizzaKind = slots.pizzaKind;

    callback(close(sessionAttributes, 'Fulfilled',
    {'contentType': 'PlainText', 'content': `Okay, I have ordered your ${size}
 ${pizzaKind} pizza on ${crust} crust`}));

}

// --------------- Main handler -----------------------

// Route the incoming request based on intent.
// The JSON body of the request is provided in the event slot.
export const handler = (event, context, callback) => {
    try {
        dispatch(event,
            (response) => {
                callback(null, response);
            });
    } catch (err) {
        callback(err);
    }
};
```

5. 选择保存。

使用示例事件数据测试 Lambda 函数

在控制台中，使用示例事件数据手动调用 Lambda 函数，以对其进行测试。

测试 Lambda 函数：

1. 登录 AWS Management Console 并打开 AWS Lambda 控制台，网址为https://console.aws.amazon.com/lambda/。
2. 在 Lambda 函数页面上，选择 Lambda 函数 (`PizzaOrderProcessor`).)
3. 在函数页面上的测试事件列表中，选择 Configure test events。
4. 在 Configure test event 页面上，执行以下操作：
   a. 选择 Create new test event（新建测试事件）。
   b. 在 Event name 字段中，为事件输入名称 (`PizzaOrderProcessorTest`)。
   c. 将以下 Amazon Lex 事件复制到窗口中。

```
{
  "messageVersion": "1.0",
  "invocationSource": "FulfillmentCodeHook",
  "userId": "user-1",
  "sessionAttributes": {},
  "bot": {
    "name": "PizzaOrderingApp",
    "alias": "$LATEST",
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "OrderPizza",
    "slots": {
      "size": "large",
      "pizzaKind": "meat",
      "crust": "thin"
    },
    "confirmationStatus": "None"
  }
}
```

5. 选择创建。

AWS Lambda 创建测试，然后返回到函数页面。选择测试，然后 Lambda 会运行您的 Lambda 函数。

在结果框中，选择 Details。控制台将在 Execution result 窗格中显示以下输出。

```
{
  "sessionAttributes": {},
  "dialogAction": {
    "type": "Close",
    "fulfillmentState": "Fulfilled",
    "message": {
      "contentType": "PlainText",
      "content": "Okay, I have ordered your large meat pizza on thin crust."
    }
}
```

下一个步骤

[步骤 2：创建自动程序]

步骤 2：创建自动程序

在本步骤中，您将创建一个自动程序来处理披萨订单。

主题

- 创建自动程序
- 创建目的
- 创建槽类型
- 配置目的
- 配置 自动程序

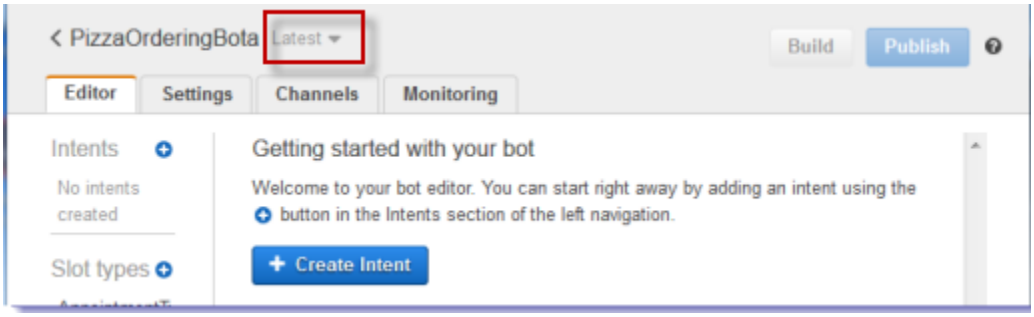
创建自动程序

用所需的最少信息创建 `PizzaOrderingBot` 自动程序。稍后为该自动程序添加目的，即用户希望执行的操作。

创建自动程序

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 创建自动程序。
   a. 如果这是您第一次创建自动程序，请选择开始。否则，请选择 Bots，然后选择 Create。
   b. 在 Create your Lex bot 页面上，选择 Custom bot 并提供以下信息：
      - 机器人名称： PizzaOrderingBot
      - 语言：为您的机器人选择语言和区域设置。
      - Output voice：Salli
      - Session timeout：5 分钟。
      - COPPA：选择适当的回应。
      - 用户言语存储：选择相应的响应。
   c. 选择创建。

      控制台将向 Amazon Lex 发送创建新机器人的请求。Amazon Lex 将机器人版本设置为 `$LATEST`。创建机器人后，Amazon Lex 会显示机器人编辑器选项卡，如下图所示：



- 自动程序版本 Latest 显示在控制台中自动程序名称的旁边。新的 Amazon Lex 资源使用 $LATEST 作为版本。有关更多信息，请参阅 版本控制和别名。
- 由于您尚未创建任何目的或槽类型，因此未列出任何内容。
- Build 和 Publish 是自动程序级别活动。在您配置整个自动程序之后，您将了解有关这些活动的更多信息。

下一个步骤

创建目的

创建目的

现在，用所需的最少信息创建 OrderPizza 目的，即用户要执行的操作。为目的添加槽类型，稍后再配置目的。

创建目的

1. 在 Amazon Lex 控制台中，选择意图旁边的加号 (+)，然后选择创建新意图。
2. 在 Create intent 对话框中，键入目的的名称 (OrderPizza)，然后选择 Add。

控制台将向 Amazon Lex 发送请求，以创建 OrderPizza 意图。在此示例中，您将在创槽类型后为目的创建槽。

下一个步骤

创建槽类型

创建槽类型

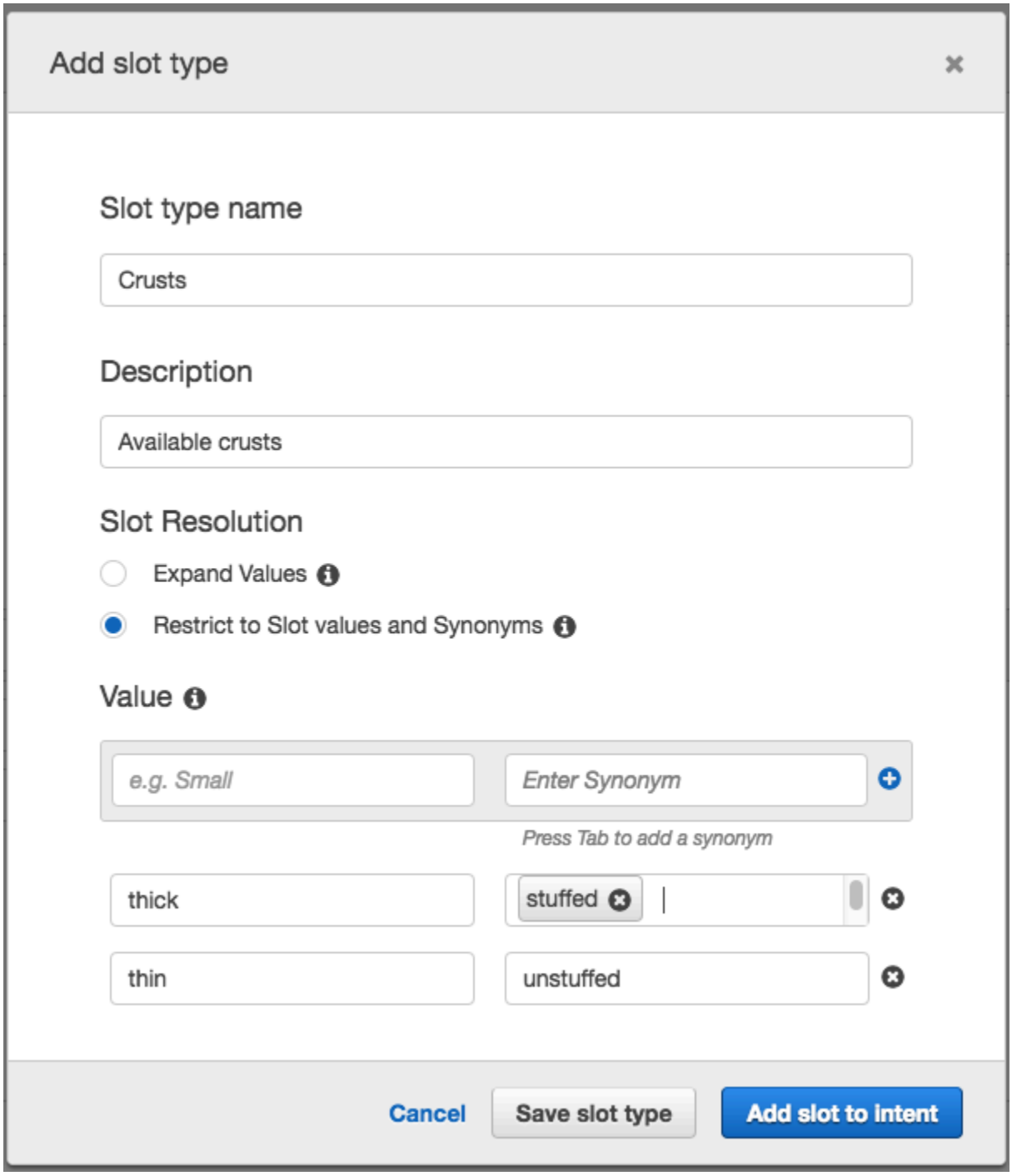
创建 OrderPizza 目的使用的槽类型 (参数值)。

创建槽类型

1. 在左侧菜单中，选择 Slot types 旁边的加号 (+)。
2. 在 Add slot type 对话框中，添加以下内容：

   - Slot type name – Crusts
   - Description – 提供的饼皮
   - 选择 Restrict to Slot values and Synonyms
   - 值 — 类型 **`thick`**。按下 Tab 键并在 Synonym (同义词) 字段中键入 **`stuffed`**。选择加号 (+)。键入 **`thin`**，然后再次选择加号 (+)。

   对话框应类似于下图所示：



3. 选择 Add slot to intent。
4. 在 Intent 页面上，选择 Required。将槽的名称由 **`slotOne`** 更改为 **`crust`**。将提示更改为 **`What kind of crust would you like?`**
5. 使用下表中的值重复Step 1 至Step 4：

名称	描述	值	槽名称	提示
尺寸	提供的尺寸	小、中、大	尺寸	多大尺寸的披萨？
PizzaKind	提供的披萨	蔬菜、芝士	pizzaKind	您是要蔬菜披萨还是芝士披萨？

下一个步骤

[配置目的]

配置目的

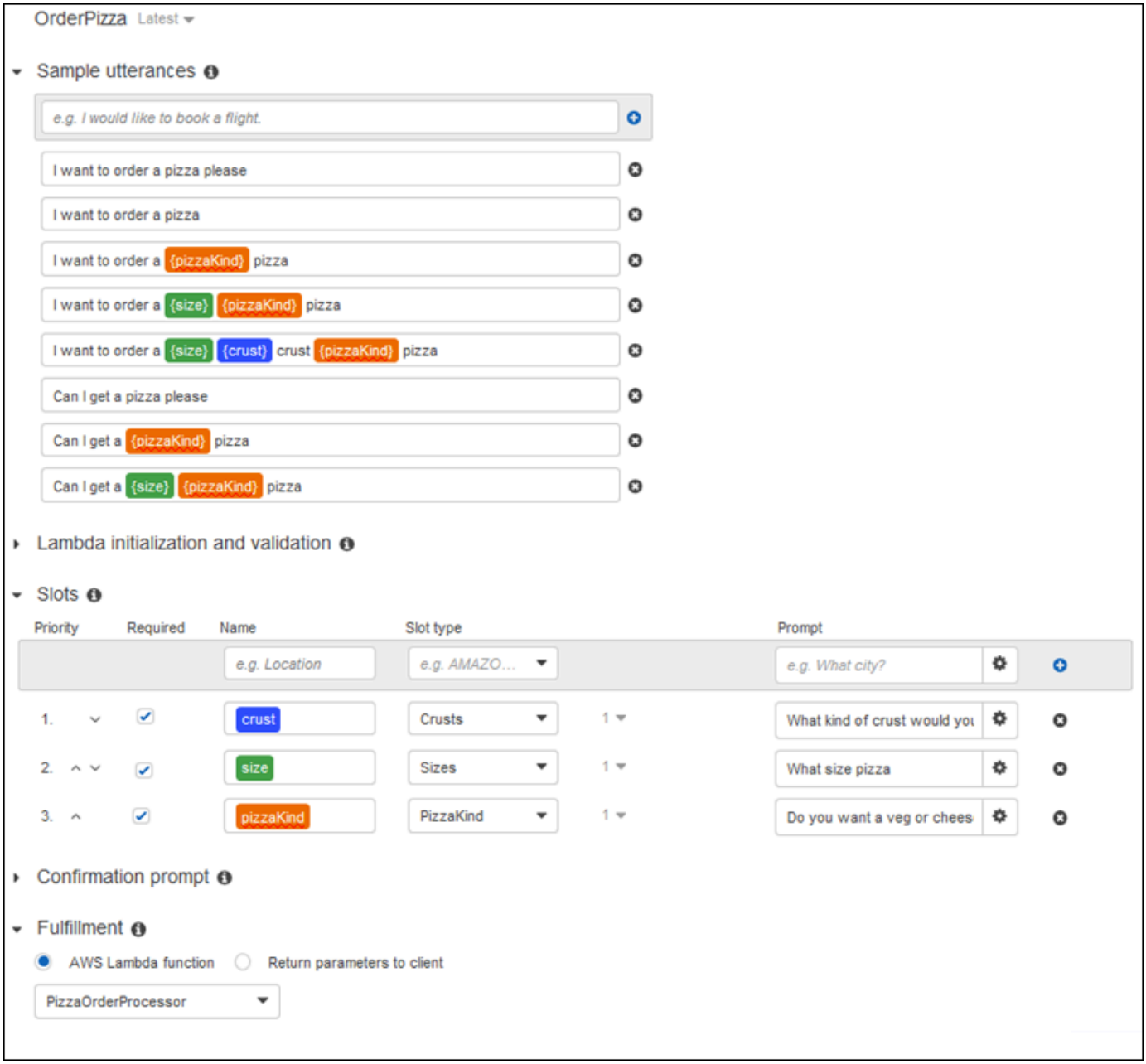
配置 OrderPizza 目的以完成用户订购披萨的请求。

配置目的

- 在OrderPizza配置页面上，按如下方式配置 Intent：
  - 示例言语 — 键入以下字符串。大括号 {} 内为槽名称。
    - 我想要订披萨
    - 我想要订一个披萨
    - 我想要订一个 {pizzaKind} 披萨
    - 我想要订一个 {size} 大小的 {pizzaKind} 披萨
    - 我想要订一个 {size} 大小的 {crust} 皮 {pizzaKind} 披萨
    - 我能否点一个披萨
    - 我能否点一个 {pizzaKind} 披萨
    - 我不能点一个 {size} 大小的 {pizzaKind} 披萨
  - Lambda initialization and validation – 保留默认设置。
  - Confirmation prompt – 保留默认设置。
  - 履行 — 执行以下任务：
    - 选择 AWS Lambda 函数。
    - 选择 **PizzaOrderProcessor**。

- 如果显示向 Lambda 函数添加权限对话框，则选择确定，以便为 `OrderPizza` 意图提供调用 `PizzaOrderProcessor` Lambda 函数的权限。
- 将 None 保留为选中状态。

目的显示如下：



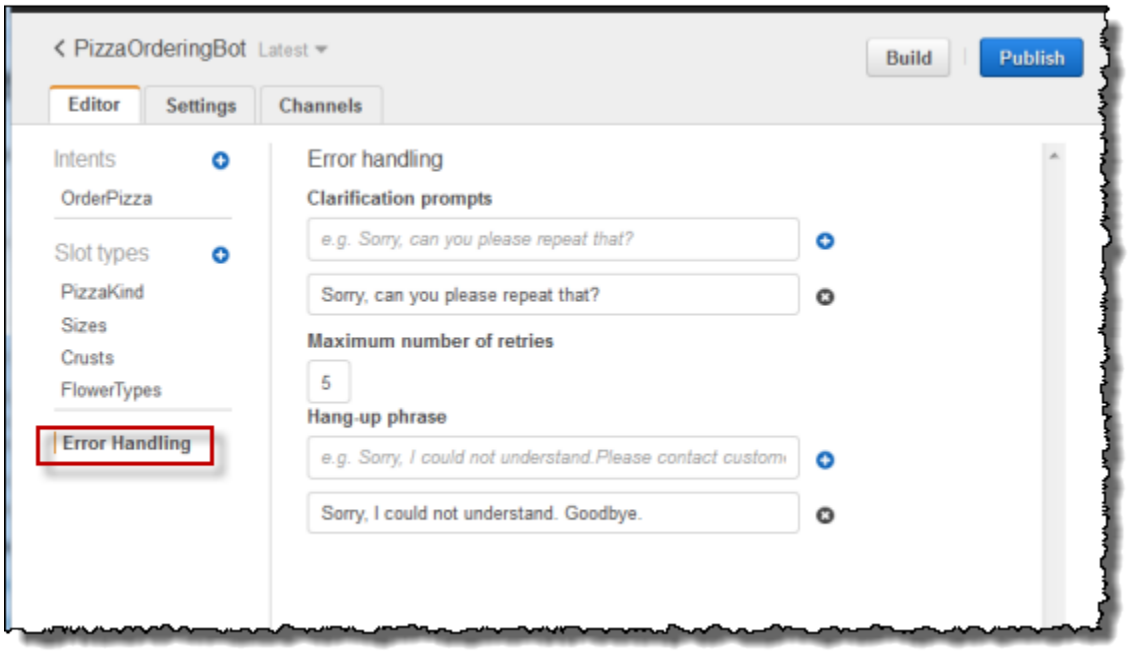
下一个步骤

[配置 自动程序]

配置 自动程序

为 PizzaOrderingBot 自动程序配置错误处理。

1. 导航到 PizzaOrderingBot 自动程序。选择编辑器，然后选择错误处理，如下图所示：



2. 使用 Editor 选项卡配置自动程序错误处理。

   - 您在 Clarification Prompts 中提供的信息将映射到自动程序的 clarificationPrompt 配置。

     当 Amazon Lex 无法确定用户意图时，该服务会返回包含此消息的响应。
   - 您在 Hang-up 短语中提供的信息将映射到自动程序的 abortStatement 配置。

     如果服务在发送一定数量的连续请求之后仍无法确定用户的意图，则 Amazon Lex 会返回包含此消息的响应。

   保留默认值。

下一个步骤

步骤 3：构建和测试自动程序

步骤 3：构建和测试自动程序

通过构建和测试自动程序确保其正常工作。

构建和测试自动程序

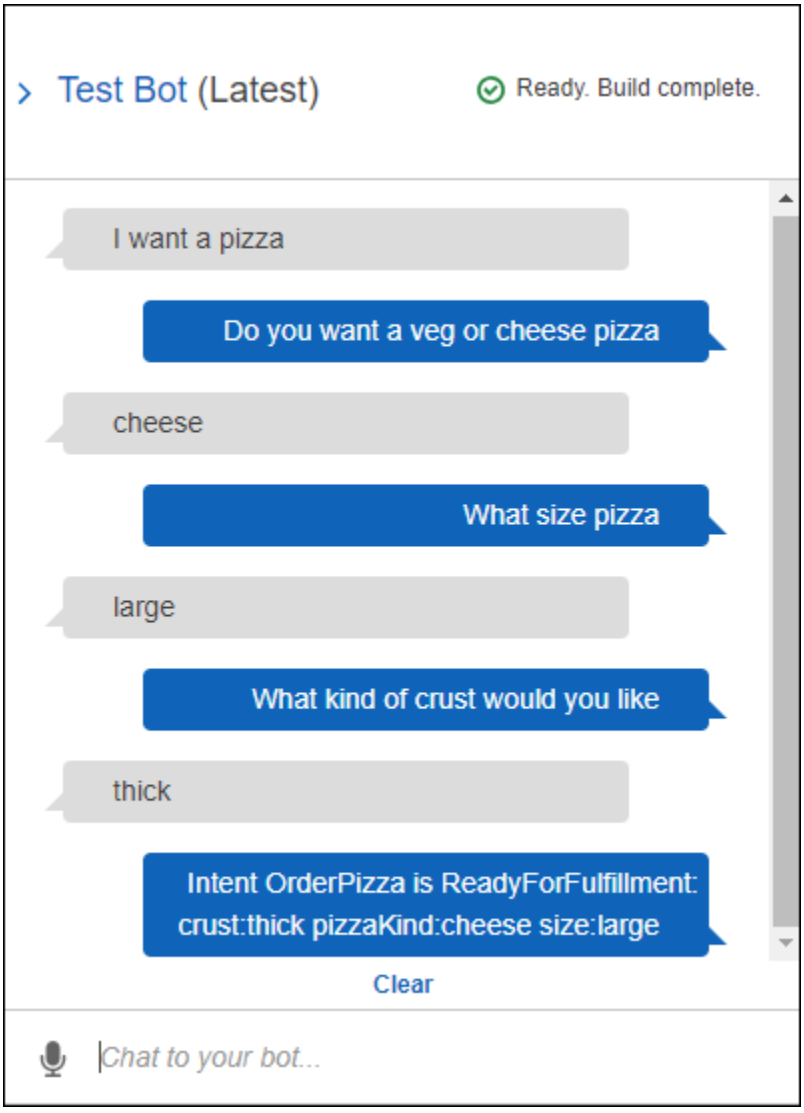
1. 要构建 PizzaOrderingBot 自动程序，请选择 Build。

   Amazon Lex 将为机器人构建机器学习模型。当您测试机器人时，控制台将使用运行时 API 将用户输入发送回 Amazon Lex。然后，Amazon Lex 将使用机器学习模型解释用户输入。

   完成构建可能需要一些时间。

2. 要测试机器人，请在测试机器人窗口中开始与您的 Amazon Lex 机器人交谈。

   - 例如，您可以说或键入：



   - 使用您在 OrderPizza 目的中配置的示例表达来测试自动程序。例如，下面是您为 PizzaOrder 目的配置的一种示例表达：

```
I want a {size} {crust} crust {pizzaKind} pizza
```

要对其进行测试，请键入以下内容：

```
I want a large thin crust cheese pizza
```

当您输入“我想要订一个披萨”时，Amazon Lex 会检测到此意图 (`OrderPizza`)。然后，Amazon Lex 会要求提供插槽信息。

在您提供所有插槽信息后，Amazon Lex 将调用您为该意图配置的 Lambda 函数。

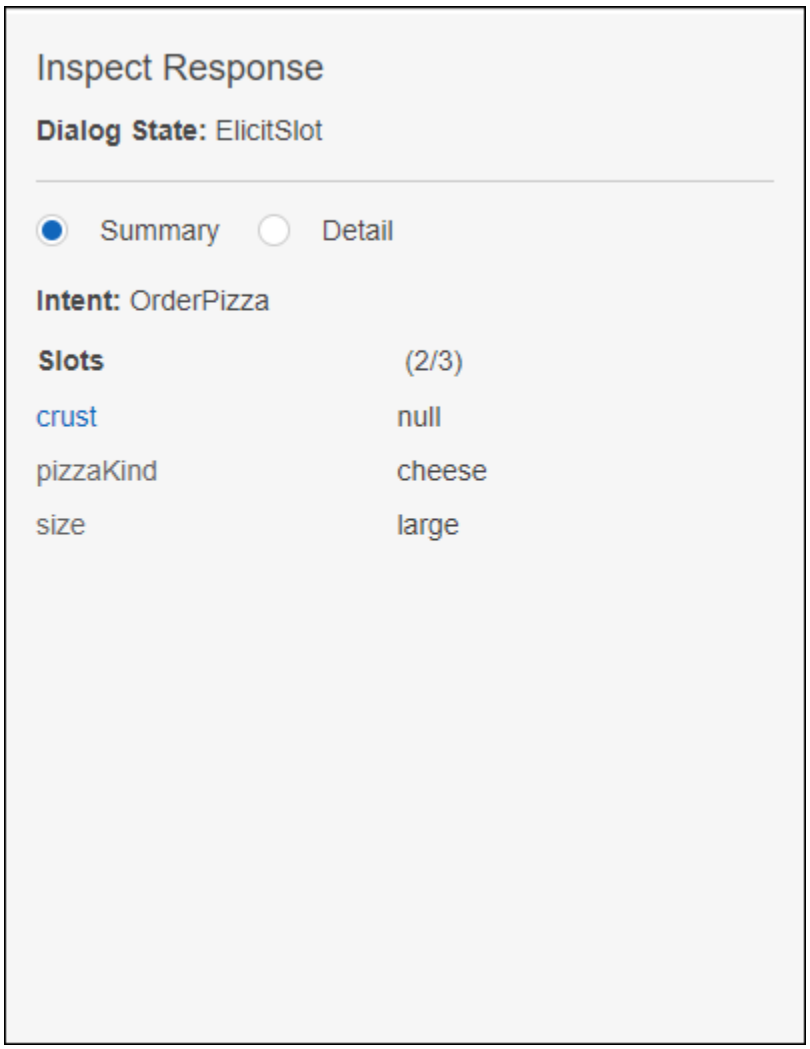
Lambda 函数会向 Amazon Lex 返回一条消息（“好的，我已订购您的...”），Amazon Lex 再将此消息返回给您。

检查响应

通过聊天窗口下方的窗格可以检查来自 Amazon Lex 的响应。该窗格可提供有关自动程序状态的全面信息，该信息会随着您与自动程序的交互而变化。窗格的内容会向您显示操作的当前状态。

- 对话状态 — 与用户对话的当前状态。可为 `ElicitIntent`、`ElicitSlot`、`ConfirmIntent` 或 `Fulfilled`。

- 摘要 — 显示对话框（其中显示正在履行的意图的插槽值）的简化视图，以便于您跟踪信息流。它显示目的名称、槽数和已填充槽数，以及所有槽及其关联值的列表。参见下图：



- 详细信息 — 显示来自聊天机器人的原始 JSON 响应，使您可以在测试和调试聊天机器人时更深入地了解机器人交互和对话的当前状态。如果您在聊天窗口中键入，检查窗格将显示来自 PostText 操作的 JSON 响应。如果您在聊天窗口中说话，检查窗格将显示来自 PostContent 操作的响应标头。参见下图：

下一个步骤

步骤 4 (可选)：清理

步骤 4 (可选)：清理

删除您创建的资源并清理您的账户，以免所创建的资源产生更多费用。

您只能删除未使用的资源。例如，您无法删除目的所引用的槽类型。您无法删除自动程序所引用的目的。

可按以下顺序删除资源：

- 删除自动程序以释放目的资源。
- 删除目的以释放槽类型资源。

- 最后删除槽类型。

清理您的账户

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 从自动程序列表中，选择 PizzaOrderingBot。
3. 要删除该自动程序，请选择 Delete，然后选择 Continue。
4. 在左侧窗格中，选择 Intents。
5. 在目的列表中，选择 OrderPizza。
6. 要删除该目的，请选择 Delete，然后选择 Continue。
7. 在左侧菜单中，选择 Slot types。
8. 在槽类型列表中，选择 Crusts。
9. 要删除该槽类型，请选择 Delete，然后选择 Continue。
10. 对于 `Sizes` 和 `PizzaKind` 槽类型，请重复 Step 8 和 Step 9。

您已经删除了创建的所有资源并清除了账户。

后续步骤

- 发布版本和创建别名
- 使用创建 Amazon Lex 机器人 AWS Command Line Interface

练习 3：发布版本和创建别名

在入门练习 1 和 2 中，您创建了一个自动程序并对其进行了测试。在本练习中，您将执行以下操作：

- 发布机器人的新版本。Amazon Lex 创建 `$LATEST` 版本的快照副本以发布新版本。
- 创建指向新版本的别名。

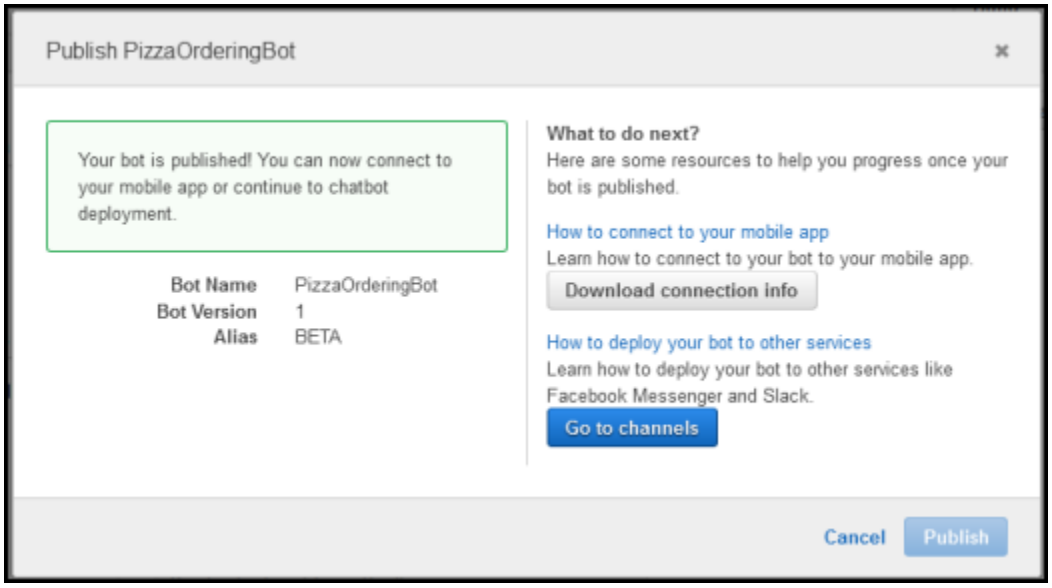
有关版本控制和别名的更多信息，请参阅版本控制和别名。

请执行以下操作以发布您在本练习中创建的自动程序的版本：

1. 在 Amazon Lex 控制台中，选择您已创建的机器人之一。

确认控制台是否在自动程序名称旁边显示 $LATEST 作为自动程序版本。

2. 选择 发布。
3. 在 *botname* “发布” 向导中，指定别名**BETA**，然后选择 “发布”。
4. 确认 Amazon Lex 控制台是否在机器人名称旁边显示新版本，如下图所示。



现在，您有一个正在运行的自动程序，该程序具有已发布的版本和别名，而且您可以部署该自动程序 (部署到移动应用程序中，或将自动程序与 Facebook Messenger 集成)。有关示例，请参阅 将 Amazon Lex 机器人与 Facebook Messenger 集成。

步骤 4：入门 (AWS CLI)

在此步骤中，您将使用创建、测试和修改 Amazon Lex 机器人。 AWS CLI 要完成这些练习，您需要熟悉如何使用 CLI 并且有文本编辑器。有关更多信息，请参阅 第 2 步：设置 AWS Command Line Interface

- 练习 1 — 创建并测试 Amazon Lex 机器人。本练习提供了创建自定义槽类型、目的和自动程序所需的所有 JSON 对象。有关更多信息，请参阅 Amazon Lex：工作原理
- 练习 2 — 更新在练习 1 中创建的机器人来添加额外的示例言语。Amazon Lex 使用示例言语为您的机器人构建机器学习模型。
- 练习 3 — 更新在练习 1 中创建的机器人来添加 Lambda 函数以验证用户输入和履行意图。
- 练习 4 — 发布在练习 1 中创建的插槽类型、意图和机器人资源的一个版本。版本是无法更改的资源快照。

- 练习 5 — 为在练习 1 中创建的机器人创建别名。
- 练习 6 — 删除在练习 1 中创建的插槽类型、意图和机器人以及在练习 5 中创建的别名以清理账户。

主题

练习 1：创建 Amazon Lex 机器人 (AWS CLI)

一般来说，创建自动程序时，您必须：

1. 创建槽类型以定义自动程序将处理的信息。
2. 创建目的以定义自动程序支持的用户操作。使用之前创建的自定义槽类型定义目的所需的槽 (参数)。
3. 创建使用您所定义的目的的自动程序。

在本练习中，您将使用 CLI 创建和测试一个新的 Amazon Lex 机器人。请使用我们提供的 JSON 结构创建自动程序。要运行本练习中的命令，您需要知道将在其中运行命令的区域。有关区域列表，请参阅模型构建配额 。

主题

步骤 1：创建服务相关角色 (AWS CLI)

Amazon Lex 扮演 AWS Identity and Access Management 与服务相关的角色，代表您的机器人调用 AWS 服务。这些角色在您的账户中，与 Amazon Lex 使用案例关联并有预定义的权限。有关更多信息，请参阅 对 Amazon Lex 使用服务相关角色。

如果您已使用控制台创建了 Amazon Lex 机器人，服务相关角色就已自动创建。跳至步骤 2：创建自定义槽类型 (AWS CLI)。

创建服务相关角色 (AWS CLI)

1. 在中 AWS CLI，键入以下命令：

```
aws iam create-service-linked-role --aws-service-name lex.amazonaws.com
```

2. 使用以下命令检查策略：

```
aws iam get-role --role-name AWSServiceRoleForLexBots
```

响应如下：

```
{
    "Role": {
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Action": "sts:AssumeRole",
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "lex.amazonaws.com"
                    }
                }
            ]
        },
        "RoleName": "AWSServiceRoleForLexBots",
        "Path": "/aws-service-role/lex.amazonaws.com/",
        "Arn": "arn:aws:iam::account-id:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
}
```

下一个步骤

步骤 2：创建自定义槽类型 (AWS CLI)

步骤 2：创建自定义槽类型 (AWS CLI)

创建一个包含可订购鲜花枚举值的自定义槽类型。在下一步骤中创建 `OrderFlowers` 目的时会用到这一类型。槽类型 定义了目的的槽 (参数) 的可能值。

要运行本练习中的命令，您需要知道将在其中运行命令的区域。有关区域列表，请参阅 模型构建配额 。

创建自定义槽类型 (AWS CLI)

1. 创建名为 **`FlowerTypes.json`** 的文本文件。将 JSON 代码从 FlowerTypes.json 复制到该文本文件中。
2. 使用调用PutSlotType操作 AWS CLI 来创建插槽类型。此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws lex-models put-slot-type \
    --region region \
    --name FlowerTypes \
    --cli-input-json file://FlowerTypes.json
```

服务器的响应如下：

```
{
    "enumerationValues": [
        {
            "value": "tulips"
        },
        {
            "value": "lilies"
        },
        {
            "value": "roses"
        }
    ],
    "name": "FlowerTypes",
    "checksum": "checksum",
    "version": "$LATEST",
```

```
    "lastUpdatedDate": timestamp,
    "createdDate": timestamp,
    "description": "Types of flowers to pick up"
}
```

下一个步骤

[步骤 3：创建目的 (AWS CLI)]

FlowerTypes.json

以下代码是创建 FlowerTypes 自定义槽类型所需的 JSON 数据：

```
{
    "enumerationValues": [
        {
            "value": "tulips"
        },
        {
            "value": "lilies"
        },
        {
            "value": "roses"
        }
    ],
    "name": "FlowerTypes",
    "description": "Types of flowers to pick up"
}
```

步骤 3：创建目的 (AWS CLI)

为 OrderFlowersBot 自动程序创建一个目的并提供三个槽 (参数)。这些槽允许自动程序完成以下目的：

- FlowerType 是一个自定义槽类型，用于指定可以订购哪些类型的鲜花。
- AMAZON.DATE 和 AMAZON.TIME 是内置槽类型，用于获取从用户那里送花的日期和时间。

要运行本练习中的命令，您需要知道将在其中运行命令的区域。有关区域列表，请参阅 模型构建配额 。

创建 **OrderFlowers**目的 (AWS CLI)

1. 创建名为 **OrderFlowers.json** 的文本文件。将 JSON 代码从 OrderFlowers.json 复制到该文本文件中。
2. 在中 AWS CLI，调用PutIntent操作来创建意图。此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws lex-models put-intent \
   --region region \
   --name OrderFlowers \
   --cli-input-json file://OrderFlowers.json
```

服务器响应如下：

```
{
    "confirmationPrompt": {
        "maxAttempts": 2,
        "messages": [
            {
                "content": "Okay, your {FlowerType} will be ready for pickup by
 {PickupTime} on {PickupDate}.  Does this sound okay?",
                "contentType": "PlainText"
            }
        ]
    },
    "name": "OrderFlowers",
    "checksum": "checksum",
    "version": "$LATEST",
    "rejectionStatement": {
        "messages": [
            {
                "content": "Okay, I will not place your order.",
                "contentType": "PlainText"
            }
        ]
    },
    "createdDate": timestamp,
    "lastUpdatedDate": timestamp,
    "sampleUtterances": [
        "I would like to pick up flowers",
        "I would like to order some flowers"
    ],
```

```
   "slots": [
       {
           "slotType": "AMAZON.TIME",
           "name": "PickupTime",
           "slotConstraint": "Required",
           "valueElicitationPrompt": {
               "maxAttempts": 2,
               "messages": [
                   {
                       "content": "Pick up the {FlowerType} at what time on
{PickupDate}?",
                       "contentType": "PlainText"
                   }
               ]
           },
           "priority": 3,
           "description": "The time to pick up the flowers"
       },
       {
           "slotType": "FlowerTypes",
           "name": "FlowerType",
           "slotConstraint": "Required",
           "valueElicitationPrompt": {
               "maxAttempts": 2,
               "messages": [
                   {
                       "content": "What type of flowers would you like to
order?",
                       "contentType": "PlainText"
                   }
               ]
           },
           "priority": 1,
           "slotTypeVersion": "$LATEST",
           "sampleUtterances": [
               "I would like to order {FlowerType}"
           ],
           "description": "The type of flowers to pick up"
       },
       {
           "slotType": "AMAZON.DATE",
           "name": "PickupDate",
           "slotConstraint": "Required",
           "valueElicitationPrompt": {
```

```
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "What day do you want the {FlowerType} to be
 picked up?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 2,
            "description": "The date to pick up the flowers"
        }
    ],
    "fulfillmentActivity": {
        "type": "ReturnIntent"
    },
    "description": "Intent to order a bouquet of flowers for pick up"
}
```

下一个步骤

步骤 4：创建自动程序 (AWS CLI)

OrderFlowers.json

以下代码是创建 `OrderFlowers` 目的所需的 JSON 数据：

```
{
    "confirmationPrompt": {
        "maxAttempts": 2,
        "messages": [
            {
                "content": "Okay, your {FlowerType} will be ready for pickup by
 {PickupTime} on {PickupDate}.  Does this sound okay?",
                "contentType": "PlainText"
            }
        ]
    },
    "name": "OrderFlowers",
    "rejectionStatement": {
        "messages": [
            {
                "content": "Okay, I will not place your order.",
```

```
                "contentType": "PlainText"
            }
        ]
    },
    "sampleUtterances": [
        "I would like to pick up flowers",
        "I would like to order some flowers"
    ],
    "slots": [
        {
            "slotType": "FlowerTypes",
            "name": "FlowerType",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "What type of flowers would you like to order?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 1,
            "slotTypeVersion": "$LATEST",
            "sampleUtterances": [
                "I would like to order {FlowerType}"
            ],
            "description": "The type of flowers to pick up"
        },
        {
            "slotType": "AMAZON.DATE",
            "name": "PickupDate",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "What day do you want the {FlowerType} to be picked
up?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 2,
```

```
            "description": "The date to pick up the flowers"
        },
        {
            "slotType": "AMAZON.TIME",
            "name": "PickupTime",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "Pick up the {FlowerType} at what time on
 {PickupDate}?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 3,
            "description": "The time to pick up the flowers"
        }
    ],
    "fulfillmentActivity": {
        "type": "ReturnIntent"
    },
    "description": "Intent to order a bouquet of flowers for pick up"
}
```

步骤 4：创建自动程序 (AWS CLI)

OrderFlowersBot 自动程序有一个目的，即您在上一步中创建的 OrderFlowers 目的。要运行本练习中的命令，您需要知道将在其中运行命令的区域。有关区域列表，请参阅 模型构建配额 。

> **Note**
>
> 以下 AWS CLI 示例是针对 Unix、Linux 和 macOS 进行格式化的。对于 Windows，请将 "\$LATEST" 改为 $LATEST。

创建 **OrderFlowersBot** 自动程序 (AWS CLI)

1. 创建名为 **OrderFlowersBot.json** 的文本文件。将 JSON 代码从 OrderFlowersBot.json 复制到该文本文件中。

2. 在中 AWS CLI，调用PutBot操作来创建机器人。此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws lex-models put-bot \
    --region region \
    --name OrderFlowersBot \
    --cli-input-json file://OrderFlowersBot.json
```

服务器响应如下。当您创建或更新自动程序时，status 字段设置为 BUILDING。这表示自动程序尚未就绪，不能使用。要确定自动程序何时可供使用，请使用下一步中的 GetBot 操作。

```
{
    "status": "BUILDING",
    "intents": [
        {
            "intentVersion": "$LATEST",
            "intentName": "OrderFlowers"
        }
    ],
    "name": "OrderFlowersBot",
    "locale": "en-US",
    "checksum": "checksum",
    "abortStatement": {
        "messages": [
            {
                "content": "Sorry, I'm not able to assist at this time",
                "contentType": "PlainText"
            }
        ]
    },
    "version": "$LATEST",
    "lastUpdatedDate": timestamp,
    "createdDate": timestamp,
    "clarificationPrompt": {
        "maxAttempts": 2,
        "messages": [
            {
                "content": "I didn't understand you, what would you like to do?",
                "contentType": "PlainText"
            }
        ]
    },
```

```
    "voiceId": "Salli",
    "childDirected": false,
    "idleSessionTTLInSeconds": 600,
    "processBehavior": "BUILD",
    "description": "Bot to order flowers on the behalf of a user"
}
```

3. 要确定您的新自动程序是否已可用，请运行以下命令。重复此命令，直到 `status` 字段返回 READY。此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws lex-models get-bot \
    --region region \
    --name OrderFlowersBot \
    --version-or-alias "\$LATEST"
```

在响应中查找 `status` 字段：

```
{
    "status": "READY",

    ...

}
```

下一个步骤

步骤 5：测试自动程序 (AWS CLI)

OrderFlowersBot.json

以下代码提供了构建 `OrderFlowers` Amazon Lex 机器人所需的 JSON 数据：

```
{
    "intents": [
        {
            "intentVersion": "$LATEST",
            "intentName": "OrderFlowers"
        }
    ],
```

```
    "name": "OrderFlowersBot",
    "locale": "en-US",
    "abortStatement": {
        "messages": [
            {
                "content": "Sorry, I'm not able to assist at this time",
                "contentType": "PlainText"
            }
        ]
    },
    "clarificationPrompt": {
        "maxAttempts": 2,
        "messages": [
            {
                "content": "I didn't understand you, what would you like to do?",
                "contentType": "PlainText"
            }
        ]
    },
    "voiceId": "Salli",
    "childDirected": false,
    "idleSessionTTLInSeconds": 600,
    "description": "Bot to order flowers on the behalf of a user"
}
```

步骤 5：测试自动程序 (AWS CLI)

可以使用基于文本的测试或基于语音的测试来测试自动程序。

主题

- 使用文本输入测试自动程序 (AWS CLI)
- 使用语音输入测试自动程序 (AWS CLI)

使用文本输入测试自动程序 (AWS CLI)

要验证自动程序能否正确处理文本输入，请使用 PostText 操作。要运行本练习中的命令，您需要知道将在其中运行命令的区域。有关区域列表，请参阅运行时服务配额。

> **Note**
>
> 以下 AWS CLI 示例是针对 Unix、Linux 和 macOS 进行格式化的。对于 Windows，请将 "\$LATEST" 更改为 $LATEST 并将每行末尾的反斜杠 (\) 继续符替换为脱字号 (^)。

使用文本测试自动程序 (AWS CLI)

1. 在中 AWS CLI，开始与OrderFlowersBot机器人对话。此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

   ```
   aws lex-runtime post-text \
       --region region \
       --bot-name OrderFlowersBot \
       --bot-alias "\$LATEST" \
       --user-id UserOne \
       --input-text "i would like to order flowers"
   ```

   Amazon Lex 识别用户意图并通过返回以下响应开始对话：

   ```
   {
       "slotToElicit": "FlowerType",
       "slots": {
           "PickupDate": null,
           "PickupTime": null,
           "FlowerType": null
       },
       "dialogState": "ElicitSlot",
       "message": "What type of flowers would you like to order?",
       "intentName": "OrderFlowers"
   }
   ```

2. 运行以下命令以完成与自动程序的对话。

   ```
   aws lex-runtime post-text \
       --region region \
       --bot-name OrderFlowersBot \
       --bot-alias "\$LATEST" \
       --user-id UserOne \
       --input-text "roses"
   ```

```
aws lex-runtime post-text  \
    --region region \
    --bot-name OrderFlowersBot \
    --bot-alias "\$LATEST" \
    --user-id UserOne \
    --input-text "tuesday"
```

```
aws lex-runtime post-text  \
    --region region \
    --bot-name OrderFlowersBot --bot-alias "\$LATEST" \
    --user-id UserOne \
    --input-text "10:00 a.m."
```

```
aws lex-runtime post-text  \
    --region region \
    --bot-name OrderFlowersBot \
    --bot-alias "\$LATEST" \
    --user-id UserOne \
    --input-text "yes"
```

在您确认订单后，Amazon Lex 会发送一个履行响应以完成对话：

```
{
    "slots": {
        "PickupDate": "2017-05-16",
        "PickupTime": "10:00",
        "FlowerType": "roses"
    },
    "dialogState": "ReadyForFulfillment",
    "intentName": "OrderFlowers"
}
```

下一个步骤

使用语音输入测试自动程序 (AWS CLI)

使用语音输入测试自动程序 (AWS CLI)

要使用音频文件测试自动程序，请使用 PostContent 操作。您可以使用 Amazon Polly text-to-speech 操作生成音频文件。

要运行本练习中的命令，您需要知道将在其中运行 Amazon Lex 和 Amazon Polly 命令的区域。有关 Amazon Lex 区域的列表，请参阅运行时服务配额。有关 Amazon Polly 区域的列表，请参阅《Amazon Web Services 一般参考》中的 AWS 区域和端点。

> **Note**
>
> 以下 AWS CLI 示例是针对 Unix、Linux 和 macOS 进行格式化的。对于 Windows，请将 "\$LATEST" 更改为 $LATEST 并将每行末尾的反斜杠 (\) 继续符替换为脱字号 (^)。

使用语音输入测试自动程序 (AWS CLI)

1. 在中 AWS CLI，使用 Amazon Polly 创建音频文件。此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws polly synthesize-speech \
    --region region \
    --output-format pcm \
    --text "i would like to order flowers" \
    --voice-id "Salli" \
    IntentSpeech.mpg
```

2. 要将音频文件发送到 Amazon Lex，请运行以下命令。Amazon Lex 将来自响应的音频保存在指定的输出文件中。

```
aws lex-runtime post-content \
    --region region \
    --bot-name OrderFlowersBot \
    --bot-alias "\$LATEST" \
    --user-id UserOne \
    --content-type "audio/l16; rate=16000; channels=1" \
    --input-stream IntentSpeech.mpg \
    IntentOutputSpeech.mpg
```

Amazon Lex 用对第一个插槽的请求进行响应。它将音频响应保存在指定的输出文件中。

```
{
    "contentType": "audio/mpeg",
    "slotToElicit": "FlowerType",
    "dialogState": "ElicitSlot",
    "intentName": "OrderFlowers",
    "inputTranscript": "i would like to order some flowers",
    "slots": {
        "PickupDate": null,
        "PickupTime": null,
        "FlowerType": null
    },
    "message": "What type of flowers would you like to order?"
}
```

3. 要订购玫瑰，请创建以下音频文件并将其发送给 Amazon Lex：

```
aws polly synthesize-speech \
    --region region \
    --output-format pcm \
    --text "roses" \
    --voice-id "Salli" \
    FlowerTypeSpeech.mpg
```

```
aws lex-runtime post-content \
    --region region \
    --bot-name OrderFlowersBot \
    --bot-alias "\$LATEST" \
    --user-id UserOne \
    --content-type "audio/l16; rate=16000; channels=1" \
    --input-stream FlowerTypeSpeech.mpg \
    FlowerTypeOutputSpeech.mpg
```

4. 要设置送货日期，请创建以下音频文件并将其发送给 Amazon Lex：

```
aws polly synthesize-speech \
    --region region \
    --output-format pcm \
    --text "tuesday" \
    --voice-id "Salli" \
    DateSpeech.mpg
```

```
aws lex-runtime post-content \
    --region region \
    --bot-name OrderFlowersBot \
    --bot-alias "\$LATEST" \
    --user-id UserOne \
    --content-type "audio/l16; rate=16000; channels=1" \
    --input-stream DateSpeech.mpg \
    DateOutputSpeech.mpg
```

5. 要设置送货时间，请创建以下音频文件并将其发送给 Amazon Lex：

```
aws polly synthesize-speech \
    --region region \
    --output-format pcm \
    --text "10:00 a.m." \
    --voice-id "Salli" \
    TimeSpeech.mpg
```

```
aws lex-runtime post-content \
    --region region \
    --bot-name OrderFlowersBot \
    --bot-alias "\$LATEST" \
    --user-id UserOne \
    --content-type "audio/l16; rate=16000; channels=1" \
    --input-stream TimeSpeech.mpg \
    TimeOutputSpeech.mpg
```

6. 要确认送货，请创建以下音频文件并将其发送给 Amazon Lex：

```
aws polly synthesize-speech \
    --region region \
    --output-format pcm \
    --text "yes" \
    --voice-id "Salli" \
    ConfirmSpeech.mpg
```

```
aws lex-runtime post-content \
    --region region \
    --bot-name OrderFlowersBot \
    --bot-alias "\$LATEST" \
    --user-id UserOne \
```

```
    --content-type "audio/l16; rate=16000; channels=1" \
    --input-stream ConfirmSpeech.mpg \
    ConfirmOutputSpeech.mpg
```

确认送货后，Amazon Lex 会发送一个响应来确认履行意图：

```
{
    "contentType": "text/plain;charset=utf-8",
    "dialogState": "ReadyForFulfillment",
    "intentName": "OrderFlowers",
    "inputTranscript": "yes",
    "slots": {
        "PickupDate": "2017-05-16",
        "PickupTime": "10:00",
        "FlowerType": "roses"
    }
}
```

下一个步骤

练习 2：添加新表达 (AWS CLI)

练习 2：添加新表达 (AWS CLI)

为完善 Amazon Lex 用来识别用户请求的机器学习模型，我们再向机器人添加一个示例语句。

添加新表达的过程分为四步。

1. 使用 GetIntent 操作从 Amazon Lex 获取意图。
2. 更新目的。
3. 使用 PutIntent 操作将更新后的意图发送回 Amazon Lex。
4. 使用 GetBot 和 PutBot 操作重建使用该目的的所有自动程序。

要运行本练习中的命令，您需要知道将在其中运行命令的区域。有关区域列表，请参阅 模型构建配额。

来自 GetIntent 操作的响应包含一个名为 checksum 的字段，该字段标识目的的一个特定修订版。在使用 PutIntent 操作更新目的时必须提供校验和值。如果不提供，您会收到以下错误消息：

```
        An error occurred (PreconditionFailedException) when calling
        the PutIntent operation: Intent intent name already exists.
        If you are trying to update intent name you must specify the
        checksum.
```

> **Note**
>
> 以下 AWS CLI 示例是针对 Unix、Linux 和 macOS 进行格式化的。对于 Windows，请将 "\$LATEST" 更改为 $LATEST 并将每行末尾的反斜杠 (\) 继续符替换为脱字号 (^)。

更新 **OrderFlowers** 目的 (AWS CLI)

1. 在《》中 AWS CLI，从 Amazon Lex 那里获取意图。Amazon Lex 将输出发送到一个名为 **OrderFlowers-V2.json.** 的文件

   ```
   aws lex-models get-intent \
       --region region \
       --name OrderFlowers \
       --intent-version "\$LATEST" > OrderFlowers-V2.json
   ```

2. 在文本编辑器中打开 **OrderFlowers-V2.json**。

   1. 找到并删除 `createdDate`、`lastUpdatedDate` 和 `version` 字段。
   2. 将下面的内容添加到 `sampleUtterances` 字段中：

      ```
      I want to order flowers
      ```

   3. 保存该文件。

3. 使用以下命令将更新后的意图发送给 Amazon Lex：

   ```
   aws lex-models put-intent  \
       --region region \
       --name OrderFlowers \
       --cli-input-json file://OrderFlowers-V2.json
   ```

   Amazon Lex 会发送以下响应：

   ```
   {
   ```

```
    "confirmationPrompt": {
        "maxAttempts": 2,
        "messages": [
            {
                "content": "Okay, your {FlowerType} will be ready for pickup by
{PickupTime} on {PickupDate}.  Does this sound okay?",
                "contentType": "PlainText"
            }
        ]
    },
    "name": "OrderFlowers",
    "checksum": "checksum",
    "version": "$LATEST",
    "rejectionStatement": {
        "messages": [
            {
                "content": "Okay, I will not place your order.",
                "contentType": "PlainText"
            }
        ]
    },
    "createdDate": timestamp,
    "lastUpdatedDate": timestamp,
    "sampleUtterances": [
        "I would like to pick up flowers",
        "I would like to order some flowers",
        "I want to order flowers"
    ],
    "slots": [
        {
            "slotType": "AMAZON.TIME",
            "name": "PickupTime",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "Pick up the {FlowerType} at what time on
{PickupDate}?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 3,
```

```
            "description": "The time to pick up the flowers"
        },
        {
            "slotType": "FlowerTypes",
            "name": "FlowerType",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "What type of flowers would you like to
order?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 1,
            "slotTypeVersion": "$LATEST",
            "sampleUtterances": [
                "I would like to order {FlowerType}"
            ],
            "description": "The type of flowers to pick up"
        },
        {
            "slotType": "AMAZON.DATE",
            "name": "PickupDate",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "What day do you want the {FlowerType} to be
picked up?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 2,
            "description": "The date to pick up the flowers"
        }
    ],
    "fulfillmentActivity": {
        "type": "ReturnIntent"
    },
```

```
    "description": "Intent to order a bouquet of flowers for pick up"
}
```

现在您已更新了目的，接下来要重建使用它的所有自动程序。

重建 **OrderFlowersBot** 自动程序 (AWS CLI)

1. 在中 AWS CLI，获取OrderFlowersBot机器人的定义并使用以下命令将其保存到文件中：

```
aws lex-models get-bot \
    --region region \
    --name OrderFlowersBot \
    --version-or-alias "\$LATEST" > OrderFlowersBot-V2.json
```

2. 在文本编辑器中，打开 **OrderFlowersBot-V2.json**。删除 createdDate、lastUpdatedDate、status 和 version 字段。
3. 在文本编辑器中，将下面一行添加到自动程序定义中。

```
"processBehavior": "BUILD",
```

4. 在中 AWS CLI，通过运行以下命令来构建机器人的新版本：

```
aws lex-models put-bot \
    --region region \
    --name OrderFlowersBot \
    --cli-input-json file://OrderFlowersBot-V2.json
```

服务器的响应如下：

```
{
    "status": "BUILDING",
    "intents": [
        {
            "intentVersion": "$LATEST",
            "intentName": "OrderFlowers"
        }
    ],
    "name": "OrderFlowersBot",
    "locale": "en-US",
    "checksum": "checksum",
```

```
    "abortStatement": {
        "messages": [
            {
                "content": "Sorry, I'm not able to assist at this time",
                "contentType": "PlainText"
            }
        ]
    },
    "version": "$LATEST",
    "lastUpdatedDate": timestamp,
    "createdDate": timestamp
    "clarificationPrompt": {
        "maxAttempts": 2,
        "messages": [
            {
                "content": "I didn't understand you, what would you like to do?",
                "contentType": "PlainText"
            }
        ]
    },
    "voiceId": "Salli",
    "childDirected": false,
    "idleSessionTTLInSeconds": 600,
    "description": "Bot to order flowers on the behalf of a user"
}
```

下一个步骤

练习 3：添加 Lambda 函数 (AWS CLI)

练习 3：添加 Lambda 函数 (AWS CLI)

向机器人添加用于验证用户输入和履行用户意图的 Lambda 函数。

添加 Lambda 表达式的过程分为五步。

1. 使用 Lambda AddPermission函数启用调用 Lambda 调用操作的OrderFlowers意图。
2. 使用 GetIntent 操作从 Amazon Lex 获取意图。
3. 更新意图以添加 Lambda 函数。
4. 使用 PutIntent 操作将更新后的意图发送回 Amazon Lex。

5. 使用 GetBot 和 PutBot 操作重建使用该目的的所有自动程序。

要运行本练习中的命令，您需要知道将在其中运行命令的区域。有关区域列表，请参阅 模型构建配额 。

如果您在添加 InvokeFunction 权限前向意图添加 Lambda 函数，会得到以下错误消息：

```
An error occurred (BadRequestException) when calling the
PutIntent operation: Lex is unable to access the Lambda
function Lambda function ARN in the context of intent
intent ARN.  Please check the resource-based policy on
the function.
```

来自 GetIntent 操作的响应包含一个名为 checksum 的字段，该字段标识目的的一个特定修订版。在使用 PutIntent 操作更新目的时必须提供校验和值。如果不提供，您会收到以下错误消息：

```
An error occurred (PreconditionFailedException) when calling
the PutIntent operation: Intent intent name already exists.
If you are trying to update intent name you must specify the
checksum.
```

本练习使用 练习 1：使用蓝图创建 Amazon Lex 机器人（控制台） 中的 Lambda 函数。有关创建 Lambda 函数的说明，请参阅步骤 3：创建 Lambda 函数（控制台）。

> **Note**
>
> 以下 AWS CLI 示例是针对 Unix、Linux 和 macOS 进行格式化的。对于 Windows，请将 "\$LATEST" 改为 $LATEST。

向意图添加 Lambda 函数

1. 在中 AWS CLI，添加OrderFlowers意图InvokeFunction权限：

```
aws lambda add-permission \
    --region region \
```

```
    --function-name OrderFlowersCodeHook \
    --statement-id LexGettingStarted-OrderFlowersBot \
    --action lambda:InvokeFunction \
    --principal lex.amazonaws.com \
    --source-arn "arn:aws:lex:region:account ID:intent:OrderFlowers:*"
    --source-account account ID
```

Lambda 会发送以下响应：

```
{
    "Statement": "{\"Sid\":\"LexGettingStarted-OrderFlowersBot\",
      \"Resource\":\"arn:aws:lambda:region:account ID:function:OrderFlowersCodeHook
\",
      \"Effect\":\"Allow\",
      \"Principal\":{\"Service\":\"lex.amazonaws.com\"},
      \"Action\":[\"lambda:InvokeFunction\"],
      \"Condition\":{\"StringEquals\":
        {\"AWS:SourceAccount\": \"account ID\"},
        {\"AWS:SourceArn\":
          \"arn:aws:lex:region:account ID:intent:OrderFlowers:*\"}}}"
}
```

2. 从 Amazon Lex 获取意图。Amazon Lex 将输出发送到一个名为 **OrderFlowers-V3.json** 的文件。

```
aws lex-models get-intent \
    --region region \
    --name OrderFlowers \
    --intent-version "\$LATEST" > OrderFlowers-V3.json
```

3. 在文本编辑器中，打开 **OrderFlowers-V3.json**。

   1. 找到并删除 `createdDate`、`lastUpdatedDate` 和 `version` 字段。
   2. 更新 `fulfillmentActivity` 字段：

```
   "fulfillmentActivity": {
       "type": "CodeHook",
       "codeHook": {
           "uri": "arn:aws:lambda:region:account
ID:function:OrderFlowersCodeHook",
           "messageVersion": "1.0"
       }
```

```
    }
```

3. 保存该文件。

4. 在中 AWS CLI，将更新的意图发送给 Amazon Lex：

```
aws lex-models put-intent \
    --region region \
    --name OrderFlowers \
    --cli-input-json file://OrderFlowers-V3.json
```

现在您已更新了目的，接下来要重建自动程序。

重建 **OrderFlowersBot** 自动程序

1. 在中 AWS CLI，获取OrderFlowersBot机器人的定义并将其保存到文件中：

```
aws lex-models get-bot \
    --region region \
    --name OrderFlowersBot \
    --version-or-alias "\$LATEST" > OrderFlowersBot-V3.json
```

2. 在文本编辑器中，打开 **OrderFlowersBot-V3.json**。删除 `createdDate`、`lastUpdatedDate`、`status` 和 `version` 字段。

3. 在文本编辑器中，将下面一行添加到自动程序定义中：

```
"processBehavior": "BUILD",
```

4. 在中 AWS CLI，构建该机器人的新修订版：

```
aws lex-models put-bot \
    --region region \
    --name OrderFlowersBot \
    --cli-input-json file://OrderFlowersBot-V3.json
```

服务器的响应如下：

```
{
    "status": "READY",
    "intents": [
        {
```

```
            "intentVersion": "$LATEST",
            "intentName": "OrderFlowers"
        }
    ],
    "name": "OrderFlowersBot",
    "locale": "en-US",
    "checksum": "checksum",
    "abortStatement": {
        "messages": [
            {
                "content": "Sorry, I'm not able to assist at this time",
                "contentType": "PlainText"
            }
        ]
    },
    "version": "$LATEST",
    "lastUpdatedDate": timestamp,
    "createdDate": timestamp,
    "clarificationPrompt": {
        "maxAttempts": 2,
        "messages": [
            {
                "content": "I didn't understand you, what would you like to do?",
                "contentType": "PlainText"
            }
        ]
    },
    "voiceId": "Salli",
    "childDirected": false,
    "idleSessionTTLInSeconds": 600,
    "description": "Bot to order flowers on the behalf of a user"
}
```

下一个步骤

[练习 4：发布版本 (AWS CLI)](#)

练习 4：发布版本 (AWS CLI)

现在，为您在练习 1 中创建的自动程序创建一个版本。版本 是自动程序的快照。版本创建后便无法更改。您可以更新的唯一自动程序版本是 $LATEST 版本。有关版本的更多信息，请参阅[版本控制和别名](#)。

在发布自动程序版本前，必须先发布它所使用的目的。同样，还必须发布这些目的引用的槽类型。一般而言，发布自动程序版本需要执行以下操作：

1. 使用 CreateSlotTypeVersion 操作发布一个槽类型版本。
2. 使用 CreateIntentVersion 操作发布一个目的版本。
3. 使用 CreateBotVersion 操作发布一个自动程序版本。

要运行本练习中的命令，您需要知道将在其中运行命令的区域。有关区域列表，请参阅 模型构建配额 。

主题

- 步骤 1：发布槽类型 (AWS CLI)
- 步骤 2：发布目的 (AWS CLI)
- 步骤 3：发布自动程序 (AWS CLI)

步骤 1：发布槽类型 (AWS CLI)

在发布使用某一槽类型的任意目的的版本之前，必须发布该槽类型的一个版本。在本例中，您将发布 `FlowerTypes` 槽类型。

> **Note**
>
> 以下 AWS CLI 示例是针对 Unix、Linux 和 macOS 进行格式化的。对于 Windows，请将 "\$LATEST" 更改为 $LATEST 并将每行末尾的反斜杠 (\) 继续符替换为脱字号 (^)。

发布槽类型 (AWS CLI)

1. 在中 AWS CLI，获取最新版本的插槽类型：

```
aws lex-models get-slot-type \
    --region region \
    --name FlowerTypes \
    --slot-type-version "\$LATEST"
```

来自 Amazon Lex 的响应如下。请记录 $LATEST 版本最新修订版的校验和。

```
{
    "enumerationValues": [
        {
            "value": "tulips"
        },
        {
            "value": "lilies"
        },
        {
            "value": "roses"
        }
    ],
    "name": "FlowerTypes",
    "checksum": "checksum",
    "version": "$LATEST",
    "lastUpdatedDate": timestamp,
    "createdDate": timestamp,
    "description": "Types of flowers to pick up"
}
```

2. 发布一个槽类型版本。请使用在上一步中记录的校验和。

```
aws lex-models create-slot-type-version \
    --region region \
    --name FlowerTypes \
    --checksum "checksum"
```

来自 Amazon Lex 的响应如下。请记录版本号以供下一步使用。

```
{
    "version": "1",
    "enumerationValues": [
        {
            "value": "tulips"
        },
        {
            "value": "lilies"
        },
        {
            "value": "roses"
        }
    ],
```

```
    "name": "FlowerTypes",
    "createdDate": timestamp,
    "lastUpdatedDate": timestamp,
    "description": "Types of flowers to pick up"
}
```

下一个步骤

步骤 2：发布目的 (AWS CLI)

步骤 2：发布目的 (AWS CLI)

在发布某个目的之前，必须先发布该目的引用的所有槽类型。槽类型必须是编号版本，而不是 $LATEST 版本。

首先，更新 OrderFlowers 目的以使用您在上一步中发布的 FlowerTypes 槽类型版本。然后发布 OrderFlowers 目的的新版本。

> **Note**
>
> 以下 AWS CLI 示例是针对 Unix、Linux 和 macOS 进行格式化的。对于 Windows，请将 "\$LATEST" 更改为 $LATEST 并将每行末尾的反斜杠 (\) 继续符替换为脱字号 (^)。

发布目的版本 (AWS CLI)

1. 在中 AWS CLI，获取 OrderFlowers Intent 的$LATEST版本并将其保存到文件中：

```
aws lex-models get-intent \
    --region region \
    --name OrderFlowers \
    --intent-version "\$LATEST" > OrderFlowers_V4.json
```

2. 在文本编辑器中，打开 **OrderFlowers_V4.json**文件。删除 createdDate、lastUpdatedDate 和 version 字段。找到 FlowerTypes 槽类型并将版本更改为您在上一步中记录的版本号。下面的 **OrderFlowers_V4.json** 文件片段显示了更改的位置：

```
        {
            "slotType": "FlowerTypes",
            "name": "FlowerType",
```

```
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "What type of flowers?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 1,
            "slotTypeVersion": "version",
            "sampleUtterances": []
        },
```

3. 在中 AWS CLI，保存意图的修订版：

```
aws lex-models put-intent \
    --name OrderFlowers \
    --cli-input-json file://OrderFlowers_V4.json
```

4. 获取最新目的修订版的校验和：

```
aws lex-models get-intent \
    --region region \
    --name OrderFlowers \
    --intent-version "\$LATEST" > OrderFlowers_V4a.json
```

以下响应片段显示了目的的校验和。请记下此校验和供下一步使用。

```
    "name": "OrderFlowers",
    "checksum": "checksum",
    "version": "$LATEST",
```

5. 发布新目的版本：

```
aws lex-models create-intent-version \
    --region region \
    --name OrderFlowers \
    --checksum "checksum"
```

以下响应片段显示了新目的版本。请记录版本号以供下一步使用。

```
    "name": "OrderFlowers",
    "checksum": "checksum",
    "version": "version",
```

下一个步骤

步骤 3：发布自动程序 (AWS CLI)

步骤 3：发布自动程序 (AWS CLI)

发布完您的自动程序所使用的所有槽类型和目的后，就可以发布自动程序了。

更新 OrderFlowersBot 自动程序以使用您在上一步中更新的 OrderFlowers 目的。然后发布 OrderFlowersBot 自动程序的新版本。

> **Note**
>
> 以下 AWS CLI 示例是针对 Unix、Linux 和 macOS 进行格式化的。对于 Windows，请将 "\$LATEST" 更改为 $LATEST 并将每行末尾的反斜杠 (\) 继续符替换为脱字号 (^)。

发布自动程序版本 (AWS CLI)

1. 在中 AWS CLI，获取OrderFlowersBot机器人的$LATEST版本并将其保存到文件中：

```
aws lex-models get-bot \
    --region region \
    --name OrderFlowersBot \
    --version-or-alias "\$LATEST" > OrderFlowersBot_V4.json
```

2. 在文本编辑器中，打开 **OrderFlowersBot_V4.json**文件。删除 createdDate、lastUpdatedDate、status 和 version 字段。找到 OrderFlowers 目的并将版本更改为您在上一步中记录的版本号。下面的 **OrderFlowersBot_V4.json** 文件片段显示了更改的位置。

```
    "intents": [
        {
            "intentVersion": "version",
```

```
            "intentName": "OrderFlowers"
        }
```

3. 在中 AWS CLI，保存机器人的新版本。记下调用 put-bot 所返回的版本号。

```
aws lex-models put-bot \
    --name OrderFlowersBot \
    --cli-input-json file://OrderFlowersBot_V4.json
```

4. 获取最新自动程序修订版的校验和。使用步骤 3 中返回的版本号。

```
aws lex-models get-bot \
    --region region \
    --version-or-alias version \
    --name OrderFlowersBot > OrderFlowersBot_V4a.json
```

以下响应片段显示了自动程序的校验和。请记下此校验和供下一步使用。

```
    "name": "OrderFlowersBot",
    "locale": "en-US",
    "checksum": "checksum",
```

5. 发布机器人的新版本：

```
aws lex-models create-bot-version \
    --region region \
    --name OrderFlowersBot \
    --checksum "checksum"
```

以下响应片段显示了自动程序的新版本。

```
    "checksum": "checksum",
    "abortStatement": {
        ...
    },
    "version": "1",
    "lastUpdatedDate": timestamp,
```

下一个步骤

[练习 5：创建别名 (AWS CLI)]

练习 5：创建别名 (AWS CLI)

别名是指向自动程序特定版本的指针。利用别名，您可以轻松更新您的客户端应用程序正在使用的版本。有关详细信息，请参阅版本控制和别名。要运行本练习中的命令，您需要知道将在其中运行命令的区域。有关区域列表，请参阅 模型构建配额 。

创建别名 (AWS CLI)

1. 在中 AWS CLI，获取你在中创建的`OrderFlowersBot`机器人的版本练习 4：发布版本 (AWS CLI)。

   ```
   aws lex-models get-bot \
       --region region \
       --name OrderFlowersBot \
       --version-or-alias version > OrderFlowersBot_V5.json
   ```

2. 在文本编辑器中，打开 **`OrderFlowersBot_v5.json`**。找到并记录版本号。
3. 在中 AWS CLI，创建机器人别名：

   ```
   aws lex-models put-bot-alias  \
       --region region \
       --name PROD \
       --bot-name OrderFlowersBot \
       --bot-version version
   ```

   服务器响应如下：

   ```
   {
       "name": "PROD",
       "createdDate": timestamp,
       "checksum": "checksum",
       "lastUpdatedDate": timestamp,
       "botName": "OrderFlowersBot",
       "botVersion": "1"
   }}
   ```

下一个步骤

步骤 6：清理 (AWS CLI)

步骤 6：清理 (AWS CLI)

删除您创建的资源并清理您的账户。

您只能删除未使用的资源。一般来说，您应该按照以下顺序删除资源。

1. 删除别名以释放自动程序资源。
2. 删除自动程序以释放目的资源。
3. 删除目的以释放槽类型资源。
4. 删除槽类型。

要运行本练习中的命令，您需要知道将在其中运行命令的区域。有关区域列表，请参阅 模型构建配额 。

清理您的账户 (AWS CLI)

1. 在 AWS CLI 命令行中，删除别名：

```
aws lex-models delete-bot-alias \
    --region region \
    --name PROD \
    --bot-name OrderFlowersBot
```

2. 在 AWS CLI 命令行中，删除机器人：

```
aws lex-models delete-bot \
    --region region \
    --name OrderFlowersBot
```

3. 在 AWS CLI 命令行中，删除意图：

```
aws lex-models delete-intent \
    --region region \
    --name OrderFlowers
```

4. 在 AWS CLI 命令行中，删除插槽类型：

```
aws lex-models delete-slot-type \
    --region region \
    --name FlowerTypes
```

您已经删除了创建的所有资源并清除了账户。

版本控制和别名

Amazon Lex 支持发布机器人、意图和插槽类型的版本，以便您可以控制客户端应用程序使用的实现。版本 是您工作的带编号快照，您可以发布版本以用于您的工作流的不同阶段，如开发、测试部署和生产。

Amazon Lex 机器人也支持别名。别名 是指向自动程序特定版本的指针。利用别名，您可以轻松更新您的客户端应用程序正在使用的版本。例如，您可以将别名指向您机器人的版本 1。当您准备更新机器人时，您可以发布版本 2，然后更改别名以指向新版本。由于您的应用程序使用的是别名而不是特定版本，因此您的所有客户端无需进行更新即可获得新功能。

主题

版本控制

当您对 Amazon Lex 资源进行版本控制时，您创建一个资源快照，从而能够以创建该版本时的资源状态使用该资源。版本创建后，在您继续处理应用程序时它将保持不变。

$LATEST 版本

当您创建 Amazon Lex 机器人、意图或插槽类型时，只有一个版本，即 $LATEST 版本。

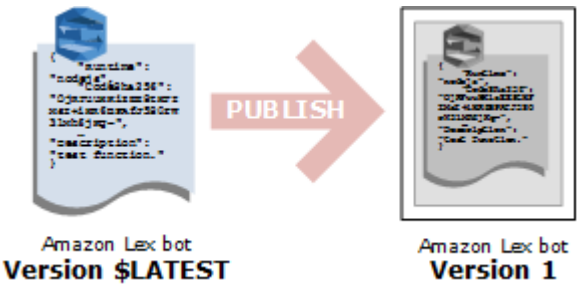
$LATEST 是您的资源的工作副本。您只能更新 $LATEST 版本，直到您发布第一个版本之前，$LATEST 是您所拥有的唯一资源版本。

只有资源的 $LATEST 版本可以使用另一个资源的 $LATEST 版本。例如，自动程序的 $LATEST 版本可以使用目的的 $LATEST 版本，目的的 $LATEST 版本可以使用槽类型的 $LATEST 版本。

$LATEST 版本的机器人只应用于手动测试。Amazon Lex 限制可对 $LATEST 版本的机器人进行的运行时请求的数量。

发布 Amazon Lex 资源版本

当您发布资源时，Amazon Lex 将创建 $LATEST 版本的副本，并将其保存为带编号的版本。无法更改已发布版本。



您可以使用 Amazon Lex 控制台或 CreateBotVersion 操作创建和发布版本。有关示例，请参阅练习 3：发布版本和创建别名。

当您修改 $LATEST 版本的资源时，可以发布新版本以做出可用于您的客户端应用程序的更改。每当您发布版本时，Amazon Lex 都会复制 $LATEST 版本以创建新版本并将版本号递增 1。版本号绝不会重复使用。例如，如果您删除版本编号为 10 的资源，然后再重新创建该资源，则 Amazon Lex 分配给它的下一个版本号是版本 11。

在发布自动程序前，必须先将它指向它所使用的任意目的的一个编号版本。如果您尝试发布使用 $LATEST 版本目的的自动程序新版本，Amazon Lex 会返回 HTTP 400 错误请求异常。在发布编号版本的目的之前，必须先将目的指向它所使用的任意槽类型的一个编号版本。否则您会收到 HTTP 400 错误请求异常。

> **Note**
>
> 仅在上次发布的版本与 $LATEST 版本不同时，Amazon Lex 才会发布新版本。如果您尝试发布 $LATEST 版本而不对它进行修改，则 Amazon Lex 不会创建或发布新版本。

更新 Amazon Lex 资源

您只能更新 Amazon Lex 机器人、意图或插槽类型的 $LATEST 版本。无法更改已发布的版本。您在控制台中或者使用 CreateBotVersion、CreateIntentVersion 或 CreateSlotTypeVersion 操作更新资源后，随时可以发布新版本。

删除 Amazon Lex 资源或版本

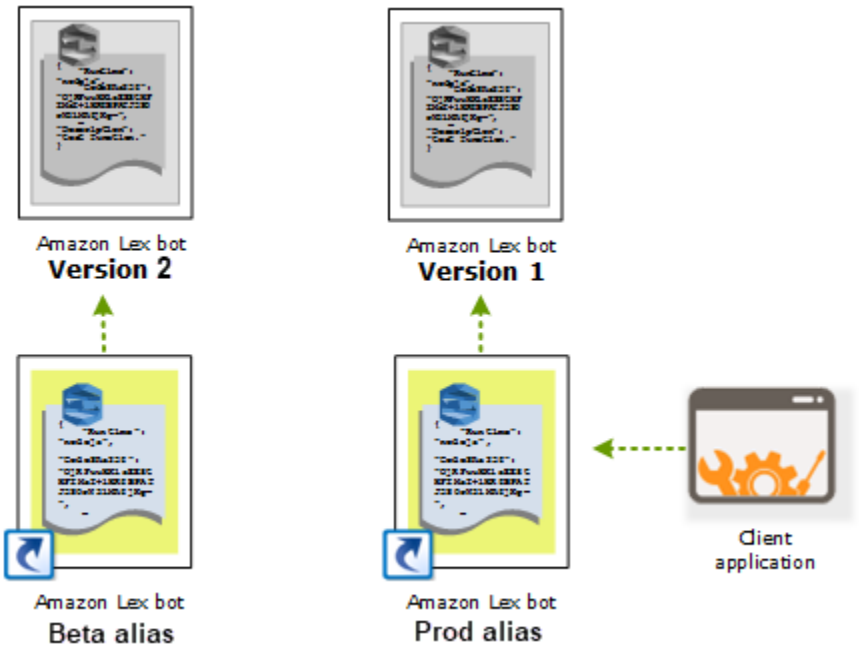
Amazon Lex 支持使用控制台或以下 API 操作之一删除资源或版本：

- DeleteBot
- DeleteBotVersion
- DeleteBotAlias
- DeleteBotChannelAssociation
- DeleteIntent
- DeleteIntentVersion
- DeleteSlotType
- DeleteSlotTypeVersion

别名

别名是指向 Amazon Lex 机器人特定版本的指针。使用别名可允许客户端应用程序使用特定版本的自动程序，而应用程序无需跟踪它是哪个版本。

以下示例显示 Amazon Lex 机器人的两个版本，版本 1 和版本 2。这两个自动程序版本各有一个关联的别名，分别为 BETA 和 PROD。客户端应用程序使用 PROD 别名来访问自动程序。



当您创建另一个版本的自动程序时，您可以使用控制台或 PutBot 操作更新别名以指向新版本的自动程序。当您更改别名时，您的所有客户端应用程序都将使用新版本。如果新版本出了问题，您只需通过更改别名以指向之前的版本，即可回滚到该版本。

> **Note**
>
> 虽然您可以在控制台中测试自动程序的 $LATEST 版本，但我们建议，在您将自动程序与您的客户端应用程序集成时，首先发布一个版本，然后创建一个指向该版本的别名。本部分中说

明了在您的客户端应用程序中使用别名的原因。当您更新别名时，Amazon Lex 会等到所有当前会话的会话超时过期之后再开始使用新版本。有关会话超时的更多信息，请参阅the section called “设置超时会话”

使用 Lambda 函数

您可以创建用作您的 Amazon Lex 机器人的代码挂钩的 AWS Lambda 函数。您可以在您的意图配置中标识 Lambda 函数以执行初始化和验证或履行操作，或者同时执行两者。

建议您使用 Lambda 函数作为机器人的代码挂钩。在没有 Lambda 函数的情况下，机器人会将意图信息返回给客户端应用程序以用于履行。

主题

Lambda 函数输入事件和响应格式

本节介绍 Amazon Lex 为 Lambda 函数提供的事件数据的结构。您可以根据此信息分析 Lambda 代码中的输入。本节还将说明 Amazon Lex 期望 Lambda 函数返回的响应的格式。

主题

输入事件格式

以下是传递到 Lambda 函数的 Amazon Lex 事件的一般格式。您在编写 Lambda 函数时可以参考此信息。

> **Note**
>
> 输入格式可以更改，但不必对 `messageVersion` 做出相应更改。您的代码不应在有新字段时引发错误。

```
{
  "currentIntent": {
    "name": "intent-name",
    "nluIntentConfidenceScore": score,
```

```
    "slots": {
      "slot name": "value",
      "slot name": "value"
    },
    "slotDetails": {
      "slot name": {
        "resolutions" : [
          { "value": "resolved value" },
          { "value": "resolved value" }
        ],
        "originalValue": "original text"
      },
      "slot name": {
        "resolutions" : [
          { "value": "resolved value" },
          { "value": "resolved value" }
        ],
        "originalValue": "original text"
      }
    },
    "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if
configured)"
  },
  "alternativeIntents": [
    {
      "name": "intent-name",
      "nluIntentConfidenceScore": score,
      "slots": {
        "slot name": "value",
        "slot name": "value"
      },
      "slotDetails": {
        "slot name": {
          "resolutions" : [
            { "value": "resolved value" },
            { "value": "resolved value" }
          ],
          "originalValue": "original text"
        },
        "slot name": {
          "resolutions" : [
            { "value": "resolved value" },
            { "value": "resolved value" }
          ],
```

```
          "originalValue": "original text"
        }
      },
      "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if
configured)"
    }
 ],
 "bot": {
   "name": "bot name",
   "alias": "bot alias",
   "version": "bot version"
 },
 "userId": "User ID specified in the POST request to Amazon Lex.",
 "inputTranscript": "Text used to process the request",
 "invocationSource": "FulfillmentCodeHook or DialogCodeHook",
 "outputDialogMode": "Text or Voice, based on ContentType request header in runtime
API request",
 "messageVersion": "1.0",
 "sessionAttributes": {
    "key": "value",
    "key": "value"
 },
 "requestAttributes": {
    "key": "value",
    "key": "value"
 },
 "recentIntentSummaryView": [
   {
       "intentName": "Name",
       "checkpointLabel": Label,
       "slots": {
         "slot name": "value",
         "slot name": "value"
       },
       "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if
configured)",
       "dialogActionType": "ElicitIntent, ElicitSlot, ConfirmIntent, Delegate, or
Close",
       "fulfillmentState": "Fulfilled or Failed",
       "slotToElicit": "Next slot to elicit"
   }
 ],
  "sentimentResponse": {
     "sentimentLabel": "sentiment",
```

```
      "sentimentScore": "score"
   },
   "kendraResponse": {
       Complete query response from Amazon Kendra
   },
   "activeContexts": [
        {
            "timeToLive": {
                "timeToLiveInSeconds": seconds,
                "turnsToLive": turns
            },
            "name": "name",
            "parameters": {
                "key name": "value"
            }
        }
    ]
}
```

请注意以下关于事件字段的额外信息：

- currentIntent – 提供目的 name、slots、slotDetails 和 confirmationStatus 字段。

  nluIntentConfidenceScore 是 Amazon Lex 对当前意图最符合用户当前意图的信心程度。

  slots 是从为意图配置的插槽名称到 Amazon Lex 在用户会话中识别出的插槽值的映射。槽值保持为空，直到用户提供一个值。

  输入事件中的槽值可能与为槽配置的值之一不匹配。例如，如果用户对“您想要什么颜色的车？”提示给出的响应是 “披萨”，Amazon Lex 会将“披萨”返回为插槽值。您的函数应对这些值进行验证，以确保它们在上下文中有意义。

  slotDetails 提供有关槽值的附加信息。resolutions 数组包含为槽识别的附加值的列表。每个槽最多可以有五个值。

originalValue 字段包含用户为槽输入的值。在将槽类型配置为将最高解析值返回为槽值时，originalValue 可能与 slots 字段中的值不同。

confirmationStatus 向确认提示 (如果有) 提供用户响应。例如，如果 Amazon Lex 询问“是否要订购大份的芝士披萨？”，根据用户响应，此字段的值可能为 Confirmed 或 Denied。否则，此字段的值为 None。

如果用户确认意图，则 Amazon Lex 会将此字段设置为 Confirmed。如果用户拒绝意图，则 Amazon Lex 会将此值设置为 Denied。

在确认响应中，用户表达可能提供槽更新。例如，用户可能会说“是的，改为中份的。”在这种情况下，后续 Lambda 事件具有更新的插槽值，PizzaSize 将设置为 medium。Amazon Lex 将 confirmationStatus 设置为 None，因为用户修改了一些插槽数据，需要 Lambda 函数执行用户数据验证。

- alternativeIntents — 如果您启用置信度分数，Amazon Lex 最多会返回四个替代意图。每个意图都包含一个分数，该分数表明 Amazon Lex 根据用户的言语对意图是正确意图的信心程度。

  替代意图的内容与 currentIntent 字段的内容相同。有关更多信息，请参阅 使用置信度分数。

- bot – 用于处理请求的自动程序的相关信息。
  - name – 用于处理请求的自动程序的名称。
  - alias – 用于处理请求的自动程序版本的别名。
  - version – 用于处理请求的自动程序的版本。

- userId — 此值由客户端应用程序提供。Amazon Lex 将此值传递给 Lambda 函数。

- inputTranscript – 用于处理请求的文本。

  如果输入为文本，则 `inputTranscript` 字段包含用户输入的文本。

  如果输入为音频流，则 `inputTranscript` 字段包含从该音频流中提取的文本。这是经过实际处理以识别目的和槽值的文本。

- invocationSource — 要指示 Amazon Lex 调用 Lambda 函数的原因，它会将此处设置为以下值之一：

  - `DialogCodeHook` — Amazon Lex 设置此值以指示 Lambda 函数来初始化函数并验证用户的数据输入。

    当配置意图以调用 Lambda 函数作为初始化和验证代码挂钩时，Amazon Lex 将在 Amazon Lex 理解意图后针对每个用户输入（言语）调用指定的 Lambda 函数。

    > **Note**
    >
    > 如果意图不明，Amazon Lex 将无法调用 Lambda 函数。

  - `FulfillmentCodeHook` — Amazon Lex 将此值设置为指示 Lambda 函数履行意图。

    如果配置意图以作为履行代码挂钩来调用 Lambda 函数，则 Amazon Lex 仅在它具有履行意图的所有插槽数据后才将 `invocationSource` 设置为此值。

在您的意图配置中，您可以拥有两个单独的 Lambda 函数来初始化和验证用户数据并履行意图。您也可使用一个 Lambda 函数执行以上两种操作。在这种情况下，您的 Lambda 函数可以使用 `invocationSource` 值来遵循正确的代码路径。

- outputDialogMode— 对于每个用户输入，客户端都会使用运行时 API 操作之一向 Amazon Lex 发送请求，PostContent或者PostText。Amazon Lex 使用请求参数来确定对客户端的响应是文本还是语音，并相应地设置此字段。

  Lambda 函数可使用此信息生成相应的消息。例如，如果客户端需要语音响应，则您的 Lambda 函数可以返回语音合成标记语言 (SSML) 而不是文本。

- messageVersion — 消息的版本，用于标识进入 Lambda 函数的事件数据的格式以及来自 Lambda 函数的响应的预期格式。

  > **Note**
  >
  > 您在定义意图时可以配置此值。在当前实现中，仅支持消息版本 1.0。因此，控制台假定 1.0 的默认值，并且不显示消息版本。

- sessionAttributes — 客户端在请求中发送的特定于应用程序的会话属性。如果您希望 Amazon Lex 在对客户端的响应中包含这些属性，则您的 Lambda 函数应在响应中将它们发送回 Amazon Lex。有关更多信息，请参阅 设置会话属性

- requestAttributes — 客户端在请求中发送的特定于请求的属性。可以使用请求属性传递不需要在整个会话中保留的信息。如果没有请求属性，该值将为空。有关更多信息，请参阅 设置请求属性

- recentIntentSummary查看-有关意图状态的信息。您可以看到有关所用最后三个意图的信息。您可以使用此信息在意图中设置值或返回到之前的意图。有关更多信息，请参阅 使用 Amazon Lex API 管理会话。

- sentimentResponse — 最后一个语句的 Amazon Comprehend 情绪分析结果。您可以使用此信息，根据用户表达的情绪来管理自动程序的对话流。有关更多信息，请参阅 情绪分析。

- kendraResponse — 对 Amazon Kendra 索引的查询结果。仅出现在实现代码挂钩的输入中，并且仅当目的扩展 AMAZON.KendraSearchIntent 内置目的时。该字段包含来自 Amazon Kendra 搜索的整个响应。有关更多信息，请参阅 AMAZON.KendraSearchIntent。

- activeContexts — 在与用户的这一轮对话中处于活动状态的一个或多个上下文。
  - timeToLive— 在与用户的对话中，上下文保持活动状态的时间长度或回合次数。
  - name — 上下文的名称。
  - parameters — 键/值对列表，其中包含激活上下文的意图中插槽的名称和值。

  有关更多信息，请参阅 设置意图上下文。

响应格式

Amazon Lex 预计来自 Lambda 函数的响应，格式如下：

```
{
  "sessionAttributes": {
    "key1": "value1",
    "key2": "value2"
    ...
  },
  "recentIntentSummaryView": [
    {
       "intentName": "Name",
       "checkpointLabel": "Label",
       "slots": {
         "slot name": "value",
         "slot name": "value"
        },
       "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if
configured)",
        "dialogActionType": "ElicitIntent, ElicitSlot, ConfirmIntent, Delegate, or
Close",
        "fulfillmentState": "Fulfilled or Failed",
        "slotToElicit": "Next slot to elicit"
```

```
    }
  ],
  "activeContexts": [
     {
       "timeToLive": {
          "timeToLiveInSeconds": seconds,
          "turnsToLive": turns
       },
       "name": "name",
       "parameters": {
         "key name": "value"
       }
     }
  ],
  "dialogAction": {
    "type": "ElicitIntent, ElicitSlot, ConfirmIntent, Delegate, or Close",
    Full structure based on the type field. See below for details.
  }
}
```

响应包含四个字段。sessionAttributes、recentIntentSummaryView 和 activeContexts 字段是可选的，dialogAction 字段是必需的。dialogAction 字段的内容取决于 type 字段的值。有关详细信息，请参阅dialogAction。

sessionAttributes

可选。如果您包含 sessionAttributes 字段，该字段可以为空。如果您的 Lambda 函数未返回会话属性，通过 API 或 Lambda 函数传递的最后一个已知 sessionAttributes 将保留。有关更多信息，请参阅 PostContent 和 PostText 操作。

```
  "sessionAttributes": {
     "key1": "value1",
     "key2": "value2"
  }
```

recentIntentSummary查看

可选。如果包含，则为一个或多个近期意图设置值。您最多可以包含三种意图的信息。例如，您可以根据当前意图收集的信息为先前意图设置值。摘要中的信息必须对意图是有效的。例如，意图名称必须是机器人中的意图。如果在摘要视图中包含槽值，则该槽必须存在于意图中。如果您不在响应中包含

recentIntentSummaryView，则最近意图的所有值保持不变。有关详细信息，请参阅 PutSession 操作或 IntentSummary 数据类型。

```
"recentIntentSummaryView": [
    {
       "intentName": "Name",
       "checkpointLabel": "Label",
       "slots": {
         "slot name": "value",
         "slot name": "value"
        },
       "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if
 configured)",
        "dialogActionType": "ElicitIntent, ElicitSlot, ConfirmIntent, Delegate, or
 Close",
        "fulfillmentState": "Fulfilled or Failed",
        "slotToElicit": "Next slot to elicit"
    }
 ]
```

activeContexts

可选。如果包含，则设置一个或多个上下文的值。例如，您可以包含一个上下文，以使一个或多个意图在下一轮对话中有资格识别该上下文作为输入。

任何未包含在响应中的活动上下文的 time-to-live值都会递减，并且可能在下一个请求时仍处于活动状态。

如果您为输入事件中包含 time-to-live的上下文指定 a of 0，则该上下文将在下一个请求中处于非活动状态。

有关更多信息，请参阅 设置意图上下文。

dialogAction

必需。dialogAction 字段指示 Amazon Lex 进行下一步操作，并说明在 Amazon Lex 将响应返回给客户端后用户应执行的操作。

type 字段指示下一步操作。它还确定 Lambda 函数需要作为 dialogAction 值的一部分提供的其他字段。

- `Close` — 通知 Amazon Lex 不要期望用户进行响应。例如，“您的披萨已下单”不需要响应。

  `fulfillmentState` 字段为必填项。Amazon Lex 使用此值在对客户端应用程序的 PostContent 或 PostText 响应中设置 `dialogState` 字段。`message` 和 `responseCard` 字段是可选的。如果您不指定消息，则 Amazon Lex 将使用为此意图配置的再见消息或后续消息。

```
"dialogAction": {
    "type": "Close",
    "fulfillmentState": "Fulfilled or Failed",
    "message": {
      "contentType": "PlainText or SSML or CustomPayload",
      "content": "Message to convey to the user. For example, Thanks, your pizza has
 been ordered."
    },
   "responseCard": {
      "version": integer-value,
      "contentType": "application/vnd.amazonaws.card.generic",
      "genericAttachments": [
          {
             "title":"card-title",
             "subTitle":"card-sub-title",
             "imageUrl":"URL of the image to be shown",
             "attachmentLinkUrl":"URL of the attachment to be associated with the
 card",
             "buttons":[
                 {
                    "text":"button-text",
                    "value":"Value sent to server on button click"
                 }
              ]
           }
       ]
     }
  }
```

- `ConfirmIntent` — 通知 Amazon Lex，用户应提供“是”或“否”答案以确认或拒绝当前意图。

  必须包含 `intentName` 和 `slots` 字段。对于指定目的的每个填充槽，`slots` 字段都必须包含一个条目。未填充槽的 `slots` 字段无需包含条目。如果目的的 `confirmationPrompt` 字段为空，

则您必须包括 message 字段。Lambda 函数返回的 message 字段的内容优先于意图中指定的 confirmationPrompt。responseCard 字段为可选项。

```
"dialogAction": {
    "type": "ConfirmIntent",
    "message": {
      "contentType": "PlainText or SSML or CustomPayload",
      "content": "Message to convey to the user. For example, Are you sure you want a
 large pizza?"
    },
   "intentName": "intent-name",
   "slots": {
      "slot-name": "value",
      "slot-name": "value",
      "slot-name": "value"
   },
   "responseCard": {
      "version": integer-value,
      "contentType": "application/vnd.amazonaws.card.generic",
      "genericAttachments": [
          {
             "title":"card-title",
             "subTitle":"card-sub-title",
             "imageUrl":"URL of the image to be shown",
             "attachmentLinkUrl":"URL of the attachment to be associated with the
 card",
             "buttons":[
                 {
                    "text":"button-text",
                    "value":"Value sent to server on button click"
                 }
              ]
           }
       ]
     }
  }
```

- Delegate — 指示 Amazon Lex 根据机器人配置选择下一步操作。如果响应不包含任何会话属性，Amazon Lex 将保留现有属性。如果您希望槽值为空，则无需在请求中包含槽字段。如果您的实现函数返回 Delegate 对话操作而没有删除任何槽，您将收到 DependencyFailedException 异常。

kendraQueryRequestPayload 和 kendraQueryFilterString 字段是可选的，并且仅当目的派生自 AMAZON.KendraSearchIntent 内置目的时才使用。有关更多信息，请参阅 AMAZON.KendraSearchIntent。

```
 "dialogAction": {
  "type": "Delegate",
  "slots": {
     "slot-name": "value",
     "slot-name": "value",
     "slot-name": "value"
  },
  "kendraQueryRequestPayload": "Amazon Kendra query",
  "kendraQueryFilterString": "Amazon Kendra attribute filters"
 }
```

- ElicitIntent — 通知 Amazon Lex，用户应使用包含意图的言语进行响应。例如，“我想要大份披萨”指示 OrderPizzaIntent。另一方面，言语“大份”不足以供 Amazon Lex 推断用户的意图。

  message 和 responseCard 字段是可选的。如果您不提供消息，则 Amazon Lex 会使用机器人的某个澄清提示。如果没有定义澄清提示，Amazon Lex 将返回“400 错误请求”异常。

```
{
  "dialogAction": {
    "type": "ElicitIntent",
    "message": {
      "contentType": "PlainText or SSML or CustomPayload",
      "content": "Message to convey to the user. For example, What can I help you
 with?"
    },
    "responseCard": {
      "version": integer-value,
      "contentType": "application/vnd.amazonaws.card.generic",
      "genericAttachments": [
          {
             "title":"card-title",
             "subTitle":"card-sub-title",
             "imageUrl":"URL of the image to be shown",
             "attachmentLinkUrl":"URL of the attachment to be associated with the
 card",
             "buttons":[
```

```
                {
                   "text":"button-text",
                   "value":"Value sent to server on button click"
                }
             ]
          }
       ]
    }
}
```

- `ElicitSlot` — 通知 Amazon Lex，用户应该在响应中提供插槽值。

  `intentName`、`slotToElicit` 和 `slots` 为必需字段。`message` 和 `responseCard` 字段是可选的。如果您不指定消息，则 Amazon Lex 将使用为插槽配置的某个插槽引发提示。

```
 "dialogAction": {
   "type": "ElicitSlot",
   "message": {
     "contentType": "PlainText or SSML or CustomPayload",
     "content": "Message to convey to the user. For example, What size pizza would
you like?"
   },
  "intentName": "intent-name",
  "slots": {
     "slot-name": "value",
     "slot-name": "value",
     "slot-name": "value"
  },
  "slotToElicit" : "slot-name",
  "responseCard": {
     "version": integer-value,
     "contentType": "application/vnd.amazonaws.card.generic",
     "genericAttachments": [
         {
            "title":"card-title",
            "subTitle":"card-sub-title",
            "imageUrl":"URL of the image to be shown",
            "attachmentLinkUrl":"URL of the attachment to be associated with the
card",
            "buttons":[
```

```
                {
                   "text":"button-text",
                   "value":"Value sent to server on button click"
                }
             ]
          }
       ]
    }
 }
```

Amazon Lex 和 AWS Lambda 蓝图

Amazon Lex 控制台提供了已预先配置的示例机器人（称为机器人蓝图），以便您可在控制台中快速创建和测试机器人。对于所有这些机器人蓝图，也提供了 Lambda 函数蓝图。这些蓝图提供了适用于它们相应的自动程序的示例代码。您可以使用这些蓝图快速创建配置为 Lambda 函数作为代码挂钩的机器人，无需编写代码即可测试 end-to-end设置。

您可以使用以下 Amazon Lex 机器人蓝图和相应的 AWS Lambda 函数蓝图作为机器人的代码挂钩：

- Amazon Lex 蓝图 — `OrderFlowers`
  - AWS Lambda 蓝图 — `lex-order-flowers-python`
- Amazon Lex 蓝图 — `ScheduleAppointment`
  - AWS Lambda 蓝图 — `lex-make-appointment-python`
- Amazon Lex 蓝图 — `BookTrip`
  - AWS Lambda 蓝图 — `lex-book-trip-python`

要使用蓝图创建机器人，并将其配置为使用 Lambda 函数作为代码挂钩，请参阅练习 1：使用蓝图创建 Amazon Lex 机器人（控制台）。有关使用其他蓝图的示例，请参阅其他示例：创建 Amazon Lex 机器人。

更新特定区域设置的蓝图

如果您在英语（美国）(en-US) 以外的区域设置中使用蓝图，则需要更新任何意图的名称以包含该区域设置。例如，如果您使用的是 `OrderFlowers` 蓝图，则需要执行以下操作。

- 在 Lambda 函数代码末尾处找到 `dispatch` 函数。

- 在 dispatch 函数中，更新意图的名称以包含您正在使用的区域设置。例如，如果您使用的是英语（澳大利亚）(en-AU) 区域设置，请更改以下行：

  ```
  if intent_name == 'OrderFlowers':
  ```

  到

  ```
  if intent_name == 'OrderFlowers_enAU':
  ```

其他蓝图使用其他意图名称，在使用它们之前，应按上述方式对其进行更新。

部署 Amazon Lex 机器人

本节提供了一些示例，演示如何在各种消息收发平台上及移动应用程序中部署 Amazon Lex 机器人。

主题

在消息收发平台上部署 Amazon Lex 机器人

本节说明如何在 Facebook、Slack 和 Twilio 消息收发平台上部署 Amazon Lex 机器人。

> **Note**
>
> 在存储您的 Facebook、Slack 或 Twilio 配置时，Amazon Lex 使用 AWS Key Management Service 客户托管式密钥来加密信息。首次创建通往这些消息收发平台之一的通道时，Amazon Lex 将创建一个默认的客户托管式密钥 (aws/lex)。或者，您可以使用创建自己的客户托管密钥 AWS KMS。这可以提高灵活性，包括提供创建、轮换和禁用密钥的能力。您还可以定义访问控制，并审核用于保护数据的加密密钥。有关更多信息，请参见AWS Key Management Service 开发人员指南。

当消息收发平台向 Amazon Lex 发送请求时，它将包含平台特定信息作为您的 Lambda 函数的请求属性。使用这些属性可自定义您的自动程序的行为方式。有关更多信息，请参阅 设置请求属性。

所有属性采用命名空间 `x-amz-lex:` 作为前缀。例如，`user-id` 属性称为 `x-amz-lex:user-id`。除了特定于特殊平台的属性之外，还有所有消息收发平台发送的通用属性。下表列出了消息收发平台发送到您的机器人的 Lambda 函数的请求属性。

通用请求属性

属性	描述
channel-id	Amazon Lex 中的通道端点标识符。
channel-name	Amazon Lex 中的通道名称。
channel-type	下列值之一：

属性	描述
	• Facebook • Kik • Slack • Twilio-SMS
webhook-endpoint-url	通道的 Amazon Lex 端点。

Facebook 请求属性

属性	描述
user-id	发件人的 Facebook 标识符。参见 https://developers.facebook.com/docs/Messenger platform/webhook-reference/message-已收到。
facebook-page-id	收件人的 Facebook 页面标识符。参见 https://developers.facebook.com/docs/Messenger platform/webhook-reference/message-已收到。

Kik 请求属性

属性	描述
kik-chat-id	您的聊天机器人正在进行的对话的标识符。有关更多信息，请参阅 https://dev.kik.com/#/docs/messaging #message-formats。
kik-chat-type	消息源自的对话的类型。有关更多信息，请参阅 https://dev.kik.com/#/docs/messaging #message-formats。
kik-message-id	标识消息的 UUID。有关更多信息，请参阅 https://dev.kik.com/#/docs/messaging #message-formats。
kik-message-type	消息类型。有关更多信息，请参阅 https://dev.kik.com/#/docs/messaging #message-types。

Twilio 请求属性

属性	描述
`user-id`	发件人 (“From”) 的电话号码。请参阅 https://www.twilio.com/docs/api/rest/message。
`twilio-target-phone-number`	收件人 (“To”) 的电话号码。请参阅 https://www.twilio.com/docs/api/rest/message。

Slack 请求属性

属性	描述
`user-id`	Slack 用户标识符。见https://api.slack.com/types/用户。
`slack-team-id`	发送消息的团队的标识符。参见 https://api.slack.com/methods/team.info。
`slack-bot-token`	允许机器人访问 Slack APIs 的开发者令牌。参见https://api.slack.com/docs/代币类型。

将 Amazon Lex 机器人与 Facebook Messenger 集成

本练习演示如何将 Facebook Messenger 与您的 Amazon Lex 机器人集成。请执行下列步骤：

1. 创建 Amazon Lex 机器人
2. 创建 Facebook 应用程序
3. 将 Facebook Messenger 与您的 Amazon Lex 机器人集成
4. 验证集成

主题

步骤 1：创建 Amazon Lex 机器人

如果您还没有 Amazon Lex 机器人，请创建并部署一个。在本主题中，我们假定您使用的是您在入门练习 1 中创建的自动程序。但是，您可以使用本指南中提供的任何示例自动程序。有关入门练习 1，请参阅 练习 1：使用蓝图创建 Amazon Lex 机器人（控制台）。

1. 创建 Amazon Lex 机器人。有关说明，请参阅 练习 1：使用蓝图创建 Amazon Lex 机器人（控制台）。
2. 部署此自动程序并创建别名。有关说明，请参阅 练习 3：发布版本和创建别名。

步骤 2：创建 Facebook 应用程序

在 Facebook 开发人员门户上，创建一个 Facebook 应用程序和一个 Facebook 页面。有关说明，请参阅 Facebook Messenger 平台文档中的快速入门。记下以下信息：

- Facebook 应用程序的 App Secret
- Facebook 页面的 Page Access Token

步骤 3：将 Facebook Messenger 与 Amazon Lex 机器人集成

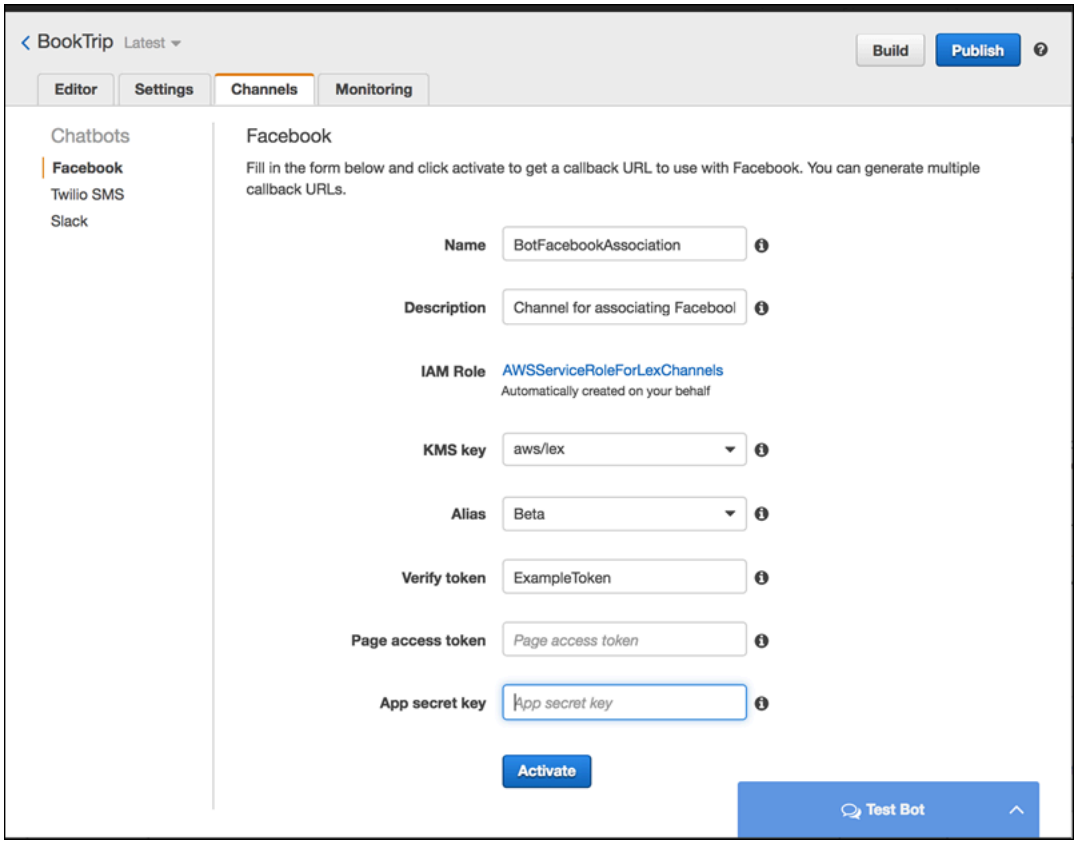
在本节中，将 Facebook Messenger 与您的 Amazon Lex 机器人集成。

完成此步骤后，此控制台将提供一个回调 URL。记下此 URL。

将 Facebook Messenger 与您的自动程序集成

1. a. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
   b. 选择您的 Amazon Lex 机器人。
   c. 选择 Channels。
   d. 在聊天机器人下面，选择 Facebook。此控制台将显示 Facebook 集成页面。
   e. 在 Facebook 集成页面上，执行以下操作：
      - 键入以下名称：`BotFacebookAssociation`。
      - 对于 KMS key，选择 aws/lex。
      - 对于 Alias，选择自动程序别名。

- 对于 Verify token，键入令牌。这可以是您选择的任何字符串 (例如，ExampleToken)。稍后设置 Webhook 时会在 Facebook 开发人员门户中使用此令牌。
- 对于 Page access token，键入在步骤 2 中从 Facebook 获得的令牌。
- 对于 App secret key，键入在步骤 2 中从 Facebook 获得的密钥。



f. 选择激活。

此控制台将创建自动程序通道关联并返回一个回调 URL。记下此 URL。

2. 在 Facebook 开发人员门户上，选择您的应用程序。
3. 选择 Messenger 产品，然后在页面的 Webhooks 部分选择 Setup webhooks。

有关说明，请参阅 Facebook Messenger 平台文档中的快速入门。

4. 在订阅向导的 webhook 页面上，执行以下操作：

- 对于回调 URL，键入本过程前面部分在 Amazon Lex 控制台中提供的回调 URL。
- 对于验证令牌，键入您在 Amazon Lex 中使用的同一令牌。
- 选择 Subscription Fields (messages、messaging_postbacks 和 messaging_optins)。

- 选择 Verify and Save。这将在 Facebook 和 Amazon Lex 之间发起握手。

5. 启用 Webhooks 集成。选择您刚刚创建的页面，然后选择 subscribe。

> **Note**
>
> 如果您更新或重新创建了 webhook，则必须取消订阅，然后重新订阅该页面。

步骤 4：测试集成

您现在可使用 Amazon Lex 机器人从 Facebook Messenger 启动对话。

1. 打开您的 Facebook 页面，然后选择 Message。
2. 在 Messenger 窗口中，使用步骤 1：创建 Amazon Lex 机器人（控制台）时提供的相同测试表达。

将 Amazon Lex 机器人与 Kik 集成

本练习提供有关将 Amazon Lex 机器人与 Kik 消息收发应用程序集成的说明。请执行下列步骤：

1. 创建 Amazon Lex 机器人。
2. 使用 Kik 应用程序和网站创建 Kik 自动程序。
3. 使用 Amazon Lex 控制台将您的 Amazon Lex 机器人与 Kik 机器人集成。
4. 使用 Kik 加入与您的 Amazon Lex 机器人的对话，以测试您的 Amazon Lex 机器人与 Kik 之间的关联。

主题

步骤 1：创建 Amazon Lex 机器人

如果您还没有 Amazon Lex 机器人，请创建并部署一个。在本主题中，我们假定您使用的是您在入门练习 1 中创建的自动程序。但是，您可以使用本指南中提供的任何示例自动程序。有关入门练习 1，请参阅 练习 1：使用蓝图创建 Amazon Lex 机器人（控制台）

1. 创建 Amazon Lex 机器人。有关说明，请参阅 练习 1：使用蓝图创建 Amazon Lex 机器人（控制台）。
2. 部署此自动程序并创建别名。有关说明，请参阅 练习 3：发布版本和创建别名。

下一个步骤

步骤 2：创建 Kik 自动程序

步骤 2：创建 Kik 自动程序

在此步骤中，您将使用 Kik 用户界面创建 Kik 自动程序。根据在创建此机器人时生成的信息将其连接到您的 Amazon Lex 机器人。

1. 请下载并安装 Kik 应用程序，然后注册一个 Kik 账户 (如果您没有此账户)。如果您有账户，请登录。
2. 打开 Kik 网站，网址为 https://dev.kik.com/。保持浏览器窗口处于打开状态。
3. 在 Kik 应用程序中，选择齿轮图标以打开设置，然后选择 Your Kik Code。
4. 扫码 Kik 网站上的 Kik 代码以打开 Botsworth 聊天自动程序。选择 Yes 以打开自动程序控制面板。
5. 在 Kik 应用程序中，选择 Create a Bot。按照提示操作以创建您的 Kik 自动程序。
6. 创建自动程序后，在您的浏览器中选择 Configuration。确保已选中您的新自动程序。
7. 记下自动程序名称和 API 密钥以在下一部分中使用。

下一个步骤

步骤 3：将 Kik 机器人与 Amazon Lex 机器人集成

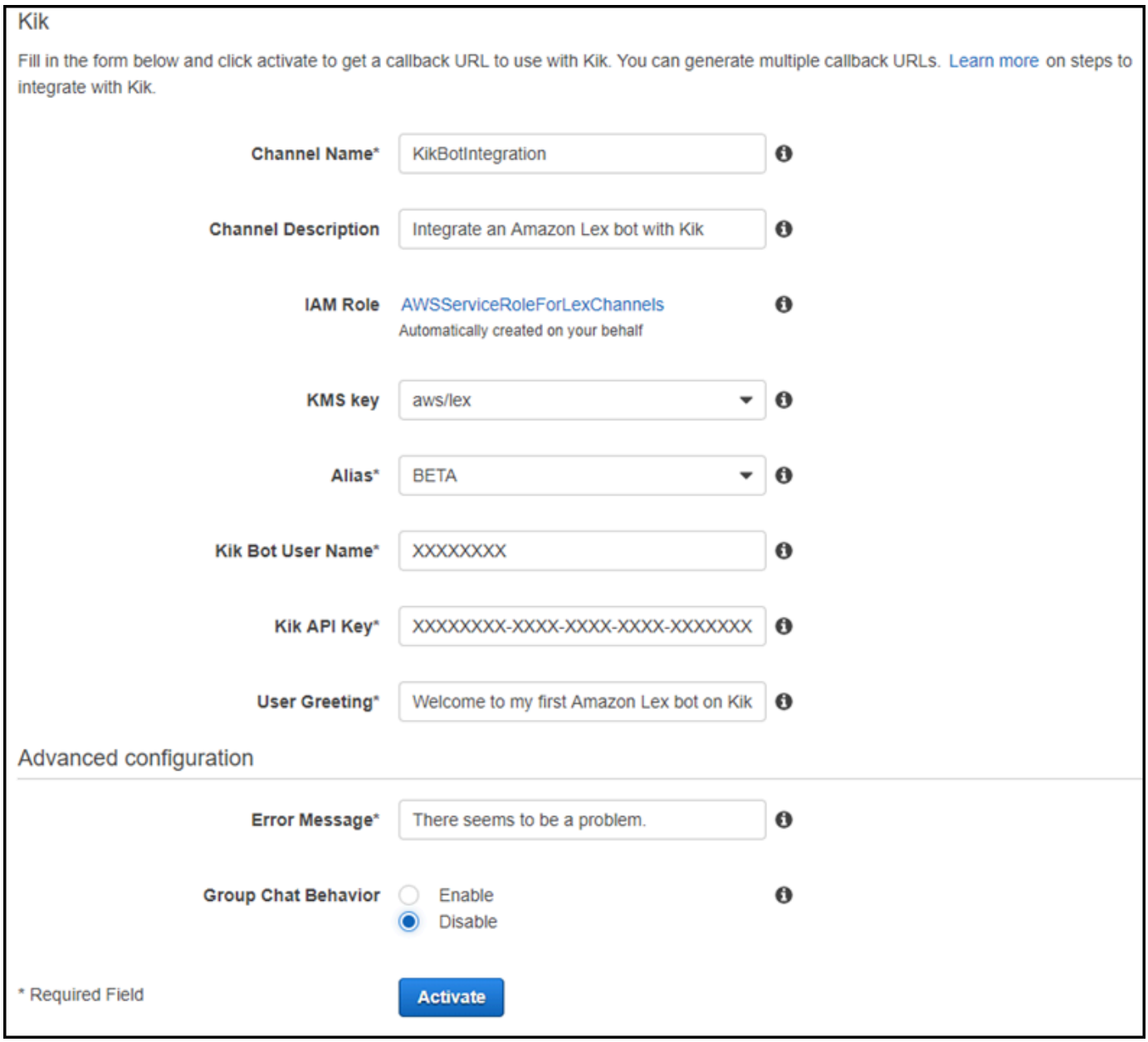
步骤 3：将 Kik 机器人与 Amazon Lex 机器人集成

现在您已经创建了一个 Amazon Lex 机器人和一个 Kik 机器人，可以使用 Amazon Lex 在这两个机器人之间创建通道关联。激活关联后，Amazon Lex 将自动使用 Kik 设置回调 URL。

1. 登录 AWS 管理控制台，然后在上打开 Amazon Lex 控制台https://console.aws.amazon.com/lex/。
2. 选择您在步骤 1 中创建的 Amazon Lex 机器人。
3. 选择 Channels 选项卡。
4. 在 Channels 部分中，选择 Kik。
5. 在 Kik 页面上，提供以下值：

   - 键入名称。例如，`BotKikIntegration`。
   - 键入描述。
   - 从 KMS key 下拉列表中选择“aws/lex”。
   - 对于 Alias，从下拉菜单中选择别名。
   - 对于 Kik bot user name，键入您在 Kik 上为自动程序提供的名称。
   - 对于 Kik API key，键入在 Kik 上分配给自动程序的 API 密钥。
   - 对于 User greeting，键入您希望您的自动程序在用户第一次与之聊天时发送的问候语。
   - 对于 Error message，输入不理解部分对话时显示给用户的错误消息。
   - 对于 Group chat behavior，选择以下选项之一：
     - Enable – 在单次对话中使整个聊天组都能与您的自动程序交互。
     - Disable – 将对话限制为聊天组中的一位用户。

   - 选择 Activate 以创建关联并将它链接到 Kik 自动程序。



下一个步骤

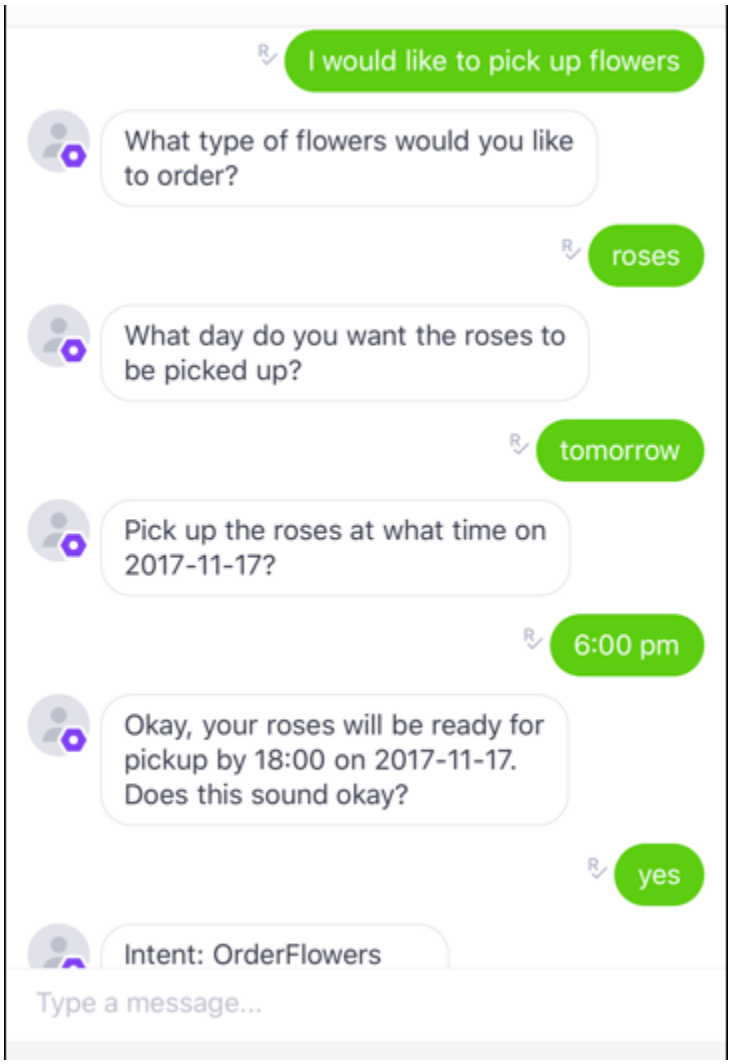
[步骤 4：测试集成](#)

步骤 4：测试集成

现在您已在 Amazon Lex 机器人和 Kik 之间创建关联，可使用 Kik 应用程序测试此关联。

1. 启动 Kik 应用程序并登录。选择您创建的自动程序。

2. 您可使用以下内容测试自动程序：



在您输入每个短语时，您的 Amazon Lex 机器人将使用您为每个插槽创建的提示通过 Kik 响应。

将 Amazon Lex 机器人与 Slack 集成

本练习提供有关将 Amazon Lex 机器人与 Slack 消息收发应用程序集成的说明。请执行下列步骤：

1. 创建 Amazon Lex 机器人。
2. 创建 Slack 消息收发应用程序。
3. 将 Slack 应用程序与您的 Amazon Lex 机器人集成。
4. 通过与您的 Amazon Lex 机器人进行对话来测试集成。您将使用 Slack 应用程序发送消息，并在浏览器窗口中进行测试。

主题

步骤 1：创建 Amazon Lex 机器人

如果您还没有 Amazon Lex 机器人，请创建并部署一个。在本主题中，我们假定您使用的是您在入门练习 1 中创建的自动程序。但是，您可以使用本指南中提供的任何示例自动程序。有关入门练习 1，请参阅 练习 1：使用蓝图创建 Amazon Lex 机器人（控制台）

1. 创建 Amazon Lex 机器人。有关说明，请参阅 练习 1：使用蓝图创建 Amazon Lex 机器人（控制台）。
2. 部署此自动程序并创建别名。有关说明，请参阅 练习 3：发布版本和创建别名。

下一个步骤

步骤 2：注册 Slack 并创建 Slack 团队

步骤 2：注册 Slack 并创建 Slack 团队

注册 Slack 账户并创建 Slack 团队。有关说明，请参阅使用 Slack。在下一部分中，您将创建一个 Slack 应用程序，所有 Slack 团队都可以安装该应用程序。

下一个步骤

步骤 3：创建 Slack 应用程序

步骤 3：创建 Slack 应用程序

请在此部分中执行以下操作：

1. 在 Slack API 控制台上创建 Slack 应用程序
2. 配置该应用程序，以将交互式消息收发功能添加到自动程序中：

在本部分结束时，您将获得应用程序凭证 (客户端 ID、客户端密钥和验证令牌)。在下一节中，您将使用这些信息在 Amazon Lex 控制台中配置机器人通道关联。

1. 通过以下网址登录 Slack API 控制台：http://api.slack.com。
2. 创建应用程序。

   在您成功创建应用程序以后，Slack 会显示应用程序的 Basic Information 页面。
3. 按如下所示配置应用程序的功能：

   - 在左侧菜单中，选择互动性和快捷方式。
     - 选择打开交互式组件的开关。
     - 在 Request URL 框中，指定任何有效的 URL。例如，您可以使用 **`https://slack.com`**。

       > **Note**
       >
       > 目前，请输入任何有效的 URL，以便获得在下一步中需要的验证令牌。您在 Amazon Lex 控制台中添加机器人通道关联后，将需要更新此 URL。

     - 选择 Save Changes（保存更改）。
4. 在左侧菜单的 Settings 中，选择 Basic Information。记录以下应用程序凭证：

   - 客户端 ID
   - 客户端密钥
   - 验证令牌

下一个步骤

步骤 4：将 Slack 应用程序与 Amazon Lex 机器人集成

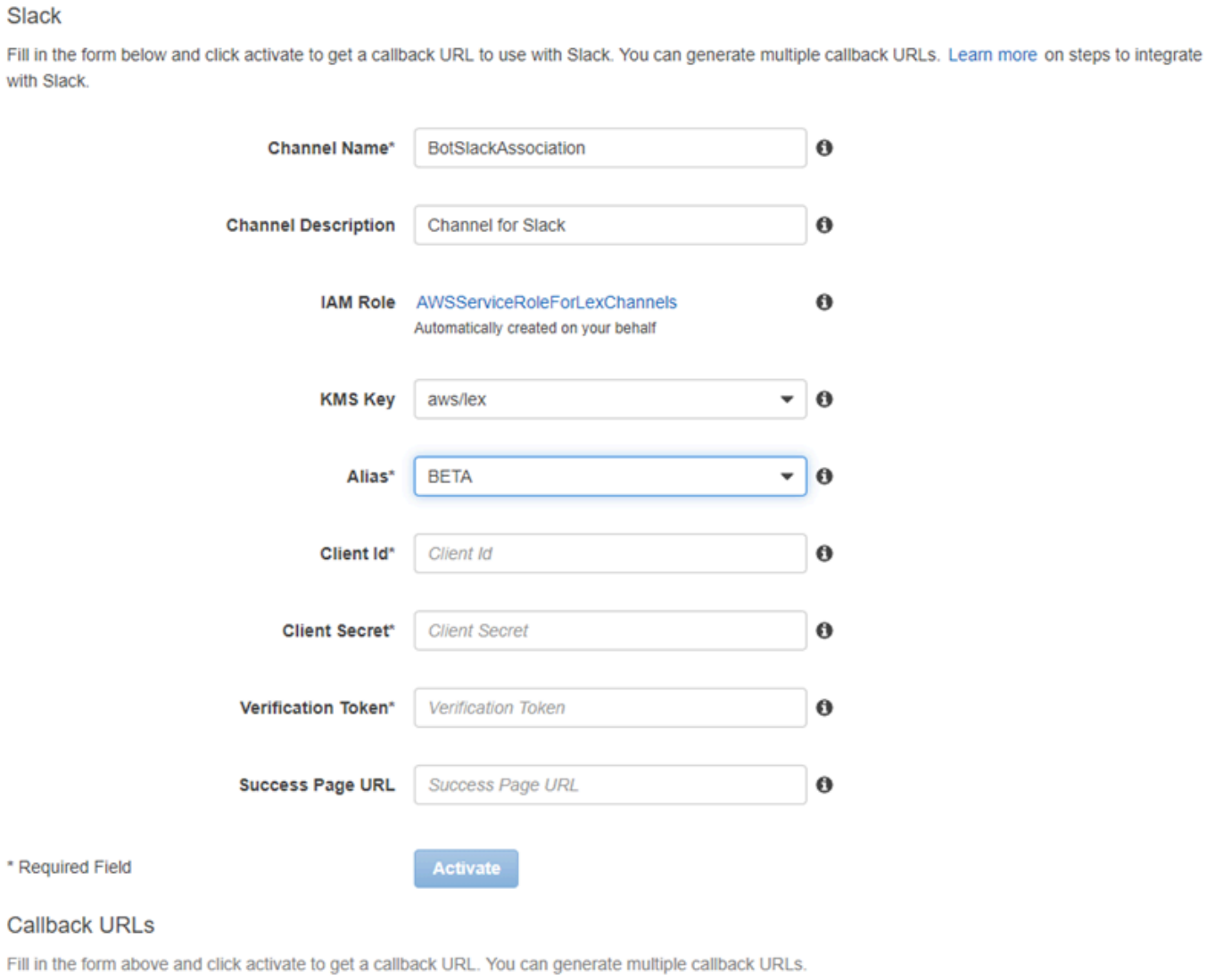
步骤 4：将 Slack 应用程序与 Amazon Lex 机器人集成

现在，您已经有了 Slack 应用程序凭证，可以将该应用程序与您的 Amazon Lex 机器人集成了。要将 Slack 应用程序与您的机器人关联，需要在 Amazon Lex 中添加机器人通道关联。

在 Amazon Lex 控制台中，激活一个机器人通道关联，以将机器人与您的 Slack 应用程序关联。激活机器人频道关联后，Amazon Lex 会返回两个 URLs（回发网址和网OAuth址）。把它们录下来，URLs 因为你以后需要它们。

将 Slack 应用程序与您的 Amazon Lex 机器人集成

1. 登录 AWS 管理控制台，然后在上打开 Amazon Lex 控制台https://console.aws.amazon.com/lex/。
2. 选择您在步骤 1 中创建的 Amazon Lex 机器人。
3. 选择 Channels 选项卡。
4. 在左侧菜单中，选择 Slack。
5. 在 Slack 页面上，提供以下值：
   - 键入名称。例如，`BotSlackIntegration`。
   - 从 KMS key 下拉列表中选择“aws/lex”。
   - 对于 Alias，选择自动程序别名。
   - 键入您在前面的步骤中记录的 Client Id、Client secret 和 Verification Token。这些是 Slack 应用程序的凭证。



6. 选择激活。

控制台创建机器人频道关联并返回两个 URLs（回发网址和网 OAuth 址）。记录这些 URL。在下一部分中，您将更新您的 Slack 应用程序配置以使用这些终端节点，如下所示：

- Postback URL 是侦听 Slack 事件的 Amazon Lex 机器人端点。您可以使用此 URL：
  - 作为 Slack 应用程序的 Event Subscriptions 功能中的请求 URL。
  - 在 Slack 应用程序的 Interactive Messages 功能中替换请求 URL 的占位符值。
- OAuth 网址是你的 Amazon Lex 机器人与 Slack OAuth 握手的终端节点。

下一个步骤

步骤 5：完成 Slack 集成

步骤 5：完成 Slack 集成

在本部分中，您将使用 Slack API 控制台完成 Slack 应用程序的集成。

1. 通过以下网址登录 Slack API 控制台：http://api.slack.com。选择您在步骤 3：创建 Slack 应用程序中创建的应用程序。
2. 按如下方式更新 “OAuth 和权限” 功能：

   a. 在左侧菜单中，选择OAuth 和权限。

   b. 在重定向 URLs部分中，添加 Amazon Lex 在上一步中提供的 OAuth 网址。选择 “添加新的重定向 URL”，然后选择 “保存” URLs。

   c. 在 Bot Token Scopes 部分中，使用添加 OAuth 作用域按钮添加两个权限。用以下文本筛选列表：

      - **`chat:write`**
      - **`team:read`**
3. 通过将请求 URL 值更新为 Amazon Lex 在上一步中提供的 Postback URL 来更新互动性和快捷方式功能。输入您在步骤 4 中保存的 Postback URL，然后选择 Save Changes (保存更改)。
4. 按如下所述订阅 Event Subscriptions 功能：

   - 通过选择 On 选项启用事件。
   - 将请求 URL 值设置为 Amazon Lex 在上一步中提供的 Postback URL。
   - 在 Subscribe to Bot Events 部分中，订阅 `message.im` 自动程序事件以在最终用户和 Slack 自动程序之间启用直接消息收发。
   - 保存更改。
5. 启用从消息选项卡发送消息，如下所示：

   - 从左侧菜单中，选择应用程序主页。
   - 在显示选项卡部分，选择允许用户从消息选项卡中发送 Slash 命令和消息。

下一个步骤

步骤 6：测试集成

步骤 6：测试集成

现在，使用浏览器窗口来测试 Slack 与 Amazon Lex 机器人的集成。

1. 在设置下，选择管理分发。选择 Add to Slack 以安装应用程序。授权自动程序对消息做出响应。
2. 您将被重定向到 Slack 团队。在左侧菜单的 Direct Messages 部分中，选择您的自动程序。如果看不到您的自动程序，请选择 Direct Messages 旁边的加号图标 (+) 进行搜索。
3. 使用您的 Slack 应用程序进行聊天，该应用程序链接到了 Amazon Lex 机器人。现在，您的自动程序会响应消息。

   如果您是使用入门练习 1 创建的自动程序，可以使用该练习中提供的示例对话。有关更多信息，请参阅 步骤 4：将 Lambda 函数添加为代码挂钩（控制台）。

将 Amazon Lex 机器人与 Twilio 可编程 SMS 集成

本练习提供将 Amazon Lex 机器人与 Twilio Simple Messaging Service (SMS) 集成的说明。请执行下列步骤：

1. 创建 Amazon Lex 机器人
2. 将 Twilio 可编程 SMS 与您的机器人 Amazon Lex 集成
3. 在您的手机上使用 SMS 服务来测试设置，从而实现与 Amazon Lex 机器人的交互
4. 测试集成

主题

- 步骤 1：创建 Amazon Lex 机器人
- 步骤 2：创建 Twilio SMS 账户
- 步骤 3：将 Twilio 消息收发服务端点与 Amazon Lex 机器人集成
- 步骤 4：测试集成

步骤 1：创建 Amazon Lex 机器人

如果您还没有 Amazon Lex 机器人，请创建并部署一个。在本主题中，我们假定您使用的是您在入门练习 1 中创建的自动程序。但是，您可以使用本指南中提供的任何示例自动程序。有关入门练习 1，请参阅 练习 1：使用蓝图创建 Amazon Lex 机器人（控制台）。

1. 创建 Amazon Lex 机器人。有关说明，请参阅 练习 1：使用蓝图创建 Amazon Lex 机器人（控制台）。
2. 部署此自动程序并创建别名。有关说明，请参阅 练习 3：发布版本和创建别名。

步骤 2：创建 Twilio SMS 账户

注册 Twilio 账户，并记录以下账户信息：

- ACCOUNT SID
- AUTH TOKEN

有关注册说明，请参阅 https://www.twilio.com/console。

步骤 3：将 Twilio 消息收发服务端点与 Amazon Lex 机器人集成

将 Twilio 与 Amazon Lex 机器人集成

1. 要将 Amazon Lex 机器人与您的 Twilio 可编程 SMS 端点关联，请在 Amazon Lex 控制台中激活机器人通道关联。机器人通道关联激活后，Amazon Lex 会返回回调 URL。请记录此回调 URL，因为您稍后需要用到它。

   a. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。

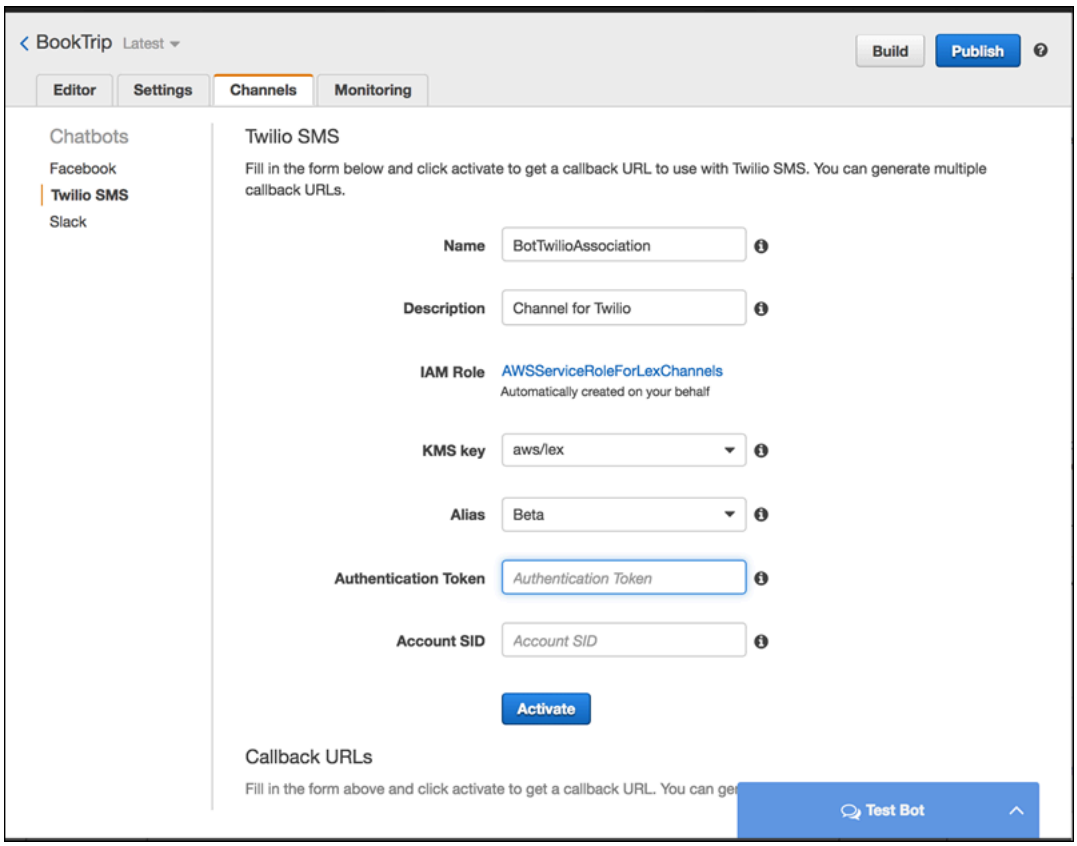
   b. 选择您在步骤 1 中创建的 Amazon Lex 机器人。

   c. 选择 Channels 选项卡。

   d. 在 Chatbots 部分，选择 Twilio SMS。

   e. 在 Twilio SMS 页面上，提供以下信息：

      - 键入名称。例如，`BotTwilioAssociation`。
      - 从 KMS key 中选择“aws/lex”。
      - 对于 Alias，选择自动程序别名。
      - 对于 Authentication Token，键入您的 Twilio 账户的 AUTH TOKEN。
      - 对于 Account SID，键入您的 Twilio 账户的 ACCOUNT SID。



f. 选择激活。

此控制台将创建自动程序通道关联并返回一个回调 URL。记录此 URL。

2. 在 Twilio 控制台中，将 Twilio SMS 端点连接到 Amazon Lex 机器人。

a. 通过 https://www.twilio.com/console 登录 Twilio 主机。

b. 如果您没有 Twilio SMS 终端节点，请进行创建。

c. 通过将请求 URL 值设置为 Amazon Lex 在上一步中提供的回调 URL，更新消息收发服务的入站设置配置。

步骤 4：测试集成

使用您的手机来测试 Twilio SMS 与您的自动程序之间的集成。

测试集成

1. 通过 https://www.twilio.com/console 登录 Twilio 主机并执行以下操作：

   a. 在 Manage Numbers 下验证您拥有与消息收发服务相关联的 Twilio 号码。

      从您的手机向此号码发送消息，并参加与 Amazon Lex 机器人的 SMS 交互。

   b. 验证您的手机是否列出为经过验证的呼叫者 ID。

      如果不是，请按照 Twilio 控制台中的说明启用您计划用于测试的手机。

      现在，您可以使用您的手机向映射到 Amazon Lex 机器人的 Twilio SMS 端点发送消息。

2. 使用您的手机向 Twilio 号码发送消息。

   Amazon Lex 机器人做出了响应。如果您是使用入门练习 1 创建的自动程序，可以使用该练习中提供的示例对话。有关更多信息，请参阅 步骤 4：将 Lambda 函数添加为代码挂钩（控制台）。

在移动应用程序中部署 Amazon Lex 机器人

使用 AWS Amplify，您可以将您的 Amazon Lex 机器人与移动或网络应用程序集成。有关更多信息，请参阅 AWS Amplify 文档中的交互 — 入门。

导入和导出 Amazon Lex 机器人、意图和插槽类型

您可以导入或导出自动程序、目的或插槽类型。例如，如果您想与另一个 AWS 账户中的同事共享一个机器人，则可以导出它，然后将其发送给同事。如果您想向自动程序中添加多个发言，则可以导出它，添加发言，然后将其导回到您的账户。

您可以采用两种格式导出机器人、意图和插槽类型：Amazon Lex（用于共享和修改它们）或 Alexa 技能格式。您只能以 Amazon Lex 格式导入。

当您导出某个资源时，您必须以一种与您要导出到的服务（Amazon Lex 或 Alexa Skills Kit）兼容的格式将其导出。如果您以 Amazon Lex 格式导出自动程序，则可以将其重新导入到您的账户中，或者其他账户中的 Amazon Lex 用户可以将其导入到其账户中。您还可以将自动程序以与 Alexa 技能兼容的格式导出。然后，您可以使用 Alexa Skills Kit 导入自动程序，以使您的自动程序可与 Alexa 一起使用。有关更多信息，请参阅 导出到 Alexa 技能。

当您导出自动程序、目的或插槽类型时，其资源将会写入 JSON 文件。要导出机器人、意图或插槽类型，您可以使用 Amazon Lex 控制台或 GetExport 操作。使用 StartImport 导出自动程序、目的或插槽类型。

主题

以 Amazon Lex 格式导出和导入

要在意图为重新导入 Amazon Lex 的情况下从 Amazon Lex 中导出机器人、意图和插槽类型，您应以 Amazon Lex 格式创建 JSON 文件。您可以在此文件中编辑您的资源并将其导入回 Amazon Lex。例如，您可以向目的中添加发言，然后将更改的目的导入回您的账户。您还可以使用 JSON 格式来共享资源。例如，您可以从一个 AWS 区域导出机器人，然后将其导入另一个区域。或者，您可以将 JSON 文件发送给同事以共享自动程序。

主题

- 用于导入和导出的 JSON 格式

以 Amazon Lex 格式导出

将您的 Amazon Lex 机器人、意图和插槽类型导出为可以导入到 AWS 账户的格式。您可以导出以下资源：

- 自动程序，包括自动程序使用的所有目的和自定义插槽类型
- 目的，包括目的使用的所有自定义槽类型
- 自定义插槽类型，包括插槽类型的所有值

您只能导出资源的带编号版本。您不能导出资源的 $LATEST 版本。

导出是一个异步过程。导出完成后，您将获得一个 Amazon S3 预签名 URL。该 URL 提供包含 JSON 格式导出资源的 .zip 存档的位置。

您可以使用控制台或 GetExport 操作来导出自动程序、目的或自定义插槽类型。

导出自动程序、目的或插槽类型的过程是相同的。在下列步骤中，替换自动程序的目的或插槽类型。

导出自动程序

导出自动程序

1. 登录 AWS 管理控制台，然后在上打开 Amazon Lex 控制台https://console.aws.amazon.com/lex/。
2. 选择 Bots，然后选择要导出的自动程序。
3. 在 Actions 菜单上，选择 Export。
4. 在 Export Bot 对话框中，选择要导出的自动程序的版本。对于 Platform，选择 Amazon Lex。
5. 选择导出。
6. 下载并保存该 .zip 存档。

Amazon Lex 将机器人导出到一个包含在 .zip 存档中的 JSON 文件。要更新机器人，请修改 JSON 文本，然后将其导入回 Amazon Lex。

后续步骤

以 Amazon Lex 格式导入

以 Amazon Lex 格式导入

将资源导出为 Amazon Lex 格式的 JSON 文件后，您可以将包含该资源的 JSON 文件导入一个或多个 AWS 账户。例如，您可以导出机器人，然后将其导入另一个 AWS 区域。或者，您可以将自动程序发送给同事，以便将其导入同事的账户。

当您导入自动程序、目的或插槽类型时，您必须决定是否要覆盖资源的 $LATEST 版本，例如导入期间的目的或插槽类型，或者如果要保留您账户中的资源，则您会希望导入失败。例如，如果您要向您的账户上传资源的已编辑版本，则可以选择覆盖 $LATEST 版本。如果您要上传同事发送给您的资源，则可以选择在存在资源冲突时让导入失败，以便不会替换您自己的资源。

在导入资源时，分配给执行导入请求的用户的权限是适用的。用户必须拥有该账户中导入影响的所有资源的权限。用户还必须具有 GetBot、PutBot、GetIntent、PutIntent、GetSlotType、PutSlotType 操作的权限。有关权限的更多信息，请参阅 Amazon Lex 如何与 IAM 配合使用。

导入会报告在处理期间发生的错误。在导入过程开始之前报告一些错误，在导入过程中报告其他错误。例如，如果导入意图的账户没有权限调用意图所使用的 Lambda 函数，则在对插槽类型或意图进行更改之前导入会失败。如果导入在导入过程中失败，在过程失败之前导入的任何目的或插槽类型的 $LATEST 版本够会被修改。您无法回滚对 $LATEST 版本所做的更改。

当您导入资源时，所有从属资源都会导入资源的 $LATEST 版本，然后给定一个带编号的版本。例如，如果一个自动程序使用一个目的，则该目的被给定一个带编号的版本。如果目的使用自定义插槽类型，则插槽类型会有给定的带编号版本。

资源仅导入一次。例如，如果自动程序包含一个 `OrderPizza` 目的和一个 `OrderDrink` 目的，并且二者都依赖于自定义插槽类型 `Size`，则 `Size` 插槽类型会导入一次并用于这两个目的。

> **Note**
>
> 如果您在导出机器人时将 `enableModelImprovements` 参数设置为 `false`，则必须打开包含机器人定义的 .zip 文件，并将以下区域中的 `enableModelImprovements` 参数更改为 `true`：
>
> - 亚太地区（新加坡）(ap-southeast-1)
> - 亚太地区（东京）(ap-northeast-1)
> - 欧洲（法兰克福）(eu-central-1)
> - 欧洲（伦敦）(eu-west-2)

导入自动程序、目的或自定义插槽类型的过程是相同的。在下列步骤中，相应地替换目的或插槽类型。

导入自动程序

导入自动程序

1. 登录 AWS 管理控制台，然后在上打开 Amazon Lex 控制台https://console.aws.amazon.com/lex/。
2. 选择 Bots，然后选择要导入的自动程序。要导入新的自动程序，请跳过此步骤。
3. 对于 Actions，选择 Import。
4. 对于 Import Bot，选择要导入的自动程序所在的 JSON 文件所在的 .zip 存档。如果您想在合并之前查看合并冲突，请选择 Notify me of merge conflicts。如果您关闭冲突检查，则自动程序使用的所有资源的 $LATEST 版本都将被覆盖。
5. 选择 Import（导入）。如果您选择了有合并冲突时发送通知并且存在冲突，则会出现一个对话框列出它们。要覆盖所有冲突资源的 $LATEST 版本，请选择 Overwrite and continue (覆盖并继续)。要停止导入，请选择 Cancel。

您现在可以在您的账户中测试自动程序。

用于导入和导出的 JSON 格式

以下示例显示用于以 Amazon Lex 格式导出和导入插槽类型、意图和机器人的 JSON 结构。

插槽类型结构

以下是自定义插槽类型的 JSON 结构。当您导入或导出插槽类型时，以及导出依赖于自定义插槽类型的目的时，使用此结构。

```
{
  "metadata": {
    "schemaVersion": "1.0",
    "importType": "LEX",
    "importFormat": "JSON"
  },
  "resource": {
    "name": "slot type name",
    "version": "version number",
    "enumerationValues": [
      {
        "value": "enumeration value",
```

```
        "synonyms": []
      },
      {
        "value": "enumeration value",
        "synonyms": []
      }
    ],
    "valueSelectionStrategy": "ORIGINAL_VALUE or TOP_RESOLUTION"
  }
}
```

目的结构

以下是目的的 JSON 结构。当您导入或导出目的和依赖目的的自动程序时，使用此结构。

```
{
  "metadata": {
    "schemaVersion": "1.0",
    "importType": "LEX",
    "importFormat": "JSON"
  },
  "resource": {
    "description": "intent description",
    "rejectionStatement": {
      "messages": [
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
        }
      ]
    },
    "name": "intent name",
    "version": "version number",
    "fulfillmentActivity": {
      "type": "ReturnIntent or CodeHook"
    },
    "sampleUtterances": [
      "string",
      "string"
    ],
    "slots": [
      {
        "name": "slot name",
```

```
        "description": "slot description",
        "slotConstraint": "Required or Optional",
        "slotType": "slot type",
        "valueElicitationPrompt": {
          "messages": [
            {
              "contentType": "PlainText or SSML or CustomPayload",
              "content": "string"
            }
          ],
          "maxAttempts": value
        },
        "priority": value,
        "sampleUtterances": []
      }
    ],
    "confirmationPrompt": {
      "messages": [
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
        },
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
        }
      ],
      "maxAttempts": value
    },
    "slotTypes": [
        List of slot type JSON structures.
        For more information, see ######.
    ]
  }
}
```

自动程序结构

以下是自动程序的 JSON 结构。当您导入或导出自动程序时，使用此结构。

```
{
  "metadata": {
    "schemaVersion": "1.0",
```

```
    "importType": "LEX",
    "importFormat": "JSON"
  },
  "resource": {
    "name": "bot name",
    "version": "version number",,
    "nluIntentConfidenceThreshold": 0.00-1.00,
    "enableModelImprovements": true | false,
    "intents": [
        List of intent JSON structures.
        For more information, see ####.
    ],
    "slotTypes": [
        List of slot type JSON structures.
        For more information, see ######.
    ],
    "voiceId": "output voice ID",
    "childDirected": boolean,
    "locale": "en-US",
    "idleSessionTTLInSeconds": timeout,
    "description": "bot description",
    "clarificationPrompt": {
      "messages": [
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
        }
      ],
      "maxAttempts": value
    },
    "abortStatement": {
      "messages": [
        {
          "contentType": "PlainText or SSML or CustomPayload",
          "content": "string"
        }
      ]
    }
  }
}
```

导出到 Alexa 技能

您可以将自动程序架构以与 Alexa 技能兼容的格式导出。将自动程序导出到 JSON 文件后，可以使用技能构建器将其上传到 Alexa。

导出自动程序及其架构（交互模型）

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 选择要导出的自动程序。
3. 对于 Actions (操作)，选择 Export (导出)。
4. 选择要导出的自动程序的版本。对于格式，选择 Alexa Skills Kit，然后选择 Export (导出)。
5. 出现下载对话框时，选择保存文件的位置，然后选择 Save (保存)。

下载的文件是一个 .zip 存档，其中包含一个具有导出自动程序名称的文件。该文件包含将自动文件作为 Alexa 技能导入所必需的信息。

> **Note**
>
> Amazon Lex 和 Alexa Skills Kit 在以下方面是不同的：
>
> - Alexa Skills Kit 不支持 (用方括号 ([]) 括起来的) 会话属性。您需要更新使用会话属性的提示。
> - Alexa Skills Kit 不支持标点符号。您需要更新使用标点符号的表达。

将自动程序上传到 Alexa 技能

1. 登录开发者门户，网址为 https://developer.amazon.com/。
2. 在 Alexa 技能页面上，选择 Create Skill (创建技能)。
3. 在创建新技能页面上，输入技能名称和该技能的默认语言。确保已为该技能模型选择 Custom (自定义)，然后选择 Create Skill (创建技能)。
4. 确保 Start from scratch (从头开始) 处于选中状态，然后选择 Choose (选择)。
5. 从左侧菜单中，选择 JSON Editor (JSON 编辑器)。将从 Amazon Lex 中导出的 JSON 文件拖动至 JSON 编辑器中。
6. 选择 Save Model (保存模型) 以保存您的交互模型。

将架构上传到 Alexa 技能后，请进行任何必要的更改，以便用 Alexa 运行技能。有关创建 Alexa 技能的更多信息，请参阅 Alexa Skills Kit 中的 Use Skill Builder (Beta)。

其他示例：创建 Amazon Lex 机器人

以下各节提供了附带 step-by-step说明的其他 Amazon Lex 练习。

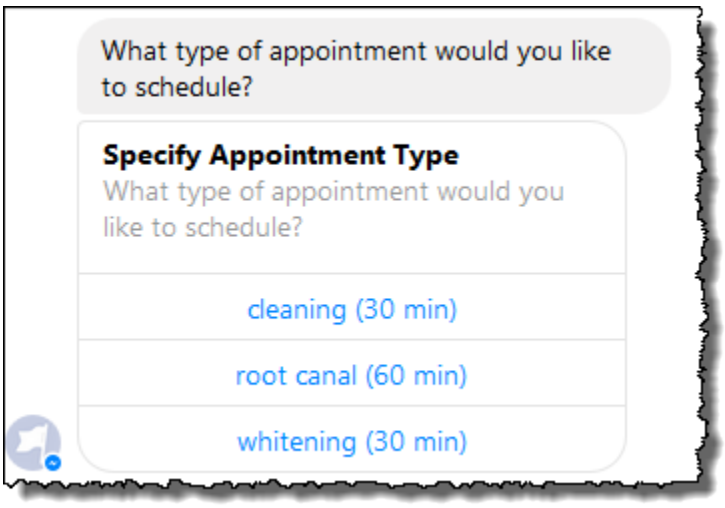
主题

安排预约

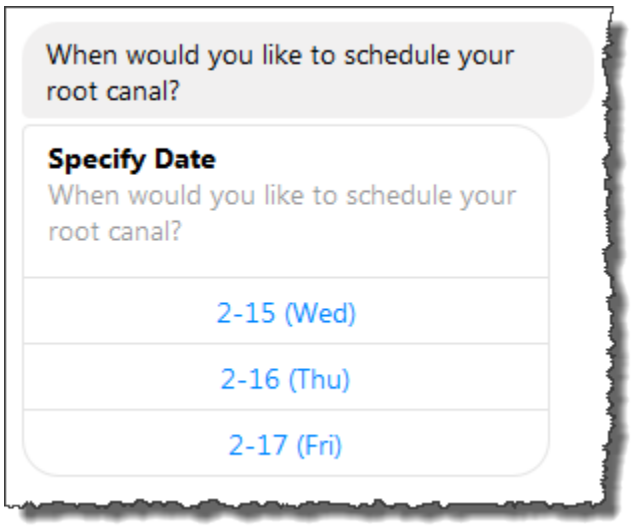
本练习中的示例自动程序将为牙医诊所安排预约。本示例还介绍了如何使用响应卡获取带有按钮的用户输入。具体来说，该示例介绍了如何在运行时动态生成响应卡。

您可以在构建时配置响应卡（也称为静态响应卡），也可以在 AWS Lambda 函数中动态生成响应卡。在本示例中，自动程序使用以下响应卡：

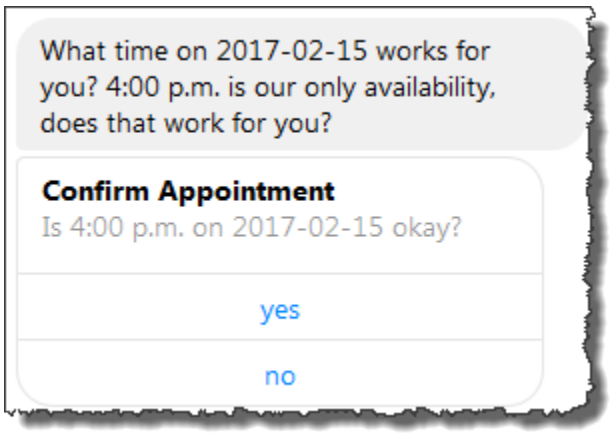
- 列出预约类型按钮的响应卡。有关示例，请参阅下图：



- 列出预约日期按钮的响应卡。有关示例，请参阅下图：



- 列出按钮以确认建议预约时间的响应卡。有关示例，请参阅下图：



可用的预约日期和时间有所不同，这就需要您在运行时生成响应卡。您可以使用 AWS Lambda 函数动态生成这些响应卡。Lambda 函数将在其发送给 Amazon Lex 的响应中返回响应卡。Amazon Lex 将在其发送给客户端的响应中包含响应卡。

如果客户端 (如 Facebook Messenger) 支持响应卡，则用户可以在按钮列表中进行选择或键入响应。如果不支持，则用户只需键入响应。

除了上述示例中显示的按钮以外，您还可在响应卡上包含映像、附件和其他有用的信息。有关响应卡的信息，请参阅 响应卡。

在本练习中，您将执行以下操作：

- 创建并测试机器人（使用 ScheduleAppointment 蓝图）。在本练习中，您将使用自动程序蓝图快速设置并测试自动程序。有关可用蓝图的列表，请参阅 Amazon Lex 和 AWS Lambda 蓝图。此自动程序预配置了一个目的 (MakeAppointment)。

- 创建和测试 Lambda 函数（使用 Lambda 提供的 lex-make-appointment-python蓝图）。您将 MakeAppointment 意图配置为将此 Lambda 函数用作代码挂钩，以执行初始化、验证和履行任务。

  > **Note**
  >
  > 所提供的示例 Lambda 函数将根据牙医预约的模型可用性显示动态对话。在实际应用程序中，您可以使用真正的日历安排预约。

- 更新 MakeAppointment 意图配置以将 Lambda 函数用作代码挂钩。然后，测试 end-to-end体验。
- 将安排预约机器人发布到 Facebook Messenger，以便您可以查看响应卡的实际操作（目前 Amazon Lex 控制台中的客户端不支持响应卡）。

以下几个部分提供了有关您在本练习中使用的蓝图的摘要信息。

主题

- 机器人蓝图概述 (ScheduleAppointment)
- Lambda 函数蓝图概述 () lex-make-appointment-python
- 步骤 1：创建 Amazon Lex 机器人
- 步骤 2：创建 Lambda 函数
- 步骤 3：更新目的：配置代码挂钩
- 步骤 4：在 Facebook Messenger 平台上部署自动程序
- 信息流详细信息

机器人蓝图概述 (ScheduleAppointment)

您用于为本练习创建机器人的 ScheduleAppointment 蓝图已预先配置了以下内容：

- 槽类型 – 一个称为 AppointmentTypeValue 的自定义槽类型，具有枚举值 root canal、cleaning 和 whitening。

- 目的 – 一个目的 (MakeAppointment)，按如下方式进行预配置：
  - 槽 – 目的配置了以下槽：
    - 槽 AppointmentType，属于 AppointmentTypes 自定义类型。
    - 槽 Date，属于 AMAZON.DATE 内置类型。
    - 槽 Time，属于 AMAZON.TIME 内置类型。
  - 表达 – 目的预配置了以下表达：
    - “我想进行预约”
    - “预约”
    - “预订 {AppointmentType}”

    如果用户表达出上述任意一种说法，Amazon Lex 将确定意图是 MakeAppointment，然后使用提示引发插槽数据。
  - 提示 – 目的预配置了以下提示：
    - 用于 AppointmentType 槽的提示 –“您想要安排哪种类型的预约？”
    - 提示进入时Date段 — “我应该什么时候安排你的 {AppointmentType}？”
    - 提示输入时Time段 — “你想在什么时候安排 {AppointmentType}？” 以及

      “{Date} 的什么时间？”
    - 确认提示 –“{Time} 可以预约，我应该直接帮您预约吗？”
    - 取消消息 –“好的，我不会安排预约。”

Lambda 函数蓝图概述 () lex-make-appointment-python

Lambda 函数蓝图 (lex-make-appointment-python) 是您使用机器人蓝图创建的机器人的代码挂钩。ScheduleAppointment

此 Lambda 函数蓝图代码可以执行初始化/验证和履行任务。

- Lambda 函数代码显示了基于牙医预约示例可用性的动态对话（在实际应用程序中，您可以使用日历）。对于用户指定的日期或时间，代码按如下方式配置：
  - 如果没有可用的预约，Lambda 函数将返回指示 Amazon Lex 提示用户选择其他时间或日期的响应（通过将 dialogAction 类型设置为 ElicitSlot））。有关更多信息，请参阅 响应格式。

- 如果在指定的时间或日期只有一个可用预约，Lambda 函数将在响应中建议可用的时间，并通过将响应中的 `dialogAction` 设置为 `ConfirmIntent` 来指示 Amazon Lex 获取用户确认。这说明了如何通过主动建议可用的预约时间来改善用户体验。
- 如果有多个可用预约，Lambda 函数将在返回到 Amazon Lex 的响应中提供可用时间列表。Amazon Lex 将在返回到客户端的响应中提供来自 Lambda 函数的消息。

- 履行代码挂钩之后，Lambda 函数将返回表示预约已安排的摘要消息（即意图已履行）。

> **Note**
>
> 在本示例中，我们将展示如何使用响应卡。Lambda 函数将构建响应卡并将其返回给 Amazon Lex。该响应卡将可用的日期和时间作为可供选择的按钮一一列出。使用 Amazon Lex 控制台提供的客户端测试机器人时，您无法查看响应卡。要进行查看，必须将自动程序与消息收发平台 (如 Facebook Messenger) 集成。有关说明，请参阅 将 Amazon Lex 机器人与 Facebook Messenger 集成。有关响应卡的更多信息，请参阅 管理消息 。

Amazon Lex 调用 Lambda 函数时，会将事件数据作为输入进行传递。其中一个事件字段是 `invocationSource`，Lambda 函数会使用该字段在输入验证和履行活动之间进行选择。有关更多信息，请参阅 输入事件格式。

下一个步骤

步骤 1：创建 Amazon Lex 机器人

步骤 1：创建 Amazon Lex 机器人

在本节中，您将使用 Amazon Lex 控制台中提供的 ScheduleAppointment蓝图创建 Amazon Lex 机器人。

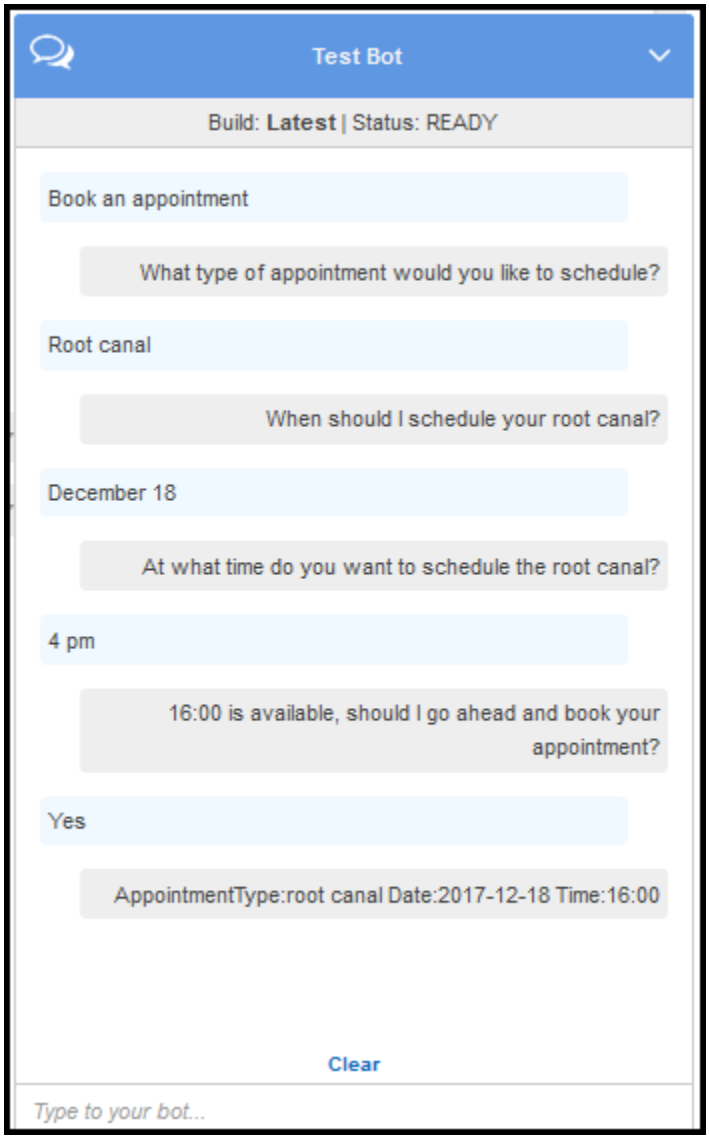
1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 在 Bots 页面上，选择 Create。
3. 在 Create your Lex bot 页面上，执行以下操作：
   - 选择 ScheduleAppointment 蓝图。
   - 保留默认的机器人名称 (ScheduleAppointment)。
4. 选择创建。

此步骤将保存并构建自动程序。在构建过程中，控制台将向 Amazon Lex 发送以下请求：

- 创建槽类型的新版本 (来自 $LATEST 版本)。有关本自动程序蓝图中定义的槽类型的信息，请参阅 机器人蓝图概述 (ScheduleAppointment)。
- 创建 `MakeAppointment` 目的的版本 (来自 $LATEST 版本)。在某些情况下，控制台会先发送有关 `update` API 操作的请求，然后再创建新版本。
- 更新自动程序的 $LATEST 版本。

  此时，Amazon Lex 将为机器人构建机器学习模型。当您在控制台中测试机器人时，控制台将使用运行时 API 将用户输入发送回 Amazon Lex。然后，Amazon Lex 将使用机器学习模型解释用户输入。

5. 控制台显示 ScheduleAppointment 机器人。在 Editor 选项卡上，查看预配置目的 (`MakeAppointment`) 的详细信息。
6. 在测试窗口中测试机器人。使用以下屏幕截图与您的自动程序进行测试对话：



请注意以下几点：

- 从初始用户输入 (“预约”) 中，自动程序可推断出目的 (MakeAppointment)。
- 然后，自动程序将使用已配置的提示来获取用户的槽数据。
- 自动程序蓝图拥有配置了以下确认提示的 MakeAppointment 目的：

  ```
  {Time} is available, should I go ahead and book your appointment?
  ```

  用户提供所有插槽数据后，Amazon Lex 将向客户端返回响应，并将确认提示作为消息包含在其中。客户端将向用户显示消息：

```
16:00 is available, should I go ahead and book your appointment?
```

请注意，自动程序将接受所有预约日期和时间值，因为您没有任何代码来初始化或验证用户数据。在下一节中，您将添加 Lambda 函数来完成此操作。

下一个步骤

步骤 2：创建 Lambda 函数

步骤 2：创建 Lambda 函数

在本节中，您将使用 Lambda 控制台中提供的蓝图 (lex-make-appointment-python) 创建 Lambda 函数。您还将使用控制台提供的示例 Amazon Lex 事件数据调用 Lambda 函数，以此来对该函数进行测试。

1. 登录 AWS Management Console 并打开 AWS Lambda 控制台，网址为https://console.aws.amazon.com/lambda/。
2. 选择 Create a Lambda function (创建 Lambda 函数)。
3. 在选择蓝图中，键入**lex**以查找蓝图，然后选择lex-make-appointment-python蓝图。
4. 按如下方式配置 Lambda 函数。
   - 键入 Lambda 函数名称 (MakeAppointmentCodeHook)。
   - 对于角色，选择 Create a new role from template(s)，然后键入角色名称。
   - 保留其他默认值。
5. 选择创建函数。
6. 如果您使用的是英语 (美国) (en-US) 以外的区域设置，请按照更新特定区域设置的蓝图中所述更新意图名称。
7. 测试 Lambda 函数
   a. 选择 Actions，然后选择 Configure test event。
   b. 从 Sample event template 列表中选择 Lex-Make Appointment (preview)。此示例事件使用 Amazon Lex request/response model, with values set to match a request from your Amazon Lex bot. For information about the Amazon Lex request/response 模型，请参阅使用 Lambda 函数。

c. 选择保存并测试。

d. 验证 Lambda 函数已成功执行。在此情况下，此响应与 Amazon Lex 响应模型相匹配。

下一个步骤

步骤 3：更新目的：配置代码挂钩

步骤 3：更新目的：配置代码挂钩

在本节中，您将更新 `MakeAppointment` 意图的配置，以将 Lambda 函数用作验证和履行活动的代码挂钩。

1. 在 Amazon Lex 控制台中，选择该 ScheduleAppointment 机器人。控制台显示MakeAppointment意图。按如下方式修改目的配置。

   > **Note**
   >
   > 您只能更新任意 Amazon Lex 资源的 $LATEST 版本，包括意图。确保将目的版本设置为 $LATEST。您尚未发布自动程序的版本，因此它在控制台中仍然应是 $LATEST 版本。

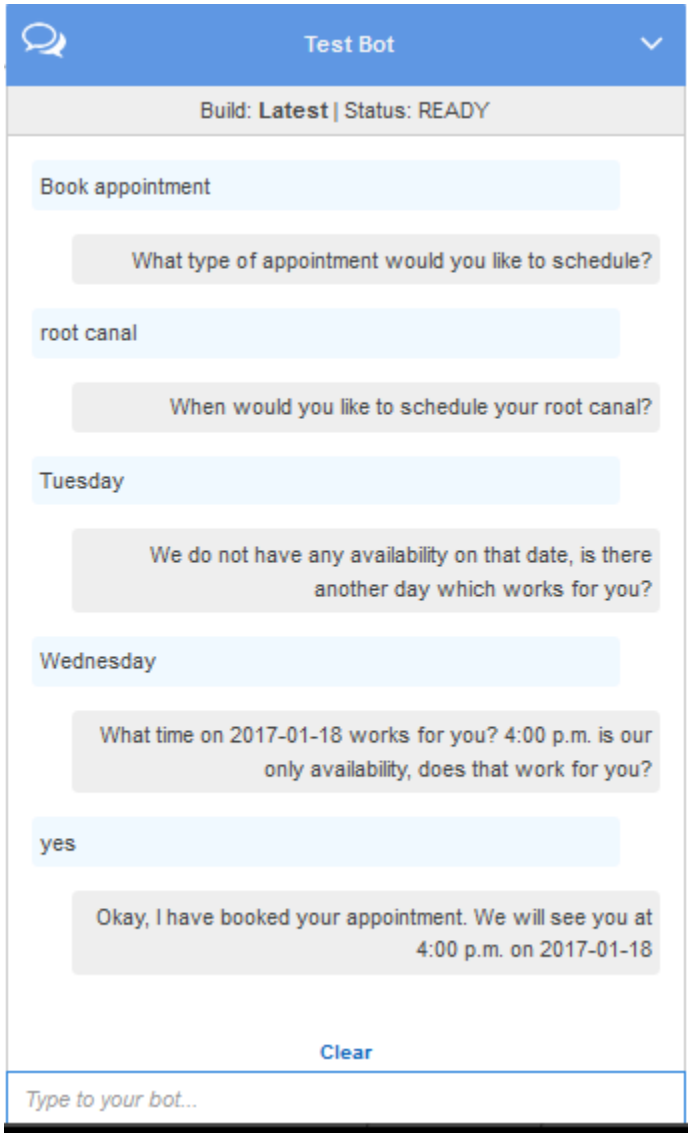
   a. 在选项部分，选择初始化和验证代码挂钩，然后从列表中选择 Lambda 函数。

   b. 在履行部分，选择 AWS Lambda 函数，然后从列表中选择 Lambda 函数。

   c. 选择 Goodbye message 并键入消息。

2. 选择 Save，然后选择 Build。

3. 测试机器人，如下图所示：



下一个步骤

步骤 4：在 Facebook Messenger 平台上部署自动程序

步骤 4：在 Facebook Messenger 平台上部署自动程序

在上一节中，您在 Amazon Lex 控制台中使用客户端测试了该 ScheduleAppointment 机器人。目前，Amazon Lex 控制台不支持响应卡。要测试动态生成的自动程序支持的响应卡，请在 Facebook Messenger 平台上部署自动程序并进行测试。

有关说明，请参阅 将 Amazon Lex 机器人与 Facebook Messenger 集成。

下一个步骤

信息流详细信息

信息流详细信息

ScheduleAppointment 自动程序蓝图主要展示动态生成的响应卡的用法。本练习中的 Lambda 函数将在其发送给 Amazon Lex 的响应中包含响应卡。Amazon Lex 将在其发送给客户端的回复中包含响应卡。本部分介绍了以下两点：

- 客户端和 Amazon Lex 之间的数据流。

  本节假定客户端使用 PostText 运行时 API 向 Amazon Lex 发送请求，并显示相应的请求/响应详细信息。有关 PostText 运行时 API 的更多信息，请参阅 PostText。

  > **Note**
  >
  > 有关客户端和 Amazon Lex（客户端在其中使用 PostContent API）之间的信息流示例，请参阅步骤 2a (可选)：查看语音信息流的详细信息 (控制台)。

- Amazon Lex 和 Lambda 函数之间的数据流。有关更多信息，请参阅 Lambda 函数输入事件和响应格式。

> **Note**
>
> 此示例假定您正在使用 Facebook Messenger 客户端，该客户端不会将请求中的会话属性传递给 Amazon Lex。因此，本部分中的示例请求显示为空 sessionAttributes。如果您使用 Amazon Lex 控制台提供的客户端测试机器人，则客户端将包含这些会话属性。

本部分描述了每个用户输入后会发生的情况。

1. 用户：类型 **Book an appointment**。

   a. 客户端 (控制台) 将向 Amazon Lex 发送以下 PostContent 请求：

```
POST /bot/ScheduleAppointment/alias/$LATEST/
user/bijt6rovckwecnzesbthrr1d7lv3ja3n/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"book appointment",
   "sessionAttributes":{}
}
```

请求 URI 和正文都将向 Amazon Lex 提供信息：

- 请求 URI-提供机器人名称 (ScheduleAppointment)、机器人别名 ($LATEST) 和用户名 ID。尾部 `text` 表示这是 `PostText` (而不是 `PostContent`) API 请求。
- 请求正文 – 包括用户输入 (`inputText`) 和空 `sessionAttributes`。

b. 在 `inputText` 中，Amazon Lex 可检测意图 (`MakeAppointment`)。该服务将调用 Lambda 函数（配置为代码挂钩），以通过传递以下事件执行初始化和验证。有关详细信息，请参阅输入事件格式。

```
{
    "currentIntent": {
        "slots": {
            "AppointmentType": null,
            "Date": null,
            "Time": null
        },
        "name": "MakeAppointment",
        "confirmationStatus": "None"
    },
    "bot": {
        "alias": null,
        "version": "$LATEST",
        "name": "ScheduleAppointment"
    },
    "userId": "bijt6rovckwecnzesbthrr1d7lv3ja3n",
    "invocationSource": "DialogCodeHook",
    "outputDialogMode": "Text",
    "messageVersion": "1.0",
    "sessionAttributes": {}
}
```

除了客户端发送的信息以外，Amazon Lex 还包含以下数据：

- `currentIntent` – 提供当前的目的信息。
- `invocationSource` — 表明 Lambda 函数调用的目的。在本例中，目的是执行用户数据初始化和验证。（Amazon Lex 知道用户尚未提供所有插槽数据来履行意图。）
- `messageVersion` — Amazon Lex 当前只支持 1.0 版。

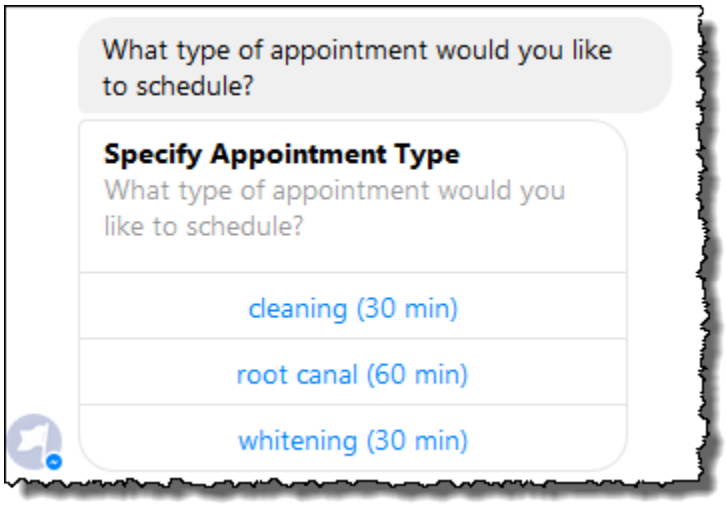
c. 此时，所有槽值均为空值 (没有需要验证的内容)。Lambda 函数会向 Amazon Lex 返回以下响应，以指示服务引发 `AppointmentType` 插槽的信息。有关响应格式的信息，请参阅 响应格式。

```
{
    "dialogAction": {
        "slotToElicit": "AppointmentType",
        "intentName": "MakeAppointment",
        "responseCard": {
            "genericAttachments": [
                {
                    "buttons": [
                        {
                            "text": "cleaning (30 min)",
                            "value": "cleaning"
                        },
                        {
                            "text": "root canal (60 min)",
                            "value": "root canal"
                        },
                        {
                            "text": "whitening (30 min)",
                            "value": "whitening"
                        }
                    ],
                    "subTitle": "What type of appointment would you like to
 schedule?",
                    "title": "Specify Appointment Type"
                }
            ],
            "version": 1,
            "contentType": "application/vnd.amazonaws.card.generic"
        },
        "slots": {
            "AppointmentType": null,
```

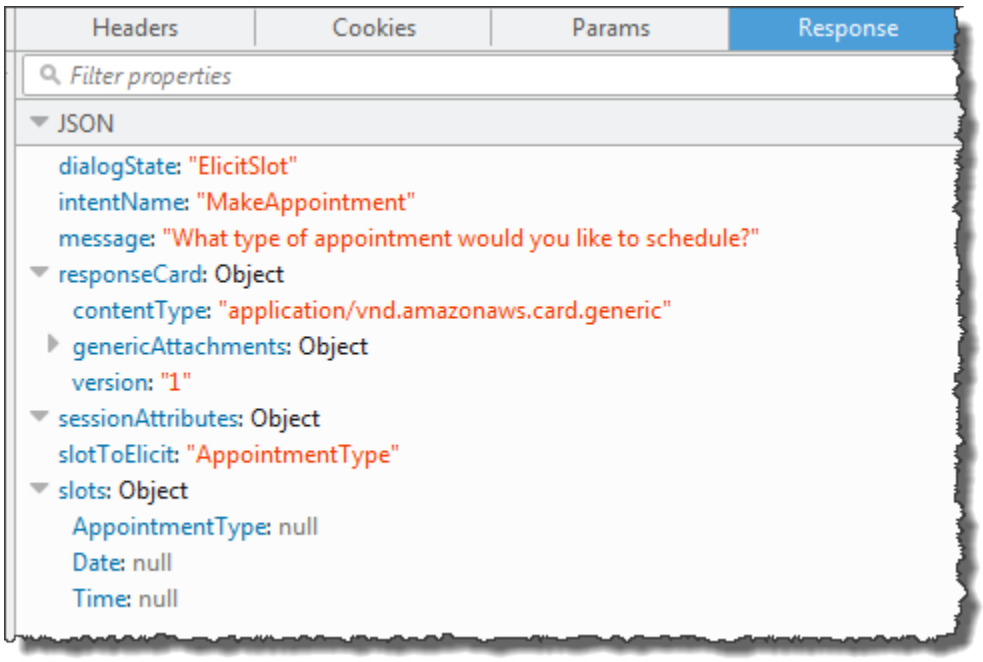
```
            "Date": null,
            "Time": null
        },
        "type": "ElicitSlot",
        "message": {
            "content": "What type of appointment would you like to schedule?",
            "contentType": "PlainText"
        }
    },
    "sessionAttributes": {}
}
```

响应包含 dialogAction 和 sessionAttributes 字段。此外，dialogAction 字段将返回以下字段：

- type — 通过将此字段设置为 ElicitSlot，Lambda 函数将指示 Amazon Lex 引发 slotToElicit 字段中指定的插槽的值。Lambda 函数还将提供要传达给用户的 message。
- responseCard — 标识 AppointmentType 插槽的可能值列表。支持响应卡的客户端（如 Facebook Messenger）将显示响应卡，以允许用户选择一个预约类型，如下图所示：



d. 如来自 Lambda 函数的响应中的 dialogAction.type 所示，Amazon Lex 会将以下响应发送回客户端：



客户端将读取响应，然后显示消息："您想要安排哪种类型的预约？" 以及响应卡 (如果客户端支持响应卡)。

2. 用户：根据客户端，用户有两个选项：

   - 如果显示响应卡，则选择 root canal (60 min) 或键入 **`root canal`**。
   - 如果客户端不支持响应卡，则键入 **`root canal`**。

   a. 客户端将向 Amazon Lex 发送以下 `PostText` 请求（我们添加了换行符以便于阅读）：

```
POST /bot/BookTrip/alias/$LATEST/user/bijt6rovckwecnzesbthrr1d7lv3ja3n/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText": "root canal",
    "sessionAttributes": {}
}
```

   b. Amazon Lex 将调用 Lambda 函数，以通过发送以下事件作为参数来验证用户数据：

```
{
    "currentIntent": {
        "slots": {
```

```
            "AppointmentType": "root canal",
            "Date": null,
            "Time": null
        },
        "name": "MakeAppointment",
        "confirmationStatus": "None"
    },
    "bot": {
        "alias": null,
        "version": "$LATEST",
        "name": "ScheduleAppointment"
    },
    "userId": "bijt6rovckwecnzesbthrr1d7lv3ja3n",
    "invocationSource": "DialogCodeHook",
    "outputDialogMode": "Text",
    "messageVersion": "1.0",
    "sessionAttributes": {}
}
```

在事件数据中，请注意以下事项：

- invocationSource 仍然是 DialogCodeHook。在此步骤中，我们只验证用户数据。
- Amazon Lex 会将 AppointmentType 插槽中的 currentIntent.slots 字段设置为 root canal。
- Amazon Lex 将仅在客户端和 Lambda 函数之间传递 sessionAttributes 字段。

c. Lambda 函数将验证用户输入并向 Amazon Lex 返回以下响应，指示服务引发预约日期的值。

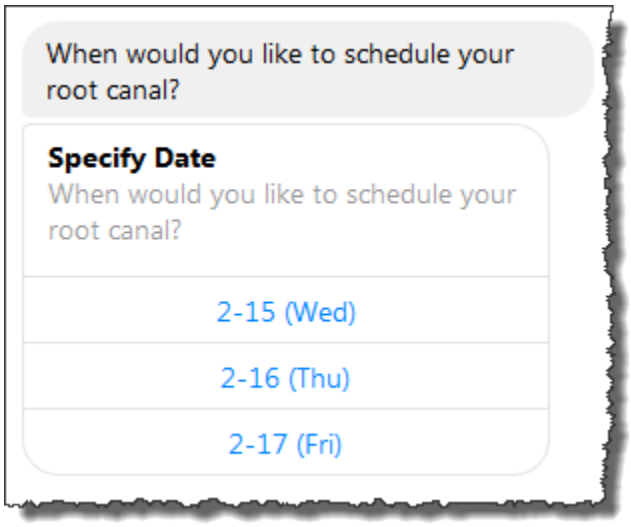
```
{
    "dialogAction": {
        "slotToElicit": "Date",
        "intentName": "MakeAppointment",
        "responseCard": {
            "genericAttachments": [
                {
                    "buttons": [
                        {
                            "text": "2-15 (Wed)",
                            "value": "Wednesday, February 15, 2017"
                        },
                        {
                            "text": "2-16 (Thu)",
                            "value": "Thursday, February 16, 2017"
```

```
                        },
                        {
                            "text": "2-17 (Fri)",
                            "value": "Friday, February 17, 2017"
                        },
                        {
                            "text": "2-20 (Mon)",
                            "value": "Monday, February 20, 2017"
                        },
                        {
                            "text": "2-21 (Tue)",
                            "value": "Tuesday, February 21, 2017"
                        }
                    ],
                    "subTitle": "When would you like to schedule your root
 canal?",
                    "title": "Specify Date"
                }
            ],
            "version": 1,
            "contentType": "application/vnd.amazonaws.card.generic"
        },
        "slots": {
            "AppointmentType": "root canal",
            "Date": null,
            "Time": null
        },
        "type": "ElicitSlot",
        "message": {
            "content": "When would you like to schedule your root canal?",
            "contentType": "PlainText"
        }
    },
    "sessionAttributes": {}
}
```

同样，响应包含 dialogAction 和 sessionAttributes 字段。此外，dialogAction 字段将返回以下字段：

- type — 通过将此字段设置为 ElicitSlot，Lambda 函数将指示 Amazon Lex 引发 slotToElicit 字段中指定的插槽的值。Lambda 函数还将提供要传达给用户的 message。

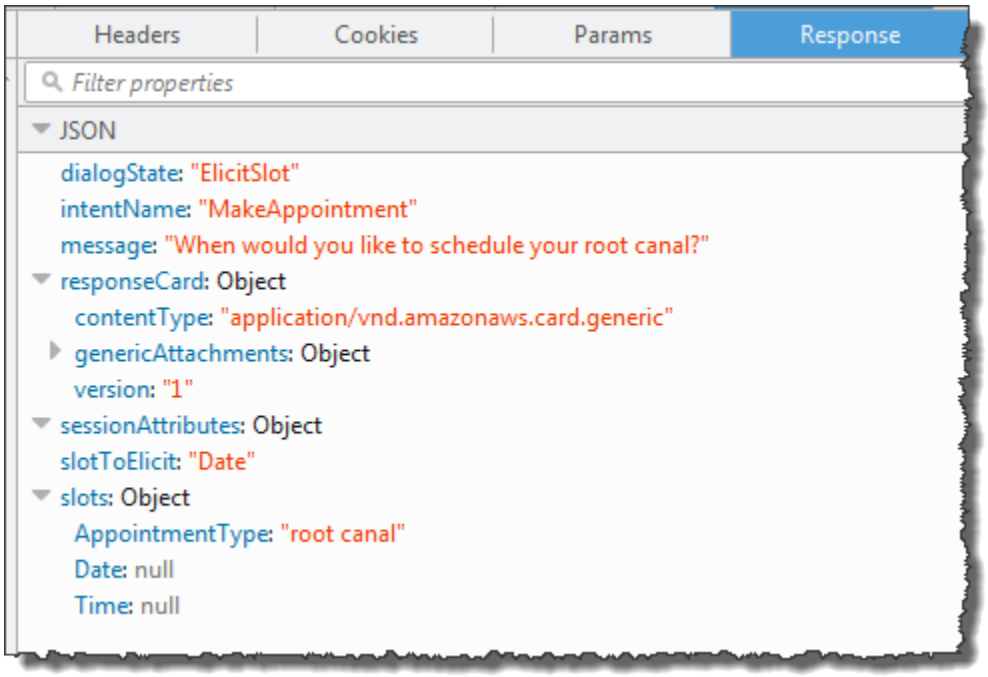
- `responseCard` — 标识 `Date` 插槽的可能值列表。支持响应卡的客户端 (如 Facebook Messenger) 将显示响应卡，以允许用户选择一个预约日期，如下图所示：



尽管 Lambda 函数返回了五个日期，但客户端 (Facebook Messenger) 对于一张响应卡只能使用三个按钮。因此，您只能在屏幕截图中看到前三个值。

这些日期在 Lambda 函数中是硬编码值。在生产应用程序中，您可以使用日历实时获得可用的日期。由于日期是动态的，因此您必须使用 Lambda 函数动态生成响应卡。

d. Amazon Lex 将注意到 `dialogAction.type` 并向客户端返回以下响应，其中包括来自 Lambda 函数的响应中的信息。



客户端将显示消息：When would you like to schedule your root canal? 以及响应卡（如果客户端支持响应卡）。

3. 用户：类型 **Thursday**。

   a. 客户端将向 Amazon Lex 发送以下 PostText 请求（我们添加了换行符以便于阅读）：

```
POST /bot/BookTrip/alias/$LATEST/user/bijt6rovckwecnzesbthrr1d7lv3ja3n/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText": "Thursday",
    "sessionAttributes": {}
}
```

   b. Amazon Lex 将调用 Lambda 函数，以通过发送以下事件作为参数来验证用户数据：

```
{
    "currentIntent": {
        "slots": {
            "AppointmentType": "root canal",
            "Date": "2017-02-16",
            "Time": null
        },
        "name": "MakeAppointment",
        "confirmationStatus": "None"
    },
    "bot": {
        "alias": null,
        "version": "$LATEST",
        "name": "ScheduleAppointment"
    },
    "userId": "u3fpr9gghj02zts7y5tpq5mm4din2xqy",
    "invocationSource": "DialogCodeHook",
    "outputDialogMode": "Text",
    "messageVersion": "1.0",
    "sessionAttributes": {}
}
```

   在事件数据中，请注意以下事项：

- invocationSource 仍然是 DialogCodeHook。在此步骤中，我们只验证用户数据。
- Amazon Lex 会将 Date 插槽中的 currentIntent.slots 字段设置为 2017-02-16。
- Amazon Lex 将仅在客户端和 Lambda 函数之间传递 sessionAttributes。

c. Lambda 函数将验证用户输入。此时，Lambda 函数将确定指定日期没有可用的预约。然后，它将向 Amazon Lex 返回以下响应，指示服务再次引发预约日期的值。

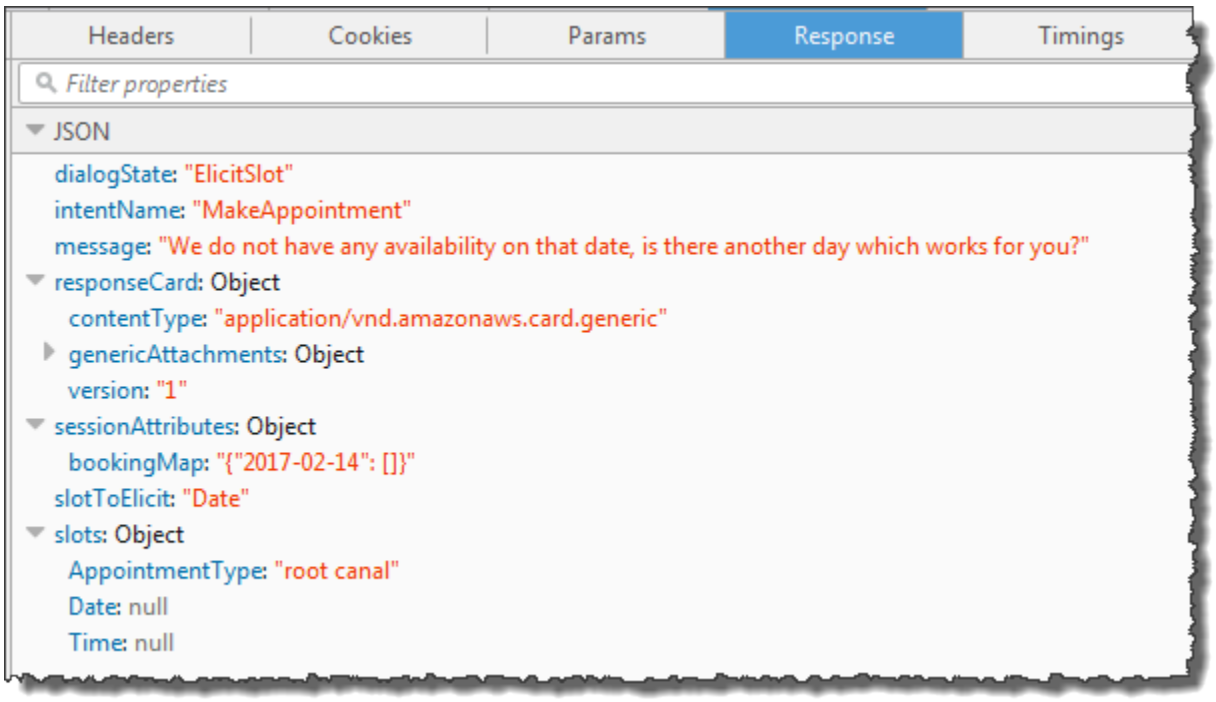
```
{
    "dialogAction": {
        "slotToElicit": "Date",
        "intentName": "MakeAppointment",
        "responseCard": {
            "genericAttachments": [
                {
                    "buttons": [
                        {
                            "text": "2-15 (Wed)",
                            "value": "Wednesday, February 15, 2017"
                        },
                        {
                            "text": "2-17 (Fri)",
                            "value": "Friday, February 17, 2017"
                        },
                        {
                            "text": "2-20 (Mon)",
                            "value": "Monday, February 20, 2017"
                        },
                        {
                            "text": "2-21 (Tue)",
                            "value": "Tuesday, February 21, 2017"
                        }
                    ],
                    "subTitle": "When would you like to schedule your root
 canal?",
                    "title": "Specify Date"
                }
            ],
            "version": 1,
            "contentType": "application/vnd.amazonaws.card.generic"
        },
        "slots": {
            "AppointmentType": "root canal",
```

```
            "Date": null,
            "Time": null
        },
        "type": "ElicitSlot",
        "message": {
            "content": "We do not have any availability on that date, is there
 another day which works for you?",
            "contentType": "PlainText"
        }
    },
    "sessionAttributes": {
     "bookingMap": "{\"2017-02-16\": []}"
   }
}
```

同样，响应包含 dialogAction 和 sessionAttributes 字段。此外，dialogAction 将返回以下字段：

- dialogAction 字段：
  - type — Lambda 函数将此值设置为 ElicitSlot 并将 slotToElicit 字段重置为 Date。Lambda 函数还将提供要传达给用户的相应 message。
  - responseCard – 返回 Date 槽的值列表。
- sessionAttributes — 此时，Lambda 函数包含 bookingMap 会话属性。其值就是预约的请求日期和可用预约 (空对象表示无可用预约)。

d. Amazon Lex 将注意到 dialogAction.type 并向客户端返回以下响应，其中包括来自 Lambda 函数的响应中的信息。



客户端将显示消息：We do not have any availability on that date, is there another day which works for you? 以及响应卡（如果客户端支持响应卡）。

4. 用户：根据客户端，用户有两个选项：

   - 如果显示响应卡，则选择 2-15 (Wed) 或键入 **Wednesday**。
   - 如果客户端不支持响应卡，则键入 **Wednesday**。

   a. 客户端将向 Amazon Lex 发送以下 `PostText` 请求：

```
POST /bot/BookTrip/alias/$LATEST/user/bijt6rovckwecnzesbthrr1d7lv3ja3n/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText": "Wednesday",
    "sessionAttributes": {
   }
}
```

> **Note**
>
> Facebook Messenger 客户端不设置任何会话属性。如果您要维护请求之间的会话状态，必须使用 Lambda 函数完成操作。在实际应用程序中，您可以在后端数据库中维护这些会话属性。

b. Amazon Lex 将调用 Lambda 函数，以通过发送以下事件作为参数来验证用户数据：

```
{
    "currentIntent": {
        "slots": {
            "AppointmentType": "root canal",
            "Date": "2017-02-15",
            "Time": null
        },
        "name": "MakeAppointment",
        "confirmationStatus": "None"
    },
    "bot": {
        "alias": null,
        "version": "$LATEST",
        "name": "ScheduleAppointment"
    },
    "userId": "u3fpr9gghj02zts7y5tpq5mm4din2xqy",
    "invocationSource": "DialogCodeHook",
    "outputDialogMode": "Text",
    "messageVersion": "1.0",
    "sessionAttributes": {
    }
}
```

Amazon Lex 通过将 `Date` 插槽设置为 `2017-02-15` 更新了 `currentIntent.slots`。

c. Lambda 函数将验证用户输入并向 Amazon Lex 返回以下响应，从而指示服务引发预约时间的值。

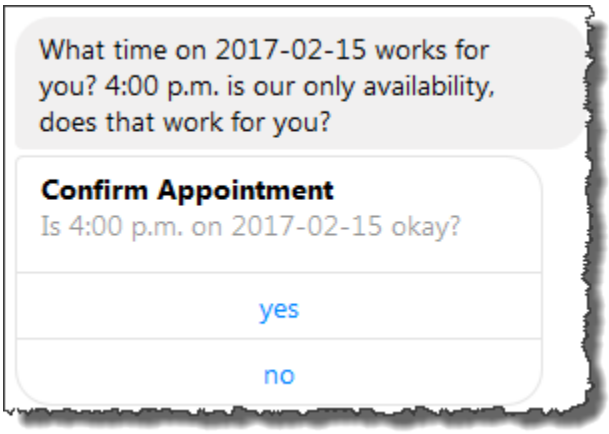
```
{
    "dialogAction": {
        "slots": {
            "AppointmentType": "root canal",
            "Date": "2017-02-15",
```

```
            "Time": "16:00"
        },
        "message": {
            "content": "What time on 2017-02-15 works for you? 4:00 p.m. is our
 only availability, does that work for you?",
            "contentType": "PlainText"
        },
        "type": "ConfirmIntent",
        "intentName": "MakeAppointment",
        "responseCard": {
            "genericAttachments": [
                {
                    "buttons": [
                        {
                            "text": "yes",
                            "value": "yes"
                        },
                        {
                            "text": "no",
                            "value": "no"
                        }
                    ],
                    "subTitle": "Is 4:00 p.m. on 2017-02-15 okay?",
                    "title": "Confirm Appointment"
                }
            ],
            "version": 1,
            "contentType": "application/vnd.amazonaws.card.generic"
        }
    },
    "sessionAttributes": {
        "bookingMap": "{\"2017-02-15\": [\"10:00\", \"16:00\", \"16:30\"]}"
    }
}
```

同样，响应包含 dialogAction 和 sessionAttributes 字段。此外，dialogAction 将返回以下字段：

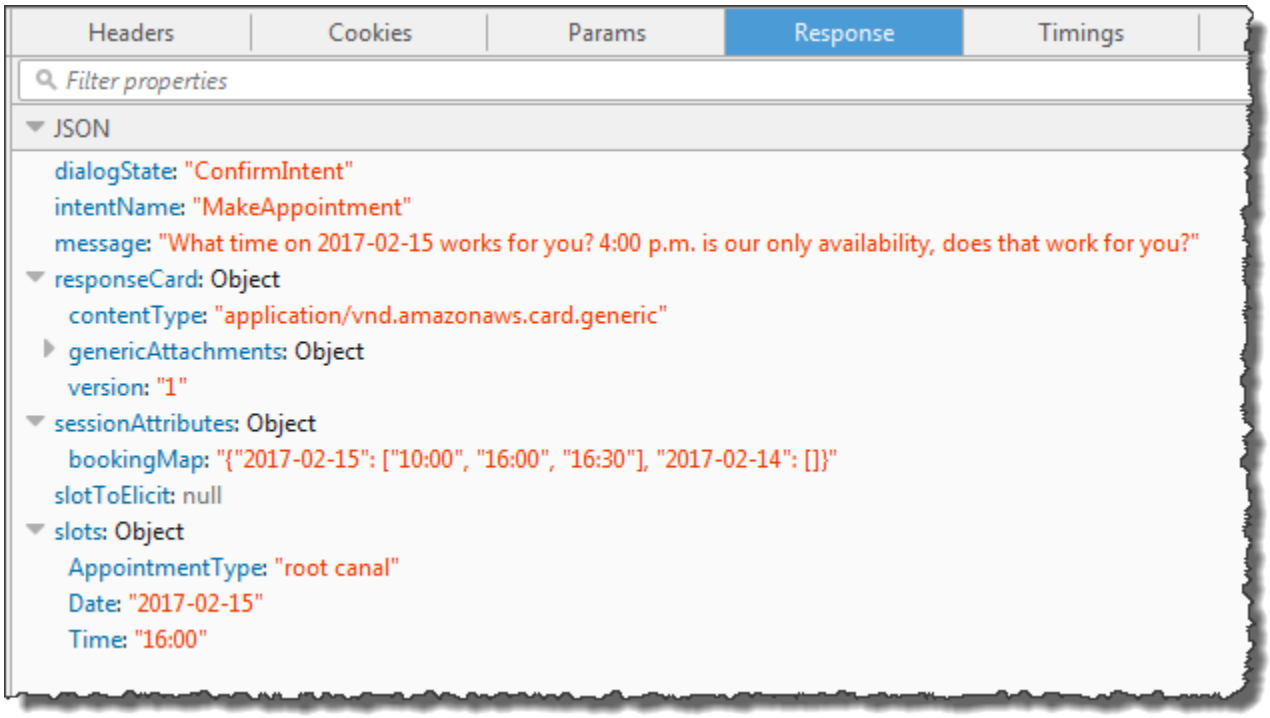
- dialogAction 字段：
  - type — Lambda 函数将此值设置为 ConfirmIntent，从而指示 Amazon Lex 获取 message 中建议的预约时间的用户确认。

- `responseCard` — 返回一个供用户选择的“是”或“否”值列表。如果客户端支持响应卡，则会显示响应卡，如下面的示例所示：



- `sessionAttributes` — Lambda 函数将设置 bookingMap 会话属性，将其值设置为预约日期以及该日期的可用预约。在本示例中，这些是 30 分钟的预约。对于需要一个小时的牙根管手术，只能在下午 4 点预约。

d. 如 Lambda 函数响应中的 `dialogAction.type` 所示，Amazon Lex 将向客户端返回以下响应：



客户端会显示以下消息：2017 年 2 月 15 日的什么时间适合您？我们只有下午 4:00 有空，这个时间适合您吗？

5. 用户：选择 **yes**。

   Amazon Lex 将使用以下事件数据调用 Lambda 函数。由于用户回复了 **yes**，Amazon Lex 将 `confirmationStatus` 设置为 `Confirmed`，并将 `currentIntent.slots` 中的 `Time` 字段设置为 `4 p.m.`。

```
{
    "currentIntent": {
        "slots": {
            "AppointmentType": "root canal",
            "Date": "2017-02-15",
            "Time": "16:00"
        },
        "name": "MakeAppointment",
        "confirmationStatus": "Confirmed"
    },
    "bot": {
        "alias": null,
        "version": "$LATEST",
        "name": "ScheduleAppointment"
    },
    "userId": "u3fpr9gghj02zts7y5tpq5mm4din2xqy",
    "invocationSource": "FulfillmentCodeHook",
    "outputDialogMode": "Text",
    "messageVersion": "1.0",
    "sessionAttributes": {
   }
}
```

   由于 `confirmationStatus` 已获得确认，Lambda 函数将处理意图（预约牙医）并向 Amazon Lex 返回以下响应：

```
{
    "dialogAction": {
        "message": {
            "content": "Okay, I have booked your appointment.  We will see you at
 4:00 p.m. on 2017-02-15",
            "contentType": "PlainText"
        },
        "type": "Close",
```

```
        "fulfillmentState": "Fulfilled"
    },
    "sessionAttributes": {
        "formattedTime": "4:00 p.m.",
        "bookingMap": "{\"2017-02-15\": [\"10:00\"]}"
    }
}
```

请注意以下几点：

- Lambda 函数已更新 `sessionAttributes`。
- `dialogAction.type` 设置为 `Close`，这将指示 Amazon Lex 不要期待用户响应。
- `dialogAction.fulfillmentState` 已设置为 `Fulfilled`，表示已成功完成目的。

客户端将显示消息：好的，我帮您预订好了。2017 年 2 月 15 日下午 4:00 见。

预订旅程

本示例介绍了如何创建可以支持多个目的的自动程序。本示例还介绍了如何使用会话属性实现跨目的信息共享。创建机器人后，您可以在 Amazon Lex 控制台中使用测试客户端来测试该机器人 (BookTrip)。客户端针对每个用户输入使用 PostText 运行时 API 操作向 Amazon Lex 发送请求。

本示例中的 BookTrip 机器人配置有两个意图（BookHotel 和 BookCar）。例如，假设用户首先预订酒店。在交互过程中，用户提供入住日期、地点和入住天数等信息。实现目的后，客户端可以使用会话属性保留这一信息。有关会话属性的更多信息，请参阅 PostText。

现在假设用户继续预订汽车。使用用户在之前的 BookHotel intent 中提供的信息（即目的地城市以及入住和退房日期），您配置为初始化和验证意图的代码挂钩（Lambda 函数）初始化 BookCar 意图的空位数据（即目的地、接送城市、提货日期和返回日期）。 BookCar 这一过程表明，跨目的信息共享让您可以构建能够与用户展开动态会话的自动程序。

在本示例中，我们使用以下会话属性。只有客户端和 Lambda 函数可以设置和更新会话属性。Amazon Lex 只在客户端和 Lambda 函数之间传递这些会话属性。Amazon Lex 不会保留或修改任何会话属性。

- `currentReservation` — 包含正在进行的预订的插槽数据及其他相关信息。下面是从客户端发到 Amazon Lex 的一个示例请求。请求正文中包含 `currentReservation` 会话属性。

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"Chicago",
   "sessionAttributes":{
       "currentReservation":"{\"ReservationType\":\"Hotel\",
                              \"Location\":\"Moscow\",
                              \"RoomType\":null,
                              \"CheckInDate\":null,
                              \"Nights\":null}"
   }
}
```

- `lastConfirmedReservation` — 包含前一个意图的类似信息（如果有）。例如，如果用户预订了酒店，然后正在预订汽车，则此会话属性会存储先前 BookHotel 意图的车位数据。

- `confirmationContext` — Lambda 函数在根据前一个预订的插槽数据（如果有）来预填充某些插槽数据时，会将本属性设置为 `AutoPopulate`。这可以实现跨目的信息共享。例如，如果用户之前预订了酒店，现在需要预订汽车，则 Amazon Lex 可以提示用户确认（或拒绝）在预订酒店的同一城市和相同日期预订汽车

在本练习中，您使用蓝图创建 Amazon Lex 机器人和 Lambda 函数。有关蓝图的更多信息，请参阅Amazon Lex 和 AWS Lambda 蓝图。

下一个步骤

步骤 1：查看本练习中使用的蓝图

步骤 1：查看本练习中使用的蓝图

主题

- 机器人蓝图概述 (BookTrip)
- Lambda 函数蓝图概述 () lex-book-trip-python

机器人蓝图概述 (BookTrip)

您用于创建自动程序的蓝图 (BookTrip) 提供以下预配置：

- 槽类型 – 两种自定义槽类型：
  - RoomTypes，使用枚举值 king、queen 和 deluxe，用于 BookHotel 目的。
  - CarTypes，使用枚举值 economy、standard、midsize、full size、luxury 和 minivan，用于 BookCar 目的。

- 意图 1 (BookHotel) — 其预配置如下：
  - 预配置的槽
    - RoomType，属于 RoomTypes 自定义槽类型
    - Location，属于 AMAZON.US_CITY 内置槽类型
    - CheckInDate，属于 AMAZON.DATE 内置槽类型
    - Nights，属于 AMAZON.NUMBER 内置槽类型
  - 预配置的表达
    - “预订酒店”
    - “我想预订酒店”
    - “在 {Location} 预订 {Nights} 晚”

    如果用户表达出上述任意一种说法，Amazon Lex 就会确定用户的意图是 BookHotel，然后提示用户提供插槽数据。
  - 预配置的提示
    - 针对 Location 槽的提示 –“您要住在哪个城市？”
    - 针对 CheckInDate 槽的提示 –“您想要在哪天入住？”
    - 针对 Nights 槽的提示 –“您要住几天？”
    - 针对 RoomType 槽的提示 –“您想预订哪种类型的房间，双人床房、大床房还是豪华大床房？”
    - 确认声明 — “好吧，我让你在 {Location} 住一晚 {Nights} 晚，从 {CheckInDate} 开始。是否要预订？”

    - 拒绝 –“好的，已经取消了正在进行的预订。”

- 意图 2 (BookCar) — 其预配置如下：
  - 预配置的槽
    - PickUpCity，属于 AMAZON.US_CITY 内置类型
    - PickUpDate，属于 AMAZON.DATE 内置类型
    - ReturnDate，属于 AMAZON.DATE 内置类型
    - DriverAge，属于 AMAZON.NUMBER 内置类型
    - CarType，属于 CarTypes 自定义类型
  - 预配置的表达
    - “预订汽车”
    - “订一辆车”
    - “办理汽车预订”

    如果用户说出其中任何一个，Amazon Lex 就会确定 BookCar 其意图，然后提示用户输入槽位数据。
  - 预配置的提示
    - 针对 PickUpCity 槽的提示 –“您需要在哪座城市租赁汽车？”
    - 针对 PickUpDate 槽的提示 –“您想从哪天开始租赁汽车？”
    - 针对 ReturnDate 槽的提示 –“您想在哪天归还汽车？”
    - 针对 DriverAge 槽的提示 –“驾驶此次所租汽车的司机多大年龄？”
    - 针对 CarType 插槽的提示 —“您想要租哪种类型的汽车？ 我们最受欢迎的车型是经济型、中型和豪华型汽车”
    - 确认声明 — “好吧，我让你在 {CarType} 到 {PickUpCity} 租一套 {ReturnDate}。PickUpDate是否要预订？”
    - 拒绝 –“好的，已经取消了正在进行的预订。”

Lambda 函数蓝图概述 () lex-book-trip-python

除了机器人蓝图外，还 AWS Lambda 提供了一个蓝图 (lex-book-trip-python)，您可以将其用作机器人蓝图的代码挂钩。关于机器人蓝图和相应 Lambda 函数蓝图的列表，请参阅Amazon Lex 和 AWS Lambda 蓝图。

使用 BookTrip 蓝图创建机器人时，您可以通过将此 Lambda 函数添加为用于初始化/验证用户数据输入 BookCar 和实现意图的代码挂钩来更新意图（ 和 BookHotel ）的配置。

提供的本 Lambda 函数展示了动态会话，该会话使用之前了解的用户信息 (保存在会话属性中) 来初始化目的的槽值。有关更多信息，请参阅 管理对话上下文。

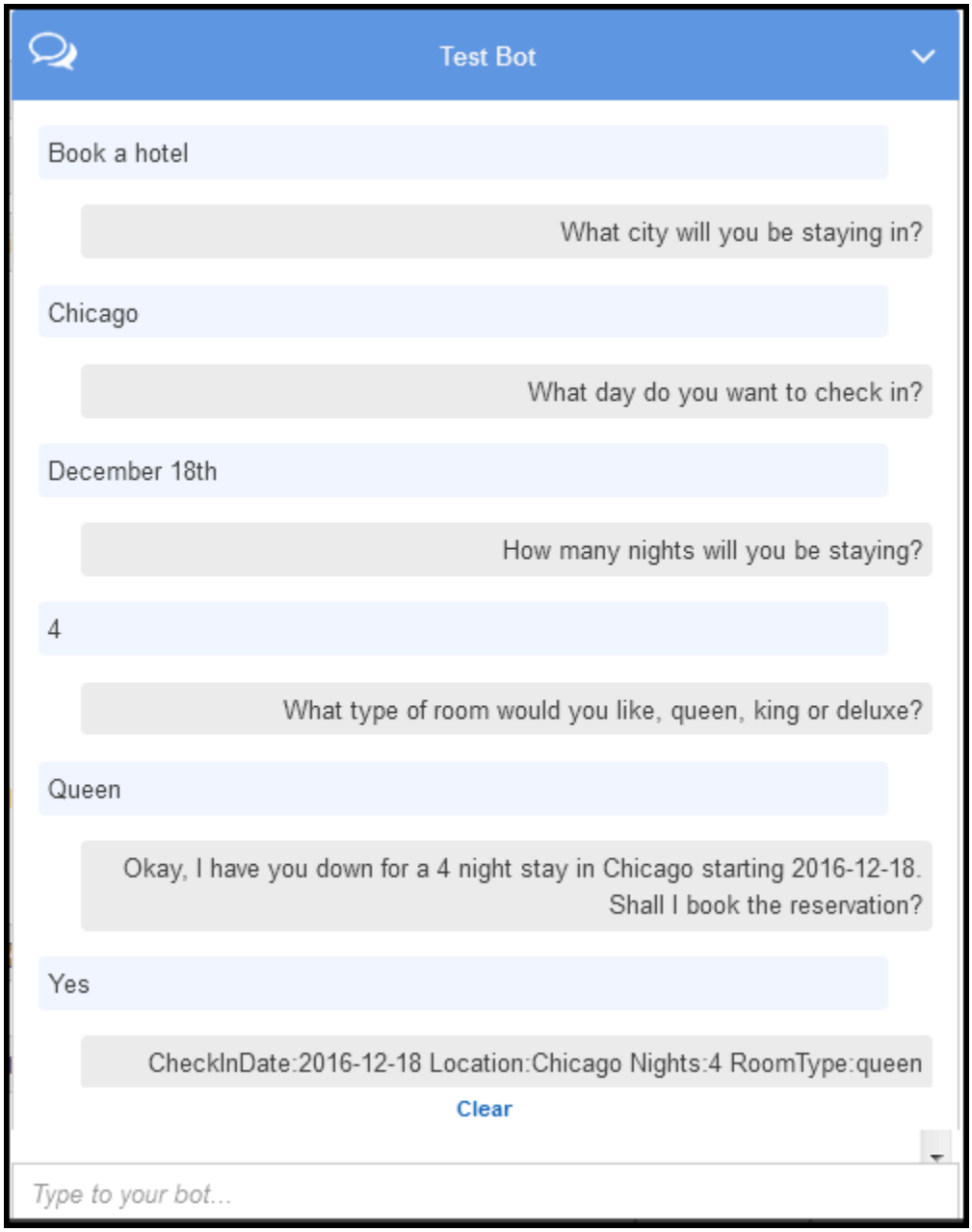
下一个步骤

步骤 2：创建 Amazon Lex 机器人

步骤 2：创建 Amazon Lex 机器人

在本节中，您将创建一个 Amazon Lex 机器人 (BookTrip)。

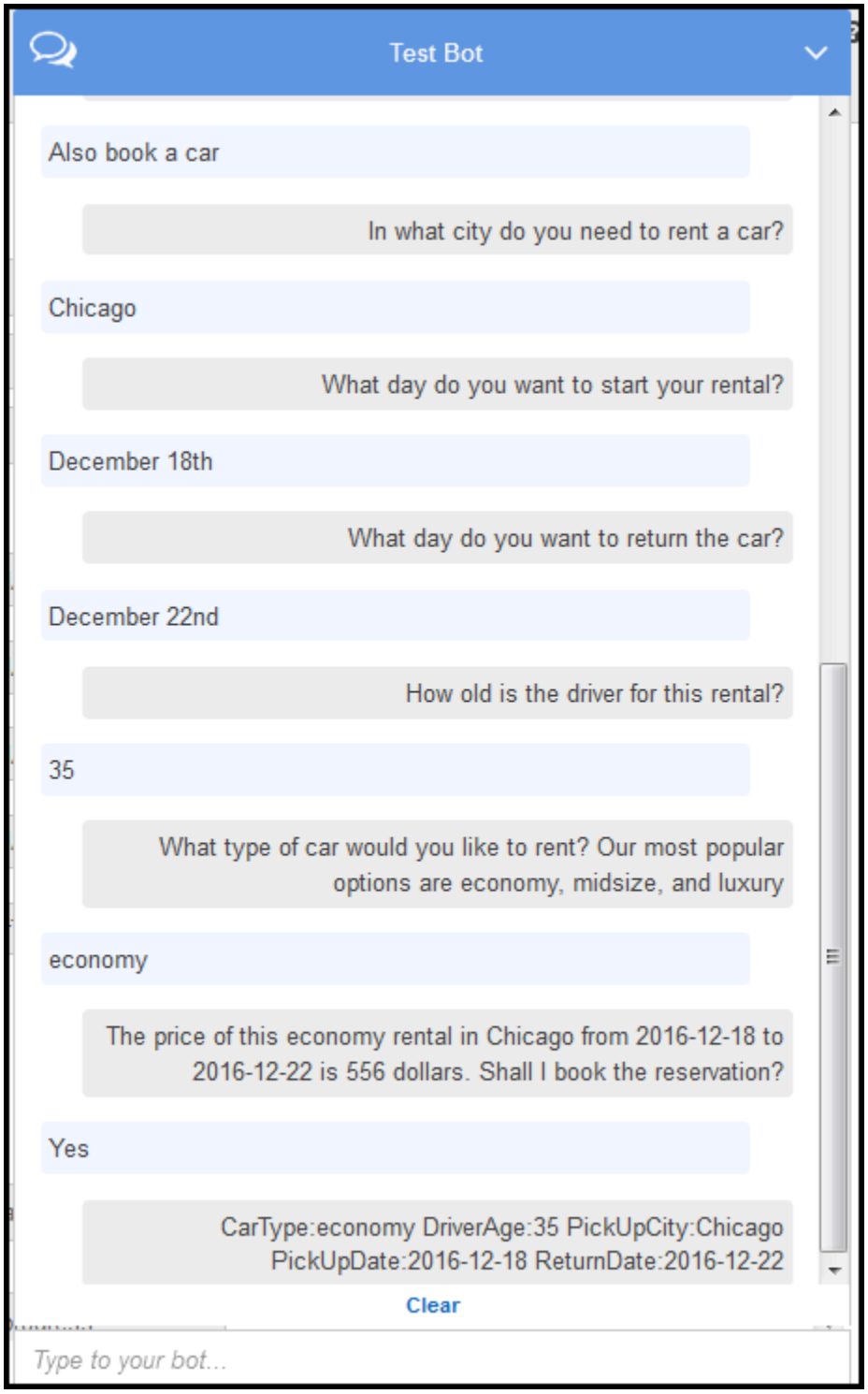
1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 在 Bots 页面上，选择 Create。
3. 在 Create your Lex bot 页面上，
   - 选择 BookTrip 蓝图。
   - 保留默认的机器人名称 (BookTrip)。
4. 选择创建。控制台会向 Amazon Lex 发送一系列请求以便创建机器人。请注意以下几点：
5. 控制台显示 BookTrip 机器人。在编辑器选项卡上，查看预配置意图的详细信息（ BookCar 和 ）。BookHotel
6. 在测试窗口中测试机器人。使用以下内容与您的自动程序进行测试会话：



根据用户的初始输入（“预订酒店”），Amazon Lex 推断出意图 (BookHotel)。然后，自动程序使用在本目的中预配置的提示来引导用户提供槽数据。用户提供所有插槽数据之后，Amazon Lex 会向客户端返回一条包含所有用户输入的消息作为响应。客户端会显示响应中的消息，如下所示。

```
CheckInDate:2016-12-18 Location:Chicago Nights:5 RoomType:queen
```

现在，您继续进行对话并尝试在接下来的对话中预订汽车。



请注意,

- 此时不进行用户数据验证。例如，您可以提供任何城市来预订酒店。
- 您会再次提供一些相同的信息 (目的地、提车城市、提车日期及归还日期) 来预订汽车。在动态会话中，您的自动程序应该根据用户之前为预订酒店而提供的输入来初始化一些信息。

在下一部分，您会创建一个 Lambda 函数，以便通过会话属性利用跨目的信息共享来进行一些用户数据验证和初始化工作。然后您要将 Lambda 函数添加为代码挂钩以便对用户输入进行初始化/验证并实现目的，从而更新目的配置。

下一个步骤

步骤 3：创建 Lambda 函数

步骤 3：创建 Lambda 函数

在本节中，您将使用控制台中提供的蓝图 (lex-book-trip-python) 创建 Lambda 函数。 AWS Lambda 您还可以使用控制台提供的示例事件数据来调用 Lambda 函数，从而对其进行测试。

此 Lambda 函数用 Python 编写。

1. 登录 AWS Management Console 并打开 AWS Lambda 控制台，网址为https://console.aws.amazon.com/lambda/。
2. 选择 Create function (创建函数)。
3. 选择使用蓝图。键入 **lex** 查找蓝图，选择 `lex-book-trip-python` 蓝图。
4. 选择配置，按如下方式配置 Lambda 函数。
   - 键入 Lambda 函数名称 (`BookTripCodeHook`)。
   - 对于角色，选择 Create a new role from template(s)，然后键入角色名称。
   - 保留其他默认值。
5. 选择 Create function (创建函数)。
6. 如果您使用的是英语 (美国) (en-US) 以外的区域设置，请按照更新特定区域设置的蓝图中所述更新意图名称。
7. 测试 Lambda 函数 您使用用于预订汽车和预订酒店的示例数据调用两次 Lambda 函数。
   a. 从选择测试事件下拉列表中，选择配置测试事件。
   b. 从示例事件模板列表中，选择 Amazon Lex 预订酒店。

      此示例事件与 Amazon Lex 请求/响应模型相匹配。有关更多信息，请参阅 使用 Lambda 函数。
   c. 选择保存并测试。

d. 验证 Lambda 函数已成功执行。在此情况下，此响应与 Amazon Lex 响应模型相匹配。

e. 重复这一步骤。这次从示例事件模板列表中选择 Amazon Lex 预订汽车。Lambda 函数会处理汽车预订工作。

下一个步骤

步骤 4：将 Lambda 函数添加为代码挂钩

步骤 4：将 Lambda 函数添加为代码挂钩

在本节中，您将通过添加 Lambda 函数作为用于初始化/验证 BookCar 和配送活动的代码挂钩来更新和 BookHotel意图的配置。请确保您选择了 $LATEST 版本的意图，因为您只能更新 $LATEST 版本的 Amazon Lex 资源。

1. 在 Amazon Lex 控制台中，选择该BookTrip机器人。
2. 在编辑器选项卡上，选择BookHotel意图。按以下方式更新意图配置：

   a. 确保意图版本 (意图名称旁边) 为 $LATEST。

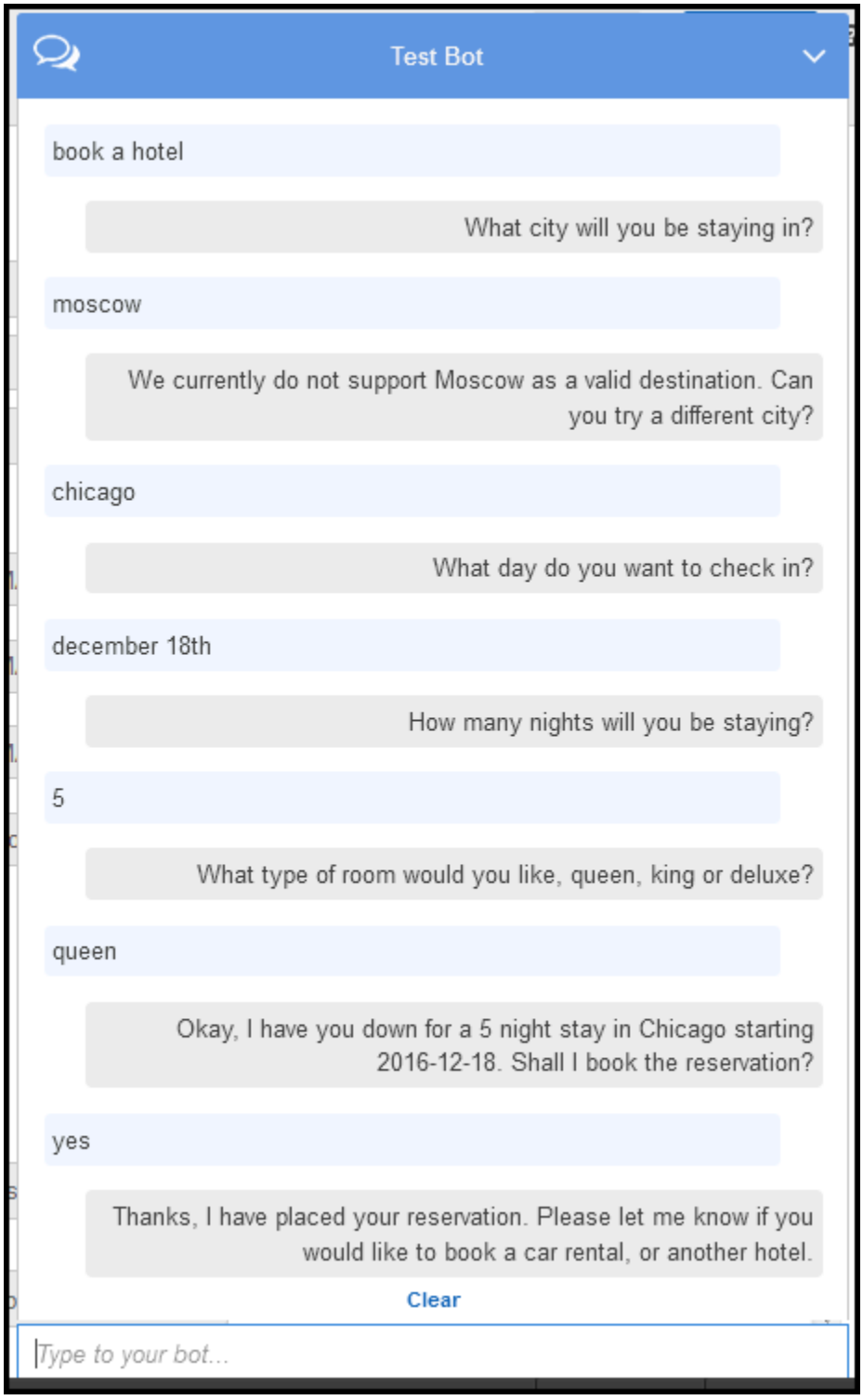
   b. 按以下方式将 Lambda 函数添加为初始化和验证代码挂钩：

      - 在选项中，选择初始化和验证代码挂钩。
      - 从列表中选择您的 Lambda 函数。

   c. 按以下方式将 Lambda 函数添加为履行代码挂钩：

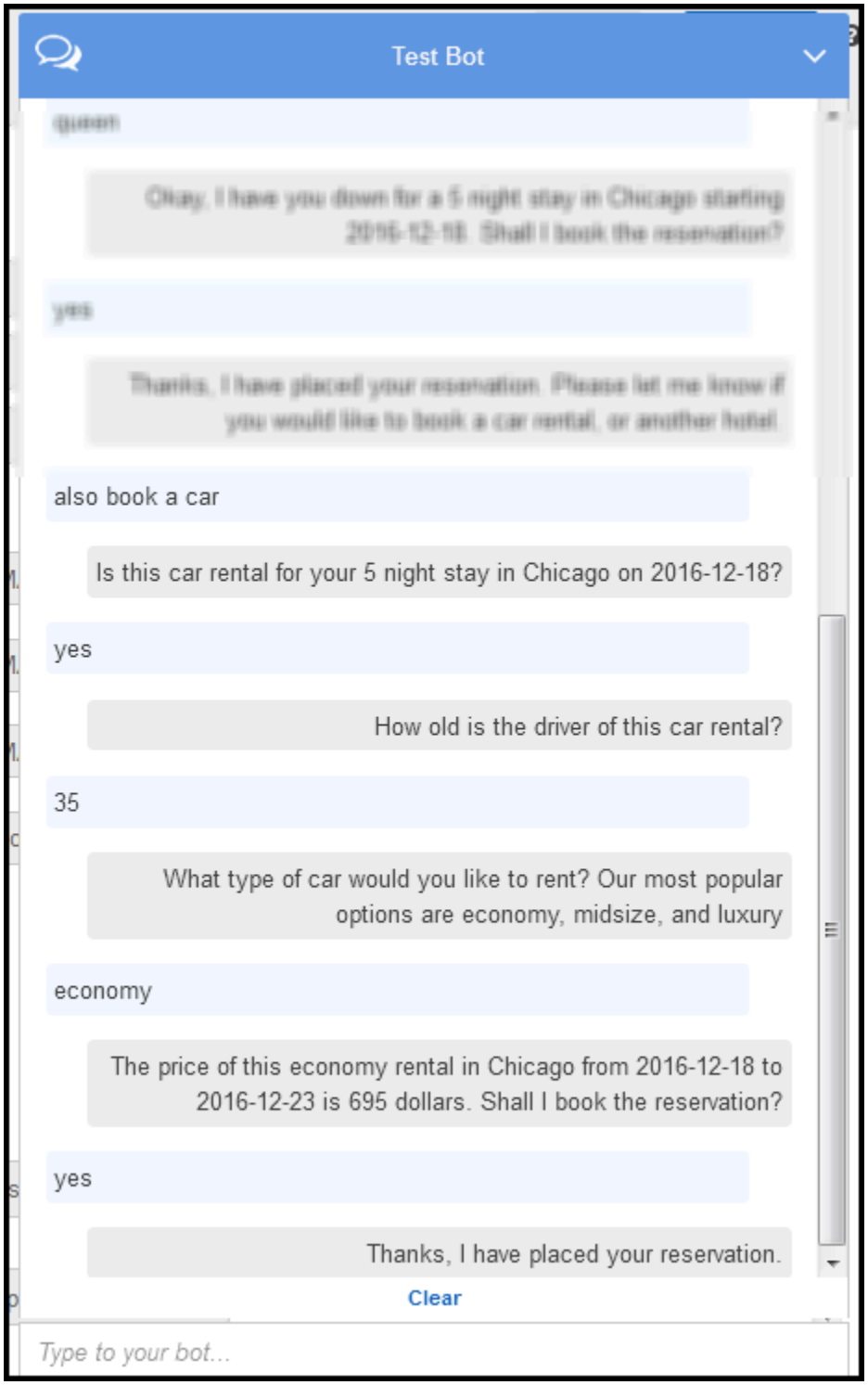
      - 在履行中，选择 AWS Lambda 函数。
      - 从列表中选择您的 Lambda 函数。
      - 选择 Goodbye message 并键入消息。

   d. 选择保存。
3. 在编辑器选项卡上，选择 BookCar 意图。按照之前的步骤将 Lambda 函数添加为验证和实现代码挂钩。
4. 选择构建。控制台会向 Amazon Lex 发送一系列请求以便保存配置。
5. 测试自动程序。有了能够执行初始化、用户数据验证和履行工作的 Lambda 函数，您就能在接下来的对话中看到用户交互中的差异：



有关从客户端（控制台）到 Amazon Lex 以及从 Amazon Lex 到 Lambda 函数的数据流的更多信息，请参阅数据流：预订酒店目的。

6. 按照下图中所示继续进行对话并预订汽车：



当您选择预订汽车时，客户端（控制台）会向 Amazon Lex 发送包含会话属性的请求（来自之前的对话 BookHotel）。Amazon Lex 将此信息传递给 Lambda 函数，然后 Lambda 函数初始化（即预填充）一些 BookCar 槽位数据（即、、 PickUpDate和）。 ReturnDate PickUpCity

> **Note**
>
> 这体现了将会话属性用于跨目的保留背景信息的过程。控制台客户端在测试窗口中提供 Clear 链接，用户可以使用此链接清除之前的所有会话属性。

有关从客户端（控制台）到 Amazon Lex 以及从 Amazon Lex 到 Lambda 函数的数据流的更多信息，请参阅数据流：预订汽车目的。

信息流的详细信息

在本练习中，您使用 Amazon Lex 控制台中提供的测试窗口客户端与 Amazon Lex BookTrip 机器人进行了对话。本节介绍以下内容：

- 客户端与 Amazon Lex 之间的数据流。

  本节假定客户端使用 `PostText` 运行时 API 向 Amazon Lex 发送请求，并显示相应的请求和响应详细信息。有关 `PostText` 运行时 API 的更多信息，请参阅 PostText。

  > **Note**
  >
  > 有关客户端和 Amazon Lex（客户端在其中使用 `PostContent` API）之间的信息流示例，请参阅步骤 2a (可选)：查看语音信息流的详细信息 (控制台)。

- Amazon Lex 与 Lambda 函数之间的数据流。有关更多信息，请参阅 Lambda 函数输入事件和响应格式。

主题

- 数据流：预订酒店目的
- 数据流：预订汽车目的

数据流：预订酒店目的

本部分介绍了获得每个用户输入后会发生的情况。

1. 用户：“book a hotel”

   a. 客户端 (控制台) 将向 Amazon Lex 发送以下 PostText 请求：

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"book a hotel",
   "sessionAttributes":{}
}
```

   请求 URI 和正文都向 Amazon Lex 提供信息：

   - 请求 URI-提供机器人名称 (BookTrip)、机器人别名 ($LATEST) 和用户名。后面的 `text` 表示它是一个 `PostText` API 请求 (而不是 `PostContent`)。
   - 请求正文 – 包括用户输入 (`inputText`) 和空 `sessionAttributes`。最初这是一个空对象，Lambda 函数首先设置会话属性。

   b. 在 `inputText` 中，Amazon Lex 可检测意图 (BookHotel)。此意图配置有 Lambda 函数作为代码挂钩，以用于用户数据初始化/验证。因此，Amazon Lex 会通过传递以下信息作为事件参数来调用 Lambda 函数（请参阅输入事件格式）：

```
{
   "messageVersion":"1.0",
   "invocationSource":"DialogCodeHook",
   "userId":"wch89kjqcpkds8seny7dly5x3otq68j3",
   "sessionAttributes":{
   },
   "bot":{
      "name":"BookTrip",
      "alias":null,
      "version":"$LATEST"
   },
   "outputDialogMode":"Text",
```

```
   "currentIntent":{
      "name":"BookHotel",
      "slots":{
         "RoomType":null,
         "CheckInDate":null,
         "Nights":null,
         "Location":null
      },
      "confirmationStatus":"None"
   }
}
```

除了客户端发送的信息以外，Amazon Lex 还包含以下额外数据：

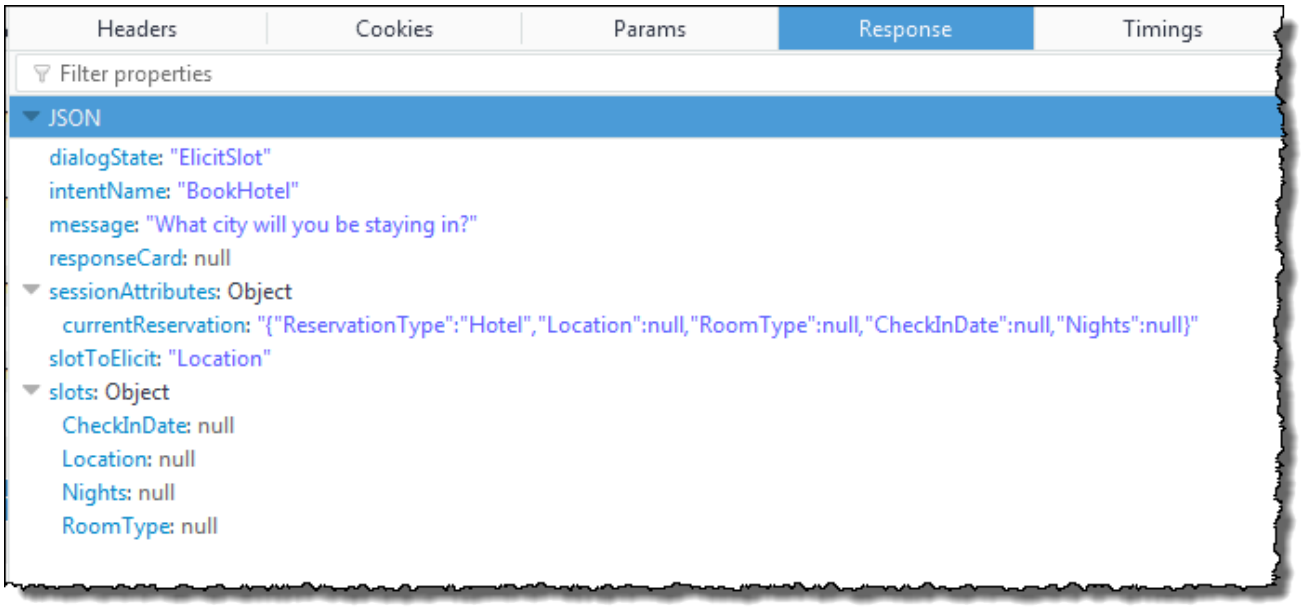
- `messageVersion` — Amazon Lex 当前只支持 1.0 版。
- `invocationSource` — 表明 Lambda 函数调用的目的。在本示例中，它会执行用户数据初始化和验证（此时 Amazon Lex 知道用户尚未提供履行意图所需的所有插槽数据）。
- `currentIntent` – 所有槽值均设置为空。

c. 此时，所有槽值均为空值。Lambda 函数没有任何内容需要验证。Lambda 函数会向 Amazon Lex 返回以下响应。有关响应格式的信息，请参阅响应格式。

```
{
   "sessionAttributes":{
      "currentReservation":"{\"ReservationType\":\"Hotel\",\"Location\":null,
\"RoomType\":null,\"CheckInDate\":null,\"Nights\":null}"
   },
   "dialogAction":{
      "type":"Delegate",
      "slots":{
         "RoomType":null,
         "CheckInDate":null,
         "Nights":null,
         "Location":null
      }
   }
}
```

> **Note**
>
> - `currentReservation` — Lambda 函数包含此会话属性。其值为当前槽信息和预订类型。
>
> 只有 Lambda 函数和客户端可以更新这些会话属性。Amazon Lex 只是传递这些值。
> - `dialogAction.type` — 通过将此值设置为 `Delegate`，Lambda 函数将下一个操作的责任委派给 Amazon Lex。
>
> 如果 Lambda 函数在用户数据验证过程中检测到任何内容，就会向 Amazon Lex 说明后续步骤。

d. 根据 `dialogAction.type`，Amazon Lex 决定下一个操作（从用户获取 `Location` 插槽的数据）。它会根据目的配置针对这个槽选择一项提示消息（“您要住在哪座城市？”），然后向用户发送以下响应：



这些会话属性会被传递到客户端。

客户端读取响应并显示消息：“您要住在哪座城市？”

2. 用户：“Moscow”

a. 客户端向 Amazon Lex 发送以下 `PostText` 请求（为便于阅读而添加了换行符）：

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"Moscow",
   "sessionAttributes":{
       "currentReservation":"{\"ReservationType\":\"Hotel\",
                              \"Location\":null,
                              \"RoomType\":null,
                              \"CheckInDate\":null,
                              \"Nights\":null}"
   }
}
```

除了 inputText 以外，客户端还会发送其收到的相同 currentReservation 会话属性。

b. Amazon Lex 首先会根据当前意图来解读 inputText（服务记得它已经向特定用户询问了关于 Location 插槽的信息）。它会更新当前意图的插槽值并使用以下事件来调用 Lambda 函数：

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook",
    "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
    "sessionAttributes": {
        "currentReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":null,\"RoomType\":null,\"CheckInDate\":null,\"Nights\":null}"
    },
    "bot": {
        "name": "BookTrip",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "BookHotel",
        "slots": {
            "RoomType": null,
            "CheckInDate": null,
            "Nights": null,
            "Location": "Moscow"
```

```
        },
        "confirmationStatus": "None"
    }
}
```

> **Note**
>
> - `invocationSource` 仍然是 `DialogCodeHook`。在此步骤中，我们只验证用户数据。
> - Amazon Lex 只是向 Lambda 函数传递会话属性。
> - 对于 `currentIntent.slots`，Amazon Lex 已经将 `Location` 插槽更新为 `Moscow`。

c. Lambda 函数执行用户数据验证，并确定 `Moscow` 是一个无效地点。

> **Note**
>
> 在本练习中，Lambda 函数具有一个简单的有效城市列表，而 `Moscow` 不在此列表中。在生产应用中，您可以使用后端数据库来获取这一信息。

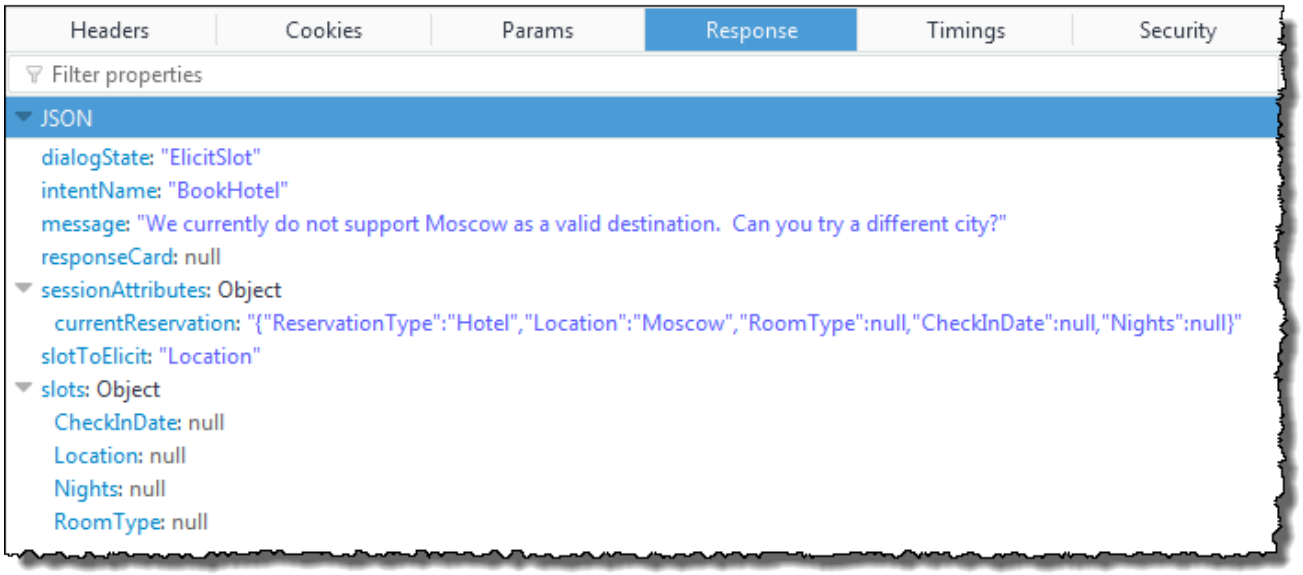
它会将插槽值重置回空值，并让 Amazon Lex 通过发送以下响应来再次提示用户提供另一个值：

```
{
    "sessionAttributes": {
        "currentReservation": "{\"ReservationType\":\"Hotel\",\"Location\":
\"Moscow\",\"RoomType\":null,\"CheckInDate\":null,\"Nights\":null}"
    },
    "dialogAction": {
        "type": "ElicitSlot",
        "intentName": "BookHotel",
        "slots": {
            "RoomType": null,
            "CheckInDate": null,
            "Nights": null,
            "Location": null
        },
        "slotToElicit": "Location",
```

```
        "message": {
            "contentType": "PlainText",
            "content": "We currently do not support Moscow as a valid
 destination.  Can you try a different city?"
        }
    }
}
```

> **Note**
>
> - `currentIntent.slots.Location` 被重置为空值。
> - `dialogAction.type` 设置为 `ElicitSlot`，这会让 Amazon Lex 通过提供以下响应来再次提示用户：
>   - `dialogAction.slotToElicit` – 要将用户提供的数据用到的槽。
>   - `dialogAction.message` - 传达给用户的 `message`。

d. Amazon Lex 发现 `dialogAction.type` 并通过以下响应向客户端传递信息：



客户端只是显示消息：“目前我们不支持莫斯科作为有效目的地，能否换一个城市？”

3. 用户：“Chicago”

   a. 客户端将向 Amazon Lex 发送以下 `PostText` 请求：

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
```

```
"Content-Encoding":"amz-1.0"

{
   "inputText":"Chicago",
   "sessionAttributes":{
       "currentReservation":"{\"ReservationType\":\"Hotel\",
                              \"Location\":\"Moscow\",
                              \"RoomType\":null,
                              \"CheckInDate\":null,
                              \"Nights\":null}"
   }
}
```

b. Amazon Lex 知道上下文，即引发 Location 插槽的数据。在这个上下文中，它知道 inputText 值用于 Location 槽。然后，它通过发送以下事件来调用 Lambda 函数：

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook",
    "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
    "sessionAttributes": {
        "currentReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":Moscow,\"RoomType\":null,\"CheckInDate\":null,\"Nights\":null}"
    },
    "bot": {
        "name": "BookTrip",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "BookHotel",
        "slots": {
            "RoomType": null,
            "CheckInDate": null,
            "Nights": null,
            "Location": "Chicago"
        },
        "confirmationStatus": "None"
    }
}
```

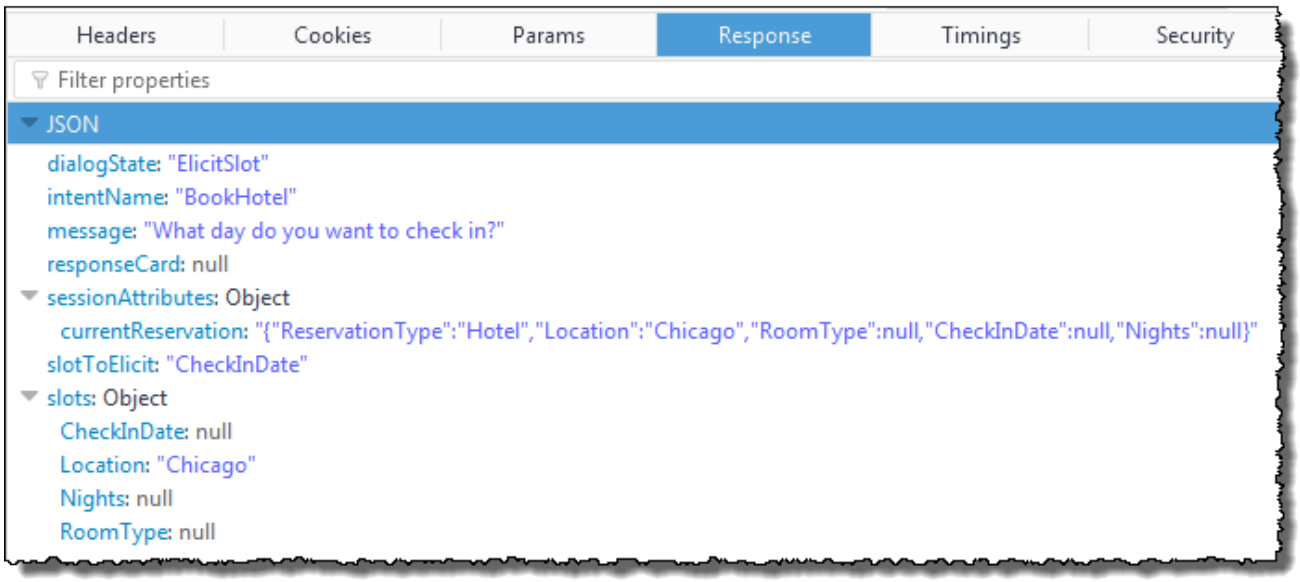
Amazon Lex 通过将 Location 插槽设置为 Chicago 来更新 currentIntent.slots。

c. 根据 DialogCodeHook 的 invocationSource 值，Lambda 函数会执行用户数据验证。它会将 Chicago 识别为有效的插槽值，相应更新会话属性，然后向 Amazon Lex 返回以下响应。

```
{
    "sessionAttributes": {
        "currentReservation": "{\"ReservationType\":\"Hotel\",\"Location\":
\"Chicago\",\"RoomType\":null,\"CheckInDate\":null,\"Nights\":null}"
    },
    "dialogAction": {
        "type": "Delegate",
        "slots": {
            "RoomType": null,
            "CheckInDate": null,
            "Nights": null,
            "Location": "Chicago"
        }
    }
}
```

> **Note**
>
> - currentReservation — Lambda 函数通过将 Location 设置为 Chicago 来更新这一会话属性。
> - dialogAction.type – 被设置为 Delegate。用户数据有效，Lambda 函数指示 Amazon Lex 选择下一个操作。

d. 根据 dialogAction.type，Amazon Lex 选择下一个操作。Amazon Lex 知道它需要更多插槽数据，因此会根据意图配置将下一个未填充的插槽 (CheckInDate) 设为最高优先级。它会根据目的配置针对这个槽选择一项提示消息（“您想要在哪天入住？”），然后向客户端发送回以下响应：

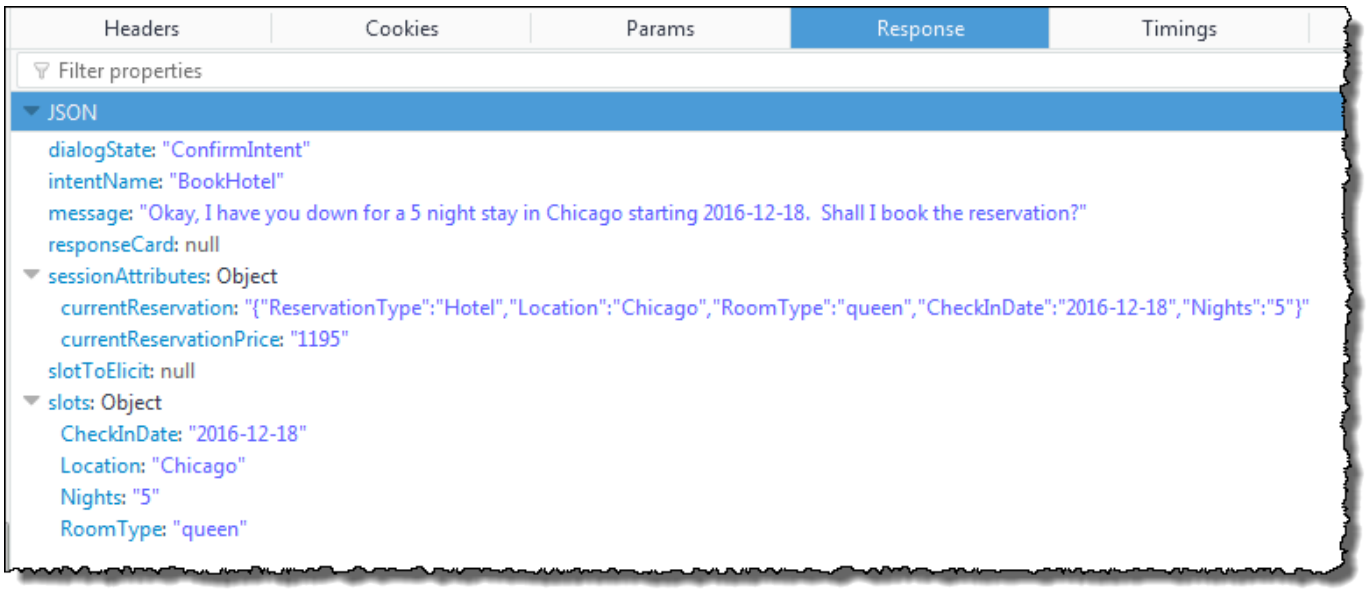


客户端显示消息：“您想在哪天入住？”

4. 用户交互继续进行（用户提供数据，Lambda 函数验证数据，然后将下一个操作委派给 Amazon Lex）。最后，用户提供所有插槽数据，Lambda 函数验证所有用户输入，然后 Amazon Lex 确定自己获得了所有插槽数据。

> **Note**
>
> 在本练习中，用户提供所有插槽数据之后，Lambda 函数会计算酒店的预订价格并将其作为另一个会话属性 (`currentReservationPrice`) 返回。

此时，意图已准备就绪，但 BookHotel意图已配置为确认提示，要求用户确认后，Amazon Lex 才能实现意图。因此，在预订酒店之前，Amazon Lex 会向客户端发送以下消息来要求确认：



客户端会显示消息：“好的，确定您要在芝加哥住 5 晚，入住日期是 2016-12-18。是否要预订？”

5. 用户：“yes”

   a. 客户端将向 Amazon Lex 发送以下 `PostText` 请求：

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"Yes",
   "sessionAttributes":{
       "currentReservation":"{\"ReservationType\":\"Hotel\",
                              \"Location\":\"Chicago\",
                              \"RoomType\":\"queen\",
                              \"CheckInDate\":\"2016-12-18\",
                              \"Nights\":\"5\"}",
      "currentReservationPrice":"1195"
   }
}
```

   b. Amazon Lex 将解释确认当前意图的上下文中的 `inputText`。Amazon Lex 知道用户希望继续预订。此时 Amazon Lex 会通过发送以下事件来调用 Lambda 函数，从而履行意图。通过将事件中的 `invocationSource` 设置为 `FulfillmentCodeHook`，它向 Lambda 函数发送信息。Amazon Lex 也将 `confirmationStatus` 设置为 `Confirmed`。

```
{
    "messageVersion": "1.0",
    "invocationSource": "FulfillmentCodeHook",
    "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
    "sessionAttributes": {
        "currentReservation": "{\"ReservationType\":\"Hotel\",\"Location\":
\"Chicago\",\"RoomType\":\"queen\",\"CheckInDate\":\"2016-12-18\",\"Nights\":
\"5\"}",
        "currentReservationPrice": "956"
    },
    "bot": {
        "name": "BookTrip",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "BookHotel",
        "slots": {
            "RoomType": "queen",
            "CheckInDate": "2016-12-18",
            "Nights": "5",
            "Location": "Chicago"
        },
        "confirmationStatus": "Confirmed"
    }
}
```

> **Note**
>
> - `invocationSource` — 此时，Amazon Lex 将此值设置为 `FulfillmentCodeHook`，指示 Lambda 函数履行意图。
> - `confirmationStatus` – 被设置为 `Confirmed`。

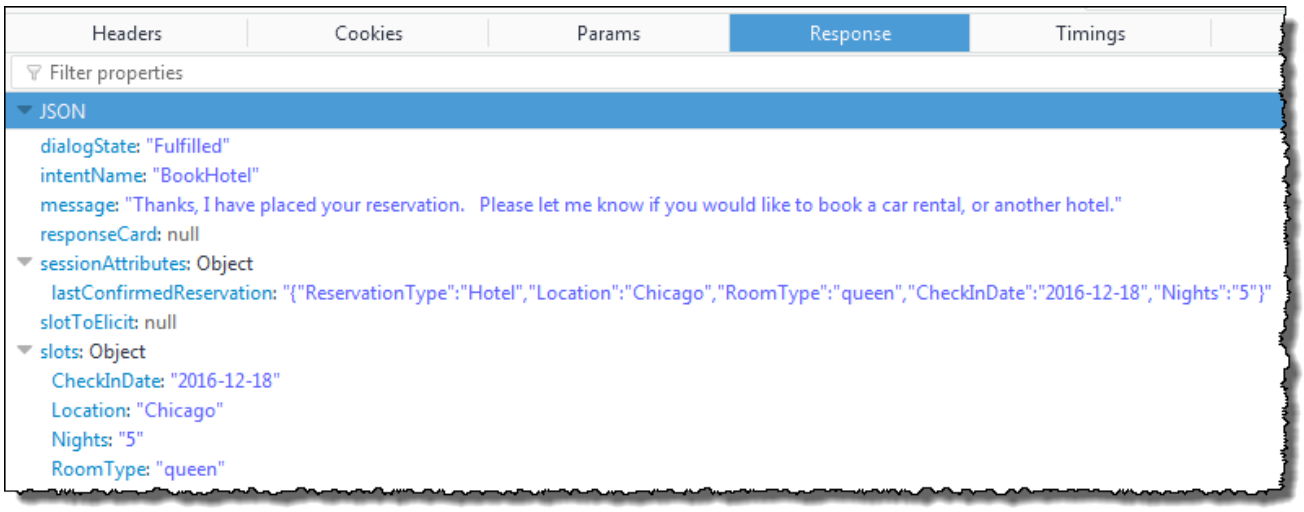
c. 这次，Lambda 函数实现了 BookHotel 意图，Amazon Lex 完成了预订，然后它会返回以下响应：

```
{
    "sessionAttributes": {
```

```
        "lastConfirmedReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":\"Chicago\",\"RoomType\":\"queen\",\"CheckInDate\":\"2016-12-18\",\"Nights
\":\"5\"}"
    },
    "dialogAction": {
        "type": "Close",
        "fulfillmentState": "Fulfilled",
        "message": {
            "contentType": "PlainText",
            "content": "Thanks, I have placed your reservation.   Please let me
 know if you would like to book a car rental, or another hotel."
        }
    }
}
```

> **Note**
>
> - `lastConfirmedReservation` — 是 Lambda 函数添加的新会话属性（不是 `currentReservation` 和 `currentReservationPrice`）。
> - `dialogAction.type` — Lambda 函数将此值设置为 `Close`，表示 Amazon Lex 不需要用户响应。
> - `dialogAction.fulfillmentState` – 被设置为 `Fulfilled` 并且包含要传达给用户的相应 `message`。

d. Amazon Lex 查看 `fulfillmentState` 并向客户端发送以下响应：



> **Note**
>
> - `dialogState` — Amazon Lex 将此值设置为 `Fulfilled`。
> - `message` — 与 Lambda 函数提供的消息相同。

客户端将显示消息。

数据流：预订汽车目的

本练习中的 BookTrip 机器人支持两种意图（BookHotel 和 BookCar）。预订酒店之后，用户可以继续进行对话以便预订汽车。只要会话未超时，客户端就会在每个后续请求中继续发送会话属性 (本示例中为 `lastConfirmedReservation`)。Lambda 函数可以使用此信息为意图初始化插槽数据。BookCar 这体现了在跨目的数据共享中使用会话属性的过程。

具体而言，当用户选择 BookCar 意图时，Lambda 函数使用会话属性中的相关信息为意图预填充插槽（PickUpDate ReturnDate、和 PickUpCity）。 BookCar

> **Note**
>
> Amazon Lex 控制台提供清除链接，您可使用此链接清除之前的所有会话属性。

按照以下步骤继续进行对话。

1. 用户：“also book a car”

   a. 客户端将向 Amazon Lex 发送以下 `PostText` 请求：

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"also book a car",
   "sessionAttributes":{
       "lastConfirmedReservation":""{\"ReservationType\":\"Hotel\",
                                    \"Location\":\"Chicago\",
```

```
                                        \"RoomType\":\"queen\",
                                        \"CheckInDate\":\"2016-12-18\",
                                        \"Nights\":\"5\"}"
    }
}
```

客户端会发送 `lastConfirmedReservation` 会话属性。

b. Amazon Lex 从中检测到意图 (BookCar) `inputText`。此意图也被配置为调用 Lambda 函数来执行初始化和用户数据验证。Amazon Lex 使用以下事件调用 Lambda 函数：

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook",
    "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
    "sessionAttributes": {
        "lastConfirmedReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":\"Chicago\",\"RoomType\":\"queen\",\"CheckInDate\":\"2016-12-18\",\"Nights
\":\"5\"}"
    },
    "bot": {
        "name": "BookTrip",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "BookCar",
        "slots": {
            "PickUpDate": null,
            "ReturnDate": null,
            "DriverAge": null,
            "CarType": null,
            "PickUpCity": null
        },
        "confirmationStatus": "None"
    }
}
```

> **Note**
>
> - `messageVersion` — Amazon Lex 当前只支持 1.0 版。

- invocationSource – 表明调用的目的是执行初始化和用户数据验证。
- currentIntent — 其中包括意图名称和插槽。此时，所有槽值均为空值。

c. Lambda 函数发现所有插槽值均为空值，没有任何可以验证的内容。但是函数会使用会话属性来初始化一些槽值 (PickUpDate、ReturnDate 和 PickUpCity)，然后返回以下响应：

```
{
    "sessionAttributes": {
        "lastConfirmedReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":\"Chicago\",\"RoomType\":\"queen\",\"CheckInDate\":\"2016-12-18\",\"Nights
\":\"5\"}",
        "currentReservation": "{\"ReservationType\":\"Car\",\"PickUpCity
\":null,\"PickUpDate\":null,\"ReturnDate\":null,\"CarType\":null}",
        "confirmationContext": "AutoPopulate"
    },
    "dialogAction": {
        "type": "ConfirmIntent",
        "intentName": "BookCar",
        "slots": {
            "PickUpCity": "Chicago",
            "PickUpDate": "2016-12-18",
            "ReturnDate": "2016-12-22",
            "CarType": null,
            "DriverAge": null
        },
        "message": {
            "contentType": "PlainText",
            "content": "Is this car rental for your 5 night stay in Chicago on
 2016-12-18?"
        }
    }
}
```

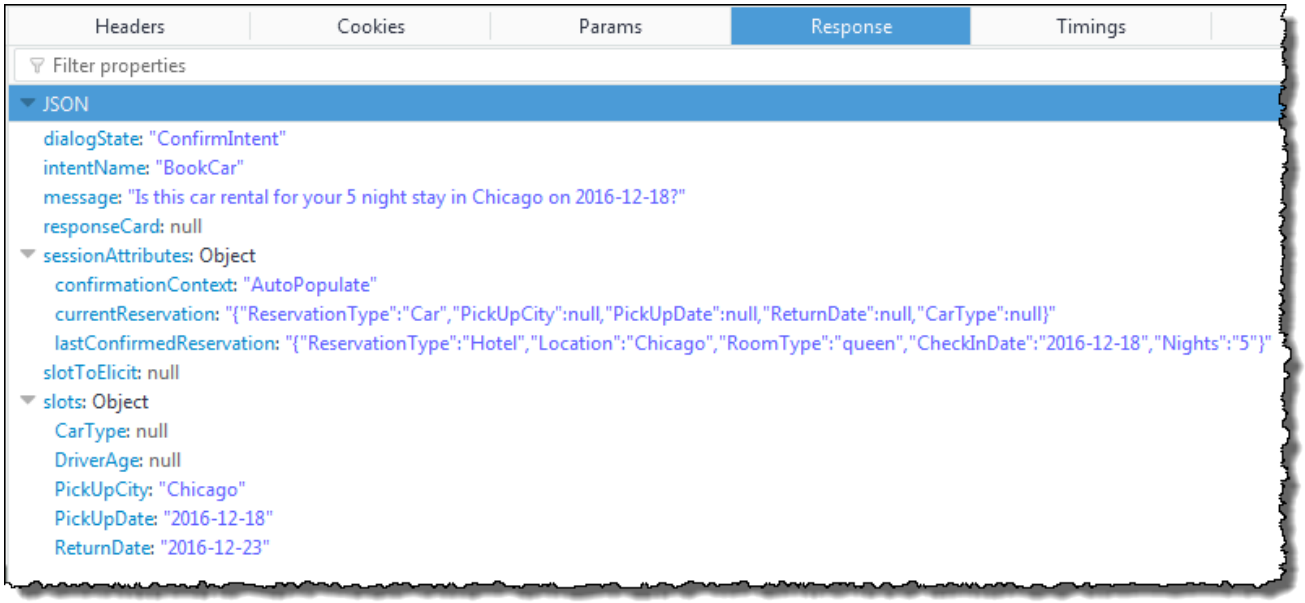
> **Note**
>
> - 除了 lastConfirmedReservation 以外，Lambda 函数中还包含更多会话属性（currentReservation 和 confirmationContext）。

- `dialogAction.type`设置为`ConfirmIntent`，这会通知 Amazon Lex 需要用户回复 “是”，“否”（确认上下文设置为，Lambda 函数知道是/否用户回复是/否是的目的是 AutoPopulate让用户确认 Lambda 函数执行的初始化（自动填充的插槽数据）。

  Lambda 函数还在响应中包含一条 `dialogAction.message` 中的通知消息，供 Amazon Lex 将其返回客户端。

  > **Note**
  >
  > `ConfirmIntent` (`dialogAction.type` 的值) 与所有自动程序目的都没有关系。在本示例中，Lambda 函数使用该术语让 Amazon Lex 获取来自用户的“是”或“否”回复。

d. 根据 `dialogAction.type`，Amazon Lex 向客户端返回以下响应：



客户端显示消息：“您是否要从 2016-12-18 开始在芝加哥租用这辆汽车 5 天？”

2. 用户：“yes”

   a. 客户端将向 Amazon Lex 发送以下 `PostText` 请求：

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
```

```
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
   "inputText":"yes",
   "sessionAttributes":{
      "confirmationContext":"AutoPopulate",
      "currentReservation":"{\"ReservationType\":\"Car\",
                            \"PickUpCity\":null,
                            \"PickUpDate\":null,
                            \"ReturnDate\":null,
                            \"CarType\":null}",
      "lastConfirmedReservation":"{\"ReservationType\":\"Hotel\",
                                  \"Location\":\"Chicago\",
                                  \"RoomType\":\"queen\",
                                  \"CheckInDate\":\"2016-12-18\",
                                  \"Nights\":\"5\"}"
   }
}
```

b. Amazon Lex 读取 `inputText` 并了解上下文（要求用户确认自动填充）。Amazon Lex 通过发送以下事件来调用 Lambda 函数：

```
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook",
    "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
    "sessionAttributes": {
        "confirmationContext": "AutoPopulate",
        "currentReservation": "{\"ReservationType\":\"Car\",\"PickUpCity
\":null,\"PickUpDate\":null,\"ReturnDate\":null,\"CarType\":null}",
        "lastConfirmedReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":\"Chicago\",\"RoomType\":\"queen\",\"CheckInDate\":\"2016-12-18\",\"Nights
\":\"5\"}"
    },
    "bot": {
        "name": "BookTrip",
        "alias": null,
        "version": "$LATEST"
    },
    "outputDialogMode": "Text",
    "currentIntent": {
        "name": "BookCar",
```

```
        "slots": {
            "PickUpDate": "2016-12-18",
            "ReturnDate": "2016-12-22",
            "DriverAge": null,
            "CarType": null,
            "PickUpCity": "Chicago"
        },
        "confirmationStatus": "Confirmed"
    }
}
```

由于用户回复“是”，所以 Amazon Lex 将 confirmationStatus 设置为 Confirmed。

c. 根据 confirmationStatus，Lambda 函数确定预填充的值正确无误。Lambda 函数执行以下操作：

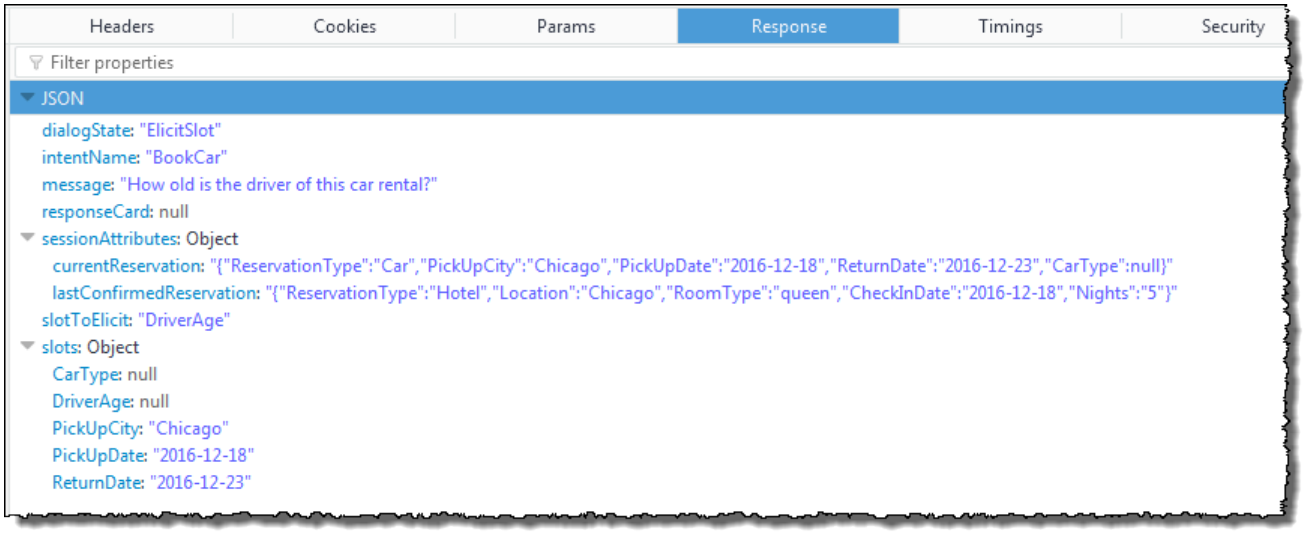
- 将 currentReservation 会话属性更新为已经预填充的槽值。
- 将 dialogAction.type 设置为 ElicitSlot
- 将 slotToElicit 值设置为 DriverAge。

发送的响应如下：

```
{
    "sessionAttributes": {
        "currentReservation": "{\"ReservationType\":\"Car\",\"PickUpCity
\":\"Chicago\",\"PickUpDate\":\"2016-12-18\",\"ReturnDate\":\"2016-12-22\",
\"CarType\":null}",
        "lastConfirmedReservation": "{\"ReservationType\":\"Hotel\",\"Location
\":\"Chicago\",\"RoomType\":\"queen\",\"CheckInDate\":\"2016-12-18\",\"Nights
\":\"5\"}"
    },
    "dialogAction": {
        "type": "ElicitSlot",
        "intentName": "BookCar",
        "slots": {
            "PickUpDate": "2016-12-18",
            "ReturnDate": "2016-12-22",
            "DriverAge": null,
            "CarType": null,
            "PickUpCity": "Chicago"
        },
```

```
        "slotToElicit": "DriverAge",
        "message": {
            "contentType": "PlainText",
            "content": "How old is the driver of this car rental?"
        }
    }
}
```

d. Amazon Lex 返回以下响应：



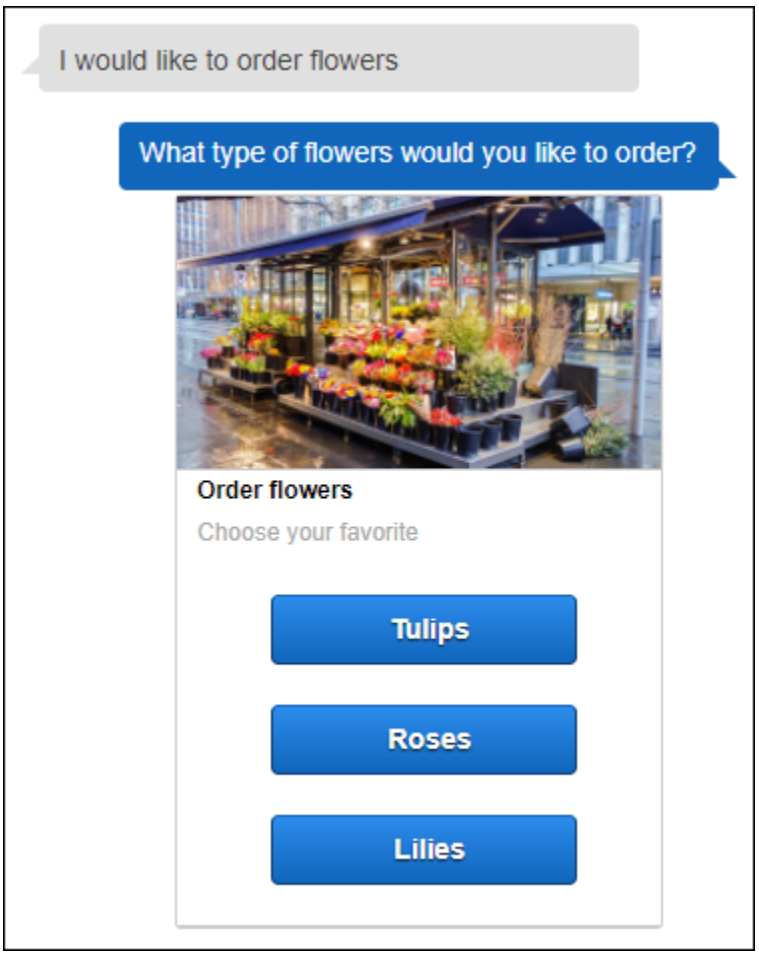
客户端显示消息“驾驶此次所租汽车的司机多大年龄？”对话继续进行。

使用响应卡

在本练习中，您将通过添加响应卡来扩展入门练习 1 中的知识。您可以创建一个支持 OrderFlowers Intent 的机器人，然后通过为该FlowerType插槽添加响应卡来更新 Intent。除了以下 FlowerType 槽的提示外，用户还可以从响应卡中选择鲜花类型：

```
What type of flowers would you like to order?
```

以下为响应卡：



自动程序用户可以键入文本或从鲜花类型列表中进行选择。此响应卡配置有图像，会按如下所示显示在客户端中。有关响应卡的更多信息，请参阅 响应卡。

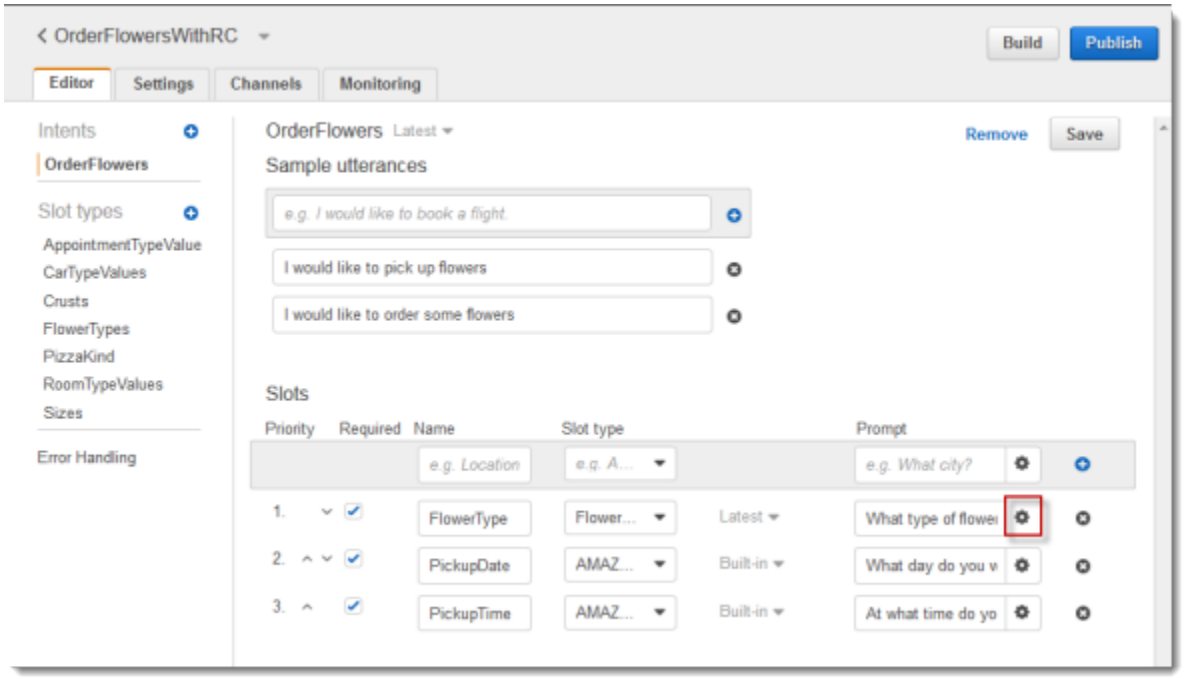
使用响应卡创建和测试自动程序：

1. 按照入门练习 1 创建和测试 OrderFlowers 机器人。您必须完成步骤 1、2 和 3。您无需添加 Lambda 函数来测试响应卡。有关说明，请参阅 练习 1：使用蓝图创建 Amazon Lex 机器人（控制台）。
2. 通过添加响应卡来更新自动程序，然后发布版本。当您发布版本时，请指定一个指向它的别名 (BETA)。

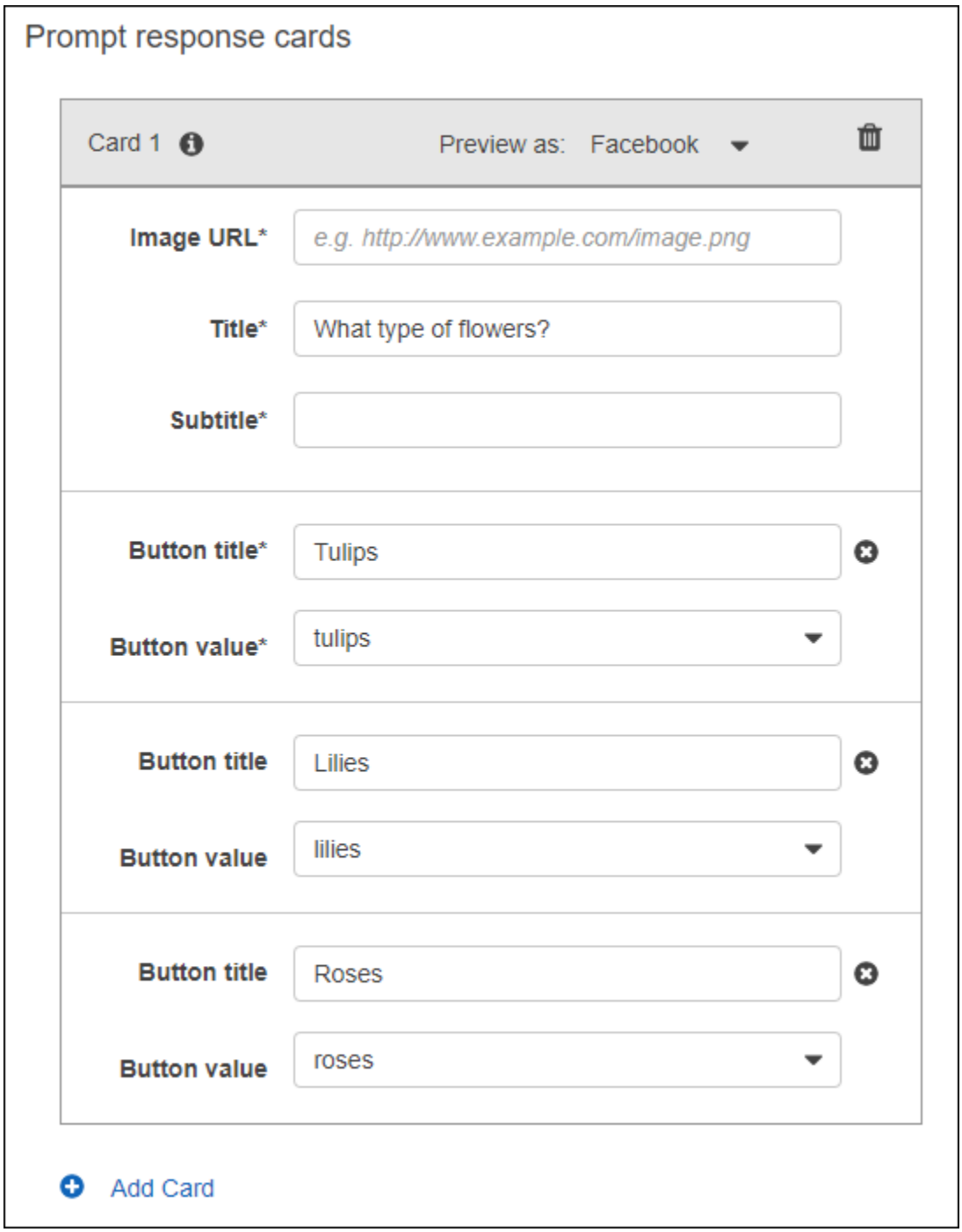
   a. 在 Amazon Lex 控制台中，选择您的机器人。

   b. 选择 `OrderFlowers` 目的。

   c. 选择“哪种类型的花”提示旁边的设置齿轮图标来配置 `FlowerType` 的响应卡，如下图中所示。



d. 为卡指定一个标题，并按以下屏幕截图所示配置三个按钮。您可以选择性地将图像添加到响应卡中，前提是您拥有图像 URL。如果您正在使用 Twilio SMS 部署自动程序，则必须提供图片 URL。



e. 选择 Save (保存) 以保存响应卡。

f. 选择 Save intent (保存目的) 以保存目的配置。

g. 要构建自动程序，请选择 Build。

h. 要发布自动程序版本，请选择 Publish。指定 BETA 作为指向自动程序版本的别名。有关版本控制的信息，请参阅版本控制和别名。

3. 在消息收发平台上部署自动程序：

- 在 Facebook Messenger 平台上部署自动程序并测试集成。有关说明，请参阅 将 Amazon Lex 机器人与 Facebook Messenger 集成。在您订花时，消息窗口会显示响应卡，以便您可以选择鲜花类型。
- 在 Slack 平台上部署自动程序并测试集成。有关说明，请参阅 将 Amazon Lex 机器人与 Slack 集成 。在您订花时，消息窗口会显示响应卡，以便您可以选择鲜花类型。
- 在 Twilio SMS 平台上部署自动程序。有关说明，请参阅 将 Amazon Lex 机器人与 Twilio 可编程 SMS 集成 。当您订购鲜花时，Twilio 发出的消息显示了响应卡上的图像。Twilio SMS 不支持在响应中使用按钮。

更新言语

在本练习中，您将向在入门练习 1 中创建的表达中添加更多表达。您可以在 Amazon Lex 控制台中使用监控选项卡查看机器人未识别的言语。为改进您的用户的体验，您可以将这些表达添加到自动程序中。

在以下条件下，系统不会生成言语统计数据：

- 创建机器人时，`childDirected` 字段设置为 True。
- 您正在对一个或多个插槽使用插槽混淆处理。
- 您已选择退出 Amazon Lex 改进计划。

> **Note**
>
> 表达统计数据每天生成一次。您可以查看未识别的表达、收听的次数以及最近一次收听该表达的日期和时间。缺失的表达可能最多需要 24 小时才能显示在控制台中。

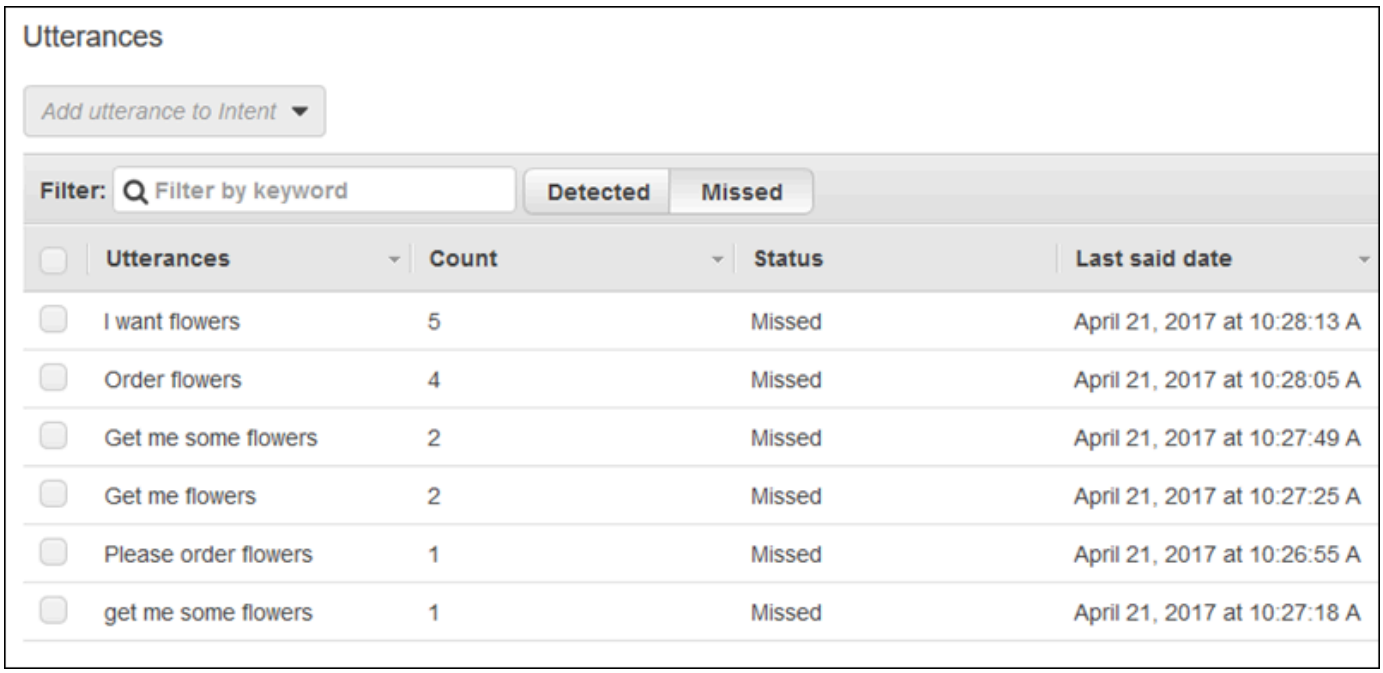
您可以查看不同版本自动程序的表达。要查看不同版本自动程序的表达，请从自动程序名称旁边的下拉列表中选择不同的版本。

查看缺失的表达并将其添加到自动程序中：

1. 按照入门练习 1 的第一步创建和测试 `OrderFlowers` 自动程序。有关说明，请参阅 练习 1：使用蓝图创建 Amazon Lex 机器人（控制台）。
2. 通过在 Test Bot 窗口中键入以下表达来测试自动程序。每个表达键入几次。示例自动程序无法识别以下表达：

- Order flowers
- Get me flowers
- Please order flowers
- Get me some flowers

3. 等待 Amazon Lex 收集有关缺失言语的使用率数据。表达数据每天生成一次，通常在晚上生成。
4. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
5. 选择 `OrderFlowers` 自动程序。
6. 选择 Monitoring 选项卡，然后从左侧菜单中选择 Utterances，再选择 Missed 按钮。下面的窗格最多显示 100 个缺失的言语。



7. 要选择想添加到自动程序中的缺失的表达，请选中旁边的复选框。要将表达添加到 `$LATEST` 版本的目的中，请选择 Add utterance to intent 下拉列表旁的向下箭头，然后选择目的。
8. 要重建您的自动程序，请选择 Build，然后再次选择 Build 来重建您的自动程序。
9. 要验证您的自动程序可以识别新表达，请使用 Test Bot 窗格。

与网站集成

在本示例中，您将使用文本和语音将自动程序与网站集成。您使用 JavaScript 和 AWS 服务为网站访问者提供交互式体验。您可以从 AWS AI 博客上记录的这些示例中进行选择：

- 为您的聊天机器人部署 Web 用户界面 — 演示一个为 Amazon Lex 聊天机器人提供 Web 客户端的功能齐全的 Web 用户界面。您可以使用它来了解 Web 客户端，或将其用作您自己的应用程序的构建块。

- “问候，访客！”—使用 Amazon Lex 吸引您的网络用户 — 演示如何使用 Amazon Lex、浏览器 JavaScript 中的 AWS 软件开发工具包和 Amazon Cognito 在您的网站上创建对话体验。

- 在@@ 浏览器中捕获语音输入并将其发送到 Amazon Lex — 演示使用浏览器中的 SDK 将基于语音的聊天机器人嵌入网站。 JavaScript 该应用程序可录制音频、将音频发送至 Amazon Lex，然后播放响应。

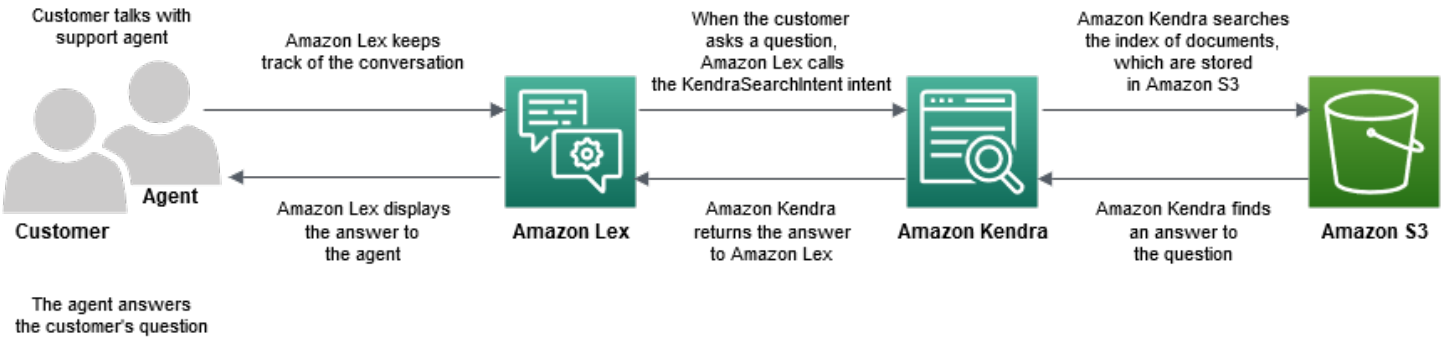
呼叫中心客服助理

在本教程中，您将使用 Amazon Lex 和 Amazon Kendra 来构建一个客服助理机器人，该机器人可为客户支持客服提供帮助，并将其发布为 Web 应用程序。Amazon Kendra 是一项企业搜索服务，它使用机器学习来搜索文档以找到答案。有关 Amazon Kendra 的更多信息，请参阅 Amazon Kendra 开发人员指南。

Amazon Lex 机器人广泛用于呼叫中心，作为客户的第一联系点。机器人通常能够解决客户的问题。当机器人无法回答问题时，它会将对话转移给客户支持员工。

在本教程中，我们会创建一个 Amazon Lex 机器人，客服使用它来实时回答客户的查询。通过阅读机器人提供的答案，客服可以免于手动查找答案。

您在本教程中创建的机器人和 Web 应用程序可通过快速提供合适的资源来帮助客服高效、准确地响应客户。下图显示了该 Web 应用程序的工作方式。



如图所示，Amazon Kendra 文档索引存储在 Amazon Simple Storage Service (Amazon S3) 存储桶中。如果还没有 S3 存储桶，可以在创建 Amazon Kendra 索引时设置一个。除了 Amazon S3 之外，您还将在本教程中使用 Amazon Cognito。Amazon Cognito 管理将机器人部署为 Web 应用程序的权限。

在本教程中，您将创建提供客户问题答案的 Amazon Kendra 索引，创建机器人并添加意图以使机器人能够根据与客户的对话提出答案，设置 Amazon Cognito 以管理访问权限，并将机器人部署为 Web 应用程序。

估计时间：75 分钟

预估费用：一个 Amazon Kendra 索引 2.50 美元/每小时，1000 个 Amazon Lex 请求 0.75 美元。完成本练习后，您的 Amazon Kendra 索引将继续运行。请务必将其删除，以免产生不必要的费用。

注意：请确保为本教程中使用的所有服务选择相同的 AWS 区域。

主题

步骤 1：创建 Amazon Kendra 索引

首先创建 Amazon Kendra 文档索引，用于回答客户问题。索引为客户端查询提供搜索 API。您可以从源文档创建索引。Amazon Kendra 会将它在已编制索引的文档中找到的答案返回给机器人，机器人会将这些答案显示给客服。

Amazon Kendra 建议的响应的质量和准确性取决于您编制索引的文档。文档应包括客服经常访问且必须存储在 S3 存储桶中的文件。您可为 .html、Microsoft Office（.doc、.ppt）、PDF 和文本格式的非结构化和半结构化数据编制索引。

要创建 Amazon Kendra 索引，请参阅《Amazon Kendra 开发人员指南》中的 Getting started with an S3 bucket (console)。

要添加有助于回答客户查询的问题和答案 (FAQs)，请参阅 Amazon Kendra 开发者指南中的添加问题和答案。在本教程中，请使用上的 ML_FAQ.csv 文件 GitHub。

后续步骤

步骤 2：创建 Amazon Lex 机器人

步骤 2：创建 Amazon Lex 机器人

Amazon Lex 提供了呼叫中心客服和 Amazon Kendra 索引之间的接口。它会跟踪客服与客户之间的对话，并根据客户提出的问题调用 `AMAZON.KendraSearchIntent` 意图。意图是指用户要执行的操作。

Amazon Kendra 搜索已编制索引的文档，并向 Amazon Lex 返回显示在机器人中的答案。该答案仅对客服可见。

创建客服助理机器人

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 在导航窗格中，选择自动程序。
3. 选择创建。
4. 选择自定义机器人并配置机器人。
   a. 机器人名称 — 输入一个指示机器人用途的名称，例如 **`AgentAssistBot`**。
   b. 输出语音 — 选择无。
   c. 会话超时 — 输入 **5**。
   d. COPPA — 选择否。
5. 选择创建。创建机器人后，Amazon Lex 会显示机器人编辑器选项卡。

后续步骤

步骤 3：添加自定义意图和内置意图

步骤 3：添加自定义意图和内置意图

意图表示呼叫中心客服希望机器人执行的操作。在此示例中，客服希望机器人根据客服与客户的对话来建议响应和有用的资源。

Amazon Lex 有两种意图：自定义意图和内置意图。`AMAZON.KendraSearchIntent` 是内置意图。机器人使用 `AMAZON.KendraSearchIntent` 意图来查询索引并显示 Amazon Kendra 建议的响应。

此示例中的机器人不需要自定义意图。但是，要成功构建机器人，您必须至少创建一个自定义意图，其中包含至少一个示例言语。只有在构建客服助理机器人时才需要此意图。它不执行任何其他功能。意图的言语不得应用于客户可能会询问的任何问题。这样可以确保调用 AMAZON.KendraSearchIntent 来回答客户的问题。有关更多信息，请参阅 AMAZON.KendraSearchIntent。

创建所需的自定义意图

1. 在自动程序入门页面上，选择创建目的。
2. 对于添加目的，选择创建目的。
3. 在创建意图对话框中，输入意图的描述性名称，例如 **RequiredIntent**。
4. 在示例言语中，输入描述性言语，例如 **Required utterance**。
5. 选择保存意图。

添加亚马逊。 KendraSearchIntent 意图和响应消息

1. 在导航窗格中，选择意图旁边的加号 (+)。
2. 选择搜索现有意图。
3. 在搜索意图框中，输入 **AMAZON.KendraSearchIntent**，然后从列表中选择它。
4. 为意图提供一个描述性名称（例如 **AgentAssistSearchIntent**），然后选择添加。
5. 在目的编辑器中，选择 Amazon Kendra 查询以打开查询选项。
6. 选择您想要意图搜索的索引。
7. 在响应部分，将以下三条消息添加到消息组。

```
I found an answer for the customer query: ((x-amz-lex:kendra-search-response-
question_answer-question-1)) and the answer is ((x-amz-lex:kendra-search-response-
question_answer-answer-1)).
I found an excerpt from a helpful document: ((x-amz-lex:kendra-search-response-
document-1)).
I think this answer will help the customer: ((x-amz-lex:kendra-search-response-
answer-1)).
```

8. 选择保存意图。
9. 选择构建以构建机器人。

后续步骤

步骤 4：设置 Amazon Cognito

步骤 4：设置 Amazon Cognito

要管理网络应用程序的权限和用户，您需要设置 Amazon Cognito。Amazon Cognito 可确保网络应用程序的安全性并具有访问控制功能。Amazon Cognito 使用身份池提供 AWS 证书，向您的用户授予访问其他 AWS 服务的权限。在本教程中，它提供对 Amazon Lex 的访问权限。

在创建身份池时，Amazon Cognito 会为经过身份验证和未经身份验证的用户提供 AWS Identity and Access Management (IAM) 角色。您可以通过添加授予 Amazon Lex 访问权限的策略来修改 IAM 角色。

设置 Amazon Cognito

1. 登录 AWS Management Console 并打开 Amazon Cognito 控制台，网址为。https://console.aws.amazon.com/cognito/
2. 选择 Manage Identity Pools（管理身份池）。
3. 选择 Create new identity pool（创建新身份池）。
4. 配置身份池。
    a. 身份池名称 — 输入一个表明池用途的名称，例如 **`BotPool`**。
    b. 在未经验证的身份部分，选择启用对未经验证的身份的访问。
5. 选择创建池。
6. 在识别要用于新身份池的 IAM 角色页面，选择查看详细信息。
7. 记录 IAM 角色的名称。稍后您将修改它们。
8. 选择允许。
9. 在 Amazon Cognito 入门页面上，对于 “平台”，选择。JavaScript
10. 在 “获取 AWS 凭证” 部分，查找并记录身份池 ID。
11. 要允许访问 Amazon Lex，请修改经过身份验证和未经身份验证的 IAM 角色。
    a. 登录 AWS Management Console 并打开 IAM 控制台，网址为https://console.aws.amazon.com/iam/。
    b. 在导航窗格中的访问管理下，选择角色。
    c. 在搜索框中，输入经过身份验证的 IAM 角色的名称，然后选择对应的复选框。

i. 选择附加策略。

ii. 在搜索框中，输入 **AmazonLexRunBotsOnly**，然后选中对应的复选框。

iii. 选择附加策略。

d. 在搜索框中，输入未经过身份验证的 IAM 角色的名称，然后选择对应的复选框。

i. 选择附加策略。

ii. 在搜索框中，输入 **AmazonLexRunBotsOnly**，然后选中对应的复选框。

iii. 选择附加策略。

后续步骤

步骤 5：将您的机器人部署为 Web 应用程序

步骤 5：将您的机器人部署为 Web 应用程序

将您的机器人部署为 Web 应用程序

1. 将位于 /_assistan blob/master/example_apps/agentce_bot https://github.com/awsdocs/amazon-lex-developer-guide/的存储库下载到您的计算机上。
2. 导航到已下载的存储库，然后在编辑器中打开 index.html 文件。
3. 进行以下更改。

   a. 在 `AWS.config.credentials` 部分，输入您的区域名称和身份池 ID。

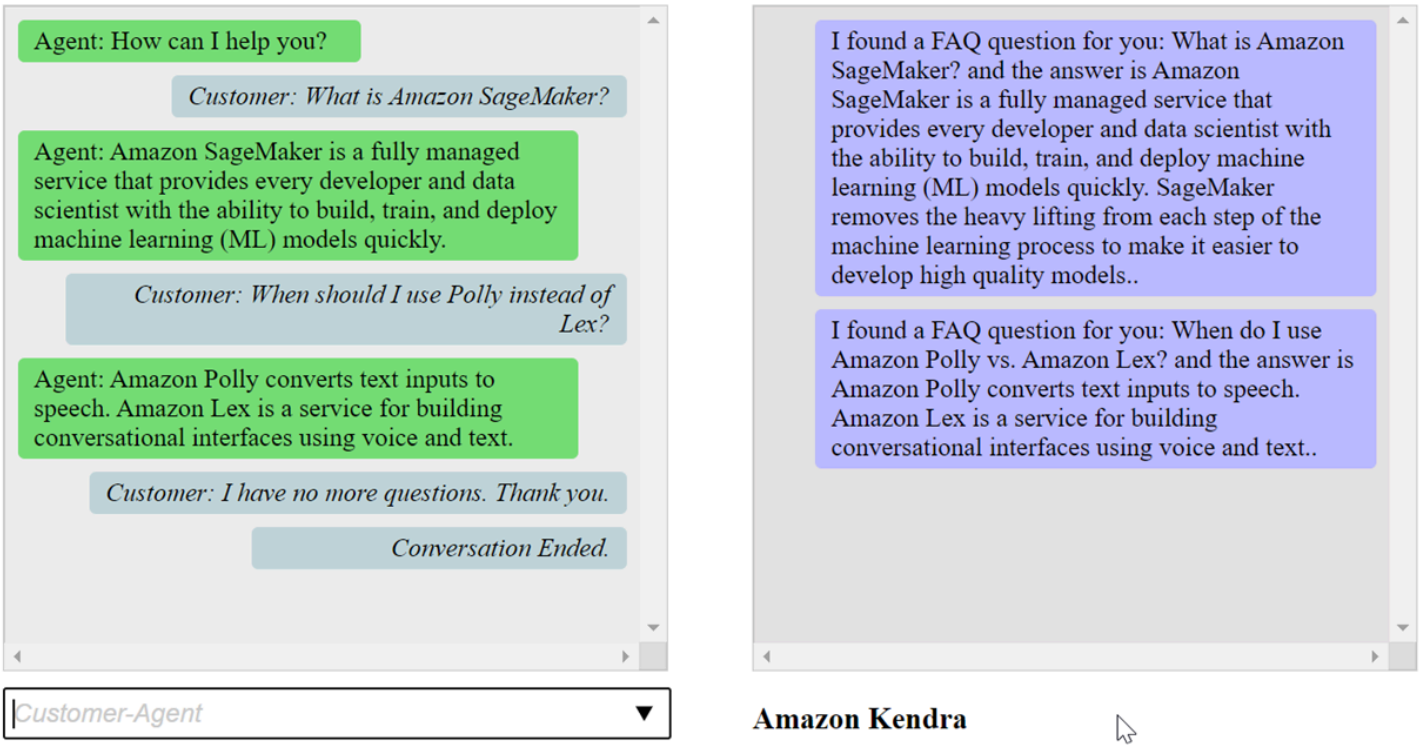
   b. 在 `Amazon Lex runtime parameters` 部分，输入机器人名称。

   c. 保存该文件。

步骤 6：使用机器人

出于演示目的，您以客户和客服的身份向机器人提供输入。为了区分两者，客户提出的问题以“客户：”开头，客服提供的答案以“客服：”开头。您可以从建议的输入菜单中进行选择。

打开 `index.html` 并与机器人进行类似于下图的对话来运行您的 Web 应用程序：



index.html 文件中的 pushChat() 函数说明如下。

```
            var endConversationStatement = "Customer: I have no more questions. Thank
you."
            // If the agent has to send a message, start the message with 'Agent'
            var inputText = document.getElementById('input');
            if (inputText && inputText.value && inputText.value.trim().length > 0 &&
inputText.value[0]=='Agent') {
                showMessage(inputText.value, 'agentRequest','conversation');
                inputText.value = "";
            }
            // If the customer has to send a message, start the message with 'Customer'
            if(inputText && inputText.value && inputText.value.trim().length > 0 &&
inputText.value[0]=='Customer') {
                // disable input to show we're sending it
                var input = inputText.value.trim();
                inputText.value = '...';
                inputText.locked = true;
                customerInput = input.substring(2);
```

```
                // Send it to the Lex runtime
                var params = {
                    botAlias: '$LATEST',
                    botName: 'KendraTestBot',
                    inputText: customerInput,
                    userId: lexUserId,
                    sessionAttributes: sessionAttributes
                };

                showMessage(input, 'customerRequest', 'conversation');
                if(input== endConversationStatement){
                    showMessage('Conversation
Ended.','conversationEndRequest','conversation');
                }
                lexruntime.postText(params, function(err, data) {
                    if (err) {
                        console.log(err, err.stack);
                        showMessage('Error:  ' + err.message + ' (see console for
details)', 'lexError', 'conversation1')
                    }

                    if (data &&input!=endConversationStatement) {
                        // capture the sessionAttributes for the next cycle
                        sessionAttributes = data.sessionAttributes;

                            showMessage(data, 'lexResponse', 'conversation1');
                    }
                    // re-enable input
                    inputText.value = '';
                    inputText.locked = false;
                });
            }
            // we always cancel form submission
            return false;
```

当您以客户身份提供输入时，Amazon Lex 运行时 API 会将其发送给 Amazon Lex。

`showMessage(daText, senderRequest, displayWindow)` 函数在聊天窗口中显示客服和客户之间的对话。Amazon Kendra 建议的响应显示在相邻的窗口中。当客户说 **“I have no more questions. Thank you.”** 时，对话就结束了

注意：不使用 Amazon Kendra 索引时，请将其删除。

迁移机器人

Amazon Lex V2 API 使用更新的信息架构，可简化资源版本控制并在机器人中支持多种语言。有关更多信息，请参阅《Amazon Lex 开发人员指南》中的迁移指南。

要使用这些新功能，您需要迁移您的机器人。当您迁移机器人时，Amazon Lex 会提供以下内容：

- 迁移会将您的自定义意图和插槽类型复制到 Amazon Lex V2 机器人中。
- 您可以通过 Amazon Lex V2 机器人添加多种语言。在 Amazon Lex V1 中，您可以为每种语言创建一个单独的机器人。您可以将多个 Amazon Lex V1 机器人（每个机器人使用不同的语言）迁移到一个 Amazon Lex V2 机器人。
- Amazon Lex 将 Amazon Lex V1 的内置插槽类型和意图映射到 Amazon Lex V2 的内置插槽类型和意图。如果内置版本无法迁移，Amazon Lex 会返回一条消息，告诉您下一步该怎么做。

迁移过程不会迁移以下内容：

- 别名
- Amazon Kendra 索引
- AWS Lambda 函数
- 对话日志设置
- 诸如 Slack 之类的消息通道
- 标签

要迁移机器人，您的用户或角色必须拥有 Amazon Lex 和 Amazon Lex V2 API 操作的 IAM 权限。有关所需的权限，请参阅允许用户将机器人迁移到 Amazon Lex V2 APIs。

迁移机器人（控制台）

使用 Amazon Lex V1 控制台将机器人的结构迁移到 Amazon Lex V2 机器人。

使用控制台将机器人迁移到 Amazon Lex V2 API

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 从左侧菜单中，选择迁移工具。

3. 从机器人列表中，选择要迁移的机器人，然后选择迁移。
4. 选择要迁移的机器人的版本，然后输入要迁移到的机器人的名称。如果您输入现有 Amazon Lex V2 机器人的名称，Amazon Lex V1 机器人将迁移到详细信息中显示的语言并覆盖该语言的草稿版本。
5. 选择下一步。
6. 选择 Amazon Lex 用于运行 Amazon Lex V2 API 版本的机器人的 IAM 角色。您可以选择创建具有运行机器人所需的最低权限的新角色，也可以选择现有的 IAM 角色。
7. 选择下一步。
8. 查看迁移设置。如果设置正确，请选择开始迁移。

开始迁移过程后，您将返回到迁移工具的起始页面。您可以在历史记录表中监控迁移进度。当迁移状态列显示完成时，迁移就完成了。

Amazon Lex 使用 Amazon Lex V2 API 中的 `StartImport` 操作来导入迁移的机器人。您会在 Amazon Lex V2 控制台的导入历史记录表中看到每个迁移的条目。

在迁移过程中，Amazon Lex 可能会在机器人中发现无法迁移的资源。对于每个无法迁移的资源，您都会收到一条错误或警告消息。每条消息都包含一个链接，指向说明如何解决问题的文档。

迁移 Lambda 函数

Amazon Lex V2 改变了为机器人定义 Lambda 函数的方式。它只允许在机器人中每种语言的别名中使用一个 Lambda 函数。有关迁移 Lambda 函数的更多信息，请参阅将 Lambda 函数从 Amazon Lex V1 迁移到 Amazon Lex V2。

迁移消息

在迁移过程中，Amazon Lex 可能会发现无法迁移到等效的 Amazon Lex V2 资源的资源，例如内置插槽类型。发生这种情况时，Amazon Lex 会返回一条迁移消息，描述发生的情况，并提供用于告诉您如何解决迁移问题的文档的链接。以下各节描述了在迁移机器人时可能出现的问题以及如何解决该问题。

主题

- 内置意图
- 内置插槽类型
- 对话日志
- 消息组

- 提示和短语
- Amazon Lex V1 的其他功能

内置意图

当您使用 Amazon Lex V2 不支持的内置意图时，该意图将映射到您的 Amazon Lex V2 机器人中的自定义意图。自定义意图不包含言语。要继续使用意图，请添加示例言语。

内置插槽类型

任何使用 Amazon Lex V2 不支持的插槽类型的已迁移插槽都不会被赋予插槽类型值。要使用此插槽，请执行以下操作：

- 创建自定义插槽类型
- 添加插槽类型所需的插槽类型值
- 更新插槽以使用新的自定义插槽类型

对话日志

迁移不会更新 Amazon Lex V2 机器人的对话日志设置。

配置对话日志

1. 在 https://console.aws.amazon.com/lexv2 上打开 Amazon Lex V2 控制台。
2. 从机器人列表中，选择要配置其对话日志的机器人。
3. 在左侧菜单中，选择别名，然后从列表中选择别名。
4. 在对话日志部分，选择管理对话日志，为机器人别名配置对话日志。

消息组

Amazon Lex V2 对于每个消息组仅支持一条消息和两条备用消息。如果您在 Amazon Lex V1 机器人中每个消息组有三条以上的消息，则仅迁移前三条消息。要在消息组中使用更多消息，请使用 Lambda 函数输出各种消息。

提示和短语

Amazon Lex V2 将不同的机制用于后续、澄清和挂断提示。

对于后续提示，请在履行后使用上下文结转切换到不同的意图。

例如，假设您打算预订一辆配置返回名为 book_car_fulfilled 的输出上下文的租车。履行意图时，Amazon Lex 会将输出上下文变量设置为 book_car_fulfilled。由于 book_car_fulfilled 是活动上下文，因此只要用户言语被识别为试图引发该意图，就可以考虑对 book_car_fulfilled 设置为输入上下文的意图进行识别。您可以将其用于只有在预订车辆后才有意义的意图，例如，通过电子邮件发送收据或修改预订。

Amazon Lex V2 不支持澄清提示和挂断短语（中止语句）。Amazon Lex V2 机器人包含默认的回退意图，如果没有匹配的意图，则会调用该意图。要发送包含重试的澄清提示，请配置 Lambda 函数并在回退意图中启用对话框代码挂钩。Lambda 函数可以输出澄清提示作为响应，并在会话属性中输出重试值。如果重试值超过最大重试次数，则可以输出挂断短语并关闭对话。

Amazon Lex V1 的其他功能

迁移工具仅支持 Amazon Lex V1 机器人及其底层意图、插槽类型和插槽的迁移。有关其他功能，请参阅 Amazon Lex V2 文档中的以下主题。

- 机器人别名：别名
- 机器人通道：在消息收发平台上部署 Amazon Lex V2 机器人
- 对话日志设置：使用对话日志进行监控
- 亚马逊 Kendra 索引：亚马逊。 KendraSearchIntent
- Lambda 函数：使用 AWS Lambda 函数
- 标签：为资源添加标签

将 Lambda 函数从 Amazon Lex V1 迁移到 Amazon Lex V2

Amazon Lex V2 只允许机器人中的每种语言使用一个 Lambda 函数。Lambda 函数及其设置是针对您在运行时使用的机器人别名进行配置的。

如果为意图启用了对话和履行代码挂钩，则会为该语言的所有意图调用 Lambda 函数。

Amazon Lex V2 Lambda 函数的输入和输出消息格式与 Amazon Lex V1 不同。下面是 Lambda 函数输入格式的区别。

- Amazon Lex V2 用 interpretations 结构取代了 currentIntent 和 alternativeIntents 结构。每种解释都包含意图、意图的 NLU 置信度分数和可选的情绪分析。

- Amazon Lex V2 将 Amazon Lex V1 中的 activeContexts 和 sessionAttributes 移至统一的 sessionState 结构。此结构提供有关对话当前状态的相关信息，包括原始请求 ID。
- Amazon Lex V2 不会返回 recentIntentSummaryView。改用 sessionState 结构中的信息。
- Amazon Lex V2 输入在 bot 属性中提供了 botId 和 localeId。
- 输入结构包含一个 inputMode 属性，该属性提供有关输入类型的信息：文本、语音或 DTMF。

下面是 Lambda 函数输出格式的区别：

- Amazon Lex V1 中的 activeContexts 和 sessionAttributes 结构被 Amazon Lex V2 中的 sessionState 结构所取代。
- recentIntentSummaryView 不包括在输出中。
- Amazon Lex V1 dialogAction 结构分为两个结构，dialogAction 是 sessionState 结构的一部分，messages 是 dialogAction.type 为 ElicitIntent 时必需的。Amazon Lex 从此结构中选择要向用户显示的消息。

当您使用 Amazon Lex V2 构建机器人时 APIs，每种语言的每个机器人别名只有一个 Lambda 函数，而不是每个意图都有一个 Lambda 函数。如果要继续使用单独的函数，则可以创建一个路由器函数，为每个意图激活单独的函数。以下是您可以为应用程序使用或修改的路由器函数。

```
import os
import json
import boto3

# reuse client connection as global
client = boto3.client('lambda')

def router(event):
    intent_name = event['sessionState']['intent']['name']
    fn_name = os.environ.get(intent_name)
    print(f"Intent: {intent_name} -> Lambda: {fn_name}")
    if (fn_name):
        # invoke lambda and return result
        invoke_response = client.invoke(FunctionName=fn_name, Payload =
 json.dumps(event))
        print(invoke_response)
        payload = json.load(invoke_response['Payload'])
        return payload
    raise Exception('No environment variable for intent: ' + intent_name)
```

```
def lambda_handler(event, context):
    print(event)
    response = router(event)
    return response
```

已更新的字段列表

下表提供了有关 Amazon Lex V2 Lambda 请求和响应中更新字段的详细信息。您可以使用这些表在版本之间映射字段。

请求

以下字段已经以 Lambda 函数请求格式进行了更新。

活动的上下文

`activeContexts` 结构现在是 `sessionState` 结构的一部分。

V1 结构	V2 结构
activeContexts	sessionState.activeContexts
ActiveContext [*]。 timeToLive	sessionState.activeContexts [timeToLive
ActiveContext [*]。 timeToLive。 timeToLiveInSeconds	sessionState.activeContexts [timeToLive。 timeToLiveInSeconds
ActiveContext [*]。 timeToLive。 turnsToLive	sessionState.activeContexts [timeToLive。 turnsToLive
activeContexts[*].name	sessionState.activeContexts[*].name
activeContexts[*].parameters	sessionState.activeContexts[*].contextAttributes

替代意图

从索引 1 到 N 的解释列表包含 Amazon Lex V2 预测的替代意图列表及其置信度分数。`recentIntentSummaryView` 已从 Amazon Lex V2 的请求结构中删除。要从 `recentIntentSummaryView` 中查看详细信息，请使用 GetSession 操作。

V1 结构	V2 结构
alternativeIntents	interpretations[1:*]
recentIntentSummary查看	不适用

机器人

Amazon Lex V2 中，机器人和别名都有标识符。机器人 ID 是代码挂钩输入的一部分。包含别名 ID，但不包含别名名称。Amazon Lex V2 支持同一个机器人的多个区域设置，因此包含了区域设置 ID。

V1 结构	V2 结构
自动程序	自动程序
bot.name	bot.name
不适用	bot.id
bot.alias	不适用
不适用	bot.aliasId
bot.version	bot.version
不适用	bot.localeId

当前意图

`sessionState.intent` 结构包含活动意图的详细信息。Amazon Lex V2 还会返回 `interpretations` 结构中所有意图（包括替代意图）的列表。解释列表中的第一个元素始终与 `sessionState.intent` 相同。

V1 结构	V2 结构
currentIntent	sessionState.intent 或 interpretations[0].intent

V1 结构	V2 结构
currentIntent.name	sessionState.intent.name 或 interpretations[0].intent.name
当前意图。 nluConfidenceScore	interpretations[0].nluConfidence.score

对话操作

`confirmationStatus` 字段现在是 `sessionState` 结构的一部分。

V1 结构	V2 结构
currentIntent.confirmationStatus	sessionState.intent.confirmationState 或 interpretations[0].intent.confirmationState
不适用	sessionState.intent.state 或 interpretations[*].intent.state

Amazon Kendra

`kendraResponse` 字段现在是 `sessionState` 和 `interpretations` 结构的一部分。

V1 结构	V2 结构
kendraResponse	sessionState.intent.kendraResponse 或 interpretations[0].intent.kendraResponse

情绪

`sentimentResponse` 结构已移至新 `interpretations` 结构。

V1 结构	V2 结构
sentimentResponse	interpretations[0].sentimentResponse

V1 结构	V2 结构
sentimentResponse.sentimentLabel	interpretations[0].sentimentResponse.sentiment
sentimentResponse.sentimentScore	interpretations[0].sentimentResponse.sentimentScore

槽值

Amazon Lex V2 在 `sessionState.intent` 结构中提供了一个 `slots` 对象，其中包含用户所说内容的解析值、解释值和原始值。Amazon Lex V2 还通过将 `slotShape` 设置为 `List` 和设置 `values` 列表来支持多值插槽。`value` 字段支持单值插槽，假设它们的形状为 `Scalar`。

V1 结构	V2 结构
currentIntent.slots	sessionState.intent.slots 或 interpretations[0].intent.slots
currentIntent.slots[*].value	sessionState.intent.slots[*].value.interpretedValue 或 interpretations[0].intent.slots[*].value.interpretedValue
不适用	sessionState.intent.slots[*].value.shape 或 interpretations[0].intent.slots[*].shape
不适用	sessionState.intent.slots[*].values 或 interpretations[0].intent.slots[*].values
currentIntent.slotDetails	sessionState.intent.slots 或 interpretations[0].intent.slots
currentIntent.slotDetails[*].resolutions	sessionState.intent.slots[*].resolvedValues 或 interpretations[0].intent.slots[*].resolvedValues
currentIntent.slotDetails[*].originalValue	sessionState.intent.slots[*].originalValue 或 interpretations[0].intent.slots[*].originalValue

其他

Amazon Lex V2 `sessionId` 字段与 Amazon Lex V1 中的 `userId` 字段相同。Amazon Lex V2 还会发送呼叫方的 `inputMode`：文本、DTMF 或语音。

V1 结构	V2 结构
userId	sessionId
inputTranscript	inputTranscript
invocationSource	invocationSource
outputDialogMode	responseContentType
messageVersion	messageVersion
sessionAttributes	sessionState.sessionAttributes
requestAttributes	requestAttributes
不适用	inputMode
不适用	originatingRequestId

响应

以下字段已经以 Lambda 函数响应消息格式进行了更新。

活动的上下文

`activeContexts` 结构已移至 `sessionState` 结构。

V1 结构	V2 结构
activeContexts	sessionState.activeContexts
ActiveContext [*]。 timeToLive	sessionState.activeContexts [timeToLive

V1 结构	V2 结构
ActiveContext [*]。 timeToLive。 timeToLiveInSeconds	sessionState.activeContexts [timeToLive。 timeToLiveInSeconds
ActiveContext [*]。 timeToLive。 turnsToLive	sessionState.activeContexts [timeToLive。 turnsToLive
activeContexts[*].name	sessionState.activeContexts[*].name
activeContexts[*].parameters	sessionState.activeContexts[*].contextAttributes

对话操作

`dialogAction` 结构已移至 `sessionState` 结构。现在，您可以在对话框操作中指定多条消息，`genericAttachments` 结构现在是 `imageResponseCard` 结构。

V1 结构	V2 结构
dialogAction	sessionState.dialogAction
dialogAction.type	sessionState.dialogAction.type
DialogAction。 slotToElicit	sessionState.intent.dialog slotToElicit
dialogAction.type.fulfillmentState	sessionState.intent.state
dialogAction.message	消息
dialogAction.message.contentType	messages[*].contentType
dialogAction.message.content	messages[*].content
dialogAction.responseCard	消息 [*]。 imageResponseCard
dialogAction.responseCard.version	不适用
dialogAction.responseCard.contentType	messages[*].contentType

V1 结构	V2 结构
dialogAction.responseCard.genericAttachments	不适用
dialogAction.responseCard.genericAttachments[*].title	消息 [*]。 imageResponseCard. 标题
dialogAction.responseCard.genericAttachments[*].subTitle	消息 [*]。 imageResponseCard.subtitle
dialogAction.responseCard.genericAttachments[*].imageUrl	消息 [*]。 imageResponseCard.imageUrl
dialogAction.responseCard.genericAttachments[*].buttons	消息 [*]。 imageResponseCard. 按钮
dialogAction.responseCard.genericAttachments[*].buttons[*].value	消息 [*]。 imageResponseCard.buttons [*] .value
dialogAction.responseCard.genericAttachments[*].buttons[*].text	消息 [*]。 imageResponseCard.buttons [*] .text
DialogAction。 kendraQueryRequest有效载荷	DialogAction。 kendraQueryRequest有效载荷
DialogAction。 kendraQueryFilter字符串	DialogAction。 kendraQueryFilter字符串

意图和插槽

作为 `dialogAction` 结构一部分的意图和插槽字段现在已成为 `sessionState` 结构的一部分。

V1 结构	V2 结构
dialogAction.intentName	sessionState.intent.name
dialogAction.slots	sessionState.intent.slots
dialogAction.slots[*].key	sessionState.intent.slots[*].key

V1 结构	V2 结构
dialogAction.slots[*].value	sessionState.intent.slots[*].value.interpreted Value
不适用	sessionState.intent.slots[*].value.shape
不适用	sessionState.intent.slots[*].values

其他

`sessionAttributes` 结构现在是 `sessionState` 结构的一部分。`recentIntentSummaryReview` 结构已删除。

V1 结构	V2 结构
sessionAttributes	sessionState.sessionAttributes
recentIntentSummary查看	不适用

Amazon Lex 中的安全性

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的 安全性和云中 的安全性：

- 云安全 — AWS 负责保护在 AWS 云中运行 AWS 服务的基础架构。 AWS 还为您提供可以安全使用的服务。作为 AWS 合规性计划的一部分，我们的安全措施的有效性定期由第三方审计员进行测试和验证。要了解适用于 Amazon Lex 的合规性计划，请参阅合规性计划范围内的AWS 服务。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您组织的要求以及适用的法律法规。

此文档将帮助您了解如何在使用 Amazon Lex 时应用责任共担模型。以下主题说明如何配置 Amazon Lex 以实现您的安全性和合规性目标。您还将了解如何使用其他 AWS 服务来帮助您监控和保护您的 Amazon Lex 资源。

主题

Amazon Lex 中的数据保护

Amazon Lex 收集客户内容以进行故障排除并帮助改进服务。默认情况下，客户内容受到安全保护。您可以使用 Amazon Lex API 为单个客户删除内容。

Amazon Lex 存储四种类型的内容：

- 示例言语，用于构建和训练机器人
- 与机器人互动的用户提供的客户言语

- 会话属性，在用户与机器人互动期间提供特定于应用程序的信息
- 请求属性，其中包含适应用于向机器人发出的单个请求的信息

任何专为儿童设计的 Amazon Lex 机器人均受《儿童在线隐私保护法》(COPPA) 的约束。您可以使用控制台或 Amazon Lex API 将 `childDirected` 字段设置为 `true`，告诉 Amazon Lex 该机器人受 COPPA 的约束。当 `childDirected` 字段设置为 `true` 时，不存储任何用户言语。

主题

静态加密

Amazon Lex 会对其存储的用户言语进行加密。

主题

示例言语

当您开发机器人时，您可以为每个意图和插槽提供示例言语。您还可以为槽位提供自定义值和同义词。这些信息是静态加密的，用于构建机器人并创建用户体验。

客户言语

除非 `childDirected` 字段设置为 `true`，否则 Amazon Lex 会加密用户发送给机器人的言语。

当 `childDirected` 字段设置为 `true` 时，不存储任何用户言语。

当 `childDirected` 字段设置为 `false`（默认）时，用户言语会被加密并存储 15 天以供 GetUtterancesView 操作使用。要为特定用户删除存储的言语，请使用 DeleteUtterances 操作。

当机器人接受语音输入时，输入内容将被无限期存储。Amazon Lex 使用它来提高机器人响应用户输入的能力。

使用 DeleteUtterances 操作来为特定用户删除存储的言语。

会话属性

会话属性包含在 Amazon Lex 与客户端应用程序之间传递的特定于应用程序的信息。Amazon Lex 将会话属性传递给为机器人配置的所有 AWS Lambda 函数。如果 Lambda 函数添加或更新会话属性，Amazon Lex 会将新信息返回给客户端应用程序。

会话属性在会话持续时间内一直存在于加密存储中。您可以将会话配置为在最后一次用户言语之后保持活动最少 1 分钟，最多 24 小时。默认会话持续时间为 5 分钟。

请求属性

请求属性包含请求特定的信息，并仅应用于当前请求。客户端应用程序使用请求属性在运行时向 Amazon Lex 发送信息。

您可以使用请求属性传递不需要在整个会话中保留的信息。由于请求属性不会跨属性持久存在，所以不存储它们。

传输中加密

Amazon Lex 使用 HTTPS 协议与客户端应用程序进行通信。它使用 HTTPS 和 AWS 签名代表您的应用程序与其他服务（例如 Amazon Polly）进行通信。 AWS Lambda

密钥管理

Amazon Lex 使用内部密钥保护您的内容免遭未经授权的使用。

适用于 Amazon Lex 的 Identity and Access Management

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员可控制能够通过身份验证（登录）和获得授权（拥有权限）来使用 Amazon Lex 资源的用户。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您在 Amazon Lex 中所做的工作。

服务用户 — 如果您使用 Amazon Lex 服务来完成任务，则您的管理员会为您提供所需的凭证和权限。随着您使用更多 Amazon Lex 功能来完成任务，您可能需要额外的权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 Amazon Lex 中的功能，请参阅Amazon Lex 身份和访问问题排查。

服务管理员 — 如果您是贵公司 Amazon Lex 资源的管理人员，您可能拥有对 Amazon Lex 的完全访问权限。在此情况下，您需要确定可被服务用户访问的 Amazon Lex 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要了解有关贵公司如何将 IAM 与 Amazon Lex 搭配使用的更多信息，请参阅Amazon Lex 如何与 IAM 配合使用。

IAM 管理员 — 如果您是 IAM 管理员，可能需要了解关于如何编写策略以管理对 Amazon Lex 的访问权限的详细信息。要查看您可在 IAM 中使用的 Amazon Lex 基于身份的策略示例，请参阅Amazon Lex 基于身份的策略示例。

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （IAM Identity Center）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅《IAM 用户指南》中的用于签署 API 请求的AWS 签名版本 4。

无论使用何种身份验证方法，您都可能需要提供其他安全信息。例如， AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的多重身份验证和《IAM 用户指南》中的 IAM 中的AWS 多重身份验证。

AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求您以根用户身份登录的任务的完整列表，请参阅 IAM 用户指南中的需要根用户凭证的任务。

联合身份

作为最佳实践，要求人类用户（包括需要管理员访问权限的用户）使用与身份提供商的联合身份验证 AWS 服务 通过临时证书进行访问。

联合身份是指您的企业用户目录、Web 身份提供商、Identity C enter 目录中的用户，或者任何使用 AWS 服务 通过身份源提供的凭据进行访问的用户。 AWS Directory Service当联合身份访问时 AWS 账户，他们将扮演角色，角色提供临时证书。

要集中管理访问权限，建议您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中创建用户和群组，也可以连接并同步到您自己的身份源中的一组用户和群组，以便在您的所有 AWS 账户 和应用程序中使用。有关 IAM Identity Center 的信息，请参阅 AWS IAM Identity Center 用户指南中的什么是 IAM Identity Center？。

IAM 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的用例，应在需要时更新访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可以拥有一个名为的群组，IAMAdmins并向该群组授予管理 IAM 资源的权限。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的 IAM 用户的使用案例。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。要在中临时担任 IAM 角色 AWS Management Console，您可以从用户切换到 IAM 角色（控制台）。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的代入角色的方法。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问：要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关用于联合身份验证的角色的信息，请参阅《IAM 用户指南》中的针对第三方身份提供商创建角色（联合身份验证）。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。
- 临时 IAM 用户权限：IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取：您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅 IAM 用户指南中的 IAM 中的跨账户资源访问。
- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在服务中拨打电话时，该服务通常会在 Amazon 中运行应用程序 EC2 或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两项操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

- 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
- 服务相关角色-服务相关角色是一种与服务相关联的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

- 在 A@@ mazon 上运行的应用程序 EC2 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这比在 EC2 实例中存储访问密钥更可取。要为 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建一个附加到该实例的实例配置文件。实例配置文件包含角色并允许在 EC2 实例上运行的程序获得临时证书。有关更多信息，请参阅 IAM 用户指南中的使用 IAM 角色向在 A mazon EC2 实例上运行的应用程序授予权限。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅 IAM 用户指南中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 AWS Management Console AWS CLI、或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户托管策略定义自定义 IAM 权限。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括

AWS 托管策略和客户托管策略。要了解如何在托管策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

访问控制列表 (ACLs)

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。 ACLs 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持的服务示例 ACLs。 AWS WAF要了解更多信息 ACLs，请参阅《亚马逊简单存储服务开发者指南》中的访问控制列表 (ACL) 概述。

其他策略类型

AWS 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界：权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅IAM 用户指南中的 IAM 实体的权限边界。
- 服务控制策略 (SCPs)- SCPs 是指定组织或组织单位 (OU) 的最大权限的 JSON 策略 AWS Organizations。 AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。如果您启用组织中的所有功能，则可以将服务控制策略 (SCPs) 应用于您的任何或所有帐户。SCP 限制成员账户中的实体（包括每个 AWS 账户根用户实体）的权限。有关 Organization SCPs s 和的更多信息，请参阅《AWS Organizations 用户指南》中的服务控制策略。
- 资源控制策略 (RCPs) — RCPs 是 JSON 策略，您可以使用它来设置账户中资源的最大可用权限，而无需更新附加到您拥有的每个资源的 IAM 策略。RCP 限制成员账户中资源的权限，并可能影响身份（包括身份）的有效权限 AWS 账户根用户，无论这些身份是否属于您的组织。有关 Organizations 的更多信息 RCPs，包括 AWS 服务 该支持的列表 RCPs，请参阅《AWS Organizations 用户指南》中的资源控制策略 (RCPs)。

- 会话策略：会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅 IAM 用户指南中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

Amazon Lex 如何与 IAM 配合使用

在使用 IAM 管理对 Amazon Lex 的访问权限之前，您需要了解哪些 IAM 功能可与 Amazon Lex 配合使用。

将 IAM 功能与 Amazon Lex 配合使用

IAM 特征	Amazon Lex 支持
基于身份的策略	是
基于资源的策略	否
策略操作	是
策略资源	是
策略条件键（特定于服务）	是
ACLs	否
ABAC（策略中的标签）	部分
临时凭证	是
主体权限	是
服务角色	是
服务相关角色	是

要全面了解 Amazon Lex 和其他 AWS 服务如何与大多数 IAM 功能配合使用，请参阅 IAM 用户指南中与 IAM 配合使用的AWS 服务。

Amazon Lex 基于身份的策略

支持基于身份的策略：是

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户管理型策略定义自定义 IAM 权限。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

Amazon Lex 基于身份的策略示例

要查看 Amazon Lex 基于身份的策略的示例，请参阅Amazon Lex 基于身份的策略示例。

Amazon Lex 基于资源的策略

支持基于资源的策略：否

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户访问，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的主体。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当委托人和资源处于不同位置时 AWS 账户，可信账户中的 IAM 管理员还必须向委托人实体（用户或角色）授予访问资源的权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

Amazon Lex 的策略操作

支持策略操作：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 Action 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

有关 Amazon Lex 操作的列表，请参阅《服务授权参考》中的 Amazon Lex 定义的操作。

Amazon Lex 中的策略操作在操作前面使用以下前缀：

```
lex
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "lex:action1",
      "lex:action2"
         ]
```

您也可以使用通配符（*）指定多个操作。例如，要指定以单词 Describe 开头的所有操作，包括以下操作：

```
"Action": "lex:Describe*"
```

Amazon Lex 的策略资源

支持策略资源：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符（*）指示语句应用于所有资源。

```
"Resource": "*"
```

Amazon Lex 机器人资源 ARN 具有以下格式。

```
arn:aws:lex:${Region}:${Account}:bot:${Bot-Name}
```

有关格式的更多信息 ARNs，请参阅 Amazon 资源名称 (ARNs) 和 AWS 服务命名空间。

例如，要在语句中指定 `OrderFlowers` 自动程序，请使用以下 ARN。

```
"Resource": "arn:aws:lex:us-east-2:123456789012:bot:OrderFlowers"
```

要指定属于特定账户的所有自动程序，请使用通配符 (*)。

```
"Resource": "arn:aws:lex:us-east-2:123456789012:bot:*"
```

无法对特定资源执行某些 Amazon Lex 操作，例如用于创建资源的操作。在这些情况下，您必须使用通配符 (*)。

```
"Resource": "*"
```

要查看 Amazon Lex 资源类型及其列表 ARNs，请参阅《服务授权参考》中的 Amazon Lex 定义的资源。要了解您可以使用哪些操作指定每个资源的 ARN，请参阅 Amazon Lex 定义的操作。

Amazon Lex 的策略条件键

支持特定于服务的策略条件键：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 `Condition` 元素（或 `Condition` 块）中，可以指定语句生效的条件。`Condition` 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 `Condition` 元素，或在单个 `Condition` 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑OR运算来 AWS 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM 策略元素：变量和标签。

AWS 支持全局条件密钥和特定于服务的条件密钥。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

要查看 Amazon Lex 条件键的列表，请参阅《服务授权参考》中的 Amazon Lex 的条件键。要了解您可以对哪些操作和资源使用条件键，请参阅 Amazon Lex 定义的操作。

下表列出了适用于 Amazon Lex 资源的 Amazon Lex 条件键。您可以在 IAM 权限策略的 `Condition` 元素中包含这些键。

ACLs 在 Amazon Lex 中

支持 ACLs：否

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。 ACLs 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

ABAC 与 Amazon Lex

支持 ABAC（策略中的标签）：部分支持

基于属性的访问控制（ABAC）是一种授权策略，该策略基于属性来定义权限。在中 AWS，这些属性称为标签。您可以将标签附加到 IAM 实体（用户或角色）和许多 AWS 资源。标记实体和资源是 ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

要基于标签控制访问，您需要使用 `aws:ResourceTag/`*`key-name`*、`aws:RequestTag/`*`key-name`* 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息，请参阅《IAM 用户指南》中的使用 ABAC 授权定义权限。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

您可以将标签与某些类型的 Amazon Lex 资源关联起来进行授权。要基于标签控制访问，请使用 `lex:ResourceTag/${TagKey}`、`aws:RequestTag/${TagKey}` 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。

有关标记 Amazon Lex 资源的更多信息，请参阅为您的 Amazon Lex 资源添加标签。

要查看基于身份的策略（用于根据资源上的标签来限制对该资源的访问）的示例，请参阅 使用标签访问资源。

下表列出了基于标签的访问控制的操作和相应的资源类型。根据与相应资源类型关联的标签对每个操作进行授权。

操作	资源类型	条件键	备注
CreateBotVersion	自动程序	lex:ResourceTag	
DeleteBot	自动程序	lex:ResourceTag	
DeleteBotAlias	别名	lex:ResourceTag	
DeleteBotChannelAssociation	渠道	lex:ResourceTag	
DeleteBotVersion	自动程序	lex:ResourceTag	
DeleteSession	自动程序或别名	lex:ResourceTag	当别名设置为 $LATEST 时，使用与自动程序关联的标签。与其他别名一起使用时，使用与指定别名关联的标签。
DeleteUtterances	自动程序	lex:ResourceTag	
GetBot	自动程序或别名	lex:ResourceTag	当 versionOrAlias 设置为 $LATEST 或数字版本时，使用与自动程序关联的标签。与别名一起使用时，使用与指定别名关联的标签。
GetBotAlias	别名	lex:ResourceTag	

操作	资源类型	条件键	备注
GetBotChannelAssociation	通道	lex:ResourceTag	
GetBotChannelAssociations	通道	lex:ResourceTag	当别名设置为“-”时，使用与自动程序关联的标签。当指定自动程序别名时，使用与指定别名关联的标签
GetBotVersions	自动程序	lex:ResourceTag	
GetExport	自动程序	lex:ResourceTag	
GetSession	自动程序或别名	lex:ResourceTag	当别名设置为 $LATEST 时，使用与自动程序关联的标签。与其他别名一起使用时，使用与指定别名关联的标签。
GetUtterancesView	自动程序	lex:ResourceTag	
ListTagsForResource	自动程序、别名或通道	lex:ResourceTag	
PostContent	自动程序或别名	lex:ResourceTag	当别名设置为 $LATEST 时，使用与自动程序关联的标签。与其他别名一起使用时，使用与指定别名关联的标签。

操作	资源类型	条件键	备注
PostText	自动程序或别名	lex:ResourceTag	当别名设置为 $LATEST 时，使用与自动程序关联的标签。与其他别名一起使用时，使用与指定别名关联的标签。
PutBot	自动程序	lex:ResourceTag, aws:RequestTag, aws:TagKeys	
PutBotAlias	别名	lex:ResourceTag, aws:RequestTag, aws:TagKeys	
PutSession	自动程序或别名	lex:ResourceTag	当别名设置为 $LATEST 时，使用与自动程序关联的标签。与其他别名一起使用时，使用与指定别名关联的标签。
StartImport	自动程序	lex:ResourceTag	依赖于 PutBot 操作的访问策略。将忽略特定于 StartImport 操作的标签和权限。
TagResource	自动程序、别名或通道	lex:ResourceTag, aws:RequestTag, aws:TagKeys	

操作	资源类型	条件键	备注
UntagResource	自动程序、别名或通道	lex:ResourceTag, aws:RequestTag, aws:TagKeys	

使用 Amazon Lex 的临时凭证

支持临时凭证：是

当你使用临时证书登录时，有些 AWS 服务 不起作用。有关更多信息，包括哪些 AWS 服务 适用于临时证书，请参阅 IAM 用户指南中的AWS 服务 与 IAM 配合使用的信息。

如果您使用除用户名和密码之外的任何方法登录，则 AWS Management Console 使用的是临时证书。例如，当您 AWS 使用公司的单点登录 (SSO) 链接进行访问时，该过程会自动创建临时证书。当您以用户身份登录控制台，然后切换角色时，您还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的从用户切换到 IAM 角色（控制台）。

您可以使用 AWS CLI 或 AWS API 手动创建临时证书。然后，您可以使用这些临时证书进行访问 AWS。 AWS 建议您动态生成临时证书，而不是使用长期访问密钥。有关更多信息，请参阅 IAM 中的临时安全凭证。

可以使用临时凭证进行联合身份验证登录，分派 IAM 角色或分派跨账户角色。您可以通过调用AssumeRole或之类的 AWS STS API 操作来获取临时安全证书GetFederationToken。

Amazon Lex 的跨服务主体权限

支持转发访问会话（FAS）：是

当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两项操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

Amazon Lex 的服务角色

支持服务角色：是

服务角色是由一项服务担任、代表您执行操作的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

> **Warning**
>
> 更改服务角色的权限可能会破坏 Amazon Lex 的功能。仅当 Amazon Lex 提供相关指导时才编辑服务角色。

在 Amazon Lex 中选择 IAM 角色

Amazon Lex 使用与服务相关的角色来调用 Amazon Comprehend 和 Amazon Polly。它使用对您的 AWS Lambda 函数的资源级权限来调用它们。

您必须提供一个 IAM 角色才能启用对话标记。有关更多信息，请参阅 为对话日志创建 IAM 角色和策略。

Amazon Lex 的服务相关角色

支持服务相关角色：是

服务相关角色是一种与服务相关联的 AWS 服务服务角色。服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

有关创建或管理 Amazon Lex 服务相关角色的详细信息，请参阅对 Amazon Lex 使用服务相关角色。

Amazon Lex 基于身份的策略示例

默认情况下，用户和角色没有创建或修改 Amazon Lex 资源的权限。他们也无法使用 AWS Management Console、 AWS Command Line Interface (AWS CLI) 或 AWS API 执行任务。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略（控制台）。

有关 Amazon Lex 定义的操作和资源类型（包括每种资源类型的格式）的详细信息，请参阅《服务授权参考》中的 Amazon Lex 的操作、资源和条件密钥。 ARNs

主题

策略最佳实践

基于身份的策略确定某个人是否可以创建、访问或删除您账户中的 Amazon Lex 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的AWS 托管式策略或工作职能的AWS 托管式策略。
- 应用最低权限：在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限：您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定的方式使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 服务，例如 AWS CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言（JSON）和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM Access Analyzer 验证策略。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的使用 MFA 保护 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。

使用 Amazon Lex 控制台

要访问 Amazon Lex 控制台，您必须具有一组最低的权限。这些权限必须允许您列出和查看有关您的 Amazon Lex 资源的详细信息 AWS 账户。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

对于仅调用 AWS CLI 或 AWS API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与其尝试执行的 API 操作相匹配的操作。

AWS 通过提供由创建和管理的独立 IAM 策略来解决许多常见用例 AWS。这些策略称为 AWS 托管式策略。与必须自己编写策略相比，通过 AWS 托管式策略可以更轻松地将适当的权限分配给用户、组和角色。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管策略。

以下 AWS 托管策略仅适用于 Amazon Lex，您可以将其附加到账户中的群组和角色：

- AmazonLexReadOnly— 授予对 Amazon Lex 资源的只读访问权限。
- AmazonLexRunBotsOnly— 授予运行 Amazon Lex 对话机器人的访问权限。
- AmazonLexFullAccess— 授予创建、读取、更新、删除和运行所有 Amazon Lex 资源的完全访问权限。还授予将名称以 `AmazonLex` 开头的 Lambda 函数与 Amazon Lex 意图关联的能力。

> **Note**
>
> 通过登录 IAM 控制台并搜索特定策略，可以查看这些权限策略。

该AmazonLexFullAccess政策不向用户授予使用`KendraSearchIntent`意图查询 Amazon Kendra 索引的权限。要查询索引，必须向策略添加其他权限。有关所需的权限，请参阅Amazon Kendra 搜索的 IAM 策略。

您还可以创建自定义 IAM 策略，以授予执行 Amazon Lex API 操作的相关权限。您可以将这些自定义策略附加到需要这些权限的 IAM 角色或组。

有关对于 Amazon Lex 的 AWS 托管式策略的详细信息，请参阅AWS Amazon Lex 的托管策略。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

删除所有 Amazon Lex 机器人

此示例策略授予 AWS 账户中的用户删除您账户中任何机器人的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lex:DeleteBot"
```

```
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

允许用户将机器人迁移到 Amazon Lex V2 APIs

以下 IAM 权限策略允许用户开始将机器人从 Amazon Lex 迁移到 Amazon Lex V2， APIs 并查看迁移列表及其进度。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "startMigration",
            "Effect": "Allow",
            "Action": "lex:StartMigration",
            "Resource": "arn:aws:lex:<Region>:<123456789012>:bot:*"
        },
        {
            "Sid": "passRole",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::<123456789012>:role/<v2 bot role>"
        },
        {
            "Sid": "allowOperations",
            "Effect": "Allow",
            "Action": [
                "lex:CreateBot",
                "lex:CreateIntent",
                "lex:UpdateSlot",
                "lex:DescribeBotLocale",
                "lex:UpdateBotAlias",
                "lex:CreateSlotType",
                "lex:DeleteBotLocale",
                "lex:DescribeBot",
                "lex:UpdateBotLocale",
                "lex:CreateSlot",
                "lex:DeleteSlot",
```

```
                "lex:UpdateBot",
                "lex:DeleteSlotType",
                "lex:DescribeBotAlias",
                "lex:CreateBotLocale",
                "lex:DeleteIntent",
                "lex:StartImport",
                "lex:UpdateSlotType",
                "lex:UpdateIntent",
                "lex:DescribeImport",
                "lex:CreateCustomVocabulary",
                "lex:UpdateCustomVocabulary",
                "lex:DeleteCustomVocabulary",
                "lex:DescribeCustomVocabulary",
                "lex:DescribeCustomVocabularyMetadata"
            ],
            "Resource": [
                "arn:aws:lex:<Region>:<123456789012>:bot/*",
                "arn:aws:lex:<Region>:<123456789012>:bot-alias/*/*"
            ]
        },
        {
            "Sid": "showBots",
            "Effect": "Allow",
            "Action": [
                "lex:CreateUploadUrl",
                "lex:ListBots"
            ],
            "Resource": "*"
        },
        {
            "Sid": "showMigrations",
            "Effect": "Allow",
            "Action": [
                "lex:GetMigration",
                "lex:GetMigrations"
            ],
            "Resource": "*"
        }
    ]
}
```

使用标签访问资源

此示例策略授予您 AWS 账户中的用户或角色对任何用键 **Department** 和值 **Support** 标记的资源使用 PostText 操作的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "lex:PostText",
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "lex:ResourceTag/Department": "Support"
                }
            }
        }
    ]
}
```

AWS Amazon Lex 的托管策略

AWS 托管策略是由创建和管理的独立策略 AWS。 AWS 托管策略旨在为许多常见用例提供权限，以便您可以开始为用户、组和角色分配权限。

请记住， AWS 托管策略可能不会为您的特定用例授予最低权限权限，因为它们可供所有 AWS 客户使用。我们建议通过定义特定于您的使用场景的客户管理型策略来进一步减少权限。

您无法更改 AWS 托管策略中定义的权限。如果 AWS 更新 AWS 托管策略中定义的权限，则更新会影响该策略所关联的所有委托人身份（用户、组和角色）。 AWS 最有可能在启动新的 API 或现有服务可以使用新 AWS 服务 的 API 操作时更新 AWS 托管策略。

有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管策略。

AWS 托管策略： AmazonLexReadOnly

您可以将 AmazonLexReadOnly 策略附加到 IAM 身份。

此策略授予只读权限，允许用户查看 Amazon Lex 和 Amazon Lex V2 模型构建服务中的所有操作。

权限详细信息

该策略包含以下权限：

- lex — 模型构建服务中对 Amazon Lex 和 Amazon Lex V2 资源的只读访问权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lex:GetBot",
                "lex:GetBotAlias",
                "lex:GetBotAliases",
                "lex:GetBots",
                "lex:GetBotChannelAssociation",
                "lex:GetBotChannelAssociations",
                "lex:GetBotVersions",
                "lex:GetBuiltinIntent",
                "lex:GetBuiltinIntents",
                "lex:GetBuiltinSlotTypes",
                "lex:GetIntent",
                "lex:GetIntents",
                "lex:GetIntentVersions",
                "lex:GetSlotType",
                "lex:GetSlotTypes",
                "lex:GetSlotTypeVersions",
                "lex:GetUtterancesView",
                "lex:DescribeBot",
                "lex:DescribeBotAlias",
                "lex:DescribeBotChannel",
                "lex:DescribeBotLocale",
                "lex:DescribeBotVersion",
                "lex:DescribeExport",
                "lex:DescribeImport",
                "lex:DescribeIntent",
```

```
                "lex:DescribeResourcePolicy",
                "lex:DescribeSlot",
                "lex:DescribeSlotType",
                "lex:ListBots",
                "lex:ListBotLocales",
                "lex:ListBotAliases",
                "lex:ListBotChannels",
                "lex:ListBotVersions",
                "lex:ListBuiltInIntents",
                "lex:ListBuiltInSlotTypes",
                "lex:ListExports",
                "lex:ListImports",
                "lex:ListIntents",
                "lex:ListSlots",
                "lex:ListSlotTypes",
                "lex:ListTagsForResource"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 托管策略： AmazonLexRunBotsOnly

您可以将 AmazonLexRunBotsOnly 策略附加到 IAM 身份。

该策略授予只读权限，允许运行 Amazon Lex 和 Amazon Lex V2 对话机器人。

权限详细信息

该策略包含以下权限：

- lex — 对 Amazon Lex 和 Amazon Lex V2 运行时中的所有操作的只读访问权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lex:PostContent",
                "lex:PostText",
                "lex:PutSession",
```

```
                "lex:GetSession",
                "lex:DeleteSession",
                "lex:RecognizeText",
                "lex:RecognizeUtterance",
                "lex:StartConversation"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 托管策略： AmazonLexFullAccess

您可以将 AmazonLexFullAccess 策略附加到 IAM 身份。

该政策授予管理权限，允许用户创建、读取、更新和删除 Amazon Lex 和 Amazon Lex V2 资源，以及运行 Amazon Lex 和 Amazon Lex V2 对话机器人。

权限详细信息

该策略包含以下权限：

- lex — 向主体授予对 Amazon Lex 和 Amazon Lex V2 模型构建和运行时服务中的所有操作的读写权限。
- cloudwatch— 允许委托人查看 Amazon CloudWatch 指标和警报。
- iam：允许主体创建和删除服务相关角色、传递角色以及为角色附加和分离策略。Amazon Lex 操作的权限仅限于“lex.amazonaws.com”，而 Amazon Lex V2 操作的权限仅限于“lexv2.amazonaws.com”。
- kendra：允许主体列出 Amazon Kendra 索引。
- kms— 允许委托人描述 AWS KMS 密钥和别名。
- lambda— 允许委托人列出 AWS Lambda 函数并管理附加到任何 Lambda 函数的权限。
- polly：允许主体描述 Amazon Polly 的声音并合成话语。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DescribeAlarmsForMetric",
                "kms:DescribeKey",
                "kms:ListAliases",
                "lambda:GetPolicy",
                "lambda:ListFunctions",
                "lex:*",
                "polly:DescribeVoices",
                "polly:SynthesizeSpeech",
                "kendra:ListIndices",
                "iam:ListRoles",
                "s3:ListAllMyBuckets",
                "logs:DescribeLogGroups",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:AddPermission",
                "lambda:RemovePermission"
            ],
            "Resource": "arn:aws:lambda:*:*:function:AmazonLex*",
            "Condition": {
                "StringEquals": {
                    "lambda:Principal": "lex.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots",
                "arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/
AWSServiceRoleForLexChannels",
```

```
                "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/
AWSServiceRoleForLexV2Bots*",
                "arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/
AWSServiceRoleForLexV2Channels*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "lex.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/
AWSServiceRoleForLexChannels"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "channels.lex.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/
AWSServiceRoleForLexV2Bots*"
```

```
            ],
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "lexv2.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/
AWSServiceRoleForLexV2Channels*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "channels.lexv2.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:DeleteServiceLinkedRole",
                "iam:GetServiceLinkedRoleDeletionStatus"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots",
                "arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/
AWSServiceRoleForLexChannels",
                "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/
AWSServiceRoleForLexV2Bots*",
                "arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/
AWSServiceRoleForLexV2Channels*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
```

```
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "lex.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/
AWSServiceRoleForLexV2Bots*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "lexv2.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/
AWSServiceRoleForLexV2Channels*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "channels.lexv2.amazonaws.com"
                    ]
                }
```

```
            }
        }
    ]
}
```

Amazon Lex 更新 AWS 了托管政策

查看自该服务开始跟踪这些更改以来对 Amazon Lex AWS 托管政策的更新的详细信息。要获得有关此页面更改的自动提示，请订阅 Amazon Lex Amazon Lex 的文档历史记录 页面上的 RSS 源。

更改	描述	日期
AmazonLexFullAccess – 对现有策略的更新	Amazon Lex 添加了新的权限，允许对 Amazon Lex V2 模型构建服务操作进行只读访问。	2021 年 8 月 18 日
AmazonLexReadOnly – 对现有策略的更新	Amazon Lex 添加了新的权限，允许对 Amazon Lex V2 模型构建服务操作进行只读访问。	2021 年 8 月 18 日
AmazonLexRunBotsOnly – 对现有策略的更新	Amazon Lex 添加了新的权限，允许对 Amazon Lex V2 运行时服务操作进行只读访问。	2021 年 8 月 18 日
Amazon Lex 开始跟踪更改	Amazon Lex 开始跟踪其 AWS 托管式策略的更改。	2021 年 8 月 18 日

对 Amazon Lex 使用服务相关角色

Amazon Lex 使用 AWS Identity and Access Management (IAM) 服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 Amazon Lex 直接相关。服务相关角色由 Amazon Lex 预定义，包括该服务代表您调用其他 AWS 服务所需的所有权限。

服务相关角色可让您更轻松地设置 Amazon Lex，因为您不必手动添加必要的权限。Amazon Lex 定义其服务相关角色的权限，并且除非另有其他定义，否则只有 Amazon Lex 可以代入该角色。定义的权限包括信任策略和权限策略，而且权限策略不能附加到任何其它 IAM 实体。

只有在首先删除服务相关角色的相关资源后，才能删除该角色。这样可保护您的 Amazon Lex 资源，因为可避免您无意中删除对资源的访问权限。

Amazon Lex 的服务相关角色权限

Amazon Lex 使用两个服务相关角色：

- AWSServiceRoleForLexBots— Amazon Lex 使用此服务相关角色来调用 Amazon Polly 来为你的机器人合成语音响应，调用 Amazon Comprehend 进行情绪分析，也可以调用 Amazon Kendra 来搜索索引。
- AWSServiceRoleForLexChannels— Amazon Lex 在管理频道时使用此服务相关角色向您的机器人发布文本。

您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

为 Amazon Lex 创建服务相关角色

您无需手动创建服务相关角色。当您在中创建机器人、机器人频道或 Amazon Kendra 搜索意图时 AWS Management Console，Amazon Lex 会为您创建与服务相关的角色。

如果您删除该服务相关角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。当您创建新的机器人、通道关联或 Amazon Kendra 搜索意图时，Amazon Lex 将再次为您创建服务相关角色。

您也可以使用创建 AWS CLI 与AWSServiceRoleForLexBots用例相关的服务角色。在 AWS CLI 创建具有 Amazon Lex 服务名称`lex.amazonaws.com`的服务相关角色中。有关更多信息，请参阅 步骤 1：创建服务相关角色 (AWS CLI)。如果您删除了此服务相关角色，可以使用同样的过程再次创建角色。

为 Amazon Lex 编辑服务相关角色

Amazon Lex 不允许您编辑 Amazon Lex 服务相关角色。创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

删除 Amazon Lex 的服务相关角色

如果不再需要使用某个需要服务相关角色的功能或服务，我们建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。但是，必须先清除服务相关角色的资源，然后才能手动删除它。

> **Note**
>
> 如果当您试图删除资源时 Amazon Lex 服务正在使用该角色，则删除操作可能会失败。如果发生这种情况，请等待几分钟后重试。

删除服务相关角色所使用的 Amazon Lex 资源：

1. 删除您正在使用的任何自动程序通道。
2. 删除您的账户中的任何自动程序。

使用 IAM 手动删除服务相关角色

使用 IAM 控制台 AWS CLI、或 AWS API 删除 Amazon Lex 服务相关角色。有关更多信息，请参见《IAM 用户指南》中的删除服务相关角色。

Amazon Lex 服务相关角色支持的区域

Amazon Lex 支持在该服务可用的所有区域中使用服务相关角色。有关更多信息，请参阅 Amazon Lex 端点和限额。

Amazon Lex 身份和访问问题排查

您可以使用以下信息来帮助诊断和修复在使用 Amazon Lex 和 IAM 时可能遇到的常见问题。

主题

- 我无权在 Amazon Lex 中执行操作
- 我无权执行 iam：PassRole
- 我想允许我以外的人访问我 AWS 账户 的 Amazon Lex 资源

我无权在 Amazon Lex 中执行操作

如果您收到错误提示，指明您无权执行某个操作，则必须更新策略以允许执行该操作。

当 mateojackson IAM 用户尝试使用控制台查看有关虚构 *my-example-widget* 资源的详细信息，但不拥有虚构 lex:*GetWidget* 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 lex:GetWidget on resource: my-example-widget
```

在此情况下，必须更新 mateojackson 用户的策略，以允许使用 lex:*GetWidget* 操作访问 *my-example-widget* 资源。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我无权执行 iam：PassRole

如果您收到一个错误，指明您无权执行 iam:PassRole 操作，则必须更新您的策略以允许您将角色传递给 Amazon Lex。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 marymajor 的 IAM 用户尝试使用控制台在 Amazon Lex 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 iam:PassRole 操作。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我想允许我以外的人访问我 AWS 账户 的 Amazon Lex 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表 (ACLs) 的服务，您可以使用这些策略向人们授予访问您的资源的权限。

要了解更多信息，请参阅以下内容：

- 要了解 Amazon Lex 是否支持这些功能，请参阅Amazon Lex 如何与 IAM 配合使用。

- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

Amazon Lex 中的监控

要维护 Amazon Lex 对话聊天机器人的可靠性、可用性和性能，实施监控非常重要。本主题介绍如何使用亚马逊 CloudWatch 日志和 AWS CloudTrail 监控 Amazon Lex，并介绍了 Amazon Lex 运行时和频道关联指标。

主题

- 使用亚马逊监控 Amazon Lex CloudWatch
- 使用 AWS CloudTrail 日志监控 Amazon Lex API 调用

使用亚马逊监控 Amazon Lex CloudWatch

要追踪您的 Amazon Lex 机器人的运行状况，请使用亚马逊 CloudWatch。借助 CloudWatch，您可以获取账户中各个 Amazon Lex 操作或全球 Amazon Lex 运营的指标。您还可以设置 CloudWatch 警报，以便在一个或多个指标超过您定义的阈值时收到通知。例如，您可以监控在特定的时间段内发送给自动程序的请求数量、查看成功请求的延迟，或在错误数超出阈值时发出警报。

CloudWatch Amazon Lex 的指标

要获取 Amazon Lex 操作的指标，必须指定以下信息：

- 指标维度。维度是用来标识指标的一组名称/值对。Amazon Lex 有三个维度：
  - `BotAlias, BotName, Operation`
  - `BotAlias, BotName, InputMode, Operation`
  - `BotName, BotVersion, InputMode, Operation`
- 指标名称，如 `MissedUtteranceCount` 或 `RuntimeRequestCount`。

您可以通过 AWS Management Console、或 CloudWatch API 获取 Amazon Lex 的指标。 AWS CLI 您可以通过其中一个 Amazon AWS 软件开发套件 (SDKs) 或 CloudWatch API 工具来使用 API。 CloudWatch Amazon Lex 控制台根据来自 CloudWatch API 的原始数据显示图表。

您必须拥有相应的 CloudWatch 权限才能使用监控 Amazon Lex CloudWatch 。有关更多信息，请参阅《亚马逊 CloudWatch 用户指南》 CloudWatch中的 Amazon 身份验证和访问控制。

查看 Amazon Lex 指标

使用亚马逊 Lex 控制台或控制台查看 Amazon Lex 指标。 CloudWatch

查看指标（Amazon Lex 控制台）

1. 登录 AWS Management Console 并打开 Amazon Lex 控制台，网址为https://console.aws.amazon.com/lex/。
2. 从自动程序列表中，选择要查看其指标的自动程序。
3. 选择监控。此时指标以图表形式显示。

查看指标（CloudWatch 控制台）

1. 登录 AWS Management Console 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 依次选择指标、所有指标，然后选择 AWS/Lex。
3. 选择维度、指标名称，然后选择 添加到图表。
4. 选择日期范围的值。所选日期范围的指标计数将显示在该图表中。

创建警报

CloudWatch 警报在指定时间段内监视单个指标，并执行一项或多项操作：向亚马逊简单通知服务 (Amazon SNS) Simple Notification Service 主题或 Auto Scaling 策略发送通知。一个或多个操作基于指标在您指定的多个时间段内相对于给定阈值的值。 CloudWatch还可以在警报状态发生变化时向您发送 Amazon SNS 消息。

CloudWatch 警报仅在状态发生变化并且持续到您指定的时间段内时才会调用操作。

设置警报

1. 登录 AWS Management Console 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 依次选择 Alarms 和 Create Alarm。
3. 选择 AWS/Lex 指标，然后选择一个指标。
4. 对于 Time Range，请选择要监控的时间范围，然后选择 Next。
5. 输入名称和描述。
6. 对于 Whenever，选择 >=，然后键入一个最大值。
7. 如果 CloudWatch 要在达到警报状态时发送电子邮件，请在 “操作” 部分的 “每当此警报” 中选择 “状态为警报”。对于发送通知到，选择一个邮件列表，或选择新建列表然后创建一个新的邮件列表。
8. 预览警报预览部分中的警报。如果对警报满意，请选择 Create Alarm (创建警报)。

CloudWatchAmazon Lex 运行时的指标

下表介绍 Amazon Lex 运行时指标。

指标	描述
KendraIndexAccessError	Amazon Lex 无法访问您的 Amazon Kendra 索引的次数。 对于 Text 或 Speech InputMode 的 PostContent 操作的有效维度： • BotName, BotAlias, Operation, InputMode PostText 操作的有效维度： • BotName, BotAlias, Operation 单位：计数
KendraLatency	Amazon Kendra 对来自 AMAZON.KendraSearchIntent 的请求做出响应所花费的时间。

指标	描述
	对于 Text 或 Speech InputMode 的 PostContent 操作的有效维度： • BotName, BotVersion, Operation, InputMode • BotName, BotAlias, Operation, InputMode PostText 操作的有效维度： • BotName, BotVersion, Operation • BotName, BotAlias, Operation 单位：毫秒
KendraSuccess	从 AMAZON.KendraSearchIntent 到您的 Amazon Kendra 索引成功请求的数量。 对于 Text 或 Speech InputMode 的 PostContent 操作的有效维度： • BotName, BotVersion, Operation, InputMode • BotName, BotAlias, Operation, InputMode PostText 操作的有效维度： • BotName, BotVersion, Operation • BotName, BotAlias, Operation 单位：计数

指标	描述
KendraSystemErrors	Amazon Lex 无法查询 Amazon Kendra 索引的次数。 对于 Text 或 Speech InputMode 的 PostContent 操作的有效维度： • BotName, BotAlias, Operation, InputMode PostText 操作的有效维度： • BotName, BotAlias, Operation 单位：计数
KendraThrottledEvents	Amazon Kendra 限制来自 AMAZON.KendraSearchIntent 的请求的次数。 对于 Text 或 Speech InputMode 的 PostContent 操作的有效维度： • BotName, BotAlias, Operation, InputMode PostText 操作的有效维度： • BotName, BotAlias, Operation 单位：计数

指标	描述
MissedUtteranceCount	在指定时间段内未能识别的表达的数量。 对于 Text 或 Speech InputMode 的 PostContent 操作的有效维度： • BotName, BotVersion, Operation, InputMode • BotName, BotAlias, Operation, InputMode PostText 操作的有效维度： • BotName, BotVersion, Operation • BotName, BotAlias, Operation
RuntimeConcurrency	指定时间段内的并发连接数。RuntimeConcurrency 将被报告为 StatisticSet 。 对于 Text 或 Speech InputMode 的 PostContent 操作的有效维度： • 操作， BotName， BotVersion， InputMode • 操作， BotName， BotAlias， InputMode 其他操作的有效维度： • 操作， BotName， BotVersion • 操作， BotName， BotAlias 单位：计数

指标	描述
RuntimeInvalidLambdaResponses	指定时间段内无效 AWS Lambda (Lambda) 响应的数量。 对于 Text 或 Speech InputMode 的 PostContent 操作的有效维度： • BotName, BotAlias, Operation, InputMode PostText 操作的有效维度： • BotName, BotAlias, Operation
RuntimeLambdaErrors	指定时间段内的 Lambda 运行时错误数量。 对于 Text 或 Speech InputMode 的 PostContent 操作的有效维度： • BotName, BotAlias, Operation, InputMode PostText 操作的有效维度： • BotName, BotAlias, Operation
RuntimePollyErrors	在指定时段内的无效 Amazon Polly 响应的数量。 对于 Text 或 Speech InputMode 的 PostContent 操作的有效维度： • BotName, BotAlias, Operation, InputMode PostText 操作的有效维度： • BotName, BotAlias, Operation

指标	描述
RuntimeRequestCount	指定时间段内的运行时请求数。 对于 Text 或 Speech InputMode 的 PostContent 操作的有效维度： • BotName, BotVersion, Operation, InputMode • BotName, BotAlias, Operation, InputMode PostText 操作的有效维度： • BotName, BotVersion, Operation • BotName, BotAlias, Operation 单位：计数
RuntimeSucessfulRequestLatency **Important** 该指标是 RuntimeSuccessfulRequestLatency ，而不是 RuntimeSuccessfulRequestLatency 。	从发送请求到传回响应这段时间内成功的请求的延迟。 对于 Text 或 Speech InputMode 的 PostContent 操作的有效维度： • BotName, BotVersion, Operation, InputMode • BotName, BotAlias, Operation, InputMode PostText 操作的有效维度： • BotName, BotVersion, Operation • BotName, BotAlias, Operation 单位：毫秒

指标	描述
RuntimeSystemErrors	指定时间段内的系统错误的数量。系统错误的响应代码范围为 500 到 599。 对于 Text 或 Speech InputMode 的 PostContent 操作的有效维度： • BotName, BotAlias, Operation, InputMode PostText 操作的有效维度： • BotName, BotAlias, Operation 单位：计数
RuntimeThrottledEvents	受限制请求的数目。当 Amazon Lex 接收的请求数超出为账户设置的每秒事务数限制时，它会限制请求。如果经常超出为账户设置的限制，您可以请求提高限制。要请求提高限制，请参阅 AWS 服务限制。 对于 Text 或 Speech InputMode 的 PostContent 操作的有效维度： • BotName、 BotAlias、操作、 InputMode PostText 操作的有效维度： • BotName, BotAlias, Operation 单位：计数

指标	描述
RuntimeUserErrors	指定时间段内的用户错误的数量。用户错误的响应代码范围为 400 到 499。 对于 Text 或 Speech InputMode 的 PostContent 操作的有效维度： • BotName, BotAlias, Operation, InputMode PostText 操作的有效维度： • BotName, BotAlias, Operation 单位：计数

Amazon Lex 运行时指标使用 AWS/Lex 命名空间并以下面的维度提供指标。您可以在 CloudWatch 控制台中按维度对指标进行分组：

维度	描述
BotName, BotAlias, Operation, InputMode	按自动程序的别名、自动程序的名称、操作 (PostContent) 以及输入是文本还是语音形式，对指标进行分组。
BotName, BotVersion, Operation, InputMode	按自动程序的名称、自动程序的版本、操作 (PostContent) 以及输入是文本还是语音形式，对指标进行分组。
BotName, BotVersion, Operation	按自动程序的名称、自动程序版本以及操作 PostText 对指标进行分组。
BotName, BotAlias, Operation	按自动程序的名称、自动程序的别名以及操作 PostText 对指标进行分组。

CloudWatch Amazon Lex 渠道关联指标

通道关联是 Amazon Lex 与消息收发通道（如 Facebook）之间的关联。下表介绍 Amazon Lex 通道关联指标。

指标	描述
BotChannelAuthErrors	指定时间内消息收发通道返回的身份验证错误的数量。身份验证错误表示在创建通道期间提供的秘密令牌无效或者已过期。
BotChannelConfigurationErrors	指定时间段内的配置错误的数量。配置错误指示通道的一个或多个配置条目无效。
BotChannelInboundThrottledEvents	在指定时间段内 Amazon Lex 限制消息收发通道发送消息的次数。
BotChannelOutboundThrottledEvents	在指定时间段内从 Amazon Lex 到消息收发通道的出站事件被限制的次数。
BotChannelRequestCount	指定时间段内对通道发出的请求的数量
BotChannelResponseCardErrors	在指定时间段内 Amazon Lex 无法发布响应卡的次数。
BotChannelSystemErrors	在指定时间段内对于一个通道在 Amazon Lex 中发生的内部错误的数量。

Amazon Lex 通道关联指标使用 AWS/Lex 命名空间，并为以下维度提供指标。您可以在 CloudWatch 控制台中按维度对指标进行分组：

维度	描述
BotAlias, BotChannelName, BotName, Source	按自动程序的别名、通道名称、自动程序的名称以及流量来源对指标进行分组。

CloudWatch 对话日志的指标

Amazon Lex 为对话日志记录使用以下指标：

指标	描述
ConversationLogsAudioDeliverySuccess	在指定时间段内成功传输到 S3 存储桶的音频日志数量。 单位：计数
ConversationLogsAudioDeliveryFailure	在指定时间段内未能传输到 S3 存储桶的音频日志数量。传输失败表示为对话日志配置的资源出现错误。错误可能包括 IAM 权限不足、 AWS KMS 密钥无法访问或 S3 存储桶无法访问。 单位：计数
ConversationLogsTextDeliverySuccess	在指定时间段内成功传送到 CloudWatch Logs 的文本日志的数量。 单位：计数
ConversationLogsTextDeliveryFailure	在指定时间段内未能传送到 CloudWatch Logs 的文本日志的数量。传输失败表示为对话日志配置的资源出现错误。错误可能包括 IAM 权限不足、 AWS KMS 密钥无法访问或 CloudWatch 日志日志组无法访问。 单位：计数

Amazon Lex 对话日志指标使用 AWS/Lex 命名空间并为以下维度提供指标。您可以在 CloudWatch 控制台中按维度对指标进行分组。

维度	描述
BotAlias	按自动程序的别名对指标进行分组。
BotName	按自动程序的名称对指标进行分组。

维度	描述
BotVersion	按自动程序的版本对指标进行分组。

使用 AWS CloudTrail 日志监控 Amazon Lex API 调用

Amazon Lex 与 AWS CloudTrail一项服务集成，该服务提供用户、角色或 AWS 服务在 Amazon Lex 中采取的操作的记录。 CloudTrail 捕获一部分 Amazon Lex 的 API 调用作为事件，包括来自亚马逊 Lex 控制台的调用和对 Amazon Lex 的代码调用 APIs。如果您创建了跟踪，则可以允许将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括 Amazon Lex 的事件。如果您未配置跟踪，您仍然可以在 CloudTrail 控制台的 “事件历史记录” 中查看最新的事件。通过收集的信息 CloudTrail，您可以确定向 Amazon Lex 发出的请求、发出请求的 IP 地址、谁提出了请求、何时提出请求以及其他详细信息。

要了解更多信息 CloudTrail，包括如何配置和启用它，请参阅《AWS CloudTrail 用户指南》。

Amazon Lex 中的信息 CloudTrail

CloudTrail 在您创建 AWS 账户时已在您的账户上启用。当 Amazon Lex 中出现支持的事件活动时，该活动会与其他 AWS 服务 CloudTrail 事件一起记录在事件历史记录中。您可以在自己的 AWS 账户中查看、搜索和下载最近发生的事件。有关更多信息，请参阅使用事件历史记录查看 CloudTrail 事件。

要持续记录您的 AWS 账户中的事件，包括 Amazon Lex 的事件，请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到亚马逊简单存储服务 (Amazon S3) 存储桶。默认情况下，在控制台中创建跟踪记录时，此跟踪记录应用于所有 AWS 区域。跟踪记录 AWS 分区中所有区域的事件，并将日志文件传送到您指定的 S3 存储桶。此外，您可以配置其他 AWS 服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅：

- 创建跟踪概述
- CloudTrail 支持的服务和集成
- 为 CloudTrail 配置 Amazon SNS 通知
- 接收来自多个区域的 CloudTrail 日志文件和接收来自多个账户的 CloudTrail 日志文件

Amazon Lex 支持将以下操作作为事件记录在 CloudTrail 日志文件中：

- CreateBotVersion
- CreateIntentVersion
- CreateSlotTypeVersion

- DeleteBot
- DeleteBotAlias
- DeleteBotChannelAssociation
- DeleteBotVersion
- DeleteIntent
- DeleteIntentVersion
- DeleteSlotType
- DeleteSlotTypeVersion
- DeleteUtterances
- GetBot
- GetBotAlias
- GetBotAliases
- GetBotChannelAssociation
- GetBotChannelAssociations
- GetBots
- GetBotVersions
- GetBuiltinIntent
- GetBuiltinIntents
- GetBuiltinSlotTypes
- GetSlotTypeVersions
- GetUtterancesView
- PutBot
- PutBotAlias
- PutIntent
- PutSlotType

每个事件或日志条目都包含有关生成请求的人员信息。此信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 用户凭证发出的
- 请求是使用角色还是联合用户的临时安全凭证发出的
- 请求是否由其他 AWS 服务发出

有关更多信息，请参阅 CloudTrail userIdentity 元素。

有关 CloudTrail 日志中记录的 Amazon Lex 操作的信息，请参阅 Amazon Lex 模型构建服务。例如，对PutBotGetBot、和DeleteBot操作的调用会在 CloudTrail 日志中生成条目。在 Amazon Lex Runtime Service 、PostContent 和 PostText 中记录的操作记录不会写入日志。

示例：Amazon Lex 日志文件条目

跟踪是一种配置，允许将事件作为日志文件传输到您指定的 S3 存储桶。 CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关请求的操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序出现。

以下示例 CloudTrail 日志条目显示了调用PutBot操作的结果。

```
   {
                "eventVersion": "1.05",
                "userIdentity": {
                    "type": "AssumedRole | FederatedUser | IAMUser | Root | SAMLUser |
WebIdentityUser",
                    "principalId": "principal ID",
                    "arn": "ARN",
                    "accountId": "account ID",
                    "accessKeyId": "access key ID",
                    "userName": "user name"
                },
                "eventTime": "timestamp",
                "eventSource": "lex.amazonaws.com",
                "eventName": "PutBot",
                "awsRegion": "region",
                "sourceIPAddress": "source IP address",
                "userAgent": "user agent",
                "requestParameters": {
                    "name": "CloudTrailBot",
                    "intents": [
                        {
                            "intentVersion": "11",
                            "intentName": "TestCloudTrail"
                        }
                    ],
                    "voiceId": "Salli",
                    "childDirected": false,
                    "locale": "en-US",
```

```
                    "idleSessionTTLInSeconds": 500,
                    "processBehavior": "BUILD",
                    "description": "CloudTrail test bot",
                    "clarificationPrompt": {
                        "messages": [
                            {
                                "contentType": "PlainText",
                                "content": "I didn't understand you. What would you
like to do?"
                            }
                        ],
                        "maxAttempts": 2
                    },
                    "abortStatement": {
                        "messages": [
                            {
                                "contentType": "PlainText",
                                "content": "Sorry. I'm not able to assist at this
time."
                            }
                        ]
                    }
                },
                "responseElements": {
                    "voiceId": "Salli",
                    "locale": "en-US",
                    "childDirected": false,
                    "abortStatement": {
                        "messages": [
                            {
                                "contentType": "PlainText",
                                "content": "Sorry. I'm not able to assist at this
time."
                            }
                        ]
                    },
                    "status": "BUILDING",
                    "createdDate": "timestamp",
                    "lastUpdatedDate": "timestamp",
                    "idleSessionTTLInSeconds": 500,
                    "intents": [
                        {
                            "intentVersion": "11",
                            "intentName": "TestCloudTrail"
```

```
                        }
                    ],
                    "clarificationPrompt": {
                        "messages": [
                            {
                                "contentType": "PlainText",
                                "content": "I didn't understand you. What would you
like to do?"
                            }
                        ],
                        "maxAttempts": 2
                    },
                    "version": "$LATEST",
                    "description": "CloudTrail test bot",
                    "checksum": "checksum",
                    "name": "CloudTrailBot"
                },
                "requestID": "request ID",
                "eventID": "event ID",
                "eventType": "AwsApiCall",
                "recipientAccountId": "account ID"
            }
        }
```

Amazon Lex 的合规性验证

作为多项合规计划的一部分，第三方审计师对 Amazon Lex 的安全与 AWS 合规性进行评估。Amazon Lex 是一项符合 HIPAA 要求的服务。它符合 PCI、SOC 和 ISO 标准。您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅在 AWS Artifact 中下载报告。

您在使用 Amazon Lex 时的合规性责任由您数据的敏感性、您组织的合规性目标以及适用的法律法规决定。如果您对 Amazon Lex 的使用需遵守 PCI 等标准， AWS 提供了以下有用资源：

- 安全与合规性快速入门指南 — 部署指南，讨论架构注意事项，并提供在上部署以安全性和合规性为重点的基准环境的步骤 AWS
- 《设计符合 HIPAA 安全性和合规性要求的架构》白皮书 — 此白皮书介绍公司如何使用 AWS 创建符合 HIPAA 要求的应用程序。
- AWS 合规性资源 — 可能适用于您的行业和所在地的业务手册和指南集合
- AWS Config — 评估资源配置对内部实践、行业指南和法规的遵循情况的服务

- AWS Security Hub— 全面了解您的安全状态 AWS ，可帮助您检查自己是否符合安全行业标准和最佳实践

有关特定合规计划范围内 AWS 服务的列表，请参阅按合规性计划划分的范围内的 AWS 服务。有关一般信息，请参阅 AWS 合规性计划。

Amazon Lex 中的恢复功能

AWS 全球基础设施是围绕 AWS 区域和可用区构建的。 AWS 区域提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络相连。利用可用区，您可以设计和操作在可用区之间无中断地自动实现失效转移的应用程序和数据库。与传统的单个或多个数据中心基础架构相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

除了 AWS 全球基础设施外，Amazon Lex 还提供多项功能来帮助支持您的数据弹性和备份需求。

Amazon Lex 中的基础设施安全性

作为一项托管服务，Amazon Lex 由 Amazon Web Services：安全流程概览白皮书中所述的 AWS 全球网络安全流程提供保护。

您可以使用已发布 AWS 的 API 调用通过网络访问 Amazon Lex。客户端必须支持 TLS（传输层安全性）1.0。建议使用 TLS 1.2 或更高版本。客户端还必须支持具有完全向前保密（PFS）的密码套件，例如 Ephemeral Diffie-Hellman（DHE）或 Elliptic Curve Ephemeral Diffie-Hellman（ECDHE）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

您可以从任何网络位置调用这些 API 操作，但 Amazon Lex 支持基于资源的访问策略，其中可以包含基于源 IP 地址的限制。您还可以使用 Amazon Lex 策略来控制来自特定亚马逊虚拟私有云（亚马逊 VPC）终端节点或特定终端节点的访问 VPCs。实际上，这可以将对给定 Amazon Lex 资源的网络访问与 AWS 网络中的特定 VPC 隔离开来。

Amazon Lex 中的准则和限额

以下各节分别提供了使用 Amazon Lex 时的准则和限额。

主题

支持的区域

有关提供 Amazon Lex 的 AWS 区域列表，请参阅《亚马逊网络服务通用参考》中的 AWS 区域和终端节点。

一般指导原则

本节描述了使用 Amazon Lex 时的一般准则。

- 签名请求 — API 参考 中的所有 Amazon Lex 模型构建和运行时 API 操作都使用签名 V4 来验证请求。有关验证请求的更多信息，请参阅《Amazon Web Services 一般参考》 中的签名版本 4 签名流程。

  对于 PostContent，Amazon Lex 使用《Amazon Simple Storage Service (S3) API 参考》中的 Authorization 标头签名计算：在单个数据块中传输负载（AWS 签名版本 4）中所述的无符号负载选项。

  当您使用无符号负载选项时，请勿在规范请求中包括有效负载的哈希。而应使用文字字符串“UNSIGNED-PAYLOAD”作为负载的哈希。还应在 `PostContent` 请求中包含一个名为 `x-amz-content-sha256` 并且值为 `UNSIGNED-PAYLOAD` 的标头。

- 请注意以下有关 Amazon Lex 从用户言语中捕获插槽值的方式的信息：

Amazon Lex 使用您在插槽类型定义中提供的枚举值来训练其机器学习模型。假设您使用以下示例表达定义了名为 GetPredictionIntent 的目的：

```
"Tell me the prediction for {Sign}"
```

其中 {Sign} 是自定义类型 ZodiacSign 的槽。它具有 12 个枚举值 (Aries 到 Pisces)。从用户言语“告诉我预测 ...”中，Amazon Lex 了解到接下来要表达的是星座。

如果使用 PutSlotType 操作将 valueSelectionStrategy 字段设置为 ORIGINAL_VALUE，或如果在控制台中选中了扩展值，则当用户说“告诉我对地球的预测”时，Amazon Lex 将推断“地球”为 ZodiacSign 并将它传递到您的客户端应用程序或 Lambda 函数。您必须检查槽值是有效的，然后才能在完成活动中使用它们。

如果您使用 PutSlotType 操作将 valueSelectionStrategy 字段设置为 TOP_RESOLUTION，或如果在控制台中选中了 Restrict to slot values and synonyms，则返回的值将限制为您为槽类型定义的值。例如，如果用户说“Tell me the prediction for earth”，则无法识别此值，因为它不是为槽类型定义的值之一。如果您定义同义词作为槽值，则会将同义词视为与槽值相同，但是，将返回槽值而不是同义词。

当 Amazon Lex 调用 Lambda 函数或返回与您的客户端应用程序的语音交互的结果时，无法保证返回插槽值。例如，如果引发 AMAZON.Movie 内置插槽类型的插槽值，而用户说出或键入“Gone with the wind”，Amazon Lex 可能会返回“Gone with the Wind”、“gone with the wind”或“Gone With The Wind”。在文本交互中，槽值的大小写将与输入的文本或槽值匹配 (具体取决于 valueResolutionStrategy 字段的值)。

- 定义包含首字母缩略词的插槽值时，请使用以下模式：
  - 用句点分隔的大写字母 (D.V.D.)
  - 用空格分隔的大写字母 (D V D)

- Amazon Lex 不支持 AMAZON.LITERAL 内置插槽类型，但 Alexa Skills Kit 支持此类型。但是，Amazon Lex 支持创建您可用于实施此功能的自定义插槽类型。如上一要点中提到，您可捕获自定义槽类型定义外部的值。可添加更多不同的枚举值来提升自动语音识别 (ASR) 和自然语言理解 (NLU) 的准确度。

- AMAZON.DATE 和 AMAZON.TIME 内置槽类型可同时捕获绝对和相对日期和时间。相对日期和时间在 Amazon Lex 处理请求的区域中解析。

  对于 AMAZON.TIME 内置插槽类型，如果用户不指定是在上午还是下午，则时间是不确定的，Amazon Lex 会再次提示用户。建议用户提示引发绝对时间。例如，使用这类提示：“您希望披萨什么时候送到？ 您可以说下午 6 点或晚上 6 点。”

- 在机器人中提供易混淆的训练数据将减弱 Amazon Lex 理解用户输入的能力。请考虑以下示例：

  假设自动程序包含两个目的 (OrderPizza 和 OrderDrink)，并且这两个目的都是使用“我需要订购”表达配置的。在构建时为机器人构建语言模型时，此言语不会映射到 Amazon Lex 可了解的特定意图。因此，当用户在运行时输入此言语时，Amazon Lex 无法高度自信地选择意图。

  再举个例子，您定义自定义目的 (例如，MyCustomConfirmationIntent) 以获取用户确认并使用表达“是”和“否”来配置目的。请注意，Amazon Lex 还有一个理解用户确认的语言模型。这可能会导致出现冲突。当用户回复“是”时，这意味着确认进行中的目的还是用户在请求您创建的自定义目的？

  通常，您提供的示例表达应映射到特定目的和 (可选) 特定槽值。

- 运行时 API 操作 PostContent 和 PostText 采用用户 ID 作为必需参数。开发人员可将此设置为满足 API 中所述约束的任何值。建议您不要使用此参数发送任何机密信息 (如用户登录名、电子邮件或身份证号)。此 ID 主要用于唯一识别与自动程序的对话 (可能有多个用户订购披萨)。

- 如果您的客户端应用程序使用 Amazon Cognito 进行身份验证，您可使用 Amazon Cognito 用户 ID 作为 Amazon Lex 用户 ID。请注意，为机器人配置的任何 Lambda 函数都必须有自己的身份验证机制才能标识 Amazon Lex 代表其调用 Lambda 函数的用户。

- 我们鼓励您定义捕获用户中断对话的意图的目的。例如，您可以定义具有示例言语（“我什么都不需要”、“退出”、“再见”）但没有插槽和配置为代码挂钩的 Lambda 函数的意图 (`NothingIntent`)。这将使用户能够正常关闭对话。

限额

本节介绍 Amazon Lex 中的当前限额。这些配额按类别进行分组。

服务限额可以调整或增加。要请求增加限额，请与 AWS 客户支持联系。增加服务限额可能需要几天时间。要在大型项目中增加限额，请将这段时间添加到您的计划中。

主题

运行时服务配额

除了 API 参考中所述的配额外，请注意以下几点：

API 配额

- PostContent 操作的语音输入可长达 15 秒。

- 在运行时 API 操作 PostContent 和 PostText 中，输入文本大小最多可达 1024 个 Unicode 字符。

- `PostContent` 标头的最大大小为 16 KB。请求和会话标头组合的最大大小为 12 KB。

- 在文本模式下使用 PostContent 或 PostText 操作时，与机器人对话的最大并发数量，对于 $LATEST 别名为 2，对于其他所有别名为 50。限额分别应用于每个 API。

- 在语音模式下使用 PostContent 操作时，与机器人的文本模式对话的最大并发数量，对于 $LATEST 别名为 2，对于所有其他别名为 125。限额分别应用于每个 API。

- 会话管理呼叫（PutSession、GetSession、和 DeleteSession）的最大并发数量，对于机器人的 $LATEST 别名为 2，对于所有其他别名为 50。

- Lambda 函数的最大输入大小为 12 KB。最大输出大小为 25 KB，其中 12 KB 可以为会话属性。

使用 **$LATEST** 版本

- $LATEST 版本的机器人只应用于手动测试。Amazon Lex 限制可对 $LATEST 版本的机器人进行的运行时请求的数量。

- 更新机器人的 $LATEST 版本时，Amazon Lex 将终止使用 $LATEST 版本的机器人的所有客户端应用程序正在进行的所有对话。一般而言，不应在生产环境中使用 $LATEST 版本的自动程序，因为 $LATEST 版本可能会更新。您应该发布一个版本并使用它。

- 更新别名时，Amazon Lex 需要几分钟使更改生效。修改 $LATEST 版本的自动程序时，更改会立即生效。

会话超时

- 创建自动程序时设置的会话超时将决定自动程序保留对话上下文 (如当前用户目的和槽数据) 多长时间。

- 在用户开始与您的机器人的对话后，一直到会话过期，Amazon Lex 都将使用相同的机器人版本（即使您更新机器人别名以指向另一版本也是如此）。

模型构建配额

模型构建是指创建和管理自动程序。这包括创建和管理自动程序、目的、槽类型、槽和自动程序通道关联。

主题

自动程序配额

- 您可以在整个模型构建 API 中配置提示和语句。其中每个提示或语句最多可包含 5 条消息，每条消息可包含 1 到 1000 个 UTF-8 字符。

- 使用消息组时，您可以为每条消息定义最多五个消息组。每个消息组最多可以包含五条消息，并且所有消息组中最多可以包含 15 条消息。

- 您可以为目的和槽定义示例表达。所有表达最多可用 200000 个字符。

- 每个槽类型最多可定义 10000 个值和同义词。每个自动程序最多可包含 50000 个槽类型值和同义词。

- 自动程序、别名和自动程序通道关联名称在创建时不区分大小写。如果创建 `PizzaBot` 后尝试创建 `pizzaBot`，则会出现错误。但是，在访问资源时，资源名称区分大小写，您必须指定 `PizzaBot`，而不是 `pizzaBot`。这些名称必须在 2 到 50 个 ASCII 字符之间。

- 可为所有资源类型发布的版本的最大数量为 100。请注意，没有适用于别名的版本控制。

- 在自动程序内，目的名称和槽名称必须是唯一的，即目的和槽不能同名。

- 您可以创建一个配置为支持多个目的的自动程序。如果两个目的有一个同名槽，则对应的槽类型必须相同。

  例如，假设创建了一个支持两个目的 (`OrderPizza` 和 `OrderDrink`) 的自动程序。如果这两个目的具有 `size` 槽，则该槽类型在两个位置必须相同。

  此外，为槽 (在其中一个目的中) 提供的示例表达也适用于其他目的中的同名槽。

- 一个机器人最多可与 250 个意图关联。

- 创建自动程序时，可以指定会话超时。会话超时可介于 1 分钟到 1 天之间。默认值为 5 分钟。

- 您可以为一个自动程序创建最多 5 个别名。

- 每个 AWS 账户可以创建最多 250 个机器人。

- 您无法创建从同一内置目的扩展的多个目的。

目的配额

- 目的和槽名称在创建时不区分大小写。也就是说，如果创建 OrderPizza 目的后尝试创建另一个 orderPizza 目的，则会发生错误。但是，在访问这些资源时，资源名称区分大小写，应指定 OrderPizza，而不是 orderPizza。这些名称必须在 1 到 100 个 ASCII 字符之间。

- 目的最多可包含 1,500 种示例表达。必须有至少一个示例表达。每个示例表达长度最多可为 200 个 UTF-8 字符。一个自动程序中的所有目的和槽表达最多可用 200000 个字符。目的的示例表达：
  - 可以引用零个或多个槽名称。
  - 可以只引用一次槽名称。

  例如：

  ```
  I want a pizza
  I want a {pizzaSize} pizza
  I want a {pizzaSize} {pizzaTopping} pizza
  ```

- 尽管每个意图支持多达 1500 个表达，如果使用的表达较少，Amazon Lex 可以更好地识别您提供的集合之外的输入。

- 您可以为意图中的每条消息创建最多五个消息组。消息的所有消息组中总共可以有 15 条消息。

- 控制台只能为 conclusionStatement 和 followUpPrompt 消息创建消息组。您可以使用 Amazon Lex API 为任何其他消息创建消息组。

- 每个槽可包含最多 10 种示例表达。每种示例表达必须精确引用一次槽名称。例如：

  ```
  {pizzaSize} please
  ```

- 每个自动程序最多可以有 200000 个字符用于目的和槽表达组合。

- 您无法为从内置目的扩展的目的提供表达。对于所有其他目的，您必须提供至少一种示例表达。目的包含槽，但槽级别示例表达是可选的。

- 内置意图
  - 目前，Amazon Lex 不支持针对内置意图插槽引发。您无法在包含派生自内置意图的意图响应中创建 Lambda 函数以返回 `ElicitSlot` 指令。有关更多信息，请参阅 响应格式。
  - 该服务不支持向内置目的添加示例表达。同样，您无法为内置目的添加或删除槽。

- 每个 AWS 账户可以创建最多 1000 个意图。一个目的中可以创建最多 100 个槽。

槽类型配额

- 槽类型名称在创建时不区分大小写。如果创建 `PizzaSize` 槽类型后再次尝试创建 `pizzaSize` 槽类型，则会发生错误。但是，在访问这些资源时，资源名称区分大小写 (您必须指定 `PizzaSize`，而不是 `pizzaSize`)。名称必须在 1 到 100 个 ASCII 字符之间。

- 创建的自定义槽类型最多可包含 10000 个枚举值和同义词。每个值最多 140 个 UTF-8 字符。枚举值和同义词不可重复。

- 对于槽类型值，请在适当时指定大写和小写。例如，对于名为 `Procedure` 的槽类型，如果值为 `MRI`，则将“MRI”和“mri”都指定为值。

- 内置插槽类型 — 目前，Amazon Lex 不支持为内置插槽类型添加枚举值或同义词。

API 参考

本节提供 Amazon Lex API 操作的文档。有关提供了 Amazon Lex 的 AWS 区域的列表，请参阅《Amazon Web Services 一般参考》中的 AWS 区域和端点。

主题

- 操作
- 数据类型

操作

Amazon Lex 模型构建服务支持以下操作：

- CreateBotVersion
- CreateIntentVersion
- CreateSlotTypeVersion
- DeleteBot
- DeleteBotAlias
- DeleteBotChannelAssociation
- DeleteBotVersion
- DeleteIntent
- DeleteIntentVersion
- DeleteSlotType
- DeleteSlotTypeVersion
- DeleteUtterances
- GetBot
- GetBotAlias
- GetBotAliases
- GetBotChannelAssociation
- GetBotChannelAssociations
- GetBots

- GetBotVersions
- GetBuiltinIntent
- GetBuiltinIntents
- GetBuiltinSlotTypes
- GetExport
- GetImport
- GetIntent
- GetIntents
- GetIntentVersions
- GetMigration
- GetMigrations
- GetSlotType
- GetSlotTypes
- GetSlotTypeVersions
- GetUtterancesView
- ListTagsForResource
- PutBot
- PutBotAlias
- PutIntent
- PutSlotType
- StartImport
- StartMigration
- TagResource
- UntagResource

Amazon Lex 运行时服务支持以下操作：

- DeleteSession
- GetSession
- PostContent
- PostText

- PutSession

Amazon Lex 模型构建服务

Amazon Lex 模型构建服务支持以下操作：

- CreateBotVersion
- CreateIntentVersion
- CreateSlotTypeVersion
- DeleteBot
- DeleteBotAlias
- DeleteBotChannelAssociation
- DeleteBotVersion
- DeleteIntent
- DeleteIntentVersion
- DeleteSlotType
- DeleteSlotTypeVersion
- DeleteUtterances
- GetBot
- GetBotAlias
- GetBotAliases
- GetBotChannelAssociation
- GetBotChannelAssociations
- GetBots
- GetBotVersions
- GetBuiltinIntent
- GetBuiltinIntents
- GetBuiltinSlotTypes
- GetExport
- GetImport
- GetIntent
- GetIntents

- GetIntentVersions
- GetMigration
- GetMigrations
- GetSlotType
- GetSlotTypes
- GetSlotTypeVersions
- GetUtterancesView
- ListTagsForResource
- PutBot
- PutBotAlias
- PutIntent
- PutSlotType
- StartImport
- StartMigration
- TagResource
- UntagResource

CreateBotVersion

服务: Amazon Lex Model Building Service

根据 $LATEST 版本创建新版本的机器人。如果自您创建上一个版本以来此资源的 $LATEST 版本没有更改，则 Amazon Lex 不会创建新版本。它返回上次创建的版本。

> **Note**
>
> 您只能更新 $LATEST 版本的机器人。您无法更新通过 CreateBotVersion 操作创建的带编号的版本。

创建机器人的第一个版本时，Amazon Lex 会将版本设置为 1。后续版本递增 1。有关更多信息，请参阅 版本控制。

此操作需要 lex:CreateBotVersion 操作权限。

请求语法

```
POST /bots/name/versions HTTP/1.1
Content-type: application/json

{
   "checksum": "string"
}
```

URI 请求参数

请求使用以下 URI 参数。

name

要为其创建新版本的机器人的名称。该名称区分大小写。

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

必需：是

请求体

请求接受采用 JSON 格式的以下数据。

checksum

标识机器人 $LATEST 版本的特定修订版。如果您指定了校验和，而机器人的 $LATEST 版本具有不同的校验和，则会返回 PreconditionFailedException 异常，并且 Amazon Lex 不会发布新版本。如果不指定校验和，Amazon Lex 会发布 $LATEST 版本。

类型：字符串

必需：否

响应语法

```
HTTP/1.1 201
Content-type: application/json

{
   "abortStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "checksum": "string",
   "childDirected": boolean,
   "clarificationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
```

```
    "createdDate": number,
    "description": "string",
    "detectSentiment": boolean,
    "enableModelImprovements": boolean,
    "failureReason": "string",
    "idleSessionTTLInSeconds": number,
    "intents": [
       {
          "intentName": "string",
          "intentVersion": "string"
       }
    ],
    "lastUpdatedDate": number,
    "locale": "string",
    "name": "string",
    "status": "string",
    "version": "string",
    "voiceId": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 201 响应。

服务以 JSON 格式返回的以下数据。

abortStatement

Amazon Lex 用来取消对话的消息。有关更多信息，请参阅 PutBot。

类型：Statement 对象

checksum

校验和标识创建的机器人版本。

类型：字符串

childDirected

对于使用 Amazon Lex 模型构建服务创建的每个 Amazon Lex 机器人，您都必须通过在 `childDirected` 字段中指定 `true` 或 `false`，指定您对 Amazon Lex 的使用是否与全部或部分针对 13 岁以下儿童且受《儿童在线隐私保护法》(COPPA) 约束的网站、程序或其他应用程序有关。在 `childDirected` 字段中指定 `true`，即表示您确认您对 Amazon Lex 的使用确实与全部或部分

针对 13 岁以下儿童且受 COPPA 约束的网站、计划或其他应用程序有关。在 childDirected 字段中指定 false，即表示您确认您对 Amazon Lex 的使用不与全部或部分针对 13 岁以下儿童且受 COPPA 约束的网站、计划或其他应用程序有关。如果在 childDirected 字段中指定默认值不能正确反映您确认您对 Amazon Lex 的使用不与全部或部分针对 13 岁以下儿童且受 COPPA 约束的网站、计划或其他应用程序有关，则您可以不指定。

如果您对 Amazon Lex 的使用涉及全部或部分针对 13 岁以下儿童的网站、程序或其他应用程序，则必须获得 COPPA 规定的任何必需的可核实的家长同意。有关将 Amazon Lex 用于全部或部分针对 13 岁以下儿童的网站、程序或其他应用程序的信息，请参阅 Amazon Lex 常见问题解答。

类型：布尔值

clarificationPrompt

Amazon Lex 在无法理解用户的请求时使用的消息。有关更多信息，请参阅 PutBot。

类型：Prompt 对象

createdDate

创建机器人版本的日期。

类型：时间戳

description

机器人的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

detectSentiment

表示是否应将用户输入的言语发送到 Amazon Comprehend 以进行情绪分析。

类型：布尔值

enableModelImprovements

表示机器人是否使用精度改进。true 表示机器人正在使用改进，否则为 false。

类型：布尔值

failureReason

如果 status 是 FAILED，则 Amazon Lex 会提供其未能构建机器人的原因。

类型：字符串

idleSessionTTLInSeconds

Amazon Lex 保留对话中收集的数据的最长时间（秒）。有关更多信息，请参阅 PutBot。

类型：整数

有效范围：最小值为 60。最大值为 86400。

intents

Intent 数据元数组。有关更多信息，请参阅 PutBot。

类型：Intent 对象数组

lastUpdatedDate

此机器人 $LATEST 版本的更新日期。

类型：时间戳

locale

指定机器人的目标区域设置。

类型：字符串

有效值：de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR

name

机器人的名称。

类型：字符串

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

status

当您发送创建或更新机器人的请求时，Amazon Lex 会将 status 响应元素设置为 BUILDING。在 Amazon Lex 构建机器人之后，它会将 status 设置为 READY。如果 Amazon Lex 无法构建机器人，则它会将 status 设置为 FAILED。Amazon Lex 会在 failureReason 响应元素中返回失败的原因。

类型：字符串

有效值：BUILDING | READY | READY_BASIC_TESTING | FAILED | NOT_BUILT

version

自动程序的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

voiceId

Amazon Lex 用于和用户进行语音交互的 Amazon Polly 语音 ID。

类型：字符串

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

PreconditionFailedException

您尝试更改的资源的校验和与请求中的校验和不匹配。检查资源的校验和并重试。

HTTP 状态代码：412

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

CreateIntentVersion

服务: Amazon Lex Model Building Service

基于意图的 $LATEST 版本创建意图的新版本。如果自您上次更新以来此意图的 $LATEST 版本没有更改，则 Amazon Lex 不会创建新版本。它返回您创建的上一个版本。

> **Note**
>
> 您只能更新 $LATEST 版本的意图。您无法更新通过 CreateIntentVersion 操作创建的带编号的版本。

创建意图的版本时，Amazon Lex 会将版本设置为 1。后续版本递增 1。有关更多信息，请参阅 版本控制。

此操作需要执行 lex:CreateIntentVersion 操作的权限。

请求语法

```
POST /intents/name/versions HTTP/1.1
Content-type: application/json

{
   "checksum": "string"
}
```

URI 请求参数

请求使用以下 URI 参数。

name

要为其创建新版本的意图的名称。该名称区分大小写。

长度限制：长度下限为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

必需：是

请求体

请求接受采用 JSON 格式的以下数据。

checksum

应该用于创建新版本的意图的 $LATEST 版本的校验和。如果您指定了校验和，而意图的 $LATEST 版本具有不同的校验和，则 Amazon Lex 会返回 PreconditionFailedException 异常，并且不会发布新版本。如果不指定校验和，Amazon Lex 会发布 $LATEST 版本。

类型：字符串

必需：否

响应语法

```
HTTP/1.1 201
Content-type: application/json

{
   "checksum": "string",
   "conclusionStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "confirmationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "createdDate": number,
```

```
    "description": "string",
    "dialogCodeHook": {
       "messageVersion": "string",
       "uri": "string"
    },
    "followUpPrompt": {
       "prompt": {
          "maxAttempts": number,
          "messages": [
             {
                "content": "string",
                "contentType": "string",
                "groupNumber": number
             }
          ],
          "responseCard": "string"
       },
       "rejectionStatement": {
          "messages": [
             {
                "content": "string",
                "contentType": "string",
                "groupNumber": number
             }
          ],
          "responseCard": "string"
       }
    },
    "fulfillmentActivity": {
       "codeHook": {
          "messageVersion": "string",
          "uri": "string"
       },
       "type": "string"
    },
    "inputContexts": [
       {
          "name": "string"
       }
    ],
    "kendraConfiguration": {
       "kendraIndex": "string",
       "queryFilterString": "string",
       "role": "string"
```

```
  },
  "lastUpdatedDate": number,
  "name": "string",
  "outputContexts": [
    {
      "name": "string",
      "timeToLiveInSeconds": number,
      "turnsToLive": number
    }
  ],
  "parentIntentSignature": "string",
  "rejectionStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string",
        "groupNumber": number
      }
    ],
    "responseCard": "string"
  },
  "sampleUtterances": [ "string" ],
  "slots": [
    {
      "defaultValueSpec": {
        "defaultValueList": [
          {
            "defaultValue": "string"
          }
        ]
      },
      "description": "string",
      "name": "string",
      "obfuscationSetting": "string",
      "priority": number,
      "responseCard": "string",
      "sampleUtterances": [ "string" ],
      "slotConstraint": "string",
      "slotType": "string",
      "slotTypeVersion": "string",
      "valueElicitationPrompt": {
        "maxAttempts": number,
        "messages": [
          {
```

```
                "content": "string",
                "contentType": "string",
                "groupNumber": number
            }
        ],
        "responseCard": "string"
    }
  }
 ],
 "version": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 201 响应。

服务以 JSON 格式返回的以下数据。

checksum

创建的意图版本的校验和。

类型：字符串

conclusionStatement

在 `fulfillmentActivity` 字段中指定的 Lambda 函数履行意图后，Amazon Lex 会将此语句传达给用户。

类型：Statement 对象

confirmationPrompt

如果已定义，则是 Amazon Lex 在履行用户意图之前用于确认其意图的提示。

类型：Prompt 对象

createdDate

意图的创建日期。

类型：时间戳

description

目的的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

dialogCodeHook

如果已定义，Amazon Lex 会为每个用户输入调用此 Lambda 函数。

类型：CodeHook 对象

followUpPrompt

如果已定义，Amazon Lex 将在履行意图后使用此提示来征求其他用户活动。

类型：FollowUpPrompt 对象

fulfillmentActivity

描述履行意图的方式。

类型：FulfillmentActivity 对象

inputContexts

`InputContext` 对象数组，列出了 Amazon Lex 在与用户的对话中选择意图时必须处于活动状态的上下文。

类型：InputContext 对象数组

数组成员：最少 0 个物品。最多 5 项。

kendraConfiguration

用于将 Amazon Kendra 索引与 `AMAZON.KendraSearchIntent` 意图关联的配置信息（如果有）。

类型：KendraConfiguration 对象

lastUpdatedDate

意图的更新日期。

类型：时间戳

name

意图的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

outputContexts

OutputContext 对象数组，列出了履行意图时意图激活的上下文。

类型：OutputContext 对象数组

数组成员：最少 0 个物品。最多 10 个物品。

parentIntentSignature

内置意图唯一标识符。

类型：字符串

rejectionStatement

当用户对 confirmationPrompt 中定义的问题回答“否”时，Amazon Lex 将使用此语句进行响应，以确认意图已被取消。

类型：Statement 对象

sampleUtterances

为意图配置的示例言语数组。

类型：字符串数组

数组成员：最少 0 项。最多 1500 项。

长度限制：长度下限为 1。最大长度为 200。

slots

插槽类型数组，用于定义履行意图所需的信息。

类型：Slot 对象数组

数组成员：最少 0 个物品。最多 100 个项目。

version

分配给新版本意图的版本号。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

PreconditionFailedException

您尝试更改的资源的校验和与请求中的校验和不匹配。检查资源的校验和并重试。

HTTP 状态代码：412

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

CreateSlotTypeVersion

服务: Amazon Lex Model Building Service

基于指定插槽类型的 $LATEST 版本创建插槽类型的新版本。如果自您创建上一个版本以来此资源的 $LATEST 版本没有更改，则 Amazon Lex 不会创建新版本。它会返回您创建的上一个版本。

> **Note**
>
> 您只能更新插槽类型的 $LATEST 版本。您无法更新通过 CreateSlotTypeVersion 操作创建的带编号的版本。

创建插槽类型的版本时，Amazon Lex 会将版本设置为 1。后续版本递增 1。有关更多信息，请参阅 版本控制。

此操作需要 lex:CreateSlotTypeVersion 操作的权限。

请求语法

```
POST /slottypes/name/versions HTTP/1.1
Content-type: application/json

{
   "checksum": "string"
}
```

URI 请求参数

请求使用以下 URI 参数。

name

要为其创建新版本的插槽类型的名称。该名称区分大小写。

长度限制：长度下限为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

必需：是

请求体

请求接受采用 JSON 格式的以下数据。

checksum

您要发布的插槽类型的 $LATEST 版本的校验和。如果您指定了校验和，并且插槽类型的 $LATEST 版本具有不同的校验和，则 Amazon Lex 会返回 PreconditionFailedException 异常且不会发布新版本。如果不指定校验和，Amazon Lex 会发布 $LATEST 版本。

类型：字符串

必需：否

响应语法

```
HTTP/1.1 201
Content-type: application/json

{
   "checksum": "string",
   "createdDate": number,
   "description": "string",
   "enumerationValues": [
      {
         "synonyms": [ "string" ],
         "value": "string"
      }
   ],
   "lastUpdatedDate": number,
   "name": "string",
   "parentSlotTypeSignature": "string",
   "slotTypeConfigurations": [
      {
         "regexConfiguration": {
            "pattern": "string"
         }
      }
   ],
   "valueSelectionStrategy": "string",
   "version": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 201 响应。

服务以 JSON 格式返回的以下数据。

checksum

插槽类型的 $LATEST 版本的校验和。

类型：字符串

createdDate

插槽类型的创建日期。

类型：时间戳

description

槽类型的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

enumerationValues

定义插槽类型可采用值的 EnumerationValue 对象的列表。

类型：EnumerationValue 对象数组

数组成员：最少 0 个物品。最多 10000 项。

lastUpdatedDate

插槽类型的更新日期。创建资源时，创建日期和上次更新日期相同。

类型：时间戳

name

槽类型的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

parentSlotTypeSignature

用作此插槽类型的父级的内置插槽类型。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：^((AMAZON\.)_?|[A-Za-z]_?)+

slotTypeConfigurations

扩展父级内置插槽类型的配置信息。

类型：SlotTypeConfiguration 对象数组

数组成员：最少 0 个物品。最多 10 个物品。

valueSelectionStrategy

Amazon Lex 用来确定插槽的值的策略。有关更多信息，请参阅 PutSlotType。

类型：字符串

有效值：ORIGINAL_VALUE | TOP_RESOLUTION

version

分配给新插槽类型版本的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

PreconditionFailedException

您尝试更改的资源的校验和与请求中的校验和不匹配。检查资源的校验和并重试。

HTTP 状态代码：412

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3

- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

DeleteBot

服务: Amazon Lex Model Building Service

删除机器人的所有版本，包括 $LATEST 版本。要删除机器人的特定版本，请使用 DeleteBotVersion 操作。DeleteBot 操作不会立即移除机器人架构。而是将它标记为删除并稍后删除。

Amazon Lex 无限期存储言语，以提高您的机器人响应用户输入的能力。删除机器人后，这些言语不会被删除。要删除言语，请使用 DeleteUtterances 操作。

如果机器人有别名，您将无法删除它。相反，DeleteBot 操作会返回一个 ResourceInUseException 异常，其中包含对引用机器人的别名的引用。要删除对机器人的引用，请删除别名。如果您再次遇到同样的异常，请删除引用别名，直到 DeleteBot 操作成功。

此操作需要 lex:DeleteBot 操作的权限。

请求语法

```
DELETE /bots/name HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

name

机器人的名称。该名称区分大小写。

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 204
```

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 204 响应。

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

ResourceInUseException

您尝试删除的资源被其他资源引用。使用此信息来删除对您正在尝试删除的资源的引用。

异常的正文包含描述资源的 JSON 对象。

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,

"resourceReference": {

"name": string, "version": string } }
```

HTTP 状态代码：400

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

DeleteBotAlias

服务: Amazon Lex Model Building Service

删除指定自动程序的别名。

您无法删除在机器人与消息收发通道之间的关联中使用的别名。如果在通道关联中使用别名，则 DeleteBot 操作会返回一个 ResourceInUseException 异常，其中包含对引用该机器人的通道关联的引用。您可以通过删除通道关联来删除对别名的引用。如果您再次遇到同样的异常，请删除引用关联，直到 DeleteBotAlias 操作成功。

请求语法

```
DELETE /bots/botName/aliases/name HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

botName

别名指向的自动程序的名称。

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

必需：是

name

要删除的别名的名称。该名称区分大小写。

长度限制：长度下限为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 204
```

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 204 响应。

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

ResourceInUseException

您尝试删除的资源被其他资源引用。使用此信息来删除对您正在尝试删除的资源的引用。

异常的正文包含描述资源的 JSON 对象。

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,
```

```
"resourceReference": {

"name": string, "version": string } }
```

HTTP 状态代码：400

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

DeleteBotChannelAssociation

服务: Amazon Lex Model Building Service

删除 Amazon Lex 机器人和消息收发平台之间的关联。

此操作需要 `lex:DeleteBotChannelAssociation` 操作权限。

请求语法

```
DELETE /bots/botName/aliases/aliasName/channels/name HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

aliasName

指向与之建立此关联的 Amazon Lex 机器人的特定版本的别名。

长度限制：长度下限为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

必需：是

botName

Amazon Lex 机器人的名称。

长度限制：最小长度为 2。最大长度为 50。

模式：`^([A-Za-z]_?)+$`

必需：是

name

关联的名称。该名称区分大小写。

长度限制：长度下限为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 204
```

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 204 响应。

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

DeleteBotVersion

服务: Amazon Lex Model Building Service

删除机器人的特定版本。要删除机器人的所有版本，请使用 DeleteBot 操作。

此操作需要 `lex:DeleteBotVersion` 操作的权限。

请求语法

```
DELETE /bots/name/versions/version HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

name

机器人的名称。

长度限制：最小长度为 2。最大长度为 50。

模式：`^([A-Za-z]_?)+$`

必需：是

version

待删除机器人的版本。机器人的 `$LATEST` 版本无法删除。要删除 `$LATEST` 版本，请使用 DeleteBot 操作。

长度限制：长度下限为 1。长度上限为 64。

模式：`[0-9]+`

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 204
```

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 204 响应。

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

ResourceInUseException

您尝试删除的资源被其他资源引用。使用此信息来删除对您正在尝试删除的资源的引用。

异常的正文包含描述资源的 JSON 对象。

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,

"resourceReference": {

"name": string, "version": string } }
```

HTTP 状态代码：400

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

DeleteIntent

服务: Amazon Lex Model Building Service

删除意图的所有版本，包括 $LATEST 版本。要删除意图的特定版本，请使用 DeleteIntentVersion 操作。

只有意图版本未被引用时，您才能删除该意图版本。要删除一个或多个机器人中引用的意图（请参阅Amazon Lex：工作原理），必须先删除这些引用。

> **Note**
>
> 如果您遇到 ResourceInUseException 异常，它会提供一个示例参考，显示意图的引用位置。要删除对意图的引用，请更新机器人或将其删除。如果您再次尝试删除意图时遇到同样的异常，请重复此操作，直到意图没有引用并且调用 DeleteIntent 成功。

此操作需要 lex:DeleteIntent 操作权限。

请求语法

```
DELETE /intents/name HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

name

意图的名称。该名称区分大小写。

长度限制：长度下限为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 204
```

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 204 响应。

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

ResourceInUseException

您尝试删除的资源被其他资源引用。使用此信息来删除对您正在尝试删除的资源的引用。

异常的正文包含描述资源的 JSON 对象。

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,
```

```
"resourceReference": {
"name": string, "version": string } }
```

HTTP 状态代码：400

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

DeleteIntentVersion

服务: Amazon Lex Model Building Service

删除目的的特定版本。要删除意图的所有版本，请使用 DeleteIntent 操作。

此操作需要 `lex:DeleteIntentVersion` 操作的权限。

请求语法

```
DELETE /intents/name/versions/version HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

name

意图的名称。

长度限制：长度下限为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

必需：是

version

要删除的意图的版本。意图的 $LATEST 版本无法删除。要删除 $LATEST 版本，请使用 DeleteIntent 操作。

长度限制：长度下限为 1。长度上限为 64。

模式：`[0-9]+`

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 204
```

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 204 响应。

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

ResourceInUseException

您尝试删除的资源被其他资源引用。使用此信息来删除对您正在尝试删除的资源的引用。

异常的正文包含描述资源的 JSON 对象。

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,

"resourceReference": {

"name": string, "version": string } }
```

HTTP 状态代码：400

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

DeleteSlotType

服务: Amazon Lex Model Building Service

删除插槽类型的所有版本，包括 $LATEST 版本。要删除插槽类型的特定版本，请使用 DeleteSlotTypeVersion 操作。

只有插槽类型版本未被引用时，您才能删除该插槽类型版本。要删除一个或多个意图中引用的插槽类型，必须先删除这些引用。

> **Note**
>
> 如果您遇到 ResourceInUseException 异常，则该异常会提供一个示例参考，其中显示了引用插槽类型的意图。要删除对插槽类型的引用，请更新意图或将其删除。如果您再次尝试删除插槽类型时遇到同样的异常，请重复此操作，直到该插槽类型没有引用并且调用 DeleteSlotType 成功。

此操作需要 lex:DeleteSlotType 操作权限。

请求语法

```
DELETE /slottypes/name HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

name

槽类型的名称。该名称区分大小写。

长度限制：长度下限为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 204
```

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 204 响应。

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

ResourceInUseException

您尝试删除的资源被其他资源引用。使用此信息来删除对您正在尝试删除的资源的引用。

异常的正文包含描述资源的 JSON 对象。

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,
```

```
"resourceReference": {
"name": string, "version": string } }
```

HTTP 状态代码：400

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

DeleteSlotTypeVersion

服务: Amazon Lex Model Building Service

删除槽类型的特定版本。要删除插槽类型的所有版本，请使用 DeleteSlotType 操作。

此操作需要 `lex:DeleteSlotTypeVersion` 操作的权限。

请求语法

```
DELETE /slottypes/name/version/version HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

name

槽类型的名称。

长度限制：长度下限为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

必需：是

version

要删除的插槽类型的版本。插槽类型的 `$LATEST` 版本无法删除。要删除 `$LATEST` 版本，请使用 DeleteSlotType 操作。

长度限制：长度下限为 1。长度上限为 64。

模式：`[0-9]+`

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 204
```

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 204 响应。

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

ResourceInUseException

您尝试删除的资源被其他资源引用。使用此信息来删除对您正在尝试删除的资源的引用。

异常的正文包含描述资源的 JSON 对象。

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,

"resourceReference": {

"name": string, "version": string } }
```

HTTP 状态代码：400

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

DeleteUtterances

服务: Amazon Lex Model Building Service

删除存储的言语。

Amazon Lex 存储用户发送给您的机器人的言语。言语会存储 15 天以供 GetUtterancesView 操作使用，然后无限期存储，用于提高机器人响应用户输入的能力。

使用 DeleteUtterances 操作来为特定用户手动删除存储的言语。当您使用 DeleteUtterances 操作时，为提高机器人响应用户输入的能力而存储的言语会立即被删除。存储用于 GetUtterancesView 操作的言语将在 15 天后删除。

此操作需要 lex:DeleteUtterances 操作的权限。

请求语法

```
DELETE /bots/botName/utterances/userId HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

botName

存储言语的机器人的名称。

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

必需：是

userId

发表言语的用户的唯一标识符。这是在包含话语的PostContent或PostText操作请求中发送的用户 ID。

长度限制：最小长度为 2。最大长度为 100。

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 204
```

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 204 响应。

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2

- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetBot

服务: Amazon Lex Model Building Service

返回特定机器人的元数据信息。您必须提供机器人名称和机器人版本或别名。

此操作需要 `lex:GetBot` 操作的权限。

请求语法

```
GET /bots/name/versions/versionoralias HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

name

机器人的名称。该名称区分大小写。

长度限制：最小长度为 2。最大长度为 50。

模式：`^([A-Za-z]_?)+$`

必需：是

versionoralias

机器人的版本或别名。

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "abortStatement": {
      "messages": [
         {
```

```
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "checksum": "string",
   "childDirected": boolean,
   "clarificationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "createdDate": number,
   "description": "string",
   "detectSentiment": boolean,
   "enableModelImprovements": boolean,
   "failureReason": "string",
   "idleSessionTTLInSeconds": number,
   "intents": [
      {
         "intentName": "string",
         "intentVersion": "string"
      }
   ],
   "lastUpdatedDate": number,
   "locale": "string",
   "name": "string",
   "nluIntentConfidenceThreshold": number,
   "status": "string",
   "version": "string",
   "voiceId": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

abortStatement

当用户选择在不完成对话的情况下结束对话时，Amazon Lex 返回的消息。有关更多信息，请参阅 PutBot。

类型：Statement 对象

checksum

用于识别机器人 $LATEST 版本的特定修订版的机器人的校验和。

类型：字符串

childDirected

对于使用 Amazon Lex 模型构建服务创建的每个 Amazon Lex 机器人，您都必须通过在 childDirected 字段中指定 true 或 false，指定您对 Amazon Lex 的使用是否与全部或部分针对 13 岁以下儿童且受《儿童在线隐私保护法》(COPPA) 约束的网站、程序或其他应用程序有关。在 childDirected 字段中指定 true，即表示您确认您对 Amazon Lex 的使用确实与全部或部分针对 13 岁以下儿童且受 COPPA 约束的网站、计划或其他应用程序有关。在 childDirected 字段中指定 false，即表示您确认您对 Amazon Lex 的使用不与全部或部分针对 13 岁以下儿童且受 COPPA 约束的网站、计划或其他应用程序有关。如果在 childDirected 字段中指定默认值不能正确反映您确认您对 Amazon Lex 的使用不与全部或部分针对 13 岁以下儿童且受 COPPA 约束的网站、计划或其他应用程序有关，则您可以不指定。

如果您对 Amazon Lex 的使用涉及全部或部分针对 13 岁以下儿童的网站、程序或其他应用程序，则必须获得 COPPA 规定的任何必需的可核实的家长同意。有关将 Amazon Lex 用于全部或部分针对 13 岁以下儿童的网站、程序或其他应用程序的信息，请参阅 Amazon Lex 常见问题解答。

类型：布尔值

clarificationPrompt

Amazon Lex 在无法理解用户的请求时使用的消息。有关更多信息，请参阅 PutBot。

类型：Prompt 对象

createdDate

机器人的创建日期。

类型：时间戳

description

机器人的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

detectSentiment

表示是否应将用户言语发送到 Amazon Comprehend 以进行情绪分析。

类型：布尔值

enableModelImprovements

表示机器人是否使用精度改进。`true` 表示机器人正在使用改进，否则为 `false`。

类型：布尔值

failureReason

如果 `status` 是 `FAILED`，Amazon Lex 会解释为什么它未能构建该机器人。

类型：字符串

idleSessionTTLInSeconds

Amazon Lex 保留对话中收集的数据的最长时间（秒）。有关更多信息，请参阅 PutBot。

类型：整数

有效范围：最小值为 60。最大值为 86400。

intents

`intent` 数据元数组。有关更多信息，请参阅 PutBot。

类型：Intent 对象数组

lastUpdatedDate

机器人的更新日期。创建资源时，创建日期和上次更新日期相同。

类型：时间戳

locale

机器人的目标区域设置。

类型：字符串

有效值：de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR

name

机器人的名称。

类型：字符串

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

nluIntentConfidenceThreshold

该分数决定 Amazon Lex 在或PostText响应中返回替代意图时在何处插入AMAZON.KendraSearchIntent、PostContent或两者。AMAZON.FallbackIntent AMAZON.FallbackIntent如果实际可信度分数低于此值，则会插入。AMAZON.KendraSearchIntent只有在为机器人配置时才会插入。

类型：双精度

有效范围：最小值为 0。最大值为 1。

status

机器人的状态。

当状态为 BUILDING 时，Amazon Lex 正在构建机器人以供测试和使用。

如果机器人的状态为 READY_BASIC_TESTING，则可以使用机器人意图中指定的确切言语来测试机器人。当机器人准备好进行全面测试或运行时，状态为 READY。

如果在构建机器人时出现问题，则状态为 FAILED，并且 failureReason 字段解释了为什么没有构建机器人。

如果机器人已保存但未构建，则状态为 NOT_BUILT。

类型：字符串

有效值：BUILDING | READY | READY_BASIC_TESTING | FAILED | NOT_BUILT

version

自动程序的版本。对于新机器人，版本始终是 $LATEST。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

voiceId

Amazon Lex 用于和用户进行语音交互的 Amazon Polly 语音 ID。有关更多信息，请参阅 PutBot。

类型：字符串

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetBotAlias

服务: Amazon Lex Model Building Service

返回有关 Amazon Lex 机器人别名的信息。有关别名的更多信息，请参阅版本控制和别名。

此操作需要 `lex:GetBotAlias` 操作的权限。

请求语法

```
GET /bots/botName/aliases/name HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

botName

机器人的名称。

长度限制：最小长度为 2。最大长度为 50。

模式：`^([A-Za-z]_?)+$`

必需：是

name

自动程序别名的名称。该名称区分大小写。

长度限制：长度下限为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json
```

```
{
   "botName": "string",
   "botVersion": "string",
   "checksum": "string",
   "conversationLogs": {
      "iamRoleArn": "string",
      "logSettings": [
         {
            "destination": "string",
            "kmsKeyArn": "string",
            "logType": "string",
            "resourceArn": "string",
            "resourcePrefix": "string"
         }
      ]
   },
   "createdDate": number,
   "description": "string",
   "lastUpdatedDate": number,
   "name": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

botName

别名指向的自动程序的名称。

类型：字符串

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

botVersion

别名指向的机器人的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

checksum

机器人别名的校验和。

类型：字符串

conversationLogs

确定 Amazon Lex 如何使用对话日志作为别名的设置。

类型：ConversationLogsResponse 对象

createdDate

机器人别名的创建日期。

类型：时间戳

description

机器人别名的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

lastUpdatedDate

机器人别名的更新日期。创建资源时，创建日期和上次更新日期相同。

类型：时间戳

name

自动程序别名的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetBotAliases

服务: Amazon Lex Model Building Service

返回指定 Amazon Lex 机器人的别名列表。

此操作需要 `lex:GetBotAliases` 操作的权限。

请求语法

```
GET /bots/botName/aliases/?
maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

botName

机器人的名称。

长度限制：最小长度为 2。最大长度为 50。

模式：`^([A-Za-z]_?)+$`

必需：是

maxResults

要在响应中返回的别名的最大数量。默认值为 50。

有效范围：最小值为 1。最大值为 50。

nameContains

机器人别名中要匹配的子字符串。如果别名名称的任何部分与子字符串匹配，则将返回别名。例如，“xyz”同时匹配“xyzabc”和“abcxyz”。

长度限制：长度下限为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

nextToken

用于获取下一页别名的分页令牌。如果对此调用的响应被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页的别名，请在下一个请求中指定分页令牌。

请求正文

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "BotAliases": [
      {
         "botName": "string",
         "botVersion": "string",
         "checksum": "string",
         "conversationLogs": {
            "iamRoleArn": "string",
            "logSettings": [
               {
                  "destination": "string",
                  "kmsKeyArn": "string",
                  "logType": "string",
                  "resourceArn": "string",
                  "resourcePrefix": "string"
               }
            ]
         },
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string"
      }
   ],
   "nextToken": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

BotAliases

BotAliasMetadata 对象数组，每个对象都描述一个机器人别名。

类型：BotAliasMetadata 对象数组

nextToken

用于获取下一页别名的分页令牌。如果对此调用的响应被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页的别名，请在下一个请求中指定分页令牌。

类型：字符串

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2

- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetBotChannelAssociation

服务: Amazon Lex Model Building Service

返回有关 Amazon Lex 机器人和消息收发平台之间的关联的信息。

此操作需要 `lex:GetBotChannelAssociation` 操作的权限。

请求语法

```
GET /bots/botName/aliases/aliasName/channels/name HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

aliasName

指向要与之建立此关联的 Amazon Lex 机器人的特定版本的别名。

长度限制：长度下限为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

必需：是

botName

Amazon Lex 机器人的名称。

长度限制：最小长度为 2。最大长度为 50。

模式：`^([A-Za-z]_?)+$`

必需：是

name

机器人与通道之间的关联的名称。该名称区分大小写。

长度限制：长度下限为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "botAlias": "string",
   "botConfiguration": {
      "string" : "string"
   },
   "botName": "string",
   "createdDate": number,
   "description": "string",
   "failureReason": "string",
   "name": "string",
   "status": "string",
   "type": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

botAlias

指向要与之建立此关联的 Amazon Lex 机器人的特定版本的别名。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

botConfiguration

提供消息收发平台与 Amazon Lex 通信所需的信息。

类型：字符串到字符串映射

映射条目：最多 10 项。

botName

Amazon Lex 机器人的名称。

类型：字符串

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

createdDate

机器人与通道之间建立关联的日期。

类型：时间戳

description

对机器人与通道之间关联的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

failureReason

如果 status 是 FAILED，则 Amazon Lex 会提供其未能创建关联的原因。

类型：字符串

name

机器人与通道之间的关联的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

status

机器人通道的状态。

- CREATED — 通道已创建，随时可用。
- IN_PROGRESS — 通道正在创建。

- FAILED — 创建通道时出错。有关失败原因的信息，请参阅 failureReason 字段。

类型：字符串

有效值：IN_PROGRESS | CREATED | FAILED

type

消息收发平台的类型。

类型：字符串

有效值：Facebook | Slack | Twilio-Sms | Kik

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面

- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetBotChannelAssociations

服务: Amazon Lex Model Building Service

返回与指定机器人关联的所有通道的列表。

GetBotChannelAssociations 操作需要 lex:GetBotChannelAssociations 操作的权限。

请求语法

```
GET /bots/botName/aliases/aliasName/channels/?
maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

aliasName

指向要与之建立此关联的 Amazon Lex 机器人的特定版本的别名。

长度限制：长度下限为 1。最大长度为 100。

模式：^(-|^([A-Za-z]_?)+$)$

必需：是

botName

关联中 Amazon Lex 机器人的名称。

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

必需：是

maxResults

要在响应中返回的关联的最大数量。默认值为 50。

有效范围：最小值为 1。最大值为 50。

nameContains

通道关联名称中要匹配的子字符串。如果关联名称的任何部分与子字符串匹配，则将返回关联。例如，“xyz”同时匹配“xyzabc”和“abcxyz”。要返回所有机器人通道关联，请使用连字符 ("-") 作为 nameContains 参数。

长度限制：长度下限为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

nextToken

用于获取下一页关联的分页令牌。如果对此调用的响应被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页关联，请在下一个请求中指定分页令牌。

请求正文

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "botChannelAssociations": [
      {
         "botAlias": "string",
         "botConfiguration": {
            "string" : "string"
         },
         "botName": "string",
         "createdDate": number,
         "description": "string",
         "failureReason": "string",
         "name": "string",
         "status": "string",
         "type": "string"
      }
   ],
   "nextToken": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

botChannelAssociations

对象数组，每个关联对应于一个对象，用于提供有关 Amazon Lex 机器人及其与通道关联的信息。

类型：BotChannelAssociation 对象数组

nextToken

用于获取下一页关联的分页令牌。如果对此调用的响应被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页关联，请在下一个请求中指定分页令牌。

类型：字符串

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面

- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetBots

服务: Amazon Lex Model Building Service

返回机器人信息，如下所示：

- 如果您提供 nameContains 字段，则响应将包含名称包含指定字符串的所有机器人的 $LATEST 版本信息。
- 如果您未指定 nameContains 字段，则该操作将返回有关所有机器人的 $LATEST 版本的信息。

此操作需要 lex:GetBots 操作权限。

请求语法

```
GET /bots/?maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

maxResults

要在请求返回的响应中返回的机器人的最大数量。默认值为 10。

有效范围：最小值为 1。最大值为 50。

nameContains

机器人名称中要匹配的子字符串。如果机器人名称的任何部分与子字符串匹配，则将返回机器人。例如，“xyz”同时匹配“xyzabc”和“abcxyz”。

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

nextToken

用于获取下一页机器人的分页令牌。如果对此调用的响应被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页机器人，请在下一个请求中指定分页令牌。

请求正文

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "bots": [
      {
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string",
         "status": "string",
         "version": "string"
      }
   ],
   "nextToken": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

bots

botMetadata 对象数组，每个机器人对应一个条目。

类型：BotMetadata 对象数组

nextToken

如果响应被截断，它会包含一个分页令牌，您可以在下次请求中指定该令牌来获取下一页机器人。

类型：字符串

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetBotVersions

服务: Amazon Lex Model Building Service

获取有关机器人的所有版本的信息。

GetBotVersions 操作会为每个版本的机器人返回一个 BotMetadata 对象。例如，如果机器人有三个带编号的版本，则 GetBotVersions 操作会在响应中返回四个 BotMetadata 对象，每个编号版本一个，$LATEST 版本一个。

GetBotVersions 操作始终返回至少一个版本，即 $LATEST 版本。

此操作需要 lex:GetBotVersions 操作的权限。

请求语法

```
GET /bots/name/versions/?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

maxResults

要在一个响应中返回的机器人版本的最大数量。默认值为 10。

有效范围：最小值为 1。最大值为 50。

name

应返回其版本的自动程序的名称。

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

必需：是

nextToken

用于获取下一页机器人版本的分页令牌。如果对此调用的响应被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页版本，请在下一个请求中指定分页令牌。

请求正文

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "bots": [
      {
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string",
         "status": "string",
         "version": "string"
      }
   ],
   "nextToken": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

bots

BotMetadata 对象数组，每个编号的机器人版本各一个，$LATEST 版本一个。

类型：BotMetadata 对象数组

nextToken

用于获取下一页机器人版本的分页令牌。如果对此调用的响应被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页版本，请在下一个请求中指定分页令牌。

类型：字符串

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetBuiltinIntent

服务: Amazon Lex Model Building Service

返回有关内置目的的信息。

此操作需要 `lex:GetBuiltinIntent` 操作权限。

请求语法

```
GET /builtins/intents/signature HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

signature

内置意图唯一标识符。要查找意图的签名，请参阅 Alexa Skills Kit 中的标准内置意图。

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "signature": "string",
   "slots": [
      {
         "name": "string"
      }
   ],
   "supportedLocales": [ "string" ]
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

signature

内置意图唯一标识符。

类型：字符串

slots

BuiltinIntentSlot 对象数组，意图中每种插槽类型对应一个条目。

类型：BuiltinIntentSlot 对象数组

supportedLocales

意图支持的区域设置列表。

类型：字符串数组

有效值：de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetBuiltinIntents

服务: Amazon Lex Model Building Service

获取符合指定条件的内置目的的列表。

此操作需要 `lex:GetBuiltinIntents` 操作权限。

请求语法

```
GET /builtins/intents/?
locale=locale&maxResults=maxResults&nextToken=nextToken&signatureContains=signatureContains
 HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

locale

意图支持的区域设置列表。

有效值：`de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

maxResults

要在响应中返回的意图的最大数量。默认值为 10。

有效范围：最小值为 1。最大值为 50。

nextToken

用于获取下一页意图的分页令牌。如果此 API 调用被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页意图，请在下一个请求中使用分页令牌。

signatureContains

内置意图签名中要匹配的子字符串。如果意图签名的任何部分与子字符串匹配，则将返回意图。例如，“xyz”同时匹配“xyzabc”和“abcxyz”。要查找意图的签名，请参阅 Alexa Skills Kit 中的标准内置意图。

请求正文

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "intents": [
      {
         "signature": "string",
         "supportedLocales": [ "string" ]
      }
   ],
   "nextToken": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

intents

builtinIntentMetadata 对象数组，响应中的每个意图对应一个条目。

类型：BuiltinIntentMetadata 对象数组

nextToken

用于获取下一页意图的分页令牌。如果对此 API 调用的响应被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页意图，请在下一个请求中指定分页令牌。

类型：字符串

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetBuiltinSlotTypes

服务: Amazon Lex Model Building Service

获取符合指定条件的内置槽类型的列表。

有关内置插槽类型的列表，请参阅 Alexa Skills Kit 中的插槽类型参考。

此操作需要 `lex:GetBuiltInSlotTypes` 操作权限。

请求语法

```
GET /builtins/slottypes/?
locale=locale&maxResults=maxResults&nextToken=nextToken&signatureContains=signatureContains
 HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

locale

插槽类型支持的区域设置列表。

有效值：`de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

maxResults

要在响应中返回的插槽类型的最大数量。默认值为 10。

有效范围：最小值为 1。最大值为 50。

nextToken

用于获取下一页插槽类型的分页令牌。如果对此 API 调用的响应被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页插槽类型，请在下一个请求中指定分页令牌。

signatureContains

内置插槽类型签名中要匹配的子字符串。如果插槽类型签名的任何部分与子字符串匹配，则将返回该插槽类型。例如，“xyz”同时匹配“xyzabc”和“abcxyz”。

请求正文

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "nextToken": "string",
   "slotTypes": [
      {
         "signature": "string",
         "supportedLocales": [ "string" ]
      }
   ]
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

nextToken

如果响应被截断，则响应中会包含一个分页令牌，您可以在下一个请求中使用该令牌来获取下一页插槽类型。

类型：字符串

slotTypes

BuiltInSlotTypeMetadata 对象数组，返回的每种插槽类型对应一个条目。

类型：BuiltinSlotTypeMetadata 对象数组

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetExport

服务: Amazon Lex Model Building Service

以指定格式导出 Amazon Lex 资源的内容。

请求语法

```
GET /exports/?exportType=exportType&name=name&resourceType=resourceType&version=version
 HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

exportType

导出的数据的格式。

有效值：ALEXA_SKILLS_KIT | LEX

必需：是

name

要导出的机器人的名称。

长度限制：长度下限为 1。最大长度为 100。

模式：[a-zA-Z_]+

必需：是

resourceType

要导出的资源的类型。

有效值：BOT | INTENT | SLOT_TYPE

必需：是

version

要导出的机器人的版本。

长度限制：长度下限为 1。长度上限为 64。

模式：[0-9]+

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "exportStatus": "string",
   "exportType": "string",
   "failureReason": "string",
   "name": "string",
   "resourceType": "string",
   "url": "string",
   "version": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

exportStatus

导出的状态。

- IN_PROGRESS — 正在导出。
- READY — 导出已完成。
- FAILED — 无法完成导出。

类型：字符串

有效值：IN_PROGRESS | READY | FAILED

exportType

导出的数据的格式。

类型：字符串

有效值：ALEXA_SKILLS_KIT | LEX

failureReason

如果 status 是 FAILED，则 Amazon Lex 会提供其未能导出资源的原因。

类型：字符串

name

正在导出的机器人的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：[a-zA-Z_]+

resourceType

已导出的资源的类型。

类型：字符串

有效值：BOT | INTENT | SLOT_TYPE

url

提供已导出的资源位置的 S3 预签名 URL。已导出的资源是包含 JSON 格式的已导出资源的 ZIP 存档。存档的结构可能会发生变化。您的代码不应依赖于存档结构。

类型：字符串

version

正在导出的机器人的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：[0-9]+

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包

- AWS 适用于 Ruby V3 的 SDK

GetImport

服务: Amazon Lex Model Building Service

获取有关使用 `StartImport` 操作启动的导入作业的信息。

请求语法

```
GET /imports/importId HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

importId

要返回的导入作业信息的标识符。

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "createdDate": number,
   "failureReason": [ "string" ],
   "importId": "string",
   "importStatus": "string",
   "mergeStrategy": "string",
   "name": "string",
   "resourceType": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

createdDate

创建导入作业的日期和时间的时间戳。

类型：时间戳

failureReason

描述导入作业未能完成的原因的字符串。

类型：字符串数组

importId

特定导入作业的标识符。

类型：字符串

importStatus

导入作业的状态。如果状态为 FAILED，则可以从 failureReason 字段获取失败的原因。

类型：字符串

有效值：IN_PROGRESS | COMPLETE | FAILED

mergeStrategy

现有资源与导入文件中的资源发生冲突时采取的操作。

类型：字符串

有效值：OVERWRITE_LATEST | FAIL_ON_CONFLICT

name

提供给导入作业的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：[a-zA-Z_]+

resourceType

已导入的资源的类型。

类型：字符串

有效值：BOT | INTENT | SLOT_TYPE

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK

- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetIntent

服务: Amazon Lex Model Building Service

返回有关意图的信息。除了意图名称外，您还必须指定意图版本。

此操作需要执行 `lex:GetIntent` 操作的权限。

请求语法

```
GET /intents/name/versions/version HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

name

意图的名称。该名称区分大小写。

长度限制：长度下限为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

必需：是

version

意图的版本。

长度限制：长度下限为 1。长度上限为 64。

模式：`\$LATEST|[0-9]+`

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
```

```
    "checksum": "string",
    "conclusionStatement": {
       "messages": [
          {
             "content": "string",
             "contentType": "string",
             "groupNumber": number
          }
       ],
       "responseCard": "string"
    },
    "confirmationPrompt": {
       "maxAttempts": number,
       "messages": [
          {
             "content": "string",
             "contentType": "string",
             "groupNumber": number
          }
       ],
       "responseCard": "string"
    },
    "createdDate": number,
    "description": "string",
    "dialogCodeHook": {
       "messageVersion": "string",
       "uri": "string"
    },
    "followUpPrompt": {
       "prompt": {
          "maxAttempts": number,
          "messages": [
             {
                "content": "string",
                "contentType": "string",
                "groupNumber": number
             }
          ],
          "responseCard": "string"
       },
       "rejectionStatement": {
          "messages": [
             {
                "content": "string",
```

```
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   }
},
"fulfillmentActivity": {
   "codeHook": {
      "messageVersion": "string",
      "uri": "string"
   },
   "type": "string"
},
"inputContexts": [
   {
      "name": "string"
   }
],
"kendraConfiguration": {
   "kendraIndex": "string",
   "queryFilterString": "string",
   "role": "string"
},
"lastUpdatedDate": number,
"name": "string",
"outputContexts": [
   {
      "name": "string",
      "timeToLiveInSeconds": number,
      "turnsToLive": number
   }
],
"parentIntentSignature": "string",
"rejectionStatement": {
   "messages": [
      {
         "content": "string",
         "contentType": "string",
         "groupNumber": number
      }
   ],
   "responseCard": "string"
},
```

```
    "sampleUtterances": [ "string" ],
    "slots": [
       {
          "defaultValueSpec": {
             "defaultValueList": [
                {
                   "defaultValue": "string"
                }
             ]
          },
          "description": "string",
          "name": "string",
          "obfuscationSetting": "string",
          "priority": number,
          "responseCard": "string",
          "sampleUtterances": [ "string" ],
          "slotConstraint": "string",
          "slotType": "string",
          "slotTypeVersion": "string",
          "valueElicitationPrompt": {
             "maxAttempts": number,
             "messages": [
                {
                   "content": "string",
                   "contentType": "string",
                   "groupNumber": number
                }
             ],
             "responseCard": "string"
          }
       }
    ],
    "version": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

checksum

意图的校验和。

类型：字符串

conclusionStatement

在 fulfillmentActivity 元素中指定的 Lambda 函数履行意图后，Amazon Lex 会将此语句传达给用户。

类型：Statement 对象

confirmationPrompt

如果在机器人中已定义，则 Amazon Lex 会在履行用户请求之前使用提示确认意图。有关更多信息，请参阅 PutIntent。

类型：Prompt 对象

createdDate

意图的创建日期。

类型：时间戳

description

目的的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

dialogCodeHook

如果已在机器人中定则 义，则 Amazon Lex 会为每个用户输入调用此 Lambda 函数。有关更多信息，请参阅 PutIntent。

类型：CodeHook 对象

followUpPrompt

如果已在机器人中定义，则 Amazon Lex 会在履行意图后使用此提示来征求其他用户活动。有关更多信息，请参阅 PutIntent。

类型：FollowUpPrompt 对象

fulfillmentActivity

描述履行意图的方式。有关更多信息，请参阅 PutIntent。

类型：FulfillmentActivity 对象

inputContexts

InputContext 对象数组，列出了 Amazon Lex 在与用户的对话中选择意图时必须处于活动状态的上下文。

类型：InputContext 对象数组

数组成员：最少 0 个物品。最多 5 项。

kendraConfiguration

用于将 Amazon Kendra 索引与 AMAZON.KendraSearchIntent 意图关联的配置信息（如果有）。

类型：KendraConfiguration 对象

lastUpdatedDate

意图的更新日期。创建资源时，创建日期和上次更新日期相同。

类型：时间戳

name

意图的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

outputContexts

OutputContext 对象数组，列出了履行意图时意图激活的上下文。

类型：OutputContext 对象数组

数组成员：最少 0 个物品。最多 10 个物品。

parentIntentSignature

内置意图唯一标识符。

类型：字符串

rejectionStatement

当用户对 confirmationPrompt 中定义的问题回答“否”时，Amazon Lex 将使用此语句进行响应，以确认意图已被取消。

类型：Statement 对象

sampleUtterances

为意图配置的示例言语数组。

类型：字符串数组

数组成员：最少 0 项。最多 1500 项。

长度限制：长度下限为 1。最大长度为 200。

slots

为意图配置的意图插槽数组。

类型：Slot 对象数组

数组成员：最少 0 个物品。最多 100 个项目。

version

意图的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetIntents

服务: Amazon Lex Model Building Service

返回意图信息，如下所示：

- 如果指定 nameContains 字段，则返回包含指定字符串的所有意图的 $LATEST 版本。
- 如果未指定 nameContains 字段，则返回有关所有意图的 $LATEST 版本的信息。

该操作需要 lex:GetIntents 操作的权限。

请求语法

```
GET /intents/?maxResults=maxResults&nameContains=nameContains&nextToken=nextToken
 HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

maxResults

要在响应中返回的意图的最大数量。默认值为 10。

有效范围：最小值为 1。最大值为 50。

nameContains

意图名称中要匹配的子字符串。如果意图名称的任何部分与子字符串匹配，则将返回意图。例如，“xyz”同时匹配“xyzabc”和“abcxyz”。

长度限制：长度下限为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

nextToken

用于获取下一页意图的分页令牌。如果对此 API 调用的响应被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页意图，请在下一个请求中指定分页令牌。

请求正文

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "intents": [
      {
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string",
         "version": "string"
      }
   ],
   "nextToken": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

intents

Intent 数据元数组。有关更多信息，请参阅 PutBot。

类型：IntentMetadata 对象数组

nextToken

如果响应被截断，则响应会包含一个分页令牌，您可以在下次请求中指定该令牌来获取下一页意图。

类型：字符串

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetIntentVersions

服务: Amazon Lex Model Building Service

获取有关意图的所有版本的信息。

GetIntentVersions 操作会为每个版本的意图返回一个 IntentMetadata 对象。例如，如果意图有三个带编号的版本，则 GetIntentVersions 操作会在响应中返回四个 IntentMetadata 对象，每个编号版本一个，$LATEST 版本一个。

GetIntentVersions 操作始终返回至少一个版本，即 $LATEST 版本。

此操作需要 lex:GetIntentVersions 操作的权限。

请求语法

```
GET /intents/name/versions/?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

maxResults

要在一个响应中返回的意图版本的最大数量。默认值为 10。

有效范围：最小值为 1。最大值为 50。

name

应返回其版本的意图的名称。

长度限制：长度下限为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

必需：是

nextToken

用于获取下一页意图版本的分页令牌。如果对此调用的响应被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页版本，请在下一个请求中指定分页令牌。

请求正文

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "intents": [
      {
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string",
         "version": "string"
      }
   ],
   "nextToken": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

intents

IntentMetadata 对象数组，每个编号的意图版本各一个，$LATEST 版本一个。

类型：IntentMetadata 对象数组

nextToken

用于获取下一页意图版本的分页令牌。如果对此调用的响应被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页版本，请在下一个请求中指定分页令牌。

类型：字符串

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetMigration

服务: Amazon Lex Model Building Service

提供有关从 Amazon Lex V1 机器人到 Amazon Lex V2 机器人的持续或完整迁移的详细信息。使用此操作可查看与迁移相关的迁移警报和警告。

请求语法

```
GET /migrations/migrationId HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

migrationId

要查看的迁移的唯一标识符。由 StartMigration 操作返回 migrationID。

长度限制：固定长度为 10。

模式：^[0-9a-zA-Z]+$

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "alerts": [
      {
         "details": [ "string" ],
         "message": "string",
         "referenceURLs": [ "string" ],
         "type": "string"
      }
   ],
```

```
    "migrationId": "string",
    "migrationStatus": "string",
    "migrationStrategy": "string",
    "migrationTimestamp": number,
    "v1BotLocale": "string",
    "v1BotName": "string",
    "v1BotVersion": "string",
    "v2BotId": "string",
    "v2BotRole": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

alerts

警报和警告列表，这些警报和警告表明 Amazon Lex V1 机器人迁移到 Amazon Lex V2 时存在问题。当 Amazon Lex V1 功能在 Amazon Lex V2 中有不同的实现方式时，您会收到警告。

有关更多信息，请参阅《Amazon Lex V2 开发人员指南》中的迁移机器人。

类型：MigrationAlert 对象数组

migrationId

迁移的唯一标识符。这与调用 GetMigration 操作时使用的标识符相同。

类型：字符串

长度限制：固定长度为 10。

模式：^[0-9a-zA-Z]+$

migrationStatus

表示迁移的状态。当迁移完成并且机器人在 Amazon Lex V2 中可用时，状态为 COMPLETE。可能需要解决警报和警告才能完成迁移。

类型：字符串

有效值：IN_PROGRESS | COMPLETED | FAILED

migrationStrategy

用于进行迁移的策略。

- CREATE_NEW — 创建新的 Amazon Lex V2 机器人并将 Amazon Lex V1 机器人迁移到新机器人。
- UPDATE_EXISTING — 覆盖现有的 Amazon Lex V2 机器人元数据和正在迁移的区域设置。它不会更改 Amazon Lex V2 机器人中的任何其他区域设置。如果该区域设置不存在，则会在 Amazon Lex V2 机器人中创建一个新的区域设置。

类型：字符串

有效值：CREATE_NEW | UPDATE_EXISTING

migrationTimestamp

启动迁移的日期和时间。

类型：时间戳

v1BotLocale

已迁移到 Amazon Lex V2 的 Amazon Lex V1 机器人的区域设置。

类型：字符串

有效值：de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR

v1BotName

已迁移到 Amazon Lex V2 的 Amazon Lex V1 机器人的名称。

类型：字符串

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

v1BotVersion

已迁移到 Amazon Lex V2 的 Amazon Lex V1 机器人的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

v2BotId

Amazon Lex V1 要迁移到的 Amazon Lex V2 机器人的唯一标识符。

类型：字符串

长度限制：固定长度为 10。

模式：^[0-9a-zA-Z]+$

v2BotRole

Amazon Lex 用来运行 Amazon Lex V2 机器人的 IAM 角色。

类型：字符串

长度约束：最小长度为 20。最大长度为 2048。

模式：^arn:[\w\-]+:iam::[\d]{12}:role/.+$

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetMigrations

服务: Amazon Lex Model Building Service

获取 Amazon Lex V1 和 Amazon Lex V2 之间的迁移列表。

请求语法

```
GET /migrations?
maxResults=maxResults&migrationStatusEquals=migrationStatusEquals&nextToken=nextToken&sortByAtt
 HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

maxResults

要在响应中返回的迁移的最大数量。默认值为 10。

有效范围：最小值为 1。最大值为 50。

migrationStatusEquals

筛选列表以仅包含处于指定状态的迁移。

有效值：`IN_PROGRESS | COMPLETED | FAILED`

nextToken

用于获取下一页迁移的分页令牌。如果对此操作的响应被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页迁移，请在下一个请求中指定分页令牌。

sortByAttribute

用于对迁移列表进行排序的字段。您可以按照 Amazon Lex V1 机器人名称或开始迁移的日期和时间进行排序。

有效值：`V1_BOT_NAME | MIGRATION_DATE_TIME`

sortByOrder

对列表进行排序的顺序。

有效值：`ASCENDING | DESCENDING`

v1BotNameContains

筛选列表以仅包含名称包含指定字符串的机器人。该字符串与机器人名称中的任意位置匹配。

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

请求正文

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "migrationSummaries": [
      {
         "migrationId": "string",
         "migrationStatus": "string",
         "migrationStrategy": "string",
         "migrationTimestamp": number,
         "v1BotLocale": "string",
         "v1BotName": "string",
         "v1BotVersion": "string",
         "v2BotId": "string",
         "v2BotRole": "string"
      }
   ],
   "nextToken": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

migrationSummaries

从 Amazon Lex V1 迁移到 Amazon Lex V2 的一系列摘要。要查看迁移的详细信息，请使用对 GetMigration 操作的调用的摘要中的 `migrationId`。

类型：MigrationSummary 对象数组

nextToken

如果响应被截断，它会包含一个分页令牌，您可以在下一个请求中指定该令牌来获取下一页迁移。

类型：字符串

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2

- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetSlotType

服务: Amazon Lex Model Building Service

返回有关槽类型的特定版本的信息。除了指定插槽类型名称外，您还必须指定插槽类型版本。

此操作需要 `lex:GetSlotType` 操作的权限。

请求语法

```
GET /slottypes/name/versions/version HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

name

槽类型的名称。该名称区分大小写。

长度限制：长度下限为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

必需：是

version

插槽类型的版本。

长度限制：长度下限为 1。长度上限为 64。

模式：`\$LATEST|[0-9]+`

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json
```

```
{
   "checksum": "string",
   "createdDate": number,
   "description": "string",
   "enumerationValues": [
      {
         "synonyms": [ "string" ],
         "value": "string"
      }
   ],
   "lastUpdatedDate": number,
   "name": "string",
   "parentSlotTypeSignature": "string",
   "slotTypeConfigurations": [
      {
         "regexConfiguration": {
            "pattern": "string"
         }
      }
   ],
   "valueSelectionStrategy": "string",
   "version": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

checksum

插槽类型的 $LATEST 版本的校验和。

类型：字符串

createdDate

插槽类型的创建日期。

类型：时间戳

description

槽类型的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

enumerationValues

定义插槽类型可采用值的 EnumerationValue 对象的列表。

类型：EnumerationValue 对象数组

数组成员：最少 0 个物品。最多 10000 项。

lastUpdatedDate

插槽类型的更新日期。创建资源时，创建日期和上次更新日期相同。

类型：时间戳

name

槽类型的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

parentSlotTypeSignature

用作此插槽类型的父级的内置插槽类型。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：^((AMAZON\.)_?|[A-Za-z]_?)+

slotTypeConfigurations

扩展父级内置插槽类型的配置信息。

类型：SlotTypeConfiguration 对象数组

数组成员：最少 0 个物品。最多 10 个物品。

valueSelectionStrategy

Amazon Lex 用来确定插槽的值的策略。有关更多信息，请参阅 PutSlotType。

类型：字符串

有效值：ORIGINAL_VALUE | TOP_RESOLUTION

version

插槽类型的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面

- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetSlotTypes

服务: Amazon Lex Model Building Service

返回插槽类型信息，如下所示：

- 如果指定 nameContains 字段，则返回包含指定字符串的所有插槽类型的 $LATEST 版本。
- 如果未指定 nameContains 字段，则返回有关所有插槽类型的 $LATEST 版本的信息。

该操作需要 lex:GetSlotTypes 操作的权限。

请求语法

```
GET /slottypes/?maxResults=maxResults&nameContains=nameContains&nextToken=nextToken
 HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

maxResults

要在响应中返回的插槽类型的最大数量。默认值为 10。

有效范围：最小值为 1。最大值为 50。

nameContains

要在插槽类型名称中匹配的子字符串。如果插槽类型名称的任何部分与子字符串匹配，则将返回该插槽类型。例如，“xyz”同时匹配“xyzabc”和“abcxyz”。

长度限制：长度下限为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

nextToken

用于获取下一页插槽类型的分页令牌。如果对此 API 调用的响应被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页插槽类型，请在下一个请求中指定分页令牌。

请求正文

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "nextToken": "string",
   "slotTypes": [
      {
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string",
         "version": "string"
      }
   ]
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

nextToken

如果响应被截断，它会包含一个分页令牌，您可以在下次请求中指定该令牌来获取下一页插槽类型。

类型：字符串

slotTypes

对象数组，每种插槽类型对应一个对象，它提供诸如插槽类型的名称、版本和描述之类的信息。

类型：SlotTypeMetadata 对象数组

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetSlotTypeVersions

服务: Amazon Lex Model Building Service

获取有关插槽类型的所有版本的信息。

GetSlotTypeVersions 操作会为每个版本的插槽类型返回一个 SlotTypeMetadata 对象。例如，如果插槽类型有三个带编号的版本，则 GetSlotTypeVersions 操作会在响应中返回四个 SlotTypeMetadata 对象，每个编号版本一个，$LATEST 版本一个。

GetSlotTypeVersions 操作始终返回至少一个版本，即 $LATEST 版本。

此操作需要 lex:GetSlotTypeVersions 操作的权限。

请求语法

```
GET /slottypes/name/versions/?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

maxResults

要在响应中返回的插槽类型版本的最大数量。默认值为 10。

有效范围：最小值为 1。最大值为 50。

name

应返回其版本的插槽类型的名称。

长度限制：长度下限为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

必需：是

nextToken

用于获取下一页插槽类型版本的分页令牌。如果对此调用的响应被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页版本，请在下一个请求中指定分页令牌。

请求正文

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "nextToken": "string",
   "slotTypes": [
      {
         "createdDate": number,
         "description": "string",
         "lastUpdatedDate": number,
         "name": "string",
         "version": "string"
      }
   ]
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

nextToken

用于获取下一页插槽类型版本的分页令牌。如果对此调用的响应被截断，Amazon Lex 将在响应中返回分页令牌。要获取下一页版本，请在下一个请求中指定分页令牌。

类型：字符串

slotTypes

SlotTypeMetadata 对象数组，每个编号的插槽类型版本各一个，$LATEST 版本一个。

类型：SlotTypeMetadata 对象数组

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetUtterancesView

服务: Amazon Lex Model Building Service

使用 `GetUtterancesView` 操作来获取有关您的用户对您的机器人所说言语的信息。您可以使用这个列表来调整您的机器人响应的言语。

例如，假设您创建了一个机器人来订花。在您的用户使用您的机器人一段时间后，使用 `GetUtterancesView` 操作查看他们发出的请求以及请求是否成功。您可能会发现“我想要花”这句话没有被识别。您可以将这个言语添加到 `OrderFlowers` 意图中，这样您的机器人就能识别出这个言语。

发布新版本的机器人后，您可以获取有关旧版本和新版本的信息，以便比较两个版本的性能。

表达统计数据每天生成一次。可查询过去 15 天的数据。在每次请求中，您最多可以请求 5 个版本的机器人的信息。Amazon Lex 会返回该机器人在过去 15 天内收到的最频繁的言语。响应包含有关每个版本最多 100 句言语的信息。

在以下条件下，系统不会生成言语统计数据：

- 创建机器人时，`childDirected` 字段设置为 True。
- 您正在对一个或多个插槽使用插槽混淆处理。
- 您已选择退出 Amazon Lex 改进计划。

此操作需要 `lex:GetUtterancesView` 操作的权限。

请求语法

```
GET /bots/botname/utterances?
view=aggregation&bot_versions=botVersions&status_type=statusType HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

botname

应返回其言语信息的机器人的名称。

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

必需：是

botVersions

应返回其言语信息的机器人版本的数组。每个请求的限制为 5 个版本。

数组成员：最少 1 个物品。最多 5 项。

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

必需：是

statusType

要返回已识别且已处理的言语，请使用 Detected。要返回未识别的言语，请使用 Missed。

有效值：Detected | Missed

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "botName": "string",
   "utterances": [
      {
         "botVersion": "string",
         "utterances": [
            {
               "count": number,
               "distinctUsers": number,
               "firstUtteredDate": number,
               "lastUtteredDate": number,
```

```
                "utteranceString": "string"
            }
        ]
    }
  ]
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

botName

要返回其言语信息的机器人的名称。

类型：字符串

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

utterances

UtteranceList 对象数组，每个对象都包含描述机器人已处理的言语的 UtteranceData 对象列表。每个版本的响应最多包含 100 个 UtteranceData 对象。Amazon Lex 会返回该机器人在过去 15 天内收到的最频繁的言语。

类型：UtteranceList 对象数组

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

ListTagsForResource

服务: Amazon Lex Model Building Service

获取与指定资源关联的标签。只有机器人、机器人别名和机器人通道可以关联标签。

请求语法

```
GET /tags/resourceArn HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

resourceArn

要获得其标签列表的资源的 Amazon 资源名称 (ARN)。

长度限制：长度下限为 1。最大长度为 1011。

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ]
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

tags

与资源关联的标签。

类型：Tag 对象数组

数组成员：最少 0 个物品。最多 200 项。

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK

- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

PutBot

服务: Amazon Lex Model Building Service

创建 Amazon Lex 对话机器人或替换现有机器人。创建或更新机器人时，您只需要指定名称、区域以及机器人是否针对未满 13 岁的儿童即可。您可以使用它在稍后添加意图，或者从现有机器人中移除意图。当您使用最少的信息创建机器人时，会创建或更新该机器人，但是 Amazon Lex 会返回响应 FAILED。您可以在添加一个或多个意图后构建机器人。有关 Amazon Lex 的更多信息，请参阅Amazon Lex：工作原理。

如果您指定现有机器人的名称，则请求中的字段将替换机器人 $LATEST 版本中的现有值。Amazon Lex 会删除您未在请求中提供值的所有字段，但 idleTTLInSeconds 和 privacySettings 字段除外，会将它们设置为默认值。如果您没有为必填字段指定值，Amazon Lex 会引发异常。

此操作需要 lex:PutBot 操作的权限。有关更多信息，请参阅 适用于 Amazon Lex 的 Identity and Access Management。

请求语法

```
PUT /bots/name/versions/$LATEST HTTP/1.1
Content-type: application/json

{
   "abortStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "checksum": "string",
   "childDirected": boolean,
   "clarificationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
```

```
      ],
      "responseCard": "string"
   },
   "createVersion": boolean,
   "description": "string",
   "detectSentiment": boolean,
   "enableModelImprovements": boolean,
   "idleSessionTTLInSeconds": number,
   "intents": [
      {
         "intentName": "string",
         "intentVersion": "string"
      }
   ],
   "locale": "string",
   "nluIntentConfidenceThreshold": number,
   "processBehavior": "string",
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ],
   "voiceId": "string"
}
```

URI 请求参数

请求使用以下 URI 参数。

name

机器人的名称。名称不区分大小写。

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

必需：是

请求体

请求接受采用 JSON 格式的以下数据。

abortStatement

当 Amazon Lex 无法理解用户在上下文中的输入时，它会尝试几次引发信息。之后，Amazon Lex 将在 abortStatement 中定义的消息发送给用户，然后取消对话。要设置重试次数，请使用插槽类型的 valueElicitationPrompt 字段。

例如，在订购披萨机器人中，Amazon Lex 可能会问用户“您想要什么类型的饼皮？” 如果用户的响应不是预期的响应之一（例如，“薄皮”、“深盘”等），Amazon Lex 会尝试几次引发正确的响应。

例如，在订购披萨应用程序中，OrderPizza 可能是意图之一。此意图可能需要 CrustType 插槽。您可以在创建 CrustType 插槽时指定 valueElicitationPrompt 字段。

如果您定义了回退意图，则不会向用户发送取消语句，而是使用回退意图。有关更多信息，请参阅亚马逊。 FallbackIntent。

类型：Statement 对象

必需：否

checksum

标识 $LATEST 版本的特定修订版。

创建新机器人时，请将 checksum 字段留空。如果指定校验和，则会出现 BadRequestException 异常。

当您想更新机器人时，请将 checksum 字段设置为 $LATEST 版本最新修订版的校验和。如果您未指定 checksum 字段，或者校验和与 $LATEST 版本不匹配，则会出现 PreconditionFailedException 异常。

类型：字符串

必需：否

childDirected

对于使用 Amazon Lex 模型构建服务创建的每个 Amazon Lex 机器人，您都必须通过在 childDirected 字段中指定 true 或 false，指定您对 Amazon Lex 的使用是否与全部或部分针对 13 岁以下儿童且受《儿童在线隐私保护法》(COPPA) 约束的网站、程序或其他应用程序有关。在 childDirected 字段中指定 true，即表示您确认您对 Amazon Lex 的使用确实与全部或部分针对 13 岁以下儿童且受 COPPA 约束的网站、计划或其他应用程序有关。在 childDirected 字段中指定 false，即表示您确认您对 Amazon Lex 的使用不与全部或部分针对 13 岁以下儿童且受

COPPA 约束的网站、计划或其他应用程序有关。如果在 childDirected 字段中指定默认值不能正确反映您确认您对 Amazon Lex 的使用不与全部或部分针对 13 岁以下儿童且受 COPPA 约束的网站、计划或其他应用程序有关，则您可以不指定。

如果您对 Amazon Lex 的使用涉及全部或部分针对 13 岁以下儿童的网站、程序或其他应用程序，则必须获得 COPPA 规定的任何必需的可核实的家长同意。有关将 Amazon Lex 用于全部或部分针对 13 岁以下儿童的网站、程序或其他应用程序的信息，请参阅 Amazon Lex 常见问题解答。

类型：布尔值

必需：是

clarificationPrompt

当 Amazon Lex 不了解用户的意图时，它会使用此消息进行澄清。要指定 Amazon Lex 应重复多少次澄清提示，请使用 maxAttempts 字段。如果 Amazon Lex 仍然无法理解，它会发送 abortStatement 字段中的消息。

当您创建澄清提示时，请确保它建议用户做出正确的响应。例如，对于订购披萨和饮品的机器人，您可以创建这样的澄清提示：“您想做什么？ 您可以说“订一杯饮品”或“订一个披萨”。

如果您定义了回退意图，则如果重复澄清提示的次数达到 maxAttempts 字段中定义的次数，则会调用该意图。有关更多信息，请参阅亚马逊。 FallbackIntent。

如果您未定义澄清提示，则在运行时，Amazon Lex 将在三种情况下返回“400 错误请求”异常：

- 后续提示 — 当用户响应后续提示但不提供意图时。例如，在响应“您今天还想要其他东西吗？”后续提示时，用户回答“是”。由于 Amazon Lex 没有发送给用户以获取用户意图的澄清提示，因此它会返回“400 错误请求”异常。
- Lambda 函数 — 使用 Lambda 函数时，您将返回 ElicitIntent 对话类型。由于 Amazon Lex 没有用于获取用户意图的澄清提示，因此它会返回“400 错误请求”异常。
- PutSession 操作-使用PutSession操作时，您可以发送ElicitIntent对话类型。由于 Amazon Lex 没有用于获取用户意图的澄清提示，因此它会返回“400 错误请求”异常。

类型：Prompt 对象

必需：否

createVersion

当设置为 true 时，将创建机器人的新编号版本。这与调用 CreateBotVersion 操作相同。如果不指定 createVersion，则默认值为 false。

类型：布尔值

必需：否

description

机器人的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

必需：否

detectSentiment

当设置为 `true` 时，会将用户言语发送到 Amazon Comprehend 进行情绪分析。如果不指定 `detectSentiment`，则默认值为 `false`。

类型：布尔值

必需：否

enableModelImprovements

设置为 `true` 以允许访问自然语言理解方面的改进。

当将 `enableModelImprovements` 参数设置为 `true` 时，可以使用 `nluIntentConfidenceThreshold` 参数配置置信度分数。有关更多信息，请参阅置信度分数。

您只能在某些区域中设置 `enableModelImprovements` 参数。如果将该参数设置为 `true`，则您的机器人可以获得准确性改进。

您可以针对 en-US 区域设置将 `enableModelImprovements` 参数设置为 `false` 的区域有：

- 美国东部（弗吉尼亚州北部）(us-east-1)
- 美国西部（俄勒冈州）(us-west-2)
- 亚太地区（悉尼）(ap-southeast-2)
- 欧洲（爱尔兰）(eu-west-1)

在其他区域和区域设置中，`enableModelImprovements` 参数默认设置为 `true`。在这些区域和区域设置中，将参数设置为 `false` 会引发 `ValidationException` 异常。

类型：布尔值

必需：否

idleSessionTTLInSeconds

Amazon Lex 保留对话中收集的数据的最长时间（秒）。

用户交互会话在指定的时间内保持活动状态。如果在此期间未发生任何对话，则会话将过期并且 Amazon Lex 会删除在超时之前提供的所有数据。

例如，假设用户选择了 OrderPizza 意图，但在下单的中途被偏离了方向。如果用户未在指定时间内完成订单，Amazon Lex 会丢弃其收集的插槽信息，用户必须重新开始。

如果您未在 PutBot 操作请求中包含 idleSessionTTLInSeconds 元素，Amazon Lex 将使用默认值。如果请求替换现有的机器人，也是如此。

默认值为 300 秒（5 分钟）。

类型：整数

有效范围：最小值为 60。最大值为 86400。

必需：否

intents

Intent 数据元数组。每个意图都代表一个用户可以表达的命令。例如，披萨订购机器人可能支持某种 OrderPizza 意图。有关更多信息，请参阅 Amazon Lex：工作原理。

类型：Intent 对象数组

必需：否

locale

指定机器人的目标区域设置。机器人中使用的任何意图都必须与机器人的区域设置兼容。

默认值为 en-US。

类型：字符串

有效值：de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR

必需：是

nluIntentConfidenceThreshold

确定在或PostText响应中返回替代意图时 AMAZON.FallbackIntentAMAZON.KendraSearchIntent，Amazon Lex 将在何处插入、PostContent或两者的阈值。 AMAZON.FallbackIntentAMAZON.KendraSearchIntent并且只有在为机器人配置时才会被插入。

必须将 enableModelImprovements 参数设置为 true，才能在以下区域中使用置信度分数。

- 美国东部（弗吉尼亚州北部）(us-east-1)
- 美国西部（俄勒冈州）(us-west-2)
- 亚太地区（悉尼）(ap-southeast-2)
- 欧洲（爱尔兰）(eu-west-1)

在其他区域中，enableModelImprovements 参数默认设置为 true。

例如，假设为机器人配置的置信度阈值为 0.80 和 AMAZON.FallbackIntent。Amazon Lex 返回了三个替代意图，置信度分数如下：IntentA (0.70)、IntentB (0.60) 和 IntentC (0.50)。来自 PostText 操作的响应将是：

- 亚马逊。 FallbackIntent
- IntentA
- IntentB
- IntentC

类型：双精度

有效范围：最小值为 0。最大值为 1。

必需：否

processBehavior

如果您将 processBehavior 元素设置为 BUILD，Amazon Lex 会构建机器人以便它可以运行。如果您将该元素设置为 SAVE，Amazon Lex 会保存机器人，但不会构建它。

如果您不指定此值，则默认值为 BUILD。

类型：字符串

有效值：SAVE | BUILD

必需：否

tags

要添加到自动程序的标签列表。您只能在创建机器人时添加标签，不能使用 PutBot 操作来更新机器人上的标签。要更新标签，请使用 TagResource 操作。

类型：Tag 对象数组

数组成员：最少 0 个物品。最多 200 项。

必需：否

voiceId

您希望 Amazon Lex 用于与用户进行语音交互的 Amazon Polly 语音 ID。为语音配置的区域设置必须与机器人的区域设置相匹配。有关更多信息，请参阅《Amazon Polly 开发人员指南》中的 Amazon Polly 中的语音。

类型：字符串

必需：否

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "abortStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "checksum": "string",
   "childDirected": boolean,
   "clarificationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
```

```
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "createdDate": number,
   "createVersion": boolean,
   "description": "string",
   "detectSentiment": boolean,
   "enableModelImprovements": boolean,
   "failureReason": "string",
   "idleSessionTTLInSeconds": number,
   "intents": [
      {
         "intentName": "string",
         "intentVersion": "string"
      }
   ],
   "lastUpdatedDate": number,
   "locale": "string",
   "name": "string",
   "nluIntentConfidenceThreshold": number,
   "status": "string",
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ],
   "version": "string",
   "voiceId": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

abortStatement

Amazon Lex 用来取消对话的消息。有关更多信息，请参阅 PutBot。

类型：Statement 对象

checksum

您创建的机器人的校验和。

类型：字符串

childDirected

对于使用 Amazon Lex 模型构建服务创建的每个 Amazon Lex 机器人，您都必须通过在 childDirected 字段中指定 true 或 false，指定您对 Amazon Lex 的使用是否与全部或部分针对 13 岁以下儿童且受《儿童在线隐私保护法》(COPPA) 约束的网站、程序或其他应用程序有关。在 childDirected 字段中指定 true，即表示您确认您对 Amazon Lex 的使用确实与全部或部分针对 13 岁以下儿童且受 COPPA 约束的网站、计划或其他应用程序有关。在 childDirected 字段中指定 false，即表示您确认您对 Amazon Lex 的使用不与全部或部分针对 13 岁以下儿童且受 COPPA 约束的网站、计划或其他应用程序有关。如果在 childDirected 字段中指定默认值不能正确反映您确认您对 Amazon Lex 的使用不与全部或部分针对 13 岁以下儿童且受 COPPA 约束的网站、计划或其他应用程序有关，则您可以不指定。

如果您对 Amazon Lex 的使用涉及全部或部分针对 13 岁以下儿童的网站、程序或其他应用程序，则必须获得 COPPA 规定的任何必需的可核实的家长同意。有关将 Amazon Lex 用于全部或部分针对 13 岁以下儿童的网站、程序或其他应用程序的信息，请参阅 Amazon Lex 常见问题解答。

类型：布尔值

clarificationPrompt

Amazon Lex 在无法理解用户的请求时使用的提示。有关更多信息，请参阅 PutBot。

类型：Prompt 对象

createdDate

机器人的创建日期。

类型：时间戳

createVersion

True（如果创建了新版本的机器人）。如果请求中未指定 createVersion 字段，则在响应中将 createVersion 字段设置为 false。

类型：布尔值

description

机器人的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

detectSentiment

`true`（如果将机器人配置为将用户言语发送到 Amazon Comprehend 进行情绪分析）。如果请求中未指定 `detectSentiment` 字段，则响应中的 `detectSentiment` 字段为 `false`。

类型：布尔值

enableModelImprovements

表示机器人是否使用精度改进。`true` 表示机器人正在使用改进，否则为 `false`。

类型：布尔值

failureReason

如果 `status` 是 `FAILED`，则 Amazon Lex 会提供其未能构建机器人的原因。

类型：字符串

idleSessionTTLInSeconds

Amazon Lex 保留对话中收集的数据的最长时间。有关更多信息，请参阅 PutBot。

类型：整数

有效范围：最小值为 60。最大值为 86400。

intents

`Intent` 数据元数组。有关更多信息，请参阅 PutBot。

类型：Intent 对象数组

lastUpdatedDate

机器人的更新日期。创建资源时，创建日期和上次更新日期相同。

类型：时间戳

locale

机器人的目标区域设置。

类型：字符串

有效值：de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR

name

机器人的名称。

类型：字符串

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

nluIntentConfidenceThreshold

该分数决定 Amazon Lex 在或PostText响应中返回替代意图时在何处插入AMAZON.KendraSearchIntent、PostContent或两者。AMAZON.FallbackIntent AMAZON.FallbackIntent如果实际可信度分数低于此值，则会插入。AMAZON.KendraSearchIntent只有在为机器人配置时才会插入。

类型：双精度

有效范围：最小值为 0。最大值为 1。

status

当您发送创建 processBehavior 设置为 BUILD 的机器人的请求时，Amazon Lex 会将 status 响应元素设置为 BUILDING。

在 READY_BASIC_TESTING 状态下，您可以使用用户输入来测试机器人，这些输入与为机器人意图和插槽类型中的值配置的言语完全匹配。

如果 Amazon Lex 无法构建机器人，则 Amazon Lex 会将 status 设置为 FAILED。Amazon Lex 会在 failureReason 响应元素中返回失败的原因。

当将 processBehavior 设置为 SAVE 时，Amazon Lex 会将状态代码设置为 NOT BUILT。

当机器人处于 READY 状态时，您可以测试和发布该机器人。

类型：字符串

有效值：BUILDING | READY | READY_BASIC_TESTING | FAILED | NOT_BUILT

tags

与机器人关联的标签的列表。

类型：Tag 对象数组

数组成员：最少 0 个物品。最多 200 项。

version

自动程序的版本。对于新机器人，版本始终是 $LATEST。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

voiceId

Amazon Lex 用于和用户进行语音交互的 Amazon Polly 语音 ID。有关更多信息，请参阅 PutBot。

类型：字符串

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

PreconditionFailedException

您尝试更改的资源的校验和与请求中的校验和不匹配。检查资源的校验和并重试。

HTTP 状态代码：412

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

PutBotAlias

服务: Amazon Lex Model Building Service

为指定版本的机器人创建别名，或替换指定机器人的别名。要更改别名指向的机器人的版本，请替换别名。有关别名的更多信息，请参阅版本控制和别名。

此操作需要 `lex:PutBotAlias` 操作的权限。

请求语法

```
PUT /bots/botName/aliases/name HTTP/1.1
Content-type: application/json

{
   "botVersion": "string",
   "checksum": "string",
   "conversationLogs": {
      "iamRoleArn": "string",
      "logSettings": [
         {
            "destination": "string",
            "kmsKeyArn": "string",
            "logType": "string",
            "resourceArn": "string"
         }
      ]
   },
   "description": "string",
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ]
}
```

URI 请求参数

请求使用以下 URI 参数。

botName

机器人的名称。

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

必需：是

name

别名的名称。名称不区分大小写。

长度限制：长度下限为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

必需：是

请求体

请求接受采用 JSON 格式的以下数据。

botVersion

自动程序的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

必需：是

checksum

标识 $LATEST 版本的特定修订版。

创建新机器人别名时，请将 checksum 字段留空。如果指定校验和，则会出现 BadRequestException 异常。

当您想更新机器人别名时，请将 checksum 字段设置为 $LATEST 版本最新修订版的校验和。如果您未指定 checksum 字段，或者校验和与 $LATEST 版本不匹配，则会出现 PreconditionFailedException 异常。

类型：字符串

必需：否

conversationLogs

别名的对话日志的设置。

类型：ConversationLogsRequest 对象

必需：否

description

别名的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

必需：否

tags

要添加到机器人的标签列表。您只能在创建别名时添加标签，不能使用 PutBotAlias 操作来更新机器人别名上的标签。要更新标签，请使用 TagResource 操作。

类型：Tag 对象数组

数组成员：最少 0 个物品。最多 200 项。

必需：否

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "botName": "string",
   "botVersion": "string",
   "checksum": "string",
   "conversationLogs": {
      "iamRoleArn": "string",
      "logSettings": [
         {
            "destination": "string",
```

```
            "kmsKeyArn": "string",
            "logType": "string",
            "resourceArn": "string",
            "resourcePrefix": "string"
         }
      ]
   },
   "createdDate": number,
   "description": "string",
   "lastUpdatedDate": number,
   "name": "string",
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ]
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

botName

别名指向的自动程序的名称。

类型：字符串

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

botVersion

别名指向的机器人的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

checksum

别名的最新版本的校验和。

类型：字符串

conversationLogs

确定 Amazon Lex 如何使用对话日志作为别名的设置。

类型：ConversationLogsResponse 对象

createdDate

机器人别名的创建日期。

类型：时间戳

description

别名的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

lastUpdatedDate

机器人别名的更新日期。创建资源时，创建日期和上次更新日期相同。

类型：时间戳

name

别名的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

tags

与机器人关联的标签的列表。

类型：Tag 对象数组

数组成员：最少 0 个物品。最多 200 项。

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

PreconditionFailedException

您尝试更改的资源的校验和与请求中的校验和不匹配。检查资源的校验和并重试。

HTTP 状态代码：412

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2

- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

PutIntent

服务: Amazon Lex Model Building Service

创建目的或替换现有目的。

要定义用户和您的机器人之间的互动，您可以使用一个或多个意图。例如，对于订购披萨机器人，您可以创建一个 `OrderPizza` 意图。

要创建意图或替换现有意图，您必须提供以下信息：

- 意图名称。例如 `OrderPizza`。
- 示例言语。例如，“请问我可以点披萨吗。” 和“我想订一个披萨。”
- 有待收集的信息。您可以为机器人向用户请求的信息指定插槽类型。您可以指定标准插槽类型（例如日期或时间），也可以指定自定义插槽类型（例如披萨的大小和外皮）。
- 履行意图的方式。您可以提供 Lambda 函数或配置意图以将意图信息返回给客户端应用程序。如果您使用 Lambda 函数，则当所有意图信息都可用时，Amazon Lex 会调用您的 Lambda 函数。如果您将意图配置为仅将意图信息返回给客户端应用程序。

您可以在请求中指定其他可选信息，例如：

- 要求用户确认意图的确认提示。例如，“我可以为您的披萨下订单了吗？”
- 意图实现后要发送给用户的结论语句。例如，“我已经为您的披萨下订单了。”
- 一个后续提示，询问用户是否要进行其他活动。例如，询问“您想和披萨一起订一杯饮品吗？”

如果您指定现有意图名称来更新意图，Amazon Lex 会将意图的 `$LATEST` 版本中的值替换为请求中的值。Amazon Lex 会删除您在请求中未提供的字段。如果您没有指定必填字段，Amazon Lex 会引发异常。更新意图的 `$LATEST` 版本时，任何使用意图的 `$LATEST` 版本的机器人的 `status` 字段都将设置为 `NOT_BUILT`。

有关更多信息，请参阅 Amazon Lex：工作原理。

此操作需要 `lex:PutIntent` 操作的权限。

请求语法

```
PUT /intents/name/versions/$LATEST HTTP/1.1
Content-type: application/json

{
```

```
    "checksum": "string",
    "conclusionStatement": {
       "messages": [
          {
             "content": "string",
             "contentType": "string",
             "groupNumber": number
          }
       ],
       "responseCard": "string"
    },
    "confirmationPrompt": {
       "maxAttempts": number,
       "messages": [
          {
             "content": "string",
             "contentType": "string",
             "groupNumber": number
          }
       ],
       "responseCard": "string"
    },
    "createVersion": boolean,
    "description": "string",
    "dialogCodeHook": {
       "messageVersion": "string",
       "uri": "string"
    },
    "followUpPrompt": {
       "prompt": {
          "maxAttempts": number,
          "messages": [
             {
                "content": "string",
                "contentType": "string",
                "groupNumber": number
             }
          ],
          "responseCard": "string"
       },
       "rejectionStatement": {
          "messages": [
             {
                "content": "string",
```

```
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   }
},
"fulfillmentActivity": {
   "codeHook": {
      "messageVersion": "string",
      "uri": "string"
   },
   "type": "string"
},
"inputContexts": [
   {
      "name": "string"
   }
],
"kendraConfiguration": {
   "kendraIndex": "string",
   "queryFilterString": "string",
   "role": "string"
},
"outputContexts": [
   {
      "name": "string",
      "timeToLiveInSeconds": number,
      "turnsToLive": number
   }
],
"parentIntentSignature": "string",
"rejectionStatement": {
   "messages": [
      {
         "content": "string",
         "contentType": "string",
         "groupNumber": number
      }
   ],
   "responseCard": "string"
},
"sampleUtterances": [ "string" ],
"slots": [
```

```
      {
         "defaultValueSpec": {
            "defaultValueList": [
               {
                  "defaultValue": "string"
               }
            ]
         },
         "description": "string",
         "name": "string",
         "obfuscationSetting": "string",
         "priority": number,
         "responseCard": "string",
         "sampleUtterances": [ "string" ],
         "slotConstraint": "string",
         "slotType": "string",
         "slotTypeVersion": "string",
         "valueElicitationPrompt": {
            "maxAttempts": number,
            "messages": [
               {
                  "content": "string",
                  "contentType": "string",
                  "groupNumber": number
               }
            ],
            "responseCard": "string"
         }
      }
   ]
}
```

URI 请求参数

请求使用以下 URI 参数。

name

意图的名称。名称不区分大小写。

该名称不能与内置意图名称匹配，否则含有“AMAZON.”的内置意图名称 会被删除。例如，由于存在名为 AMAZON.HelpIntent 的内置意图，因此您无法创建名为的 HelpIntent 的自定义意图。

有关内置目的的列表，请参阅 Alexa Skills Kit 中的标准内置目的。

长度限制：长度下限为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

必需：是

请求体

请求接受采用 JSON 格式的以下数据。

checksum

标识 $LATEST 版本的特定修订版。

创建新意图时，请将 checksum 字段留空。如果指定校验和，则会出现 BadRequestException 异常。

当您想更新意图时，请将 checksum 字段设置为 $LATEST 版本最新修订版的校验和。如果您未指定 checksum 字段，或者校验和与 $LATEST 版本不匹配，则会出现 PreconditionFailedException 异常。

类型：字符串

必需：否

conclusionStatement

您希望 Amazon Lex 在 Lambda 函数成功履行意图后向用户传达的语句。

仅当您在 fulfillmentActivity 中提供 Lambda 函数时，此元素才有意义。如果您将意图返回给客户端应用程序，则无法指定此元素。

> **Note**
>
> followUpPrompt 和 conclusionStatement 是互斥的。您只能指定其中一个。

类型：Statement 对象

必需：否

confirmationPrompt

提示用户确认目的。此问题的回答应为 yes 或 no。

Amazon Lex 使用此提示来确保用户确认意图可履行。例如，对于 OrderPizza 意图，您可能需要在下单之前确认订单是正确的。对于其他意图，例如仅回答用户问题的意图，在提供信息之前，您可能无需要求用户进行确认。

> **Note**
>
> 您必须同时提供 rejectionStatement 和 confirmationPrompt，或者两者都不提供。

类型：Prompt 对象

必需：否

createVersion

当设置为 true 时，将创建意图的新编号版本。这与调用 CreateIntentVersion 操作相同。如果不指定 createVersion，则默认值为 false。

类型：布尔值

必需：否

description

目的的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

必需：否

dialogCodeHook

指定为每个用户输入调用的 Lambda 函数。您可以调用此 Lambda 函数对用户交互进行个性化。

例如，假设您的机器人确定用户是 John。您的 Lambda 函数可能会从后端数据库中检索 John 的信息并预先填充一些值。例如，如果您发现 John 对谷蛋白过敏，则可以将相应的意图插槽 GlutenIntolerant 设置为 true。您可能会找到 John 的电话号码并设置相应的会话属性。

类型：CodeHook 对象

必需：否

followUpPrompt

Amazon Lex 将在履行意图后使用此提示来征求其他活动。例如，在履行 OrderPizza 意图后，您可以提示用户订购饮品。

Amazon Lex 采取的操作取决于用户的响应，如下所示：

- 如果用户说“是”，则会使用为机器人配置的澄清提示进行响应。
- 如果用户说“是”，然后继续说一句触发意图的言语，则会启动针对该意图的对话。
- 如果用户说“否”，则会使用为后续提示配置的拒绝语句进行响应。
- 如果它无法识别出这句话，它会再次重复后续提示。

followUpPrompt 字段和 conclusionStatement 字段是互斥的。您只能指定其中一个。

类型：FollowUpPrompt 对象

必需：否

fulfillmentActivity

必需。描述履行意图的方式。例如，在用户提供披萨订单的所有信息后，fulfillmentActivity 定义机器人如何向当地披萨店下订单。

您可以将 Amazon Lex 配置为将所有意图信息返回给客户端应用程序，或者指示其调用可以处理意图的 Lambda 函数（例如，向披萨店下订单）。

类型：FulfillmentActivity 对象

必需：否

inputContexts

InputContext 对象数组，列出了 Amazon Lex 在与用户的对话中选择意图时必须处于活动状态的上下文。

类型：InputContext 对象数组

数组成员：最少 0 个物品。最多 5 项。

必需：否

kendraConfiguration

使用 AMAZON.KendraSearchIntent 意图以连接至 Amazon Kendra 索引所需的配置信息。有关更多信息，请参阅亚马逊。 KendraSearchIntent。

类型：KendraConfiguration 对象

必需：否

outputContexts

OutputContext 对象数组，列出了履行意图时意图激活的上下文。

类型：OutputContext 对象数组

数组成员：最少 0 个物品。最多 10 个物品。

必需：否

parentIntentSignature

内置目的的唯一标识符，此目的建立在它的基础之上。要查找意图的签名，请参阅 Alexa Skills Kit 中的标准内置意图。

类型：字符串

必需：否

rejectionStatement

当用户对 confirmationPrompt 中定义的问题回答“否”时，Amazon Lex 将使用此响应进行响应，以确认意图已被取消。

> **Note**
>
> 您必须同时提供 rejectionStatement 和 confirmationPrompt，或者两者都不提供。

类型：Statement 对象

必需：否

sampleUtterances

用户可能用以传递意图的一组言语（字符串）。例如，“我想要 {PizzaSize} 披萨”、“订购 {Quantity} {PizzaSize} 个披萨”。

在每句言语中，插槽名称都用大括号括起来。

类型：字符串数组

数组成员：最少 0 项。最多 1500 项。

长度限制：长度下限为 1。最大长度为 200。

必需：否

slots

意图插槽数组。在运行时，Amazon Lex 使用插槽中定义的提示从用户那里引发所需的插槽值。有关更多信息，请参阅 Amazon Lex：工作原理。

类型：Slot 对象数组

数组成员：最少 0 个物品。最多 100 个项目。

必需：否

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "checksum": "string",
   "conclusionStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "confirmationPrompt": {
      "maxAttempts": number,
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
```

```
  },
  "createdDate": number,
  "createVersion": boolean,
  "description": "string",
  "dialogCodeHook": {
     "messageVersion": "string",
     "uri": "string"
  },
  "followUpPrompt": {
     "prompt": {
        "maxAttempts": number,
        "messages": [
           {
              "content": "string",
              "contentType": "string",
              "groupNumber": number
           }
        ],
        "responseCard": "string"
     },
     "rejectionStatement": {
        "messages": [
           {
              "content": "string",
              "contentType": "string",
              "groupNumber": number
           }
        ],
        "responseCard": "string"
     }
  },
  "fulfillmentActivity": {
     "codeHook": {
        "messageVersion": "string",
        "uri": "string"
     },
     "type": "string"
  },
  "inputContexts": [
     {
        "name": "string"
     }
  ],
  "kendraConfiguration": {
```

```
      "kendraIndex": "string",
      "queryFilterString": "string",
      "role": "string"
   },
   "lastUpdatedDate": number,
   "name": "string",
   "outputContexts": [
      {
         "name": "string",
         "timeToLiveInSeconds": number,
         "turnsToLive": number
      }
   ],
   "parentIntentSignature": "string",
   "rejectionStatement": {
      "messages": [
         {
            "content": "string",
            "contentType": "string",
            "groupNumber": number
         }
      ],
      "responseCard": "string"
   },
   "sampleUtterances": [ "string" ],
   "slots": [
      {
         "defaultValueSpec": {
            "defaultValueList": [
               {
                  "defaultValue": "string"
               }
            ]
         },
         "description": "string",
         "name": "string",
         "obfuscationSetting": "string",
         "priority": number,
         "responseCard": "string",
         "sampleUtterances": [ "string" ],
         "slotConstraint": "string",
         "slotType": "string",
         "slotTypeVersion": "string",
         "valueElicitationPrompt": {
```

```
            "maxAttempts": number,
            "messages": [
               {
                  "content": "string",
                  "contentType": "string",
                  "groupNumber": number
               }
            ],
            "responseCard": "string"
         }
      }
   ],
   "version": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

checksum

创建或更新意图的 $LATEST 版本的校验和。

类型：字符串

conclusionStatement

在 fulfillmentActivity 意图中指定的 Lambda 函数履行意图后，Amazon Lex 会将此语句传达给用户。

类型：Statement 对象

confirmationPrompt

如果在意图中已定义，则 Amazon Lex 会在履行意图之前提示用户确认意图。

类型：Prompt 对象

createdDate

意图的创建日期。

类型：时间戳

createVersion

True（如果创建了新版本的意图）。如果请求中未指定 createVersion 字段，则在响应中将 createVersion 字段设置为 false。

类型：布尔值

description

目的的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

dialogCodeHook

如果在意图中已定义，Amazon Lex 会为每个用户输入调用 Lambda 函数。

类型：CodeHook 对象

followUpPrompt

如果已在意图中定义，则 Amazon Lex 会在履行意图后使用此提示来征求其他用户活动。

类型：FollowUpPrompt 对象

fulfillmentActivity

如果已在意图中定义，则在用户提供意图所需的所有信息后，Amazon Lex 会调用 Lambda 函数来履行意图。

类型：FulfillmentActivity 对象

inputContexts

InputContext 对象数组，列出了 Amazon Lex 在与用户的对话中选择意图时必须处于活动状态的上下文。

类型：InputContext 对象数组

数组成员：最少 0 个物品。最多 5 项。

kendraConfiguration

连接至 Amazon Kendra 索引并使用 AMAZON.KendraSearchIntent 意图所需的配置信息（如果有）。

类型：KendraConfiguration 对象

lastUpdatedDate

意图的更新日期。创建资源时，创建日期和上次更新日期相同。

类型：时间戳

name

意图的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

outputContexts

OutputContext 对象数组，列出了履行意图时意图激活的上下文。

类型：OutputContext 对象数组

数组成员：最少 0 个物品。最多 10 个物品。

parentIntentSignature

内置意图唯一标识符，此意图建立在该意图的基础之上。

类型：字符串

rejectionStatement

当用户对 confirmationPrompt 中定义的问题回答“否”时，Amazon Lex 将使用此语句进行响应，以确认意图已被取消。

类型：Statement 对象

sampleUtterances

为意图配置的一组示例言语。

类型：字符串数组

数组成员：最少 0 项。最多 1500 项。

长度限制：长度下限为 1。最大长度为 200。

slots

为意图配置的一系列意图插槽。

类型：Slot 对象数组

数组成员：最少 0 个物品。最多 100 个项目。

version

意图的版本。对于新意图，版本始终是 $LATEST。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

PreconditionFailedException

您尝试更改的资源的校验和与请求中的校验和不匹配。检查资源的校验和并重试。

HTTP 状态代码：412

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

PutSlotType

服务: Amazon Lex Model Building Service

创建自定义槽类型或替换现有自定义槽类型。

要创建自定义插槽类型，请为插槽类型指定名称和一组枚举值，这些值是此类插槽可以假设的值。有关更多信息，请参阅 Amazon Lex：工作原理。

如果您指定现有插槽类型的名称，则请求中的字段将替换该插槽类型 $LATEST 版本中的现有值。Amazon Lex 会删除您在请求中未提供的字段。如果您没有指定必填字段，Amazon Lex 会引发异常。更新插槽类型的 $LATEST 版本时，如果机器人使用包含该插槽类型的意图的 $LATEST 版本，则机器人的 status 字段将设置为 NOT_BUILT。

此操作需要 lex:PutSlotType 操作的权限。

请求语法

```
PUT /slottypes/name/versions/$LATEST HTTP/1.1
Content-type: application/json

{
   "checksum": "string",
   "createVersion": boolean,
   "description": "string",
   "enumerationValues": [
      {
         "synonyms": [ "string" ],
         "value": "string"
      }
   ],
   "parentSlotTypeSignature": "string",
   "slotTypeConfigurations": [
      {
         "regexConfiguration": {
            "pattern": "string"
         }
      }
   ],
   "valueSelectionStrategy": "string"
}
```

URI 请求参数

请求使用以下 URI 参数。

name

槽类型的名称。名称不区分大小写。

该名称不能与内置插槽类型名称匹配，否则带有“AMAZON”的内置插槽类型名称 会被删除。例如，由于有一个名为 AMAZON.DATE 的内置插槽类型，因此您无法创建名为 DATE 的自定义插槽类型。

有关内置插槽类型的列表，请参阅 Alexa Skills Kit 中的插槽类型参考。

长度限制：长度下限为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

必需：是

请求体

请求接受采用 JSON 格式的以下数据。

checksum

标识 $LATEST 版本的特定修订版。

创建新的插槽类型时，请将 checksum 字段留空。如果指定校验和，则会出现 BadRequestException 异常。

当您想更新插槽类型时，请将 checksum 字段设置为 $LATEST 版本最新修订版的校验和。如果您未指定 checksum 字段，或者校验和与 $LATEST 版本不匹配，则会出现 PreconditionFailedException 异常。

类型：字符串

必需：否

createVersion

当设置为 true 时，将创建插槽类型的新编号版本。这与调用 CreateSlotTypeVersion 操作相同。如果不指定 createVersion，则默认值为 false。

类型：布尔值

必需：否

description

槽类型的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

必需：否

enumerationValues

定义插槽类型可采用值的 EnumerationValue 对象的列表。每个值可以有 synonyms 的列表，后者可帮助训练机器学习模型以便解析插槽值。

正则表达式插槽类型不需要枚举值。所有其他插槽类型都需要枚举值列表。

当 Amazon Lex 解析插槽值时，它会生成一个分辨率列表，其中包含该插槽最多五个可能的值。如果您使用 Lambda 函数，则此解析列表将传递给该函数。如果您未使用 Lambda 函数，可以选择返回用户输入的值或解析列表中的第一个值并作为插槽值。valueSelectionStrategy 字段表示要使用的选项。

类型：EnumerationValue 对象数组

数组成员：最少 0 个物品。最多 10000 项。

必需：否

parentSlotTypeSignature

用作此插槽类型的父级的内置插槽类型。当您定义父级插槽类型时，新的插槽类型将具有父级插槽类型的所有相同配置。

仅支持 AMAZON.AlphaNumeric。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：^((AMAZON\.)_?|[A-Za-z]_?)+

必需：否

slotTypeConfigurations

扩展父级内置插槽类型的配置信息。配置已添加到父插槽类型的设置中。

类型：SlotTypeConfiguration 对象数组

数组成员：最少 0 个物品。最多 10 个物品。

必需：否

valueSelectionStrategy

确定 Amazon Lex 用来返回插槽值类型的插槽解析策略。该字段可以为以下值之一：

- ORIGINAL_VALUE — 如果用户值与插槽值相近，返回由用户输入的值。
- TOP_RESOLUTION — 如果有插槽解析列表，返回解析列表中的第一个值并作为插槽类型值。如果没有解析列表，则返回 null。

如果不指定 valueSelectionStrategy，则默认值为 ORIGINAL_VALUE。

类型：字符串

有效值：ORIGINAL_VALUE | TOP_RESOLUTION

必需：否

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "checksum": "string",
   "createdDate": number,
   "createVersion": boolean,
   "description": "string",
   "enumerationValues": [
      {
         "synonyms": [ "string" ],
         "value": "string"
      }
```

```
    ],
    "lastUpdatedDate": number,
    "name": "string",
    "parentSlotTypeSignature": "string",
    "slotTypeConfigurations": [
       {
          "regexConfiguration": {
             "pattern": "string"
          }
       }
    ],
    "valueSelectionStrategy": "string",
    "version": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

checksum

插槽类型的 $LATEST 版本的校验和。

类型：字符串

createdDate

插槽类型的创建日期。

类型：时间戳

createVersion

True（如果创建了新版本的插槽类型）。如果请求中未指定 createVersion 字段，则在响应中将 createVersion 字段设置为 false。

类型：布尔值

description

槽类型的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

enumerationValues

定义插槽类型可采用值的 EnumerationValue 对象的列表。

类型：EnumerationValue 对象数组

数组成员：最少 0 个物品。最多 10000 项。

lastUpdatedDate

插槽类型的更新日期。创建插槽类型时，创建日期和上次更新日期相同。

类型：时间戳

name

槽类型的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

parentSlotTypeSignature

用作此插槽类型的父级的内置插槽类型。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：^((AMAZON\.)_?|[A-Za-z]_?)+

slotTypeConfigurations

扩展父级内置插槽类型的配置信息。

类型：SlotTypeConfiguration 对象数组

数组成员：最少 0 个物品。最多 10 个物品。

valueSelectionStrategy

Amazon Lex 用来确定插槽值的插槽解析策略。有关更多信息，请参阅 PutSlotType。

类型：字符串

有效值：ORIGINAL_VALUE | TOP_RESOLUTION

version

插槽类型的版本。对于新的插槽类型，版本始终为 $LATEST。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

PreconditionFailedException

您尝试更改的资源的校验和与请求中的校验和不匹配。检查资源的校验和并重试。

HTTP 状态代码：412

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

StartImport

服务: Amazon Lex Model Building Service

启动作业以将资源导入到 Amazon Lex 中。

请求语法

```
POST /imports/ HTTP/1.1
Content-type: application/json

{
   "mergeStrategy": "string",
   "payload": blob,
   "resourceType": "string",
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ]
}
```

URI 请求参数

该请求不使用任何 URI 参数。

请求正文

请求接受采用 JSON 格式的以下数据。

mergeStrategy

指定当存在同名的现有资源时，应采取的 StartImport 操作。

- FAIL_ON_CONFLICT — 导入文件中的资源与现有资源发生第一次冲突时，导入操作将停止。导致冲突的资源名称位于对 GetImport 操作的响应的 failureReason 字段中。

  OVERWRITE_LATEST — 即使与现有资源发生冲突，导入操作也会继续进行。现有资源的 $LASTEST 版本会被导入文件中的数据覆盖。

类型：字符串

有效值：OVERWRITE_LATEST | FAIL_ON_CONFLICT

必需：是

payload

二进制格式的 zip 存档。存档应包含一个文件：一个包含要导入的资源的 JSON 文件。资源应与在 `resourceType` 字段中指定的类型相匹配。

类型：Base64 编码的二进制数据对象

必需：是

resourceType

指定要导出的资源的类型。每种资源还会导出它所依赖的任何资源。

- 机器人会导出依赖的意图。
- 意图会导出依赖的插槽类型。

类型：字符串

有效值：`BOT | INTENT | SLOT_TYPE`

必需：是

tags

要添加到导入的机器人的标签列表。您只能在导入机器人时添加标签，不能为意图或插槽类型添加标签。

类型：Tag 对象数组

数组成员：最少 0 个物品。最多 200 项。

必需：否

响应语法

```
HTTP/1.1 201
Content-type: application/json

{
   "createdDate": number,
   "importId": "string",
   "importStatus": "string",
   "mergeStrategy": "string",
```

```
   "name": "string",
   "resourceType": "string",
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ]
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 201 响应。

服务以 JSON 格式返回的以下数据。

createdDate

请求导入作业的日期和时间的时间戳。

类型：时间戳

importId

特定导入作业的标识符。

类型：字符串

importStatus

导入作业的状态。如果状态为 FAILED，则可以使用 GetImport 操作获取失败的原因。

类型：字符串

有效值：IN_PROGRESS | COMPLETE | FAILED

mergeStrategy

发生合并冲突时要执行的操作。

类型：字符串

有效值：OVERWRITE_LATEST | FAIL_ON_CONFLICT

name

提供给导入作业的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：[a-zA-Z_]+

resourceType

要导入的资源的类型。

类型：字符串

有效值：BOT | INTENT | SLOT_TYPE

tags

添加到导入的机器人的标签列表。

类型：Tag 对象数组

数组成员：最少 0 个物品。最多 200 项。

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

StartMigration

服务: Amazon Lex Model Building Service

开始从 Amazon Lex V1 向 Amazon Lex V2 迁移机器人。当您想要利用 Amazon Lex V2 的新功能时，请迁移机器人。

有关更多信息，请参阅 Amazon Lex 开发人员指南中的迁移机器人。

请求语法

```
POST /migrations HTTP/1.1
Content-type: application/json

{
   "migrationStrategy": "string",
   "v1BotName": "string",
   "v1BotVersion": "string",
   "v2BotName": "string",
   "v2BotRole": "string"
}
```

URI 请求参数

该请求不使用任何 URI 参数。

请求正文

请求接受采用 JSON 格式的以下数据。

migrationStrategy

用于进行迁移的策略。

- CREATE_NEW — 创建新的 Amazon Lex V2 机器人并将 Amazon Lex V1 机器人迁移到新机器人。
- UPDATE_EXISTING — 覆盖现有的 Amazon Lex V2 机器人元数据和正在迁移的区域设置。它不会更改 Amazon Lex V2 机器人中的任何其他区域设置。如果该区域设置不存在，则会在 Amazon Lex V2 机器人中创建一个新的区域设置。

类型：字符串

有效值：CREATE_NEW | UPDATE_EXISTING

必需：是

v1BotName

要迁移到 Amazon Lex V2 的 Amazon Lex V1 机器人的名称。

类型：字符串

长度限制：最小长度为 2。最大长度为 50。

模式：`^([A-Za-z]_?)+$`

必需：是

v1BotVersion

要迁移到 Amazon Lex V2 的机器人的版本。您可以迁移 `$LATEST` 版本以及任何编号的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：`\$LATEST|[0-9]+`

必需：是

v2BotName

要将 Amazon Lex V1 机器人迁移到其中的 Amazon Lex V2 的名称。

- 如果 Amazon Lex V2 机器人不存在，则必须使用 `CREATE_NEW` 迁移策略。
- 如果 Amazon Lex V2 机器人存在，则必须使用 `UPDATE_EXISTING` 迁移策略来更改 Amazon Lex V2 机器人的内容。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：`^([0-9a-zA-Z][_-]?)+$`

必需：是

v2BotRole

Amazon Lex 用来运行 Amazon Lex V2 机器人的 IAM 角色。

类型：字符串

长度约束：最小长度为 20。最大长度为 2048。

模式：^arn:[\w\-]+:iam::[\d]{12}:role/.+$

必需：是

响应语法

```
HTTP/1.1 202
Content-type: application/json

{
   "migrationId": "string",
   "migrationStrategy": "string",
   "migrationTimestamp": number,
   "v1BotLocale": "string",
   "v1BotName": "string",
   "v1BotVersion": "string",
   "v2BotId": "string",
   "v2BotRole": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 202 响应。

服务以 JSON 格式返回以下数据。

migrationId

Amazon Lex 为迁移分配的唯一标识符。

类型：字符串

长度限制：固定长度为 10。

模式：^[0-9a-zA-Z]+$

migrationStrategy

用于进行迁移的策略。

类型：字符串

有效值：CREATE_NEW | UPDATE_EXISTING

migrationTimestamp

启动迁移的日期和时间。

类型：时间戳

v1BotLocale

Amazon Lex V1 机器人使用的区域设置。

类型：字符串

有效值：de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR

v1BotName

要迁移到 Amazon Lex V2 的 Amazon Lex V1 机器人的名称。

类型：字符串

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

v1BotVersion

要迁移到 Amazon Lex V2 的机器人的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

v2BotId

Amazon Lex V2 机器人的唯一标识符。

类型：字符串

长度限制：固定长度为 10。

模式：^[0-9a-zA-Z]+$

v2BotRole

Amazon Lex 用来运行 Amazon Lex V2 机器人的 IAM 角色。

类型：字符串

长度约束：最小长度为 20。最大长度为 2048。

模式：`^arn:[\w\-]+:iam::[\d]{12}:role/.+$`

错误

AccessDeniedException

您的 IAM 用户或角色无权调用迁移机器人 APIs 所需的 Amazon Lex V2。

HTTP 状态代码：403

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

TagResource

服务: Amazon Lex Model Building Service

将指定的标签添加到指定的资源中。如果标签键已经存在，则会用新值替换现有的值。

请求语法

```
POST /tags/resourceArn HTTP/1.1
Content-type: application/json

{
   "tags": [
      {
         "key": "string",
         "value": "string"
      }
   ]
}
```

URI 请求参数

请求使用以下 URI 参数。

resourceArn

要标记的机器人、机器人别名或机器人通道的 Amazon 资源名称 (ARN)。

长度限制：长度下限为 1。最大长度为 1011。

必需：是

请求体

请求接受采用 JSON 格式的以下数据。

tags

要添加到资源的标签键的列表。如果标签键已经存在，则会用新值替换现有的值。

类型：Tag 对象数组

数组成员：最少 0 个物品。最多 200 项。

必需：是

响应语法

```
HTTP/1.1 204
```

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 204 响应。

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

UntagResource

服务: Amazon Lex Model Building Service

从机器人、机器人别名或机器人通道中删除标签。

请求语法

```
DELETE /tags/resourceArn?tagKeys=tagKeys HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

resourceArn

要从中删除标签的资源的 Amazon 资源名称 (ARN)。

长度限制：长度下限为 1。最大长度为 1011。

必需：是

tagKeys

要从资源中删除的标签键的列表。如果资源上不存在标签键，则会将其忽略。

数组成员：最少 0 个物品。最多 200 项。

长度限制：长度下限为 1。最大长度为 128。

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 204
```

响应元素

如果此操作成功，则该服务会发送回带有空 HTTP 正文的 HTTP 204 响应。

错误

BadRequestException

请求格式不正确。例如，值无效或必填字段未填充。检查字段值，然后重试。

HTTP 状态代码：400

ConflictException

处理请求时出现冲突。请再次尝试您的请求。

HTTP 状态代码：409

InternalFailureException

出现内部 Amazon Lex 错误。请再次尝试您的请求。

HTTP 状态代码：500

LimitExceededException

请求超出了限制。请再次尝试您的请求。

HTTP 状态代码：429

NotFoundException

找不到在请求中指定的资源。检查资源并重试。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK

- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

Amazon Lex 运行时服务

Amazon Lex 运行时服务支持以下操作：

- DeleteSession
- GetSession
- PostContent
- PostText
- PutSession

DeleteSession

服务: Amazon Lex Runtime Service

删除指定自动程序、别名和用户 ID 的会话信息。

请求语法

```
DELETE /bot/botName/alias/botAlias/user/userId/session HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

botAlias

包含会话数据的机器人的使用中别名。

必需：是

botName

包含会话数据的机器人的名称。

必需：是

userId

与会话数据关联的用户的标识符。

长度限制：最小长度为 2。最大长度为 100。

模式：[0-9a-zA-Z._:-]+

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
```

```
    "botAlias": "string",
    "botName": "string",
    "sessionId": "string",
    "userId": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

botAlias

与会话数据关联的机器人的使用中别名。

类型：字符串

botName

与会话关联的机器人的名称。

类型：字符串

sessionId

会话的唯一标识符。

类型：字符串

userId

客户端应用程序用户的 ID。

类型：字符串

长度限制：最小长度为 2。最大长度为 100。

模式：`[0-9a-zA-Z._:-]+`

错误

BadRequestException

请求验证失败，上下文中没有可用的消息，或者机器人构建失败、仍在进行中或者包含未构建的更改。

HTTP 状态代码：400

ConflictException

两个客户端使用相同的 AWS 账户、Amazon Lex 机器人和用户 ID。

HTTP 状态代码：409

InternalFailureException

内部服务错误。重试调用。

HTTP 状态代码：500

LimitExceededException

已超出限制。

HTTP 状态代码：429

NotFoundException

未找到所引用的资源（例如 Amazon Lex 机器人或别名）。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

GetSession

服务: Amazon Lex Runtime Service

返回指定自动程序、别名和用户 ID 的会话信息。

请求语法

```
GET /bot/botName/alias/botAlias/user/userId/session/?
checkpointLabelFilter=checkpointLabelFilter HTTP/1.1
```

URI 请求参数

请求使用以下 URI 参数。

botAlias

包含会话数据的机器人的使用中别名。

必需：是

botName

包含会话数据的机器人的名称。

必需：是

checkpointLabelFilter

用于筛选 `recentIntentSummaryView` 结构中返回的意图的字符串。

指定过滤器时，仅返回其 `checkpointLabel` 字段设置为该字符串的意图。

长度约束：最小长度为 1。最大长度为 255。

模式：`[a-zA-Z0-9-]+`

userId

客户端应用程序用户的 ID。Amazon Lex 使用它来识别用户与您的机器人的对话。

长度限制：最小长度为 2。最大长度为 100。

模式：`[0-9a-zA-Z._:-]+`

必需：是

请求体

该请求没有请求正文。

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "activeContexts": [
      {
         "name": "string",
         "parameters": {
            "string" : "string"
         },
         "timeToLive": {
            "timeToLiveInSeconds": number,
            "turnsToLive": number
         }
      }
   ],
   "dialogAction": {
      "fulfillmentState": "string",
      "intentName": "string",
      "message": "string",
      "messageFormat": "string",
      "slots": {
         "string" : "string"
      },
      "slotToElicit": "string",
      "type": "string"
   },
   "recentIntentSummaryView": [
      {
         "checkpointLabel": "string",
         "confirmationStatus": "string",
         "dialogActionType": "string",
         "fulfillmentState": "string",
         "intentName": "string",
         "slots": {
            "string" : "string"
         },
         "slotToElicit": "string"
```

```
      }
   ],
   "sessionAttributes": {
      "string" : "string"
   },
   "sessionId": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

activeContexts

会话的活动上下文列表。可以在履行意图时设置上下文，也可以通过调用 PostContent、PostText 或 PutSession 操作来设置上下文。

您可以使用上下文来控制可以跟进意图的意图，也可以修改应用程序的操作。

类型：ActiveContext 对象数组

数组成员：最少 0 个物品。最多 20 个项目。

dialogAction

描述机器人的当前状态。

类型：DialogAction 对象

recentIntentSummaryView

有关会话中使用的意图的一系列信息。系列最多可以包含三个摘要。如果会话中使用的意图超过三个，则 recentIntentSummaryView 操作将包含有关最近使用的三个意图的信息。

如果您在请求中设置 checkpointLabelFilter 参数，则系列仅包含带有指定标签的意图。

类型：IntentSummary 对象数组

数组成员：最少 0 个物品。最多 3 项。

sessionAttributes

表示会话特定上下文信息的键值对的映射。它包含在 Amazon Lex 与客户端应用程序之间传递的应用程序信息。

类型：字符串到字符串映射

sessionId

会话的唯一标识符。

类型：字符串

错误

BadRequestException

请求验证失败，上下文中没有可用的消息，或者机器人构建失败、仍在进行中或者包含未构建的更改。

HTTP 状态代码：400

InternalFailureException

内部服务错误。重试调用。

HTTP 状态代码：500

LimitExceededException

已超出限制。

HTTP 状态代码：429

NotFoundException

未找到所引用的资源（例如 Amazon Lex 机器人或别名）。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK

- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

PostContent

服务: Amazon Lex Runtime Service

将用户输入（文本或语音）发送到 Amazon Lex。客户端使用此 API 在运行时向 Amazon Lex 发送文本和音频请求。Amazon Lex 使用其为机器人构建的机器学习模型来解释用户输入。

`PostContent` 操作支持 8kHz 和 16kHz 的音频输入。在电话音频应用中，您可以使用 8kHz 音频来实现更高的语音识别精度。

在响应中，Amazon Lex 会返回下一条要传达给用户的消息。考虑以下示例消息：

- 对于用户输入“我想要披萨”，Amazon Lex 可能会返回一条响应，其中包含一条引发插槽数据的消息（例如，`PizzaSize`）：“您想要什么尺寸的披萨？”。
- 在用户提供所有披萨订单信息后，Amazon Lex 可能会返回一条消息以获取用户确认：“是否下披萨订单？”。
- 用户对确认提示回答“是”后，Amazon Lex 可能会返回结论语句：“谢谢，已经为您的奶酪披萨下订单。”。

并非所有 Amazon Lex 消息都需要用户响应。例如，结论语句不需要响应。有些消息只需要“是”或“否”响应。除了 `message` 之外，Amazon Lex 还提供了有关响应中消息的其他上下文，您可以使用这些上下文来增强客户行为，例如显示相应的客户端用户界面。考虑以下示例：

- 如果消息是为了引发插槽数据，Amazon Lex 会返回以下上下文信息：
  - `x-amz-lex-dialog-state` 标头设置为 `ElicitSlot`
  - `x-amz-lex-intent-name` 标头设置为当前上下文中的意图名称
  - `x-amz-lex-slot-to-elicit` 标头设置为 `message` 正在引发信息的插槽名称
  - `x-amz-lex-slots` 标头设置为使用当前值为意图配置的插槽映射
- 如果消息是确认提示，则 `x-amz-lex-dialog-state` 标头设置为 `Confirmation`，`x-amz-lex-slot-to-elicit` 标头将被省略。
- 如果消息是为意图配置的澄清提示，表示用户意图未被理解，则 `x-amz-dialog-state` 标头设置为 `ElicitIntent`，`x-amz-slot-to-elicit` 标头将被省略。

此外，Amazon Lex 还会返回您的应用程序特定的 `sessionAttributes`。有关更多信息，请参阅管理对话上下文。

请求语法

```
POST /bot/botName/alias/botAlias/user/userId/content HTTP/1.1
x-amz-lex-session-attributes: sessionAttributes
x-amz-lex-request-attributes: requestAttributes
Content-Type: contentType
Accept: accept
x-amz-lex-active-contexts: activeContexts

inputStream
```

URI 请求参数

请求使用以下 URI 参数。

accept

您将此值作为 Accept HTTP 标头进行传递。

根据请求中的 Accept HTTP 标头值，Amazon Lex 在响应中返回的消息可以是文本，也可以是语音。

- 如果值为 text/plain; charset=utf-8，则 Amazon Lex 会在响应中返回文本。
- 如果值以 audio/ 开头，则 Amazon Lex 会在响应中返回语音。Amazon Lex 使用 Amazon Polly 生成语音（使用您在 Accept 标头中指定的配置）。例如，如果您将 audio/mpeg 指定为值，Amazon Lex 将返回 MPEG 格式的语音。
- 如果值为 audio/pcm，则返回的语音采用 16 位小端序格式的 audio/pcm。
- 以下是接受的值：
  - audio/mpeg
  - audio/ogg
  - audio/pcm
  - text/plain; charset=utf-8
  - audio/*（默认值为 mpeg）

activeContexts

请求的活动上下文列表。可以在履行先前的意图时激活上下文，也可以通过在请求中包含上下文来激活上下文，

如果您未指定上下文列表，Amazon Lex 将使用会话的上下文的当前列表。如果您指定一个空列表，则会话的所有上下文都将被清除。

botAlias

Amazon Lex 机器人的别名。

必需：是

botName

Amazon Lex 机器人的名称。

必需：是

contentType

您将此值作为 `Content-Type` HTTP 标头进行传递。

表示音频格式或文本。标头值必须以下列其中一个前缀开头：

- PCM 格式，音频数据必须按小端序字节顺序排列。
  - audio/l16; rate=16000; channels=1
  - audio/x-l16; sample-rate=16000; channel-count=1
  - audio/lpcm；sample-rate=8000；=16；channel-count=1；=false sample-size-bits is-big-endian
- Opus 格式
  - audio/ x-cbr-opus-with-preamble；preamble-size=0；bit-rate=256000；=4 frame-size-milliseconds
- 文本格式
  - text/plain; charset=utf-8

必需：是

requestAttributes

您将此值作为 `x-amz-lex-request-attributes` HTTP 标头进行传递。

在 Amazon Lex 和客户端应用程序之间传递的特定于请求的信息。该值必须是带有字符串键和值的 JSON 序列化和 base64 编码映射。`requestAttributes` 和 `sessionAttributes` 标头的总大小限制为 12 KB。

命名空间 `x-amz-lex:` 保留供特殊属性使用。请勿创建带有 `x-amz-lex:` 前缀的任何请求属性。

有关更多信息，请参阅设置请求属性。

sessionAttributes

您将此值作为 x-amz-lex-session-attributes HTTP 标头进行传递。

在 Amazon Lex 和客户端应用程序之间传递的特定于应用程序的信息。该值必须是带有字符串键和值的 JSON 序列化和 base64 编码映射。sessionAttributes 和 requestAttributes 标头的总大小限制为 12 KB。

有关更多信息，请参阅设置会话属性。

userId

客户端应用程序用户的 ID。Amazon Lex 使用它来识别用户与您的机器人的对话。在运行时，每个请求都必须包含 userID 字段。

要决定用于您的应用程序的用户 ID，请考虑以下因素。

- userID 字段不得包含用户的任何个人身份信息，例如姓名、个人识别号或其他最终用户的个人信息。
- 如果您希望用户在一台设备上开始对话，然后在另一台设备上继续对话，请使用用户特定的标识符。
- 如果您希望同一个用户能够在两台不同的设备上进行两次独立的对话，请选择设备特定的标识符。
- 用户不能与同一个机器人的两个不同版本进行两个独立的对话。例如，用户无法与同一个机器人的 PROD 和 BETA 版本进行对话。如果您预计用户需要与两个不同的版本进行对话（例如，在测试期间），请在用户 ID 中包含机器人别名以将这两个对话分开。

长度限制：最小长度为 2。最大长度为 100。

模式：[0-9a-zA-Z._:-]+

必需：是

请求正文

请求接受以下二进制数据。

inputStream

用户以 PCM 或 Opus 音频格式或文本格式输入，如 Content-Type HTTP 标头中所述。

您可以将音频数据流式传输到 Amazon Lex，也可以创建本地缓冲区，在发送之前捕获所有音频数据。通常，如果您流式传输音频数据而不是在本地缓冲数据，则性能会更好。

必需：是

响应语法

```
HTTP/1.1 200
Content-Type: contentType
x-amz-lex-intent-name: intentName
x-amz-lex-nlu-intent-confidence: nluIntentConfidence
x-amz-lex-alternative-intents: alternativeIntents
x-amz-lex-slots: slots
x-amz-lex-session-attributes: sessionAttributes
x-amz-lex-sentiment: sentimentResponse
x-amz-lex-message: message
x-amz-lex-encoded-message: encodedMessage
x-amz-lex-message-format: messageFormat
x-amz-lex-dialog-state: dialogState
x-amz-lex-slot-to-elicit: slotToElicit
x-amz-lex-input-transcript: inputTranscript
x-amz-lex-encoded-input-transcript: encodedInputTranscript
x-amz-lex-bot-version: botVersion
x-amz-lex-session-id: sessionId
x-amz-lex-active-contexts: activeContexts

audioStream
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

响应将返回以下 HTTP 标头。

activeContexts

会话的活动上下文列表。可以在履行意图时设置上下文，也可以通过调用 PostContent、PostText 或 PutSession 操作来设置上下文。

您可以使用上下文来控制可以跟进意图的意图，也可以修改应用程序的操作。

alternativeIntents

一到四个可能适用于用户意图的替代意图。

每个替代意图都包括一个分数，该分数表明 Amazon Lex 对意图与用户意图相匹配的信心程度。意图按置信度分数排序。

botVersion

响应对话的机器人的版本。您可以使用此信息来帮助确定机器人的一个版本的性能是否优于另一个版本。

长度限制：长度下限为 1。长度上限为 64。

模式：`[0-9]+|\$LATEST`

contentType

请求的 `Accept` HTTP 标头中指定的内容类型。

dialogState

标识用户交互的当前状态。Amazon Lex 返回以下任一值作为 `dialogState`。客户可以选择使用此信息来自定义用户界面。

- `ElicitIntent` — Amazon Lex 想要引发的用户的意图。考虑以下示例：

  例如，用户可能会说出意图（“我想点披萨”）。如果 Amazon Lex 无法从此言语中推断出用户的意图，它将返回此对话状态。

- `ConfirmIntent` — Amazon Lex 预计会有“是”或“否”响应。

  例如，Amazon Lex 希望用户在履行意图之前进行确认。用户可能会响应其他信息，而不是简单的“是”或“否”响应。例如，“是，但要做个厚皮披萨”或“否，我想点一份饮品”。Amazon Lex 可以处理此类额外信息（在这些示例中，更新外壳类型槽或将意图从更改 OrderPizza 为 OrderDrink）。

- `ElicitSlot` — Amazon Lex 期望当前意图的插槽值。

  例如，假设 Amazon Lex 在响应中发送了以下消息：“您想要什么尺寸的披萨？”。用户可能会响应插槽值（例如，“中等”）。用户还可能在响应中提供其他信息（例如，“中厚皮披萨”）。Amazon Lex 可以适当地处理此类额外信息。

- `Fulfilled` — 表示 Lambda 函数已成功履行意图。
- `ReadyForFulfillment` — 表示客户必须满足请求。
- `Failed` — 表示与用户的对话失败。

  发生这种情况的原因可能多种多样，包括用户没有对服务提示做出适当的响应（您可以配置 Amazon Lex 可以提示用户输入特定信息的次数），或者 Lambda 函数无法履行意图。

有效值：ElicitIntent | ConfirmIntent | ElicitSlot | Fulfilled | ReadyForFulfillment | Failed

encodedInputTranscript

用于处理请求的文本。

如果输入为音频流，则 encodedInputTranscript 字段包含从该音频流中提取的文本。这是经过实际处理以识别目的和槽值的文本。您可以使用该信息来确定 Amazon Lex 是否正确处理了您发送的音频。

encodedInputTranscript 字段采用 base-64 编码。必须先解码该字段，然后才能使用该值。

encodedMessage

要传达给用户的消息。消息可以来自机器人的配置或 Lambda 函数。

如果未使用 Lambda 函数配置意图，或者如果 Lambda 函数在其响应中返回 Delegate 作为 dialogAction.type，则 Amazon Lex 会决定下一步操作，并根据当前的交互上下文从机器人的配置中选择相应的消息。例如，如果 Amazon Lex 无法理解用户输入，则会使用澄清提示消息。

创建意图时，您可以将消息分配给群组。将消息分配给群组后，Amazon Lex 会在响应中返回来自每个群组的一条消息。消息字段是一个包含消息的转义的 JSON 字符串。有关返回的 JSON 字符串结构的更多信息，请参阅受支持的消息格式。

如果 Lambda 函数返回消息，则 Amazon Lex 会在响应中将其传递给客户端。

encodedMessage 字段采用 base-64 编码。必须先解码该字段，然后才能使用该值。

长度限制：长度下限为 1。最大长度为 1366。

inputTranscript

此标头已被弃用。

用于处理请求的文本。

您只能在 de-DE、en-AU、en-GB、en-US、es-419、es-ES、es-US、fr-CA、fr-FR 和 it-IT 区域设置中使用此字段。在所有其他区域设置中，inputTranscript 字段均为空。您应改用 encodedInputTranscript 字段。

如果输入为音频流，则 inputTranscript 字段包含从该音频流中提取的文本。这是经过实际处理以识别目的和槽值的文本。您可以使用该信息来确定 Amazon Lex 是否正确处理了您发送的音频。

intentName

Amazon Lex 知道的当前用户意图。

message

此标头已被弃用。

您只能在 de-DE、en-AU、en-GB、en-US、es-419、es-ES、es-US、fr-CA、fr-FR 和 it-IT 区域设置中使用此字段。在所有其他区域设置中，message 字段均为空。您应改用 encodedMessage 字段。

要传达给用户的消息。消息可以来自机器人的配置或 Lambda 函数。

如果未使用 Lambda 函数配置意图，或者如果 Lambda 函数在其响应中返回 Delegate 作为 dialogAction.type，则 Amazon Lex 会决定下一步操作，并根据当前的交互上下文从机器人的配置中选择相应的消息。例如，如果 Amazon Lex 无法理解用户输入，则会使用澄清提示消息。

创建意图时，您可以将消息分配给群组。将消息分配给群组后，Amazon Lex 会在响应中返回来自每个群组的一条消息。消息字段是一个包含消息的转义的 JSON 字符串。有关返回的 JSON 字符串结构的更多信息，请参阅受支持的消息格式。

如果 Lambda 函数返回消息，则 Amazon Lex 会在响应中将其传递给客户端。

长度限制：长度下限为 1。最大长度为 1024。

messageFormat

响应消息的格式。下列值之一：

- PlainText — 消息包含 UTF-8 纯文本。
- CustomPayload — 消息是客户端的自定义格式。
- SSML — 消息包含为语音输出设置格式的文本。
- Composite — 消息包含一个转义的 JSON 对象，其中包含一条或多条来自创建意图时消息分配到的群组中的一条或多条消息。

有效值：PlainText | CustomPayload | SSML | Composite

nluIntentConfidence

提供一个分数，表示 Amazon Lex 对返回的意图是符合用户的意图的信心程度。分数必须介于 0.0 到 1.0 之间。

分数是相对分数，而不是绝对分数。根据对 Amazon Lex 的改进，分数可能会发生变化。

sentimentResponse

用言语表达的情绪。

当机器人配置为向 Amazon Comprehend 发送言语以进行情绪分析时，此字段将包含分析结果。

sessionAttributes

表示会话特定上下文信息的键值对的映射。

sessionId

会话的唯一标识符。

slots

Amazon Lex 在对话期间从用户输入中检测到的零个或零个以上意图插槽值（键值对）的映射。字段采用 base-64 编码。

Amazon Lex 会创建包含插槽可能值的列表。它返回的值由创建或更新插槽类型时 `valueSelectionStrategy` 所选的值决定。如果 `valueSelectionStrategy` 设置为 `ORIGINAL_VALUE`，并且用户值与插槽值相近，则返回用户提供的值。如果 `valueSelectionStrategy` 设置为 `TOP_RESOLUTION`，Amazon Lex 会返回解决方案列表中的第一个值，如果没有解决方案列表，则返回空值。如果不指定 `valueSelectionStrategy`，则默认值为 `ORIGINAL_VALUE`。

slotToElicit

如果 `dialogState` 值为 `ElicitSlot`，则返回 Amazon Lex 正在为其获取值的插槽的名称。

响应将以下内容作为 HTTP 正文返回。

audioStream

要向用户传达的提示（或语句）。这取决于机器人的配置和上下文。例如，如果 Amazon Lex 不了解用户意图，则会发送为机器人配置的 `clarificationPrompt`。如果意图需要在采取履行操作之前进行确认，则它会发送 `confirmationPrompt`。另一个示例：假设 Lambda 函数成功履行了意图，并向用户发送了一条消息。然后，Amazon Lex 在响应中发送了该消息。

错误

BadGatewayException

要么是 Amazon Lex 机器人仍在构建，要么其中一个依赖服务（Amazon Polly、AWS Lambda）因内部服务错误而失败。

HTTP 状态代码：502

BadRequestException

请求验证失败，上下文中没有可用的消息，或者机器人构建失败、仍在进行中或者包含未构建的更改。

HTTP 状态代码：400

ConflictException

两个客户端使用相同的 AWS 账户、Amazon Lex 机器人和用户 ID。

HTTP 状态代码：409

DependencyFailedException

其中一个依赖项（例如 AWS Lambda 或 Amazon Polly）引发了异常。例如，

- 如果 Amazon Lex 没有足够的权限来调用 Lambda 函数。
- 如果 Lambda 函数的执行时间超过 30 秒。
- 如果履行 Lambda 函数返回 `Delegate` 对话操作而不删除任何插槽值。

HTTP 状态代码：424

InternalFailureException

内部服务错误。重试调用。

HTTP 状态代码：500

LimitExceededException

已超出限制。

HTTP 状态代码：429

LoopDetectedException

不使用此异常。

HTTP 状态代码：508

NotAcceptableException

请求中的接受标头没有有效值。

HTTP 状态代码：406

NotFoundException

未找到所引用的资源（例如 Amazon Lex 机器人或别名）。

HTTP 状态代码：404

RequestTimeoutException

输入的语音太长。

HTTP 状态代码：408

UnsupportedMediaTypeException

内容类型标头 (PostContent API) 的值无效。

HTTP 状态代码：415

示例

示例 1

在此请求中，URI 标识了机器人 (Traffic)、机器人版本 ($LATEST) 和最终用户名 (someuser)。Content-Type 标头标识正文中音频的格式。Amazon Lex 还支持其他格式。如有必要，要将音频从一种格式转换为另一种格式，可以使用 SoX 开源软件。您可以通过添加 Accept HTTP 标头来指定获取响应的格式。

在响应中，x-amz-lex-message 标头显示了 Amazon Lex 返回的响应。然后，客户端可以向用户发送此响应。相同的消息通过分块编码（按要求）以 audio/MPEG 格式发送。

示例请求

```
"POST /bot/Traffic/alias/$LATEST/user/someuser/content HTTP/1.1[\r][\n]"
"x-amz-lex-session-attributes: eyJ1c2VyTmFtZSI6IkJvYiJ9[\r][\n]"
```

```
"Content-Type: audio/x-l16; channel-count=1; sample-rate=16000f[\r][\n]"
"Accept: audio/mpeg[\r][\n]"
"Host: runtime.lex.us-east-1.amazonaws.com[\r][\n]"
"Authorization: AWS4-HMAC-SHA256 Credential=BLANKED_OUT/20161230/us-east-1/lex/
aws4_request,
SignedHeaders=accept;content-type;host;x-amz-content-sha256;x-amz-date;x-amz-lex-
session-attributes,
 Signature=78ca5b54ea3f64a17ff7522de02cd90a9acd2365b45a9ce9b96ea105bb1c7ec2[\r][\n]"
"X-Amz-Date: 20161230T181426Z[\r][\n]"
"X-Amz-Content-Sha256:
 e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855[\r][\n]"
"Transfer-Encoding: chunked[\r][\n]"
"Connection: Keep-Alive[\r][\n]"
"User-Agent: Apache-HttpClient/4.5.x (Java/1.8.0_112)[\r][\n]"
"Accept-Encoding: gzip,deflate[\r][\n]"
"[\r][\n]"
"1000[\r][\n]"
"[0x7][0x0][0x7][0x0][\n]"
"[0x0][0x7][0x0][0xfc][0xff][\n]"
"[0x0][\n]"
…
```

示例响应

```
"HTTP/1.1 200 OK[\r][\n]"
"x-amzn-RequestId: cc8b34af-cebb-11e6-a35c-55f3a992f28d[\r][\n]"
"x-amz-lex-message: Sorry, can you repeat that?[\r][\n]"
"x-amz-lex-dialog-state: ElicitIntent[\r][\n]"
"x-amz-lex-session-attributes: eyJ1c2VyTmFtZSI6IkJvYiJ9[\r][\n]"
"Content-Type: audio/mpeg[\r][\n]"
"Transfer-Encoding: chunked[\r][\n]"
"Date: Fri, 30 Dec 2016 18:14:28 GMT[\r][\n]"
"[\r][\n]"
"2000[\r][\n]"
"ID3[0x4][0x0][0x0][0x0][0x0][0x0]#TSSE[0x0][0x0][0x0][0xf][0x0][0x0]
[0x3]Lavf57.41.100[0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0xff]
[0xf3]`[0xc4][0x0][0x1b]{[0x8d][0xe8][0x1]C[0x18][0x1][0x0]J[0xe0]`b[0xdd][0xd1]
[0xb][0xfd][0x11][0xdf][0xfe]";[0xbb][0xbb][0x9f][0xee][0xee][0xee][0xee]|DDD/[0xff]
[0xff][0xff][0xff]www?D[0xf7]w^?[0xff][0xfa]h[0x88][0x85][0xfe][0x88][0x88][0x88]
[[0xa2]'[0xff][0xfa]"{[0x9f][0xe8][0x88]]D[0xeb][0xbb][0xbb][0xa2]!u[0xfd][0xdd][0xdf]
[0x88][0x94][0x0]F[0xef][0xa1]8[0x0][0x82]w[0x88]N[0x0][0x0][0x9b][0xbb][0xe8][0xe
…
```

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

PostText

服务: Amazon Lex Runtime Service

将用户输入发送到 Amazon Lex。客户端应用程序可以使用此 API 在运行时向 Amazon Lex 发送请求。Amazon Lex 随后使用其为机器人构建的机器学习模型来解释用户输入。

在响应中，Amazon Lex 会返回要传达给用户的下一个 `message` 以及要显示的可选 `responseCard`。考虑以下示例消息：

- 对于用户输入 “我想要披萨”，Amazon Lex 可能会返回一条回复，其中包含一条引出槽位数据的消息（例如， PizzaSize）：“你想要什么尺寸的披萨？”
- 在用户提供所有披萨订单信息后，Amazon Lex 可能会返回一条消息以获取用户确认：“是否可以进到下披萨订单这一步？”。
- 用户对确认提示回答“是”后，Amazon Lex 可能会返回结论语句：“谢谢，已经为您的奶酪披萨下订单。”。

并非所有 Amazon Lex 消息都需要用户响应。例如，结论语句不需要响应。有些消息只需要“是”或“否”用户响应。除了 `message` 之外，Amazon Lex 还提供了有关响应中消息的其他上下文，您可以使用这些上下文来增强客户行为，例如显示相应的客户端用户界面。这些是响应中的 `slotToElicit`、`dialogState`、`intentName` 和 `slots` 字段。考虑以下示例：

- 如果消息是为了引发插槽数据，Amazon Lex 会返回以下上下文信息：
  - `dialogState`设置为 ElicitSlot
  - `intentName` 设置为当前上下文中的意图名称
  - `slotToElicit` 设置为 `message` 正在引发信息的插槽名称
  - `slots` 设置为具有当前已知值的为意图配置的插槽映射
- 如果消息是确认提示，`dialogState`则设置为 ConfirmIntent `SlotToElicit`且设置为空。
- 如果消息是表明用户意图未被理解的澄清提示（为意图配置），`dialogState`则设置为 ElicitIntent 且`slotToElicit`设置为空。

此外，Amazon Lex 还会返回您的应用程序特定的 `sessionAttributes`。有关更多信息，请参阅管理对话上下文。

请求语法

```
POST /bot/botName/alias/botAlias/user/userId/text HTTP/1.1
```

```
Content-type: application/json

{
   "activeContexts": [
      {
         "name": "string",
         "parameters": {
            "string" : "string"
         },
         "timeToLive": {
            "timeToLiveInSeconds": number,
            "turnsToLive": number
         }
      }
   ],
   "inputText": "string",
   "requestAttributes": {
      "string" : "string"
   },
   "sessionAttributes": {
      "string" : "string"
   }
}
```

URI 请求参数

请求使用以下 URI 参数。

botAlias

Amazon Lex 机器人的别名。

必需：是

botName

Amazon Lex 机器人的名称。

必需：是

userId

客户端应用程序用户的 ID。Amazon Lex 使用它来识别用户与您的机器人的对话。在运行时，每个请求都必须包含 `userID` 字段。

要决定用于您的应用程序的用户 ID，请考虑以下因素。

- `userID` 字段不得包含用户的任何个人身份信息，例如姓名、个人识别号或其他最终用户的个人信息。
- 如果您希望用户在一台设备上开始对话，然后在另一台设备上继续对话，请使用用户特定的标识符。
- 如果您希望同一个用户能够在两台不同的设备上进行两次独立的对话，请选择设备特定的标识符。
- 用户不能与同一个机器人的两个不同版本进行两个独立的对话。例如，用户无法与同一个机器人的 PROD 和 BETA 版本进行对话。如果您预计用户需要与两个不同的版本进行对话（例如，在测试期间），请在用户 ID 中包含机器人别名以将这两个对话分开。

长度限制：最小长度为 2。最大长度为 100。

模式：`[0-9a-zA-Z._:-]+`

必需：是

请求体

请求接受采用 JSON 格式的以下数据。

activeContexts

请求的活动上下文列表。可以在履行先前的意图时激活上下文，也可以通过在请求中包含上下文来激活上下文，

如果您未指定上下文列表，Amazon Lex 将使用会话的上下文的当前列表。如果您指定一个空列表，则会话的所有上下文都将被清除。

类型：ActiveContext 对象数组

数组成员：最少 0 个物品。最多 20 个项目。

必需：否

inputText

用户输入的文本（Amazon Lex 会解释此文本）。

当您使用 AWS CLI 时，您无法在 `--input-text` 参数中传递 URL。改为使用 `--cli-input-json` 参数传递 URL。

类型：字符串

长度限制：长度下限为 1。最大长度为 1024。

必需：是

requestAttributes

在 Amazon Lex 和客户端应用程序之间传递的特定于请求的信息。

命名空间 x-amz-lex: 保留供特殊属性使用。请勿创建带有 x-amz-lex: 前缀的任何请求属性。

有关更多信息，请参阅设置请求属性。

类型：字符串到字符串映射

必需：否

sessionAttributes

在 Amazon Lex 和客户端应用程序之间传递的特定于应用程序的信息。

有关更多信息，请参阅设置会话属性。

类型：字符串到字符串映射

必需：否

响应语法

```
HTTP/1.1 200
Content-type: application/json

{
   "activeContexts": [
      {
         "name": "string",
         "parameters": {
            "string" : "string"
         },
         "timeToLive": {
            "timeToLiveInSeconds": number,
            "turnsToLive": number
         }
      }
   ],
   "alternativeIntents": [
```

```
      {
         "intentName": "string",
         "nluIntentConfidence": {
            "score": number
         },
         "slots": {
            "string" : "string"
         }
      }
   ],
   "botVersion": "string",
   "dialogState": "string",
   "intentName": "string",
   "message": "string",
   "messageFormat": "string",
   "nluIntentConfidence": {
      "score": number
   },
   "responseCard": {
      "contentType": "string",
      "genericAttachments": [
         {
            "attachmentLinkUrl": "string",
            "buttons": [
               {
                  "text": "string",
                  "value": "string"
               }
            ],
            "imageUrl": "string",
            "subTitle": "string",
            "title": "string"
         }
      ],
      "version": "string"
   },
   "sentimentResponse": {
      "sentimentLabel": "string",
      "sentimentScore": "string"
   },
   "sessionAttributes": {
      "string" : "string"
   },
   "sessionId": "string",
```

```
   "slots": {
      "string" : "string"
   },
   "slotToElicit": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

activeContexts

会话的活动上下文列表。可以在履行意图时设置上下文，也可以通过调用 PostContent、PostText 或 PutSession 操作来设置上下文。

您可以使用上下文来控制可以跟进意图的意图，也可以修改应用程序的操作。

类型：ActiveContext 对象数组

数组成员：最少 0 个物品。最多 20 个项目。

alternativeIntents

一到四个可能适用于用户意图的替代意图。

每个替代意图都包括一个分数，该分数表明 Amazon Lex 对意图与用户意图相匹配的信心程度。意图按置信度分数排序。

类型：PredictedIntent 对象数组

数组成员：最多 4 项。

botVersion

响应对话的机器人的版本。您可以使用此信息来帮助确定机器人的一个版本的性能是否优于另一个版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：[0-9]+|\$LATEST

dialogState

标识用户交互的当前状态。Amazon Lex 返回以下任一值作为 `dialogState`。客户可以选择使用此信息来自定义用户界面。

- `ElicitIntent` — Amazon Lex 想要引发用户意图。

  例如，用户可能会说出意图（“我想点披萨”）。如果 Amazon Lex 无法从此言语中推断出用户的意图，它将返回此对话状态。

- `ConfirmIntent` — Amazon Lex 预计会有“是”或“否”响应。

  例如，Amazon Lex 希望用户在履行意图之前进行确认。

  用户可能会响应其他信息，而不是简单的“是”或“否”。例如，“是，但要做个厚皮披萨”或“否，我想点一份饮品”。Amazon Lex 可以处理此类额外信息（在这些示例中，更新外壳类型槽值，或者将意图从更改 OrderPizza 为 OrderDrink）。

- `ElicitSlot` — Amazon Lex 期望当前意图的插槽值。

  例如，假设 Amazon Lex 在响应中发送了以下消息：“您想要什么尺寸的披萨？”。用户可能会响应插槽值（例如，“中等”）。用户还可能在响应中提供其他信息（例如，“中厚皮披萨”）。Amazon Lex 可以适当地处理此类额外信息。

- `Fulfilled` — 表示为意图配置的 Lambda 函数已成功履行意图。
- `ReadyForFulfillment` — 传达客户端必须履行意图。
- `Failed` — 表示与用户的对话失败。

  发生这种情况的原因可能多种多样，包括用户没有对服务提示做出适当的响应（您可以配置 Amazon Lex 可以提示用户输入特定信息的次数），或者 Lambda 函数无法履行意图。

类型：字符串

有效值：`ElicitIntent | ConfirmIntent | ElicitSlot | Fulfilled | ReadyForFulfillment | Failed`

intentName

Amazon Lex 知道的当前用户意图。

类型：字符串

message

要传达给用户的消息。消息可以来自机器人的配置或 Lambda 函数。

如果未使用 Lambda 函数配置意图，或者如果 Lambda 函数在其响应中返回 `Delegate` 作为 `dialogAction.type`，则 Amazon Lex 会决定下一步的操作方案，并根据当前的交互上下文从机器人的配置中选择相应的消息。例如，如果 Amazon Lex 无法理解用户输入，则会使用澄清提示消息。

创建意图时，您可以将消息分配给群组。将消息分配给群组后，Amazon Lex 会在响应中返回来自每个群组的一条消息。消息字段是一个包含消息的转义的 JSON 字符串。有关返回的 JSON 字符串结构的更多信息，请参阅受支持的消息格式。

如果 Lambda 函数返回消息，则 Amazon Lex 会在响应中将其传递给客户端。

类型：字符串

长度限制：长度下限为 1。最大长度为 1024。

messageFormat

响应消息的格式。下列值之一：

- `PlainText` — 消息包含 UTF-8 纯文本。
- `CustomPayload` — 消息是由 Lambda 函数定义的自定义格式。
- `SSML` — 消息包含为语音输出设置格式的文本。
- `Composite` — 消息包含一个转义的 JSON 对象，其中包含一条或多条来自创建意图时消息分配到的群组中的一条或多条消息。

类型：字符串

有效值：`PlainText | CustomPayload | SSML | Composite`

nluIntentConfidence

提供一个分数，表示 Amazon Lex 对返回的意图是符合用户的意图的信心程度。分数必须介于 0.0 到 1.0 之间。有关更多信息，请参阅置信度分数。

分数是相对分数，而不是绝对分数。根据对 Amazon Lex 的改进，分数可能会发生变化。

类型：IntentConfidence 对象

responseCard

表示用户响应当前提示时必须使用的选项。响应卡可以来自机器人配置（在 Amazon Lex 控制台中，选择插槽旁边的设置按钮）或代码挂钩（Lambda 函数）。

类型：ResponseCard 对象

sentimentResponse

用言语表达的情绪。

当机器人配置为向 Amazon Comprehend 发送言语以进行情绪分析时，此字段将包含分析结果。

类型：SentimentResponse 对象

sessionAttributes

表示会话特定上下文信息的键值对的映射。

类型：字符串到字符串映射

sessionId

会话的唯一标识符。

类型：字符串

slots

Amazon Lex 从用户在对话中输入的内容中检测到的意图插槽。

Amazon Lex 会创建包含插槽可能值的列表。它返回的值由创建或更新插槽类型时 `valueSelectionStrategy` 所选的值决定。如果 `valueSelectionStrategy` 设置为 `ORIGINAL_VALUE`，并且用户值与插槽值相近，则返回用户提供的值。如果 `valueSelectionStrategy` 设置为 `TOP_RESOLUTION`，Amazon Lex 会返回解决方案列表中的第一个值，如果没有解决方案列表，则返回空值。如果不指定 `valueSelectionStrategy`，则默认值为 `ORIGINAL_VALUE`。

类型：字符串到字符串映射

slotToElicit

如果 `dialogState` 值为 `ElicitSlot`，则返回 Amazon Lex 正在为其获取值的插槽的名称。

类型：字符串

错误

BadGatewayException

要么是 Amazon Lex 机器人仍在构建，要么其中一个依赖服务（Amazon Polly、AWS Lambda）因内部服务错误而失败。

HTTP 状态代码：502

BadRequestException

请求验证失败，上下文中没有可用的消息，或者机器人构建失败、仍在进行中或者包含未构建的更改。

HTTP 状态代码：400

ConflictException

两个客户端使用相同的 AWS 账户、Amazon Lex 机器人和用户 ID。

HTTP 状态代码：409

DependencyFailedException

其中一个依赖项（例如 AWS Lambda 或 Amazon Polly）引发了异常。例如，

- 如果 Amazon Lex 没有足够的权限来调用 Lambda 函数。
- 如果 Lambda 函数的执行时间超过 30 秒。
- 如果履行 Lambda 函数返回 `Delegate` 对话操作而不删除任何插槽值。

HTTP 状态代码：424

InternalFailureException

内部服务错误。重试调用。

HTTP 状态代码：500

LimitExceededException

已超出限制。

HTTP 状态代码：429

LoopDetectedException

不使用此异常。

HTTP 状态代码：508

NotFoundException

未找到所引用的资源（例如 Amazon Lex 机器人或别名）。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

PutSession

服务: Amazon Lex Runtime Service

使用 Amazon Lex 自动程序创建新会话或修改现有会话。使用此操作使您的应用程序能够设置机器人的状态。

有关更多信息，请参阅管理会话。

请求语法

```
POST /bot/botName/alias/botAlias/user/userId/session HTTP/1.1
Accept: accept
Content-type: application/json

{
   "activeContexts": [
      {
         "name": "string",
         "parameters": {
            "string" : "string"
         },
         "timeToLive": {
            "timeToLiveInSeconds": number,
            "turnsToLive": number
         }
      }
   ],
   "dialogAction": {
      "fulfillmentState": "string",
      "intentName": "string",
      "message": "string",
      "messageFormat": "string",
      "slots": {
         "string" : "string"
      },
      "slotToElicit": "string",
      "type": "string"
   },
   "recentIntentSummaryView": [
      {
         "checkpointLabel": "string",
         "confirmationStatus": "string",
         "dialogActionType": "string",
```

```
        "fulfillmentState": "string",
        "intentName": "string",
        "slots": {
           "string" : "string"
        },
        "slotToElicit": "string"
     }
  ],
  "sessionAttributes": {
     "string" : "string"
  }
}
```

URI 请求参数

请求使用以下 URI 参数。

accept

Amazon Lex 在响应中返回的消息可以是文本，也可以是语音，取决于此字段的值。

- 如果值为 `text/plain; charset=utf-8`，则 Amazon Lex 会在响应中返回文本。
- 如果值以 `audio/` 开头，则 Amazon Lex 会在响应中返回语音。Amazon Lex 使用 Amazon Polly 按照您指定的配置生成语音。例如，如果您将 `audio/mpeg` 指定为值，Amazon Lex 将返回 MPEG 格式的语音。
- 如果值为 `audio/pcm`，则返回的语音采用 16 位小端序格式的 `audio/pcm`。
- 以下是接受的值：
  - `audio/mpeg`
  - `audio/ogg`
  - `audio/pcm`
  - `audio/*`（默认值为 mpeg）
  - `text/plain; charset=utf-8`

botAlias

包含会话数据的机器人的使用中别名。

必需：是

botName

包含会话数据的机器人的名称。

必需：是

userId

客户端应用程序用户的 ID。Amazon Lex 使用它来识别用户与您的机器人的对话。

长度限制：最小长度为 2。最大长度为 100。

模式：`[0-9a-zA-Z._:-]+`

必需：是

请求体

请求接受采用 JSON 格式的以下数据。

activeContexts

请求的活动上下文列表。可以在履行先前的意图时激活上下文，也可以通过在请求中包含上下文来激活上下文，

如果您未指定上下文列表，Amazon Lex 将使用会话的上下文的当前列表。如果您指定一个空列表，则会话的所有上下文都将被清除。

类型：ActiveContext 对象数组

数组成员：最少 0 个物品。最多 20 个项目。

必需：否

dialogAction

设置机器人为完成对话而应采取的下一个操作。

类型：DialogAction 对象

必需：否

recentIntentSummaryView

该机器人最近的意图摘要。您可以使用意图摘要视图在意图上设置检查点标签并修改意图的属性。您也可以使用它来移除意图摘要对象或向列表中添加意图摘要对象。

您修改或添加到列表中的意图必须对机器人有意义。例如，意图名称对于机器人而言必须有效。您必须为以下项提供值：

- intentName
- 插槽名称
- slotToElict

如果您在 PutSession 请求中发送 recentIntentSummaryView 参数，则新摘要视图的内容将替换旧的摘要视图。例如，如果 GetSession 请求在摘要视图中返回三个意图，而您在摘要视图中使用一个意图呼叫 PutSession，则对 GetSession 的下一个调用将只返回一个意图。

类型：IntentSummary 对象数组

数组成员：最少 0 个物品。最多 3 项。

必需：否

sessionAttributes

表示会话特定上下文信息的键值对的映射。它包含在 Amazon Lex 与客户端应用程序之间传递的应用程序信息。

类型：字符串到字符串映射

必需：否

响应语法

```
HTTP/1.1 200
Content-Type: contentType
x-amz-lex-intent-name: intentName
x-amz-lex-slots: slots
x-amz-lex-session-attributes: sessionAttributes
x-amz-lex-message: message
x-amz-lex-encoded-message: encodedMessage
x-amz-lex-message-format: messageFormat
x-amz-lex-dialog-state: dialogState
x-amz-lex-slot-to-elicit: slotToElicit
x-amz-lex-session-id: sessionId
x-amz-lex-active-contexts: activeContexts

audioStream
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

响应将返回以下 HTTP 标头。

activeContexts

会话的活动上下文列表。

contentType

请求的 Accept HTTP 标头中指定的内容类型。

dialogState

- ConfirmIntent — Amazon Lex 预计在履行意图之前会有“是”或“否”响应来确认意图。
- ElicitIntent — Amazon Lex 想要引发的用户的意图。
- ElicitSlot — Amazon Lex 期望当前意图的插槽值。
- Failed — 传达与用户的对话失败。发生这种情况的原因可能多种多样，包括用户没有对服务提示做出适当的响应，或者 Lambda 函数无法履行意图。
- Fulfilled — 表示 Lambda 函数已成功履行意图。
- ReadyForFulfillment — 传达客户端必须履行意图。

有效值：ElicitIntent | ConfirmIntent | ElicitSlot | Fulfilled | ReadyForFulfillment | Failed

encodedMessage

应向用户显示的下一条消息。

encodedMessage 字段采用 base-64 编码。必须先解码该字段，然后才能使用该值。

长度限制：长度下限为 1。最大长度为 1366。

intentName

当前意图的名称。

message

此标头已被弃用。

应向用户显示的下一条消息。

您只能在 de-DE、en-AU、en-GB、en-US、es-419、es-ES、es-US、fr-CA、fr-FR 和 it-IT 区域设置中使用此字段。在所有其他区域设置中，message 字段均为空。您应改用 encodedMessage 字段。

长度限制：长度下限为 1。最大长度为 1024。

messageFormat

响应消息的格式。下列值之一：

- PlainText — 消息包含 UTF-8 纯文本。
- CustomPayload — 消息是客户端的自定义格式。
- SSML — 消息包含为语音输出设置格式的文本。
- Composite — 消息包含一个转义的 JSON 对象，其中包含一条或多条来自创建意图时消息分配到的群组中的一条或多条消息。

有效值：PlainText | CustomPayload | SSML | Composite

sessionAttributes

表示会话特定上下文信息的键值对的映射。

sessionId

会话的唯一标识符。

slots

Amazon Lex 在对话期间从用户输入中检测到的零个或零个以上意图插槽值的映射。

Amazon Lex 会创建包含插槽可能值的列表。它返回的值由创建或更新插槽类型时 valueSelectionStrategy 所选的值决定。如果 valueSelectionStrategy 设置为 ORIGINAL_VALUE，并且用户值与插槽值相近，则返回用户提供的值。如果 valueSelectionStrategy 设置为 TOP_RESOLUTION，Amazon Lex 会返回解决方案列表中的第一个值，如果没有解决方案列表，则返回空值。如果不指定 valueSelectionStrategy，则默认值为 ORIGINAL_VALUE。

slotToElicit

如果 dialogState 为 ElicitSlot，则返回 Amazon Lex 正在为其获取值的插槽的名称。

响应将以下内容作为 HTTP 正文返回。

audioStream

要传达给用户的消息的音频版本。

错误

BadGatewayException

要么是 Amazon Lex 机器人仍在构建，要么其中一个依赖服务（Amazon Polly、AWS Lambda）因内部服务错误而失败。

HTTP 状态代码：502

BadRequestException

请求验证失败，上下文中没有可用的消息，或者机器人构建失败、仍在进行中或者包含未构建的更改。

HTTP 状态代码：400

ConflictException

两个客户端使用相同的 AWS 账户、Amazon Lex 机器人和用户 ID。

HTTP 状态代码：409

DependencyFailedException

其中一个依赖项（例如 AWS Lambda 或 Amazon Polly）引发了异常。例如，

- 如果 Amazon Lex 没有足够的权限来调用 Lambda 函数。
- 如果 Lambda 函数的执行时间超过 30 秒。
- 如果履行 Lambda 函数返回 `Delegate` 对话操作而不删除任何插槽值。

HTTP 状态代码：424

InternalFailureException

内部服务错误。重试调用。

HTTP 状态代码：500

LimitExceededException

已超出限制。

HTTP 状态代码：429

NotAcceptableException

请求中的接受标头没有有效值。

HTTP 状态代码：406

NotFoundException

未找到所引用的资源（例如 Amazon Lex 机器人或别名）。

HTTP 状态代码：404

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 命令行界面
- AWS 适用于 .NET 的 SDK
- AWS 适用于 C++ 的 SDK
- AWS 适用于 Go v2 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS JavaScript V3 版软件开发工具包
- AWS 适用于 Kotlin 的 SDK
- AWS 适用于 PHP 的 SDK V3
- AWS Python 软件开发工具包
- AWS 适用于 Ruby V3 的 SDK

数据类型

Amazon Lex 模型构建服务支持以下数据类型：

- BotAliasMetadata
- BotChannelAssociation
- BotMetadata
- BuiltinIntentMetadata

- BuiltinIntentSlot
- BuiltinSlotTypeMetadata
- CodeHook
- ConversationLogsRequest
- ConversationLogsResponse
- EnumerationValue
- FollowUpPrompt
- FulfillmentActivity
- InputContext
- Intent
- IntentMetadata
- KendraConfiguration
- LogSettingsRequest
- LogSettingsResponse
- Message
- MigrationAlert
- MigrationSummary
- OutputContext
- Prompt
- ResourceReference
- Slot
- SlotDefaultValue
- SlotDefaultValueSpec
- SlotTypeConfiguration
- SlotTypeMetadata
- SlotTypeRegexConfiguration
- Statement
- Tag
- UtteranceData
- UtteranceList

Amazon Lex 运行时服务支持以下数据类型：

- ActiveContext
- ActiveContextTimeToLive
- Button
- DialogAction
- GenericAttachment
- IntentConfidence
- IntentSummary
- PredictedIntent
- ResponseCard
- SentimentResponse

Amazon Lex 模型构建服务

Amazon Lex 模型构建服务支持以下数据类型：

- BotAliasMetadata
- BotChannelAssociation
- BotMetadata
- BuiltinIntentMetadata
- BuiltinIntentSlot
- BuiltinSlotTypeMetadata
- CodeHook
- ConversationLogsRequest
- ConversationLogsResponse
- EnumerationValue
- FollowUpPrompt
- FulfillmentActivity
- InputContext
- Intent
- IntentMetadata

- KendraConfiguration
- LogSettingsRequest
- LogSettingsResponse
- Message
- MigrationAlert
- MigrationSummary
- OutputContext
- Prompt
- ResourceReference
- Slot
- SlotDefaultValue
- SlotDefaultValueSpec
- SlotTypeConfiguration
- SlotTypeMetadata
- SlotTypeRegexConfiguration
- Statement
- Tag
- UtteranceData
- UtteranceList

BotAliasMetadata

服务: Amazon Lex Model Building Service

提供有关机器人别名的信息。

内容

botName

别名指向的自动程序的名称。

类型：字符串

长度限制：最小长度为 2。最大长度为 50。

模式：`^([A-Za-z]_?)+$`

必需：否

botVersion

别名指向的 Amazon Lex 机器人的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：`\$LATEST|[0-9]+`

必需：否

checksum

机器人别名的校验和。

类型：字符串

必需：否

conversationLogs

确定 Amazon Lex 如何使用对话日志作为别名的设置。

类型：ConversationLogsResponse 对象

必需：否

createdDate

机器人别名的创建日期。

类型：时间戳

必需：否

description

机器人别名的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

必需：否

lastUpdatedDate

机器人别名的更新日期。创建资源时，创建日期和上次更新日期相同。

类型：时间戳

必需：否

name

自动程序别名的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2

- AWS 适用于 Ruby V3 的 SDK

BotChannelAssociation

服务: Amazon Lex Model Building Service

表示 Amazon Lex 机器人与外部消息收发平台之间的关联。

内容

botAlias

指向要与之建立此关联的 Amazon Lex 机器人的特定版本的别名。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

必需：否

botConfiguration

提供与消息收发平台通信所需的信息。

类型：字符串到字符串映射

映射条目：最多 10 项。

必需：否

botName

要与之建立此关联的 Amazon Lex 机器人的名称。

> **Note**
>
> 目前，Amazon Lex 支持与 Facebook、Slack 和 Twilio 的关联。

类型：字符串

长度限制：最小长度为 2。最大长度为 50。

模式：`^([A-Za-z]_?)+$`

必需：否

createdDate

Amazon Lex 机器人与通道之间建立关联的日期。

类型：时间戳

必需：否

description

您正在创建的关联的文本描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

必需：否

failureReason

如果 `status` 是 `FAILED`，则 Amazon Lex 会提供其未能创建关联的原因。

类型：字符串

必需：否

name

机器人与通道之间的关联的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

必需：否

status

机器人通道的状态。

- `CREATED` — 通道已创建，随时可用。
- `IN_PROGRESS` — 通道正在创建。
- `FAILED` — 创建通道时出错。有关失败原因的信息，请参阅 `failureReason` 字段。

类型：字符串

有效值：`IN_PROGRESS | CREATED | FAILED`

必需：否

type

通过指明 Amazon Lex 机器人和外部消息收发平台之间建立的通道类型，指定关联的类型。

类型：字符串

有效值：`Facebook | Slack | Twilio-Sms | Kik`

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

BotMetadata

服务: Amazon Lex Model Building Service

提供有关机器人的信息。

内容

createdDate

机器人的创建日期。

类型：时间戳

必需：否

description

机器人的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

必需：否

lastUpdatedDate

机器人的更新日期。创建机器人时，创建日期和上次更新日期相同。

类型：时间戳

必需：否

name

机器人的名称。

类型：字符串

长度限制：最小长度为 2。最大长度为 50。

模式：`^([A-Za-z]_?)+$`

必需：否

status

机器人的状态。

类型：字符串

有效值：BUILDING | READY | READY_BASIC_TESTING | FAILED | NOT_BUILT

必需：否

version

自动程序的版本。对于新机器人，版本始终是 $LATEST。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

BuiltinIntentMetadata

服务: Amazon Lex Model Building Service

提供内置意图的元数据。

内容

signature

内置意图的唯一标识符。要查找意图的签名，请参阅 Alexa Skills Kit 中的标准内置意图。

类型：字符串

必需：否

supportedLocales

意图支持的区域设置的标识符列表。

类型：字符串数组

有效值：`de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

BuiltinIntentSlot

服务: Amazon Lex Model Building Service

提供有关在内置意图中使用的插槽的信息。

内容

name

为意图定义的插槽列表。

类型：字符串

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

BuiltinSlotTypeMetadata

服务: Amazon Lex Model Building Service

提供有关内置插槽类型的信息。

内容

signature

内置插槽类型的唯一标识符。要查找插槽类型的签名，请参阅 Alexa Skills Kit 中的插槽类型参考。

类型：字符串

必需：否

supportedLocales

插槽的目标区域设置列表。

类型：字符串数组

有效值：`de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR`

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

CodeHook

服务: Amazon Lex Model Building Service

指定 Lambda 函数，其用于验证对机器人的请求，或履行用户向机器人发出的请求。

内容

messageVersion

您想要 Amazon Lex 用来调用您的 Lambda 函数的请求-响应版本。有关更多信息，请参阅 使用 Lambda 函数。

类型：字符串

长度限制：长度下限为 1。最大长度为 5。

必需：是

uri

Lambda 函数的 Amazon 资源名称（ARN）。

类型：字符串

长度约束：最小长度为 20。最大长度为 2048。

模式：`arn:aws[a-zA-Z-]*:lambda:[a-z]+-[a-z]+(-[a-z]+)*-[0-9]:[0-9]{12}:function:[a-zA-Z0-9-_]+(\/[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12})?(:[a-zA-Z0-9-_]+)?`

必需：是

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

ConversationLogsRequest

服务: Amazon Lex Model Building Service

提供对话日志所需的设置。

内容

iamRoleArn

一个 IAM 角色的 Amazon 资源名称 (ARN)，该角色有权写入您的 CloudWatch 日志以获取文本日志，并有权写入您的 S3 存储桶（用于存放音频日志）。如果启用了音频加密，则此角色还提供针对用于加密音频日志的 AWS KMS 密钥的访问权限。有关更多信息，请参阅为对话日志创建 IAM 角色和策略。

类型：字符串

长度约束：最小长度为 20。最大长度为 2048。

模式：`^arn:[\w\-]+:iam::[\d]{12}:role/.+$`

必需：是

logSettings

对话日志的设置。您可以记录对话文本和/或对话音频。

类型：LogSettingsRequest 对象数组

必需：是

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

ConversationLogsResponse

服务: Amazon Lex Model Building Service

包含有关对话日志设置的信息。

内容

iamRoleArn

用于将日志写入 CloudWatch 日志或 S3 存储桶的 IAM 角色的 Amazon 资源名称 (ARN)。

类型：字符串

长度约束：最小长度为 20。最大长度为 2048。

模式：`^arn:[\w\-]+:iam::[\d]{12}:role/.+$`

必需：否

logSettings

对话日志的设置。您可以记录文本和/或音频。

类型：LogSettingsResponse 对象数组

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

EnumerationValue

服务: Amazon Lex Model Building Service

每种槽类型可以有一组值。每个枚举值表示插槽类型可采用的值。

例如，订购披萨机器人可能有一个插槽类型，用于指定披萨应有的饼皮类型。插槽类型可以包含值

- 厚
- thin
- 装馅

内容

value

插槽类型的值。

类型：字符串

长度限制：最小长度为 1。长度上限为 140。

必需：是

synonyms

与插槽类型值相关的额外值。

类型：字符串数组

长度限制：最小长度为 1。长度上限为 140。

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

FollowUpPrompt

服务: Amazon Lex Model Building Service

履行意图后提示是否要进行其他活动。例如，在履行 `OrderPizza` 意图后，您可以提示用户以了解用户是否想要订购饮品。

内容

prompt

提示用户提供信息。

类型：Prompt 对象

必需：是

rejectionStatement

当用户对 `prompt` 字段中定义的问题回答“否”时，Amazon Lex 将使用此语句进行响应，以确认意图已被取消。

类型：Statement 对象

必需：是

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

FulfillmentActivity

服务: Amazon Lex Model Building Service

描述用户提供意图所需的所有信息后如何履行意图。您可以提供 Lambda 函数来处理意图，也可以将意图信息返回给客户端应用程序。我们建议您使用 Lambda 函数，以便相关逻辑存在于云中，并将客户端代码主要限于演示。如果您需要更新逻辑，则只需更新 Lambda 函数；无需升级您的客户端应用程序。

考虑以下示例：

- 在订购披萨应用程序中，在用户提供所有下单信息后，您可以使用 Lambda 函数向披萨店下订单。
- 在游戏应用程序中，当用户说“捡起石头”时，这些信息必须返回到客户端应用程序，这样它才能执行操作并更新图形。在这种情况下，您希望 Amazon Lex 将意图数据返回给客户。

内容

type

应如何通过运行 Lambda 函数或将插槽数据返回到客户端应用程序来履行意图。

类型：字符串

有效值：`ReturnIntent | CodeHook`

必需：是

codeHook

对为履行意图而运行的 Lambda 函数的描述。

类型：CodeHook 对象

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

InputContext

服务: Amazon Lex Model Building Service

上下文的名称，该上下文必须对 Amazon Lex 选择的意图保持活动状态。

内容

name

上下文的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

必需：是

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

Intent

服务: Amazon Lex Model Building Service

标识意图的特定版本。

内容

intentName

意图的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

必需：是

intentVersion

意图的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：`\$LATEST|[0-9]+`

必需：是

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

IntentMetadata

服务: Amazon Lex Model Building Service

提供有关意图的信息。

内容

createdDate

意图的创建日期。

类型：时间戳

必需：否

description

目的的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

必需：否

lastUpdatedDate

意图的更新日期。创建意图时，创建日期和上次更新日期相同。

类型：时间戳

必需：否

name

意图的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

必需：否

version

意图的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：`\$LATEST|[0-9]+`

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

KendraConfiguration

服务: Amazon Lex Model Building Service

提供亚马逊的配置信息。 KendraSearchIntent意图。当您使用此意图时，Amazon Lex 会搜索特定的 Amazon Kendra 索引，并从与用户表达匹配的索引返回文档。有关更多信息，请参阅亚马逊。KendraSearchIntent。

内容

kendraIndex

你想要亚马逊的亚马逊 Kendra 索引的亚马逊资源名称 (ARN)。 KendraSearchIntent 打算搜寻。索引必须与 Amazon Lex 机器人位于相同的账户和区域中。如果 Amazon Kendra 索引不存在，则在调用 PutIntent 操作时会出现异常。

类型：字符串

长度约束：最小长度为 20。最大长度为 2048。

模式：`arn:aws:kendra:[a-z]+-[a-z]+-[0-9]:[0-9]{12}:index\/[a-zA-Z0-9][a-zA-Z0-9_-]*`

必需：是

role

具有搜索 Amazon Kendra 索引权限的 IAM 角色的 Amazon 资源名称 (ARN)。该角色必须与 Amazon Lex 机器人位于相同的账户和区域中。如果角色不存在，则在调用 PutIntent 操作时会出现异常。

类型：字符串

长度约束：最小长度为 20。最大长度为 2048。

模式：`arn:aws:iam::[0-9]{12}:role/.*`

必需：是

queryFilterString

Amazon Lex 发送给 Amazon Kendra 以筛选查询响应的查询筛选条件。筛选条件采用 Amazon Kendra 定义的格式。有关更多信息，请参阅筛选查询。

您可以在运行时用新的筛选字符串覆盖此筛选器字符串。

类型：字符串

长度约束：最小长度为 0。

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

LogSettingsRequest

服务: Amazon Lex Model Building Service

用于配置对话日志的传送模式和目标的设置。

内容

destination

将日志传送到哪里。文本日志被传送到 CloudWatch 日志日志组。音频日志将传输到 S3 存储桶

类型：字符串

有效值：`CLOUDWATCH_LOGS | S3`

必需：是

logType

要启用的日志类型。文本日志被传送到 CloudWatch日志日志组。音频日志将传输到 S3 存储桶

类型：字符串

有效值：`AUDIO | TEXT`

必需：是

resourceArn

应将日志传送到的 CloudWatch 日志组或 S3 存储桶的 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

模式：`^arn:[\w\-]+:(?:logs:[\w\-]+:[\d]{12}:log-group:[\.\-_/#A-Za-z0-9]{1,512}(?::\*)?|s3:::[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9])$`

必需：是

kmsKeyArn

AWS KMS 客户托管式密钥的 Amazon 资源名称 (ARN)，用于对传输到 S3 存储桶中的音频日志进行加密的 Amazon 资源名称 (ARN)。该密钥不适用于 CloudWatch 日志，对于 S3 存储桶是可选的。

类型：字符串

长度约束：最小长度为 20。最大长度为 2048。

模式：`^arn:[\w\-]+:kms:[\w\-]+:[\d]{12}:(?:key\/[\w\-]+|alias\/[a-zA-Z0-9:\/_\-]{1,256})$`

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

LogSettingsResponse

服务: Amazon Lex Model Building Service

对话日志的设置。

内容

destination

日志的传送目的地。

类型：字符串

有效值：`CLOUDWATCH_LOGS | S3`

必需：否

kmsKeyArn

用于对存储在 S3 存储桶中的音频日志进行加密的密钥的 Amazon 资源名称（ARN）。

类型：字符串

长度约束：最小长度为 20。最大长度为 2048。

模式：`^arn:[\w\-]+:kms:[\w\-]+:[\d]{12}:(?:key\/[\w\-]+|alias\/[a-zA-Z0-9:\/_\-]{1,256})$`

必需：否

logType

启用的日志记录类型。

类型：字符串

有效值：`AUDIO | TEXT`

必需：否

resourceArn

发送日志的 CloudWatch 日志组或 S3 存储桶的 Amazon 资源名称 (ARN)。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

模式：`^arn:[\w\-]+:(?:logs:[\w\-]+:[\d]{12}:log-group:[\.\-_/#A-Za-z0-9]{1,512}(?::\*)?|s3:::[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9])$`

必需：否

resourcePrefix

资源前缀是您指定用于包含音频日志的 S3 存储桶中的 S3 对象键的第一部分。对于 CloudWatch 日志，它是您指定的日志组中日志流名称的前缀。

类型：字符串

长度限制：最大长度为 1024。

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

Message

服务: Amazon Lex Model Building Service

提供消息文本及其类型的消息对象。

内容

content

消息的文本。

类型：字符串

长度限制：长度下限为 1。最大长度为 1000。

必需：是

contentType

消息字符串的内容类型。

类型：字符串

有效值：`PlainText | SSML | CustomPayload`

必需：是

groupNumber

标识消息所属的消息组。将群组分配给消息后，Amazon Lex 会在响应中返回来自每个群组的一条消息。

类型：整数

有效范围：最小值为 1。最大值为 5。

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2

- AWS 适用于 Ruby V3 的 SDK

MigrationAlert

服务: Amazon Lex Model Building Service

提供有关 Amazon Lex 在迁移期间发送的警报和警告的信息。这些警报包含有关如何解决该问题的信息。

内容

details

有关该警报的其他详细信息。

类型：字符串数组

必需：否

message

描述发出警报的原因的消息。

类型：字符串

必需：否

referenceURLs

指向描述如何解决警报的 Amazon Lex 文档的链接。

类型：字符串数组

必需：否

type

错误的类型。有两种警报：

- ERROR — 迁移存在无法解决的问题。迁移将会停止。
- WARN — 迁移存在问题，需要手动更改新的 Amazon Lex V2 机器人。迁移仍在继续。

类型：字符串

有效值：ERROR | WARN

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

MigrationSummary

服务: Amazon Lex Model Building Service

提供有关将机器人从 Amazon Lex V1 迁移到 Amazon Lex V2 的信息。

内容

migrationId

Amazon Lex 为迁移分配的唯一标识符。

类型：字符串

长度限制：固定长度为 10。

模式：^[0-9a-zA-Z]+$

必需：否

migrationStatus

操作的状态。当状态为 COMPLETE 时，机器人在 Amazon Lex V2 中可用。可能需要解决警报和警告才能完成迁移。

类型：字符串

有效值：IN_PROGRESS | COMPLETED | FAILED

必需：否

migrationStrategy

用于进行迁移的策略。

类型：字符串

有效值：CREATE_NEW | UPDATE_EXISTING

必需：否

migrationTimestamp

启动迁移的日期和时间。

类型：时间戳

必需：否

v1BotLocale

作为迁移来源的 Amazon Lex V1 机器人的区域设置。

类型：字符串

有效值：de-DE | en-AU | en-GB | en-IN | en-US | es-419 | es-ES | es-US | fr-FR | fr-CA | it-IT | ja-JP | ko-KR

必需：否

v1BotName

作为迁移来源的 Amazon Lex V1 机器人的名称。

类型：字符串

长度限制：最小长度为 2。最大长度为 50。

模式：^([A-Za-z]_?)+$

必需：否

v1BotVersion

作为迁移来源的 Amazon Lex V1 机器人的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

必需：否

v2BotId

作为迁移目标的 Amazon Lex V2 的唯一标识符。

类型：字符串

长度限制：固定长度为 10。

模式：^[0-9a-zA-Z]+$

必需：否

v2BotRole

Amazon Lex 用来运行 Amazon Lex V2 机器人的 IAM 角色。

类型：字符串

长度约束：最小长度为 20。最大长度为 2048。

模式：`^arn:[\w\-]+:iam::[\d]{12}:role/.+$`

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

OutputContext

服务: Amazon Lex Model Building Service

当履行意图时设置的输出上下文的规范。

内容

name

上下文的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

必需：是

timeToLiveInSeconds

在 PostContent 或 PostText 响应中首次发送上下文后，上下文应处于活动状态的秒数。您可以设置 5 到 86,400 秒（24 小时）之间的值。

类型：整数

有效范围：最小值为 5。最大值为 86400。

必需：是

turnsToLive

上下文应处于活动状态的对话回合次数。对话回合是指一个 PostContent 或 PostText 请求以及来自 Amazon Lex 的相应响应。

类型：整数

有效范围：最小值为 1。最大值为 20。

必需：是

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

Prompt

服务: Amazon Lex Model Building Service

从用户那里获取信息。要定义提示，请提供一条或多条消息，并指定尝试从用户那里获取信息的次数。如果您提供多条消息，Amazon Lex 会选择其中一条消息来提示用户。有关更多信息，请参阅 Amazon Lex：工作原理。

内容

maxAttempts

提示用户输入信息的次数。

类型：整数

有效范围：最小值为 1。最大值为 5。

必需：是

messages

对象的数组，每个对象均提供消息字符串及其类型。可以采用纯文本或语音合成标记语言 (SSML) 指定消息字符串。

类型：Message 对象数组

数组成员：最少 1 个物品。最多 15 项。

必需：是

responseCard

响应卡。在运行时，Amazon Lex 在 `PostText` API 响应中使用此提示。它使用会话属性和插槽值替换响应卡中的占位符。有关更多信息，请参阅 使用响应卡。

类型：字符串

长度限制：长度下限为 1。最大长度为 50000。

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

ResourceReference

服务: Amazon Lex Model Building Service

描述的资源引用您正在尝试删除的资源。此对象作为 `ResourceInUseException` 异常的一部分返回。

内容

name

资源的名称，该资源正在使用您正在尝试删除的资源。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：`[a-zA-Z_]+`

必需：否

version

资源的版本，该资源正在使用您正在尝试删除的资源。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：`\$LATEST|[0-9]+`

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

Slot

服务: Amazon Lex Model Building Service

标识特定插槽的版本。

内容

name

槽的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：`^([A-Za-z](-|_|.)?)+$`

必需：是

slotConstraint

指定槽是必需的还是可选的。

类型：字符串

有效值：`Required | Optional`

必需：是

defaultValueSpec

插槽的默认值的列表。当 Amazon Lex 未决定插槽值时，将使用默认值。您可以从上下文变量、会话属性和定义的值指定默认值。

类型：SlotDefaultValueSpec 对象

必需：否

description

插槽的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

必需：否

obfuscationSetting

确定对话日志和存储的言语中是否对插槽进行了混淆处理。对插槽进行混淆处理时，该值将替换为大括号 ({}) 中的插槽名称。例如，如果插槽名称为“full_name”，则混淆处理后的值将替换为“{full_name}”。有关更多信息，请参阅插槽混淆处理。

类型：字符串

有效值：`NONE | DEFAULT_OBFUSCATION`

必需：否

priority

指示 Amazon Lex 从用户引发插槽值的顺序。例如，如果意图有两个优先级分别为 1 和 2 的插槽，则 AWS Amazon Lex 会首先为优先级为 1 的插槽引发一个值。

如果多个插槽优先级相同，则 Amazon Lex 引发值的顺序是任意的。

类型：整数

有效范围：最小值为 0。最大值为 100。

必需：否

responseCard

基于文本的客户端使用的插槽类型的一组可能的响应。用户从响应卡中选择一个选项，而不是使用文字进行回复。

类型：字符串

长度限制：长度下限为 1。最大长度为 50000。

必需：否

sampleUtterances

如果您知道用户可能用于响应 Amazon Lex 插槽值请求的特定模式，则可以提供这些言语来提高准确性。该项为可选项。在大多数情况下，Amazon Lex 能够理解用户的言语。

类型：字符串数组

数组成员：最少 0 项。最多 10 项。

长度限制：长度下限为 1。最大长度为 200。

必需：否

slotType

插槽的类型，可以是您定义的自定义插槽类型，也可以是内置插槽类型之一。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：`^((AMAZON\.)_?|[A-Za-z]_?)+`

必需：否

slotTypeVersion

插槽类型的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：`\$LATEST|[0-9]+`

必需：否

valueElicitationPrompt

Amazon Lex 用来从用户引发插槽值的提示。

类型：Prompt 对象

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

SlotDefaultValue

服务: Amazon Lex Model Building Service

插槽的默认值。

内容

defaultValue

插槽的默认值。您可以指定以下几项之一：

- `#context-name.slot-name` — 上下文“context-name”中的插槽值“slot-name”。
- `{attribute}` — 会话属性“attribute”的插槽值。
- `'value'` — 离散值“value”。

类型：字符串

长度限制：长度下限为 1。最大长度为 202。

必需：是

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

SlotDefaultValueSpec

服务: Amazon Lex Model Building Service

包含插槽的默认值。当 Amazon Lex 未决定插槽值时，将使用默认值。

内容

defaultValueList

插槽的默认值。您可以指定多个默认值。例如，您可以从匹配的上下文变量、会话属性或固定值中指定要使用的默认值。

所选的默认值是根据您在列表中指定的顺序选择的。例如，如果您按顺序指定上下文变量和固定值，Amazon Lex 将使用上下文变量（如果可用），否则它将使用固定值。

类型：SlotDefaultValue 对象数组

数组成员：最少 0 个物品。最多 10 个物品。

必需：是

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

SlotTypeConfiguration

服务: Amazon Lex Model Building Service

提供插槽类型的配置信息。

内容

regexConfiguration

用于验证槽值的正则表达式。

类型：SlotTypeRegexConfiguration 对象

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

SlotTypeMetadata

服务: Amazon Lex Model Building Service

提供有关插槽类型的信息。

内容

createdDate

插槽类型的创建日期。

类型：时间戳

必需：否

description

槽类型的描述。

类型：字符串

长度约束：最小长度为 0。最大长度为 200。

必需：否

lastUpdatedDate

插槽类型的更新日期。创建资源时，创建日期和上次更新日期相同。

类型：时间戳

必需：否

name

槽类型的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：`^([A-Za-z]_?)+$`

必需：否

version

插槽类型的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：\$LATEST|[0-9]+

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

SlotTypeRegexConfiguration

服务: Amazon Lex Model Building Service

提供用于验证槽值的正则表达式。

内容

pattern

用于验证槽值的正则表达式。

使用标准正则表达式。Amazon Lex 支持在正则表达式中使用以下字符：

- A-Z, a-z
- 0-9
- Unicode 字符（“\ u<Unicode>”）

使用四位数表示 Unicode 字符，如“\u0041”或“\u005A”。

不支持以下正则表达式运算符：

- 无限重复符：*、+ 或 {x,}，无上限。
- 通配符（.）

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

必需：是

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

Statement

服务: Amazon Lex Model Building Service

向用户传达信息的消息集合。在运行时，Amazon Lex 会选择要传达的消息。

内容

messages

消息对象的集合。

类型：Message 对象数组

数组成员：最少 1 个物品。最多 15 项。

必需：是

responseCard

在运行时，如果客户端正在使用 PostTextAPI，Amazon Lex 会在响应中包含响应卡。它使用所有会话属性和插槽值替换响应卡中的占位符。

类型：字符串

长度限制：长度下限为 1。最大长度为 50000。

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

Tag

服务: Amazon Lex Model Building Service

标识机器人、机器人别名或机器人通道的键/值对列表。标签键和值可以包含 Unicode 字母、数字、空格及以下符号：_ . : / = + - @。

内容

key

用于标签的键。键不区分大小写，并且必须是唯一的。

类型：字符串

长度限制：长度下限为 1。最大长度为 128。

必需：是

value

与键关联的值。该值可以是空字符串，但不能为 null。

类型：字符串

长度约束：最小长度为 0。最大长度为 256。

必需：是

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

UtteranceData

服务: Amazon Lex Model Building Service

提供有关向您的机器人发出的单句言语的信息。

内容

count

言语受到处理的次数。

类型：整数

必需：否

distinctUsers

使用该言语的个人总数。

类型：整数

必需：否

firstUtteredDate

首次录制言语的日期。

类型：时间戳

必需：否

lastUtteredDate

末次录制言语的日期。

类型：时间戳

必需：否

utteranceString

用户输入的文本或音频片段的文本表示形式。

类型：字符串

长度限制：长度下限为 1。最大长度为 2000。

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

UtteranceList

服务: Amazon Lex Model Building Service

提供针对特定版本的机器人所说的言语列表。该列表最多包含 100 句言语。

内容

botVersion

用于处理列表的机器人的版本。

类型：字符串

长度限制：长度下限为 1。长度上限为 64。

模式：`\$LATEST|[0-9]+`

必需：否

utterances

一个或多个 UtteranceData 对象，其中包含有关已向机器人发出的言语的信息。每个对象的最大数目为 100。

类型：UtteranceData 对象数组

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

Amazon Lex 运行时服务

Amazon Lex 运行时服务支持以下数据类型：

- ActiveContext

- ActiveContextTimeToLive
- Button
- DialogAction
- GenericAttachment
- IntentConfidence
- IntentSummary
- PredictedIntent
- ResponseCard
- SentimentResponse

ActiveContext

服务: Amazon Lex Runtime Service

上下文是一个变量，它包含有关用户与 Amazon Lex 之间对话的当前状态的信息。上下文可以在履行意图时由 Amazon Lex 自动设置，也可以在运行时使用 PutContent、PutText 或 PutSession 操作进行设置。

内容

name

上下文的名称。

类型：字符串

长度约束：最小长度为 1。最大长度为 100。

模式：^([A-Za-z]_?)+$

必需：是

parameters

当前上下文的状态变量。在后续事件中，您可以使用这些值作为插槽的默认值。

类型：字符串到字符串映射

映射条目：最低 0 项。最多 10 个物品。

密钥长度限制：最小长度为 1。最大长度为 100。

值长度限制：最小长度为 1。最大长度为 1024。

必需：是

timeToLive

上下文保持活动状态的时间长度或回合数。

类型：ActiveContextTimeToLive 对象

必需：是

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

ActiveContextTimeToLive

服务: Amazon Lex Runtime Service

上下文保持活动状态的时间长度或回合数。

内容

timeToLiveInSeconds

在 PostContent 或 PostText 响应中首次发送上下文后，上下文应处于活动状态的秒数。您可以设置 5 到 86,400 秒（24 小时）之间的值。

类型：整数

有效范围：最小值为 5。最大值为 86400。

必需：否

turnsToLive

上下文应处于活动状态的对话回合次数。对话回合是指一个 PostContent 或 PostText 请求以及来自 Amazon Lex 的相应响应。

类型：整数

有效范围：最小值为 1。最大值为 20。

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

Button

服务: Amazon Lex Runtime Service

表示要在客户端平台（Facebook、Slack 等）上显示的选项

内容

text

用户可以在按钮上看到的文本。

类型：字符串

长度限制：长度下限为 1。最大长度为 15。

必需：是

value

当用户选择按钮时发送给 Amazon Lex 的值。例如，考虑按钮文本“NYC”。当用户选择该按钮时，发送的值可以是“纽约市”。

类型：字符串

长度限制：长度下限为 1。最大长度为 1000。

必需：是

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

DialogAction

服务: Amazon Lex Runtime Service

描述机器人与用户互动时应采取的下一个操作，并提供有关该操作发生的上下文的信息。使用 DialogAction 数据类型将交互设置为特定状态，或让交互返回到先前的状态。

内容

type

机器人与用户交互时应采取的下一个操作。可能的值包括：

- ConfirmIntent — 下一个操作是询问用户意图是否已完成并准备好履行。这是一个回答“是”或“否”的问题，例如“是否下订单？”
- Close — 表示用户不会做出响应。例如，“您的披萨已下单”不需要响应。
- Delegate — 下一个操作由 Amazon Lex 决定。
- ElicitIntent — 下一个操作是确定用户想要履行的意图。
- ElicitSlot — 下一个操作是从用户引发插槽值。

类型：字符串

有效值：ElicitIntent | ConfirmIntent | ElicitSlot | Close | Delegate

必需：是

fulfillmentState

意图的履行状态。可能的值包括：

- Failed — 与意图关联的 Lambda 函数未能履行意图。
- Fulfilled — 与意图关联的 Lambda 函数已履行意图。
- ReadyForFulfillment — 存在意图所需的所有信息，并且客户端应用程序已准备好履行意图。

类型：字符串

有效值：Fulfilled | Failed | ReadyForFulfillment

必需：否

intentName

意图的名称。

类型：字符串

必需：否

message

应显示给用户的消息。如果您不指定消息，Amazon Lex 将使用为意图配置的消息。

类型：字符串

长度限制：长度下限为 1。最大长度为 1024。

必需：否

messageFormat

- `PlainText` — 消息包含 UTF-8 纯文本。
- `CustomPayload` — 消息是客户端的自定义格式。
- `SSML` — 消息包含为语音输出设置格式的文本。
- `Composite` — 消息包含一个转义的 JSON 对象，其中包含一条或多条消息。有关更多信息，请参阅消息组。

类型：字符串

有效值：`PlainText | CustomPayload | SSML | Composite`

必需：否

slots

已收集到的插槽及其值的映射。

类型：字符串到字符串映射

必需：否

slotToElicit

应从用户引发的插槽的名称。

类型：字符串

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

GenericAttachment

服务: Amazon Lex Runtime Service

表示显示提示时向用户呈现的选项。它可以是图像、按钮、链接或文本。

内容

attachmentLinkUrl

响应卡上附件的 URL。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：否

buttons

要向用户显示的选项列表。

类型：Button 对象数组

数组成员：最少 0 个物品。最多 5 项。

必需：否

imageUrl

向用户显示的图像的 URL。

类型：字符串

长度限制：最小长度为 0。最大长度为 2048。

必需：否

subTitle

标题下方显示的副标题。

类型：字符串

长度限制：长度下限为 1。最大长度为 80。

必需：否

title

选项的标题。

类型：字符串

长度限制：长度下限为 1。最大长度为 80。

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

IntentConfidence

服务: Amazon Lex Runtime Service

提供一个分数，表示 Amazon Lex 对意图可满足用户的意图的置信度。

内容

score

该分数表示 Amazon Lex 对意图可满足用户的意图的置信度。范围介于 0.00 和 1.00 之间。分数越高，置信度越高。

类型：双精度

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

IntentSummary

服务: Amazon Lex Runtime Service

提供有关意图状态的信息。您可以使用此信息来获取意图的当前状态，以便可以处理意图，也可以将意图恢复到其先前的状态。

内容

dialogActionType

机器人与用户交互时应采取的下一个操作。可能的值包括：

- ConfirmIntent — 下一个操作是询问用户意图是否已完成并准备好履行。这是一个回答“是”或“否”的问题，例如“是否下订单？”
- Close — 表示用户不会做出响应。例如，“您的披萨已下单”不需要响应。
- ElicitIntent — 下一个操作是确定用户想要履行的意图。
- ElicitSlot — 下一个操作是从用户引发插槽值。

类型：字符串

有效值：ElicitIntent | ConfirmIntent | ElicitSlot | Close | Delegate

必需：是

checkpointLabel

用于标识特定意图的用户定义标签。您可以使用此标签返回到之前的意图。

使用 GetSessionRequest 操作的 checkpointLabelFilter 参数筛选操作返回的意图，以仅显示带有指定标签的意图。

类型：字符串

长度限制：长度下限为 1。最大长度为 255。

模式：[a-zA-Z0-9-]+

必需：否

confirmationStatus

用户响应确认提示后意图的状态。如果用户确认意图，则 Amazon Lex 会将此字段设置为 Confirmed。如果用户拒绝意图，则 Amazon Lex 会将此值设置为 Denied。可能的值包括：

- Confirmed — 用户已对确认提示响应“是”，从而确认意图已完成且已准备好履行。
- Denied — 用户已对确认提示响应“否”。
- None — 永不提示用户进行确认；或者，系统已提示用户，但用户未确认也未拒绝提示。

类型：字符串

有效值：None | Confirmed | Denied

必需：否

fulfillmentState

意图的履行状态。可能的值包括：

- Failed — 与意图关联的 Lambda 函数未能履行意图。
- Fulfilled — 与意图关联的 Lambda 函数已履行意图。
- ReadyForFulfillment — 存在意图所需的所有信息，并且客户端应用程序已准备好履行意图。

类型：字符串

有效值：Fulfilled | Failed | ReadyForFulfillment

必需：否

intentName

意图的名称。

类型：字符串

必需：否

slots

已收集到的插槽及其值的映射。

类型：字符串到字符串映射

必需：否

slotToElicit

要从用户引发的下一个插槽。如果没有插槽可以引发，则该字段为空。

类型：字符串

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

PredictedIntent

服务: Amazon Lex Runtime Service

Amazon Lex 提出的意图可以满足用户的意图。包括意图名称、Amazon Lex 对用户意图得到满足的置信度，以及为意图定义的插槽。

内容

intentName

Amazon Lex 提出可以满足用户的意图的意图名称。

类型：字符串

必需：否

nluIntentConfidence

表示 Amazon Lex 对意图是满足用户的意图的置信度。

类型：IntentConfidence 对象

必需：否

slots

与预测意图关联的插槽和插槽值。

类型：字符串到字符串映射

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

ResponseCard

服务: Amazon Lex Runtime Service

如果您在创建机器人时配置了响应卡，Amazon Lex 会替换可用的会话属性和插槽值，然后将其返回。响应卡也可以来自 Lambda 函数（意图上的 `dialogCodeHook` 和 `fulfillmentActivity`）。

内容

contentType

响应的内容类型。

类型：字符串

有效值：`application/vnd.amazonaws.card.generic`

必需：否

genericAttachments

代表选项的附件对象数组。

类型：GenericAttachment 对象数组

数组成员：最少 0 个物品。最多 10 个物品。

必需：否

version

响应卡格式的版本。

类型：字符串

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

SentimentResponse

服务: Amazon Lex Runtime Service

用言语表达的情绪。

当机器人配置为向 Amazon Comprehend 发送言语以进行情绪分析时，此字段结构将包含分析结果。

内容

sentimentLabel

Amazon Comprehend 置信度最高的情绪推断。

类型：字符串

必需：否

sentimentScore

情绪推断正确的可能性。

类型：字符串

必需：否

另请参阅

有关以特定语言之一使用此 API 的更多信息 AWS SDKs，请参阅以下内容：

- AWS 适用于 C++ 的 SDK
- AWS 适用于 Java 的 SDK V2
- AWS 适用于 Ruby V3 的 SDK

Amazon Lex 的文档历史记录

- 最近文档更新时间：2021 年 9 月 9 日

下表介绍每一版 Amazon Lex 中的重大更改。要获得本文档的更新通知，您可以订阅 RSS 源。

变更	说明	日期
新特征	Amazon Lex 现在支持韩语 (ko-KR) 区域设置。有关更多信息，请参阅 Amazon Lex 支持的语言。	2021 年 9 月 9 日
新特征	Amazon Lex 现在支持英语（印度）区域设置。有关更多信息，请参阅 Amazon Lex 支持的语言。	2021 年 7 月 15 日
新特征	现在，Amazon Lex 提供了一种工具，可以将机器人迁移到 Amazon Lex V2 API。有关更多信息，请参阅迁移机器人。	2021 年 7 月 13 日
新特征	Amazon Lex 现在支持日语（日本）区域设置。有关更多信息，请参阅 Amazon Lex 支持的语言。	2021 年 4 月 1 日
新特征	Amazon Lex 现在支持德语（德国）(de-DE) 和西班牙语（拉丁美洲）(es-419) 区域设置。有关更多信息，请参阅 Amazon Lex 支持的语言。	2020 年 11 月 23 日
新特征	Amazon Lex 现在支持使用上下文来管理激活意图。有关更	2020 年 11 月 19 日

	多信息，请参阅设置意图上下文。	
新特征	Amazon Lex 现在支持法语 (fr-FR)、加拿大法语 (fr-CA)、意大利语 (it-IT) 和西班牙语 (es-ES) 区域设置。有关支持的区域设置的完整列表，请参阅 Amazon Lex 支持的语言。	2020 年 11 月 11 日
新特征	Amazon Lex 现在支持西班牙语（美国）(es-US) 区域设置。有关更多信息，请参阅 Amazon Lex 支持的语言。	2020 年 9 月 22 日
新特征	Amazon Lex 现在支持英语（英国）(en-GB) 区域设置。有关更多信息，请参阅 Amazon Lex 支持的语言。	2020 年 9 月 15 日
新特征	Amazon Lex 现在支持英语（澳大利亚）(en-AU) 区域设置。有关更多信息，请参阅 Amazon Lex 支持的语言。	2020 年 9 月 8 日
新特征	Amazon Lex 现在有 7 种新的内置意图和 9 种新的内置插槽类型。有关更多信息，请参阅内置意图和插槽类型。	2020 年 9 月 8 日
新示例	通过使用 Amazon Kendra 搜索答案，了解如何创建 Amazon Lex 机器人，客户支持客服可以使用该机器人来回答客户的问题。有关更多信息，请参阅示例：呼叫中心客服助理。	2020 年 8 月 10 日

新特征	Amazon Lex 现在可以根据置信度分数返回多达四个替代意图。有关更多信息，请参阅使用置信度分数。	2020 年 8 月 6 日
区域扩展	Amazon Lex 现已在亚太地区（东京）(ap-northeast-1) 区域发售。	2020 年 6 月 30 日
新特征	Amazon Lex 现在支持搜索 Amazon Kendra 索引以获得常见问题的答案。有关更多信息，请参阅亚马逊。KendraSearchIntent。	2020 年 6 月 11 日
新特征	Amazon Lex 现可在对话日志中返回更多信息。有关更多信息，请参阅在 Amazon 日志中查看文本 CloudWatch 日志。	2020 年 6 月 9 日
区域扩展	Amazon Lex 现已在亚太地区（新加坡）(ap-southeast-1)、欧洲地区（法兰克福）(eu-central-1) 和欧洲地区（伦敦）(eu-west-2) 发售。	2020 年 4 月 23 日
新特征	Amazon Lex 现在支持标记。您可以使用标记来识别资源、分配成本和控制访问权限。有关更多信息，请参阅标记 Amazon Lex 资源。	2020 年 3 月 12 日
新特征	Amazon Lex 现在支持亚马逊的正则表达式。 AlphaNumeric 内置插槽类型。有关更多信息，请参阅亚马逊。AlphaNumeric。	2020 年 2 月 6 日

新特征	Amazon Lex 现在可以记录对话信息并对这些日志中的插槽值进行混淆处理。有关详细信息，请参阅创建对话日志和槽混淆处理。	2019 年 12 月 19 日
区域扩展	Amazon Lex 现已在亚太地区（悉尼）(ap-southeast-2) 发售。	2019 年 12 月 17 日
新特征	Amazon Lex 现在符合 HIPAA 标准。有关更多信息，请参阅 Amazon Lex 的合规性验证。	2019 年 12 月 10 日
新特征	Amazon Lex 现在可以将用户的言语发送到 Amazon Comprehend 来分析言语的绪情。有关更多信息，请参阅情绪分析。	2019 年 11 月 21 日
新特征	Amazon Lex 现在符合 SOC 标准。有关更多信息，请参阅 Amazon Lex 的合规性验证。	2019 年 11 月 19 日
新特征	Amazon Lex 现在符合 PCI 标准。有关更多信息，请参阅 Amazon Lex 的合规性验证。	2019 年 10 月 17 日
新特征	添加了对向意图添加检查点的支持，以便您可以在对话过程中轻松返回到意图。有关更多信息，请参阅管理会话。	2019 年 10 月 10 日

新特征	添加了对 AMAZON.FallbackIntent 的支持，以便您的自动程序可以处理用户输入不符合预期的情况。有关更多信息，请参阅亚马逊。FallbackIntent。	2019 年 10 月 3 日
新特征	借助 Amazon Lex，您可以管理机器人的会话信息。有关更多信息，请参阅使用 Amazon Lex API 管理会话。	2019 年 8 月 8 日
区域扩展	Amazon Lex 现已在美国西部（俄勒冈州）(us-west-2) 发售。	2018 年 5 月 8 日
新特征	添加了对以 Amazon Lex 格式导出和导入的支持。有关更多信息，请参阅导入和导出 Amazon Lex 机器人、意图和插槽类型。	2018 年 2 月 13 日
新特征	Amazon Lex 现在支持机器人的更多响应消息。有关更多信息，请参阅 响应。	2018 年 2 月 8 日
区域扩展	Amazon Lex 现已在欧洲地区（爱尔兰）(eu—west-1) 发售。	2017 年 11 月 21 日
新特征	添加了对在 Kik 上部署 Amazon Lex 机器人的支持。有关更多信息，请参阅将 Amazon Lex 机器人与 Kik 集成。	2017 年 11 月 20 日

新特征	添加了对新的内置槽类型和请求属性的支持。有关更多信息，请参阅内置槽类型和设置请求属性。	2017 年 11 月 3 日
新特征	增加了导出到 Alexa Skills Kit 功能。有关更多信息，请参阅导出到 Alexa Skill。	2017 年 9 月 7 日
新特征	增加了槽类型值的同义词支持。有关更多信息，请参阅自定义槽类型。	2017 年 8 月 31 日
新特征	增加了 AWS CloudTrail 集成。有关更多信息，请参阅使用 AWS CloudTrail 日志监控 Amazon Lex API 调用。	2017 年 8 月 15 日
扩展了文档	为添加了入门示例 AWS CLI。有关更多信息，请参阅 https://docs.aws.amazon.com/lex/latest/dg/gs-cli.html	2017 年 5 月 22 日
新指南	这是《Amazon Lex 用户指南》的首次发布。	2017 年 4 月 19 日

AWS 词汇表

有关最新 AWS 术语，请参阅《AWS 词汇表 参考资料》中的AWS 词汇表。