

Contents

[Azure Bot 服務文件](#)

[概觀](#)

[關於 Azure Bot Service](#)

[新功能](#)

[快速入門](#)

[建立具有 Azure Bot Service 的 Bot](#)

[在本機上建立 Bot](#)

[.NET](#)

[JavaScript](#)

[教學課程](#)

[1. 建立及部署基本 Bot](#)

[2. 新增 QnA Maker, 並重新部署 Bot](#)

[在您的 Bot 中新增驗證](#)

[範例](#)

[GitHub 的 Bot Framework 範例存放庫](#)

[概念](#)

[Bot 的運作方式](#)

[管理狀態](#)

[對話方塊程式庫](#)

[中介軟體](#)

[管理 bot 資源](#)

[Bot 服務範本](#)

[認知服務](#)

[Bot 的重要案例](#)

[商務 Bot](#)

[Cortana 技能 Bot](#)

[企業生產力 Bot](#)

[資訊 Bot](#)

[物聯網 Bot](#)

作法

設計

Bot 設計準則

首次互動

設計和控制交談流程

設計 bot 導覽

設計使用者體驗

模式

工作自動化

知識庫

將對話遞交給人員處理

應用程式中的 Bot

網站中的 Bot

開發

傳送及接收文字訊息

將媒體新增至訊息

新增按鈕來引導使用者的動作

儲存使用者和對話資料

提示使用者輸入

將歡迎訊息傳送給使用者

將主動式通知傳送給使用者

實作循序對話流程

將自然語言理解新增至您的 Bot

使用 QnA Maker 回答使用者的問題

使用多個 LUIS 和 QnA 模型

使用分支和迴圈建立進階的交談流程

重複使用對話方塊

處理使用者中斷

直接寫入儲存體

在您的 Bot 中新增驗證

為您的 Bot 實作自訂儲存空間

將遙測資料新增至 Bot

在 Bot 中使用 Direct Line Speech

.NET

適用於 .NET 的 Bot Framework SDK

重要概念

訊息和活動

活動概觀

建立訊息

將媒體附件新增至訊息

將複合式資訊卡 (Rich Card) 新增至訊息

將語音新增至訊息

將輸入提示新增至訊息

將建議的動作新增至訊息

傳送及接收活動

實作通用訊息處理常式

攔截訊息

傳送主動式訊息

對話方塊

對話方塊概觀

管理對話流程

可評分的對話方塊

FormFlow

FormFlow 的基本功能

FormFlow 的進階功能

FormBuilder

模式語言

當地語系化

JSON 結構描述

聲道

實作通道特有功能

建置 Cortana 技能

使用 Skype 執行音訊通話

狀態資料

[管理狀態資料](#)

[使用 Cosmos DB 管理狀態資料](#)

[使用資料表儲存體管理狀態資料](#)

[使用 LUIS 辨識意圖](#)

[要求付款](#)

[新增 Azure 搜尋服務](#)

[保護 Bot](#)

[Node.js](#)

[適用於 Node.js 的 Bot Framework SDK](#)

[重要概念](#)

[對話方塊](#)

[對話方塊概觀](#)

[定義對話的步驟](#)

[提示使用者輸入](#)

[管理對話流程](#)

[取代對話方塊](#)

[處理使用者動作](#)

[訊息](#)

[建立訊息](#)

[傳送和接收附件](#)

[傳送主動式訊息](#)

[將複合式資訊卡 \(Rich Card\) 新增至訊息](#)

[將語音新增至訊息](#)

[將輸入提示新增至訊息](#)

[將建議的動作新增至訊息](#)

[傳送輸入指標](#)

[攔截訊息](#)

[聲道](#)

[建置 Cortana 技能](#)

[進行 Skype 通話](#)

[狀態資料](#)

[管理狀態資料](#)

[使用 Cosmos DB 管理狀態資料](#)

[使用資料表儲存體管理狀態資料](#)

[從訊息內容辨識意圖](#)

[使用 LUIS 辨識意圖](#)

[處理使用者和對話事件](#)

[支援當地語系化](#)

[使用 backchannel 機制](#)

[要求付款](#)

[新增 Azure 搜尋服務](#)

[偵錯](#)

[對 Bot 進行偵錯](#)

[測試及偵錯指導方針](#)

[使用 Bot Framework 模擬器進行偵錯](#)

[使用文字記錄檔進行測試及偵錯](#)

[測試 Cortana 技能](#)

[部署](#)

[將 Bot 部署至 Azure](#)

[設定持續部署](#)

[管理](#)

[管理 Bot](#)

[Bot 分析](#)

[聲道](#)

[將 Bot 連線至通道](#)

[實作通道特有功能](#)

[Cortana](#)

[Direct Line](#)

[關於 Direct Line](#)

[連線至 Direct Line](#)

[連線至 Direct Line Speech](#)

[電子郵件](#)

[啟用網路聊天中的語音](#)

[Facebook](#)

[GroupMe](#)
[Kik](#)
[線條](#)
[Microsoft Teams](#)
[Skype](#)
[商務用 Skype](#)
[Slack](#)
[Telegram](#)
[Twilio](#)
[網路聊天](#)

[設定 Bot 設定](#)
[設定語音預備](#)
[使用 Azure Bot 服務註冊 Bot](#)
[遷移 Bot](#)

[移轉](#)
[v3 和 v4.NET SDK 之間的差異](#)
[將 .NET SDK v3 bot 遷移至 v4](#)

[參考](#)

[.NET SDK v4](#)
[JavaScript SDK v4](#)
[REST](#)
[Bot Framework REST API](#)

[概觀](#)
[重要概念](#)
[使用 REST 建立 Bot](#)
[API 參考](#)

[連接器](#)
[驗證](#)
[活動概觀](#)
[建立訊息](#)
[傳送及接收訊息](#)
[將媒體附件新增至訊息](#)

[將複合式資訊卡 \(Rich Card\) 新增至訊息](#)

[將語音新增至訊息](#)

[將輸入提示新增至訊息](#)

[將建議的動作新增至訊息](#)

[實作通道特定的功能](#)

[管理狀態資料](#)

[Direct Line API 3.0](#)

[重要概念](#)

[驗證](#)

[開始對話](#)

[重新連線到對話](#)

[將活動傳送到 Bot](#)

[從 Bot 接收活動](#)

[結束對話](#)

[API 參考](#)

[Swagger 檔案](#)

[Direct Line API 1.1](#)

[重要概念](#)

[驗證](#)

[開始對話](#)

[將訊息傳送至 Bot](#)

[從 Bot 接收訊息](#)

[API 參考](#)

[Swagger 檔案](#)

[.NET SDK v3](#)

[Node.js SDK v3](#)

[實體和活動類型](#)

[資源](#)

[虛擬小幫手](#)

[概觀](#)

[範本簡介](#)

[技術](#)

[概觀](#)

[常見問題集](#)

[取得支援](#)

[通道參考](#)

[識別碼指南](#)

[App Insights 金鑰](#)

[使用者代理程式要求](#)

[Bot 檢閱指導方針](#)

[Bot Framework -- 活動結構描述](#)

[Bot Framework -- 卡片結構描述](#)

[Bot Framework -- 文字記錄結構描述](#)

[Bot 服務合規性](#)

[疑難排解](#)

[針對一般問題進行疑難排解](#)

[針對 Bot 組態問題進行疑難排解](#)

[針對 HTTP 500 錯誤進行疑難排解](#)

[針對驗證進行疑難排解](#)

適用於:  SDK v4  SDK v3

Azure Bot 服務文件

Azure Bot 服務提供專為 Bot 開發打造的整合式環境，可讓您在同一處建置、連線、測試、部署及管理智慧型 Bot。Azure Bot 服務會利用 Bot Framework SDK，並支援 C# 和 JavaScript。了解如何透過我們的快速入門、教學課程和範例使用 Bot 服務。

5 分鐘快速入門



Azure 入口網站

C#

JavaScript

逐步教學課程

[1. 建立及部署基本 Bot](#)

[2. 新增 QnA Maker，並重新部署 Bot](#)

參考

[API](#)

[REST](#)

[SDK](#)

[.NET](#)

[JavaScript](#)

關於 Azure Bot Service

2019/5/14 • [Edit Online](#)

適用於:  SDK v4  SDK v3

Azure Bot 服務和 Bot Framework 提供可在單一位置建立、測試、部署及管理智慧型 Bot 的工具。利用 SDK 提供的模組化可延伸架構，工具、範本和 AI 服務開發人員可以建立可使用語音、理解自然語言、處理問與答等功能的 Bot。

什麼是 Bot？

Bot 提供的體驗比較不像使用電腦，比較像是與人溝通，或至少是與智慧型機器人溝通。在不再需要直接人為介入的自動化系統上，Bot 可用於輪替簡單、重複性工作，例如預訂晚餐或蒐集設定檔資訊。使用者可使用文字、互動式卡片和語音來與 Bot 交談。Bot 互動可以是快速的問與答，也可以是以智慧方式提供服務存取權的複雜對話。

Bot 很類似現代化 Web 應用程式，在網際網路上運作，並使用 API 來傳送和接收訊息。視 Bot 的種類而定，Bot 的功能差異很大。現代化 Bot 軟體依賴一些技術和工具，可在各種平台上提供日益複雜的體驗。不過，簡單的 Bot 可能只會收到訊息，並以極少的相關程式碼來回應使用者。

Bot 可執行其他類型的軟體可以執行的作業：讀取和寫入檔案、使用資料庫和 API，以及進行一般計算工作。Bot 的特點就是其使用通常保留給人與人通訊的機制。

Azure Bot 服務和 Bot Framework 可提供：

- 可供開發 Bot 的 Bot Framework SDK
- Bot Framework 工具，可涵蓋端對端 Bot 開發工作流程
- Bot Framework Service (BFS)，可在 Bot 與通道之間傳送及接收訊息和事件
- Azure 中的 Bot 部署和通道組態

此外，Bot 可使用其他 Azure 服務，例如：

- 用以建置智慧型應用程式的 Azure 認知服務
- 適用於雲端儲存體解決方案的 Azure 儲存體

建立 Bot

Azure Bot 服務和 Bot Framework 可提供一組整合式工具與服務，可加快此程序。選擇您慣用的開發環境或命令列工具來建立 Bot。C#、JavaScript 和 Typescript 均有 SDK。(適用於 Java 和 Python 的 SDK 正在開發中。)我們在 Bot 各種開發階段皆提供各項工具，協助您設計及建置 Bot。



規劃

如同任何類型的軟體，務必徹底了解目標、程序及使用者需求，才能建立成功的 Bot。撰寫程式碼之前，請先參閱 Bot [設計指南](#)了解最佳作法，並找出您的 Bot 需求。您可以建立簡單的 Bot 或納入更複雜的功能，例如語音、自然語言理解或問題解答。

建置

Bot 是一項 Web 服務，可實作對話式介面並透過 Bot Framework Service 進行通訊，以傳送和接收訊息和事件。Bot Framework Service 是 Azure Bot Service 和 Bot Framework 的其中一個元件。您可以使用任意多個環境和語言建立 Bot。您可以在 [Azure 入口網站](#) 中開始進行 Bot 開發，或使用 [C# | JavaScript] 範本進行本機開發。

我們在 Azure Bot Service 和 Bot Framework 中提供可用來增強 Bot 功能的其他元件

功能	說明	連結
新增自然語言處理	讓您的 Bot 能夠理解自然語言、理解拼字錯誤、使用語音，以及辨識使用者的意圖	如何使用 LUIS
回答問題	新增知識庫，以更自然的對話方式回答使用者的問題	如何使用 QnA Maker
管理多個模型	使用多個模型時（例如 LUIS 和 QnA Maker），可在 Bot 對話期間，採用智慧方式判斷各個模型的使用時機	分派工具
新增卡片和按鈕	利用文字以外的媒體（例如圖形、功能表和卡片）來強化使用者體驗	如何 新增卡片

NOTE

上表並非完整清單。請參閱左側文章了解更多 Bot 功能，第一篇為[傳送訊息](#)。

此外，我們會提供命令列工具，協助您建立、管理及測試 Bot 資產。這些工具可設定 LUIS 應用程式、建置 QnA 知識庫、建置模型以在元件之間分派、模擬對話等等。您可以在命令列工具[讀我檔案](#)中找到更多詳細資訊。

您也可以存取各種範例，這些範例展現許多可透過 SDK 取得的功能。這些很適合尋求更多元用途起點的開發人員。

測試

Bot 是將許多不同組件整合在一起運作的複雜應用程式。和其他複雜的應用程式一樣，這種方式會導致一些有趣的錯誤，或是讓 Bot 產生出乎意料的行為。在發佈之前，請先測試 Bot。在發行 Bot 以供使用前，我們會提供數種方式來測試 Bot：

- 使用 [模擬器](#) 在本機測試 Bot。Bot Framework 模擬器是獨立的應用程式，不僅提供交談介面，還提供偵錯和訊問工具來協助您了解 Bot 的運作方式和原因。模擬器可以隨著開發中的 Bot 應用程式在本機執行。
- 在 [Web](#) 上測試您的 Bot。透過 Azure 入口網站進行設定後，Bot 也可透過網路聊天介面觸達。網路聊天介面適合用來將 Bot 的存取權授與給測試人員，以及其他無法直接存取 Bot 執行中程式碼的人員。

發佈

當您準備在 Web 上提供您的 Bot 時，請將 Bot 發佈至 [Azure](#) 或自己的 Web 服務或資料中心。在公用網際網路上有一個位址是 Bot 在您的網站上，或在聊天通道內活化的第一個步驟。

連線

將您的 Bot 連接到 Facebook、Messenger、Kik、Skype、Slack、Microsoft Teams、Telegram、簡訊 /SMS、Twilio、Cortana 及 Skype 等通道中。Bot Framework 會進行從上述各種平台傳送和接收訊息所需的大部分工作 - 不論所連線到的通道數目和類型為何，Bot 應用程式都會接收統一、正規化的訊息串流。如需新增通道的資訊，請參閱[通道](#)主題。

評估

使用在 Azure 入口網站收集的資料，就有機會改善 Bot 功能和效能。您可以取得服務層級和流量、延遲與整合等檢測資料。Analytics 提供使用者、訊息和通道資料的相關交談層級報告。如需詳細資訊，請參閱[如何收集分析資料](#)。

後續步驟

請查看這些 Bot [個案研究](#), 或按一下下方連結, 即可建立 Bot。

[建立 Bot](#)

Bot Framework 中的新功能

2019/5/14 • [Edit Online](#)

Bot Framework SDK v4 是[開放原始碼 SDK](#), 可讓開發人員使用其慣用的程式設計語言塑造與建置複雜的對話。

本文摘要說明 Bot Framework 和 Azure Bot Service 中的重要新功能和增強功能。

	C#	JS	PYTHON	JAVA
SDK	4.4.3	4.4.0	4.4.0b1 (預覽)	4.0.0a6 (預覽)
Docs	docs	docs		
範例	.NET Core、WebAPI	Node.js、TypeScript、es6	Python	

Bot Framework SDK (預覽中的新功能) In preview)

- [自適性對話方塊 | docs | C# 範例](#): 自適性對話方塊可讓開發人員建置對話，並隨著對話的進展而動態變化。傳統上，開發人員已事先安排好整個對話流程，但這會限制對話彈性。自適性對話方塊則可讓開發人員更有彈性地回應內容變化，並隨著對話的進展而插入新步驟或整個子對話方塊。
- [語言產生 | docs | C# 範例](#): 語言產生可讓開發人員從程式碼和資源檔案中擷取出內嵌字串，並透過語言產生執行階段和檔案格式來進行管理。語言產生可讓客戶對片語定義多個變化、執行以內容為基礎的簡單運算式、參考對話式記憶體，久而久之，我們將能夠讓所有額外功能變成更自然的對話式體驗。
- [常見的運算式語言 | api](#): 自適性對話方塊和語言產生均依賴並使用常見的運算式語言來支援 Bot 對話。

Botkit

Botkit 是一種開發人員工具和 SDK，可用於為主要的傳訊平台建置聊天 Bot、應用程式和自訂整合。Botkit Bot 會 `hear()` 觸發程序、`ask()` 問題和 `say()` 回覆。開發人員可以使用此語法來建置對話方塊 - 現在可與 Bot Framework SDK 的最新版本交叉相容。

此外，Botkit 還帶來 6 個平台接器，可讓 Javascript Bot 應用程式直接與傳訊平台通訊：[Slack](#)、[Webex Teams](#)、[Google Hangouts](#)、[Facebook Messenger](#)、[Twilio](#) 和[網路聊天](#)。

Botkit 是 Microsoft Bot Framework 的一部分，其依據 [MIT 開放原始碼授權](#)來發行

Bot Framework 解決方案 (預覽中的新功能) In preview)

[Bot Framework 解決方案存放庫](#)是一組範本、解決方案加速器和技能的所在地，可用來協助建置類似小幫手的進階對話式體驗。

NAME	說明
虛擬小幫手	客戶極需要提供專屬於其品牌的對話式小幫手，並且還需要為他們的使用者量身打造，以及在各種畫布和裝置上提供使用。 企業範本可大幅簡化新 Bot 專案的建立程序，包括：基本的對話式意圖、分派整合、QnA Maker、Application Insights 和自動化部署。

NAME	說明
技能	開發人員可以藉由將可重複使用的對話式功能 (稱為技能) 拼接在一起，來撰寫對話式體驗。技能本身就是可從遠端叫用的 Bot，而且有技能開發人員範本 (.NET、TS) 可供加速新技能的建立。
分析	可使用對話式 AI 分析解決方案來重點了解 Bot 的健康情況和行為。檢閱可用的遙測資料、Application Insights 查詢範例和 Power BI 儀表板，來了解 Bot 與使用者的完整對話範圍。

Azure Bot 服務

Azure Bot Service 可讓您裝載有智慧的企業級 Bot，且您可以完整擁有和控制您的資料。開發人員可以註冊其 Bot 並將 Bot 連線至 Skype、Microsoft Teams、Cortana、網路聊天等通道上的使用者。[Azure | docs | 連線至通道](#)

- **Direct Line JS 用戶端**：如果您想要在 Azure Bot Service 中使用 Direct Line 通道，而不要使用網路聊天用戶端，則可以在自訂應用程式中使用 Direct Line JS 用戶端。如需詳細資訊，請移至 [GitHub](#)。
- **新功能！Direct Line Speech 通道**：我們會結合 Bot Framework 和 Microsoft 的語音服務來提供通道，以便能在用戶端與 Bot 應用程式之間雙向串流語音和文字。如需詳細資訊，請參閱如何新增語音通道至 Bot。

Bot Framework 模擬器

[Bot Framework Emulator](#) 是跨平台的桌面應用程式，可讓 Bot 開發人員針對使用 Bot Framework SDK 所建置的 Bot 進行測試和偵錯。您可以使用 Bot Framework Emulator 來測試在電腦本機上執行的 Bot，或用來連線至遠端執行的 Bot。

- [下載最新版本 | Docs](#)

Bot Inspector (預覽中的新功能) In preview)

Bot Framework Emulator 已發行新 Bot Inspector 功能的搶鮮版 (Beta)。其可讓您在 Microsoft Teams、Slack、Cortana、Facebook Messenger、Skype 等通道上對 Bot Framework SDK v4 Bot 進行偵錯和測試。當您有對話時，訊息便會鏡像傳送到 Bot Framework Emulator，您可以在其中檢查 Bot 所收到的訊息資料。此外，也會呈現通道和 Bot 之間任何給定回合的 Bot 狀態快照集。深入了解 [Bot Inspector](#)

相關服務

Language Understanding

該服務以機器學習為基礎，可用來建置自然語言體驗。快速建立持續改進的企業級自訂模型。[Language Understanding 服務 \(LUIS\)](#) 可讓應用程式了解人在文字中所表達的意思。

- **新功能！角色、外部實體和動態實體**：LUIS 新增了數個功能，可讓開發人員從文字擷取更詳細的資訊，讓使用者現在可以用最少的力氣建置更有智慧的解決方案。LUIS 也將角色延伸到所有實體類型，讓相同的實體可根據內容使用不同的子類型來分類。開發人員現在可以更細微的控制其可以使用 LUIS 執行的動作，包括能夠在執行階段透過動態清單和外部實體識別和更新模型。動態清單可用來在預測階段附加至清單實體，而讓使用者專屬資訊可以完全相符。個別的增補實體擷取器會使用外部實體來執行，且該資訊可以附加至 LUIS 作為其他模型的強烈訊號。
- **新功能！分析儀表板**：LUIS 會釋出更詳細、有豐富視覺效果的完善分析儀表板。其方便使用的設計會醒目指出大部分使用者在設計應用程式時所面臨的常見問題，其方法是提供簡單的解決方式說明，以協助使用者深入了解其模型的品質、潛在的資料問題和採用最佳做法的指導方針。

[Docs | 將語言理解新增至 Bot](#)

[QnA Maker](#)

[QnA Maker](#) 是一項雲端式 API 服務，可對您的資料建立交談式的問答層。透過 QnA Maker，您可以根據常見問題集 URL、結構化文件、產品手冊或期刊內容，在短短幾分鐘內建置、定型及發佈簡單的問答 Bot。

- **新功能！擷取管線**：現在，您可以從 URL、檔案和 SharePoint 擷取階層式資訊
- **新功能！智慧**：內容相關的排名模型、主動式學習建議
- **新功能！對話**：QnA Maker 中的多回合對話。

[Docs | 將 QnAMaker 新增至 Bot](#)

語音服務

[語音服務](#)會使用整合的語音服務將音訊轉換成文字，並執行語音翻譯和文字轉換語音。透過語音服務，您可以將語音整合至 Bot、建立自訂喚醒字組，並以多種語言撰寫。

調適型卡片

[自適性卡片](#)是一套開放式標準，可讓開發人員以通用且一致的方式交換卡片內容，並可讓 Bot Framework 的開發人員用來建立絕佳的跨通道對話式體驗。

其他資訊

- 如需詳細**資訊**，請瀏覽 GitHub 頁面。

建立具有 Azure Bot Service 的 Bot

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

Bot 服務會提供建立 Bot 的核心元件，包括用於開發 Bot 的 Bot Framework SDK 和將 Bot 連線至通道的 Bot Framework。建立支援 .NET 和 Node.js 的 Bot 時，Bot 服務提供五種範本供您選擇。在本主題中，您將了解如何使用 Bot 服務來建立使用 Bot Framework SDK 的新 Bot。

登入 Azure

登入 [Azure 入口網站](#)。

TIP

若您還沒有訂用帳戶，則可以註冊[免費帳戶](#)。

建立新的 Bot 服務

1. 按一下 Azure 入口網站左上角的 [建立新資源] 連結，然後選取 [AI + 機器學習服務] > [Web 應用程式 Bot]。
2. 隨即開啟含有 **Web 應用程式 Bot** 相關資訊的新刀鋒視窗。
3. 在 [Bot 服務] 刀鋒視窗中，依照下圖下方表格中說明的內容，提供有關您 Bot 的必要資訊。

Web App Bot

Bot Service

* Bot name [?](#)

* Subscription

* Resource group
 [▼](#)
[Create new](#)

* Location [?](#)
 [▼](#)

Pricing tier ([View full pricing details](#))
 [▼](#)

* App name [?](#)
 .azurewebsites.net

* Bot template
 >

* LUIS App location
 [▼](#)

* App service plan/Location
 >

Application Insights [?](#)
 On Off

* Application Insights Location [?](#)
 [▼](#)

Microsoft App ID and password [?](#)
 >

[Create](#) [Automation options](#)

設定	建議的值	說明
Bot 名稱	您的 Bot 顯示名稱	Bot 在通道和目錄中的顯示名稱。您可以隨時變更此名稱。
訂用帳戶	您的訂用帳戶	選取您要使用的 Azure 訂用帳戶。
資源群組	myResourceGroup	您可以建立新的 資源群組 , 或選擇現有的資源群組。
位置	預設的位置	選取資源群組的地理位置。您可以選擇任何列出的位置, 但通常最佳的選擇是最靠近您客戶的位置。一旦建立 Bot, 就無法變更位置。
定價層	F0	選取定價層。您可以隨時更新定價層。如需詳細資訊, 請參閱 Bot 服務價格 。

設定	建議的值	說明
應用程式名稱	唯一的名稱	Bot 的唯一 URL 名稱。例如，如果您將 Bot 命名為 <i>myawesomebot</i> ，則 Bot 的 URL 將會是 <code>http://myawesomebot.azurewebsites.net</code> 。 名稱只能使用英數字元和底線字元。此欄位有 35 個字元的長度限制。一旦建立 Bot，就無法變更應用程式名稱。
Bot 範本	基本	選擇 [C#] 或 [Node.js]，並選取 [基本] 範本以供此快速入門使用，然後按一下 [選取]。基本範本會建立回應 Bot。 深入了解範本 。
App Service 方案/位置	您的 App Service 方案	選取 App Service 方案 位置。您可以選擇任何列出的位置，但通常最佳的選擇是最靠近您客戶的位置。(不適用於 Functions Bot)。
Application Insights	另一	決定您要開啟或關閉 Application Insights 。如果您選取 [開啟]，您也必須指定區域位置。您可以選擇任何列出的位置，但通常最好是選擇與 Bot 服務位置相同的位置。
Microsoft 應用程式識別碼和密碼	自動建立應用程式識別碼和密碼	如果您需要手動輸入 Microsoft 應用程式識別碼和密碼，請使用此選項。否則，在 Bot 建立程序期間，便會為您建立新的 Microsoft 應用程式識別碼和密碼。

NOTE

如果您建立的是 **Functions Bot**，則不會看到 [App Service 方案/位置] 欄位。相反地，您會看到 [主控方案] 欄位。在該案例中，請選擇[主控方案](#)。

- 按一下 [建立] 以建立服務，並將 Bot 部署到雲端。此程序可能需要幾分鐘的時間。

勾選 [通知] 來確認已部署 Bot。通知會從 [部署進行中] 變更為 [部署成功]。按一下 [前往資源] 按鈕以開啟 Bot 的資源刀鋒視窗。

TIP

基於效能考量，執行 Node.js 範本的 **Functions Bot** 已經使用 Azure Functions Pack 工具封裝。Azure Functions Pack 工具會擷取所有 Node.js 模組，並將它們結合成一個 *.js 檔案。如需詳細資訊，請參閱 [Azure Functions Pack \(英文\)](#)。

測試 Bot

現在，既然您已建立 Bot，請在 [Web 聊天](#)中測試它。輸入訊息，您的 Bot 應會回應。

後續步驟

在此主題中，您已了解如何使用 Bot 服務建立基本 Web 應用程式 Bot/Functions Bot，並在 Azure 中使用內建的 Web 聊天控制項驗證 Bot 功能。現在，請了解如何管理 Bot 並開始處理其原始程式碼。

管理 Bot

適用於:  SDK v4  SDK v3

Azure Bot 服務提供建立 Bot 的核心元件，包括用於開發 Bot 的 Bot Framework SDK 和連接 Bot 與通道的 Bot 服務。在此主題中，您可以使用 Bot Framework SDK v4，選擇 .NET 或 Node.js 範本來建立 Bot。

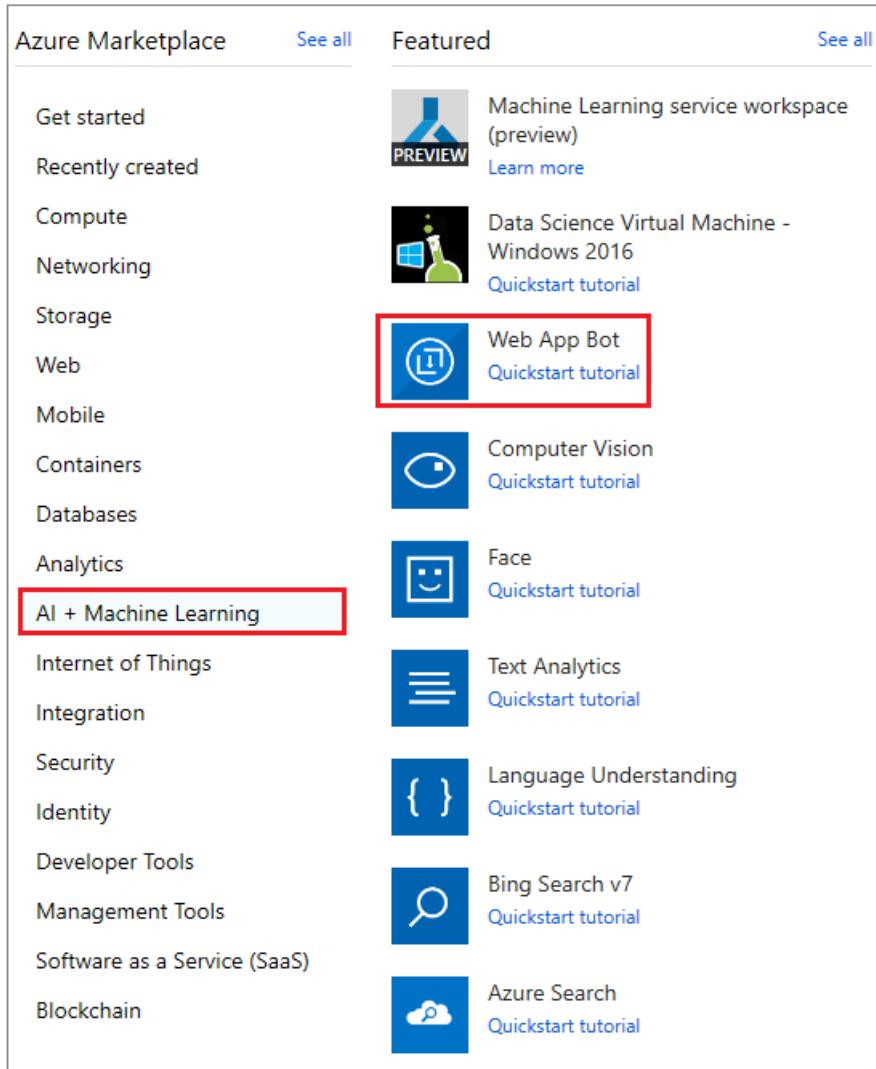
使用 Azure Bot Service 建立 Bot 與在本機建立 Bot 是建立 Bot 的獨立、平行方式。

必要條件

- [Azure 帳戶](#)

建立新的 Bot 服務

1. 登入 [Azure 入口網站](#)。
2. 按一下 Azure 入口網站左上角的 [建立新資源] 連結，然後選取 [AI + 機器學習服務] > [Web 應用程式 Bot]。



The screenshot shows the Azure Marketplace interface. On the left, there's a sidebar with categories like Get started, Recently created, Compute, Networking, Storage, Web, Mobile, Containers, Databases, Analytics, AI + Machine Learning (which is highlighted with a red box), Internet of Things, Integration, Security, Identity, Developer Tools, Management Tools, Software as a Service (SaaS), and Blockchain. The main area is titled 'Featured' and contains several service cards. One card for 'Machine Learning service workspace (preview)' has a 'PREVIEW' badge. Another card for 'Data Science Virtual Machine - Windows 2016' includes a 'Quickstart tutorial'. The 'Web App Bot' service card, which is highlighted with a blue box, features a logo of a person icon inside a circle, with 'Quickstart tutorial' text below it. Other cards include 'Computer Vision', 'Face', 'Text Analytics', 'Language Understanding', 'Bing Search v7', and 'Azure Search', each with their respective icons and 'Quickstart tutorial' links.

2. 隨即開啟含有 Web 應用程式 Bot 相關資訊的「新刀鋒視窗」。
3. 在 [Bot 服務] 刀鋒視窗中，依照下圖下方表格中說明的內容，提供有關您 Bot 的必要資訊。

Web App Bot

Bot Service

* Bot name [?](#)

* Subscription

* Resource group
 [▼](#)
[Create new](#)

* Location [?](#)
 [▼](#)

Pricing tier ([View full pricing details](#))
 [▼](#)

* App name [?](#)
 [.azurewebsites.net](#)

* Bot template
 [>](#)

* LUIS App location
 [▼](#)

* App service plan/Location
 [>](#)

Application Insights [?](#)

* Application Insights Location [?](#)
 [▼](#)

Microsoft App ID and password [?](#)
 [>](#)

[Create](#) [Automation options](#)

設定	建議的值	說明
Bot 名稱	您的 Bot 顯示名稱	Bot 在通道和目錄中的顯示名稱。您可以隨時變更此名稱。
訂用帳戶	您的訂用帳戶	選取您要使用的 Azure 訂用帳戶。
資源群組	myResourceGroup	您可以建立新的 資源群組 , 或選擇現有的資源群組。
位置	預設的位置	選取資源群組的地理位置。您可以選擇任何列出的位置, 但通常最佳的選擇是最靠近您客戶的位置。一旦建立 Bot, 就無法變更位置。
定價層	F0	選取定價層。您可以隨時更新定價層。如需詳細資訊, 請參閱 Bot 服務價格 。

設定	建議的值	說明
應用程式名稱	唯一的名稱	Bot 的唯一 URL 名稱。例如，如果您將 Bot 命名為 <i>myawesomebot</i> ，則 Bot 的 URL 將會是 <code>http://myawesomebot.azurewebsites.net</code> 。名稱只能使用英數字元和底線字元。此欄位有 35 個字元的長度限制。一旦建立 Bot，就無法變更應用程式名稱。
Bot 範本	回應 Bot	選擇 [SDK v4]。選取 [C#] 或 [Node.js] 以供本快速入門使用，然後按一下 [選取]。
App Service 方案/位置	您的 App Service 方案	選取 App Service 方案 位置。您可以選擇任何列出的位置，但通常最好是選擇與 Bot 服務相同的位置。
Application Insights	另一	決定您要開啟或關閉 Application Insights 。如果您選取 [開啟]，您也必須指定區域位置。您可以選擇任何列出的位置，但通常最好是選擇與 Bot 服務相同的位置。
Microsoft 應用程式識別碼和密碼	自動建立應用程式識別碼和密碼	如果您需要手動輸入 Microsoft 應用程式識別碼和密碼，請使用此選項。否則，在 Bot 建立程序期間，便會為您建立新的 Microsoft 應用程式識別碼和密碼。

4. 按一下 [建立] 以建立服務，並將 Bot 部署到雲端。此程序可能需要幾分鐘的時間。

勾選 [通知] 來確認已部署 Bot。通知會從 [部署進行中] 變更為 [部署成功]。按一下 [前往資源] 按鈕以開啟 Bot 的資源刀鋒視窗。

現在，既然您已建立 Bot，請在 Web 聊天中測試。

測試 Bot

在 [Bot 管理] 區段中，按一下 [在網路聊天中測試]。Azure Bot Service 會將網路聊天控制項載入，並連線至 Bot。

The screenshot shows the Azure Bot Management interface. On the left, there's a sidebar with various options like Bot management, Build, Test in Web Chat (which is highlighted with a red box), Analytics, Channels, Settings, Speech priming, Bot Service pricing, App Service Settings, Application Settings, All App service settings, Support + troubleshooting, and New support request. The main right pane shows a conversation window. It starts with a message 'conversationUpdate event detected' from 'sdlkv4bot'. Then, a message 'Hi' is sent from 'You'. Below that, another message 'conversationUpdate event detected' is shown from 'sdlkv4bot'. At the bottom, a message 'Turn 1: You sent 'Hi'' is displayed, along with the timestamp 'sdlkv4bot at 2:06:13 PM'. A text input field at the bottom says 'Type your message...'.

輸入訊息，您的 Bot 應會回應。

下載程式碼

您可以下載程式碼，以在本機上處理。

1. 在 [Bot 管理] 區段中，按一下 [組建]。
2. 在右窗格中按一下**下載 Bot 原始程式碼連結**。
3. 遵循提示以下載程式碼，然後將資料夾解壓縮。
 - a. 下載 Bot 時，您可以選擇在下載中包含 Bot 的設定 (包含金鑰和祕密)，Bot 可能需要這些項目才能正常運作。如果您選擇 [是]，則 `appsettings.json` 或 `.env` 檔案會具有金鑰。

後續步驟

下載程式碼之後，您可以繼續在電腦本機開發 Bot。當您測試 Bot 並準備將 Bot 程式碼上傳至 Azure 入口網站時，請遵循[設定持續部署](#)底下所列的指示，以在進行變更後自動更新程式碼。

使用適用於 .NET 的 Bot Framework SDK 建立 Bot

2019/4/26 • [Edit Online](#)

適用於： SDK v4 SDK v3

本快速入門會逐步引導您藉由使用 C# 範本，然後使用 Bot Framework Emulator 來測試，以建置 Bot。

使用 Azure Bot Service 建立 Bot 與在本機建立 Bot 是建立 Bot 的獨立、平行方式。

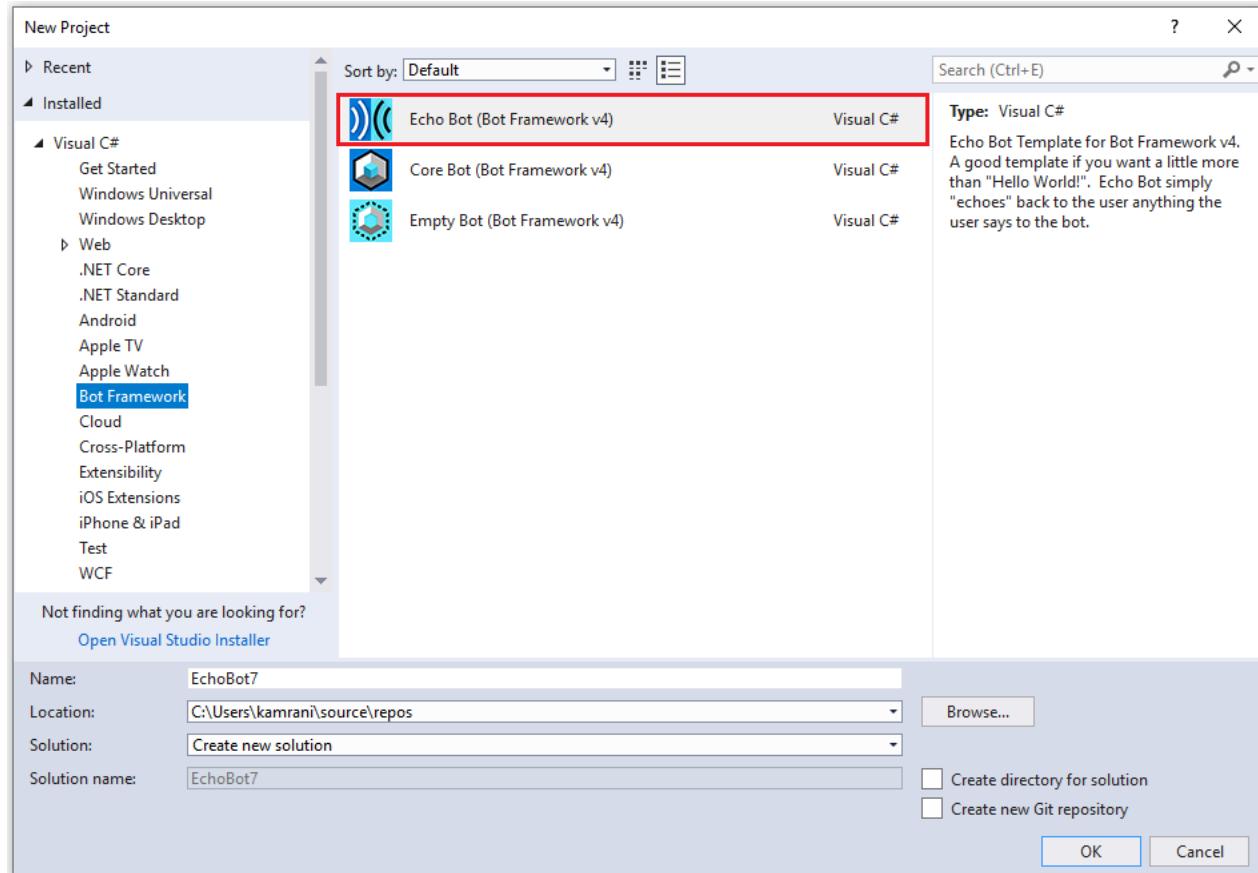
必要條件

- Visual Studio 2017 或更新版本
- 適用於 C# (<https://aka.ms/bot-vsix>) 的 Bot Framework SDK v4 [範本]
- [Bot Framework 模擬器 \(英文\)](#)
- 了解 [ASP.NET Core](#) 和 [C# 中的非同步程式設計](#)

建立 Bot

安裝您在必要條件區段下載的 BotBuilderVSIX.vsix 範本。

在 Visual Studio 中，使用 **Echo Bot (Bot Framework v4)** 範本建立新的 Bot 專案。



TIP

如有需要，請將專案組建類型變更為 [.Net Core 2.1](#)。此外，視需要更新 [Microsoft.Bot.Builder](#) NuGet 套件。

由於有範本，專案中會包含要在本快速入門建立 Bot 所需的所有程式碼。您實際上不需要撰寫任何額外的程式碼。

在 Visual Studio 中啟動 Bot

當您按一下執行按鈕時, Visual Studio 將會建置應用程式、將其部署到 localhost, 並啟動 Web 瀏覽器來顯示應用程式的 `default.htm` 頁面。此時, Bot 正在本機執行。

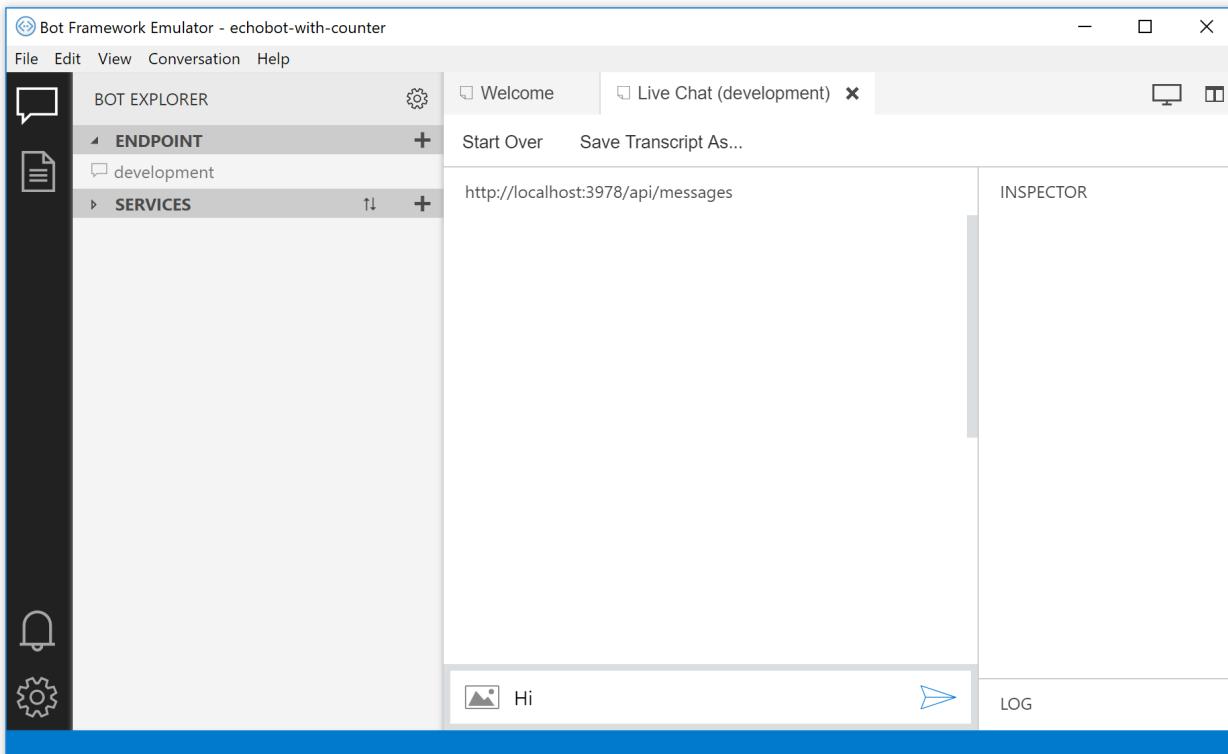
啟動模擬器並且連線至您的 Bot

接下來, 請啟動模擬器, 然後在模擬器中連線至您的 Bot:

1. 按一下模擬器 [歡迎使用] 索引標籤中的 [建立新的 Bot 設定] 連結。
2. 填寫 Bot 的欄位, 然後按一下 [儲存並連接]。

與您的 Bot 互動

傳送訊息給 Bot, Bot 就會以訊息回應。



NOTE

如果您發現訊息無法傳送, 則可能需要重新啟動電腦, 因為 ngrok 尚未在您的系統上取得所需的權限 (只需要進行一次)。

其他資源

請參閱[通道 \(ngrok\)](#), 了解如何連線到遠端裝載的 Bot。

後續步驟

[將 Bot 部署至 Azure](#)

使用適用於 JavaScript 的 Bot Framework SDK 建立 Bot

2019/4/26 • [Edit Online](#)

適用於:  SDK v4  SDK v3

本快速入門會逐步引導您使用 Yeoman Bot Builder 產生器和適用於 JavaScript 的 Bot Framework SDK 建置單一 Bot，然後使用 Bot Framework 模擬器進行測試。

使用 Azure Bot Service 建立 Bot 與在本機建立 Bot 是建立 Bot 的獨立、平行方式。

必要條件

- [Visual Studio Code](#)
- [Node.js](#)
- [Yeoman](#), 可使用產生器為您建立 Bot
- [git](#)
- [Bot Framework 模擬器 \(英文\)](#)
- 了解 [restify](#) 和 JavaScript 中的非同步程式設計

NOTE

只有在您使用 Windows 作為開發作業系統時，才需要安裝以下所列的 Windows 建置工具。在某些安裝中，[restify](#) 的安裝步驟會產生與 node-gyp 相關的錯誤。如果情況如此，您可嘗試以提升的權限執行此命令。如果您的系統上已安裝 python，此呼叫可能也會停止回應，但不會結束：

```
npm install -g windows-build-tools
```

建立 Bot

建立 Bot 並初始化其套件

1. 開啟終端機或提升權限的命令提示字元。
2. 如果您的 JavaScript Bot 還沒有目錄，請加以建立並切換到該目錄。(即使我們在本教學課程中只建立一個 Bot，但是在一般情況下，我們會為 JavaScript Bot 建立目錄。)

```
mkdir myJsBots  
cd myJsBots
```

3. 確保您的 npm 版本是最新版本。

```
npm install -g npm
```

4. 接下來，安裝適用於 JavaScript 的 Yeoman 和產生器。

```
npm install -g yo generator-botbuilder
```

5. 然後，使用產生器來建立 echo Bot。

```
yo botbuilder
```

Yeoman 會提示您輸入一些用來建立 Bot 的資訊。本教學課程使用預設值。

- 輸入 Bot 的名稱。(myChatBot)
- 輸入描述。(示範 Microsoft Bot Framework 的核心功能)
- 選擇 Bot 的語言。(JavaScript)
- 選擇要使用的範本。(Echo)

由於有範本，專案中會包含要在本快速入門建立 Bot 所需的所有程式碼。您實際上不需要撰寫任何額外的程式碼。

NOTE

如果選擇建立 **Basic** Bot，您需要 LUIS 語言模型。您可以在 [luis.ai](#) 上建立一個模型。建立模型之後，請更新設定檔。

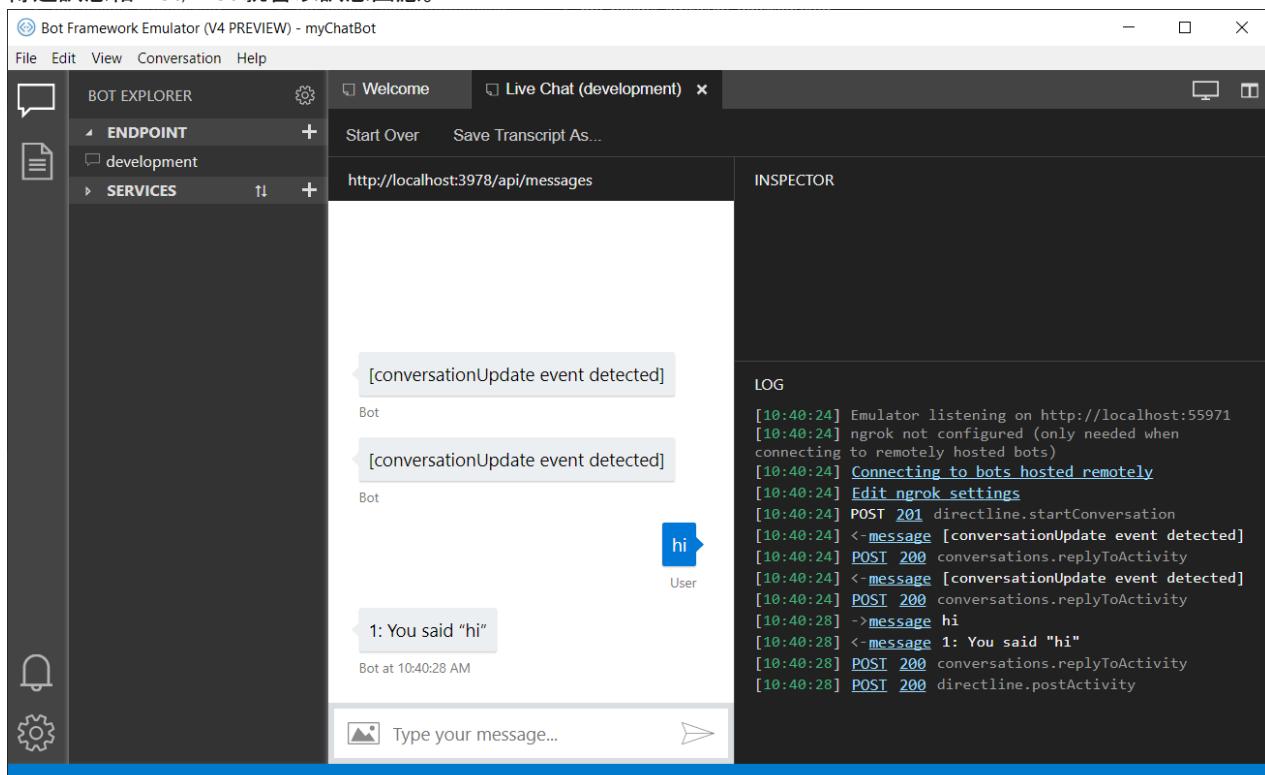
啟動 Bot

在終端機或命令提示字元中，將目錄變更為針對您的 Bot 所建立的目錄，並以 `npm start` 開頭。此時，Bot 正在本機執行。

啟動模擬器並連線至您的 Bot

1. 啟動 Bot Framework 模擬器。
2. 按一下模擬器 [歡迎使用] 索引標籤中的 [建立新的 Bot 設定] 連結。
3. 填寫 Bot 的欄位，然後按一下 [儲存並連接]。

傳送訊息給 Bot，Bot 就會以訊息回應。



其他資源

請參閱[通道 \(ngrok\)](#)，了解如何連線到遠端裝載的 Bot。

後續步驟

[將 Bot 部署至 Azure](#)

教學課程: 建立及部署基本 Bot

2019/5/10 • [Edit Online](#)

適用於:  SDK v4  SDK v3

本教學課程會引導您使用 Bot Framework SDK 建立基本 Bot，然後將其部署至 Azure。如果您已經建立基本 Bot 並且在本機執行，請直接跳到[部署您的 Bot](#)一節。

在本教學課程中，您了解如何：

- 建立基本 Echo Bot
- 在本機執行並與其互動
- 發佈您的 Bot

如果您沒有 Azure 訂用帳戶，請在開始前建立[免費帳戶](#)。

- C#
- JavaScript

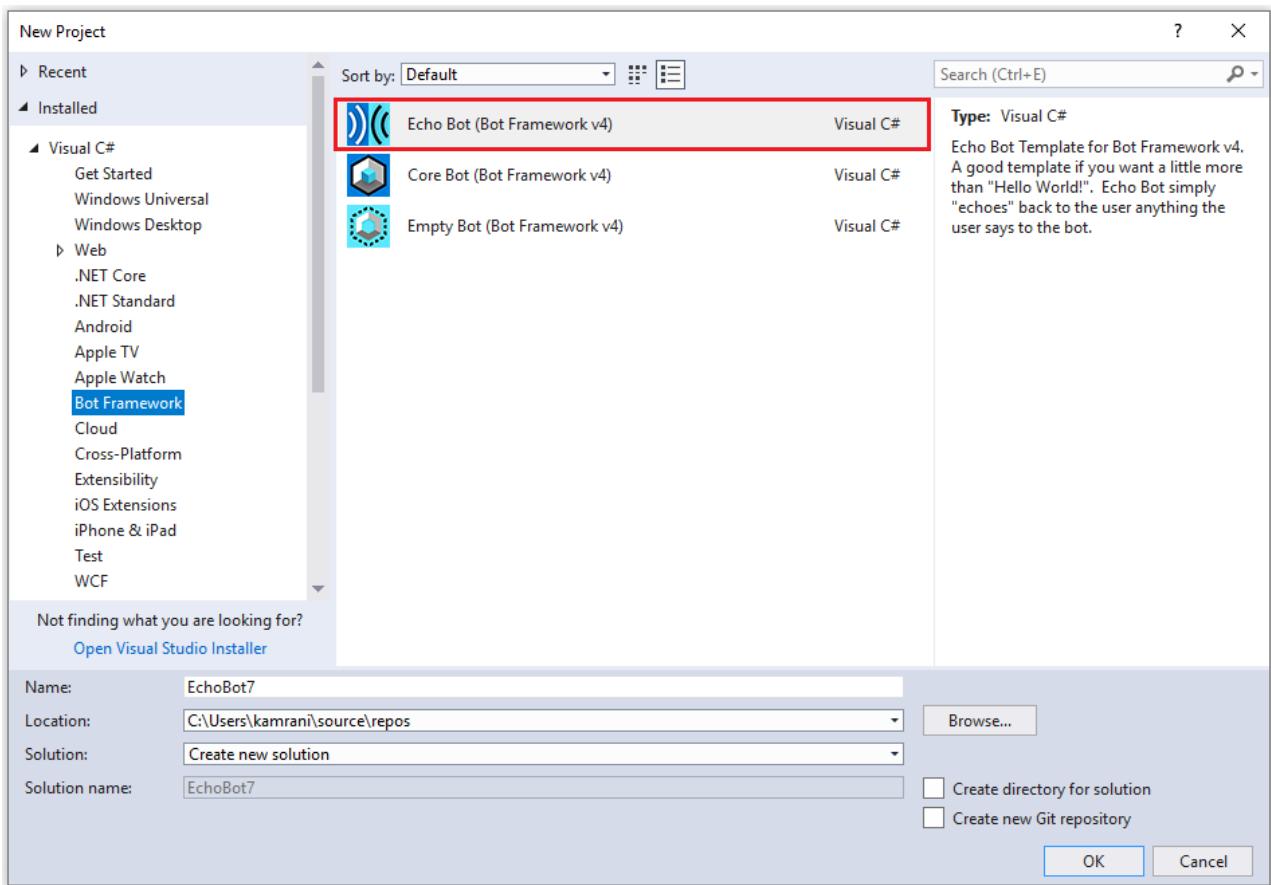
必要條件

- Visual Studio 2017 或更新版本
- 適用於 C#](<https://aka.ms/bot-vsix>) 的 Bot Framework SDK v4 [範本
- [Bot Framework 模擬器](#) (英文)
- 了解 [ASP.NET Core](#) 和 C# 中的非同步程式設計

建立 Bot

安裝您在必要條件區段下載的 BotBuilderVSIX.vsix 範本。

在 Visual Studio 中，使用 **Echo Bot (Bot Framework v4)** 範本建立新的 Bot 專案。



TIP

如有需要，請將專案組建類型變更為 `.Net Core 2.1`。此外，視需要更新 `Microsoft.Bot.Builder` NuGet 套件。

由於有範本，專案中會包含要在本快速入門建立 Bot 所需的所有程式碼。您實際上不需要撰寫任何額外的程式碼。

在 Visual Studio 中啟動 Bot

當您按一下執行按鈕時，Visual Studio 將會建置應用程式、將其部署到 localhost，並啟動 Web 瀏覽器來顯示應用程式的 `default.htm` 頁面。此時，Bot 正在本機執行。

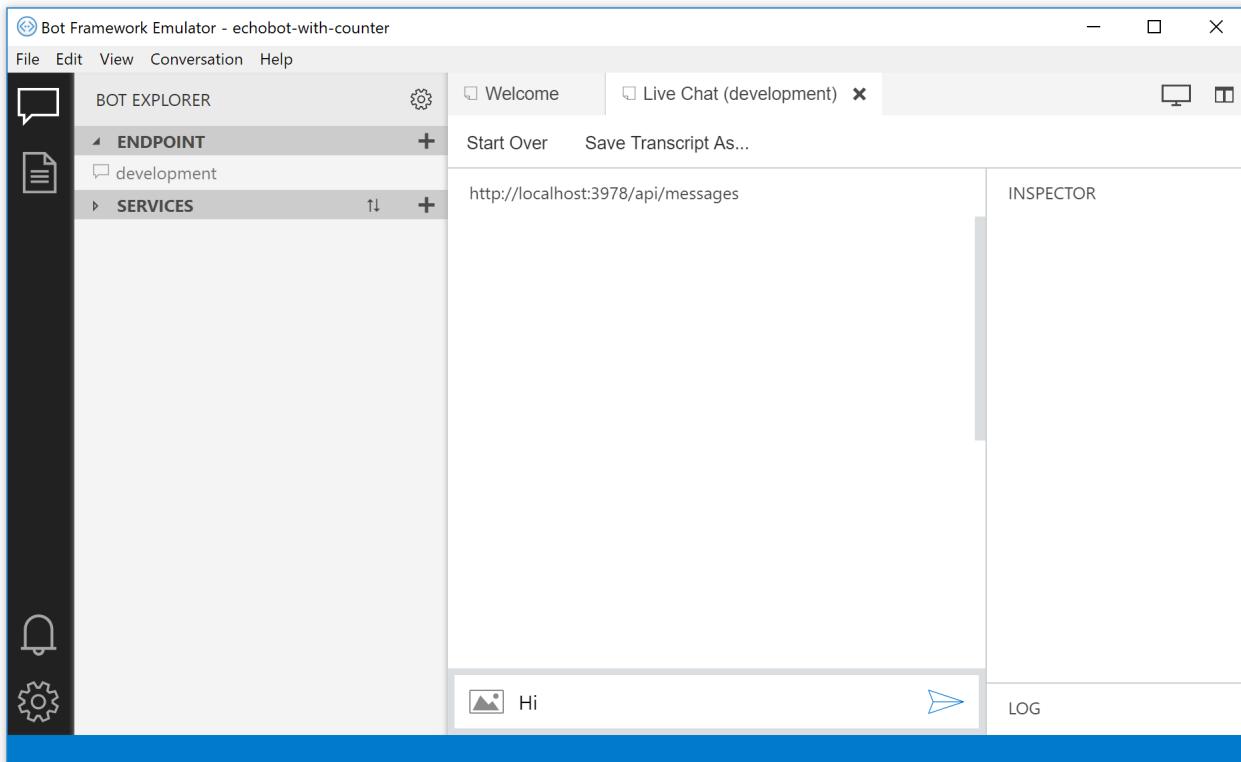
啟動模擬器並且連線至您的 Bot

接下來，請啟動模擬器，然後在模擬器中連線至您的 Bot：

1. 按一下模擬器 [歡迎使用] 索引標籤中的 [建立新的 Bot 設定] 連結。
2. 填寫 Bot 的欄位，然後按一下 [儲存並連接]。

與您的 Bot 互動

傳送訊息給 Bot，Bot 就會以訊息回應。



NOTE

如果您發現訊息無法傳送，則可能需要重新啟動電腦，因為 ngrok 尚未在您的系統上取得所需的權限（只需要進行一次）。

部署您的 Bot

必要條件

- 如果您沒有 Azure 訂用帳戶，請在開始前建立 [免費帳戶](#)。
- 在您本機電腦上執行的上述 Bot。
- 最新版的 [Azure CLI](#)。

1.準備部署

當您使用 Visual Studio 或 Yeoman 範本建立 Bot 時，所產生的原始程式碼會包含 `deploymentTemplates` 資料夾與 ARM 範本。此處敘述的部署程序會使用 ARM 範本，然後以 Azure CLI 在 Azure 中佈建所需的 Bot 資源。

登入 Azure

您已在本機建立及測試 Bot，而現在想要將其部署至 Azure。開啟命令提示字元以登入 Azure 入口網站。

```
az login
```

瀏覽器視窗會隨即開啟，以便您登入。

設定訂用帳戶

設定要使用的預設訂用帳戶。

```
az account set --subscription "<azure-subscription>"
```

如果您不確定哪個訂用帳戶要用於部署 Bot，您可以使用 `az account list` 命令，檢視您帳戶的訂用帳戶清單。瀏覽至 Bot 資料夾。

建立應用程式註冊

註冊應用程式意謂著您可以使用 Azure AD 來驗證使用者，以及要求存取使用者資源。在 Azure 中，您的 Bot 需要已註冊的應用程式來允許 Bot 存取 Bot Framework 服務，以便傳送和接收已驗證的訊息。若要透過 Azure CLI 建立註冊應用程式，請執行下列命令：

```
az ad app create --display-name "displayName" --password "AtLeastSixteenCharacters_0" --available-to-other-tenants
```

選項	說明
display-name	應用程式的顯示名稱。
password	應用程式密碼，也稱為「用戶端密碼」。密碼必須至少有 16 個字元長，至少包含 1 個大寫或小寫字母，以及至少包含 1 個特殊字元。
available-to-other-tenants	應用程式可以從任何 Azure AD 租用戶使用。允許的值：false、true。預設為 true。這會讓您的 Bot 使用 Azure Bot 服務通道。

上述命令會輸出具有 `appId` 索引鍵的 JSON，請儲存此索引鍵的值以用於 ARM 部署，該值會用在 `appId` 參數上。提供的密碼將會用於 `appSecret` 參數。

您可以將 Bot 部署在新的資源群組或現有的資源群組中。選擇最適合您的選項。

- [透過 ARM 範本部署 \(使用新資源群組\)](#)
- [透過 ARM 範本部署 \(使用現有的資源群組\)](#)

建立 Azure 資源

您會在 Azure 中建立新的資源群組，然後使用 ARM 範本在其中建立指定資源。在此案例中，我們提供 App Service 方案、Web 應用程式和 Bot 通道註冊。

```
az deployment create --name "<name-of-deployment>" --template-file "template-with-new-rg.json" --location "location-name" --parameters appId=<msa-app-guid> appSecret=<msa-app-password> botId=<id-or-name-of-bot> botSku=F0 newAppServicePlanName=<name-of-app-service-plan> newWebAppName=<name-of-web-app> groupName=<new-group-name> groupLocation=<location> newAppServicePlanLocation=<location>"
```

選項	說明
name	部署的自訂名稱。
template-file	ARM 範本的路徑。您可以使用專案資料夾 <code>deploymentTemplates</code> 中提供的 <code>template-with-new-rg.json</code> 檔案。
location	位置。值的來源： <code>az account list-locations</code> 。您可以使用 <code>az configure --defaults location=<location></code> 來設定預設位置。

選項	說明
parameters	<p>提供部署參數值。執行 <code>az ad app create</code> 命令時所得的 <code>appId</code> 值。<code>appSecret</code> 是您在上一個步驟中提供的密碼。<code>botId</code> 參數應該是全域唯一，而且會用來作為不可變的 Bot 識別碼。此參數也會用來設定 Bot 的顯示名稱，而這是可變動的。<code>botSku</code> 是定價層，可以是 F0 (免費) 或 S1 (標準)。<code>newAppServicePlanName</code> 是 App Service 方案的名稱。<code>newWebAppName</code> 是您要建立的 Web 應用程式名稱。<code>groupName</code> 是您要建立的 Azure 資源群組名稱。<code>groupLocation</code> 是 Azure 資源群組的位置。<code>newAppServicePlanLocation</code> 是 App Service 方案的位置。</p>

擷取或建立必要的 IIS/Kudu 檔案

針對 C# Bot

```
az bot prepare-deploy --lang Csharp --code-dir "." --proj-file-path "MyBot.csproj"
```

您必須提供與 `--code-dir` 對應的 `.csproj` 檔案路徑。這可以透過 `--proj-file-path` 引數來執行。此命令會將 `--code-dir` 和 `--proj-file-path` 解析為 `./MyBot.csproj`

針對 JavaScript Bot

```
az bot prepare-deploy --code-dir "." --lang Javascript
```

此命令會擷取 `web.config`，這是讓 Node.js 應用程式可以在 Azure App Service 上與 IIS 搭配運作的必要項目。請確定 `web.config` 已儲存到您 Bot 的根目錄。

手動壓縮程式碼目錄

使用未設定的 [zip 部署 API](#) 來部署您 Bot 的程式碼時，Web 應用程式/Kudu 的行為會如下所示：

根據預設，Kudu 會假設來自 zip 檔案的部署都已可供執行，而且部署期間不需要額外的建置步驟，例如 `npm install` 或 `dotnet restore/dotnet publish`。

因此，務必在要部署至 Web 應用程式的 zip 檔案中包含建置程式碼和所有必要相依性，否則您的 Bot 不會如預期般運作。

IMPORTANT

壓縮專案檔之前，請確定您位在正確的資料夾。

- 針對 C# Bot，這是具有 `.csproj` 檔案的資料夾。
- 針對 JS Bot，這是具有 `app.js` 或 `index.js` 檔案的資料夾。

如果您的根資料夾位置不正確，**Bot 將無法在 Azure 入口網站中執行**。

2. 將程式碼部署至 Azure

此時，我們已準備好將程式碼部署至 Azure Web 應用程式。從命令列執行下列命令，即可對 Web 應用程式使用 kudu zip 推送部署來執行部署。

```
az webapp deployment source config-zip --resource-group "<new-group-name>" --name "<name-of-web-app>" --src "code.zip"
```

選項	說明
resource-group	您先前在 Azure 中建立的資源群組名稱。
name	您稍早使用的 Web 應用程式名稱。
src	您所建立的 zip 檔案路徑。

3. 在網路聊天中測試

- 在 Azure 入口網站中，移至 Web 應用程式 Bot 的刀鋒視窗。
- 在 [Bot 管理] 區段中，按一下 [在網絡聊天中測試]。Azure Bot Service 會將網路聊天控制項載入，並連線至 Bot。
- 在成功部署後等候幾秒，選擇性地重新啟動您的 Web 應用程式，以清除任何快取。回到您的 Web 應用程式 Bot 刀鋒視窗，使用 Azure 入口網站中提供的網路聊天進行測試。

其他資源

當您部署 Bot 時，這些資源通常會建立於 Azure 入口網站中：

資源	說明
Web 應用程式 Bot	已部署至 Azure App Service 的 Azure Bot Service Bot。
App Service	可讓您建置及裝載 Web 應用程式。
App Service 計劃	針對要執行的 Web 應用程式定義一組計算資源。
Application Insights	提供用來收集和分析遙測的工具。
儲存體帳戶	提供高可用性、安全、耐用、可擴充和備援性雲端儲存體。

若要查看 `az bot` 命令的文件，請參閱[參考](#)主題。

如果您不熟悉 Azure 資源群組，請參閱這個[術語](#)主題。

後續步驟

[在 Bot 中使用 QnA Maker 回答問題](#)

教學課程：在 Bot 中使用 QnA Maker 回答問題

2019/5/14 • [Edit Online](#)

適用於： SDK v4  SDK v3

您可以使用 QnA Maker 服務和知識庫，對 Bot 新增問與答支援。當您建立知識庫時，您會在其中植入問題與答案。

在本教學課程中，您了解如何：

- 建立 QnA Maker 服務和知識庫
- 將知識庫資訊新增至設定檔
- 更新 Bot 以查詢知識庫
- 重新發佈 Bot

如果您沒有 Azure 訂用帳戶，請在開始前建立[免費帳戶](#)。

必要條件

- 在[上一個教學課程](#)中建立的 Bot。我們會將問與答功能新增至 Bot。
- 熟悉一下 [QnA Maker](#) 很有幫助。我們將使用 QnA Maker 入口網站來建立、訓練及發佈要搭配 Bot 使用的知識庫。
- 熟悉如何使用 Azure Bot 服務來[建立 QnA Bot](#)。

您應該也已具備上一個教學課程的必要條件。

登入 QnA Maker 入口網站

使用您的 Azure 認證登入 [QnA Maker 入口網站](#)。

建立 QnA Maker 服務和知識庫

我們將從 [Microsoft/BotBuilder-Samples](#) 存放庫中的 QnA Maker 範例匯入現有的知識庫定義。

1. 將範例存放庫複製到您的電腦。
2. 在 QnA Maker 入口網站中，[建立知識庫](#)。
 - a. 如有必要，請建立 QnA 服務。(您可以在此教學課程中使用現有的 QnA Maker 服務或建立新的服務)。如需詳細的 QnA Maker 指示，請參閱[建立 QnA Maker 服務和建立、訓練及發佈 QnA Maker 知識庫](#)。
 - b. 將 QnA 服務連線到知識庫。
 - c. 為您的知識庫命名。
 - d. 若要填入您的知識庫，請使用範例存放庫中的 **BotBuilder-Samples\samples\csharp_dotnetcore\11.qnamaker\CognitiveModels\smartLightFAQ.tsv** 檔案。
 - e. 按一下 [建立知識庫] 以建立知識庫。
3. [儲存並訓練知識庫](#)。
4. [發佈知識庫](#)。

您的知識庫現在可供您的 Bot 使用。記錄知識庫識別碼、端點金鑰和主機名稱。您在下一個步驟中需要這些資訊。

將知識庫資訊新增至 Bot

從 Bot Framework v4.3 開始，Azure 不會再於您下載的 Bot 原始程式碼中提供 .bot 檔案。使用下列指示將 CSharp

或 JavaScript Bot 連線到您的知識庫。

- [C#](#)
- [JavaScript](#)

在 appsetting.json 檔案中新增下列值：

```
{  
    "MicrosoftAppId": "",  
    "MicrosoftAppPassword": "",  
    "ScmType": "None",  
  
    "QnAKnowledgebaseId": "<your-knowledge-base-id>",  
    "QnAAuthKey": "<your-knowledge-base-endpoint-key>",  
    "QnAEndpointHostName": "<your-qna-service-hostname>" // This is a URL  
}
```

欄位	值
kblid	QnA Maker 入口網站為您產生的知識庫識別碼。
endpointKey	QnA Maker 入口網站為您產生的端點金鑰。
主機名稱	QnA Maker 入口網站所產生的主機 URL。使用完整的 URL，其開頭為 https:// 並以 /qnamaker 結尾。完整的 URL 字串看起來會類似 "https://< >.azure.net/qnamaker"。

現在儲存您的編輯。

更新 Bot 以查詢知識庫

更新您的初始化程式碼，以載入您知識庫的服務資訊。

- [C#](#)
- [JavaScript](#)

1. 將 **Microsoft.Bot.Builder.AI.QnA** NuGet 套件新增至專案。
2. 將 **Microsoft.Extensions.Configuration** NuGet 套件新增至專案。
3. 在 **Startup.cs** 檔案中，新增這些命名空間參考。

Startup.cs

```
using Microsoft.Bot.Builder.AI.QnA;  
using Microsoft.Extensions.Configuration;
```

4. 然後修改 *ConfigureServices* 方法並建立 QnAMakerEndpoint，其會連線到 **appsettings.json** 檔案中定義的知識庫。

Startup.cs

```
// Create QnAMaker endpoint as a singleton
services.AddSingleton(new QnAMakerEndpoint
{
    KnowledgeBaseId = Configuration.GetValue<string>($"QnAKnowledgebaseId"),
    EndpointKey = Configuration.GetValue<string>($"QnAAuthKey"),
    Host = Configuration.GetValue<string>($"QnAEndpointHostName")
});
```

5. 在 **EchoBot.cs** 檔案中，新增下列命名空間參考。

EchoBot.cs

```
using System.Linq;
using Microsoft.Bot.Builder.AI.QnA;
```

6. 新增 **EchoBotQnA** 連接器並在 Bot 的建構函式中加以初始化。

EchoBot.cs

```
public QnAMaker EchoBotQnA { get; private set; }
public EchoBot(QnAMakerEndpoint endpoint)
{
    // connects to QnA Maker endpoint for each turn
    EchoBotQnA = new QnAMaker(endpoint);
}
```

7. 新增下列程式碼，以在 *OnMembersAddedAsync()* 方法下方建立 *AccessQnAMaker()* 方法：

EchoBot.cs

```
private async Task AccessQnAMaker(ITurnContext<IMessageActivity> turnContext, CancellationToken cancellationToken)
{
    var results = await EchoBotQnA.GetAnswersAsync(turnContext);
    if (results.Any())
    {
        await turnContext.SendActivityAsync(MessageFactory.Text("QnA Maker Returned: " +
results.First().Answer), cancellationToken);
    }
    else
    {
        await turnContext.SendActivityAsync(MessageFactory.Text("Sorry, could not find an answer in the Q
and A system."), cancellationToken);
    }
}
```

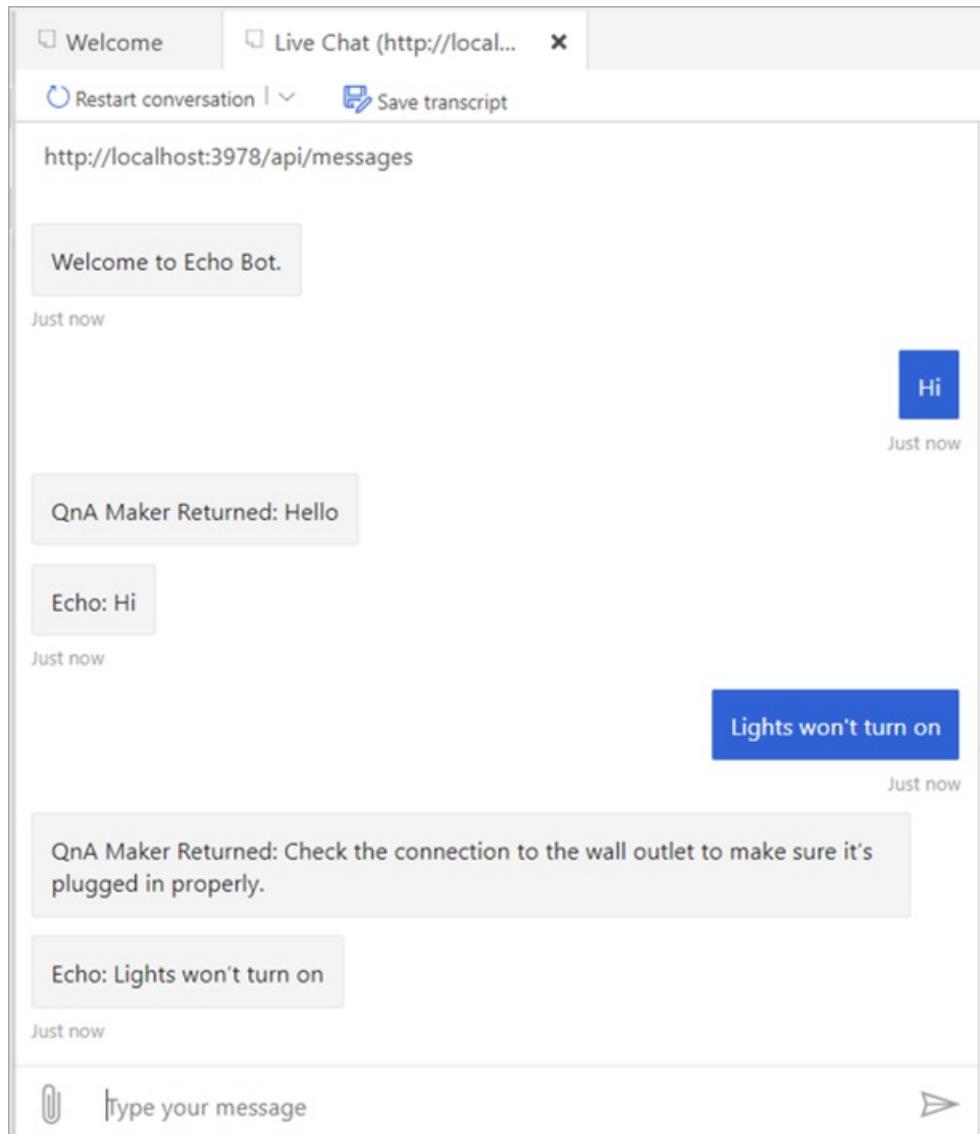
8. 現在在 *OnMessageActivityAsync()* 內呼叫您的新方法 *AccessQnAMaker()*，如下所示：

EchoBot.cs

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    // First send the user input to your QnA Maker knowledgebase
    await AccessQnAMaker(turnContext, cancellationToken);
    ...
}
```

在本機測試 Bot

此時您的 Bot 應該能夠回答一些問題。在本機執行 Bot 並且在模擬器中開啟。



重新發佈 Bot

我們現在可以將您的 Bot 重新發佈回到 Azure。

- [C#](#)
- [JavaScript](#)

```
az webapp deployment source config-zip --resource-group <resource-group-name> --name <bot-name-in-azure> --src "c:\bot\mybot.zip"
```

測試已發佈的 Bot

發佈 Bot 之後，讓 Azure 有幾分鐘的時間來更新並啟動 Bot。

使用模擬器來測試 Bot 的生產端點，或使用 Azure 入口網站在網路聊天中測試 Bot。在任一情況下，您應會看見如同在本機測試時的相同行為。

清除資源

如果您不打算繼續使用此應用程式，請使用下列步驟來刪除相關聯的資源：

1. 在 Azure 入口網站中，開啟您 Bot 的資源群組。

2. 按一下 [刪除資源群組] 來刪除群組及其包含的所有資源。
3. 在確認窗格中，輸入資源群組名稱，然後按一下 [刪除]。

後續步驟

如需如何將功能新增至 Bot 的相關資訊，請參閱開發操作說明一節中的**傳送及接收文字訊息**一文和其他文章。

[傳送及接收文字訊息](#)

透過 Azure Bot 服務將驗證新增至您的 Bot

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

此教學課程將使用 Azure Bot 服務中的全新 Bot 驗證功能，並提供相關功能，讓您輕鬆開發可針對 Azure AD (Azure Active Directory)、GitHub、Uber 等不同身分識別提供者驗證使用者的 Bot。此外，這些更新也排除了部分用戶端採用的_神奇代碼驗證 (Magic code verification)_，可進一步改善使用者體驗。

在此之前，您的 Bot 必須加入 OAuth 控制器和登入連結、儲存目標用戶端識別碼及密碼，並執行使用者權杖管理作業。

現在，Bot 開發人員無須際遇主控 OAuth 控制器或管理權杖的生命週期，這些工作現在全都可由 Azure Bot 服務執行。

這些功能包括：

- 改善通道以支援全新驗證功能 (例如新的 WebChat 和 DirectLineJS 程式庫)，無須再使用 6 位數神奇代碼驗證方式。
- 改善 Azure 入口網站，以針對不同的 OAuth 身分識別提供者新增、刪除和配置連線設定。
- 支援各種現成的身分識別提供者服務，例如：Azure AD (包括 v1 和 v2 端點)、GitHub 等等。
- 更新 C# 和 Node.js Bot Framework SDK，以支援擷取權杖、建立 OAuthCard 及處理 TokenResponse 事件。
- 如何建立能針對 Azure AD (包括 v1 和 v2 端點) 和 GitHub 進行驗證之 Bot 的範例。

您可從本文中的步驟推測如何將這類功能新增至現有的 Bot。以下是展示全新驗證功能的範例 Bot

範例	BOTBUILDER 版本	說明
AadV1Bot	v3	展使用 Azure AD v1 端點示 v3 C# SDK 中的 OAuthCard 支援
AadV2Bot	v3	使用 Azure AD v2 端點展示 v3 C# SDK 中的 OAuthCard 支援
GitHubBot	v3	使用 GitHub 展示 v3 C# SDK 中的 OAuthCard 支援
BasicOAuth	v3	展示 v3 C# SDK 中的 OAuth 2.0 支援

NOTE

驗證功能亦可搭配 Node.js 和 BotBuilder v3 使用。不過，本文內容僅涵蓋範例 C# 程式碼。

如需其他資訊和支援，請參閱 [Bot Framework 其他資源](#)。

概觀

本教學課程使用 Azure AD v1 或 v2 權杖建立了一個連線至 Microsoft Graph 的範例 Bot。而在此程序中，您將使用

來自 GitHub 存放庫的程式碼；本教學課程將說明如何進行設定，包括 Bot 應用程式。

- [建立 Bot 和驗證應用程式](#)
- [準備 Bot 範例程式碼](#)
- [使用模擬器測試您的 Bot](#)

若要完成這些步驟，您必須先行安裝 Visual Studio 2017、npm、節點和 git。此外，您也應熟悉 Azure、OAuth 2.0 及 Bot 開發流程。

完成後，您就有一個 Bot 能針對 Azure AD 應用程式回應幾項簡單工作，例如：檢查及傳送電子郵件，或者顯示您或管理員的身份。您的 Bot 必須針對 Microsoft.Graph 程式庫使用來自 Azure AD 應用程式的權杖，才能達到上述目的。

最後章節會詳細說明部分 Bot 程式碼

- [權杖擷取流程注意事項](#)

建立 Bot 和驗證應用程式

您必須建立註冊 Bot，其中您將設定傳訊端點至部署的 Bot 程式碼，以及建立 Azure AD (v1 或 v2) 應用程式以允許 Bot 存取 Office 365。

NOTE

這些驗證功能皆可搭配其他類型的 Bot 使用。不過，本教學課程所用的範例是僅具有註冊功能的 Bot。

在 Azure AD 中註冊應用程式

您需要一個可讓 Bot 用來連線至 Microsoft Graph API 的 Azure AD 應用程式，以及專屬的 Azure AD 保護資源等等。

針對此 Bot，您可使用 Azure AD v1 或 v2 端點。如需 v1 和 v2 端點之間差異的相關資訊，請參閱 [v1 與 v2 比較](#)，以及 [Azure AD v2.0 端點概觀](#)。

建立 Azure AD v1 應用程式

1. 移至 [Azure 入口網站的 Azure AD](#)。
2. 按一下 [應用程式註冊]。
3. 在 [應用程式註冊] 面板中，按一下 [新增應用程式註冊]。
4. 填寫必要欄位並建立應用程式註冊。
 - a. 為您的應用程式命名。
 - b. 將 [應用程式類型] 設為 [Web 應用程式/API]。
 - c. 將 [登入 URL] 設為 `https://token.botframework.com/.auth/web/redirect`。
 - d. 按一下頁面底部的 [新增]。
 - 建立後，即會顯示在 [註冊的應用程式] 窗格中。
 - 記錄 [應用程式識別碼] 值。您稍後需要提供此值做為 _用戶端識別碼_。
5. 按一下 [設定] 以設定應用程式。
6. 按一下 [金鑰] 以開啟 [金鑰] 面板。
 - a. 在 [密碼] 下方建立 `BotLogin` 金鑰。
 - b. 將其 [持續時間] 設為 [永不過期]。
 - c. 按一下 [儲存] 並記錄金鑰值。您稍後需要提供此值做為 _應用程式密碼_。
 - d. 關閉 [金鑰] 面板。
7. 按一下 [必要權限] 以開啟 [必要權限] 面板。
 - a. 按一下 [新增]。

b. 按一下 [選取 API]，然後選取 [Microsoft Graph]，再按一下 [選取]。

c. 按一下 [選取權限]。選擇您要讓應用程式使用的應用程式權限。

NOTE

只要是標記為 [需要系統管理員] 的權限，則使用者和租用戶管理員皆必須登入，才能將 Bot 隔離在這些權限之外。

選取下列 Microsoft Graph 委派的權限：

- 讀取所有使用者的基本個人資料
 - 讀取使用者的郵件
 - 登入及讀取使用者設定檔
 - 以使用者身分傳送郵件
 - 檢視所有使用者的基本個人資料
 - 檢視使用者的電子郵件地址
- d. 按一下 [選取]，然後按一下 [完成]。
- e. 關閉 [必要權限] 面板。

您現在已完成 Azure AD v1 應用程式設定。

建立 Azure AD v2 應用程式

1. 移至 [Microsoft 應用程式註冊入口網站](#)。

2. 按一下 [新增應用程式]

3. 為您的 Azure AD 應用程式輸入名稱，然後按一下 [建立]。

記錄應用程式識別碼的 GUID。您稍後需要提供此 GUID 做為連線設定的用戶端識別碼。

4. 在 [應用程式密碼] 下，按一下 [產生新密碼]。

記錄快顯視窗中的密碼。您稍後需要提供此密碼做為連線設定的用戶端密碼。

5. 按一下 [平台] 下方的 [新增平台]。

6. 在 [新增平台] 快顯視窗中，按一下 [Web]。

a. 將 [允許隱含流程] 保持勾選。

b. 在 [重新導向 URL] 中輸入 `https://token.botframework.com/.auth/web/redirect`。

c. 將 [登出 URL] 欄位保留空白。

7. 在 [Microsoft Graph 權限] 下方，您可新增其他委派的權限。

- 在本教學課程中，新增的是 **Mail.Read**、**Mail.Send**、**openid**、**profile**、**User.Read** 和 **User.ReadBasic.All** 權限。連線設定的範圍需要使用 **openid** 和 Azure AD Graph 中的資源，例如：**Mail.Read**。
- 記錄您選擇的權限。您稍後需要提供此權限做為連線設定的範圍。

8. 按一下頁面底部的 [儲存]。

在 Azure 建立 Bot

使用 [Azure 入口網站](#)建立 Bot 通道註冊。

向 Bot 註冊 Azure AD 應用程式

下一步是向 Bot 註冊您剛剛建立的 Azure AD 應用程式。

註冊 Azure AD v1 應用程式

1. 在 [Azure 入口網站](#) 瀏覽至 Bot 的資源頁面。
2. 按一下 [設定]。
3. 在靠近頁面底部的 [OAuth 連線設定] 下方，按一下 [新增設定]。
4. 填寫表單，如下所示：
 - a. 若為 [名稱] 欄位，請輸入連線的名稱。Bot 程式碼中會用到此名稱。
 - b. 若為 [服務提供者]，請選取 [Azure Active Directory]。選取此項目後，Azure AD 專用的欄位隨即顯示。
 - c. 若為 [用戶端識別碼] 欄位，請輸入您為 Azure AD v1 應用程式記錄的應用程式識別碼。
 - d. 若為 [用戶端密碼] 欄位，請輸入您為應用程式 `BotLogin` 金鑰記錄的金鑰。
 - e. 若為 [授與類型] 欄位，請輸入「`authorization_code`」。
 - f. 若為 [登入 URL] 欄位，請輸入「`https://login.microsoftonline.com`」。
 - g. 若為 [租用戶識別碼] 欄位，請輸入 Azure Active Directory 的租用戶識別碼，例如：`microsoft.com` 或 `common`。

這將會是與可驗證之使用者相關聯的租用戶。若要允許任何人透過 Bot 驗證身分，請使用 `common` 租用戶。
 - h. 針對 [資源 URL]，輸入 `https://graph.microsoft.com/`。
 - i. 請將 [範圍] 欄位保留空白。
5. 按一下 [檔案]。

NOTE

這些值可讓應用程式透過 Microsoft Graph API 存取 Office 365 資料。

您現可在 Bot 程式碼中使用此連線名稱，以擷取使用者權杖。

註冊 Azure AD v2 應用程式

1. 在 [Azure 入口網站](#) 上瀏覽至 Bot 的 [Bot 通道註冊] 頁面。
2. 按一下 [設定]。
3. 在靠近頁面底部的 [OAuth 連線設定] 下方，按一下 [新增設定]。
4. 填寫表單，如下所示：
 - a. 若為 [名稱] 欄位，請輸入連線的名稱。Bot 程式碼中會用到此名稱。
 - b. 若為 [服務提供者]，請選取 [Azure Active Directory v2]。選取此項目後，Azure AD 專用的欄位隨即顯示。
 - c. 若為 [用戶端識別碼] 欄位，請輸入應用程式註冊時的 Azure AD v2 應用程式識別碼。
 - d. 若為 [用戶端密碼] 欄位，請輸入應用程式註冊時的 Azure AD v2 應用程式密碼。
 - e. 若為 [租用戶識別碼] 欄位，請輸入 Azure Active Directory 的租用戶識別碼，例如：`microsoft.com` 或 `common`。

這將會是與可驗證之使用者相關聯的租用戶。若要允許任何人透過 Bot 驗證身分，請使用 `common` 租用戶。
 - f. 若為 [範圍] 欄位，請輸入應用程式註冊時您選擇之權限的名稱：

```
Mail.Read Mail.Send openid profile User.Read User.ReadBasic.All。
```

NOTE

若為 Azure AD v2, [範圍] 欄位接受區分大小寫並以空格區隔值的清單。

- 按一下 [檔案]。

NOTE

這些值可讓應用程式透過 Microsoft Graph API 存取 Office 365 資料。

您現可在 Bot 程式碼中使用此連線名稱，以擷取使用者權杖。

測試連線

- 開啟您剛建立的連線。
- 按一下 [服務提供者連線設定] 窗格頂端的 [測試連線]。
- 第一次將開啟新的瀏覽器索引標籤，其中列出應用程式要求的權限，並提示您接受要求。
- 按一下 [接受]。
- 此動作會將您重新導向 [測試對 `` 的連線已成功] 頁面。

準備 Bot 範例程式碼

- 複製 <https://github.com/Microsoft/BotBuilder> 的 github 存放庫。
- 開啟並建置解決方案: `BotBuilder\CSharp\Microsoft.Bot.Builder.sln`。
- 關閉該解決方案，然後開啟: `BotBuilder\CSharp\Samples\Microsoft.Bot.Builder.Samples.sln`。
- 設定啟動專案。
 - 若為採用 v1 Azure AD 應用程式的 Bot，請使用 `Microsoft.Bot.Sample.AadV1Bot` 專案。
 - 若為採用 v2 Azure AD 應用程式的 Bot，請使用 `Microsoft.Bot.Sample.AadV2Bot` 專案。
- 開啟 `Web.config` 檔案，並按照以下指示修改應用程式設定：
 - 將 `ConnectionName` 設為您在配置 Bot 的 OAuth 2.0 連線設定時所用的值。
 - 將 `MicrosoftAppId` 值設為 Bot 的應用程式識別碼。
 - 將 `MicrosoftAppPassword` 值設為 Bot 的密碼。

IMPORTANT

視密碼中的字元而定，您可能需要讓 XML 逸出該密碼。例如，任何 & 符號都必須編碼為 &；。

```
<appSettings>
  <add key="ConnectionName" value="<your-AAD-connection-name>"/>
  <add key="MicrosoftAppId" value="<your-bot-appId>" />
  <add key="MicrosoftAppPassword" value="<your-bot-password>" />
</appSettings>
```

如果您不知道如何取得 Microsoft 應用程式識別碼和 Microsoft 應用程式密碼值，您可以建立如下所述的新密碼：[bot-channels-registration-password](#) 或擷取 Microsoft 應用程式識別碼和 Microsoft 應用程式密碼，從如下所述的部署使用 Bot 通道註冊佈建：[find-your-azure-bots-appid-and-appsecret](#)

NOTE

您現在可將此 Bot 程式碼發佈至 Azure 訂用帳戶 (以滑鼠右鍵按一下專案，然後選擇 [發佈])，但此動作在本教學課程的範例中並非必要。在 Azure 入口網站中配置 Bot 時，您必須進行發佈設定，其應使用您所用的應用程式和主控方案。

使用模擬器測試您的 Bot

您必須安裝 [Bot 模擬器](#)以便在本機測試 Bot。您可使用 v3 或 v4 模擬器。

- 啟動 Bot (無論是否啟用偵錯)。
- 請記下該頁面的 localhost 連接埠號碼。稍後與 Bot 互動時，您會需要使用此資訊。
- 啟動模擬器。
- 連線至 Bot。

如果您尚未設定連線，請提供位址及 Bot 的 Microsoft 應用程式識別碼和密碼。將 `/api/messages` 新增至 Bot 的 URL。URL 外觀應該會類似於 `http://localhost:portNumber/api/messages`。

- 輸入「`help`」以檢視適用於 Bot 的可用命令清單，以及測試驗證功能。
- 登入後一直到登出前，您都不需要再次提供認證。
- 若要登出並取消驗證，請輸入「`signout`」。

NOTE

Bot 驗證需要使用 Bot 連接器服務。該服務將針對您的 Bot 存取 Bot 通道註冊資訊，因此您必須在入口網站上設定 Bot 的傳訊端點。此外，驗證也會需要使用 HTTPS，因此您必須建立 HTTPS 轉送位址，以供 Bot 在本機上執行。

權杖擷取流程注意事項

使用者要求 Bot 執行需要使用者登入的工作時，Bot 可使用 `Microsoft.Bot.Builder.Dialogs.GetTokenDialog` 起始擷取特定連線的權杖。接下來幾個程式碼片段是取自 `GetTokenDialog` 類別。

檢查已快取的權杖

在此程式碼中，首先 Bot 會快速檢查以判斷 Azure Bot 服務是否已有該使用者的權杖 (藉由目前的活動傳送者識別) 和指定的 ConnectionName (即設定中使用的連線名稱)。Azure Bot 服務可能會快取權杖，也可能不會。針對 `GetUserTokenAsync` 的呼叫會執行此「快速檢查」。如果 Azure Bot 服務具有權杖並將其傳回，則該權杖可立即使用。如果 Azure Bot 服務沒有權杖，此方法將傳回 Null。在此情況下，Bot 可傳送自訂的 OAuthCard 供使用者登入。

```
// First ask Bot Service if it already has a token for this user
var token = await context.GetUserTokenAsync(ConnectionName).ConfigureAwait(false);
if (token != null)
{
    // use the token to do exciting things!
}
else
{
    // If Bot Service does not have a token, send an OAuth card to sign in
    await SendOAuthCardAsync(context, (Activity)context.Activity);
}
```

傳送 OAuthCard 給使用者

您可將 OAuthCard 自訂為任何您想用的文字和按鈕文字。需要注意的重點如下：

- 將 `ContentType` 設為 `OAuthCard.ContentType`。
- 將 `ConnectionName` 屬性設為您要使用之連線的名稱。
- 包含一個具有 `Type ActionTypes.Signin` 之 `CardAction` 的按鈕；請注意，您不需要為登入連結指定任何值。

在此呼叫結束時，Bot 必須「等候權杖」回傳。由於可能有許多使用者需要登入，因此等候過程會在主要的活動流中發生。

```
private async Task SendOAuthCardAsync(IDialogContext context, Activity activity)
{
    await context.PostAsync($"To do this, you'll first need to sign in.");

    var reply = await context.Activity.CreateOAuthReplyAsync(_connectionName, _signInMessage,
    _buttonLabel).ConfigureAwait(false);
    await context.PostAsync(reply);

    context.Wait(WaitForToken);
}
```

等候 TokenResponseEvent

在此程式碼中，Bot 的對話方塊類別正在等候 `TokenResponseEvent`（下文將詳述其如何路由至對話方塊堆疊）。

`WaitForToken` 方法會先判斷此事件是否已傳送。如果其已傳送，則可供 Bot 使用。如果未傳送，則 `WaitForToken` 方法將擷取任何傳送給 Bot 的文字，再將其傳遞至 `GetUserTokenAsync`。此作法的原因是部分用戶端（例如：WebChat）不需要進行神奇代碼驗證碼，而可直接在 `TokenResponseEvent` 中傳送權杖。其他用戶端（例如：Facebook 或 Slack）仍須使用神奇代碼。Azure Bot 服務會向用戶端顯示一組六位數的神奇代碼，並要求使用者在聊天視窗中輸入該代碼。此方式並非最理想的「回復」行為，因此，假如 `WaitForToken` 收到代碼，則 Bot 可將代碼傳送至 Azure Bot Service 並獲取權杖。若此呼叫也失敗，則您可決定是否要回報錯誤或採取其他行動。不過大部分情況下，Bot 現在都會有使用者權杖。

若您檢視 `MessageController.cs` 檔案，即可發現此類型的 `Event` 活動也會路由至對話方塊堆疊。

```
private async Task WaitForToken(IDialogContext context, IAwaitable<object> result)
{
    var activity = await result as Activity;

    var tokenResponse = activity.ReadTokenResponseContent();
    if (tokenResponse != null)
    {
        // Use the token to do exciting things!
    }
    else
    {
        if (!string.IsNullOrEmpty(activity.Text))
        {
            tokenResponse = await context.GetUserTokenAsync(ConnectionString,
                activity.Text);

            if (tokenResponse != null)
            {
                // Use the token to do exciting things!
                return;
            }
        }
    }
    await context.PostAsync($"Hmm. Something went wrong. Let's try again.");
    await SendOAuthCardAsync(context, activity);
}
```

在對 Bot 的後續呼叫中，請注意此範例 Bot 永遠不會快取權杖。這是因為 Bot 可一律向 Azure Bot 服務要求權杖。當 Azure Bot 服務為您執行一切工作時，如此可避免 Bot 必須管理權杖生命週期、更新權杖等繁瑣作業。

```
else if(message.Type == ActivityTypes.Event)
{
    if(message.IsTokenResponseEvent())
    {
        await Conversation.SendAsync(message, () => new Dialogs.RootDialog());
    }
}
```

其他資源

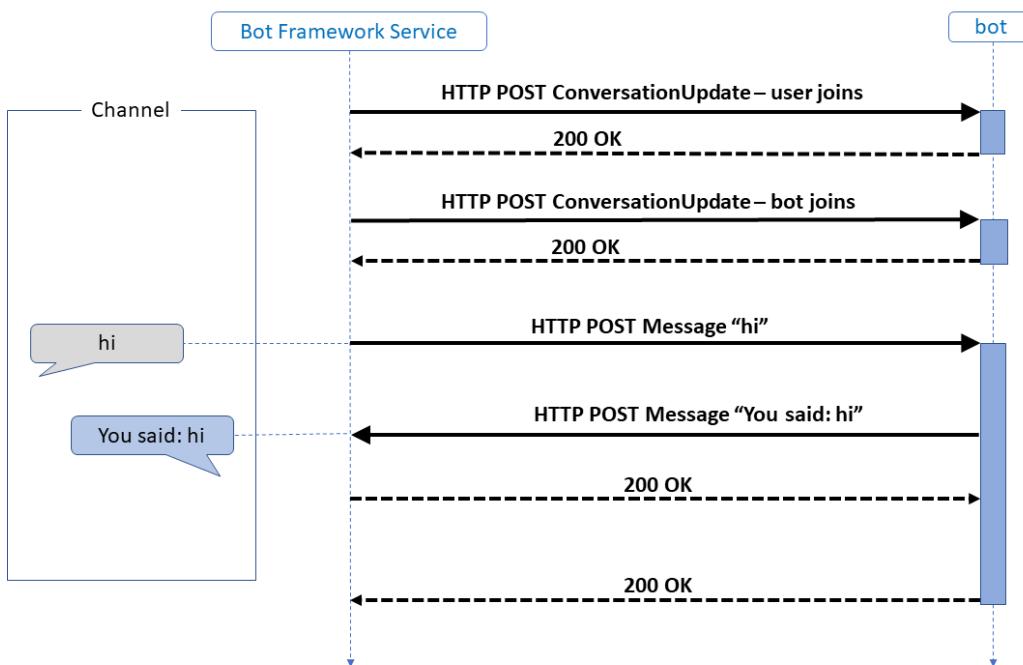
[Bot Framework SDK](#)

Bot 的運作方式

2019/5/14 • [Edit Online](#)

適用於:  SDK v4  SDK v3

Bot 是使用者可運用文字、圖形 (例如卡片或影像) 或語音等對話方式進行互動的應用程式。使用者與 Bot 之間的每次互動都會產生「活動」。Bot Framework Service 是 Azure Bot Service 的元件，會在使用者的 Bot 連線應用程式 (例如 Facebook、Skype、Slack 等，我們稱之為「通道」) 與 Bot 之間傳送資訊。每個通道都可以在其傳送的活動中包含其他資訊。建立 Bot 之前，請務必了解 Bot 如何使用活動物件來與其使用者通訊。讓我們先看看當我們執行簡單回應 Bot 時所交換的活動。



這裡說明的兩種活動類型為：「對話更新」和「訊息」。

有對象加入對話時，Bot Framework Service 可以傳送對話更新。例如，使用 Bot Framework 模擬器開始對話時，您會看到兩個對話更新活動（一個用於使用者加入對話，另一個用於 Bot 加入對話）。若要區分這些對話更新活動，請檢查「已新增成員」屬性是否包含 Bot 以外的成員。

訊息活動會隨附合作對象之間的對話資訊。在回應 Bot 範例中，訊息活動會隨附簡單的文字，而通道會呈現這段文字。或者，訊息活動可能隨附要說出的文字、建議的動作或要顯示的卡片。

在此範例中，Bot 建立並傳送了訊息活動，以回應其已接收的輸入訊息活動。不過，Bot 可以用其他方式來回應所收到的訊息活動；Bot 通常不會在訊息活動中傳送一些歡迎文字來回應對話更新活動。在[歡迎使用者](#)中可找到詳細資訊。

HTTP 詳細資料

活動會透過 HTTP POST 要求從 Bot Framework Service 送達 Bot。Bot 會以 200 HTTP 狀態碼回應傳入 POST 要求。從 Bot 傳送至通道的活動，會以個別的 HTTP POST 傳送至 Bot Framework Service。接著會以 200 HTTP 狀態碼確認。

通訊協定不會指定進行這些 POST 要求及其確認的順序。不過，為了配合常見的 HTTP 服務架構，這些要求通常為巢狀，表示輸出 HTTP 要求是從輸入 HTTP 要求範圍內的 Bot 提出。上面的圖表可說明此模式。由於有兩個不同的背對背 HTTP 連線，所以必須針對兩者提供安全性模型。

定義一個回合

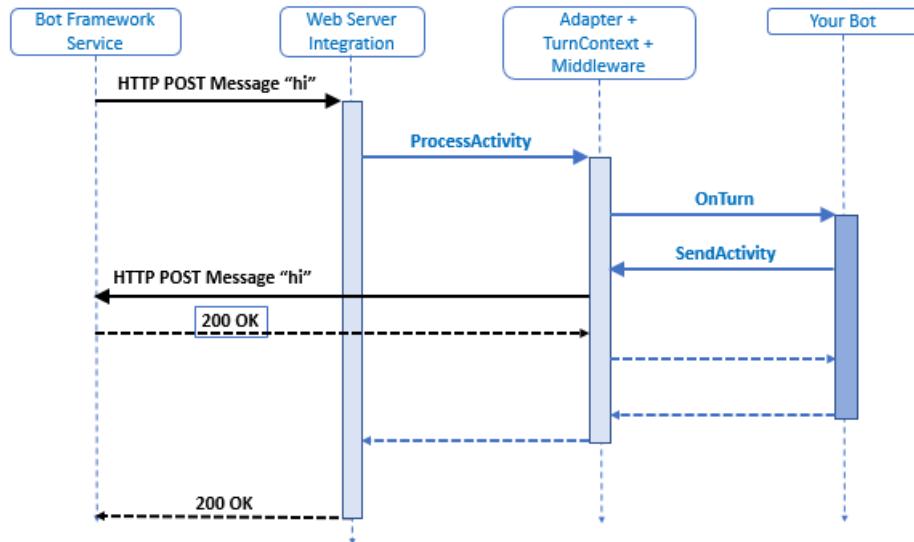
在對話中，人們通常一次一個人發言，輪流發言。Bot 通常會回應使用者輸入。在 Bot Framework SDK 內，一個「回合」是由使用者對 Bot 的傳入活動，以及 Bot 傳回給使用者作為立即回應的任何活動所組成。您可以將回合視為與特定活動送達相關聯的處理。

「回合內容」物件會提供活動相關資訊，例如傳送者和接收者、通道、以及處理活動所需的其他資料。也允許在遍及 Bot 各種階層的回合期間新增資訊。

回合內容是 SDK 中最重要的抽象概念之一。不只隨附所有中介軟體元件的輸入活動和應用程式邏輯，但也提供中介軟體元件和應用程式邏輯可藉以傳送輸出活動的機制。

活動處理堆疊

讓我們深入探索先前的圖表，著重於訊息活動的送達。



在上述範例中，Bot 使用了包含相同文字訊息的另一個訊息活動回覆訊息活動。處理會從送達網路伺服器的 HTTP POST 要求 (以 JSON 承載形式隨附活動資訊) 著手。在 C# 中，這通常會是 ASP.NET 專案，在 JavaScript Node.js 專案中，這可能是其中一種熱門架構，例如 Express 或 Restify。

「配接器」(SDK 的整合式元件) 是 SDK 執行階段的核心。在 HTTP POST 主體中會以 JSON 形式隨附活動。此 JSON 會還原序列化以建立活動物件，而該物件會透過呼叫「處理活動」方法送交配接器。接收活動時，配接器會建立「回合內容」並呼叫中介軟體。

如上所述，回合內容會提供 Bot 用以傳送輸出活動的機制，其最常用於回應入活動。為了達成此目的，回合內容會提供_傳送、更新和刪除活動_回應方法。每個回應方法都會以非同步程序執行。

IMPORTANT

處理主要 Bot 回合的執行緒可在其完成後處置內容物件。請務必 `await` 任何活動呼叫，這樣主要執行緒才能先等候已產生的活動，再結束其正在處理的工作並處置回合內容。否則，如果回應 (包含其處理常式) 佔用大量時間，並嘗試在內容物件上動作，則可能會取得「內容已處置」錯誤。

活動處理常式

當 Bot 收到活動時，其會將活動傳遞至其活動處理常式。實際上，有一個叫做回合處理常式的基底處理常式。所有活動都會透過該處路由傳送。然後，該回合處理常式會針對其所收到的活動類型，呼叫個別的活動處理常式。

- [C#](#)
- [JavaScript](#)

比方說，如果 Bot 收到訊息活動，則回合處理常式會看見該傳入活動並將它傳送給 `OnMessageActivityAsync` 活動

處理常式。

建置您的 Bot 時，用於處理和回應訊息的 Bot 邏輯會進入此 `OnMessageActivityAsync` 處理常式。同樣地，您用於處理加入交談成員的邏輯會進入 `OnMembersAddedAsync` 處理常式，每當成員加入交談時就會呼叫該處理常式。

若要針對這些處理常式實作您的邏輯，您將會覆寫 Bot 中的這些方法，如下面 [Bot 邏輯](#)一節所示。上述每個處理常式都沒有基底實作，所以只要新增您想要的覆寫邏輯即可。

在某些情況中，您會想要覆寫基底回合處理常式，例如在回合結束時 [儲存狀態](#)。若要這麼做，請務必先呼叫 `await base.OnTurnAsync(turnContext, cancellationToken);`，確保在其他程式碼之前執行 `OnTurnAsync` 的基底實作。此外，該基底實作會負責呼叫其餘的活動處理常式，例如 `OnMessageActivityAsync`。

中介軟體

中介軟體就像任何其他傳訊中介軟體，包含一組依序執行的線性元件，可讓每個元件都有機會在活動上運作。中介軟體管線的最後一個階段是在應用程式已向配接器的程序活動方法註冊的 Bot 類別上，回呼回合處理常式。回合處理常式通常是採用 C# 的 `OnTurnAsync` 和採用 JavaScript 的 `onTurn`。

回合處理常式會以回合內容作為其引數，在回合處理常式函式內執行的應用程式邏輯，通常會處理輸入活動的內容，並且在回應時產生一或多個活動，然後使用回合內容的「傳送活動」函式將這些活動送出。對回合內容呼叫「傳送活動」會導致在輸出活動上叫用中介軟體元件。中介軟體元件會在 Bot 的回合處理常式函式前後執行。此執行原本就是巢狀，因此有時候會比喻為俄羅斯娃娃。如需有關中介軟體的深入詳細資訊，請參閱 [中介軟體主題](#)。

Bot 結構

在下列各節中，我們會檢查 EchoBot 的 _主要片段_，而您可使用針對 [CSharp](#) 或 [JavaScript](#) 提供的範本輕鬆地建立。

Bot 是一種 Web 應用程式，而我們會針對每種語言提供範本。

- [C#](#)
- [JavaScript](#)

VSIX 範本可產生 [ASP.NET MVC Core](#) Web 應用程式。如果您查看 [ASP.NET](#) 基礎，您會在 `Program.cs` 和 `Startup.cs` 等檔案中看到類似的程式碼。所有 Web 應用程式都需要這些檔案，並不是 Bot 特有的。

appsettings.json 檔案

`appsettings.json` 檔案指定 Bot 的組態資訊，例如應用程式識別碼和密碼，以及其他項目。如果在生產環境中使用特定技術或使用此 Bot，您必須將您的特定金鑰或 URL 新增至此組態。不過，針對此 Echo Bot，您不需要立即在此執行任何動作；應用程式識別碼和密碼此時可保持未定義。

Bot 邏輯

Bot 邏輯會處理來自從一或多個通道的傳入活動，並產生傳出活動來回應。

- [C#](#)
- [JavaScript](#)

主要 Bot 邏輯會定義於 Bot 程式碼，在此名為 `Bots/EchoBot.cs`。`EchoBot` 衍生自 `ActivityHandler`，而後者衍生自 `IBot` 介面。`ActivityHandler` 會針對不同類型的活動定義各種處理常式，例如以下定義的兩項：`OnMessageActivityAsync` 和 `OnMembersAddedAsync`。這些方法會受到保護，但可加以覆寫，因為是衍生自 `ActivityHandler`。

`ActivityHandler` 中定義的處理常式如下：

EVENT	處理常式	說明
收到的任何活動類型	<code>OnTurnAsync</code>	根據所收到的活動型別，呼叫其中一個其他處理常式。
收到的訊息活動	<code>OnMessageActivityAsync</code>	覆寫此項目以處理 <code>Message</code> 活動。
收到的交談更新活動	<code>OnConversationUpdateActivityAsync</code>	在 <code>ConversationUpdate</code> 活動上，如有 Bot 以外的成員加入或離開交談，則會呼叫處理常式。
已加入交談的非 Bot 成員	<code>OnMembersAddedAsync</code>	覆寫此項目以處理要加入交談的成員。
已離開交談的非 Bot 成員	<code>OnMembersRemovedAsync</code>	覆寫此項目以處理要離開交談的成員。
收到的事件活動	<code>OnEventActivityAsync</code>	在 <code>Event</code> 活動上，呼叫事件類型特有的處理常式。
收到的權杖回應事件活動	<code>OnTokenResponseEventAsync</code>	覆寫此項目以處理權杖回應事件。
收到的非權杖回應事件活動	<code>OnEventAsync</code>	覆寫此項目以處理其他類型的事件。
收到的其他活動類型	<code>OnUnrecognizedActivityTypeAsync</code>	覆寫此項目以處理任何未處理的活動類型。

這些不同的處理常式都有 `turnContext`，其可提供連入活動（對應至輸入 HTTP 要求）的相關資訊。活動可以是各種類型，所以每個處理常式都會在其回合內容參數中提供強型別活動；在大部分情況下，一律處理通常最常見的 `OnMessageActivityAsync`。

如同舊版 4.x 的這個架構，也有實作公用方法 `OnTurnAsync` 的選項。目前，這個方法的基底實作會處理錯誤檢查，然後根據連入活動的類型，呼叫每個特定處理常式（如我們在此範例中定義的兩個處理常式）。在大部分的情況下，您可不理會該方法並使用個別的處理常式，但如果您的情況需要 `OnTurnAsync` 的自訂實作，這仍是一個選項。

IMPORTANT

如果您覆寫 `OnTurnAsync` 方法，您必須呼叫 `base.OnTurnAsync` 來取得基底實作，以呼叫所有其他 `On<activity>Async` 處理常式或自行呼叫這些處理常式。否則不會呼叫這些處理常式，且不會執行該程式碼。

在此範例中，我們歡迎新使用者使用 `SendActivityAsync` 呼叫來回應使用者所傳送的訊息。輸出活動會對應至輸出 HTTP POST 要求。

```

public class MyBot : ActivityHandler
{
    protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
    CancellationToken cancellationToken)
    {
        await turnContext.SendActivityAsync(MessageFactory.Text($"Echo: {turnContext.Activity.Text}"),
    cancellationToken);
    }

    protected override async Task OnMembersAddedAsync(IList<ChannelAccount> membersAdded,
    ITurnContext<IConversationUpdateActivity> turnContext, CancellationToken cancellationToken)
    {
        foreach (var member in membersAdded)
        {
            await turnContext.SendActivityAsync(MessageFactory.Text($"welcome {member.Name}"),
    cancellationToken);
        }
    }
}

```

從您的應用程式存取 Bot

- [C#](#)
- [JavaScript](#)

設定服務

`Startup.cs` 檔案中的 `ConfigureServices` 方法會載入連線服務，以及從 `appsettings.json` 或 Azure Key Vault (如果有的話) 載入其金鑰、連接狀態等等。在此，我們會在服務上新增 MVC 及設定相容性版本，然後設定可透過對 Bot 控制器插入相依性取得的配接器和 Bot。

```

// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    // Create the credential provider to be used with the Bot Framework Adapter.
    services.AddSingleton<ICredentialProvider, ConfigurationCredentialProvider>();

    // Create the Bot Framework Adapter.
    services.AddSingleton<IBotFrameworkHttpAdapter, BotFrameworkHttpAdapter>();

    // Create the bot as a transient. In this case the ASP Controller is expecting an IBot.
    services.AddTransient<IBot, EchoBot>();
}

```

`Configure` 方法會指定應用程式使用 MVC 和一些其他檔案，以完成您應用程式的設定。使用 Bot Framework 的所有 Bot 都需要該組態呼叫，不過，當您建置 Bot 時會已經定義於範例或 VSIX 範本中。`ConfigureServices` 和 `Configure` 會在應用程式啟動由執行階段呼叫。

Bot 控制器

控制站 (遵循標準 MVC 結構) 可讓您決定訊息和 HTTP POST 要求的路由。對於我們的 Bot，我們會將傳入要求傳遞至配接器的程序非同步活動方法，如上面[活動處理堆疊](#)一節所述。在該呼叫中，我們會指定 Bot 和任何其他可能需要的授權資訊。

此控制器會實作 `ControllerBase`、保留我們在 `Startup.cs` 中設定的配接器和 Bot (在此可透過插入相依性取得)，並在收到傳入 HTTP POST 時將所需的資訊傳遞到 Bot。

在此，您會看到由路由和控制器屬性所繼續處理的類別。這些屬性可協助架構適當地路由傳送訊息，並知道要使用哪一個控制器。如果您變更路由屬性的值，則會變更模擬器或其他通道用於存取您 Bot 的端點。

```
// This ASP Controller is created to handle a request. Dependency Injection will provide the Adapter and
IBot
// implementation at runtime. Multiple different IBot implementations running at different endpoints can be
// achieved by specifying a more specific type for the bot constructor argument.
[Route("api/messages")]
[ApiController]
public class BotController : ControllerBase
{
    private readonly IBotFrameworkHttpAdapter Adapter;
    private readonly IBot Bot;

    public BotController(IBotFrameworkHttpAdapter adapter, IBot bot)
    {
        Adapter = adapter;
        Bot = bot;
    }

    [HttpPost]
    public async Task PostAsync()
    {
        // Delegate the processing of the HTTP POST to the adapter.
        // The adapter will invoke the bot.
        await Adapter.ProcessAsync(Request, Response, Bot);
    }
}
```

管理 Bot 資源

Bot 資源 (例如連線服務的應用程式識別碼、密碼、金鑰或祕密) 必須受到適當管理。如需如何執行這項操作的詳細資訊，請參閱[管理 Bot 資源](#)。

其他資源

- 若要了解 bot 中的狀態角色，請參閱[管理狀態](#)。

管理狀態

2019/4/26 • [Edit Online](#)

適用於:  SDK v4  SDK v3

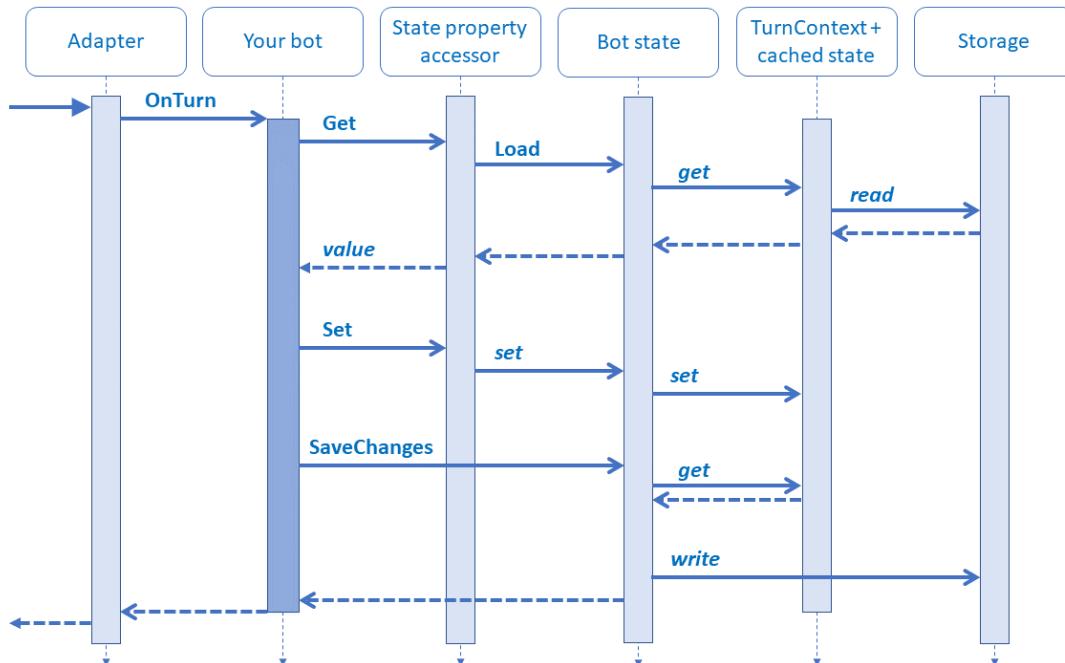
Bot 內的狀態會遵循與新式 Web 應用程式相同的架構，而 Bot Framework SDK 會提供一些抽象概念，讓狀態管理變得更容易。

如同 Web 應用程式，Bot 原本是無狀態的；不同的 Bot 執行個體可處理任何指定的交談回合。有些 Bot 偏好此種隱含性 - Bot 不需要額外資訊即可運作，或所需的資訊保證在內送訊息中。而其他 Bot 則需要狀態（例如，我們在交談中的位置或先前收到的使用者相關資料）才能進行有用的交談。

為什麼需要狀態？

維護狀態可讓 Bot 藉由記住有關使用者或交談的特定事項，以便進行更有意義的交談。比方說，如果您之前已跟使用者談話，即可儲存其先前的相關資訊，您就不需要再次詢問。狀態也可讓資料保存超過目前的回合，以便 Bot 保存多回合交談過程中的資訊。

由於屬於 Bot，所以使用狀態有幾個層次，我們將在此討論：儲存層、狀態管理（包含在下圖的 Bot 狀態中）和狀態屬性存取子。此圖說明這些層次之間的互動順序部分，實心箭頭表示方法呼叫，而虛線箭頭表示回應（不論是否有傳回值）。



下列各節說明此圖的流程，包含各層次的詳細資訊。

儲存層

從後端說起，這是狀態資訊的實際儲存位置，也就是我們的「儲存層」。這可視為實體儲存體，例如記憶體內部、Azure 或第三方伺服器。

Bot Framework SDK 包含儲存層的一些實作：

- **記憶體內部儲存體**可將記憶體內部儲存體運用於測試用途。記憶體內部資料儲存體僅限用於本機測試，因為此儲存體是易變且暫存的。每次 Bot 重新啟動時，就會清除資料。
- **Azure Blob 儲存體**會連線到 Azure Blob 儲存體物件資料庫。

- Azure Cosmos DB 儲存體會連線到 Cosmos DB NoSQL 資料庫。

如需有關如何連線到其他儲存體選項的指示，請參閱 [直接寫入儲存體](#)。

狀態管理

「狀態管理」可自動讀取 Bot 的狀態以及將其寫入至基礎儲存層。狀態會儲存為「狀態屬性」，這是 Bot 無須擔心特定基礎實作，即可透過狀態管理物件讀取和寫入的有效索引鍵 / 值組。這些狀態屬性會定義該資訊的儲存方式。例如，當您擷取定義為特定類別或物件的屬性時，您知道該資料將如何結構化。

這些狀態屬性會歸併為限域「貯體」，而這只是協助組織這些屬性的集合。SDK 包含三個「貯體」：

- 使用者狀態
- 交談狀態
- 私人交談狀態

上述所有貯體都是「Bot 狀態」類別的子類別，可加以衍生來定義具有不同範圍的其他貯體類型。

視貯體而定，這些預先定義的貯體受限於特定可見性：

- 不論交談內容為何，使用者狀態適用於 Bot 在該頻道上與該使用者正在交談的任何回合
- 不論使用者為何（亦即群組交談），交談狀態適用於特定交談中的任何回合
- 私人交談狀態受限於特定交談和該特定使用者

TIP

使用者和交談狀態是由通道界定。使用不同通道來存取 Bot 的相同人員會顯示為不同的使用者，每個通道一個使用者，而各有不同的使用者狀態。

每個預先定義的貯體所用的索引鍵為使用者和交談所特有。設定狀態屬性的值時，系統會使用回合內容內含的資訊在內部為您定義索引鍵，以確保每個使用者或交談取均位於正確的貯體和屬性。具體而言，索引鍵的定義如下所示：

- 使用者狀態會使用「頻道識別碼」和「傳送者識別碼」來建立索引鍵。例如，`{Activity.ChannelId}/users/{Activity.From.Id}#YourPropertyName`
- 交談狀態會使用「頻道識別碼」和「交談識別碼」來建立索引鍵。例如，`{Activity.ChannelId}/conversations/{Activity.Conversation.Id}#YourPropertyName`
- 私人交談狀態會使用「頻道識別碼」、「傳送者識別碼」和「交談識別碼」來建立索引鍵。例如，`{Activity.ChannelId}/conversations/{Activity.Conversation.Id}/users/{Activity.From.Id}#YourPropertyName`

何時使用每種類型的狀態

交談狀態很適合追蹤交談內容，例如：

- Bot 是否詢問使用者問題，且問題為何
- 交談的目前主題為何，或最後一個主題是什麼

使用者狀態適合用來追蹤使用者的相關資訊，例如：

- 非重大使用者資訊，例如名稱和喜好設定、警示設定或警示喜好設定
- 他們與 Bot 最後一次交談的相關資訊
 - 例如，產品支援 Bot 可能會追蹤使用者所詢問的產品。

私人交談狀態適合於支援群組交談的通道，但您想要在哪裡追蹤使用者和交談特定資訊。例如，如果您有教室 Clicker Bot：

- Bot 無法彙總並顯示學生對於指定問題的回應。

- Bot 可以彙總每個學生的表現，並且在課程結束時，將該資訊私下轉達給他們。

如需使用這些預先定義貯體的詳細資訊，請參閱[狀態操作說明文章](#)。

連線到多個資料庫

如果 Bot 需要連線到多個資料庫，請為每個資料庫建立儲存層。針對每個儲存層，建立您需要支援狀態屬性的狀態管理物件。

狀態屬性存取子

「狀態屬性存取子」用來實際讀取或寫入其中一個狀態屬性，並提供 *get*、*set* 和 *delete* 方法，以便存取一個回合內的狀態屬性。若要建立存取子，您必須提供屬性名稱，這通常發生於您初始化 Bot 時。然後，您可以使用該存取子來取得及操作 Bot 狀態的該屬性。

存取子允許 SDK 從基礎儲存體中取得狀態，並且為您更新 Bot 的「狀態快取」。狀態快取是由 Bot 維護的本機快取，可為您儲存狀態物件，並允許不需存取基礎儲存體的讀取和寫入作業。如果狀態尚未位於快取中，則呼叫存取子的 *get* 方法可擷取狀態並其放入快取中。擷取之後，即可如同本機變數一樣操作狀態屬性。

存取子的 *delete* 方法可從快取中移除屬性，也可從基礎儲存體中刪除屬性。

IMPORTANT

第一次呼叫存取子的 *get* 方法時，您必須提供 Factory 方法來建立物件（如果其尚未存在於您的狀態中）。如果未提供任何 Factory 方法，則會發生例外狀況。在[狀態操作說明文章](#)中可找到有關如何使用 Factory 方法的詳細資訊。

對於經由存取子所取得的狀態屬性，若要保存您對其所做的任何變更，就必須更新狀態快取中的屬性。您可以透過呼叫存取子的 *set* 方法，該方法會在快取中設定您的屬性值，而且適用於稍後在該回合中讀取或更新屬性。若要將該資料實際保存到基礎儲存體（因而在目前回合之後提供使用），您必須接著**儲存狀態**。

狀態屬性存取子方法的運作方式

存取子方法是 Bot 與狀態互動的主要方式。每個存取子方法的運作方式，以及基礎層的互動方式，如下所示：

- 存取子的 *get* 方法：
 - 存取子會要求狀態快取中的屬性。
 - 如果屬性位於快取中，請將其傳回。否則，從狀態管理物件中取得。（如果屬性尚未存在於狀態中，請使用存取子的 *get* 呼叫中所提供的 Factory 方法。）
- 存取子的 *set* 方法：
 - 使用新的屬性值更新狀態快取。
- 狀態管理物件的 *save changes* 方法：
 - 檢查狀態快取中的屬性變更。
 - 將該屬性寫入至儲存體。

儲存狀態

當您呼叫存取子的 *set* 方法來記錄已更新的狀態時，該狀態屬性尚未儲存到保存的儲存體，而改為只儲存到 Bot 的狀態快取。若要將狀態快取中的所有變更儲存到保存狀態，您必須呼叫狀態管理物件的「儲存變更」方法，該方法適用於上述 Bot 狀態類別（例如使用者狀態或交談狀態）的實作。

針對狀態管理物件（也就是上述的貯體）呼叫「儲存變更」方法，就會在狀態快取中儲存您為該貯體設定的所有屬性，但不適合存在於 Bot 狀態中的任何其他貯體。

TIP

Bot 狀態會實作「以最後寫入者為準」行為，因此最後寫入的狀態將蓋過先前寫入的狀態。這對許多應用程式來說可能還可以運作，但會有些顧慮，特別是在向外延展的案例中，其中可能正在進行某種程度的並行或延遲作業。

如果您有一些自訂中介軟體，可能在回合處理常式完成後更新狀態，請考慮[在中介軟體中處理狀態](#)。

其他資源

- [對話狀態](#)
- [直接寫入儲存體](#)
- [儲存交談和使用者資料](#)

對話方塊程式庫

2019/5/10 • [Edit Online](#)

適用於:  SDK v4  SDK v3

「對話」是 SDK 的中心概念，可提供實用的方法來管理與使用者的交談。對話是 Bot 中的結構，其作用類似 Bot 程式中的函式；每個對話的設計都是依照特定順序來執行特定工作。您可以指定個別對話的順序來引導交談，並透過不同的方式叫用對話，有時透過給使用者的回應來叫用，有時透過給一些外部刺激的回應來叫用，也從其他對話來叫用。

Dialogs 程式庫會提供一些內建功能（例如「提示」和「瀑布圖對話」），讓 Bot 交談更易於管理。[提示](#)可用來詢問不同類型的資訊，例如文字、數字或日期。[瀑布式對話](#)可將多個步驟結合成一個序列，讓 Bot 能夠輕鬆地遵循該預先定義的序列，並將資訊傳遞至下一個步驟。

對話及其片段

Dialogs 程式庫包含一些額外的片段，可讓對話更有用。除了下面討論的不同[對話類型](#)，程式庫還包含「對話集」、「對話內容」和「對話結果」的概念。

簡單來說，「對話集」是對話的集合。這可以是提示、瀑布式對話或元件對話之類的事物。每一項都是對話的實作，並且每一項都會新增至具有特定字串識別碼的對話集。當 Bot 想要開始對話集內的特定對話或提示時，其會使用該字串識別碼來指定要使用哪個對話。

「對話內容」包含屬於對話的資訊，且用來從 Bot 的回合處理常式內與對話集互動。對話內容包含目前的回合內容、父代對話，以及[對話狀態](#)，可供保留對話中的資訊。對話內容可讓您開始一個具有其字串識別碼的對話，或繼續目前的對話（例如具有多個步驟的瀑布式對話）。

當對話結束時，其可以傳回「對話結果」，以及來自對話的一些結果資訊。傳回此資訊，讓呼叫方法看到對話中發生的狀況，並將資訊儲存至某個永續性位置（如有需要）。

對話方塊狀態

對話是一種實作多回合交談邏輯的方法，因此，這是 SDK 中依賴多個回合中保存狀態的功能範例。對話中若無狀態，Bot 便無法得知其位於對話集中的何處，或其已經蒐集哪些資訊。

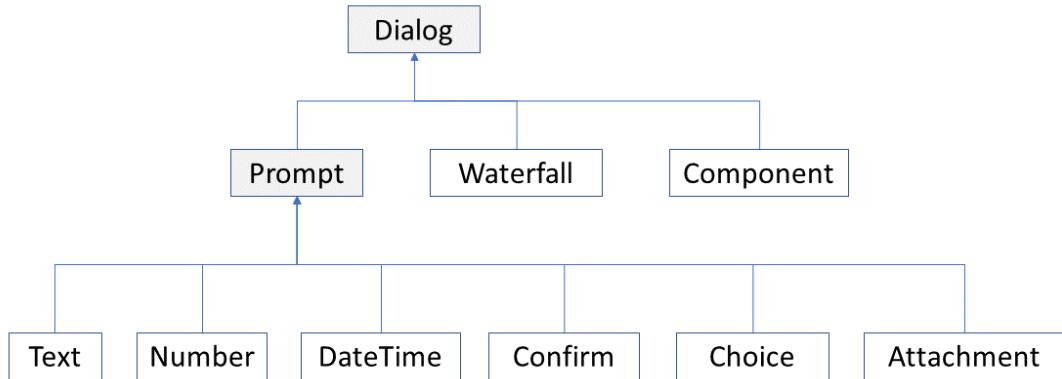
對話式 Bot 通常會保留對話集合做為其 Bot 實作中的成員變數。該對話集會透過存取子物件（可供存取保存狀態）的控點來建立。如需 Bot 內狀態的背景，請參閱[管理狀態](#)。

在 Bot 的開啟回合處理常式內，Bot 會藉由在對話集上呼叫 *create context* 來初始化對話子系統，以傳回「對話內容」。該對話內容包含對話所需的必要資訊。

建立對話內容時需要狀態，透過建立對話集時所提供的存取子即可存取狀態。對話集可以透過存取子，取得適當的對話狀態。在[儲存交談和使用者資料](#)中可找到狀態存取子的詳細資料。

對話類型

對話有幾個不同的類型：提示、瀑布式對話和元件對話，如此類別階層所示。



提示

Dialogs 程式庫中的提示，可讓您輕鬆地向使用者詢問資訊並評估其回應。例如對於「數字提示」，您可指定所詢問的問題或資訊，而提示會自動查看其是否收到有效的數字回應。如果收到，交談即可繼續；如果未收到，則會重新提示使用者輸入有效的答案。

在幕後，提示為兩個步驟的對話方塊。第一步，提示會要求輸入；第二步，其會傳回有效值，或利用重新提示從頂端開始。

呼叫提示時，提示會有「提示選項」，您可以在其中指定要提示的文字、驗證失敗時的重試提示，以及回答提示的選項。

此外，您可以在建立提示時，選擇為您的提示新增一些自訂驗證。例如，假設我們想要使用數字提示來取得派對大小，但該派對大小必須大於 2 且小於 12。提示會先查看其是否收到有效數字，然後執行自訂驗證（若已提供）。如果自訂驗證失敗，則會如上所述重新提示使用者。

提示完成時，其會明確地傳回所要求的結果值。傳回該值後，我們可以確定其已通過內建的提示驗證及任何其他可能已提供的自訂驗證。

如需使用各種提示的範例，請查看如何使用 [Dialogs 程式庫來收集使用者輸入](#)。

提示類型

在幕後，提示為兩個步驟的對話方塊。第一步，提示會要求輸入；第二步，會傳回有效值，或利用重新提示從頂端重新開始。對話方塊 程式庫提供數個基本提示，分別用於收集不同類型的回應。基本提示可解譯自然語言輸入，例如 "ten" 或 "a dozen" 是指數字，或 "tomorrow" 或 "Friday at 10am" 是指日期時間。

PROMPT	說明	傳回
附件提示	要求一或多個附件，例如文件或影像。	「附件」物件的集合。
選擇提示	要求從一組選項中選擇。	「找到的選擇」物件。
確認提示	要求確認。	布林值。
日期時間提示	要求日期時間。	「日期時間解析」物件的集合。
數字提示	要求數字。	數值。

PROMPT	說明	傳回
文字提示	要求一般文字輸入。	字串。

若要提示使用者提供輸入，請使用其中一個內建類別（例如 `_文字提示_`）定義提示，然後將其新增至對話方塊集。提示已修正在對話集內必須是唯一的識別碼。您可以讓每個提示都有自訂驗證程式，而對於某些提示，您可以指定「預設地區設定」。

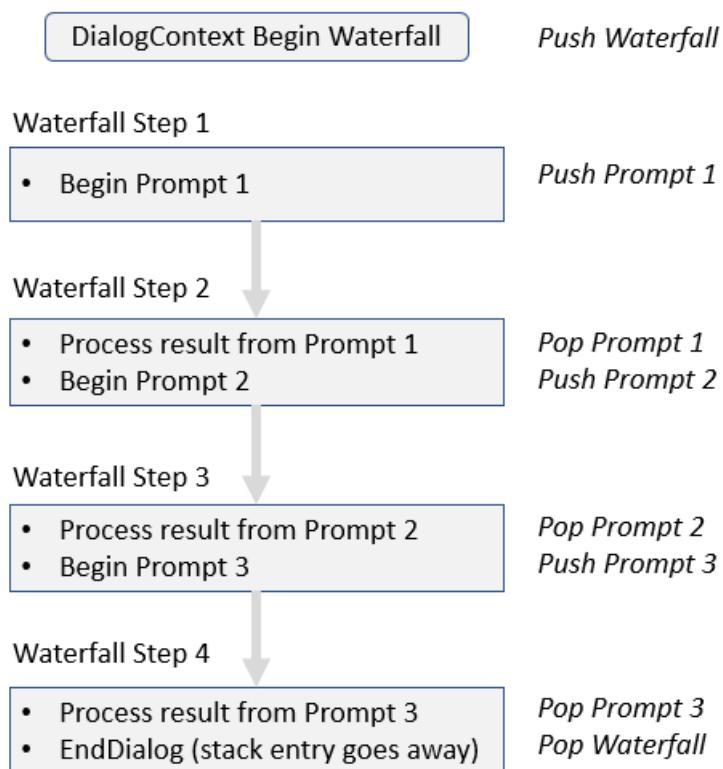
提示地區設定

地區設定用來決定選擇、確認、日期時間和數字提示的特定語言行為。對於來自使用者的任何指定輸入，如果通道在使用者的訊息中提供了 `locale` 屬性，則會使用該屬性。否則，如果在呼叫提示的建構函式時提供提示的「預設地區設定」，或藉由稍後設定，則會使用該預設地區設定。若未提供任何一項，則會以英文（"en-us"）作為地區設定。注意：地區設定是 2、3 或 4 個字元的 ISO 639 代碼，其代表某個語言或語言系列。

瀑布式對話

瀑布式對話是特定的對話實作，常用來向使用者收集資訊或引導使用者完成一系列的工作。每個交談步驟都會以非同步函式的形式實作，而且採用「瀑布式步驟內容」（`step`）參數。在每個步驟中，Bot 會 **提示使用者提供輸入**（或者可以開始子對話，但這通常是提示）、等候回應，然後將結果傳遞給下一個步驟。第一個函式的結果將作為引數傳遞至下一個函式，依此類推。

下圖顯示瀑布步驟序列和所發生的堆疊作業。底下的[使用對話](#)一節會詳細說明對話堆疊的使用方式。



在瀑布式步驟內，瀑布式對話的內容會儲存在其**瀑布式步驟內容**中。這類似於對話內容，因為其可供存取目前的回合內容和狀態。請使用**瀑布步驟內容**物件，從瀑布步驟內與對話方塊集合互動。

您所能處理的傳回值可來自對話中瀑布式步驟內的對話，或來自 Bot 的開啟回合處理常式，儘管您通常只需要檢查 Bot 回合邏輯的對話回合結果狀態。在瀑布步驟內，對話方塊會在瀑布步驟內容的「結果」屬性中提供傳回值。

瀑布式步驟內容屬性

瀑布式步驟內容包含下列各項：

- 選項：包含對話的輸入資訊。
- 值：包含您可以新增至內容，而後轉入後續步驟中的資訊。
- 結果：包含上一個步驟的結果。

此外, `next` 方法會繼續進行相同回合中瀑布式對話的下一個步驟, 並且讓 Bot 視需要略過特定步驟。

元件對話方塊

您有時想要撰寫可重複使用的對話, 以便使用於不同的案例中, 例如要求使用者提供街道、城市和郵遞區號值的地址對話。

「元件對話」提供以下策略: 建立獨立的對話來處理特定案例, 並將大型對話集分割成更多可管理的片段。這些片段都有自己的對話集, 而且會避免與所屬對話集發生任何名稱衝突。請參閱[元件對話操作說明](#), 以獲得詳細資訊。

使用對話

您可以使用對話內容來開始、繼續、取代或結束對話。您也可以取消對話堆疊上的所有對話。

對話可被視為程式設計堆疊, 我們稱之為「對話堆疊」, 其由回合處理常式引導, 而如果堆疊是空的, 則會作為後援。該堆疊最上方的項目會被視為「作用中對話」, 而對話內容會將所有輸入導向作用中對話。

當對話開始時, 其會推送至堆疊, 而成為作用中對話。其會在結束前保持為作用中對話, 而 `replace dialog` 方法會將它移除, 或另一個對話會推送至堆疊(經由回合處理常式或作用中對話本身)並成為作用中對話。當新的對話結束時, 其就會從堆疊中彈出, 而下一個對話會再度成為作用中對話。這可允許[重複對話方塊](#)或[建立對話分支](#), 如下所述。

建立對話內容

若要建立對話內容, 請呼叫對話集的 `create context` 方法。「建立內容」可取得對話集的「對話狀態」屬性, 並使用其來建立對話內容。然後對話內容用來開始、繼續或者控制集合中的對話。

對話集需要使用「狀態屬性存取子」來存取對話狀態。存取子的建立和使用方式與其他狀態存取子相同, 但是會根據交談狀態建立為其自己的專屬屬性。在[管理狀態主題](#)中可找到管理狀態的詳細資訊, 而對話狀態的使用方式則會顯示在[循序交談流程](#)操作說明中。

開始對話方塊

若要開始對話, 請將您要開始的「對話識別碼」傳遞到對話內容的 `begin dialog`、`prompt` 或 `replace dialog` 方法。

- `begin dialog` 方法會將對話推送至堆疊的頂端。
- `replace dialog` 方法會將目前的對話從堆疊取出, 並將取代對話推送至堆疊。被取代的對話已被取消, 而該執行個體包含的資訊已處置。

使用 `options` 參數, 將資訊傳遞至對話的新執行個體。傳遞至新對話方塊的選項, 可透過步驟內容在任何對話方塊步驟中的「選項」屬性來存取。如需範例程式碼, 請參閱[使用分支和迴圈建立進階交談流程](#)操作說明。

繼續對話

若要繼續對話, 請呼叫 `continue dialog` 方法。如果有作用中對話, `continue` 方法一律會繼續進行堆疊最上層的對話(作用中對話)。如果持續的對話結束, 則控制權會傳給在相同回合中繼續的父代內容。

使用步驟內容的 `values` 屬性, 以保存回合之間的狀態。在後續回合中可使用在先前回合中新增至這個集合的值。如需範例程式碼, 請參閱[使用分支和迴圈建立進階交談流程](#)操作說明。

結束對話方塊

`end dialog` 方法可將對話從堆疊中取出, 並將選擇性結果傳回給父代內容(例如呼叫它的對話, 或 Bot 的回合處理常式), 以結束對話。此方法最常從對話中進行呼叫, 以結束本身目前的執行個體。

您可以從具有對話內容的任何地方呼叫 `end dialog` 方法, 但是會對 Bot 顯示為從目前作用中的對話進行呼叫。

TIP

最好是在對話結束時，明確地呼叫 end dialog 方法。

清除所有對話

如果您想要將所有對話方塊從堆疊中移除，您可以藉由呼叫對話方塊內容的 cancel all dialogs 方法來清除對話方塊堆疊。

重複對話方塊

您可使用對話來取代對話本身，並建立迴圈。這很適合用來處理[複雜的反覆運作](#)，也是管理功能表的好方法。

NOTE

如果您需要保存目前對話的內部狀態，則必須將資訊傳遞給 replace dialog 方法呼叫中的新對話執行個體，然後適當地將對話初始化。

產生對話分支

對話內容會維護對話堆疊，並針對堆疊上的每個對話，追蹤接下來是哪個步驟。begin dialog 方法會建立一個子項目並將對話推送至堆疊頂端，而其 end dialog 方法則會將頂端對話從堆疊中取出。End dialog 通常會從即將結束的對話中進行呼叫。

對話方塊可以藉由呼叫對話方塊內容的「開始對話方塊」方法，並提供新對話方塊的識別碼，在同一對話集合內開始新的對話方塊，然後讓新的對話成為目前使用中的對話方塊。原始對話方塊仍在堆疊中，但呼叫對話方塊內容的「繼續 (continue) 對話方塊」方法只會傳送到堆疊頂端的對話方塊，也就是「使用中的對話方塊」。當對話方塊從堆疊中移除時，對話方塊內容會從原始對話方塊停止的地方，繼續 (resume) 使用堆疊中瀑布的下一個步驟。

因此，藉由包含在對話方塊中，可以有條件地從一組可用的對話方塊中選擇對話方塊來開始的步驟，您即可在對話流程內建立分支。

後續步驟

[使用對話方塊程式庫來收集使用者輸入](#)

中介軟體

2019/5/10 • [Edit Online](#)

適用於：  SDK v4  SDK v3

中介軟體只是一個介於介面卡和 Bot 邏輯之間的類別，會在初始化期間新增至您介面卡的中介軟體集合中。SDK 可讓您撰寫自己的中介軟體，或新增他人所建立的中介軟體。每個透過中介軟體進出 Bot 流程的活動。

介面器透過 Bot 中介軟體管道處理內送活動，並導向至 Bot 邏輯再送出。如同每個進出 Bot 的活動流程，每個中介軟體都可檢視活動，或在 Bot 邏輯執行之前或之後採取行動。

在說明中介軟體之前，請務必先了解 [Bot 基本資訊](#)，以及其[如何處理活動](#)。

使用中介軟體

我們常會遇到這個問題：「何時應該將動作實作為中介軟體，何時又該使用一般的 Bot 邏輯？」中介軟體會讓您有額外的機會，可在對話的每個「回合」進行處理之前和之後，與使用者的對話流程互動。中介軟體也可讓您儲存和擷取關於對話的資訊，並於需要時呼叫其他處理邏輯。以下是一些常見的案例，可說明中介軟體實用之處。

查詢每個活動或據以採取行動

在很多情況下，我們會要求 Bot 對每個活動或特定類型的每個活動採取行動。例如：您可能想記錄 Bot 接收到的每個訊息活動，或如果 Bot 未在該回合產生回應則提供後援回應。中介軟體是很好的著手點，其可在其餘 Bot 邏輯執行之前或之後執行。

修改或增強回合內容

如果 Bot 具有的資訊多過於活動中提供的資訊，則特定交談的成果內容可能更豐富。在此情況下，中介軟體可查看其至目前為止的對話狀態、[查詢外部資料來源](#)，並將資料附加至[回合內容](#)物件，然後再將執行作業傳遞至該 Bot 邏輯。

SDK 會定義可記錄傳入和傳出活動的記錄中介軟體，但您也可以定義自己的中介軟體。

Bot 中介軟體管道

針對每個活動，介面卡可依您新增中介軟體的順序進行呼叫。針對該回合和 *next* 委派，介面卡會在內容物件中傳遞，然後中介軟體會呼叫委派，並將控制項傳遞至管道中的下個中介軟體。中介軟體也有機會在 *next* 委派傳回之後，先執行其他工作，再完成方法。您可以想成每個中介軟體物件都有第一次和最後一次的機會，能在管道中和接續於中介軟體物件之後的項目進行互動。

例如：

- 第 1 個中介軟體物件的回合處理常式在呼叫 *next* 之前先執行程式碼。
 - 第 2 個中介軟體物件的回合處理常式在呼叫 *next* 之前先執行程式碼。
 - Bot 的回合處理常式執行並傳回。
 - 第 2 個中介軟體物件的回合處理常式在傳回之前，執行任何其餘程式碼。
- 第 1 個中介軟體物件的回合處理常式在傳回之前，執行任何其餘程式碼。

如果中介軟體未呼叫下個委派，則介面卡不會呼叫任何後續中介軟體或 Bot 回合處理常式，以及管道最少運算路由。

Bot 中介軟體管道完成後，該回合即結束，且回合內容將超出範圍。

中介軟體或 Bot 可產生回應和註冊回應事件處理常式，不過請注意，這些回應會在個別程式中處理。

中介軟體的順序

由於中介軟體新增的順序會影響其處理活動的順序，因此請務必先決定好新增順序。

NOTE

這表示您會獲得適用於大部分的 Bot 的模式；但請務必考量在您的情境中，中介軟體和使用情境下與其他使用者的互動方式。

中介軟體管道中的第一個項目可能是每次都會執行的最低層級工作。例如：記錄、例外狀況處理和翻譯。排列順序取決於您的需求，例如，您想要先翻譯內送訊息，再儲存訊息；還是應該先儲存訊息，這表示不會翻譯所儲存的訊息。

中介軟體管道的最後一個項目是 Bot 特定中介軟體，即您實作的中介軟體，用來處理每次傳送給 Bot 的訊息。如果您的中介軟體使用狀態資訊或其他 Bot 內容中設定的資訊，請將其新增至中介軟體管道中，用於修改狀態或內容的中介軟體之後。

最少運算路由

另一個有關中介軟體和回應處理常式的重要概念是「最少運算路由」。如果系統繼續透過後續的執行層處理該執行作業，則中介軟體（或回應處理常式）必須呼叫其 *next* 委派傳遞執行。如果未在該中介軟體（或回應處理常式）中呼叫下個委派，則相關聯的管線會產生最少運算路由，而後續執行層也不會執行。這表示所有 Bot 邏輯，以及關線中後續的任何中介軟體都會略過。中介軟體與對一回合，產生最少運算路由的回應處理常式之間有細微差異。

中介軟體會對一回合，產生最少運算路由，不會呼叫 Bot 回合處理常式，但在管線中此時點之前執行的所有中介軟體程式碼仍會執行到完成為止。

針對事件處理常式，不呼叫 *next* 代表事件已取消，這與中介軟體略過邏輯大為不同。藉由不處理其餘事件，介面卡就永遠無法傳送。

TIP

如果您執行最少運算路由回應事件（例如 `SendActivities`），請確保其為您預期的行為。否則可能會導致您難以修正錯誤。

回應事件處理常式

除了應用程式和中介軟體邏輯，回應處理常式（有時也指事件處理常式，或活動事件處理常式）亦可新增至內容物件。目前的內容物件發生相關聯回應時，系統在執行實際回應之前會先呼叫處理常式。如果您已經知道想要針對其餘目前回應中，該類型的每個活動要執行哪些動作（無論是在實際事件之前或之後執行），這些處理常式非常實用。

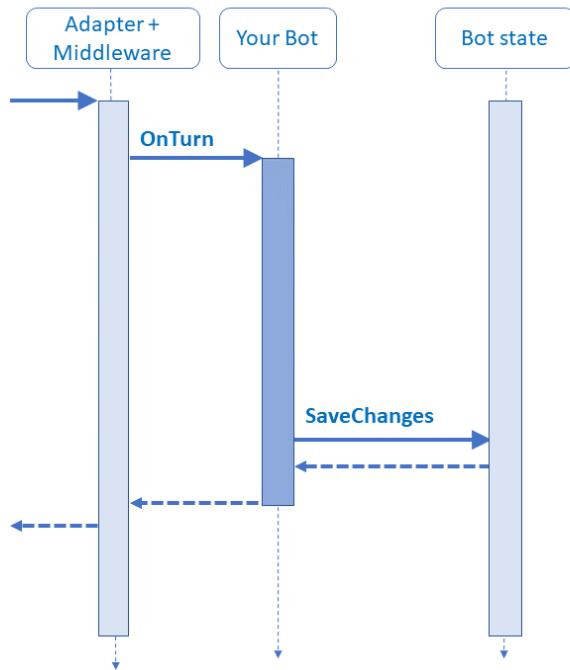
WARNING

請特別小心，不要在其本身的個別回應事件處理常式內，呼叫活動回應方法，例如：從傳送活動處理常式內呼叫傳送活動方法。這樣可能會產生無限迴圈。

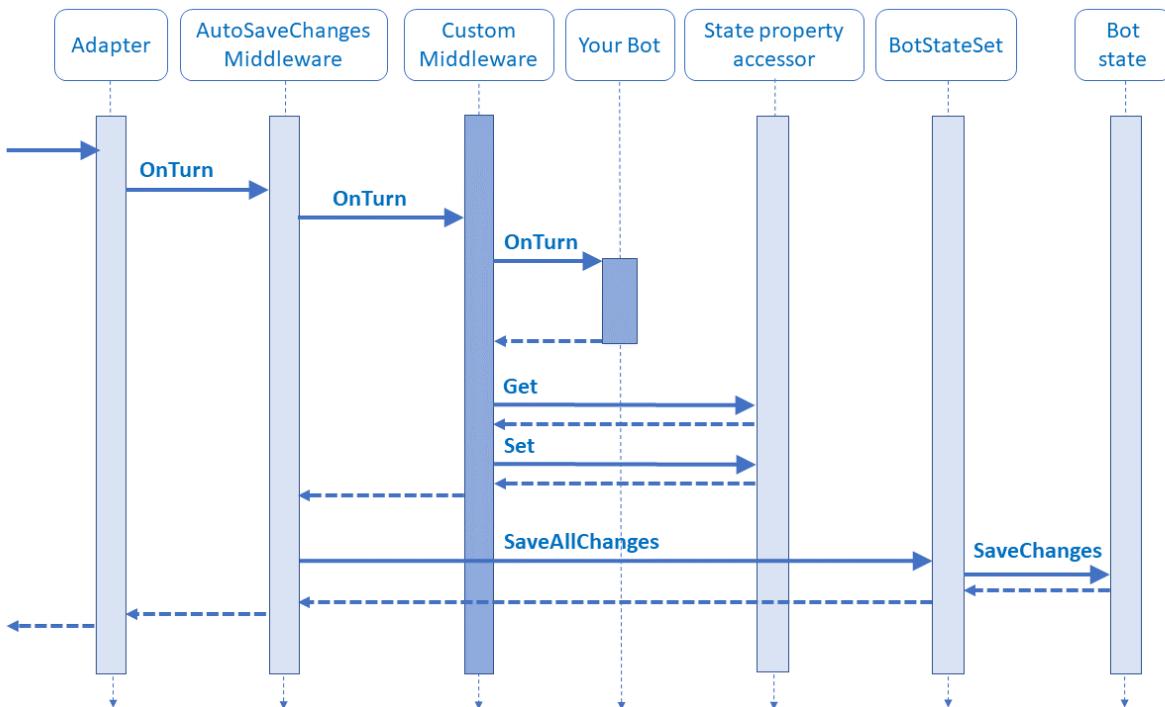
請記住，每個新活動都取得要在其中執行的新執行緒。建立執行緒處理活動時，該活動的處理常式清單會複製到該執行緒。系統不會針對該特定活動事件，執行該時間點後新增的任何處理常式。內容物件上所註冊的處理常式，其處理方式非常類似於介面卡管理中介軟體管線的方式。也就是說，系統將依照處理常式新增的順序進行呼叫，然後再呼叫下一個委派，將控制項傳遞至下個註冊的事件處理常式。如果處理常式未呼叫下個委派，則不會呼叫任何後續的事件處理常式，事件會產生最少運算路由，而且介面卡不會傳送回應給通道。

處理中介軟體中的狀態

儲存狀態的常見方法是在回合處理常式的結尾呼叫「儲存變更」方法。以下是著重於該呼叫的圖表。



此方法的問題是，若是在 Bot 的回合處理常式傳回之後，從某些自訂中介軟體更新任何狀態，則狀態不會儲存到永久性儲存體。解決方法是將 *auto-save changes* 中介軟體的執行個體新增至中介軟體堆疊的開頭，或至少在任何可能更新狀態的中介軟體之前，以將對「儲存變更」方法的呼叫移到完成自訂中介軟體之後。執行如下所示。



將需要更新的狀態管理物件新增至「Bot 狀態集」物件，然後在您建立自動儲存變更中介軟體時使用。

其他資源

您可以看一下文稿記錄器中介軟體，其實作於 Bot Framework SDK 中 [[C#](#) | [JS](#)]。

管理 Bot 資源

2019/5/10 • [Edit Online](#)

適用於:  SDK v4  SDK v3

Bot 通常會取用不同的服務，例如 [LUIS.ai](#) 或 [QnaMaker.ai](#)。當您開發 Bot 時，您需要能夠追蹤一切。您可以使用各種方法，例如 appsettings.json、web.config 或 .env。

IMPORTANT

在 Bot Framework SDK 4.3 發行之前，我們提供 .bot 檔案作為管理資源的機制。不過，我們建議您今後使用 appsettings.json 或 .env 檔案來管理這些資源。即使 .bot 檔案已經「淘汰」，使用 .bot 檔案的 Bot 目前仍會繼續運作。如果您已使用 .bot 檔案來管理資源，請遵循適用於遷移設定的步驟。

從 .bot 檔案遷移設定

下列各節討論如何從 .bot 檔案遷移設定。請遵循您適用的案例。

案例 #1: 具有 .bot 檔案的本機 Bot

在此案例中，您具有使用 .bot 檔案的本機 Bot，但是「Bot 尚未遷移」至 Azure 入口網站。請遵循下列步驟，將設定從 .bot 檔案遷移至 appsettings.json 或 .env 檔案。

- 如果 .bot 檔案已加密，您必須使用下列命令將其解密：

```
msbot secret --bot <name-of-bot-file> --secret "<bot-file-secret>" --clear
```

- 開啟已解密的 .bot 檔案，複製其中的值並將其新增到 appsettings.json 或 .env 檔案。
- 從 appsettings.json 或 .env 檔案將程式碼更新為讀取設定。
- [C#](#)
- [JavaScript](#)

在 `ConfigureServices` 方法中，使用 ASP.NET Core 提供的組態物件，例如：

Startup.cs

```
var appId = Configuration.GetSection("MicrosoftAppId").Value;
var appPassword = Configuration.GetSection("MicrosoftAppPassword").Value;
options.CredentialProvider = new SimpleCredentialProvider(appId, appPassword);
```

如有需要，佈建資源並將其連線到使用 appsettings.json 或 .env 檔案的 Bot。

案例 2: Bot 部署至具有 .bot 檔案的 Azure

在此案例中，您已將 Bot 部署至使用 .bot 檔案的 Azure 入口網站，而您現在想要將設定從 .bot 遷移至 appsettings.json 或 .env 檔案。

- 從 Azure 入口網站下載 Bot 程式碼。當您下載程式碼時，系統會提示您包含 appsettings.json 或 .env 檔案，此檔案會有您的 MicrosoftAppId 和 MicrosoftAppPassword 以及任何其他設定。
- 開啟「已下載」的 appsettings.json 或 .env 檔案，並將其中的設定複製到「本機」appsettings.json 或 .env 檔案。別忘了從本機 appsettings.json 或 .env 檔案中移除 botSecret 和 botFilePath 項目。

- 從 appsettings.json 或 .env 檔案將程式碼更新為讀取設定。

- [C#](#)

- [JavaScript](#)

在 `ConfigureServices` 方法中，使用 ASP.NET Core 提供的組態物件，例如：

Startup.cs

```
var appId = Configuration.GetSection("MicrosoftAppId").Value;
var appPassword = Configuration.GetSection("MicrosoftAppPassword").Value;
options.CredentialProvider = new SimpleCredentialProvider(appId, appPassword);
```

您也需要從 **Azure 入口網站** 的 [應用程式設定] 區段中移除 `botFilePath` 和 `botFileSecret`。

如有需要，佈建資源並將其連線到使用 appsettings.json 或 .env 檔案的 Bot。

案例 3：對於使用 appsettings.json 或 .env 檔案的 Bot

此案例涵蓋以下情況：您使用 SDK 4.3 從頭開始開發 Bot，且沒有要遷移的現有 .bot 檔案。您想要在 Bot 中使用的所有設定都位於 appsettings.json 或 .env 檔案中，如下所示：

```
{
  "MicrosoftAppId": "<your-AppId>",
  "MicrosoftAppPassword": "<your-AppPwd>"
}
```

- [C#](#)

- [JavaScript](#)

若要讀取 C# 程式碼中的上述設定，您可使用 ASP.NET Core 提供的組態物件，例如：**Startup.cs**

```
var appId = Configuration.GetSection("MicrosoftAppId").Value;
var appPassword = Configuration.GetSection("MicrosoftAppPassword").Value;
options.CredentialProvider = new SimpleCredentialProvider(appId, appPassword);
```

如有需要，佈建資源並將其連線到使用 appsettings.json 或 .env 檔案的 Bot。

其他資源

- 如需部署 Bot 的步驟，請參閱[部署](#)主題。
- 了解如何使用 [Azure Key Vault](#) 來保護及管理雲端應用程式和服務所使用的密碼編譯金鑰和密碼。

Bot 服務範本

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

Bot 服務包含五個範本以協助您開始建置 Bot。這些範本會提供現成的完整功能 Bot 以協助您快速入門。當您[建立 Bot](#)時，您會選擇適用於該 Bot 的範本和 SDK 語言。

每個範本都會提供以 Bot 的其中一項核心功能為基礎的起始點。

基本 Bot

若要建立使用對話方塊來回應使用者輸入的 Bot，請選擇 [基本] 範本。[基本] 範本會建立一開始具有最少檔案集合和程式碼的 Bot。Bot 會在使用者輸入時做出回應。您可以使用此範本來開始在 Bot 中建置對話流程。

表單 Bot

若要建立會透過引導式對話收集使用者輸入的 Bot，請選擇 [表單] 範本。[表單] 範本是設計來從使用者收集特定的資訊集合。例如，設計來取得使用者三明治訂單的 Bot，將會需要收集如麵包類型、配料選擇、三明治大小等的資訊。

以 C# 語言搭配 [表單] 範本所建立的 Bot 會使用 [FormFlow](#) 來管理表單，而以 Node.js 語言搭配 [表單] 範本所建立的 Bot 則會使用 [waterfalls](#) 來管理表單。

語言理解 Bot

若要建立使用自然語言模型來了解使用者意圖的 Bot，請選擇 [語言理解] 範本。此範本會利用 [Language Understanding \(LUIS\)](#) (英文) 來提供了解自然語言的功能。

若使用者傳送如 "get news about virtual reality companies" (取得虛擬實境公司的相關新聞) 的訊息，您的 Bot 便可以使用 LUIS 來解譯該訊息的意思。透過使用 LUIS，您可以迅速地部署 HTTP 端點，其將會根據使用者輸入所傳遞的意圖 (尋找新聞) 和存在的主要實體 (虛擬實境公司) 來解譯該輸入。LUIS 可讓您指定與應用程式相關的意圖和實體集合，並引導您完成語言理解應用程式的建置程序。

當您使用 [語言理解] 範本建立 Bot 時，Bot 服務會建立相對應的空白 LUIS 應用程式 (也就是會一律傳回 `None`)。若要更新 LUIS 應用程式模型以使它能解譯使用者輸入，您必須登入 [LUIS](#) (英文)，按一下 [My applications] (我的應用程式)，選取服務為您建立的應用程式，然後建立意圖，指定實體，並將應用程式定型。

問題與答案 Bot

若要建立能將如問題與答案配對的半結構化資料擷取為具區別性且有幫助之答案的 Bot，請選擇 [問題與答案] 範本。此範本會利用 [QnA Maker](#) (英文) 服務來剖析問題並提供答案。

當您使用 [問題與答案] 範本建立 Bot 時，您必須登入 [QnA Maker](#) (英文) 並建立新的 QnA 服務。此 QnA 服務將會提供知識庫識別碼和訂用帳戶金鑰，可讓您用來更新 **QnAKnowledgebaseId** 和 **QnASubscriptionKey** 的**應用程式設定**值。設定好那些值之後，您的 Bot 便能回答 QnA 服務在其知識庫中具有答案的任何問題。

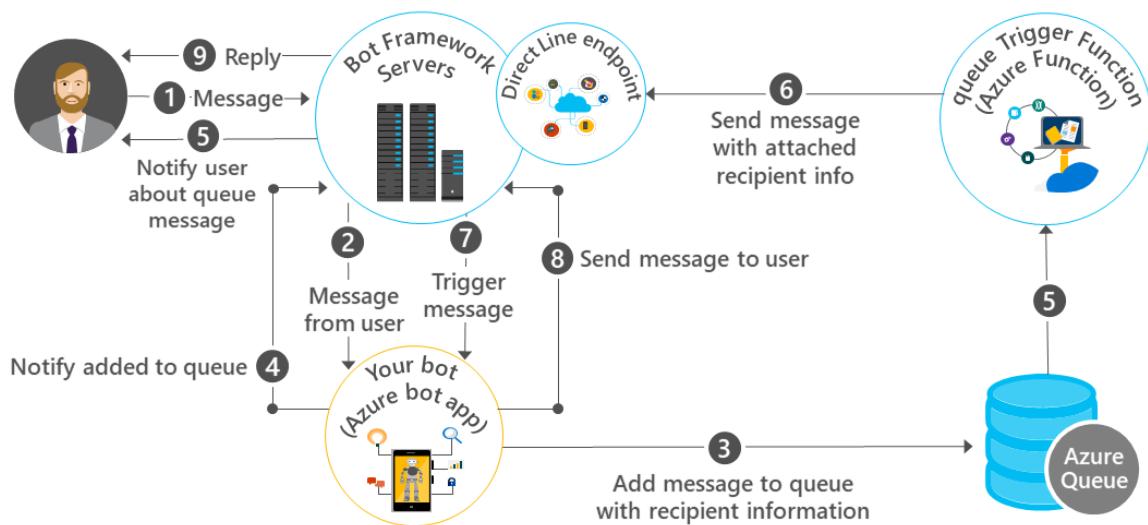
主動式 Bot

若要建立可以傳送主動式訊息給使用者的 Bot，請選擇 [主動式] 範本。一般而言，Bot 直接傳送給使用者的每個訊息都與使用者先前的輸入相關。不過，在某些情況下，Bot 可能需要將與使用者的最新訊息沒有直接關聯的資訊傳送給該使用者。這些類型的訊息稱為主動式訊息。主動式訊息在許多案例中皆很有用。例如，如果 Bot 設定了計時器或提醒，它便可能需要在到達該時間時通知使用者。或者，如果 Bot 接收到有關外部事件的通知，它可能需要將該資訊傳達給使用者。

當您使用 [主動式] 範本建立 Bot 時，系統會自動建立數個 Azure 資源並將它們加入至您的資源群組。根據預設，這些 Azure 資源都會設定為啟用非常簡單的主動式傳訊案例。

資源	說明
Azure 儲存體	用來建立佇列。
Azure 函數應用程式	queueTrigger Azure 函式，會在佇列中有訊息時觸發。它會使用 Direct Line (英文) 與 Bot 服務通訊。此函式會使用 Bot 繫結來以觸發程序承載之一部分的形式傳送訊息。我們的範例函式會從佇列以現況的方式轉送使用者的訊息。
Bot 服務	您的 Bot。包含下列邏輯：接收來自使用者的訊息、將訊息加入至 Azure 佇列、接收來自 Azure 函式的觸發程序，以及透過觸發程序的承載將其所接收到的訊息傳回。

下圖顯示在您使用 [主動式] 範本建立 Bot 時，觸發事件的運作方式。



程序會從使用者透過 Bot Framework 伺服器傳送訊息至您的 Bot 開始 (步驟 1)。

後續步驟

您已了解各種不同的範本，現在請繼續了解用於 Bot 內的認知服務。

Bot 的認知服務

認知服務

2019/2/28 • [Edit Online](#)

您可以藉由 Microsoft 認知服務，應用各領域專家不斷開發之一系列的強大 AI 演算法，包括電腦視覺、語音、自然語言處理、知識擷取和 Web 搜尋。此服務簡化了各種 AI 工作，只要幾行程式碼，您就能快速在 Bot 中新增最新的智慧技術。API 整合了大部分的新語言和平台。API 亦可持續改善與學習，變得更聰明，讓服務體驗隨時保持最新狀態。

智慧 Bot 回應的方式就如同真人思考。他們會從不同來源發掘資訊並擷取知識，以提供實用的解答，最重要的是，還可從每次累積的經驗中學習，持續精進本身能力。

語言理解

使用者與 Bot 的互動大多格式自由，因此 Bot 需要了解自然語言及與內容相關的語言。認知服務語言 API 提供強大的語言模型，可協助您判斷使用者想要什麼、識別句子中的概念和實體，最終讓您的 Bot 以適當的動作來回應。這五個 API 支援多種文字分析功能，例如拼字檢查、情感偵測、語言模型，且可從文字中擷取正確且豐富的深入解析。

認知服務能為這些 API 提供語言理解能力：

- [Language Understanding Intelligent Service \(LUIS\)](#) 可使用預先建置或自訂訓練的語言模型處理自然語言。
如需更多詳細資訊，請參閱 [Bot 的 Language Understanding](#)
- [文字分析 API](#) 可從文字中偵測情緒、主要片語、主題和語言。
- [Bing 拼字檢查 API](#) 提供強大的拼字檢查功能，且可辨識名稱、品牌名稱和俚語之間的差異。
- [Azure Machine Learning Studio 中的文字分析模型](#) 可讓您建置及操作文字分析模型，例如：詞形還原及文字預先處理。這些模型可協助您解決文件分類或情緒分析等問題。

深入了解採用 Microsoft 認知服務的 [Language Understanding](#)。

知識擷取

認知服務提供四種知識 API，可讓您識別非結構化文字中的具名實體或片語、新增個人化建議、根據自然解譯使用者查詢提供自動完成建議，以及搜尋學術文章及其他研究，如同個人化常見問答集服務。

- [實體連結智慧服務](#) 可使用文字中提及的相關實體標註非結構化文字。視內容而定，相同的字詞或片語可能指涉不同事物。此服務了解所提供之文字中的情境，且能識別您文字中的每個實體。
- [知識探索服務](#) 可解譯自然語言使用者查詢字串，再傳回加上註釋的解譯，進而支援可預期使用者輸入內容的豐富搜尋及自動完成功能體驗。立即查詢完成建議和預期查詢調整皆以您自己的資料和應用程式特定文法為依據，可讓使用者執行更快的查詢。
- [學術知識 API](#) 可從 [Microsoft Academic Graph](#) 傳回學術研究文章、作者、期刊、研討會、主題和大學資訊。學術知識 API 以網域特定形式的範例內建於知識探索服務，可使用圖表式的對話方塊提供知識庫，且可從數千萬筆研究相關實體中搜尋內容。搜尋主題、教授、大學或研討會，然後 API 為提供相關的出版刊物及相關實體。其文法亦支援自然查詢如：「Michael Jordan 在 2010 年後有關機器學習的研究文章」。
- [QnA Maker](#) 是一項簡單易用的免費 REST API 及 Web 服務，可訓練 AI 以自然的交談方式回應使用者問題。QnA Maker 具備最佳化機器學習邏輯，以及整合領先業界語言處理技術的能力，可抽取半結構化資料，例如：由問與答組成清楚且實用的解答。

深入了解採用 Microsoft 認知服務的 [知識擷取](#)。

語音辨識和轉換

使用語音 API 為 Bot 新增進階語音技術，即可運用領先業界的語音轉文字演算法及語音辨識功能。語音 API 採用內建的語言和原音模式，其可高準確涵蓋各種情境。

對於需要進一步自訂的應用程式，您可使用自訂辨識智慧型服務 (CRIS)。此可讓您根據應用程式的詞彙，甚至使用者的說話方式進行調整，以自訂語音辨識器的原音模型。

認知服務中有三個語音 API 可處理或合成語音：

- [Bing 語音 API](#) 支援語音轉文字和文字轉語音功能。
- [自訂辨識智慧型服務 \(CRIS\)](#) 可讓您建立自訂語音辨識模型，並根據應用程式的詞彙或使用者的說話方式調整語音轉文字設定。
- [說話者辨識 API](#) 可透過聲音辨識及驗證說話者。

下列資源提供有關新增語音辨識功能至 Bot 的額外資訊。

- [應用程式的 Bot 交談影片概觀](#)
- [適用於 UWP 或 Xamarin 應用程式的 Microsoft.Bot.Client 程式庫](#)
- [Bot 用戶端程式庫範例](#)
- [支援語音的 WebChat 用戶端](#)

深入了解採用 Microsoft 認知服務的[語音辨識和轉換](#)。

Web 搜尋

Bing 搜尋 API 可讓您新增智慧 Web 搜尋功能至 Bot。只要使用幾行程式碼，即可存取數十億筆網頁、影像、影片、新聞及其他結果類型。您可設定 API 按地理位置、市場或語言傳回結果，以提升相關性。您可使用支援的搜尋參數進一步自訂搜尋，例如：使用 Safesearch 篩選出成人內容，或使用 Freshness 根據特定日期傳回結果。

認知服務中有五種可用的 Bing 搜尋 API。

- [Web 搜尋 API](#) 可透過單一 API 呼叫提供網頁、影像、影片、新聞和相關搜尋結果。
- [影像搜尋 API](#) 可傳回具有增強中繼資料（主色、影像類型等等）的影像結果，並支援多種影像篩選功能以便自訂結果。
- [影片搜尋 API](#) 可擷取具有豐富中繼資料（影片大小、品質、價格等等）的影片結果、影片預覽，並支援多種影像篩選功能以便自訂結果。
- [新聞搜尋 API](#) 可尋找符合搜尋查詢的全球新聞報導，或目前在網際網路上的熱門趨勢話題。
- [自動建議 API](#) 提供立即查詢完成建議，可讓您用更少的輸入字元，更快完成搜尋查詢。

深入了解採用 Microsoft 認知服務的 [Web 搜尋](#)。

影像與影片理解

視覺 API 可為您的 Bot 新增進階影像和影片理解技術。最新驗算法可讓您處理影像或影片，並從中取得相關資訊以採取動作。例如，您可使用這些資訊辨識物件、人臉、年齡、性別甚至情緒。

視覺 API 支援各種影像理解功能。其可識別完整或明確的內容、預估並強調輔色、分類影像內容、進行光學字元辨識，且可以完整英文句子描述影像。視覺 API 亦支援多種影像和影片處理功能，例如以智慧方式產生影像或影片縮圖，或穩定影片輸出。

認知服務提供四種您可用於處理影像或影片的 API：

- [電腦視覺 API](#) 可擷取與影像相關的豐富資訊（例如物件或人員）、判斷影像是否包含完整或明確的內容，以及（使用 OCR）處理影像中的文字。

- [表情 API](#) 可分析人類臉部表情，並以人類可能出現的八種情緒分類來辨識情緒。
- [臉部 API](#) 可偵測人臉、與類似的臉比較，甚至根據視覺上的相似程度將人員分組。
- [影片 API](#) 可分析並處理影片，以穩定影片輸出、偵測動作、追蹤臉部，且可產生影片的動態縮圖摘要。

深入了解採用 Microsoft 認知服務的[影像和影片理解](#)。

其他資源

如需每項產品的完整文件及其對應的 API 參考資料，請參閱[認知服務文件](#)。

Bot 案例

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

本主題會針對使用 Bot Service 所建置強大且成功的 Bot，探索其重要案例。

您可以從[常見的 Bot Framework 案例範例](#)下載或複製所有案例 Bot 範例的原始程式碼。

商務 Bot 案例

[商務 Bot](#) 案例說明 Bot 如何取代傳統電子郵件和電話互動等常見的旅館接待服務。Bot 會利用認知服務，以更順暢地處理客戶的文字和語音要求，並搭配蒐集來自與後端服務整合的內容。

在商務 Bot 案例中，客戶可以向旅館提出門房服務要求。其會透過 Azure Active Directory v2 驗證端點進行驗證。Bot 可以查閱客戶的預約，並提供不同服務選項。例如，客戶可能預訂泳池畔的小屋。Bot 會使用 Language Understanding Intelligent Services (LUIS) 剖析要求，然後 Bot 會逐步引導使用者進行針對現有預約，來預訂小屋的程序。

Cortana 技能 Bot 案例

[Cortana 技能 Bot](#) 案例會利用 Cortana。使用語音的自然介面和自訂的 Cortana 技能 Bot，您可以讓 Cortana 與組織（如行動汽車美容公司）交談，以協助您根據撥打電話時的所在地進行預約。此 Bot 可以提供一份服務、可用時間和持續時間的清單。

企業生產力 Bot 案例

[企業生產力 Bot](#) 案例會說明如何整合 Bot 與 Office 365 行事曆和其他服務，以提高生產力。

Bot 會與 Office 365 整合，以便更快速且更輕鬆地建立與其他人開會的要求。在這麼做的過程中，您可以存取 Dynamics CRM 等其他服務。此範例會提供所需程式碼，以供透過 Azure Active Directory 進行驗證來與 Office 365 整合。它會為外部服務提供模擬進入點，以作為讀者的練習。

資訊 Bot 案例

[資訊 Bot](#) 可使用認知服務 QnA Maker 回答在知識集或常見問題集中定義的問題，也會使用 Azure 搜尋服務回答更開放式的問題。

通常資訊會藏在 SQL Server 等結構化的資料存放區，可以透過搜尋輕鬆地呈現。想像一下使用簡單的交談式命令查閱客戶的訂單狀態。使用認知服務 QnA Maker，使用者會看到一組有效的搜尋選項，例如查閱客戶、檢閱客戶的最新訂單等。使用者可以利用 QnA 格式定義，輕鬆詢問 Azure 搜尋支援的問題，並在 SQL Database 中查閱儲存的資料。

IoT Bot 案例

這個[物聯網 \(IoT\) Bot](#) 可讓您輕鬆地使用互動式聊天命令控制住家內的裝置，例如 Philips Hue 燈光。

您可以使用這個簡單的 Bot，搭配免費的 If This Then That (IFTTT) 服務來控制 Philips Hue 燈光。身為 IoT 裝置，Philips Hue 可透過其公開的 API 在本機進行控制。不過，此 API 並未公開用於從本機網路外部所進行的一般存

取。不過，IFTTT 是 "Friend of Hue"，因此，已公開多個您可以發出的控制命令，例如，開燈和關燈、變更燈色或光源強度。

後續步驟

您現已大致了解各種案例，接下來請深入了解每個案例。

商務 Bot

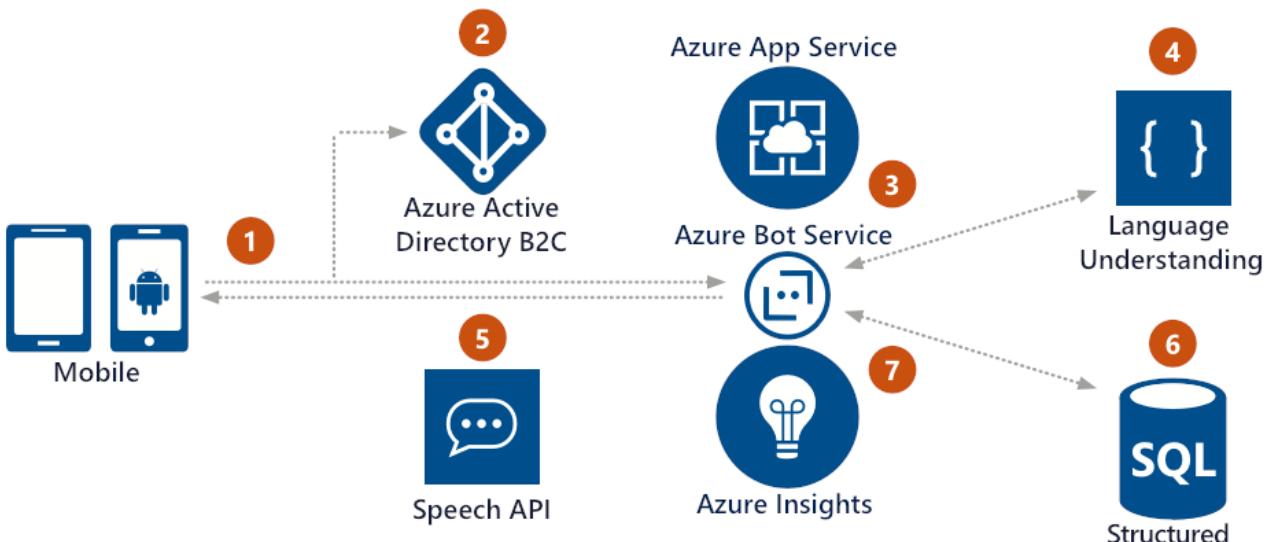
商務 Bot 案例

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

商務 Bot 案例說明 Bot 如何取代傳統電子郵件和電話互動等常見的旅館接待服務。Bot 會利用認知服務，以更順暢地處理客戶的文字和語音要求，並搭配蒐集自與後端服務整合的內容。



以下是作為旅館接待的商務 Bot 邏輯流程：

1. 客戶可使用旅館的行動應用程式。
2. 使用者透過 Azure AD B2C 進行驗證。
3. 使用者透過自訂應用程式 Bot 來要求資訊。
4. 認知服務可協助處理自然語言要求。
5. 回應會由能利用自然對話修正問題的客戶進行檢閱。
6. 若使用者滿意該結果，應用程式 Bot 就會更新客戶的預約。
7. Application Insights 會收集執行階段遙測，以協助開發 Bot 效能和使用量。

範例 Bot

範例商務 Bot 的設計訴求是打造虛構的旅館接待服務。在透過鏈結的成員服務行動應用程式向旅館驗證 Azure AD B2C 後，客戶可存取以 C# 撰寫的 Bot。鏈結會將預約儲存在 SQL Database 中。客戶可以使用自然的問句，如「住宿期間租一間泳池小屋的費用多少」。Bot 因此會擁有來賓住在哪間旅館和住宿期間的相關內容。此外，Language Understanding (LUIS) 服務可讓 Bot 從簡單的詞語如「泳池小屋」等輕鬆取得內容。Bot 會提供答案，然後為來賓預訂小屋，並提供天數和小屋類型的選擇。在 Bot 擁有所有必要資料後，即會依要求進行預訂。來賓也可以使用語音進行相同的要求。

您可以從[常見的 Bot Framework 案例範例](#)下載或複製此範例 Bot 的來源程式碼。

您將使用的元件

商務 Bot 會使用下列元件：

- 用以驗證的 Azure AD
- 認知服務: LUIS
- Application Insights

Azure Active Directory (Azure AD)

Azure Active Directory (Azure AD) 是 Microsoft 的多租用戶雲端型目錄和身分識別管理服務。身為 Bot 開發人員, Azure AD 可讓您專注於建置 Bot, 快速而簡單地整合數百萬個全球各地組織所使用的世界級身分識別管理解決方案。Azure AD 支援 B2C 連接器, 可讓您使用外部識別碼 (例如 Google、Facebook 或 Microsoft 帳戶) 來識別個人。Azure AD 讓您不必管理使用者的認證, 改而專注於 Bot 的解決方案, 了解您可以將 Bot 使用者與應用程式所公開的正確資料相互關聯。

認知服務: LUIS

作為認知服務系列技術之一員, Language Understanding (LUIS) 會為您的應用程式帶來機器學習的強大功能。目前, LUIS 支援數種語言, 可讓 Bot 了解人想要什麼。與 LUIS 整合時, 您可表達意圖並定義 Bot 能了解的實體。然後使用範例語句進行訓練, 教導 Bot 了解這些意圖和實體。您可以使用片語清單和 RegEx 功能調校整合, 針對特定的對話需求讓 Bot 盡可能順暢。

Application Insights

Application Insights 可協助您透過應用程式效能管理和立即分析, 取得可操作的見解。根據預設, 您可以取得豐富的效能監視、強大的警示與便於取用的儀表板, 以協助確保您的 Bot 可供使用, 並如預期般執行。您可以快速查看是否遇到問題, 然後執行根本原因分析以找出問題, 並加以修正。

後續步驟

接下來, 深入了解 Cortana 技能 Bot 案例。

[Cortana 技能 Bot 案例](#)

Cortana 技能 Bot 案例

2019/2/28 • [Edit Online](#)

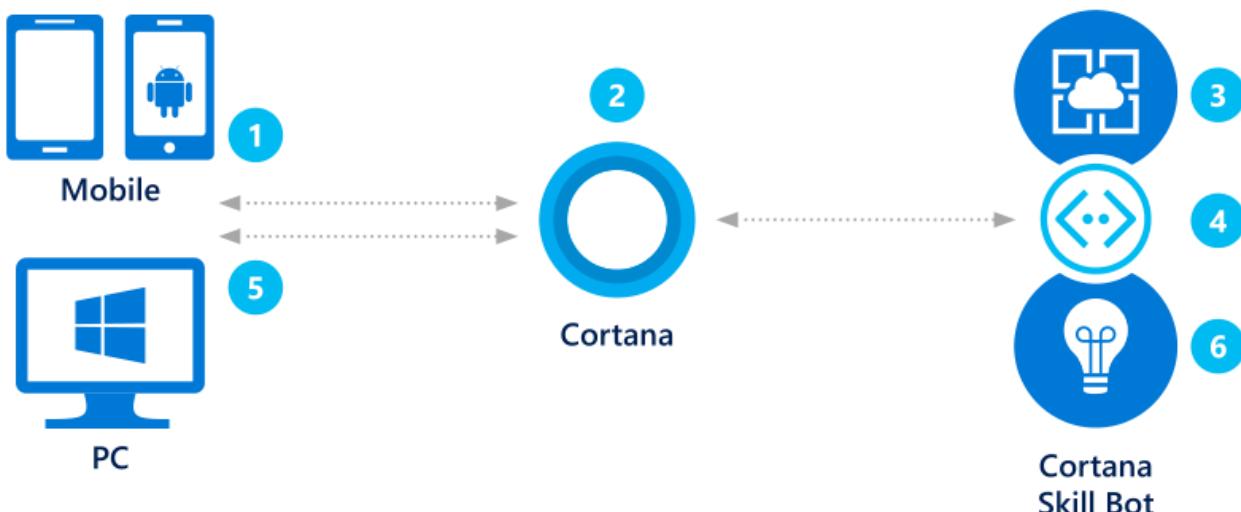
NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

Cortana 技能 Bot 擴充了 Cortana，可以輕鬆地從您行事曆的內容，使用語音預訂行動裝置自動維護預約。

Cortana 是您的個人助理。使用您的語音和自訂的 Cortana 技能 Bot 的自然介面，您可以讓 Cortana 與組織（如汽車商店）交談，以協助您預約。該服務可以提供一份服務、可用時間和持續時間的清單。Cortana 可以查看您的行事曆，以查看您是否在衝突的時間有某些內容，如果沒有，則建立約會並將它新增至您的行事曆中。

Cortana Skill Bot



以下是 Cortana 技能 Bot 針對汽車商店的邏輯流程：

1. 使用者從他們的電腦或行動裝置存取 Cortana。
2. 使用文字或語音命令，使用者要求進行汽車保養預約。
3. 由於 Bot 與 Cortana 整合，因此它可以存取使用者的行事曆，並將邏輯套用至要求。
4. 使用該資訊，Bot 可以查詢自動服務以取得有效的約會。
5. 使用者可以利用內容相關的選項，來預約約會。
6. Application Insights 會收集執行階段遙測，以協助開發 Bot 效能和使用量。

範例 Bot

使用 Cortana 技能 Bot 完全取決於個人背景。使用 Cortana，您可以使用語音要求「Bob 的行動裝置維護」根據您的位置來處理您的車輛。使用透過 Cortana 公開的個人資訊，您的 Bot 可以根據使用者與 Bot 交談時的位置來確認位置。

您可以從[常見的 Bot Framework 案例範例](#)下載或複製此範例 Bot 的原始程式碼。

您將使用的元件

Cortana Bot 會使用下列元件：

- Cortana
- Application Insights

Cortana

現在，您可以透過建立 Cortana 技能，將支援新增至您的 Bot。您可以使用 Cortana 技能集為 Cortana 建置新功能（稱為技能）。技能是一種允許 Cortana 做更多事情的建構。您可以建置與 Bot 整合的技能，從而讓 Cortana 完成工作。作為叫用程序的一部分，Cortana 可以（在使用者同意之後）在執行階段將使用者的相關資訊傳遞給技能，以便技能可以相應地自訂其體驗。Cortana 的內容相關的知識可讓您的 Bot 變得實用，甚至可能更聰明。一旦叫用，特定類型的技能可以操作 Cortana 的介面，以在技能和使用者之間進行交談。發佈之後，使用者就可以在 Cortana for Windows 10 年度更新+（電腦和行動裝置）、iOS 和 Android 上查看和使用您的技能。

Application Insights

Application Insights 可協助您透過應用程式效能管理和立即分析，取得可操作的見解。根據預設，您可以取得豐富的效能監視、強大的警報與便於取用的儀表板，以協助確保您的 Bot 可供使用，並如預期般執行。您可以快速查看是否遇到問題，然後執行根本原因分析以找出問題，並加以修正。

後續步驟

接下來，了解企業生產力 Bot 案例。

[企業生產力 Bot 案例](#)

企業生產力 Bot 案例

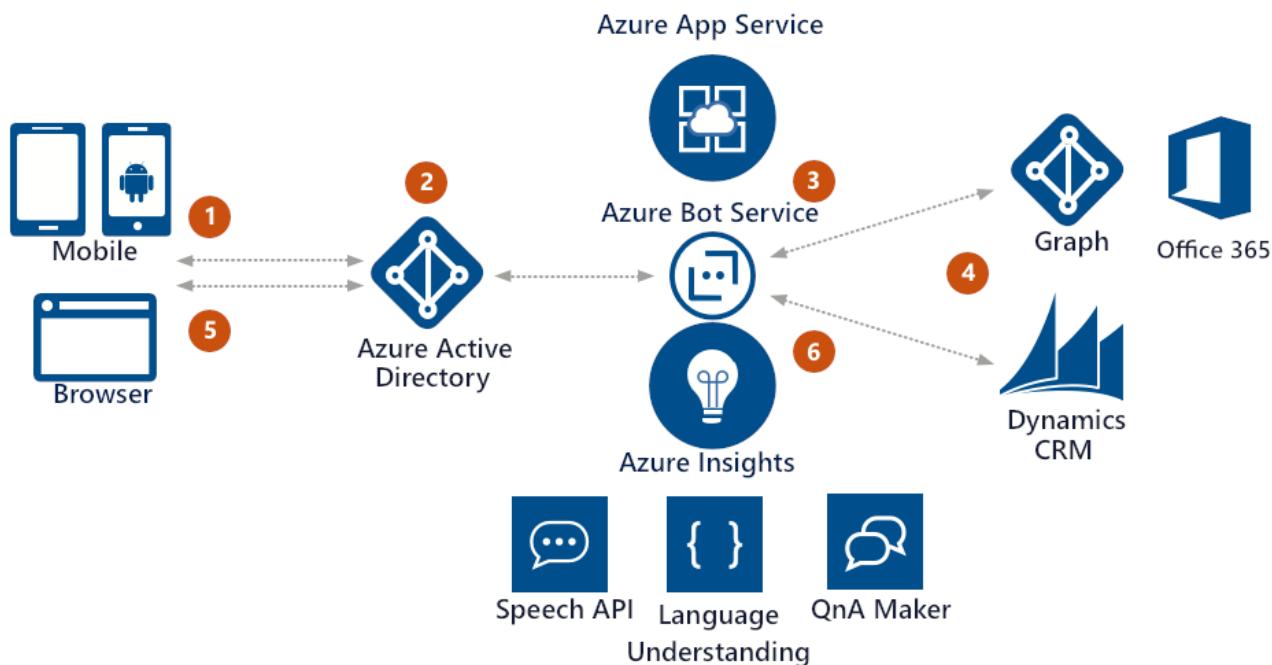
2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

企業 Bot 說明如何將 Bot 與您的 Office 365 行事曆和其他服務整合，來提升您的生產力。

企業生產力 Bot 的重點在於能快速地存取客戶資訊，而不需要開啟一大堆的視窗。使用簡單的聊天命令，銷售人員便能查閱客戶資料，並透過圖形 API 和 Office 365 檢查下一個約會。他們可以從中存取儲存在 Dynamics CRM 的客戶特定資訊，例如取得案例資訊，或是建立新的案例。



以下是企業生產力 Bot 的邏輯流程：

- 員工存取企業生產力 Bot。
- Azure Active Directory 驗證該員工的身份。
- 企業生產力 Bot 可透過 Azure Graph 查詢該員工的 Office 365 行事曆。
- Bot 可利用從行事曆收集而來的資料，存取 Dynamics CRM 中的案例資訊。
- 此資訊會傳回給員工，他可以在不離開 Bot 的情況下篩出資料。
- Application Insights 會收集執行階段遙測，以協助開發 Bot 效能和使用量。

您可以從[常見的 Bot Framework 案例範例](#) (英文) 下載或複製此範例 Bot 的原始程式碼。

範例 Bot

因為 Bot 可透過各種通道來存取，只需要經過驗證，您就可以在辦公室時透過公司入口網站來存取，或在外出時透過 Skype 來存取。透過與 Azure AD 整合，您的企業生產力 Bot 就會知道您是否已經過 Azure AD 驗證，進而得知您是否能夠存取它。接著，您就可以要求 Bot，以檢查與下一個特定客戶的約會時間。Bot 是藉由透過圖形 API 查詢 Office 365 來取得此資訊。然後，如果接下來一星期有約會，Bot 會查詢 CRM，尋找該客戶任何近期的案例資訊。Bot 會回應找不到任何案例，或是數個已開立和已結案的案例。您可以在該處要求 Bot 依類型列出案例，並向下鑽研個別案例。

您將會使用的元件

企業生產力 Bot 使用下列元件：

- 用於驗證的 Azure AD
- 用於 Office 365 的圖形 API
- Dynamics CRM
- Application Insights

Azure Active Directory (Azure AD)

Azure Active Directory (Azure AD) 是 Microsoft 的多租用戶雲端型目錄和身分識別管理服務。身為 Bot 開發人員，Azure AD 可讓您專注於建置 Bot，快速而簡單地整合數百萬個全球各地組織所使用的世界級身分識別管理解決方案。透過定義 Azure AD 應用程式，您可以控制哪些人可存取您的 Bot 以及其所公開的資料，而不需要自己實作複雜的驗證和授權系統。

用於 Office 365 的圖形 API

Microsoft Graph 透過位於 <https://graph.microsoft.com> 的單一端點，公開多個 Office 365 的 API 和其他 Microsoft 雲端服務。Microsoft Graph 可讓您和 Bot 更輕鬆地執行查詢。API 公開多個 Microsoft 雲端服務的資料，包括屬於 Office 365 一部分的 Exchange Online、Azure Active Directory、SharePoint 等等。您可以使用 API 在實體和關聯性之間瀏覽。您可以使用 SDK 或 REST 端點，以及其他原生支援 Android、iOS、Ruby、UWP、Xamarin 等的應用程式，從 Bot 使用 API。

Dynamics CRM

Dynamics CRM 是客戶業務開發平台。使用 Bot 和 CRM API，您可以建置豐富的互動式 Bot，並存取儲存在 CRM 中的大量資料。Dynamics CRM 的強大功能，可供您的 Bot 用來建立案例、檢查狀態、知識管理搜尋等等。

Application Insights

Application Insights 可協助您透過應用程式效能管理和立即分析，取得可操作的見解。根據預設，您可以取得豐富的效能監視、強大的警報與便於取用的儀表板，以協助確保您的 Bot 可供使用，並如預期般執行。您可以快速查看是否遇到問題，然後執行根本原因分析以找出問題，並加以修正。

後續步驟

接下來，深入了解 Bot 案例。

[資訊 Bot 案例](#)

資訊 Bot 案例

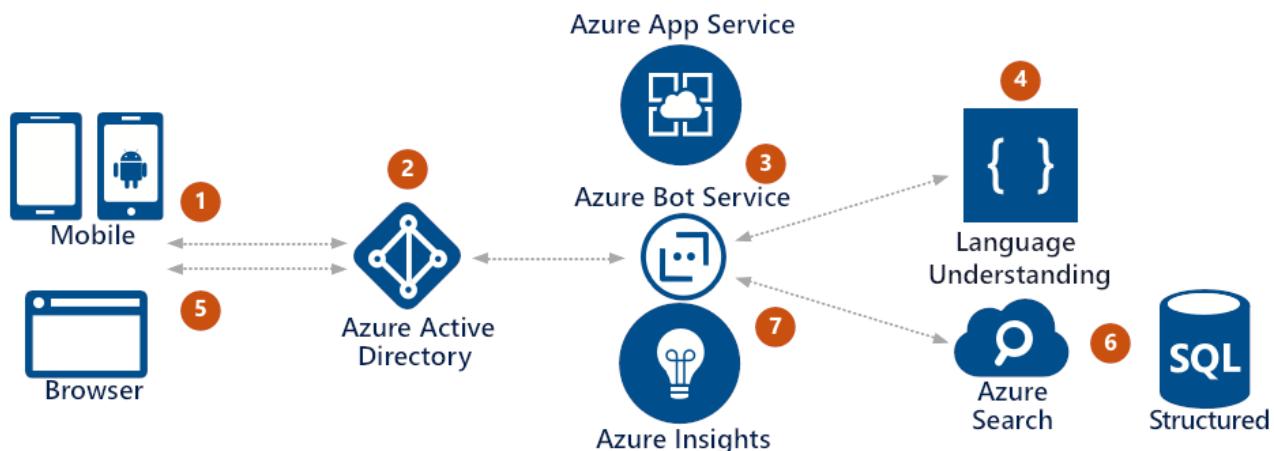
2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

此資訊 Bot 可透過認知服務 QnA Maker 回答在知識集或常見問題集中定義的問題，也會透過 Azure 搜尋服務回答更開放式的問題。

通常資訊會藏在 SQL Server 等結構化的資料存放區，不過透過搜尋即可輕鬆找到。想像一下如此方便的情境：使用簡單的交談式命令查閱客戶的訂單狀態。使用者使用認知服務 QnA Maker 後，會看到一組有效的搜尋選項，例如查閱客戶、檢閱客戶的最新訂單等。使用者可以利用 QnA 格式定義，輕鬆詢問 Azure 搜尋服務所支援的問題，並在 SQL Database 中查閱儲存的資料。



以下是資訊 Bot 的邏輯流程：

1. 員工啟動資訊 Bot。
2. Azure Active Directory 驗證該員工的身分。
3. 員工可詢問機器人支援何種查詢。
4. 認知服務傳回使用 QnA Maker 建置的常見問題集 Bot。
5. 員工定義有效的查詢。
6. Bot 將查詢提交至 Azure 搜尋服務，搜尋服務再傳回應用程式資料的相關資訊。
7. Application Insights 會收集執行階段遙測，以協助開發 Bot 效能和使用量。

範例 Bot

範例 Bot 以 C# 撰寫，在 Microsoft Azure 中執行，並搭配 Azure 搜尋服務從 SQL Database 執行個體索引的資料使用。Bot 會公開一份問題清單，以詢問如何使用認知服務將問題（答案）分段：QnA Maker。Bot 的使用者可以輸入查詢，以透過 Azure 搜尋服務，在索引資料庫中的廣泛或特定區域中查閱資料。此範例提供簡單的資料庫與客戶和訂單資訊。Application Insights 可追蹤 Bot 的使用方式，並協助您監視例外狀況的 Bot。Bot 發行為 Azure AD 應用程式，讓您可以限制存取資訊的人選。

您可以從[常見的 Bot Framework 案例範例](#)下載或複製此範例 Bot 的原始程式碼。

您將會使用的元件

資訊 Bot 會使用下列元件：

- 用於驗證的 Azure AD
- 認知服務 : QnA Maker
- Azure 搜尋服務
- Application Insights

Azure Active Directory (Azure AD)

Azure Active Directory (Azure AD) 是 Microsoft 的多租用戶雲端型目錄和身分識別管理服務。身為 Bot 開發人員，Azure AD 可讓您專注於建置 Bot，快速而簡單地整合數百萬個全球各地組織所使用的世界級身分識別管理解決方案。您可以藉由定義 Azure AD 應用程式，來控制哪些人可存取您的 Bot，以及其所公開的資料，而不需要自己實作複雜的驗證和授權系統。

認知服務 : QnA Maker

認知服務 QnA Maker 可協助您提供常見問題集的資料來源，讓使用者可以從 Bot 進行查詢。遇到儲存在不同系統中的大量資訊時，此服務可以有效協助使用者篩選資訊來源和資訊集。單一 SQL 資料庫擁有大量資訊，以至於執行自由格式搜尋時會提供過多資訊。初次使用 QnA Maker 時，您可以為 Bot 使用者定義藍圖，讓使用者了解如何問對問題，以透過 Azure 搜尋服務找到答案。

Azure 搜尋服務

Azure 搜尋服務是應用程式專用的雲端搜尋服務，可讓您快速啟用搜尋索引並執行。在 Microsoft Azure 上執行，您可以視需求輕鬆增加或減少使用量。您可以透過對搜尋排名的極佳控制，並找出隱藏在資料庫中的資料，讓搜尋結果促成商業目標。

Application Insights

Application Insights 可協助您透過應用程式效能管理和立即分析，取得可操作的見解。根據預設，您可以取得豐富的效能監視、強大的警報與便於取用的儀表板，以協助確保您的 Bot 可供使用，並如預期般執行。您可以快速查看是否遇到問題，然後執行根本原因分析以找出問題，並加以修正。

後續步驟

接下來，請深入了解物聯網 Bot 案例。

[物聯網 Bot 案例](#)

物聯網 (IoT) Bot 案例

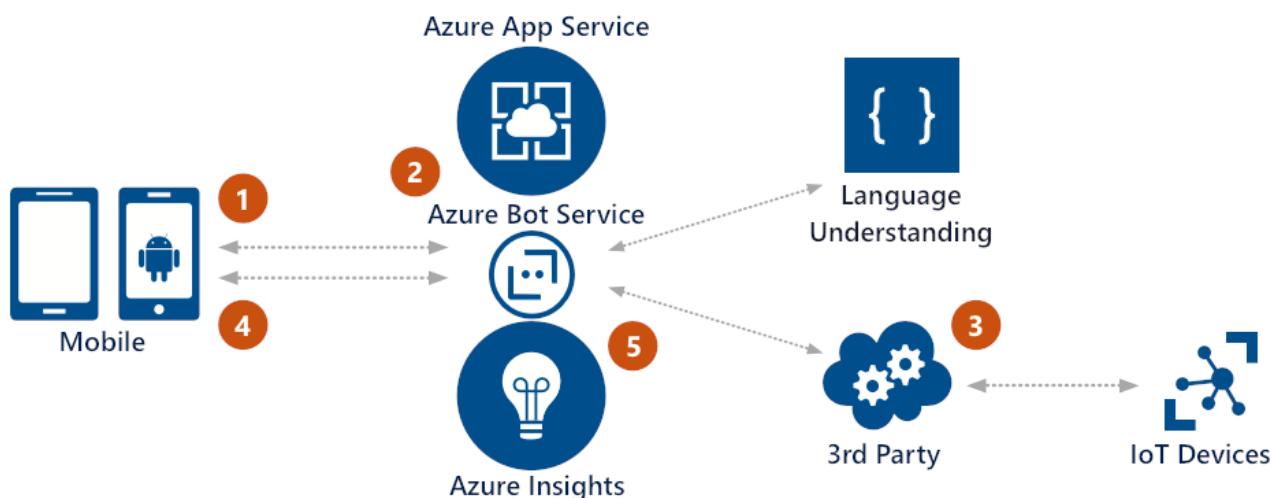
2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

這個物聯網 (IoT) Bot 可讓您輕鬆地使用語音或互動式聊天命令控制住家內的裝置，例如 Philips Hue 燈光。

人們喜歡以對話來操作自己的裝置。自從第一台電視遙控器問世以來，人們就愛上這種不需要移動即可影響環境的操作方式。此 IoT Bot 讓使用者透過簡單的聊天命令或語音來管理 Philips Hue。此外，使用聊天功能時，使用者可獲得與色彩相關的視覺化選項。



以下是 IoT Bot 的邏輯流程：

1. 使用者登入 Skype 並存取 IoT Bot。
2. 使用者透過語音要求 Bot 經由 IoT 裝置開啟燈光。
3. 將該要求轉送至可存取 IoT 裝置網路的協力廠商服務。
4. 將該命令結果傳回給使用者。
5. Application Insights 會收集執行階段遙測，以協助開發 Bot 效能和使用量。

範例 Bot

IoT Bot 將能讓您快速地使用 Skype 或 Slack 等頻道的聊天命令，來控制您的 Hue。為了方便遠端存取，您可以呼叫預先定義的 IFTTT 小程式來搭配 Hue 使用。

您可以從[常見的 Bot Framework 案例範例](#)下載或複製此範例 Bot 的原始程式碼。

您將使用的元件

物聯網 (IoT) Bot 會使用下列元件：

- Philips Hue
- If This Then That (IFTTT)
- Application Insights

Philips Hue

與 Philips Hue 連接的燈泡和橋接器可讓您充分掌握您的照明設備。無論您想要如何處理自己的照明設備，Hue 都能辦到。Hue 備有您可透過區域網路使用的 API。不過，如果您想要從任何地方使用簡單好用的 Bot 介面存取由 Hue 控制的裝置和照明設備，請透過 IFTTT 存取 Hue。

IFTTT

IFTTT 是免費的 Web 型服務，可供使用者建立簡單的條件陳述式鏈結（稱為小程式）。您可以從自己的 Bot 觸發小程式來代替您執行一些動作。有許多預先定義的 Hue 小程式可用來開關燈光、變更場景等。

Application Insights

Application Insights 可協助您透過應用程式效能管理和立即分析，取得可操作的見解。根據預設，您可以取得豐富的效能監視、強大的警報與便於取用的儀表板，以協助確保您的 Bot 可供使用，並如預期般執行。您可以快速查看是否遇到問題，然後執行根本原因分析以找出問題並加以修正。

Bot 設計準則

2019/2/28 • [Edit Online](#)

Bot Framework 可讓開發人員建立引人注目的 Bot 體驗，解決各種商務問題。藉由學習本節所述的概念，您就能夠設計符合最佳做法的 Bot，並且在這個相對而言較新的領域中，利用至今所學的課程。

設計 Bot

如果您要建置 Bot，可以肯定地說您預期使用者會加以使用。我們也可以肯定地說，您會希望使用者偏好 Bot 體驗而不是替代方案（例如應用程式、網站、電話通話及其他解決其特定需求的方式）。換句話說，您的 Bot 會與應用程式和網站等項目爭用使用者的時間。那麼，該如何讓您的 Bot 穩操勝券，達到吸引和保留使用者的最終目標？其實只要在設計 Bot 時，決定正確因素的優先順序，就這麼簡單而已。

不保證 Bot 成功的因素

在設計您的 Bot 時，請注意下列因素都不保證 Bot 成功：

- **Bot 有多「聰明」：**在大部分情況下，讓您的 Bot 更有智慧，並不保證使用者在採用您的平台時就能夠更滿意。事實上，許多 Bot 有一些進階機器學習或自然語言功能。如果 Bot 需要解決其設計目的的問題時，就會包含這些功能，但是您不應該假設 Bot 智慧與使用者採用 Bot 之間有任何相互關聯。
- **Bot 支援多少自然語言：**您的 Bot 可以很適合對話。它可以有大量的詞彙，甚至笑話可以說得很好。但是，除非它能夠解決您的使用者需要解決的問題，否則這些功能對於讓您的 Bot 成功並無太大助益。事實上，有些 Bot 完全沒有對話功能。而在許多情況下，這完全沒有問題。
- **語音：**讓 Bot 具有語音功能，不一定可得到良好的使用者體驗。通常，強制使用者使用語音可能會導致令人沮喪的使用者體驗。當您設計 Bot 時，請一定要考慮語音是否能有效解決指定的問題。那裡會不會是吵雜的環境？語音能夠傳達需要與使用者共用的資訊嗎？

可讓 Bot 成功的因素

大部分成功的應用程式或網站都有至少一個共通之處：絕佳的使用者體驗。這點對各種 Bot 都是一樣的。因此，確保絕佳的使用者體驗，應該是您在設計 Bot 時的第一要務。一些主要考量包括：

- Bot 可以使用最少的步驟，輕鬆地解決使用者的問題嗎？
- Bot 可以比其他任何替代方案，更有效/更輕易/更快速地解決使用者的問題嗎？
- Bot 可以在使用者關切的裝置和平台上執行嗎？
- Bot 是否可供探索？使用者在使用 Bot 時，是否自然知道要怎麼做？

以上問題與 Bot 有多聰明、有多少自然語言功能、是否使用機器學習，或者是使用哪種程式設計語言來建立等因素，都沒有直接關係。如果 Bot 能夠解決使用者需要解決的問題，並且提供絕佳的使用者體驗，那麼使用者不太會關心這些因素。絕佳的 Bot 使用者體驗不需要使用者打太多字、說太多話、需要重複問題，或是解釋 Bot 應該自己要知道的事項。

TIP

不論您建立的是什麼類型的應用程式（Bot、網站或應用程式），使用者體驗都是第一要務。

設計 Bot 的程序就像是設計應用程式或網站，因此數十年來建置應用程式與網站的 UI 和提供 UX 的經驗，在設計

Bot 時仍然適用。

每當您不確定設計方法對於您的 Bot 是否正確時，請回過頭來問問自己下列問題：在應用程式或網站中會如何解決該問題？相同的答案經常都能套用到 Bot 設計。

後續步驟

既然您已熟悉 Bot 設計的一些基本準則，請在本節其餘部分深入了解設計使用者體驗和通用模式。

設計 Bot 的首次使用者互動

2019/2/28 • [Edit Online](#)

第一印象非常重要

使用者與 Bot 之間的首次互動，對於使用者體驗而言是非常重要的。設計 Bot 時，請記得 Bot 對使用者的第一則訊息不應該僅僅是一句「您好」而已。在建置應用程式時，您應該將第一個畫面設計成能夠提供重要的瀏覽提示。使用者應該要能直覺地了解像是功能表的所在位置及其運作方式、應該去哪裡尋求協助、隱私權原則的內容為何之類的項目。當您設計 Bot 時，使用者與 Bot 之間的首次互動應該要能提供上述類型的資訊。

語言與功能表

請考量下列兩種設計：

設計 1



Hello user, how can I help you?

設計 2



Hello! How can I help you?

Orders

Products

Help

一開始，讓 Bot 詢問像是 "How can I help you?" (我可以如何協助您？) 這樣的開放式問題 通常是不建議的。假設您的 Bot 能夠執行一百種不同的工作，使用者應該不太可能自行猜出大部分的工作。這是因為您的 Bot 並沒有告訴使用者它可以做什麼，所以他們怎麼可能會知道呢？

功能表能為此問題提供簡單的解決方案。首先，透過列出可用的選項，您的 Bot 便能將自己的功能傳達給使用者。再來，功能表可讓使用者無需自行輸入太多內容，而只需要按一下即可。最後，使用功能表將能縮小 Bot 可從使用者接收之輸入的範圍，並大幅簡化您的自然語言模型。

TIP

功能表對於設計能提供絕佳使用者體驗的 Bot 來說，是一個非常重要的工具，因此請不要因為它們「不夠聰明」而棄之不用。您依舊可以在搭配功能表設計 Bot 的情況下支援自由格式輸入。如果使用者回應初始功能表的方式是透過輸入，而非選取功能表項目，您的 Bot 可以嘗試剖析使用者的文字輸入。

或者，在 Bot 具有特定功能的情況下，您可以詢問更具針對性的問題來引導使用者。例如，如果您的 Bot 是負責接收三明治訂單，您的第一個互動便可以是「您好！我可以幫您點三明治。請問您想要哪一種麵包？我們有白麵包、全麥麵包或黑麵包。」如此一來，使用者便能透過交談中所提供的瀏覽提示了解回應的方式。

其他考量

除了提供直覺且能輕鬆瀏覽的首次互動之外，一個設計完善的 Bot 也應該讓使用者能輕鬆存取其隱私權原則和使用規定。

TIP

如果您的 Bot 會收集使用者的個人資料，請務必將此作法告知使用者，並描述該資料的後續使用方式。

後續步驟

您已熟悉設計使用者與 Bot 之間首次互動的幾個基本原則，請繼續深入了解[設計交談流程](#)。

設計和控制對話流程

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

在傳統的應用程式中，使用者介面 (UI) 是一系列的畫面。單一應用程式或網站可以視需要，使用一個或多個畫面與使用者交換資訊。大部分的應用程式是以主畫面啟動，使用者在其中開始進入並提供導覽，以導向其他畫面來使用不同的功能，例如開始新的訂單、瀏覽產品或尋求協助。

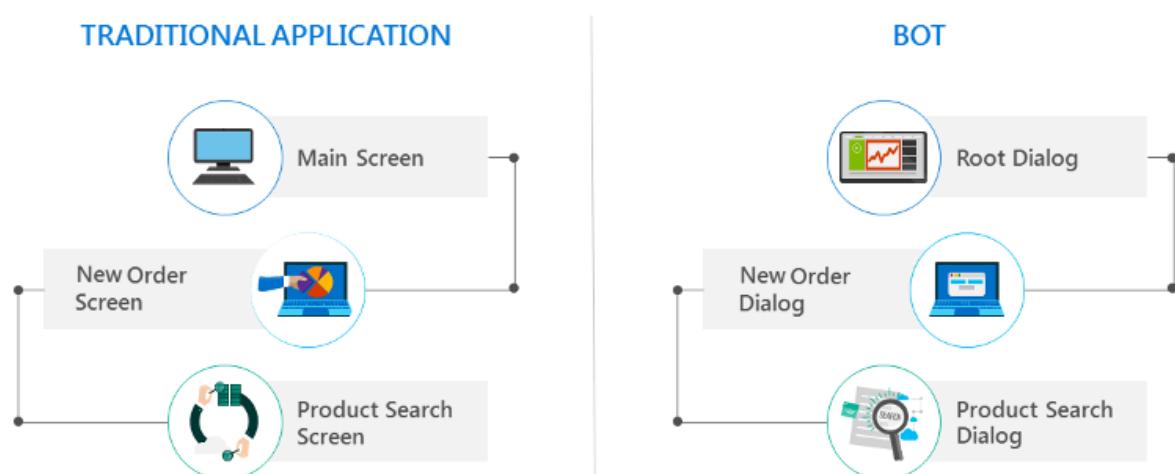
如同應用程式和網站，Bot 具有使用者介面，不過是由對話方塊組成，而不是畫面。對話方塊有助於保留您在對話內的位置，必要時提示使用者，以及執行輸入驗證。很適合用於管理多回合對話和簡單的「表單型」資訊收集，以完成預訂班機等活動。

對話方塊可讓 Bot 開發人員以邏輯方式區隔 Bot 功能的各個區域，並引導對話式流程。例如，您可能會設計一個包含可協助使用者瀏覽產品邏輯的對話方塊，以及一個包含可協助使用者建立新訂單的不同對話方塊。

對話方塊不一定具有圖形化介面。它們可能包含按鈕、文字和其他元素，或完全以語音為基礎。對話方塊也包含可執行工作的動作，例如叫用其他對話方塊，或是處理使用者輸入。

使用對話方塊來管理對話流程

下圖顯示傳統應用程式的畫面流程與 Bot 對話流程的比較。



在傳統的應用程式中，所有項目都是以主畫面開始。主畫面會叫用新訂單畫面。新訂單畫面會保有持控制權，直到它關閉或叫用其他畫面為止。如果新訂單畫面關閉，使用者就會返回主畫面。

在 bot 中，所有項目都是以根對話方塊開頭。根對話會叫用新訂單對話。此時，新訂單對話會接管對話 (conversation) 並保有控制權，直到它關閉或叫用其他對話 (dialog) 為止。如果新訂單對話 (dialog) 關閉，則會將對話 (conversation) 的控制權還給根對話 (dialog)。

如需使用對話和 Bot Framework SDK 來管理交談流程的詳細逐步解說，請參閱：

- [使用對話方塊來管理對話流程 \(.NET\)](#)

- 使用對話方塊來管理對話流程 (Node.js)

對話方塊堆疊

當一個對話方塊叫用另一個對話方塊時，Bot 建立器 會將新的對話方塊新增至對話方塊堆疊的頂端。堆疊頂端的對話方塊會控制對話。使用者所傳送的每個新訊息都必須由該對話方塊處理，直到關閉或重新導向至另一個對話方塊為止。當對話方塊關閉時，會從堆疊中移除，且堆疊中的前一個對話方塊會取得對話的控制權。

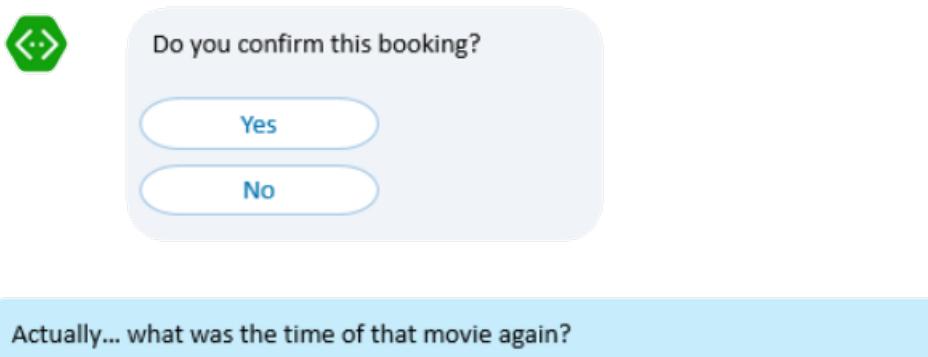
IMPORTANT

若要能夠有效地設計 Bot 的對話流程，則了解在對話方塊互相叫用、關閉等等時，Bot 建立器如何建構及解構對話方塊堆疊的概念至關重要。

對話方塊、堆疊和人類

您很自然地會想要假設使用者會建立對話方塊堆疊來巡覽對話方塊，且在某個時間點會巡覽回到他們的來源方向，以整齊而有條理的方式，依序將對話方塊取消堆疊。例如，使用者會從根對話方塊中開始，從該處叫用新的訂單對話方塊，並接著叫用產品搜尋對話方塊。然後使用者會結束 [產品搜尋] 對話方塊來選取產品並確認、結束 [新增訂單] 對話方塊來完成訂單，然後回到根對話方塊。

雖然使用者一律經過這類線性、邏輯路徑的話會很棒，但這卻很少發生。人類不會以「堆疊」方式通訊。他們通常會頻繁地改變心意。請思考下列範例：



雖然您的 Bot 可能會以邏輯方式建構對話方塊的堆疊，但使用者可能會決定要進行完全不同的事項，或是詢問可能與目前主題不相關的問題。在範例中，使用者會詢問問題，而不是提供對話方塊所預期的 [是]/[否] 回應。您的對話方塊應該如何回應？

- 堅持由使用者先回答問題。
- 忽略使用者先前已完成的所有項目、重設整個對話方塊堆疊，並嘗試回答使用者的問題來從頭開始。
- 嘗試回答使用者的問題，然後返回是/否問題，接著嘗試從該處繼續執行。

這個問題沒有任何「正確」答案，因為最好的解決方案將取決於您案例的細節，以及使用者會如何合理預期 Bot 回應。不過，隨著您的對話複雜度增加，對話方塊會變得更難管理。在複雜分支的情況下，建立您自己的控制邏輯流程，可能比追蹤您的使用者對話更為輕鬆。

後續步驟

透過讓使用者可以達成其目標（即使以非線性的方式）的方式來管理使用者的對話方塊瀏覽及設計對話流程，是 Bot 設計的基本挑戰。[下一篇文章](#)會檢閱一些常見的瀏覽設計不良錯誤，並討論避免這些陷阱的策略。

設計和控制對話流程

在傳統的應用程式中，使用者介面 (UI) 包含一系列的畫面，且單一應用程式或網站可以視需要使用一個或多個畫面與使用者交換資訊。大部分的應用程式是以主畫面啟動，使用者在其中開始進入，該畫面會提供導覽，以導向其他畫面來使用不同的功能，例如開始新的訂單、瀏覽產品或尋求協助。

如同應用程式和網站，Bot 具有使用者介面，不過是由訊息組成，而不是畫面。訊息可能包含按鈕、文字和其他元素，或完全以語音為基礎。

雖然傳統的應用程式或網站可以要求在畫面上一次顯示全部資訊，但 Bot 會使用多個訊息來收集相同數量的資訊。如此一來，向使用者收集資訊的程序就是主動的體驗；使用者可在其中與 Bot 主動對話。

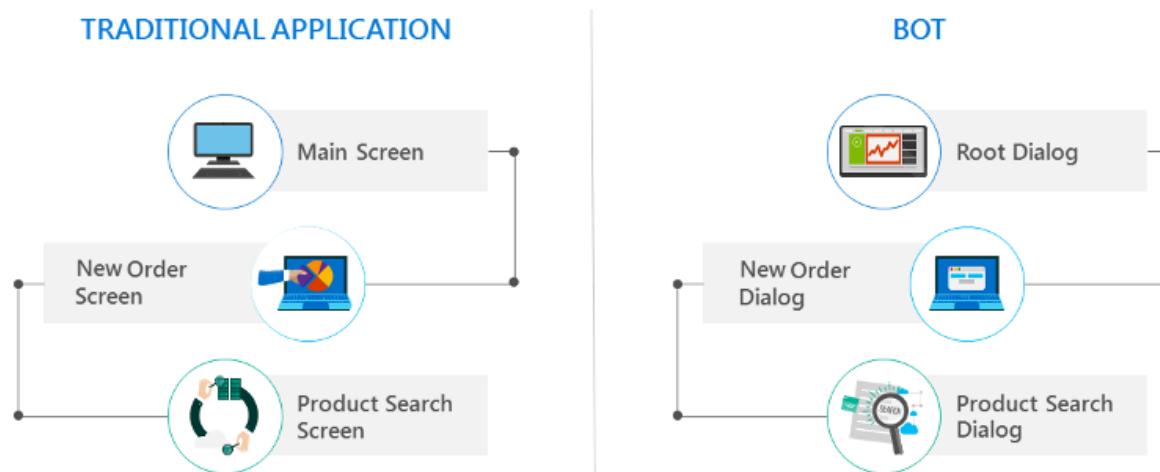
設計良好的 Bot 會有自然的對話流程。Bot 應該能夠順暢地處理核心對話，且能夠正常地處理中斷或切換對話的主題。

程序的對話流程

與 Bot 對話通常著重於 Bot 要嘗試達成的工作，也就是所謂的程序流程。Bot 會在其中向使用者詢問一系列的問題，以收集在處理工作之前所需的所有資訊。

在程序對話流程中，您要定義問題的順序，且 Bot 會以您定義的順序來詢問問題。您可以將問題組織成邏輯「模組」以保持程式碼集中，同時聚焦於引導對話。例如，您可能會設計一個包含可協助使用者瀏覽產品邏輯的模組，以及一個包含可協助使用者建立新訂單邏輯的不同模組。

您可用任何偏好的方式來建構這些模組，從自由格式到循序方式皆可。Bot Framework SDK 提供多個程式庫，可讓您建構 Bot 所需的任何交談式流程。例如，`prompts` 程式庫可讓您要求使用者輸入，`waterfall` 程式庫可讓您定義問題/答案組的序列，`dialog control` 程式庫可讓您將對話流程邏輯模組化等等。這些程式庫全都會透過 `dialogs` 物件繫結在一起。讓我們進一步看看模組會如何實作為 `dialogs` 來設計和管理對話流程，並查看該流程與傳統應用程式流程有何相似之處。



在傳統的應用程式中，所有項目都是以主畫面開始。主畫面會叫用新訂單畫面。新訂單畫面會保持控制權，直到它關閉或叫用其他畫面（例如產品搜尋畫面）為止。如果新訂單畫面關閉，使用者就會回到主畫面。

在 bot 中，所有項目都是以根對話方塊開頭。根對話方塊會叫用新訂單對話方塊。此時，新訂單對話方塊會接管對話，並保持控制權，直到它關閉或叫用其他對話方塊（例如產品搜尋對話方塊）為止。如果新訂單對話 (`dialog`) 關閉，則會將對話 (conversation) 的控制權還給根對話 (`dialog`)。

如需如何實作交談流程的範例，請參閱如何[建立自己的提示以蒐集使用者輸入](#)，並使用對話來[實作循序交談流程](#)。

處理中斷

您很自然地會想要假設使用者會以整齊而有條理的方式，依序執行程序性工作。例如，在使用 `dialogs` 的程序性

對話流程中，使用者會從根對話方塊中開始，從該處叫用新訂單，並接著叫用產品搜尋對話方塊。然後使用者會結束 [產品搜尋] 對話方塊來選取產品並確認、結束 [新增訂單] 對話方塊來完成訂單，然後回到根對話方塊。

雖然使用者一律經過這類線性、邏輯路徑的話會很棒，但這卻很少發生。人類不會以循序 dialogs 方式通訊。他們通常會頻繁地改變心意。請思考下列範例：



Actually... what was the time of that movie again?

雖然您的 Bot 可能會以程序為中心，但使用者可能會決定要進行完全不同的事項，或是詢問可能與目前主題不相關的問題。在上述範例中，使用者會詢問問題，而不是提供 Bot 所預期的 [是]/[否] 回應。您的 Bot 應該如何回應？

- 堅持由使用者先回答問題。
- 忽略使用者先前已完成的所有項目、重設整個對話方塊堆疊，並嘗試回答使用者的問題來從頭開始。
- 嘗試回答使用者的問題，然後返回 [是]/[否] 問題，接著嘗試從該處繼續執行。

這個問題沒有任何「正確」答案，因為最好的解決方案將取決於您案例的細節，以及使用者會如何合理預期 Bot 回應。如需詳細資訊，請參閱[處理使用者中斷](#)。

後續步驟

透過讓使用者可以達成其目標（即使以非線性的方式）的方式來管理使用者的對話方塊瀏覽及設計對話流程，是 Bot 設計的基本挑戰。[下一篇文章](#)會檢閱一些常見的瀏覽設計不良錯誤，並討論避免這些陷阱的策略。

設計 Bot 導覽

2019/2/28 • [Edit Online](#)

使用者可以使用階層連結瀏覽網站，使用功能表瀏覽應用程式，以及使用下一頁和上一頁等按鈕瀏覽網頁。不過，上述這些固有的導覽技術都無法完全解決 Bot 內的導覽需求。如先前所述，使用者通常會以非線性的方式與 Bot 互動，因此要設計出可提供一致的優良使用者體驗之 Bot 導覽，變得非常困難。

需要考量以下難題：

- 如何確保使用者不會在與 Bot 的交談中感到混淆？
- 使用者能否在與 Bot 的交談中進行「回溯」瀏覽？
- 使用者如何在與 Bot 交談期間瀏覽至「主功能表」？
- 使用者如何在與 Bot 交談期間「取消」作業？

Bot 導覽設計的具體細節絕大部分取決於 Bot 可支援的特色與功能。不論要開發的 Bot 類型為何，您都要避免常見的錯誤，以免設計出不良的交談介面。本文透過下列五種性格來說明這些錯誤：「固執的 Bot」、「毫無頭緒的 Bot」、「神祕的 Bot」、「講廢話的 Bot」和「念念不忘的 Bot」。

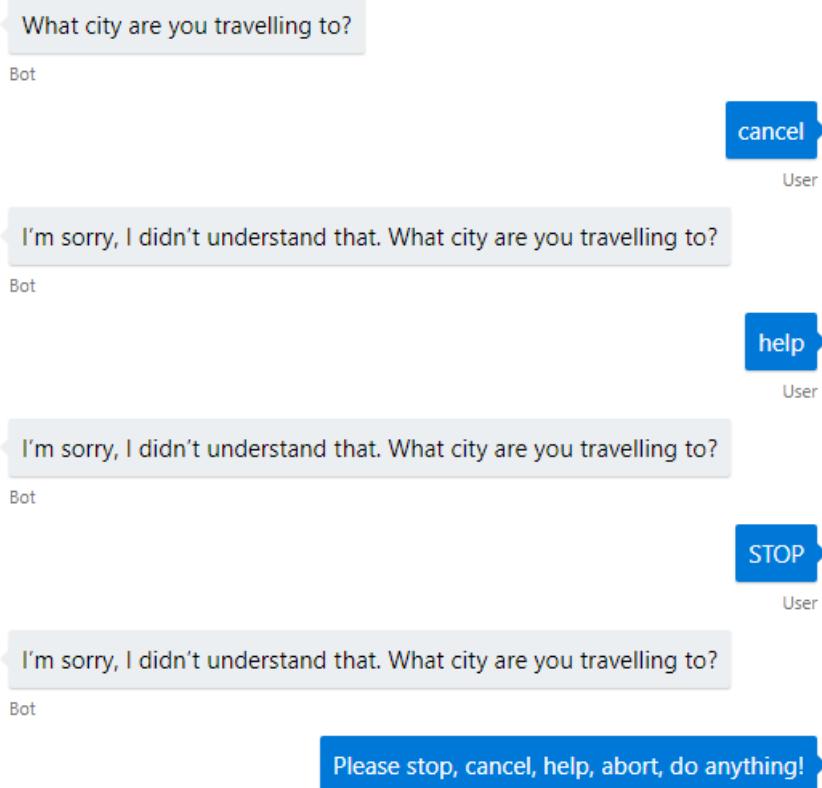
TIP

只要正確地處理使用者中斷，通常就能改善 Bot 的這些性格。

「固執的 Bot」

固執的 Bot 會堅持繼續目前的交談方向，即使使用者嘗試轉換話題，仍依然故我。

請考慮下列狀況：



使用者會經常改變心意、決定取消或全部從頭開始。

TIP

切記：設計 Bot 時，請將使用者預設為可能會隨時改變交談方向納入考量。

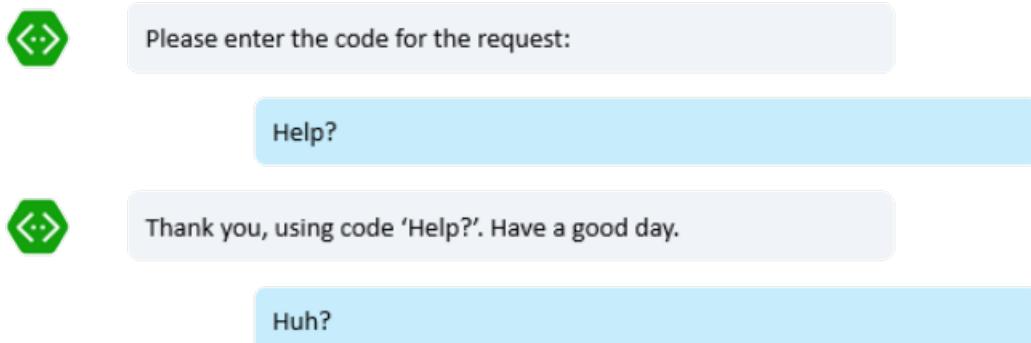
請勿：設計時，切勿使 Bot 忽略使用者輸入，並永無止境地持續重複相同問題。

避免這項錯誤的方法有很多種，但要避免 Bot 無止盡地詢問相同問題，最簡單的方法可能是指定每個問題重試次數的上限。若以這種方式來設計，Bot 雖不會「聰明」地理解使用者輸入內容並給予恰當回應，但至少能夠避免永無止境地反覆詢問相同問題。

「毫無頭緒的 Bot」

毫無頭緒的 Bot 會在無法理解使用者想要存取特定功能時，以無意義的方式回應。使用者可能會嘗試輸入常見關鍵字命令，例如「幫助」或「取消」，並合理地期望 Bot 能妥善回應。

請考慮下列狀況：



雖然您可能很想將 Bot 中的每個對話方塊設計成能夠聽取特定關鍵字，並給予恰當回應，但我們並不建議這麼做。

TIP

切記：請執行[中介軟體](#)，藉此檢查使用者輸入中是否有您指定的關鍵字（例如「幫助」、「取消」、「從新開始」等），並給予適當回應。

請勿：將每個對話方塊設計成按照關鍵字清單檢查使用者輸入內容。

只要定義[中介軟體](#)的邏輯，中介軟體就能存取每次與使用者進行的交流。運用這個方法，個別對話方塊和提示就能在必要時安全地忽略關鍵字。

「神祕的 Bot」

神祕的 Bot 無論如何都無法立即認可使用者輸入。

請考慮下列狀況：



How can I help you today?

Which movies are showing this week?

Hello?

Which movies are showing this week?

Hey bot, are you there?

某些情況下，這類狀況可能表示 Bot 發生中斷。不過，也有可能 Bot 正忙於處理使用者輸入，而且回應的編譯作業尚未完成。

TIP

切記：請將 Bot 設計成可立即認可使用者輸入內容，即便 Bot 可能需要花一些時間來編譯回應亦然。

請勿：將 Bot 設計成要先完成回應的編譯作業，才認可使用者輸入內容。

立即認可使用者輸入內容，您才能消除任何可能對 Bot 狀態產生的混淆。如果您需要花很長的時間來編譯，不妨傳送「輸入中」訊息，表示 Bot 正在運作，接著再透過[主動訊息](#)傳送後續內容

「講廢話的 Bot」

講廢話的 Bot 會提供未經要求且顯而易見的資訊，因此對使用者不具實用價值。

請考慮下列狀況：



You just spent \$10 on your credit card!

Yup, I know...



It looks like you are driving to work!

Really...



It looks like you arrived at work! Have a nice day!

Jeez, get a life, bot...

TIP

切記：請妥善設計 Bot 以便為使用者提供實用資訊。

請勿：將 Bot 設計成向使用者提供未經要求且可能不實用的資訊。

將 Bot 設計成可提供實用資訊，使用者與 Bot 的互動機率就會提高。

「念念不忘的 Bot」

念念不忘的 Bot 會在不適當的時機，將過往交談中的資訊整合進目前的交談中。

請考慮下列狀況：



I want to travel to Italy

Please confirm: Are you ok with me charging \$200 for your trip to Las Vegas?



Which Trip to Las Vegas???

Your trip to Las Vegas you wanted me to book for June the 5th

OMG that was 3 months ago, bot...

TIP

切記：請將 Bot 設計成只要使用者並未重新提起之前的主題，就會維持目前的交談主題。

請勿：將 Bot 設計成會將過往交談中的資訊插入目前並不相關的交談中。

只要能維持目前的交談主題，您就能減少造成的混淆和挫折感的可能性，同時提高使用者繼續和 Bot 互動的機率。

後續步驟

如果能將 Bot 設計成可避免以上設計不良之交談介面常見的錯誤，您就能踏出重要的一步，從而確保絕佳的使用者體驗。

接下來，請論了解通常 Bot 與使用者與交換資訊所仰賴的 [UX 元素](#)。

設計使用者體驗

2019/2/28 • [Edit Online](#)

您可以建立各種不同功能的 Bot，例如文字、按鈕、影像、展示在轉盤或清單格式的多媒體卡片或更多功能。不過，Facebook、Slack、Skype 等每個通道 最後會控制其訊息用戶端呈現功能。即使有多個通道支援某項功能時，每個通道可以稍有不同的方式呈現該功能。在一訊息中包含通道原本不支援的功能之情況下，該通道會嘗試關閉呈現訊息的內容變為文字或靜態影像，這可能會嚴重影響用戶端上的訊息的外觀。在某些情況下，通道可能完全不支援特定的功能。舉例來說，GroupMe 用戶端無法顯示正在輸入的指標。

豐富使用者控制項

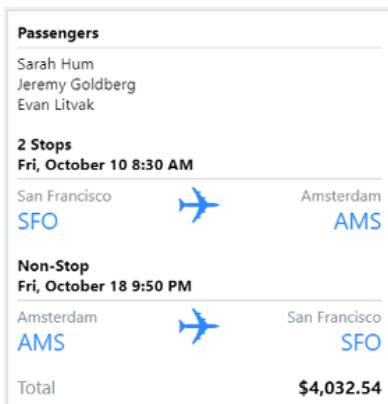
豐富使用者控制項是 Bot 向使用者顯示的一般 UI 控制項，例如按鈕、影像、浮動切換和功能表，而使用者會和這些控制項互動以傳達選擇和意圖。Bot 可以使用一組 UI 控制項來模擬應用程式，或甚至可以內嵌在應用程式中執行。當 Bot 是內嵌在應用程式或網站中時，它可以利用裝載它的應用程式的功能，來代表幾乎任何 UI 控制項。

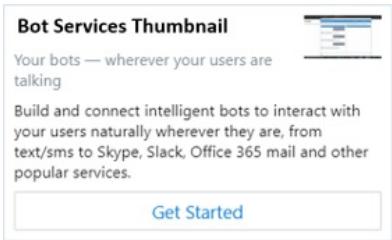
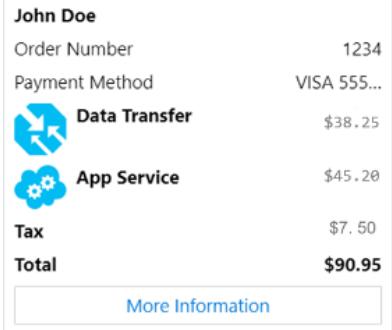
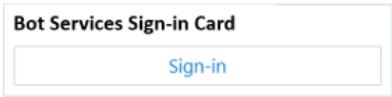
數十年來，應用程式和網頁開發人員都依靠 UI 控制項來讓使用者和其應用程式互動，而這些相同的 UI 控制項在 Bot 中也非常有效。例如，按鈕很適合用來向使用者顯示簡單的選擇。讓使用者以按一下標示為 [旅館] 的按鈕來傳達「旅館」，比強迫使用者輸入「旅館」更容易且更快。對於在行動裝置上更是如此，因為在這些裝置上，按一下比輸入更容易。

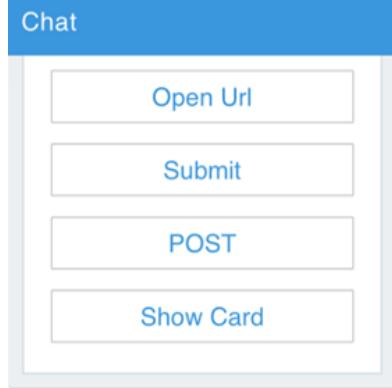
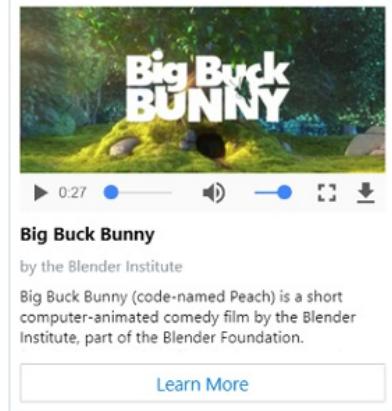
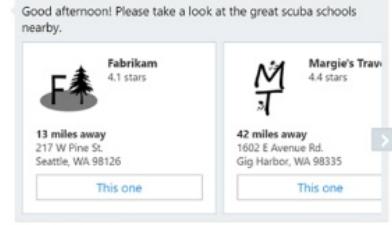
卡片

卡片可讓您向使用者顯示各種視覺、音訊和/或可選取的訊息，並輔助對話流程。如果使用者需要從一組固定的項目中選取，您可以顯示卡片的浮動切換，每個卡片都包含影像、文字描述和單一選取按鈕。如果使用者對於單一項目有一組選擇，您可以顯示較小的單一影像及一組包含不同選項的按鈕來供選擇。他們是否在主題上要求詳細資訊？卡片可以使用音訊或視訊輸出，或是詳細列出其購物內容的收據來提供深入的資訊。卡片有非常廣泛的用法可以用於協助引導使用者和 Bot 之間的交談。您使用的卡片類型將會由您應用程式的需求來決定。讓我們看看卡片、其動作及一些建議的用法。

Microsoft Bot 服務卡片是可以程式設計的物件，其中包含可在各種通道被識別的標準豐富使用者控制項集合。下表說明可用卡片的清單，以及使用每個卡片類型的最佳做法建議。

卡片類型	範例	說明
AdaptiveCard		轉譯為 JSON 物件的開放式卡片交換格式。通常用於卡片的跨通道部署。卡片會隨每個主通道調適其外觀與風格。

卡片類型	範例	說明
AnimationCard	 <p>Azure Bot Service Animation Card</p>	可以播放動畫 GIF 或短片的卡片。
AudioCard	 <p>I am your father Star Wars: Episode V - The Empire Strikes Back The Empire Strikes Back (also known as Star Wars: Episode V – The Empire Strikes Back) is a 1980 American epic space opera film directed by Irvin Kershner. Leigh Brackett and Lawrence Kasdan wrote the screenplay, with George Lucas writing the film's story and serving as executive producer.</p> <p>Read more</p>	可以播放音訊檔案的卡片。
HeroCard	 <p>Music / Tame Impala (Recommended) 7:40 pm Sep 4th AT Memorial Stadium Purchase Tickets</p>	包含單一大型影像、一或多個按鈕與文字的卡片。通常用來在視覺上醒目提示潛在的使用者選取。
ThumbnailCard	 <p>Bot Services Thumbnail Your bots — wherever your users are talking Build and connect intelligent bots to interact with your users naturally wherever they are, from text/sms to Skype, Slack, Office 365 mail and other popular services. Get Started</p>	包含單一縮圖影像、一或多個按鈕與文字的卡片。通常用來在視覺上醒目提示潛在的使用者選取的按鈕。
ReceiptCard	 <p>John Doe Order Number 1234 Payment Method VISA 555...  Data Transfer \$38.25  App Service \$45.20 Tax \$7.50 Total \$90.95 More Information</p>	讓 Bot 向使用者提供收據的卡片。它通常包含收據上的項目清單 (稅金和總金額資訊) 與其他文字。
SignInCard	 <p>Bot Services Sign-in Card Sign-in</p>	可讓 Bot 要求使用者登入的卡片。它通常包含文字，以及一或多個按鈕，使用者可以按一下按鈕來起始登入程序。

卡片類型	範例	說明
SuggestedAction		向使用者顯示一組代表使用者選擇的 CardActions。選取其中任一個建議的動作之後，此卡片就會消失。
VideoCard		可以播放影片的卡片。通常用來開啟 URL 和串流可用的影片。
CardCarousel		一組可水平捲動的卡片，可讓使用者輕鬆檢視一系列可能的使用者選擇。

卡片讓您只要設計一次 Bot，然後就能讓它在各種通道上運作。不過，並非所有可用的通道都完整支援所有卡片類型。

將卡片新增到 Bot 的詳細指示可以在以下各節中找到：[新增豐富卡片媒體附件](#)、[新增訊息的建議動作](#)。這裡也可以找到卡片的範例程式碼：[C#/JS 調適型卡片](#)：[C#/JS](#)，附件：[C#/JS](#)，以及建議的動作：[C#/JS](#)。

在設計您的 Bot 時，請不要因為「不夠智慧」而自動捨棄一般 UI 元素。如[先前所述](#)，您的 Bot 應該設計成盡可能以最佳、最快且最輕鬆的方式來解決使用者的問題。請避免因受到引誘而一開始就整合自然語言理解，它通常是非必要的，而且會造成不必要的複雜性。

TIP

一開始請使用最少的 UI 控制項來讓 Bot 解決使用者的問題，如果那些控制項後來不足以解決問題，此時再新增其他元素。

文字和自然語言理解

Bot 可以接受使用者的文字輸入，然後嘗試使用規則運算式比對或自然語言理解 API (例如 [LUIS](#)) 來剖析該輸入。根據使用者提供的輸入類型，自然語言理解可能是好的也可能是不好的解決方案。

在某些情況下，使用者可能會回答非常特定的問題。例如，如果 Bot 問：「您叫什麼名字？」，使用者可能會回答只

指定名字的文字 "John", 或是回答句子「我的名字是 John」。

問特定問題可以縮小 Bot 可能收到之回覆的潛在範圍，這樣可以減少剖析和理解回應所需之邏輯的複雜性。例如，請考慮以下的廣泛、開放式問題：「您感覺如何？」。理解這種問題的潛在回答之眾多可能的排列是非常複雜的工作。

相較之下，特定問題如「您是否覺得疼痛？是/否」和「您覺得哪裡疼痛？胸/頭/手臂/腿」可能可以得到更特定的回答，且 Bot 可以剖析並理解，而不需要實作自然語言理解。

TIP

盡可能地問不需要使用自然語言理解功能來剖析回應的特定問題。這樣將會簡化您的 Bot，並增加 Bot 成功理解使用者的機會。

在其他情況下，使用者可能會輸入特定命令。例如，DevOps Bot 可讓開發人員管理虛擬機器，它可設計成接受 "/STOP VM XYZ" 或 "/START VM XYZ" 等特定命令。設計這種接收特定命令的 Bot 可以提供好的使用者體驗，因為語法容易學習，且每個命令的預期結果都很清楚。此外，這種 Bot 不會需要自然語言理解功能，因為使用規則運算式很容易就能剖析使用者的輸入。

TIP

設計需要使用者提供特定命令的 Bot，通常可以提供好的使用者體驗，同時也消除對自然語言理解功能的需求。

對於「知識庫」Bot 或「問與答」Bot 的案例，使用者可能會問一般性問題。例如，請想像一個可以根據好幾萬份文件來回答問題的 Bot。QnA Maker (英文) 和 Azure 搜尋服務都是專門為此類型案例而設計的。如需詳細資訊，請參閱[設計知識 Bot](#)。

TIP

如果您要設計的 Bot 將根據資料庫、網頁或文件的結構化或非結構化資料來回答問題，請考慮專門為解決這種案例而設計的技術，而不是嘗試使用自然語言理解來解決問題。

在其他情況下，使用者可能以自然語言輸入簡單的要求。例如，使用者可能輸入「我想吃辣肉腸披薩」或「離我家 3 英哩內有任何營業中的素食餐廳嗎？」。[LUIS.ai](#) 等自然語言理解 API 非常適合這種案例。

使用該 API，您的 Bot 就能從使用者的文字擷取關鍵要素以識別使用者的意圖。在您的 Bot 中實作自然語言理解時，請針對使用者在其輸入中可能提供的詳細資料層級設定實際預期。

"I want to find a house for sale that has 3 or 4 bedrooms, priced between \$300 and \$350 with a large garden, about 2000 square feet, preferably green, within 10 miles from my work which is in the city center, with a large garage and a backyard with a pool"

-Said nobody, ever

"I want to find a house"

-Said everybody else

TIP

建置自然語言模型時，請勿假設使用者將會在初始查詢中就提供所有必要資訊。請將您的 Bot 設計成明確地要求它所需的資訊，且如有必要，可以透過詢問一系列的問題來引導使用者提供資訊。

語音

Bot 可以使用語音輸入和/或輸出與使用者通訊。如果 Bot 是設計成支援沒有鍵盤或監視器的裝置，則語音就是唯一與使用者通訊的方法。

選擇豐富使用者控制項、文字和自然語言或語音

就像人使用手勢、聲音和符號的組合來互相溝通，Bot 可以使用豐富使用者控制項、文字（有時包含自然語言）與語音的組合來和使用者通訊。這些通訊方法可以搭配在一起使用，您不需要選擇一個而捨棄其他的。

例如，請想像協助使用者使用食譜的「烹飪 Bot」，它可能透過撥放影片來提供指示，或顯示一系列的圖片來解釋要處理的事項。某些使用者可能偏好在食譜上翻頁，或在按照食譜做菜時使用語音問 Bot 問題。其他人可能偏好在裝置的螢幕上觸控，而不是透過語音和 Bot 互動。在設計您的 Bot 時，請根據要支援的特定使用案例來整合使用者可能會想和您的 Bot 互動的 UX 元素。

建立工作自動化 Bot

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

工作自動化 Bot 可讓使用者完成特定工作或一組工作，而不需要任何人類的協助。這種類型的 Bot 通常十分類似一般應用程式或網站，主要是透過豐富的使用者控制項和文字，與使用者通訊。可能會有了解自然語言的功能，可擴充與使用者的對話。

範例使用案例：密碼重設

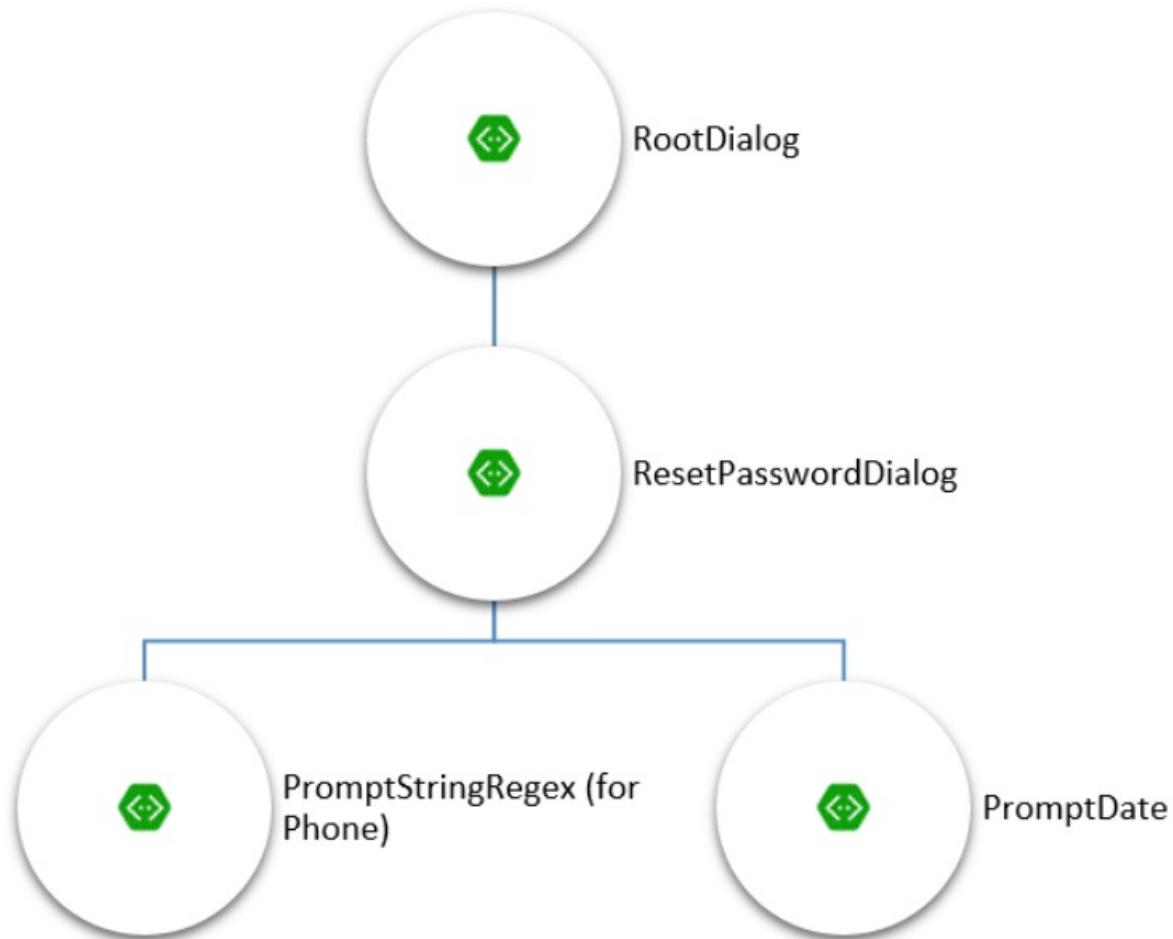
為了進一步了解工作 Bot 的本質，請思考使用案例範例：密碼重設。

每天都會有員工打好幾通電話來 Contoso 公司的技術支援中心，要求重設密碼。Contoso 想要將重設員工密碼這個簡單、可重複執行的工作自動化，以便技術支援中心代表可以將時間花費在處理更複雜的問題。

Contoso 的資深開發人員 John 決定建立 Bot，以將密碼重設工作自動化。他開始撰寫 Bot 的設計規格，就像在建立新的應用程式或網站時一樣。

巡覽模式

此規格會定義巡覽模式：



使用者會從 `RootDialog` 開始。當他們要求密碼重設時，系統會將他們導向至 `ResetPasswordDialog`。使用 `ResetPasswordDialog` 時，Bot 將會提示使用者提供兩項資訊：電話號碼和出生日期。

IMPORTANT

本文中所述的 Bot 僅供範例用途。在真實世界案例中，密碼重設 Bot 可能會實作更健全的身分識別驗證流程。

對話方塊

接下來，規格會描述每個對話方塊的外觀和功能。

根對話方塊

根對話方塊會提供兩個選項給使用者：

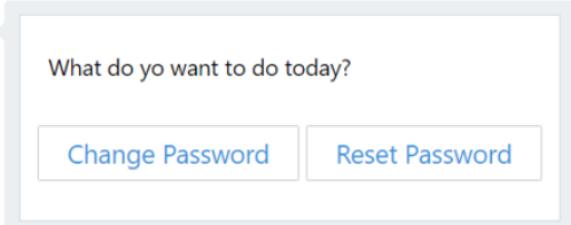
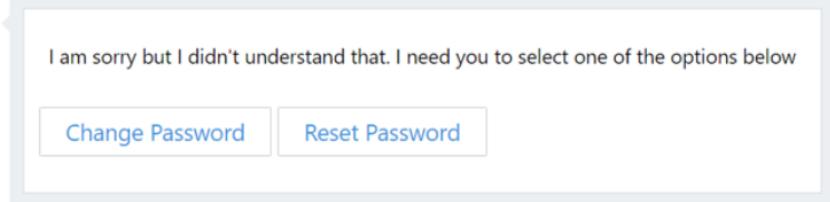
1. 變更密碼的適用情況，是使用者知道他們目前的密碼，並只想要加以變更。
2. 重設密碼的適用情況，是使用者已忘記或錯置其密碼，且必須產生一個新的密碼。

NOTE

為了簡單起見，本文只說明重設密碼流程。

此規格會描述根對話方塊，如下列螢幕擷取畫面所示。

4.1. RootDialog

Event	Description	Notes						
On Start:	Dialog initiates with the following message: 	On Start is what initially happens when the dialog is invoked						
User Input:	User will be allowed to click at one of the buttons above (or type one of those options on channels that don't support buttons): <table border="1" data-bbox="314 698 885 844"><thead><tr><th>Action</th><th>Effect</th></tr></thead><tbody><tr><td>"Change Password"</td><td>Flow not implemented. Send a message to the user</td></tr><tr><td>"Reset Password"</td><td>Call to ResetPassword dialog</td></tr></tbody></table>	Action	Effect	"Change Password"	Flow not implemented. Send a message to the user	"Reset Password"	Call to ResetPassword dialog	
Action	Effect							
"Change Password"	Flow not implemented. Send a message to the user							
"Reset Password"	Call to ResetPassword dialog							
Invalid user input:	In case of an invalid user input: 	Assuming the user types something else and no scorables picks that up, we will reject the input and retry the question.						
Exit Criteria:	This is the root dialog and therefore it never exits.	Root dialogs shouldn't exit.						
		They are the very entry point where the whole experience starts.						

ResetPassword 對話方塊

當使用者從根對話方塊中選擇 [重設密碼] 時，會叫用 `ResetPassword` 對話方塊。接著，`ResetPassword` 對話方塊會叫用另外兩個對話方塊。首先，它會叫用 `PromptStringRegex` 對話方塊來收集使用者的電話號碼。然後它會叫用 `PromptDate` 對話方塊來收集使用者的出生日期。

NOTE

在此範例中，John 選擇實作的邏輯，會使用兩個不同的對話方塊來收集使用者的電話號碼與出生日期。此方法不僅可簡化每個對話方塊所需的程式碼，也可增加未來在其他案例中使用這些對話方塊的機率。

此規格描述 `ResetPassword` 對話方塊。

4.2. ResetPassword dialog

Event	Description	Notes
On Start:	<p>Dialog initiates calling the PromptStringRegex dialog to ask the user for a phone number.</p> <p>Once that dialog completes, if the phone number was provided, it will be shown:</p> <p>The phone you provided is: (425) 555-4235</p> <p>And then the PromptDate dialog will be called to ask the user for the date of birth.</p>	
User Input:	No user input is provided in this dialog	
Invalid user input:	No user input is provided in this dialog	
Exit Criteria:	<p>If the phone number and the date of birth are correctly provided, the dialog will provide a new password and ends with a “true” value to indicate success.</p> <p>Thanks! Your new password is b3667b6b34ec4101bfe02e269fa8485c</p> <p>If the phone number or date of birth are not correctly provided or any of those prompts is cancelled, the dialog will end with a “false” value to indicate that it failed. RootDialog will detect that and will show:</p> <p>You identity was not verified and your password cannot be reset</p>	

PromptStringRegex 對話方塊

PromptStringRegex 對話方塊會提示使用者輸入其電話號碼，並驗證使用者提供的電話號碼符合預期的格式。它也會負責使用者重複提供無效輸入的案例。此規格描述 PromptStringRegex 對話方塊。

4.3. PromptStringRegex dialog

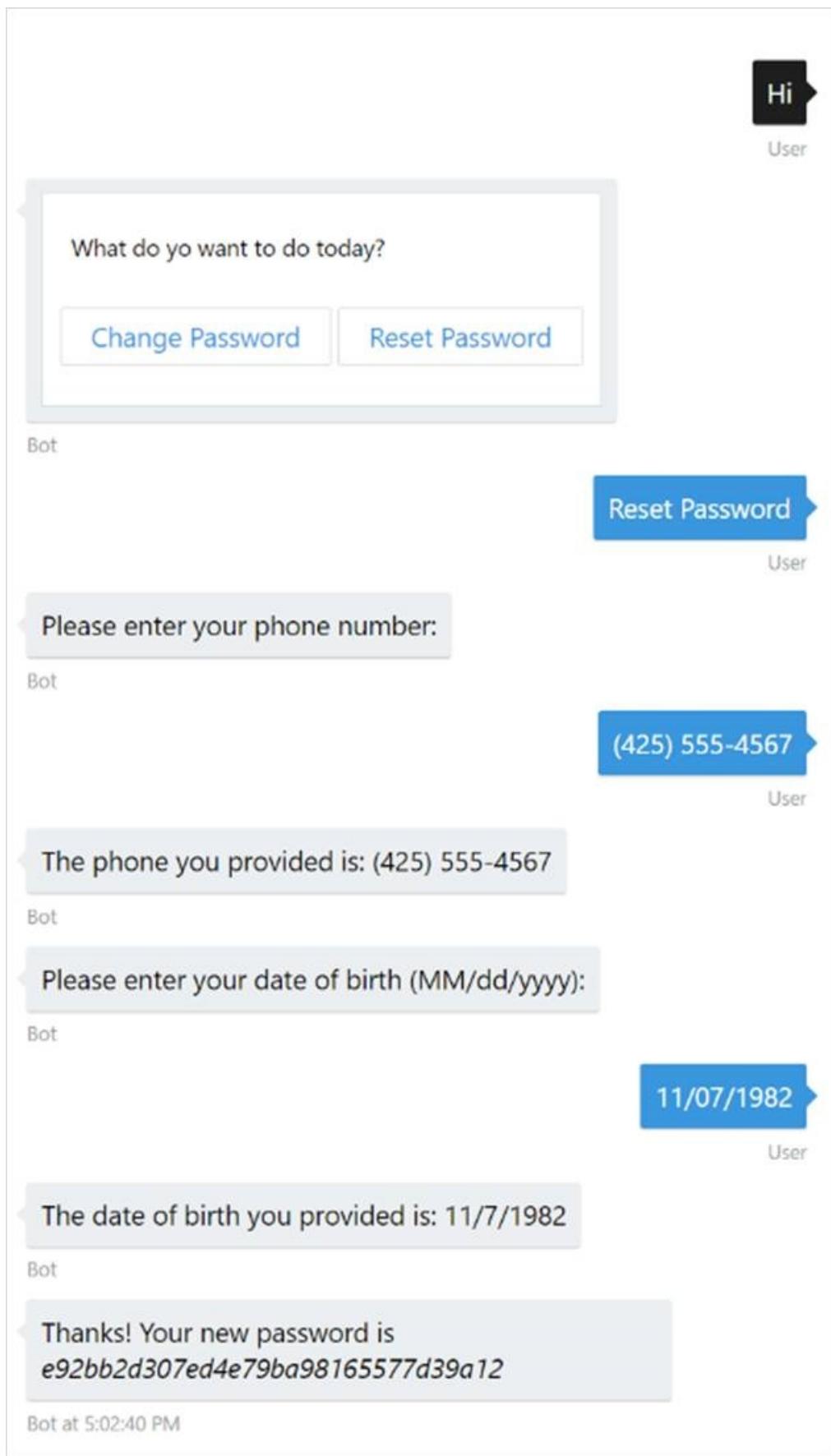
Event	Description	Notes		
On Start:	Dialog initiates with the following question: Please enter your phone number:	Since the options are too many, we will use free text input here. A custom PromptString dialog with Regex validation is being used		
User Input:	User will be allowed to type a phone number	A regular expression is being used to validate the input		
	<table border="1"> <thead> <tr> <th>Action</th><th>Effect</th></tr> </thead> <tbody> <tr> <td>Any input</td><td>Run a phone number regex validation</td></tr> </tbody> </table>		Action	Effect
Action	Effect			
Any input	Run a phone number regex validation			
Invalid user input:	In case on an invalid phone number: The value entered is not phone number. Please try again using the following format (xyz) xyz-wxyz:			
Exit Criteria:	To be defined in the alternative flows below			

Alternative Flow 1

Condition	An invalid phone was provided many times	
Effect	PromptStringRegex dialog ends with the following message: You have tried to enter your phone number many times. Please try again later.	
Exit Criteria:	PromptStringRegex throws a TooManyAttemptsException	No phone number is provided

原型

最後，此規格會提供一個範例，說明使用者與 Bot 進行通訊，以成功完成密碼重設工作。



Bot、應用程式還是網站？

您可能會好奇，如果工作自動化 Bot 十分類似應用程式或網站，為什麼不改為建置應用程式或網站就好？根據您的特定案例，建置應用程式或網站而不建置 Bot 可能是完全合理的選擇。您可能甚至會使用 [Bot Framework Direct Line API](#) 或 [網路聊天控制項](#)，選擇將您的 Bot 內嵌到應用程式中。在應用程式的內容中實作 Bot 有兩個優點：在同一個位置提供豐富的應用程式與對話體驗。

不過，在許多情況下，建置應用程式或網站可能遠比建置 Bot 還更複雜且更昂貴。應用程式或網站通常必須支援多個用戶端和平台，因此封裝和部署可能會是既繁瑣又費時的流程，且必須下載及安裝應用程式的使用者經驗並不一定很理想。基於這些理由，Bot 通常可提供較簡單的方式，來解決眼前的問題。

此外，Bot 能夠自由地輕鬆延伸及擴充。例如，開發人員可以選擇將自然語言和語音功能新增至密碼重設 Bot，使其可透過語音通話進行存取，或可新增文字簡訊的支援。該公司可能會在整個建置中設定 kiosk，並在這個體驗中內嵌密碼重設 Bot。

其他資源

- [對話方塊](#)
- [使用對話方塊來管理對話流程 \(.NET\)](#)
- [使用對話方塊來管理對話流程 \(Node.js\)](#)

設計知識型 Bot

2019/2/28 • [Edit Online](#)

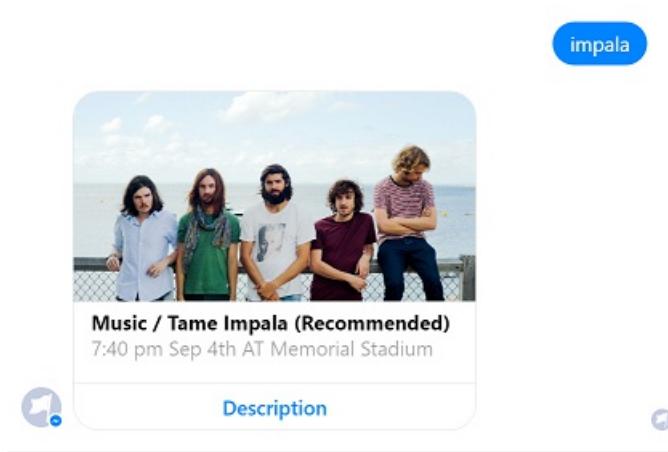
您可設計知識型 Bot 來提供有關任何主題的資訊。例如，您可讓一個知識型 Bot 回答「這場研討會有哪些 Bot 活動？」、「下場雷鬼表演什麼時候？」或「誰是 Tame Impala？」等事件的問題。再設計一個知識型 Bot 負責回答「我如何更新作業系統？」或「我要去哪裡重設密碼？」這類 IT 相關問題。而另一個 Bot 則可回答「誰是 John Doe？」或「John Doe 的電子郵件地址是什麼？」等有關連絡資訊的問題。

無論知識型 Bot 是專為哪些使用案例而設計，其基本目標皆相同：利用大量資料（例如：SQL 資料庫中的關聯式資料、非關聯式存放區中的 JSON 資料或文件存放區中的 PDF），尋找並傳回使用者要求的資訊。

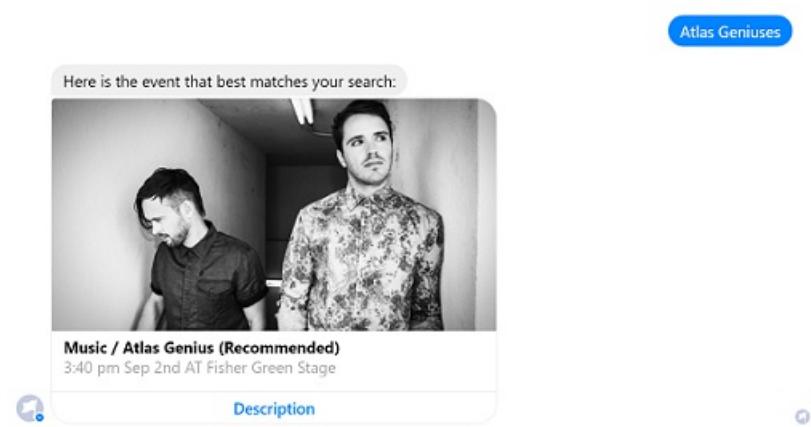
Search

搜尋功能在 Bot 中是相當重要的工具。

首先，「模糊搜尋」不需要使用者提供精確的輸入內容，也能讓 Bot 傳回可能與使用者問題相關的資訊。例如，如果使用者詢問音樂知識型 Bot 有關「impala」的資訊（而非精準輸入「Tame Impala」），則 Bot 還是可在回應中提供最有可能與該輸入相關的資訊。



搜尋分數表示特定搜尋結果的可信度，Bot 便可依此排序結果，甚至根據可信度客製化其通訊內容。例如，如果可信度很高，則 Bot 即可用「以下是最符合您搜尋的活動：」回應使用者。



如果可信度很低，則 Bot 可回答「嗯...您要找的是這些活動嗎？」

super smart atlas

Hmm... were you looking for any of these events?

Music / Atlas Genius (Recommended)
3:40 pm Sep 2nd AT Fisher Green Stage

Description

Music / Super Square
4:30 pm Sep 3rd AT KeyArena

Desc

使用搜尋功能引導交談

如果您建置 Bot 只是為了使用基本的搜尋引擎功能，則可能根本不需要 Bot。交談式介面能提供使用者哪些無法從網頁瀏覽器上一般搜尋引擎取得的內容？

當知識型 Bot 是設計用於引導交談時，通常最能發揮效益。交談就是指使用者 和 Bot 來回交換訊息，因此可讓 Bot 有機會進一步詢問以釐清問題、提供選項並驗證結果；而這些都是基本搜尋功能無法辦到的。例如，以下 Bot 可透過交談引導使用者利用 Facet 和篩選條件過濾資料集，直到其找出使用者所需的資訊。

Events

What are you interested in?

Music Comedy Film Laser Dome >

Music

What Music are you interested in?

Any Rock/Pop Hiphop/Rap Soul/R&B >

Rock/Pop

What day would you like to see Rock/Pop?

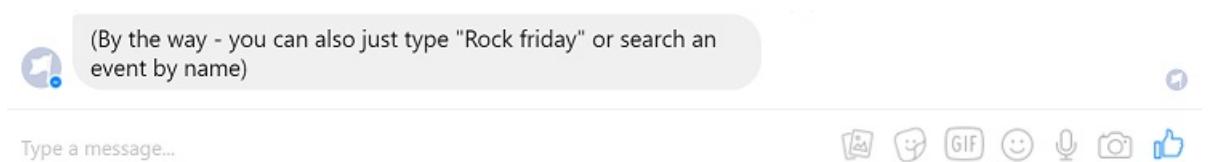
Friday Saturday Sunday Any

Here were the Rock/Pop shows Saturday:

Music / Pony Time (Recommended)
3:00 pm Sep 3rd AT KEXP

Music / Desi Valentine
3:30 pm Sep 3rd AT Starbucks Stage

藉由處理使用者每個步驟的輸入內容並呈現相關選項，Bot 可引導使用者找到他們所需的資訊。Bot 提供該資訊後，未來甚至可透過更有效率的方式提供引導，進而找出類似的資訊。



Azure 搜尋服務

透過 [Azure 搜尋服務](#)，您可建立有效率的搜尋索引，讓 Bot 輕鬆搜尋、利用 Facet 過濾資料和篩選。考慮使用 Azure 入口網站建立的搜尋索引。

A screenshot of the Azure portal's 'Search service' configuration page. On the left, there's a sidebar with 'Data Source' (musiciandatasource), 'Index' (Customize target index), and 'Indexer' (Import your data). The main area shows a table for defining fields in an index named 'musicianindex'. The table has columns: FIELD NAME, TYPE, RETRIEVABLE, FILTERABLE, SORTABLE, FACETABLE, and SEARCHABLE. Rows include 'imageURL' (Edm.String, checked for all), 'Name' (Edm.String, checked for all), 'Era' (Edm.String, checked for all), 'Description' (Edm.String, checked for all), 'id' (Edm.String, checked for RETRIEVABLE), and 'rid' (Edm.String, checked for RETRIEVABLE). A note at the top says: 'We provided a default index for you. Right-click to delete the fields you don't need. Everything is editable, but once the index is built, deleting or changing existing fields will require re-indexing your documents.'

您想要存取資料存放區的所有屬性，因此將每個屬性設為「可擷取」。您想要按姓名尋找音樂家，因此將 **Name** 屬性設為「可搜尋」。最後，您希望可利用 Facet 進一步篩選音樂家的年代，因此將 **Eras** 屬性設為「可 Facet」和「可篩選」。

Facet 功能可針對指定屬性判斷存在於資料存放區中的值，以及每個值的規模。例如，此螢幕擷取畫面顯示資料存放區中有 5 個不同的年代：

A screenshot of a facet selection interface. It asks 'Which era of music are you interested in?'. Below are five buttons: 'Romantic (11)', 'Classical (3)', 'Baroque (2)', 'Impressionist (2)', and 'Modernist (1)'. The 'Romantic' button is highlighted with a blue arrow pointing to it from the left.

篩選可進一步僅選取特定屬性的指定執行個體。例如，您可篩選上述結果集，以僅包含 **Era** 等於「浪漫時期」的項目。

NOTE

如需完整範例，以了解如何使用 Azure Document DB、Azure 搜尋服務和 Microsoft Bot Framework 建立知識型 Bot，請參閱範例 Bot。

簡單起見，以上範例顯示的搜尋索引是使用 Azure 入口網站建立。亦可透過程式設計方式建立索引。

QnA Maker

有些知識型 Bot 的功能僅單純回答常見問題集 (FAQ)。QnA Maker 是功能強大的工具，專用於此使用案例。QnA Maker 的內建功能可從現有的常見問題集網站抓取問答，此外還可讓您手動配置專屬的自訂問答清單。QnA Maker 支援自然語言處理功能，甚至可針對用字與預期有些微不同的問題提供回答。不過，其並無語意語言理解能力。舉例來說，QnA Maker 就無法判斷「幼犬 (puppy)」是「狗 (dog)」的一種。

使用 QnA Maker Web 介面，您即可設定具有三組問答的知識庫：

Knowledge base

The screenshot shows the QnA Maker Knowledge base interface. At the top, there is a search bar labeled "Search the knowledge base" and a button "+ Add QnA pair". Below the search bar, there is a gear icon. The main area is divided into two columns: "Question" and "Answer". There are three entries:

- Question: can I bring my dog? Answer: Dogs are not allowed
- Question: can I bring my vodka? Answer: Alcohol is not allowed
- Question: hi Answer: hello

接著，您可提出一連串的問題進行測試：

The screenshot shows the QnA Maker Test interface. At the top, there is a "Test" button and a "Start over" button. Below the buttons, there is a text input field with placeholder text "Type your message...". The interface displays three messages:

- Message 1: Can I bring my tea? Inspect You Response: Alcohol is not allowed
- Message 2: ScreenShots at 9:35 AM
- Message 3: Can I bring my vodka? Inspect You Response: Alcohol is not allowed
- Message 4: ScreenShots at 9:34 AM
- Message 5: Can I bring my dog? Inspect You Response: Dogs are not allowed
- Message 6: ScreenShots at 9:34 AM

Bot 可正確回答直接對應至知識庫中已經配置的問題。不過，卻無法正確回答「我能不能自備茶飲？」這樣的問題，因為此問題最類似於「我能不能自備伏特加？」的問題結構。QnA Maker 之所以無法提供正確回答問題，是因為其本身並無法理解文字的意義。QnA Maker 並不知道「茶」是一種無酒精飲料。因此便會回答「禁帶酒精飲料」。

TIP

建立 QnA 組合，然後利用交談下方的 [檢查] 按鈕，針對每個錯誤回答選取其他答案，藉此進行測試並反覆訓練 Bot。

LUIS

有些知識型 Bot 需要自然語言處理 (NLP) 能力，以便分析使用者訊息並判斷其意圖。[Language Understanding \(LUIS\)](#) 提供快速且有效率的方式讓您為 Bot 新增 NLP 功能。LUIS 可讓您依據需求從 Bing 和 Cortana 選用預先建置的現有模型，亦可打造專屬的特製化模型。

使用龐大資料集時，以每種變異實體訓練 NLP 模型不一定可行。例如，在音樂播放 Bot 中，使用者可傳送「播放雷鬼」、「播放 Bob Marley」或「播放 One Love」等訊息。雖然 Bot 可將這每一個訊息對應到「playMusic」意圖，但若未以每個表演者、類別和歌曲名稱逐一訓練，NLP 模型可能無法識別該實體是類別、表演者或歌曲。利用 NLP 模型識別類型為「音樂」的一般實體後，Bot 則可在其資料存放區中搜尋該實體，並接續處理。

結合搜尋、QnA Maker 和/或 LUIS

搜尋、QnA Maker 和 LUIS 各自都是強大的工具，而相互搭配運用則可打造出能擁有上述多種能力的知識型 Bot。

LUIS 和搜尋

在[前文提及](#)的音樂祭 Bot 範例中，Bot 可藉由顯示呈現節目表按鈕引導交談進行。不過，此 Bot 也可使用 LUIS 整合自然語言理解能力，以判斷問題的意圖和實體，例如「Romit Girdhar 演奏什麼類型的音樂？」接著，Bot 可使用音樂家姓名在 Azure 搜尋索引中搜尋。

由於有很多可能的值，因此不太可能以每個音樂家的姓名來訓練模型，但您可提供足夠的代表性範例，讓 LUIS 可正確識別出眼前的實體。例如，您可考慮提供以下音樂家姓名來訓練模型：

The screenshot shows a user interface for LUIS. On the left, there is a text input field containing the question "what kind of music does romit girdhar play ?". To the right of the input field is a dropdown menu labeled "answerGenre". Below the input field is a large green "Submit" button.

The screenshot shows a second instance of the LUIS input interface. The text input field contains the question "what genre of music is foxygen ?". To the right of the input field is a dropdown menu labeled "answerGenre". Below the input field is a large green "Submit" button.

使用新的表達方式如「披頭四演奏什麼類型的音樂？」測試此模型時，LUIS 可成功判斷意圖「answerGenre」並識別實體「披頭四」。不過，若您送出較長的問題如「The Devil Makes Three 演奏什麼類型的音樂？」，LUIS 則只會將「The Devil」判斷為實體。

```
"entities": [
  {
    "entity": "the devil",
    "type": "musician",
    "startIndex": 25,
    "endIndex": 33,
    "score": 0.300864249
  }
]
```

利用可代表基本資料集的範例實體訓練模型，有助於提高 Bot 語言理解的準確性。

TIP

一般而言，建議讓模型遇到在實體辨識中識別贅字詞時出錯，例如，從「請打電話給 John Smith」表達方式中識別「請 John Smith」；而不是以識別的字詞太少來做為出錯依據，例如，從「請打電話給 John Smith」表達方式識別「John」。搜尋索引會忽略「請 John Smith」片語中無關的字詞如：「請」。

LUIS 和 QnA Maker

部分知識型 Bot 可能搭配 LUIS 使用 QnA Maker 回答基本問題，以判斷意圖、擷取實體並叫用其他更詳盡的對話方塊。例如，請考慮使用簡易型 IT 技術支援中心 Bot。此 Bot 可使用 QnA Maker 回答有關 Windows 或 Outlook 的基本問題，此外也必須協助需要意圖辨識及使用者和 Bot 來回通訊的狀況，例如：重設密碼。有幾種方式可讓 Bot 混合實作 LUIS 和 QnA Maker：

1. 同時呼叫 QnA Maker 和 LUIS，然後使用第一個傳回特定閾值分數的資訊回應使用者。
2. 先呼叫 LUIS，如果沒有任何意圖符合特定閾值分數（即觸發「無」意圖），則呼叫 QnA Maker。或者，針對 QnA Maker 建立 LUIS 意圖，並使用對應至「QnAIIntent」的範例 QnA 問題饋送至 LUIS 模型。
3. 先呼叫 QnA Maker，如果沒有任何答案符合特定閾值分數，則呼叫 LUIS。

Bot Framework SDK 具備 LUIS 和 QnA Maker 內建支援。此可讓您觸發對話方塊，或使用 LUIS 和/或 QnA Maker 自動回答問題，而無須實作這兩項工具的自訂呼叫。如需詳細資訊，請參閱[分派工具教學課程](#)。

TIP

實作 LUIS、QnA Maker 和/或 Azure 搜尋服務組合時，使用每項工具測試輸入，以判斷每個模型的閾值分數。LUIS、QnA Maker 和 Azure 搜尋服務會各自使用不同的評分條件來產生分數，因此這些工具產生的分數不一定相容。此外，LUIS 和 QnA Maker 會標準化分數。在 LUIS 模型中視為「良好」的分數，在其他模型不一定分類到「良好」。

範例程式碼

- 如需相關範例以了解如何使用適用於 .NET 的 Bot Framework SDK 建立基本知識型 Bot，請參閱 GitHub 中的[知識型 Bot 範例](#)。

將交談從 Bot 切換為人類

2019/5/10 • [Edit Online](#)

無論 Bot 擁有多高的人工智慧，有時仍可能需要將交談遞交給人類。Bot 應該要辨識何時需要進行遞交，並為使用者提供明確且順暢的切換。

需要人為介入的情況

有各種不同的情況可能需要 Bot 將交談的控制權切換給人類。這些情況其中一些是「分級」、「呈報」及「監督」。

分級

一般技術支援中心的來電是從一些 Bot 可以輕鬆回答的基本問題開始。作為輸入要求之第一個回應者的 Bot，可以收集使用者名稱、地址、問題的描述或任何其他相關資訊，然後將交談的控制權切換給代理人員。使用 Bot 來分類連入要求，可讓代理人員將時間花在解決問題，而不是收集資訊。

升級

在技術支援中心案例中，Bot 除了收集資訊（例如重設使用者的密碼）之外，還可以回答基本問題並解決簡單的問題。不過，如果交談指出使用者的問題太過複雜而需要人為介入，Bot 必須呈報此問題給人類的代理人員。若要實作這種類型的案例，Bot 必須能夠區別它可以獨立解決的問題，以及必須呈報給人的問題。Bot 可透過許多方法來判定它需要將交談的控制權移轉給人類。

使用者導向的功能表

也許讓 Bot 處理這個難題的最簡單方式，是向使用者呈現包含多個選項的功能表。Bot 可獨立處理的工作會出現在標示為 [與代理人員聊天] 之連結上方的功能表。這種類型的實作不需要任何進階的機器學習服務或自然語言理解。當使用者選取 [與代理人員聊天] 選項時，Bot 只會將交談的控制權移轉給人類的代理人員。

案例導向

Bot 可能會根據它是否判斷為能夠處理手邊的案例，來決定是否要轉移控制權。Bot 會收集有關使用者要求的一些資訊，然後查詢其功能的內部清單，以判定它是否能夠處理該要求。如果 Bot 判定它能夠處理要求，就會這樣做，但如果 Bot 判定要求超出它可以解決的問題範圍，則會將交談的控制權轉移給人類的代理人員。

自然語言

自然語言理解和情感分析可協助 Bot 決定何時要將交談的控制權轉移給人類的代理人員。當試圖判斷使用者何時感到挫折，或想要與人類的代理人員交談時，這極具價值。

Bot 透過使用可推斷情感的文字分析 API 或使用 LUIS API 來分析使用者訊息的內容。

TIP

自然語言理解不一定是判定 Bot 何時應該將交談控制權轉移給人類的最佳方法。與人類一樣，Bot 不見得一律能夠猜對，但無效的回應會讓使用者感到挫折。不過，如果使用者從包含有效選項的功能表中進行選取，Bot 一律會適當回應該輸入。

監督

在某些情況下，人類的代理人員想要監視交談，而不是控制。

例如，請考慮使用技術支援中心案例，其中 Bot 正在與使用者通訊以診斷電腦問題。機器學習服務模型可協助 Bot 判定問題的最可能原因；不過，在建議使用者採取特定動作之前，Bot 可以私下與人類代理人員確認診斷和解決方法，並要求授權以繼續進行。代理人員接著會按一下按鈕，Bot 即會向使用者呈現解決方案，這個問題就解決了。Bot 仍然在執行大部分的工作，但代理人員保留最終決定的控制權。

切換交談的控制權

當 Bot 決定要將交談的控制權移轉給人類時，它可以通知使用者即將進行移轉並將交談置入「等待」狀態，直到它確認代理人員有空為止。

Bot 正在等待人類時，它可能會以預設回應（例如「在佇列中等待」）自動回答所有的連入使用者訊息。此外，如果使用者傳送特定訊息（例如「沒關係」或「取消」），您可以讓 Bot 從「等待」狀態中移除交談。

您可以指定在設計 Bot 時，如何將代理人員指派給等待的使用者。例如，Bot 可以實作一個簡單的佇列系統：先進先出。更複雜的邏輯會根據地理位置、語言或某些其他因素，將使用者指派給代理人員。Bot 還可以向代理人員呈現某種類型的 UI，供他們用來選取使用者。當代理人員有空時，其會連線到 Bot 並加入交談。

IMPORTANT

即使在代理人員參與之後，Bot 仍然是交談的幕後推動者。使用者和代理人員永遠不會直接彼此通訊；它們只是透過 Bot 路由傳送訊息。

在使用者與代理人員之間路由傳送訊息

代理人員連線到 Bot 之後，Bot 就會開始在使用者與代理人員之間路由傳送訊息。雖然看起來使用者和代理人員是直接與彼此聊天，但實際上他們是透過 Bot 交換訊息。Bot 會接收來自使用者的訊息並將這些訊息傳送給代理人員，接著接收來自代理人員的訊息，並將這些訊息傳送給使用者。

NOTE

在更進階的案例中，Bot 可以承擔除了只是在使用者與代理人員之間路由傳送訊息之外的責任。例如，Bot 可以決定哪個回應合適，並且只要求代理人員確認以繼續進行。

其他資源

- [對話方塊](#)
- [文字分析 API](#)
- [使用對話方塊來管理對話流程 \(.NET\)](#)
- [使用對話方塊來管理對話流程 \(Node.js\)](#)
- [文字分析 API](#)

將 Bot 內嵌至應用程式

2019/2/28 • [Edit Online](#)

雖然 Bot 最常存在於應用程式之外，但也可以與應用程式整合。例如，您可以將[知識 Bot](#)內嵌在應用程式內，以協助使用者在複雜的應用程式結構內尋找不易找到的資訊。您可以將 Bot 內嵌在技術支援中心的應用程式內，做為傳入使用者要求的第一個回應程式。Bot 可以獨立解決簡單的問題，並將較為複雜的問題遞交給人類專員。

整合應用程式與 Bot

整合 Bot 與應用程式的方式需視應用程式的類型而有所不同。

原生行動應用程式

以原生程式碼建立的應用程式可以使用[直接線路 API](#)，透過 REST 或 websocket，與 Bot Framework 通訊。

網頁型行動應用程式

使用 Web 語言和架構 (例如 [Cordova](#)) 建立的行動應用程式，可能會藉由使用與[內嵌在網站內的 Bot](#)所使用的相同的元件來與 Bot Framework 通訊，只要封裝在原生應用程式殼層中即可。

IoT 應用程式

IoT 應用程式可以使用[直接線路 API](#) 與 Bot Framework 進行通訊。在某些案例中，它也可以使用 [Microsoft 認知服務](#)啟用影像辨識和語音等功能。

其他類型的應用程式和遊戲

其他類型的應用程式和遊戲可以使用[直接線路 API](#) 與 Bot Framework 進行通訊。

建立執行 Bot 的跨平台行動應用程式

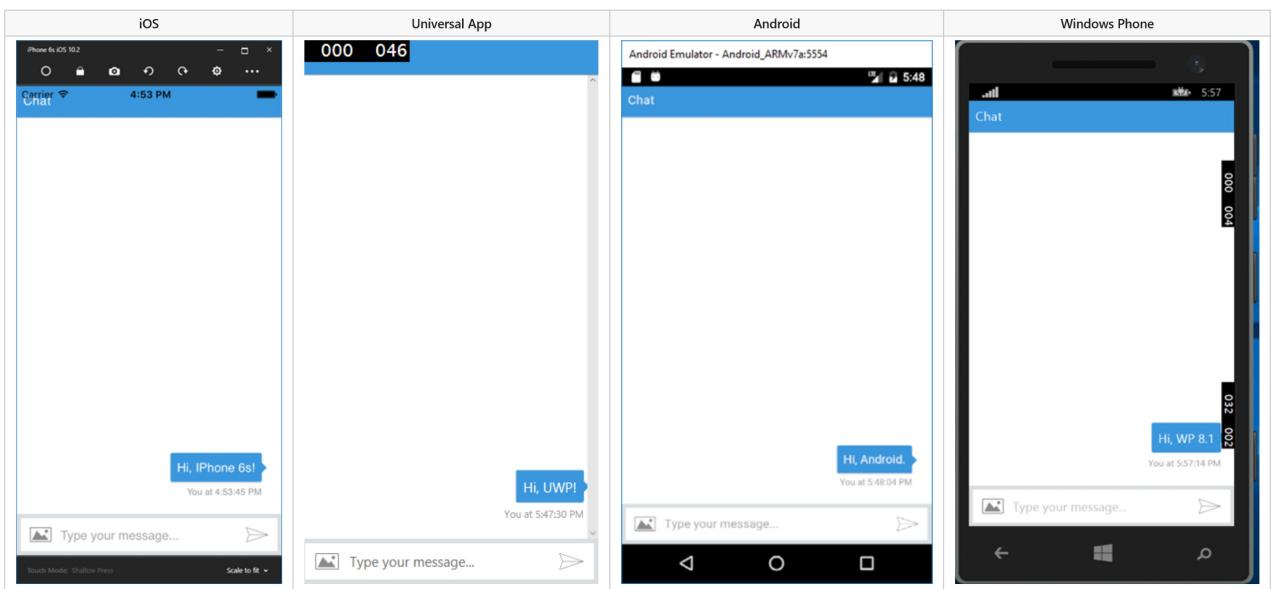
在此範例中，是使用建立跨平台行動應用程式的熱門工具 [Xamarin](#) 來建立執行 Bot 的行動應用程式。

首先，建立簡單的 Web 檢視元件，並運用在架設[網路聊天控制項](#)。然後，使用 Azure 入口網站，新增網路聊天頻道。

接下來，指定已註冊的網路聊天 URL 做為 Xamarin 應用程式中 Web 檢視控制項的來源：

```
public class WebPage : ContentPage
{
    public WebPage()
    {
        var browser = new WebView();
        browser.Source = "https://webchat.botframework.com/embed/<YOUR SECRET KEY HERE>";
        this.Content = browser;
    }
}
```

您可以使用此程序來建立跨平台行動應用程式，以網站聊天控制項呈現內嵌的 Web 檢視。



其他資源

- [Direct Line API](#)
- [Microsoft 認知服務](#)

將 Bot 嵌入網站

2019/5/10 • [Edit Online](#)

Bot 通常位於網站外部，但也可能內嵌於網站中。例如，您可以將[知識 Bot](#)內嵌在網站中，以便使用者快速尋找資訊，否則在複雜的網站結構內往往不容易找到。您也可以將 Bot 內嵌在技術支援中心的網站內，做為傳入使用者要求的第一個回應程式。Bot 可以獨立解決簡單的問題，並將較為複雜的問題遞交給人類專員。

本文章探討 Bot 與網站的整合，以及使用 *backchannel* 機制的過程，以便在網頁和 Bot 之間進行私密通訊。

Microsoft 提供兩種不同的 Bot 與網站整合方式：[Skype web 控制項](#)和開放原始碼 web 控制項。

Skype web 控制項

[Skype web 控制項](#)本質上是已啟用 web 控制項中的 Skype 用戶端。內建的 Skype 驗證可讓 Bot 驗證和辨識使用者，而不需要開發人員撰寫任何自訂程式碼。Skype 會自動辨識其網頁用戶端中使用的 Microsoft 帳戶。

由於 Skype web 控制項僅做為 Skype 前端，使用者的 Skype 用戶端會自動存取任何 web 控制項促成的完整對話內容。即使在網頁瀏覽器關閉之後，使用者也可以繼續與 Skype 用戶端的 Bot 進行互動。

開放原始碼 web 控制項

[開放原始碼網路聊天控制項](#)是以 ReactJS 為基礎，並使用 [Direct Line API] [directLineAPI](#)與 Bot Framework 進行通訊。網路聊天控制項提供實作網路聊天所需的空白畫布，讓您完整掌控其行為和其所提供的使用者體驗。

Backchannel 機制可讓裝載控制項的網頁，運用使用者完全無法察覺的方式與 Bot 直接進行通訊。這項功能可讓數個實用案例變得可行：

- 網頁可以傳送相關資料至 Bot (例如 GPS 位置)。
- 網頁可以向 Bot 通知使用者動作 (例如「使用者剛才從下拉式清單中選取了選項 A」)。
- 網頁可將登入使用者的驗證權杖傳送給 Bot。
- Bot 可以傳送相關資料至網頁 (例如使用者組合目前的值)。
- Bot 可以傳送「命令」至網頁 (例如變更背景色彩)。

使用 backchannel 機制

[開放原始碼網站聊天控制項](#)使用 [直接線路 API](#) 與 Bot 進行通訊，如此會允許 `activities` 在用戶端與 Bot 之間來回傳送。最常見的活動類型是 `message`，但也有其他類型。比方說，活動類型 `typing` 指明使用者正在輸入或是 bot 正在編譯回覆的工作。

您可以使用 backchannel 機制在用戶端與 Bot 之間交換資訊，而不需要藉由將活動類型設定為 `event` 而將這些資料呈現給使用者。網路聊天控制項會自動忽略 `type="event"` 的任何活動。

範例程式碼

[開放原始碼網路聊天控制項](#)可透過 GitHub 取得。如要深入了解如何使用開放原始碼網路聊天控制項實作 backchannel 機制和適用於 Node.js 的 Bot Framework SDK，請參閱[使用 backchannel 機制](#)。

其他資源

- [Direct Line API](#)
- [開放原始碼網路聊天控制項](#)

- 使用 backchannel 機制

傳送及接收文字訊息

2019/5/10 • [Edit Online](#)

適用於:  SDK v4  SDK v3

您的 Bot 與使用者進行往來通訊的主要方式，都是透過訊息活動。有些訊息可能只包含純文字，有些則可能包含更豐富的內容，例如卡片或附件。Bot 回合處理常式會接收來自使用者的訊息，而您可以從該處將回應傳送給使用者。回合內容物件會提供將訊息傳回給使用者的方法。本文說明如何傳送簡單的文字訊息。

大部分文字欄位都支援 Markdown，但是支援可能因通道而有所不同。

傳送文字訊息

若要傳送簡單的文字訊息，請指定要以活動的形式傳送的字串：

- [C#](#)
- [JavaScript](#)

在 Bot 的活動處理常式中，使用回合內容物件的 `SendActivityAsync` 方法傳送單一訊息回應。您也可以使用該物件的 `SendActivitiesAsync` 方法同時傳送多個回應。

```
await turnContext.SendActivityAsync($"Welcome!");
```

接收文字訊息

若要接收簡單的文字訊息，請使用 *activity* 物件的 *text* 屬性。

- [C#](#)
- [JavaScript](#)

在 Bot 的活動處理常式中，使用下列程式碼來接收訊息。

```
var responseMessage = turnContext.Activity.Text;
```

其他資源

- 如需一般活動處理方式的詳細資訊，請參閱[活動處理](#)。
- 如需格式化的詳細資訊，請參閱 Bot Framework 活動結構描述的[訊息活動區段](#)。

後續步驟

[將媒體新增至訊息](#)

將媒體新增至訊息

2019/5/10 • [Edit Online](#)

適用於:  SDK v4  SDK v3

使用者與 Bot 之間交換的訊息可以包含媒體附件，例如影像、視訊、音訊和檔案等。Bot Framework SDK 支援將豐富訊息傳送給使用者的工作。若要判斷通道 (Facebook、Skype、Slack 等等) 支援的豐富訊息類型，請參閱通道文件中的限制相關資訊。

請參閱[設計使用者體驗](#)，以取得可用卡片的範例。

傳送附件

若要傳送影像或視訊之類的使用者內容，您可以將附件或附件清單新增至訊息。

請參閱[設計使用者體驗](#)，以取得可用卡片的範例。

- [C#](#)
- [JavaScript](#)

`Activity` 物件的 `Attachments` 屬性包含 `Attachment` 物件的陣列，代表附加至訊息的媒體附件和複合式資訊卡 (Rich Card)。若要將媒體附件新增至訊息，請建立 `reply` 活動的 `Attachment` 物件 (有別於搭配 `CreateReply()` 建立的活動)，並設定 `ContentType`、`ContentUrl` 和 `Name` 屬性。

這裡顯示的原始程式碼是以[處理附件](#)範例為基礎。

若要建立回覆訊息，請定義文字然後設定附件。將每個附件類型指派給回覆的方式都相同，但是各種附件的設定與定義都不同，如下列程式碼片段所示。下列程式碼會設定內嵌附件的回覆：

Bots/AttachmentsBot.cs [[!code-csharpinline attachment](#)]

接下來，我們來看看附件的類型。首先是內嵌附件：

Bots/AttachmentsBot.cs [[!code-csharpinline attachment](#)]

然後是上傳的附件：

Bots/AttachmentsBot.cs [[!code-csharpuploaded attachment](#)]

最後是網際網路附件：

Bots/AttachmentsBot.cs [[!code-csharponline attachment](#)]

如果附件是影像、音訊或視訊，則連接器服務會以可讓[通道](#)在對話內轉譯該附件的方式，將附件資料傳輸至通道。如果附件是檔案，檔案 URL 將會轉譯為對話內的超連結。

傳送主圖卡

除了簡單的影像或影片附件以外，您也可以附加主圖卡，這可讓您將影像和按鈕結合在單一物件中，並將其傳送給使用者。大部分文字欄位都支援 Markdown，但是支援可能因通道而有所不同。

- [C#](#)
- [JavaScript](#)

若要撰寫含有主圖卡和按鈕的訊息，您可以將 `HeroCard` 附加至訊息。

這裡顯示的原始程式碼是以[處理附件](#)範例為基礎。

Bots/AttachmentsBot.cs [!code-csharpHero card]

處理複合式資訊卡 (Rich Card) 內的事件

若要處理複合式資訊卡 (Rich Card) 內的事件，請使用`_卡片動作_物件`，指定在使用者按一下按鈕或點選卡片的某區段時，所應發生的情況。每個卡片動作都有「類型」和「值」。

為達到正常運作，請為卡片上每個可點選的項目指派一個動作類型。下表列出和說明可用的動作類型，以及相關聯的值屬性應有內容。

類型	說明	值
openUrl	在內建瀏覽器中開啟 URL。	要開啟的 URL。
imBack	將訊息傳送到 Bot，並在聊天中張貼可見的回應。	要傳送之訊息的文字。
postBack	將訊息傳送到 Bot，但無法在聊天中張貼可見的回應。	要傳送之訊息的文字。
call	開始撥打電話。	以下列格式撥打電話的目的地： <code>tel:123123123123</code> 。
playAudio	播放音訊。	要播放的音訊 URL。
playVideo	播放影片。	要播放的影片 URL。
showImage	顯示影像。	要顯示的影像 URL。
downloadFile	下載檔案。	要下載的檔案 URL。
signin	起始 OAuth 登入程序。	要起始的 OAuth 流程 URL。

使用各種不同事件類型的主圖卡

下列程式碼說明使用各種複合式資訊卡 (Rich Card) 事件的範例。

- [C#](#)
- [JavaScript](#)

如需所有可用卡片的範例，請參閱[C# 卡片範例](#)。

Cards.cs [!code-csharphero cards]

Cards.cs [!code-csharpcards]

傳送調適型卡片

調適型卡片和 MessageFactory 都用於傳送豐富訊息 (包括文字、影像、視訊、音訊和檔案) 來與使用者進行通訊。但是，兩者之間有一些差異。

首先，只有某些通道支援調適型卡片，而提供支援的通道可能部分支援調適型卡片。例如，如果您在 Facebook 中傳送調適型卡片，當文字和影像正常運作時，按鈕將無法運作。MessageFactory 只是 Bot Framework SDK 內的協助程式類別，可為您將建立步驟自動化，而且大部分通道都提供支援。

再者，調適型卡片會以卡片格式傳遞訊息，而通道會決定卡片的版面配置。MessageFactory 傳遞的訊息格式取決於通道，而且不一定是以卡片格式傳遞，除非調適型卡片是附件的一部分。

如需調適型卡片通道支援的最新資訊，請參閱[調適型卡片設計工具](#)。

NOTE

您應對 Bot 所將使用的通道測試這項功能，以確認這些通道是否支援調適型卡片。

- [C#](#)
- [JavaScript](#)

若要使用調適型卡片，請務必新增 `AdaptiveCards` NuGet 套件。

這裡顯示的原始程式碼是以[使用卡片](#)範例為基礎：

Cards.cs [!code-csharpadaptive cards]

傳送浮動切換的卡片

訊息也可以在浮動切換配置中包含多個附件，此時附件會並排顯示，讓使用者可加以捲動。

- [C#](#)
- [JavaScript](#)

這裡顯示的原始程式碼是以[卡片範例](#)為基礎。

首先，建立回覆並將附件定義為清單。

Dialogs/MainDialog.cs [!code-csharpcarousel of cards]

然後新增附件。在此，我們一次一個地新增附件，但是您可以依照喜好隨意地操作要新增卡片的清單。

Dialogs/MainDialog.cs [!code-csharpcarousel of cards]

新增附件之後，您就可以像其他人一樣傳送回覆。

Dialogs/MainDialog.cs [!code-csharpcarousel of cards]

其他資源

請參閱[設計使用者體驗](#)，以取得可用卡片的範例。

如需結構描述的詳細資訊，請參閱 Bot Framework 活動結構描述的 [Bot Framework 卡片結構描述](#)和[訊息活動區段](#)。

範例程式碼	C#	JS
卡片	C# 範例	JS 範例
附件	C# 範例	JS 範例
建議動作	C# 範例	JS 範例

如需其他範例，請參考 [GitHub](#) 上的 Bot Builder 範例存放庫。

後續步驟

新增按鈕來引導使用者的動作

使用按鈕進行輸入

2019/5/10 • [Edit Online](#)

適用於:  SDK v4  SDK v3

您可以讓 Bot 顯示可供使用者點選，以提供輸入的按鈕。按鈕讓使用者只要點選按鈕，即可回答問題或進行選取，而無需使用鍵盤輸入回應，藉以提升使用者體驗。不同於複合式資訊卡 (Rich Card) 中出現的按鈕 (即使在點選之後，使用者仍可看見並可存取)，出現在建議動作窗格內的按鈕會在使用者進行選取後消失。這可以避免使用者在對話中點選過時的按鈕，並簡化 Bot 的開發 (因為您不需要再說明該情境)。

使用按鈕建議動作

「建議動作」可讓 Bot 顯示按鈕。您可以建立一份將在交談單一回合顯示給使用者的建議動作清單 (也稱為「快速回覆」)：

- [C#](#)
- [JavaScript](#)

這裡顯示的原始程式碼是以[建議動作範例](#)為基礎。

```
private static async Task SendSuggestedActionsAsync(ITurnContext turnContext, CancellationToken cancellationToken)
{
    var reply = turnContext.Activity.CreateReply("What is your favorite color?");
    reply.SuggestedActions = new SuggestedActions()
    {
        Actions = new List<CardAction>()
        {
            new CardAction() { Title = "Red", Type = ActionTypes.ImBack, Value = "Red" },
            new CardAction() { Title = "Yellow", Type = ActionTypes.ImBack, Value = "Yellow" },
            new CardAction() { Title = "Blue", Type = ActionTypes.ImBack, Value = "Blue" },
        },
    };
    await turnContext.SendActivityAsync(reply, cancellationToken);
}
```

其他資源

您可以在此存取完整的原始程式碼：[CSharp 範例](#)或[JavaScript 範例](#)。

後續步驟

[儲存使用者和對話資料](#)

儲存使用者和對話資料

2019/5/10 • [Edit Online](#)

適用於:  SDK v4  SDK v3

Bot 原本就是無狀態。部署 Bot 後，在不同的回合中，Bot 便無法在相同程序中或同部電腦上執行。不過，Bot 可能需要追蹤對話的內容，以便管理其行為，以及記住先前問題的答案。Bot Framework SDK 的狀態和儲存體功能可讓您將狀態新增至 Bot。Bot 會使用狀態管理和儲存體物件來管理和保存狀態。狀態管理員會提供抽象層，讓您使用屬性存取子存取狀態屬性，而不受基礎儲存體類型所影響。

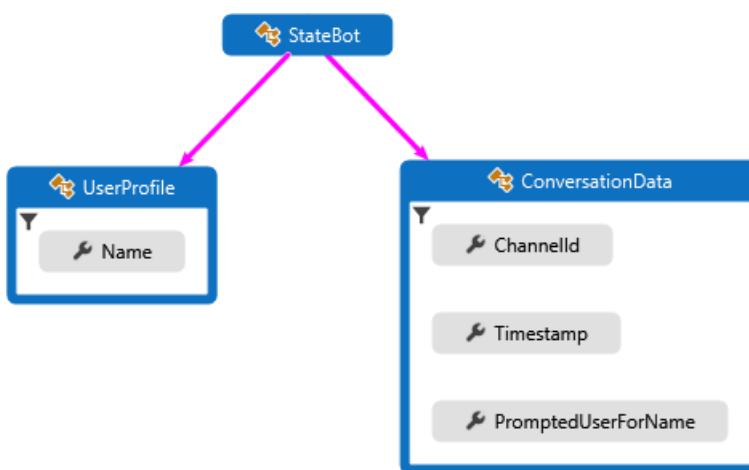
必要條件

- 需具備 [Bot 基本概念](#) 以及 Bot 如何 [管理狀態](#) 的知識。
- 本文中的程式碼是以 [狀態管理 Bot 範例](#) 為基礎。您需要 [CSharp](#) 或 [JavaScript](#) 中的一份範例。

關於此範例

收到使用者輸入後，此範例會檢查儲存的對話狀態，以查看之前是否已提示此使用者提供名稱。如果沒有，則會要求使用者的名稱，並將該輸入儲存在使用者狀態中。如果有，則使用者狀態中儲存的名稱就會用來與使用者對話，而其輸入的資料（包括接收時間與輸入通道識別碼）會傳回給使用者。時間和通道識別碼值會從使用者對話資料中擷取，然後再儲存到對話狀態。下圖顯示 Bot、使用者設定檔及對話資料類別之間的關聯性。

- [C#](#)
- [JavaScript](#)



定義類別

- [C#](#)
- [JavaScript](#)

設定狀態管理的第一步是定義類別，以包含我們要在使用者和對話狀態中管理的所有資訊。在此範例中，我們已定義下列項目：

- 在 **UserProfile.cs** 中，我們針對 Bot 要收集的使用者資訊定義了 **UserProfile** 類別。
- 在 **ConversationData.cs** 中，我們定義了 **ConversationData** 類別來控制收集使用者資訊時的對話狀態。

下列程式碼範例示範如何建立 **UserProfile** 類別的定義。

UserProfile.cs [!code-csharpUserProfile]

建立對話和使用者狀態物件

- [C#](#)
- [JavaScript](#)

接下來，我們會註冊用來建立 `UserState` 和 `ConversationState` 物件的 `MemoryStorage`。使用者和對話狀態物件會在 `Startup` 上建立，而相依性會插入 Bot 建構函式。為 Bot 註冊的其他服務為：認證提供者、配接器及 Bot 實作。

Startup.cs [!code-csharpConfigureServices]

StateManagementBot.cs [!code-csharpStateManagement]

新增狀態屬性存取子

- [C#](#)
- [JavaScript](#)

現在我們要使用可提供控制代碼給 `BotState` 物件的 `CreateProperty` 方法建立屬性存取子。每個狀態屬性存取子，都可讓您取得或設定相關聯的狀態屬性值。在我們使用狀態屬性之前，我們會使用每個存取子從儲存體載入屬性，並從狀態快取中取得該屬性。為取得狀態屬性相關金鑰的正確範圍，我們會呼叫 `GetAsync` 方法。

StateManagementBot.cs [!code-csharpStateAccessors]

從 Bot 存取狀態

前一節討論了初始階段步驟，以將狀態屬性存取子新增至 Bot。現在，我們可以在執行階段使用這些存取子，來讀取和寫入狀態資訊。以下範例程式碼會使用下列邏輯流程：

- 如果 `userProfile.Name` 為空白，且 `conversationData.PromptedUserForName` 為 `true`，我們就會擷取所提供的使用者名稱，並將其儲存在使用者狀態中。
- 如果 `userProfile.Name` 為空白，且 `conversationData.PromptedUserForName` 為 `false`，我們就會要求使用者的名稱。
- 如果先前已儲存 `userProfile.Name`，則我們會從使用者輸入中擷取訊息時間和通道識別碼，將所有資料回應給使用者，並將擷取的資料儲存在對話狀態中。

- [C#](#)
- [JavaScript](#)

StateManagementBot.cs [!code-csharpOnMessageActivityAsync]

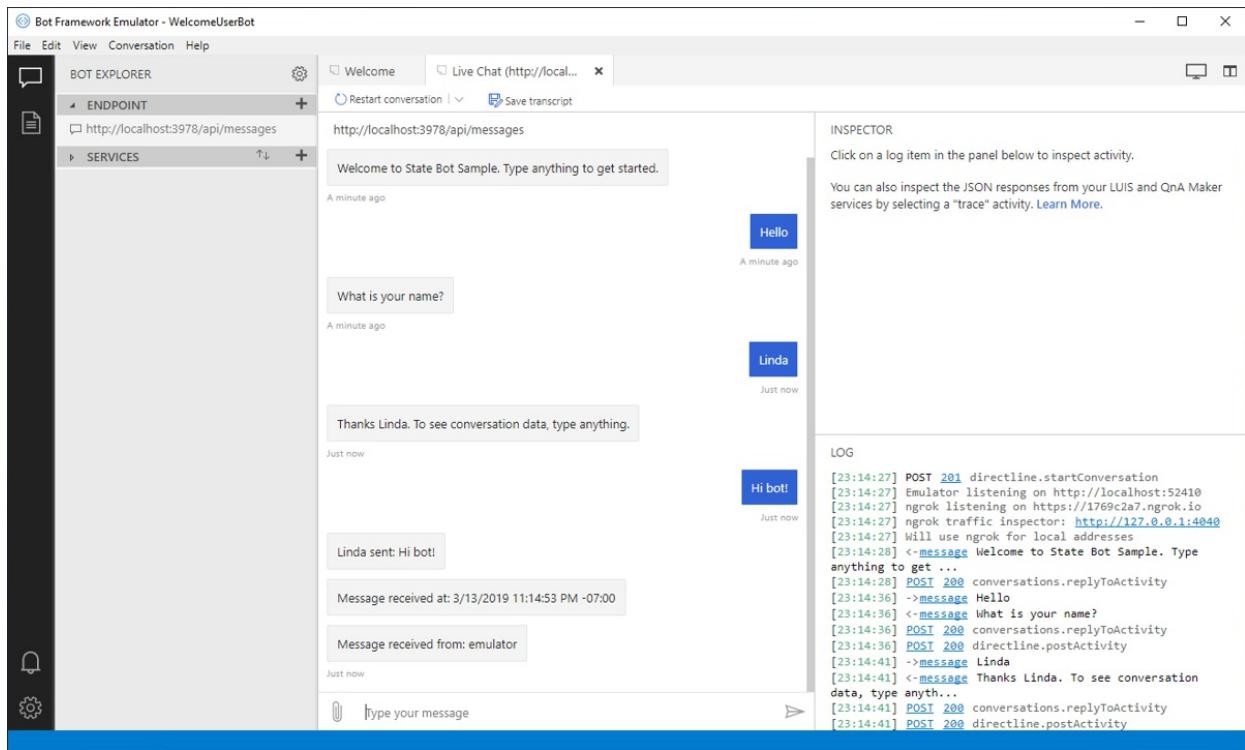
結束回合處理常式之前，我們會使用狀態管理物件的 `SaveChangesAsync()` 方法，將所有狀態變更寫回儲存體。

StateManagementBot.cs [!code-csharpOnTurnAsync]

測試 Bot

下載並安裝最新版 [Bot Framework Emulator](#)

1. 在您的電腦本機執行範例。如需相關指示，請參閱 [C# 範例](#) 或 [JS 範例](#) 的讀我檔案。
2. 使用模擬器來測試 Bot，如下所示。



其他資源

隱私權：如果您想要儲存使用者的個人資料，請務必符合[一般資料保護規範](#)。

狀態管理：所有狀態管理呼叫都是非同步的，預設會使用「最後寫入為準」。實務上，您應該在 Bot 中儘可能密集取得、設定及儲存狀態。

重要商務資料：使用 Bot 狀態來儲存喜好設定、使用者名稱或其所排序的最後一個項目，但不要來儲存重要的商務資料。對於重要資料，請[建立自己的儲存體元件](#)，或直接寫入[儲存體](#)。

Recognizer-Text: 此範例會使用 Microsoft/Recognizers-Text 程式庫來剖析及驗證使用者輸入。如需詳細資訊，請參閱[概觀](#)頁面。

後續步驟

您現在已知道如何設定狀態來協助您讀取 Bot 資料以及將其寫入儲存體，讓我們說明如何詢問使用者一連串的問題、驗證其答案，以及儲存其輸入。

[建立您自己的提示，以收集使用者輸入。](#)

建立您自己的提示，以收集使用者輸入

2019/5/10 • [Edit Online](#)

適用於：  SDK v4  SDK v3

Bot 和使用者之間的對話，通常牽涉到要求（提示）使用者輸入資訊、剖析使用者的回應，然後根據該資訊行動。Bot 應該追蹤交談的內容，以便管理其行為，以及記住先前問題的答案。Bot 的「狀態」是其所追蹤的資訊，以便適當地回應內送訊息。

TIP

對話方塊程式庫會提供內建提示，讓使用者可使用更多的功能。您可以在[實作循序對話流程](#)文章中找到這些提示的範例。

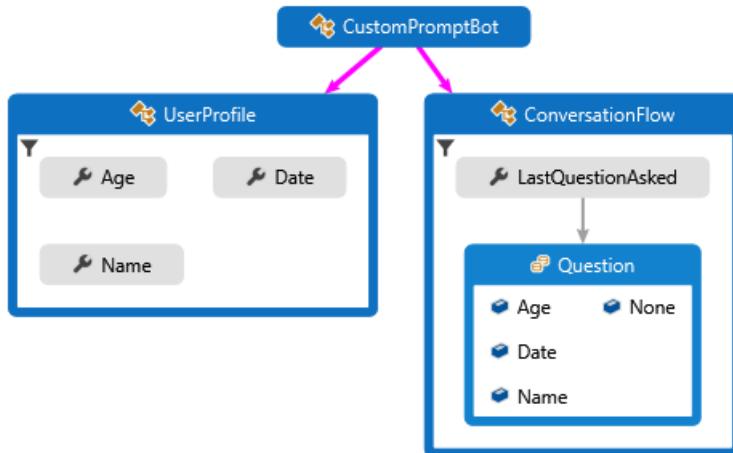
必要條件

- 本文中的程式碼是以「提示使用者輸入」範例為基礎。您需要一份 [C# 範例](#) 或 [JavaScript 範例](#)。
- 管理狀態以及如何 [儲存使用者和交談資料](#) 的知識。

關於範例程式碼

範例 Bot 會詢問使用者一連串的問題，驗證他們的一些答案，以及儲存其輸入。下圖顯示 Bot、使用者設定檔及對話流程類別之間的關聯性。

- [C#](#)
- [JavaScript](#)



- Bot 將要收集的使用者資訊的 **UserProfile** 類別。
- ConversationFlow** 類別會控制收集使用者資訊時的對話狀態。
- 內部 **ConversationFlow.Question** 列舉可用於追蹤我們在交談中的位置。

使用者狀態會追蹤使用者的名稱、年齡與所選日期，而對話狀態會追蹤我們剛才詢問使用者的內容。因為我們不打算部署此 Bot，所以我們會將使用者和交談狀態設定為使用「記憶體儲存體」。

我們會使用 Bot 的訊息回合處理常式加上使用者和對話狀態屬性，來管理對話流程和輸入集合。在 Bot 中，我們會記錄每個訊息回合處理常式循環期間收到的狀態屬性資訊。

建立對話和使用者物件

- [C#](#)
- [JavaScript](#)

使用者和對話狀態物件會在啟動時建立，而相依性會插入 Bot 建構函式。

Startup.cs [!code-csharpStartup]

Bots/CustomPromptBot.cs [!code-csharpcustom prompt bot]

建立屬性存取子

- [C#](#)
- [JavaScript](#)

我們會從建立屬性存取子開始，好讓我們有 `OnMessageActivityAsync` 方法中 `BotState` 的控制代碼。然後，我們會呼叫 `GetAsync` 方法來取得正確範圍的金鑰：

Bots/CustomPromptBot.cs [!code-csharpcustom prompt bot]

最後，我們會使用 `SaveChangesAsync` 方法來儲存資料。

```
await _conversationState.SaveChangesAsync(turnContext);
await _userState.SaveChangesAsync(turnContext);
```

Bot 的訊息回合處理常式

- [C#](#)
- [JavaScript](#)
- [C#](#)
- [JavaScript](#)

若要處理訊息活動，我們會先使用協助程式方法 `FillOutUserProfileAsync()`，然後再使用 `SaveChangesAsync()` 儲存狀態。以下是完整程式碼。

Bots/CustomPromptBot.cs [!code-csharpcustom prompt bot]

剖析及驗證輸入

我們將使用下列準則來驗證輸入。

- 名稱必須是非空白字串。我們的正規做法是修剪空白字元。
- 年齡必須介於 18 與 120 之間。我們的正規做法是傳回一個整數。
- 日期必須是至少未來一小時的任何日期或時間。我們的正規做法是只傳回所剖析輸入的日期部分。

NOTE

對於年齡和日期輸入，我們會使用 [Microsoft/Recognizers-Text](#) 程式庫來執行初始剖析。我們雖然提供範例程式碼，但不會說明文字辨識器程式庫的運作方式，而這只是剖析輸入的方法之一。如需這些程式庫的詳細資訊，請參閱存放庫的讀我。

- [C#](#)
- [JavaScript](#)

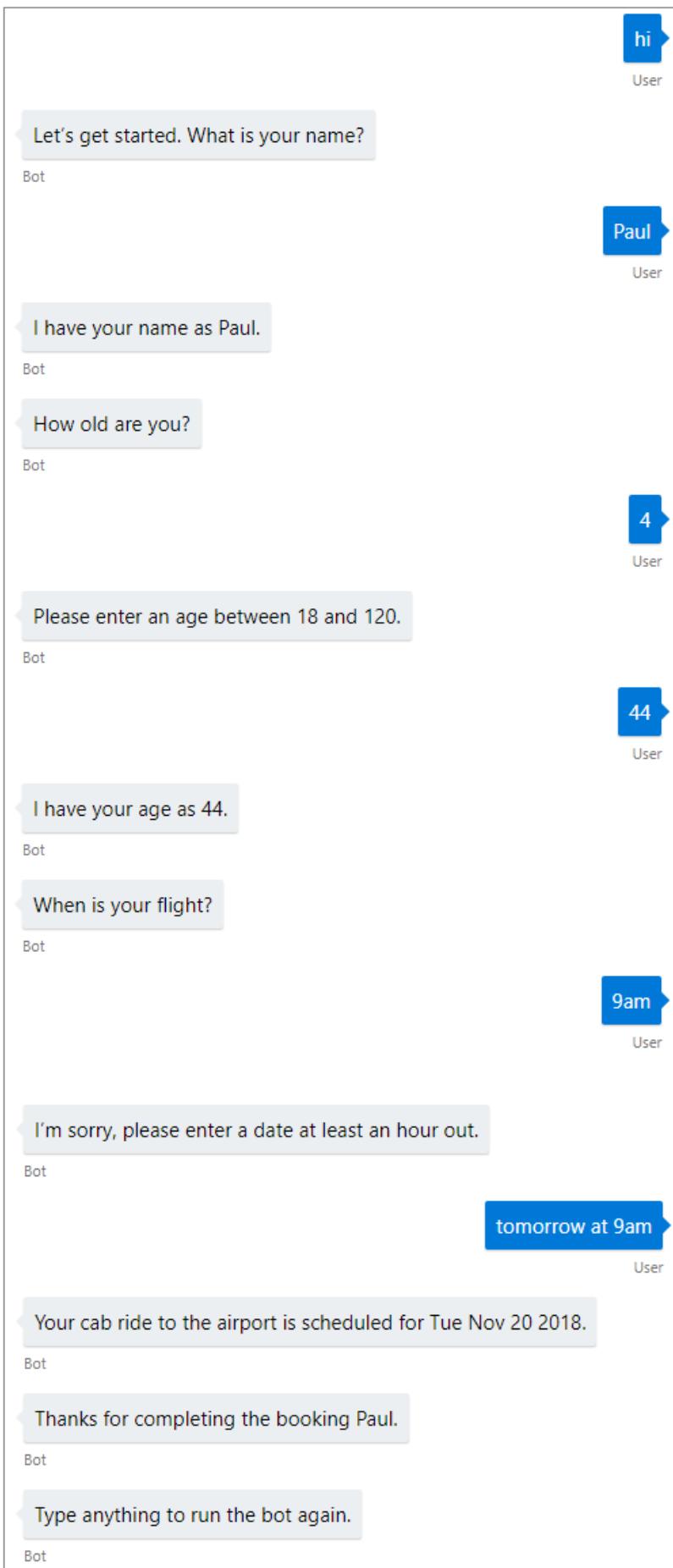
將下列驗證方法新增到您的 Bot。

Bots/CustomPromptBot.cs [!code-csharpcustom prompt bot]

在本機測試 Bot

下載並安裝 [Bot Framework Emulator](#), 以在本機測試 Bot。

1. 在您的電腦本機執行範例。如需相關指示, 請參閱 [C# 範例](#)或 [JS 範例](#)的讀我檔案。
2. 如下所示, 使用模擬器進行測試。



其他資源

Dialogs 程式庫會提供一些類別，以將管理交談的許多層面自動化。

後續步驟

[實作循序對話流程](#)

將歡迎訊息傳送給使用者

2019/5/14 • [Edit Online](#)

適用於:  SDK v4  SDK v3

建立任何 Bot 時的主要目標就是讓您的使用者參與有意義的對話。達成此目標的最佳方式之一就是確保從使用者第一次連線的那一刻，他們就了解您 Bot 的主要用途和功能，以及您 Bot 的建立原因。這篇文章提供程式碼範例，協助您歡迎使用者使用 Bot。

必要條件

- 了解 [bot 基本概念](#)。
- 採用 [C# 範例](#)或 [JS 範例](#)的一份歡迎使用者範例。此範例中的程式碼用來說明如何傳送歡迎訊息。

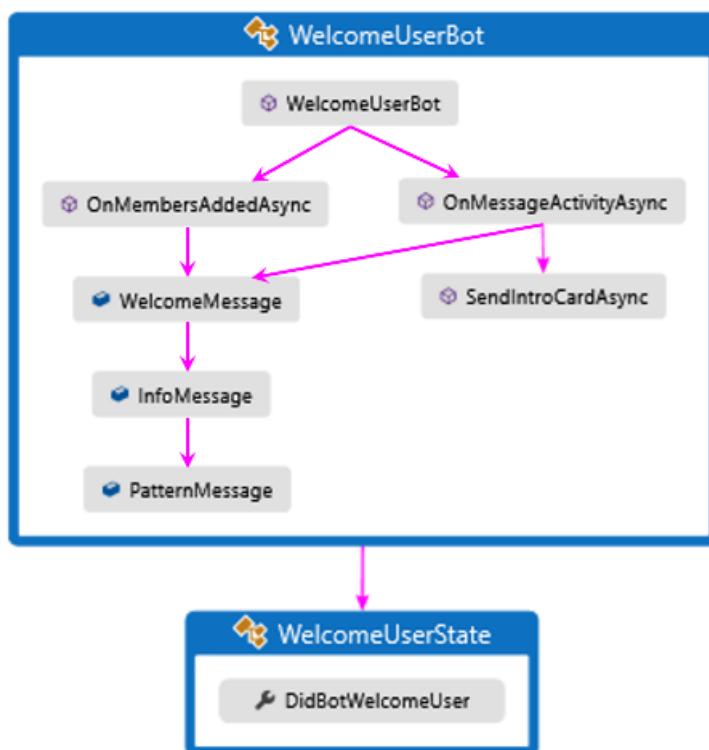
關於此程式碼範例

此程式碼範例示範，如何在新的使用者一開始連線至 Bot 時就偵測到並表示歡迎。下圖顯示此 Bot 的邏輯流程。

- [C#](#)
- [JavaScript](#)

Bot 會遇到的兩個主要事件為

- `OnMembersAddedAsync`，每當有新的使用者連線至 Bot 時便會加以呼叫
- `OnMessageActivityAsync`，每次收到新的使用者輸入時便會加以呼叫。



每當有新的使用者連線時，Bot 便會向其提供 `WelcomeMessage`、`InfoMessage` 和 `PatternMessage`。收到新的使用者輸入時，便會檢查 `WelcomeUserState` 以查看 `DidBotWelcomeUser` 是否設定為 `true`。如果沒有，便會對使用者傳回初始的歡迎使用者訊息。

如果 `DidBotWelcomeUser` 是 `true`，便會評估使用者的輸入。根據使用者輸入的內容，此 Bot 會執行下列其中一項：

- 針對從使用者收到的問候發出回應。
- 顯示主圖卡片來提供關於 Bot 的其他資訊。
- 重新傳送 `WelcomeMessage` 來說明此 Bot 預期的輸入。

建立使用者物件

- [C#](#)
- [JavaScript](#)

使用者狀態物件會在啟動時建立，而相依性會插入 Bot 建構函式。

Startup.cs [!code-csharpConfigureServices]

WelcomeUserBot.cs [!code-csharpClass]

建立屬性存取子

- [C#](#)
- [JavaScript](#)

現在，我們會建立屬性存取子來獲得 `OnMessageActivityAsync` 方法內 `WelcomeUserState` 的控制代碼。然後，呼叫 `GetAsync` 方法來取得正確範圍的金鑰。我們隨後會使用 `SaveChangesAsync` 方法在每個使用者輸入反覆項目之後儲存使用者狀態資料。

WelcomeUserBot.cs [!code-csharp[OnMessageActivityAsync](~/../.BotBuilder-Samples/samples/csharp_dotnetcore/03.welcome-user/bots/WelcomeUserBot.cs?range=68-71, 102-105)]

偵測並歡迎新連線的使用者

- [C#](#)
- [JavaScript](#)

在 **WelcomeUserBot** 中，我們會使用 `OnMembersAddedAsync()` 檢查是否有活動更新，以了解對話中是否新增了新的使用者，然後向其傳送一組三個的初始歡迎訊息，分別是 `WelcomeMessage`、`InfoMessage` 和 `PatternMessage`。此互動的完整程式碼如下所示。

WelcomeUserBot.cs [!code-csharp[WelcomeMessages](~/../.BotBuilder-Samples/samples/csharp_dotnetcore/03.welcome-user/bots/WelcomeUserBot.cs?range=20-40, 55-66)]

歡迎新的使用者並捨棄初始輸入

- [C#](#)
- [JavaScript](#)

此外，務必考量使用者的輸入何時實際上可能包含實用資訊，而這可能因通道而異。若要確保使用者在所有可能的通道上都有美好的體驗，我們會檢查狀態旗標 `didBotWelcomeUser`，如果這是 "false"，我們就不會處理此初始使用者輸入。我們會改為向使用者提供初始歡迎訊息。布林值 `welcomedUserProperty` 於是會設定為 "true"（儲存於 `UserState` 內），我們的程式碼現在則會處理這位使用者來自所有其他訊息活動的輸入。

WelcomeUserBot.cs [!code-csharpDidBotWelcomeUser]

處理其他輸入

在歡迎新的使用者後，便會在每個訊息回合評估使用者輸入資訊，且 Bot 會根據該使用者輸入的內容來提供回應。下列程式碼顯示用來產生該回應的決策邏輯。

- C#
- JavaScript

輸入 'intro' 或 'help' 會呼叫 `SendIntroCardAsync` 函式來向使用者呈現資訊主圖卡片。本文的下一節會檢驗該程式碼。

WelcomeUserBot.cs [!code-csharpSwitchOnUtterance]

使用主圖卡片問候語

如先前所述，某些使用者輸入會產生「主圖卡片」以回應其要求。您可以在[傳送簡介卡片](#)中深入了解主圖卡片問候語。必須有下列程式碼才能建立此 Bot 的主圖卡片回應。

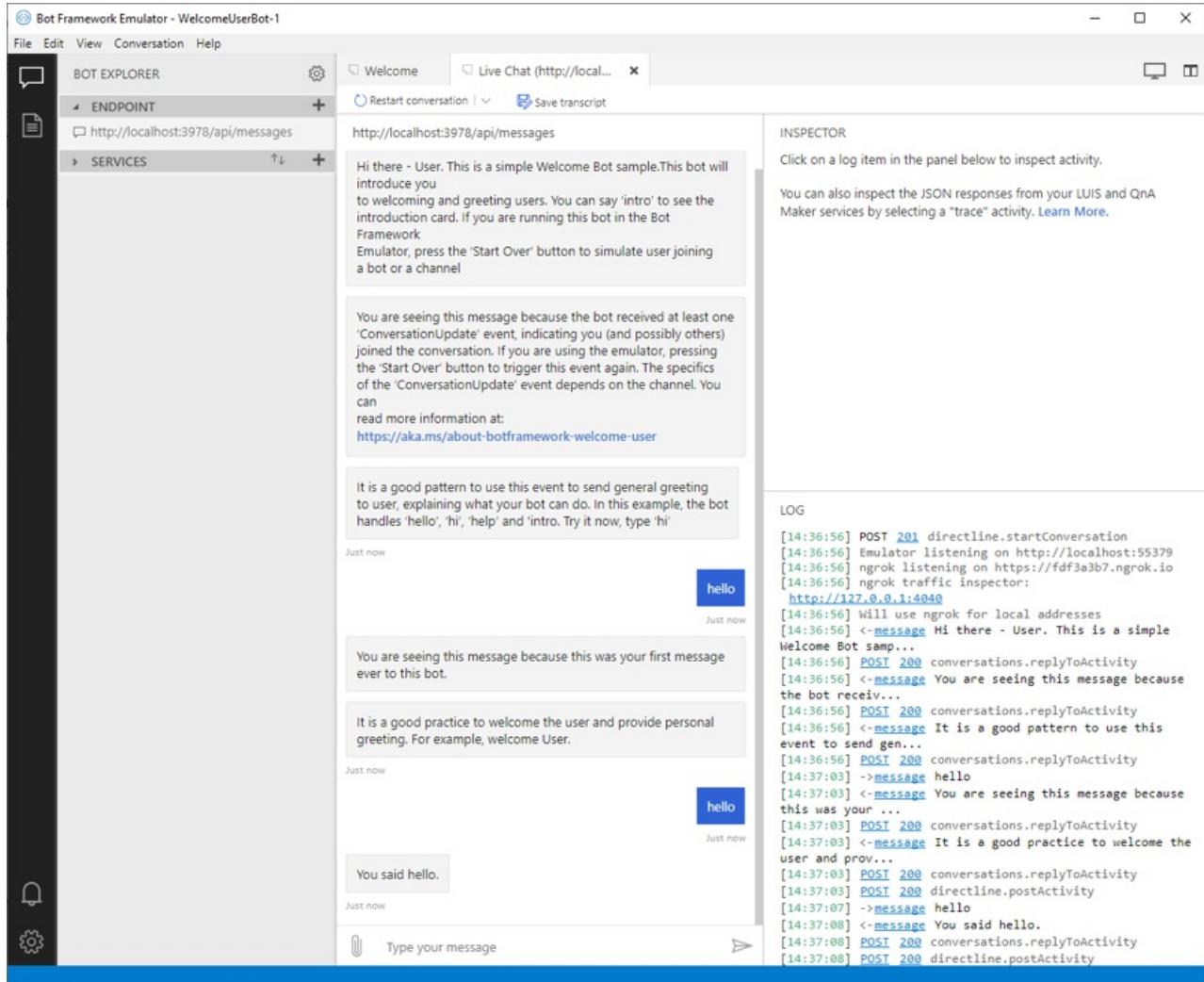
- C#
- JavaScript

WelcomeUserBot.cs [!code-csharpSendHeroCardGreeting]

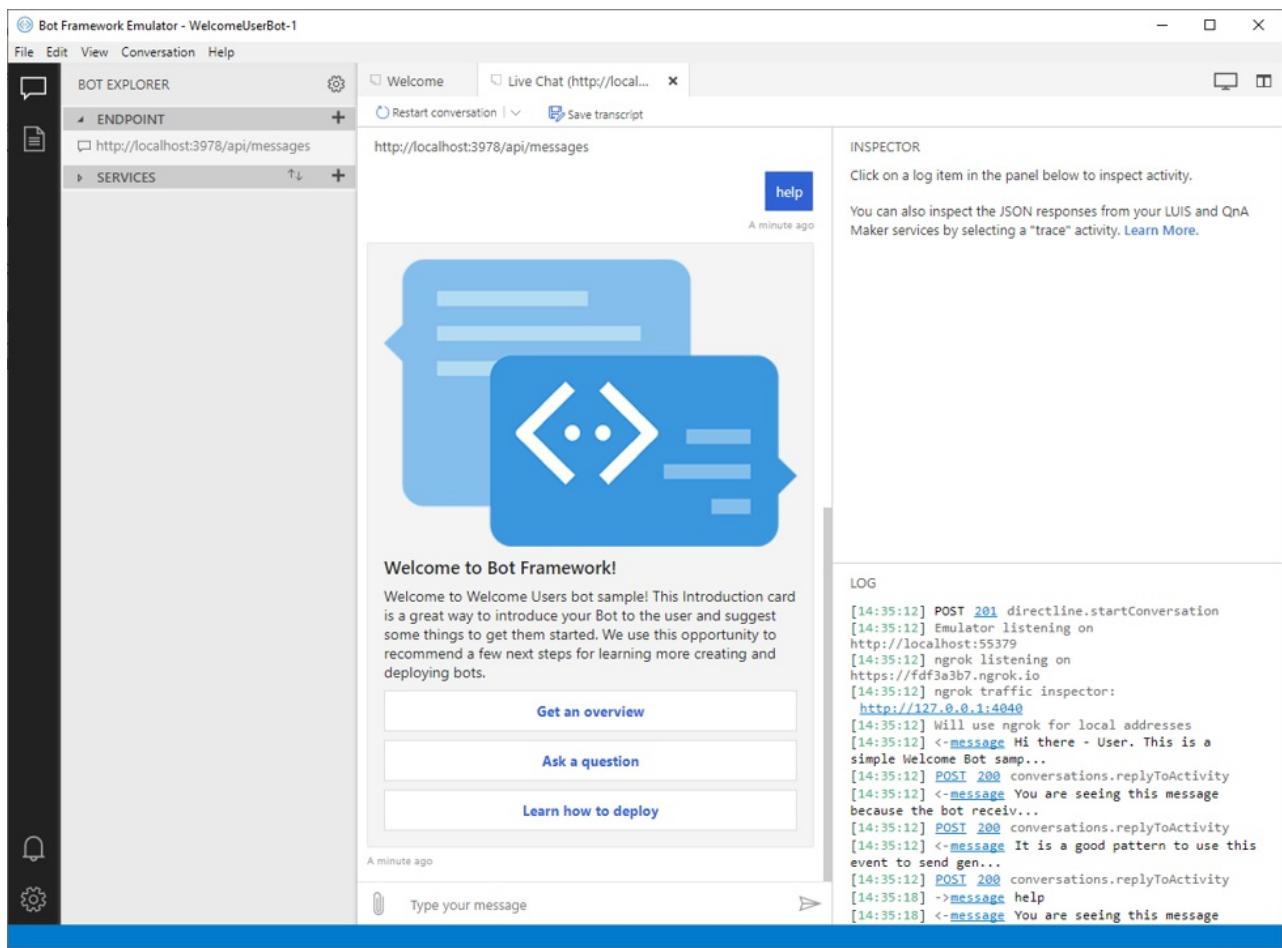
測試 Bot

下載並安裝最新版 Bot Framework Emulator

1. 在您的電腦本機執行範例。如需相關指示，請參閱 [C# 範例或 JS 範例](#)的讀我檔案。
2. 使用模擬器來測試 Bot，如下所示。



測試主圖卡片問候語。



其他資源

在[將媒體新增至訊息](#)中深入了解各種媒體回應。

後續步驟

[收集使用者輸入](#)

將主動式通知傳送給使用者

2019/5/10 • [Edit Online](#)

適用於： SDK v4 SDK v3

一般而言，Bot 直接傳送給使用者的每個訊息都與使用者先前的輸入相關。在某些情況下，Bot 可能需要將與目前對話主題或使用者最後傳送的訊息不直接相關的訊息傳送給使用者。這些類型的訊息稱為_主動訊息_。

主動訊息可用於各種情況。比方說，如果使用者先前已要求 Bot 監視產品的價格，則 Bot 可以在產品的價格下降了 20% 時對使用者發出警報。或者，如果 Bot 需要一些時間來編譯對使用者問題的回應，它可能會通知使用者已延遲，並且在此同時允許交談繼續進行。當 Bot 完成問題回應的編譯時，它會與使用者分享該資訊。

在您的 Bot 中實作主動式訊息時，請勿在短時間內傳送數個主動式訊息。某些通道會強制限制 Bot 將訊息傳送給使用者的頻率，如果違反了這些限制，將會停用 Bot。

臨機操作主動式訊息是最簡單的主動式訊息類型。Bot 只會在每次觸發時將訊息插入對話，不會顧及使用者目前是否與 Bot 在其他對話主題中，也不會嘗試以任何方式變更對話。

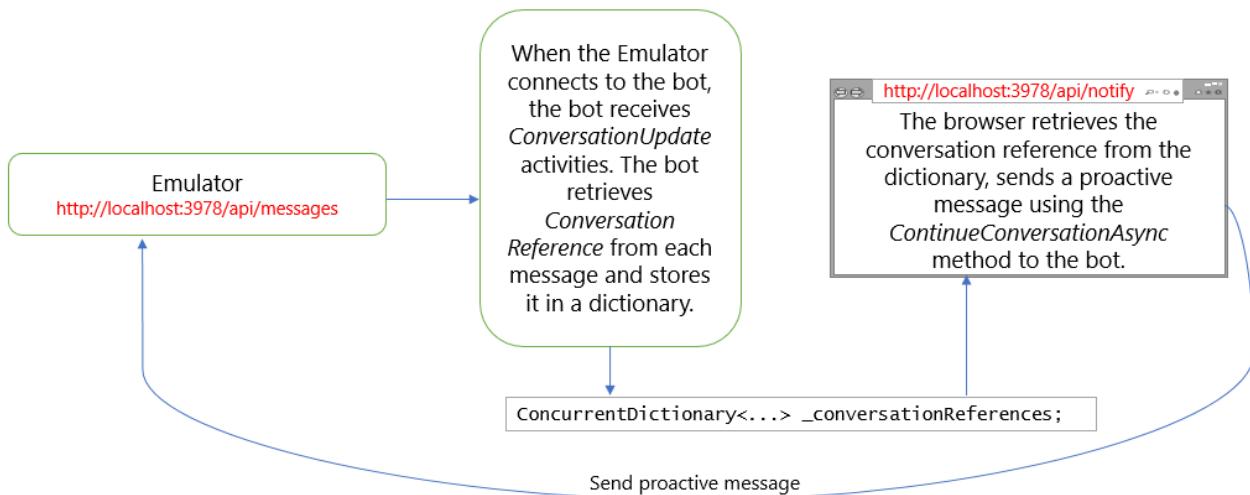
若要更順利地處理通知，請考慮使用其他方式將通知整合到對話流程中，例如在對話狀態中設定旗標，或將通知新增至佇列。

必要條件

- 了解 [bot 基本概念](#)。
- [CSharp](#) 或 [JavaScript](#) 中的主動訊息範例副本。這個範例用來說明本文中的主動式傳訊。

關於主動式範例

此範例有 Bot，以及用來將主動訊息傳送到 Bot 的額外控制器，如下圖所示。



擷取與儲存對話參考

當模擬器連線到 Bot 時，Bot 會收到兩個對話更新活動。在 Bot 的對話更新活動處理常式中，Bot 會擷取對話參考並將其儲存在字典中，如下所示。

- [C#](#)
- [JavaScript](#)

Bots\ProactiveBot.cs

```
private void AddConversationReference(Activity activity)
{
    var conversationReference = activity.GetConversationReference();
    _conversationReferences.AddOrUpdate(conversationReference.User.Id, conversationReference, (key, newValue)
=> conversationReference);
}

protected override Task OnConversationUpdateActivityAsync(ITurnContext<IConversationUpdateActivity>
turnContext, CancellationToken cancellationToken)
{
    AddConversationReference(turnContext.Activity as Activity);

    return base.OnConversationUpdateActivityAsync(turnContext, cancellationToken);
}
```

注意：在實際案例中，您會將對話參考保存在資料庫中，而不是使用記憶體中的物件。

對話參考具有_對話_屬性，用來描述其中有活動的對話。對話具有_使用者_屬性，該屬性會列出參與對話的使用者，對話還具有_服務 URL_屬性，通道會用此屬性來表示可能傳送目前活動回覆的 URL。傳送主動訊息給使用者需要有效的對話參考。

傳送主動訊息

第二個控制器（通知控制器）會負責將主動訊息傳送至 Bot。請使用下列步驟來產生主動訊息。

1. 摄取對話參考，以對其傳送主動訊息。
2. 呼叫接器的_繼續對話_方法，並提供要使用的對話參考和回合處理常式委派。繼續對話方法會為參考對話產生回合內容，然後呼叫指定的回合處理常式委派。
3. 在委派中，使用回合內容來傳送主動訊息。

- C#

- JavaScript

Controllers\NotifyController.cs

每當要求 Bot 的通知頁面時，通知控制器就會從字典中攝取對話參考。接著，控制器會使用

`ContinueConversationAsync` 和 `BotCallback` 方法來傳送主動訊息。

```

[Route("api/notify")]
[ApiController]
public class NotifyController : ControllerBase
{
    private readonly IBotFrameworkHttpAdapter _adapter;
    private readonly string _appId;
    private readonly ConcurrentDictionary<string, ConversationReference> _conversationReferences;

    public NotifyController(IBotFrameworkHttpAdapter adapter, ICredentialProvider credentials,
    ConcurrentDictionary<string, ConversationReference> conversationReferences)
    {
        _adapter = adapter;
        _conversationReferences = conversationReferences;
        _appId = ((SimpleCredentialProvider)credentials).AppId;

        // If the channel is the Emulator, and authentication is not in use,
        // the AppId will be null. We generate a random AppId for this case only.
        // This is not required for production, since the AppId will have a value.
        if (string.IsNullOrEmpty(_appId))
        {
            _appId = Guid.NewGuid().ToString(); //if no AppId, use a random Guid
        }
    }

    public async Task<IActionResult> Get()
    {
        foreach (var conversationReference in _conversationReferences.Values)
        {
            await ((BotAdapter)_adapter).ContinueConversationAsync(_appId, conversationReference, BotCallback,
default(CancellationToken));
        }

        // Let the caller know proactive messages have been sent
        return new ContentResult()
        {
            Content = "<html><body><h1>Proactive messages have been sent.</h1></body></html>",
            ContentType = "text/html",
            StatusCode = (int) HttpStatusCode.OK,
        };
    }

    private async Task BotCallback(ITurnContext turnContext, CancellationToken cancellationToken)
    {
        await turnContext.SendActivityAsync("proactive hello");
    }
}

```

若要傳送主動訊息，配接器會需要 Bot 的應用程式識別碼。在生產環境中，您可以使用 Bot 的應用程式識別碼。在本機測試環境中，您可以使用任何 GUID。如果 Bot 目前尚未獲派應用程式識別碼，通知控制器會自行產生用於呼叫的預留位置識別碼。

測試 Bot

- 如果您尚未安裝 [Bot Framework Emulator](#)，請進行安裝。
- 在您的電腦本機執行範例。
- 啟動模擬器並連線至您的 Bot。
- 載入至您 Bot 的 api/notify 頁面。這會在模擬器中產生主動訊息。

其他資訊

除了本文使用的範例之外，您也可以在 [GitHub](#) 上取得 C# 和 JS 的其他範例。

避免 401「未經授權」錯誤

根據預設，如果傳入的要求已經過 BotAuthentication 的驗證，則 BotBuilder SDK 會將 `serviceUrl` 加入受信任的主機名稱清單。這些名稱會保留在記憶體內部快取中。如果 Bot 重新啟動，正在等待主動訊息的使用者會無法接收該訊息，必須在 Bot 重新啟動後，再次對 Bot 傳送訊息才能解決此問題。

若要避免這個問題，您必須以手動方式將 `serviceUrl` 加入受信任的主機名稱清單，方式如下：

- C#
- JavaScript

```
MicrosoftAppCredentials.TrustServiceUrl(serviceUrl);
```

針對主動式傳訊，`serviceUrl` 是主動訊息接收者使用的通道 URL，您可以在 `Activity.ServiceUrl` 中找到此項目。

您可以直接將上述程式碼加在傳送主動訊息的程式碼之前。此範例將其放在 `ProactiveBot.cs` 中 `CreateCallback()` 的結尾附近，但已標記為註解，因為若沒有 `appId` 和 `appPassword`，其無法在模擬器中運作。

後續步驟

[實作循序對話流程](#)

實作循序對話流程

2019/5/14 • [Edit Online](#)

適用於:  SDK v4  SDK v3

張貼問題來收集資訊是 Bot 與使用者互動時的其中一種主要方式。dialogs 程式庫可讓您輕鬆地詢問問題，以及驗證回應以確保回應符合特定資料類型或符合自訂驗證規則。

您可以使用對話方塊程式庫來管理簡單和複雜的對話流程。在簡單互動中，Bot 透過一連串固定的步驟執行，然後完成對話。一般而言，當 Bot 需要向使用者蒐集資訊時，對話就很實用。本主題詳細說明如何建立提示並從瀑布對話進行呼叫，以實作簡單的交談流程。

必要條件

- [Bot 基本概念、管理狀態和 dialogs 程式庫](#) 的知識。
- [CSharp](#) 或 [JavaScript](#) 中的一份多回合提示範例。

關於此範例

在多回合提示範例中，我們會使用瀑布式對話、一些提示和元件對話來建立簡單互動，以詢問使用者一系列的問題。程式碼會使用對話來循環下列步驟：

步驟	提示類型
要求使用者的運輸模式	選擇提示
要求使用者的名稱	文字提示
詢問使用者是否要提供年齡	確認提示
如果他們回答 [是]，則要求年齡	包含驗證的數字提示，只接受大於 0 且小於 150 的年齡。
詢問所收集的資訊是否「合適」	重複使用確認提示

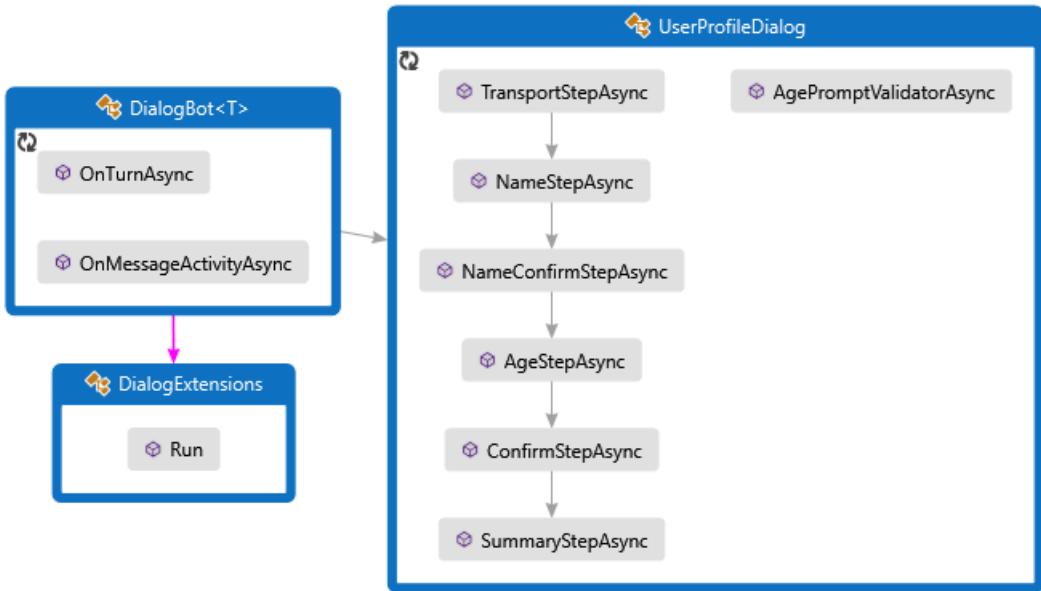
最後，如果他們回答 [是]，則顯示所收集的資訊；否則，告訴使用者將不會保留其資訊。

建立主要對話

- [C#](#)
- [JavaScript](#)

若要使用對話，請安裝 [Microsoft.Bot.Builder.Dialogs](#) NuGet 套件。

Bot 會透過 `UserProfileDialog` 與使用者互動。當我們建立 Bot 的 `DialogBot` 類別時，我們會設定 `UserProfileDialog` 作為其主要對話。Bot 接著會使用 `Run` 協助程式方法來存取此對話。



Dialogs\UserProfileDialog.cs

我們首先會建立 `UserProfileDialog`，其衍生自 `ComponentDialog` 類別並具有 6 個步驟。

在 `UserProfileDialog` 建構函式中，建立瀑布式步驟、提示和瀑布式對話，然後將其新增至對話集。提示必須位於其使用所在的相同對話集中。

```

// This array defines how the Waterfall will execute.
var waterfallSteps = new WaterfallStep[]
{
    TransportStepAsync,
    NameStepAsync,
    NameConfirmStepAsync,
    AgeStepAsync,
    ConfirmStepAsync,
    SummaryStepAsync,
};

// Add named dialogs to the DialogSet. These names are saved in the dialog state.
AddDialog(new WaterfallDialog(nameof(WaterfallDialog), waterfallSteps));
AddDialog(new TextPrompt(nameof(TextPrompt)));
AddDialog(new NumberPrompt<int>(nameof(NumberPrompt<int>), AgePromptValidatorAsync));
AddDialog(new ChoicePrompt(nameof(ChoicePrompt)));
AddDialog(new ConfirmPrompt(nameof(ConfirmPrompt)));

// The initial child Dialog to run.
InitialDialogId = nameof(WaterfallDialog);

```

接下來，我們會實作對話使用的步驟。若要使用提示，請從您對話中的步驟中呼叫，並在下列步驟中使用 `stepContext.Result` 擷取提示結果。在幕後，提示為兩個步驟的對話方塊。第一步，提示會要求輸入；第二步，其會傳回有效值，或利用重新提示從頭開始，直到其收到有效輸入為止。

您應該一律從瀑布式步驟傳回非 Null 的 `DialogTurnResult`。如果不這麼做，您的對話可能無法依照設計方式運作。我們在此示範如何在瀑布式對話中實作 `NameStepAsync`。

```

private static async Task<DialogTurnResult> NameStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    stepContext.Values["transport"] = ((FoundChoice)stepContext.Result).Value;

    return await stepContext.PromptAsync(nameof(TextPrompt), new PromptOptions { Prompt =
MessageFactory.Text("Please enter your name.") }, cancellationToken);
}

```

在 `AgeStepAsync` 中，我們會在使用者的輸入驗證失敗時，指定重試提示，而驗證失敗是因為提示無法剖析其格式，或輸入不符合驗證準則。在此情況下，如果未提供任何重試提示，則提示會使用初始提示文字來重新提示使用者提供輸入。

```

private async Task<DialogTurnResult> AgeStepAsync(WaterfallStepContext stepContext, CancellationToken
cancellationToken)
{
    if ((bool)stepContext.Result)
    {
        // User said "yes" so we will be prompting for the age.
        // WaterfallStep always finishes with the end of the Waterfall or with another dialog, here it is
        a Prompt Dialog.
        var promptOptions = new PromptOptions
        {
            Prompt = MessageFactory.Text("Please enter your age."),
            RetryPrompt = MessageFactory.Text("The value entered must be greater than 0 and less than
150."),
        };

        return await stepContext.PromptAsync(nameof(NumberPrompt<int>), promptOptions, cancellationToken);
    }
    else
    {
        // User said "no" so we will skip the next step. Give -1 as the age.
        return await stepContext.NextAsync(-1, cancellationToken);
    }
}

```

UserProfile.cs

使用者的運輸方式、名稱和年齡都會儲存在 `UserProfile` 類別的執行個體中。

```

public class UserProfile
{
    public string Transport { get; set; }

    public string Name { get; set; }

    public int Age { get; set; }
}

```

Dialogs\UserProfileDialog.cs

在最後一個步驟中，我們會檢查由前一個瀑布式步驟中呼叫的對話所傳回的 `stepContext.Result`。如果傳回的值為 true，我們會使用使用者設定檔存取子來取得及更新使用者設定檔。若要取得使用者設定檔，我們會呼叫 `GetAsync` 方法，然後設定 `userProfile.Transport`、`userProfile.Name` 和 `userProfile.Age` 屬性的值。最後，我們會先摘要說明使用者資訊，再呼叫 `EndDialogAsync` 來結束對話。結束對話就會將其從對話堆疊中取出，並將選擇性結果傳回給對話的父代。父代是開始剛結束之對話的對話或方法。

```

private async Task<DialogTurnResult> SummaryStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    if ((bool)stepContext.Result)
    {
        // Get the current profile object from user state.
        var userProfile = await _userProfileAccessor.GetAsync(stepContext.Context, () => new UserProfile(), cancellationToken);

        userProfile.Transport = (string)stepContext.Values["transport"];
        userProfile.Name = (string)stepContext.Values["name"];
        userProfile.Age = (int)stepContext.Values["age"];

        var msg = $"I have your mode of transport as {userProfile.Transport} and your name as {userProfile.Name}.";
        if (userProfile.Age != -1)
        {
            msg += $" And age as {userProfile.Age}.";
        }

        await stepContext.Context.SendActivityAsync(MessageFactory.Text(msg), cancellationToken);
    }
    else
    {
        await stepContext.Context.SendActivityAsync(MessageFactory.Text("Thanks. Your profile will not be kept."), cancellationToken);
    }

    // WaterfallStep always finishes with the end of the Waterfall or with another dialog, here it is the end.
    return await stepContext.EndDialogAsync(cancellationToken: cancellationToken);
}

```

建立擴充方法來執行瀑布式對話

- C#
- JavaScript

我們已定義 `Run` 擴充方法，其將用於建立及存取對話內容。在此，`accessor` 是對話狀態屬性的狀態屬性存取子，而 `dialog` 是使用者設定檔元件對話。由於元件對話會定義內部對話集，因此我們必須建立可讓訊息處理常式程式碼看見的外部對話集，並使用其建立對話內容。

對話內容是經由呼叫 `createContext` 方法來建立，且用於在 Bot 的回合處理常式內與對話集互動。對話內容包含目前的回合內容、父代對話，以及對話狀態，可供保留對話中的資訊。

對話內容可讓您開始一個具有字串識別碼的對話，或繼續目前的對話（例如具有多個步驟的瀑布式對話）。對話內容會傳遞至 Bot 的對話和瀑布式步驟。

DialogExtensions.cs

```

public static async Task Run(this Dialog dialog, ITurnContext turnContext,
IStatePropertyAccessor<DialogState> accessor, CancellationToken cancellationToken =
default(CancellationToken))
{
    var dialogSet = new DialogSet(accessor);
    dialogSet.Add(dialog);

    var dialogContext = await dialogSet.CreateContextAsync(turnContext, cancellationToken);
    var results = await dialogContext.ContinueDialogAsync(cancellationToken);
    if (results.Status == DialogTurnStatus.Empty)
    {
        await dialogContext.BeginDialogAsync(dialog.Id, null, cancellationToken);
    }
}

```

執行對話

- C#
- JavaScript

Bots\DialogBot.cs

`OnMessageActivityAsync` 處理常式會使用擴充方法來開始或繼續對話。在 `OnTurnAsync` 中，我們會使用 Bot 的狀態管理物件將任何狀態變更保存到儲存體。`(ActivityHandler.OnTurnAsync)` 方法會呼叫各種活動處理常式方法，例如 `OnMessageActivityAsync`。如此一來，我們會在訊息處理常式完成後，但在回合本身完成之前儲存狀態。)

```

public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken =
default(CancellationToken))
{
    await base.OnTurnAsync(turnContext, cancellationToken);

    // Save any state changes that might have occurred during the turn.
    await ConversationState.SaveChangesAsync(turnContext, false, cancellationToken);
    await UserState.SaveChangesAsync(turnContext, false, cancellationToken);
}

protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    Logger.LogInformation("Running dialog with Message Activity.");

    // Run the Dialog with the new message Activity.
    await Dialog.Run(turnContext, ConversationState.CreateProperty<DialogState>(nameof(DialogState)),
cancellationToken);
}

```

為 Bot 註冊服務

此 Bot 會使用下列_服務_。

- Bot 的基本服務：認證提供者、配接器及 Bot 實作。
- 用於管理狀態的服務：儲存體、使用者狀態及交談狀態。
- Bot 會使用的對話。
- C#
- JavaScript

Startup.cs

我們會在 `Startup` 中為 Bot 註冊服務。這些服務可透過相依性插入來提供給程式碼的其他部分。

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    // Create the credential provider to be used with the Bot Framework Adapter.
    services.AddSingleton<ICredentialProvider, ConfigurationCredentialProvider>();

    // Create the Bot Framework Adapter with error handling enabled.
    services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithErrorHandler>();

    // Create the storage we'll be using for User and Conversation state. (Memory is great for testing purposes.)
    services.AddSingleton<IStorage, MemoryStorage>();

    // Create the User state. (Used in this bot's Dialog implementation.)
    services.AddSingleton<UserState>();

    // Create the Conversation state. (Used by the Dialog system itself.)
    services.AddSingleton<ConversationState>();

    // The Dialog that will be run by the bot.
    services.AddSingleton<UserProfileDialog>();

    // Create the bot as a transient. In this case the ASP Controller is expecting an IBot.
    services.AddTransient<IBot, DialogBot<UserProfileDialog>>();
}
```

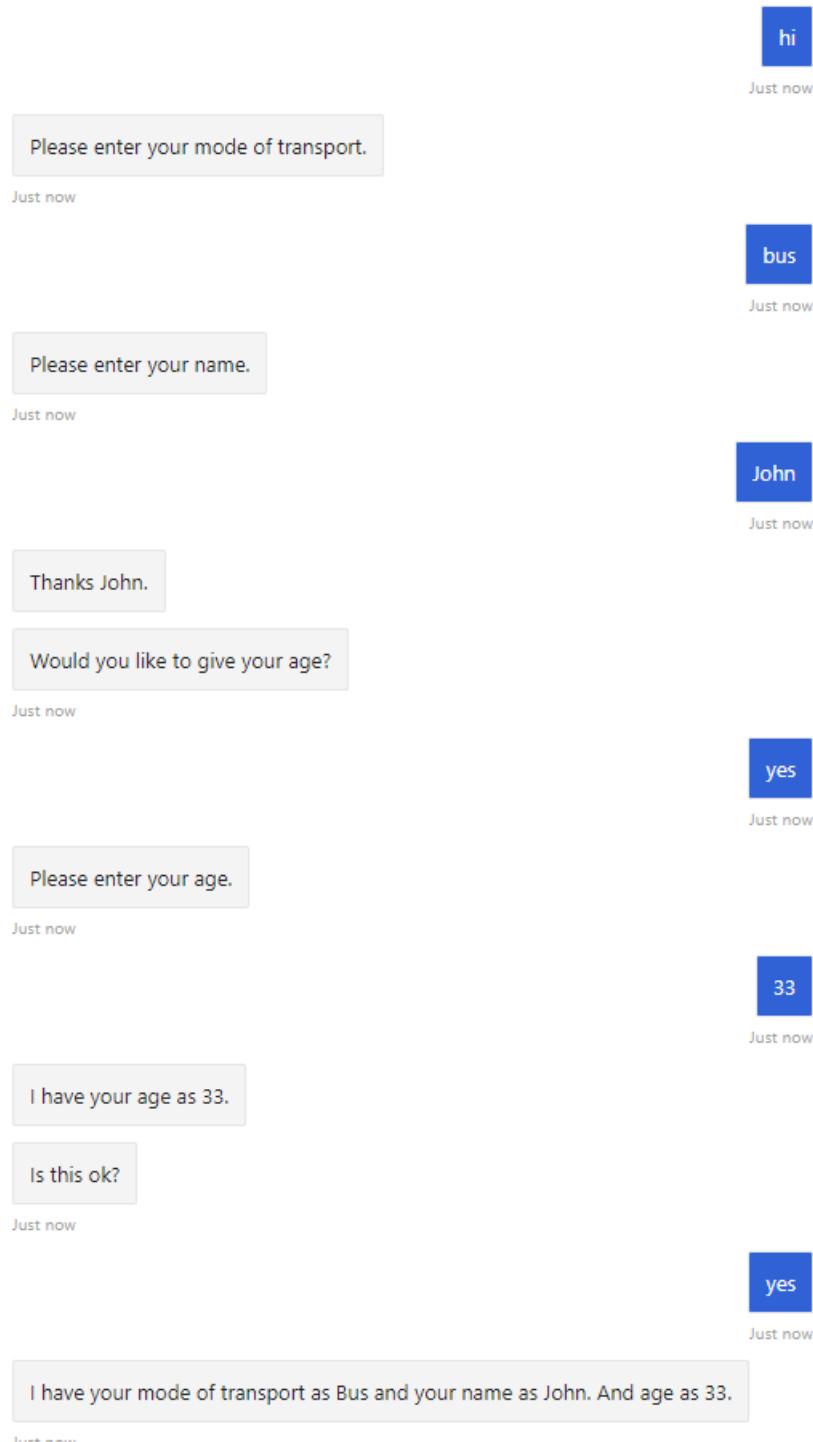
NOTE

記憶體儲存體僅供測試用途，而不適用於生產環境。針對生產 Bot，請務必使用永續性的儲存體類型。

測試 Bot

1. 如果您尚未安裝 [Bot Framework Emulator](#)，請進行安裝。
2. 在您的電腦本機執行範例。
3. 啟動模擬器、連線到您的 Bot 並傳送如下所示的訊息。

<http://localhost:3978/api/messages>



其他資訊

關於對話方塊和 Bot 狀態

在此 Bot 中，我們已定義兩個狀態屬性存取子：

- 一個建立於對話狀態屬性的對話方塊狀態內。對話方塊狀態會追蹤使用者在對話方塊集中對話方塊的位置，並由對話方塊內容更新，例如當我們呼叫開始對話方塊或繼續對話方塊方法時。
- 一個建立於使用者設定檔屬性的使用者狀態內。Bot 會使用此項來追蹤其使用者的相關資訊，而我們會在對話程式碼中明確地管理此狀態。

狀態屬性存取子的 `get` 和 `set` 方法，會取得及設定狀態管理物件的快取中的屬性值。快取會第一次填入回合中要求的狀態屬性值，但必須明確地保存。為了保存這兩個狀態屬性的變更，我們會呼叫對應狀態管理物件的「儲存變更」方法。

此範例會在對話方塊內更新使用者設定檔狀態。這種做法適用於簡單的 Bot，但如果您想要在所有 Bot 中重複使用一個對話方塊，則不可行。

有各種選項可將對話方塊步驟和 Bot 狀態分開。例如，一旦對話方塊蒐集完整資訊，您即可：

- 使用結束對話方法，提供所收集的資料作為父代內容的傳回值。這可以是 Bot 的回合處理常式，或對話方塊堆疊上先前作用中的對話方塊。這就是提示類別的設計方式。
- 產生適當服務的要求。如果 Bot 作為大型服務的前端，這可能效果良好。

後續步驟

[將自然語言理解新增至您的 Bot](#)

將自然語言理解新增至您的 Bot

2019/5/14 • [Edit Online](#)

適用於:  SDK v4  SDK v3

讓您的 Bot 能透過對話和上下文了解使用者的意思，並不是簡單的工作，但這樣的功能可讓您的 Bot 使用起來更有自然對話的感覺。Language Understanding (稱為 LUIS) 能讓您這麼做，使您的 Bot 可以辨識使用者訊息的意圖，這樣使用者就能使用更自然的語言，且更順利地引導交談流程。本主題會逐步引導您將 LUIS 新增至航班預訂應用程式，以辨識使用者輸入內含的不同意圖和實體。

必要條件

- [LUIS 帳戶](#)
- 本文中的程式碼是以 **Core Bot** 範例為基礎。您需要 [CSharp](#) 或 [JavaScript](#) 中的一份範例。
- [Bot 基本概念](#)、[自然語言處理](#) 和 [管理 Bot 資源](#) 的知識。

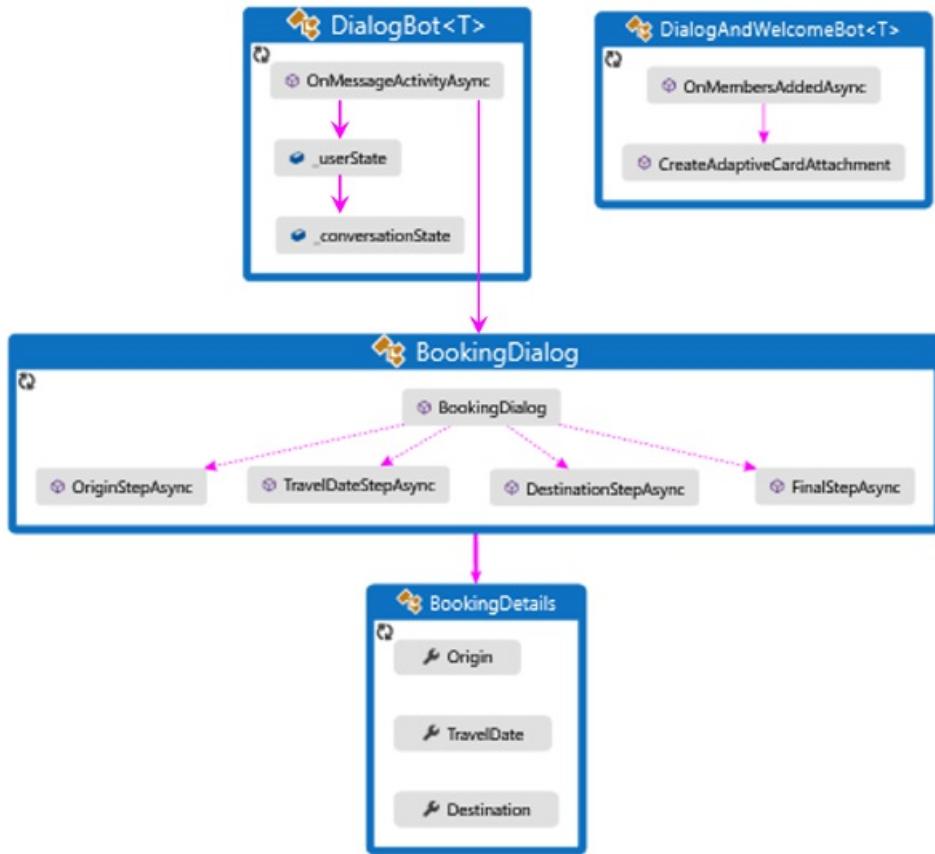
關於此範例

此 Core Bot 編碼範例顯示機場航班預訂應用程式的範例。其會使用 LUIS 服務來辨識使用者輸入，並傳回最常辨識的 LUIS 意圖。

- [C#](#)
- [JavaScript](#)

在每此處理使用者輸入之後，`DialogBot` 會儲存 `UserState` 和 `ConversationState` 的目前狀態。蒐集所有必要的資訊後，程式碼範例會建立示範航班預訂保留。這本文中，我們將探討這的範例的 LUIS 層面。不過，此範例的一般流程如下所示：

- 當新的使用者連線並顯示歡迎卡片時，就會呼叫 `OnMembersAddedAsync`。
- 系統會針對每個收到的使用者輸入呼叫 `OnMessageActivityAsync`。



`OnMessageActivityAsync` 模組會透過 `Run` 對話擴充方法執行適當的對話。該主要對話會呼叫 LUIS 協助程式，以尋找評分最高的使用者意圖。如果使用者輸入的最高意圖傳回 "Book_Flight"，則協助程式會填寫 LUIS 傳回的使用者資訊，並啟動 `BookingDialog`，它會視需要向使用者取得其他資訊，例如：

- `Origin` 原始城市。
- `TravelDate` 要預訂航班的日期。
- `Destination` 目的地城市。

如需範例的其他層面（例如對話或狀態）詳細資訊，請參閱[使用對話提示蒐集使用者輸入或儲存使用者和交談資料](#)。

在 LUIS 入口網站中建立 LUIS 應用程式

登入 LUIS 入口網站以建立自己的範例 LUIS 應用程式版本。您可以在 [我的應用程式] 上建立和管理應用程式。

1. 選取 [匯入新的應用程式]。
2. 按一下 [選擇應用程式檔案 (JSON 格式)...]
3. 選取 `FlightBooking.json` 檔案，該檔案位於範例的 `CognitiveModels` 資料夾中。在 [選擇性名稱] 中，輸入 **FlightBooking**。此檔案包含三個意圖：[預訂航班]、[取消] 和 [無]。我們將使用這些意圖，了解使用者將訊息傳送給 Bot 時的用意。
4. [訓練](#)應用程式。
5. 將應用程式[發佈](#)到「生產」環境。

為何使用實體

LUIS 實體可讓 Bot 以智慧方式了解與標準意圖不同的特定事項或事件。這可讓您向使用者收集額外資訊，進而讓 Bot 更聰明地回應，或可能略過要求使用者提供該資訊的某些問題。除了三個 LUIS 意圖 [預訂航班]、[取消] 和 [無] 的定義，`FlightBooking.json` 檔案還包含一組實體，例如 '`From.Airport`' 和 '`To.Airport`'。這些實體可讓 LUIS 偵測使用者原始輸入內含的其他資訊，並且在使用者要求新的旅行預約時傳回這些資訊。

取得值以連線到您的 LUIS 應用程式

在您的 LUIS 應用程式發佈之後，您即可從 Bot 進行存取。您必須記錄數個值，以從 Bot 存取您的 LUIS 應用程式。您可以使用 LUIS 入口網站擷取該資訊。

從 LUIS.ai 入口網站擷取應用程式資訊

settings 檔案 (`appsettings.json` 或 `.env`) 檔案可作為將所有服務參考匯合在一起的位置。您所擷取的資訊會新增至下一節中的這個檔案。

1. 從 [luis.ai](#) 中選取已發佈的 LUIS 應用程式。
2. 開啟已發佈的 LUIS 應用程式後，選取 [管理] 索引標籤。

The screenshot shows the LUIS.ai application management interface. At the top, there's a navigation bar with links for Language Understanding, My apps, Docs, Pricing, Support, and About. Below that is a secondary navigation bar for the 'FlightBooking (v 0.1)' application, with tabs for DASHBOARD, BUILD, MANAGE (which is highlighted with a red box), Train, Test, and Publish. On the left, a sidebar menu is open under 'Application Settings', showing options like Application Information (which is highlighted with a blue box), Keys and Endpoints (which is highlighted with a red box), Publish Settings, Versions, and Collaborators. The main content area is titled 'Application Information' and contains fields for Application ID (00297793-0c85-xxxx-xxxx-e9da56dcf0c8), Display Name (Required) (FlightBooking), and Description (A LUIS model that uses intent and entities). There's also a 'Train' button at the bottom of the main content area.

3. 選取左側的 [應用程式資訊] 索引標籤，將針對 [應用程式識別碼] 顯示的值記錄為 <YOUR_APP_ID>。
4. 選取左側 [金鑰和端點] 索引標籤，將針對 [撰寫金鑰] 所顯示的值記錄為 <YOUR_AUTHORIZING_KEY>。
5. 向下捲動到頁面結尾，將針對 [地區] 顯示的值記錄為 <YOUR_REGION>。

更新 settings 檔案

- [C#](#)
- [JavaScript](#)

將存取 LUIS 應用程式所需的資訊（包括應用程式識別碼、撰寫金鑰和區域）新增至 `appsettings.json` 檔案中。這些是您先前從已發佈的 LUIS 應用程式儲存的值。請注意，API 主機名稱應該採用 `<your region>.api.cognitive.microsoft.com` 格式。

`appsetting.json` [[!code-jsonappsettings](#)]

設定 Bot 來使用 LUIS 應用程式

- [C#](#)
- [JavaScript](#)

請確定您已為專案安裝 **Microsoft.Bot.Builder.AI.Luis** NuGet 套件。

若要連線到 LUIS 服務，Bot 會從 `appsetting.json` 檔案提取您前面新增的資訊。`LuisHelper` 類別包含的程式碼可從 `appsetting.json` 檔案匯入您的設定，以及藉由呼叫 `RecognizeAsync` 方法來查詢 LUIS 服務。如果傳回的最高意圖是 'Book_Flight'，其會檢查包含預訂 To、From 和 TravelDate 資訊等實體。

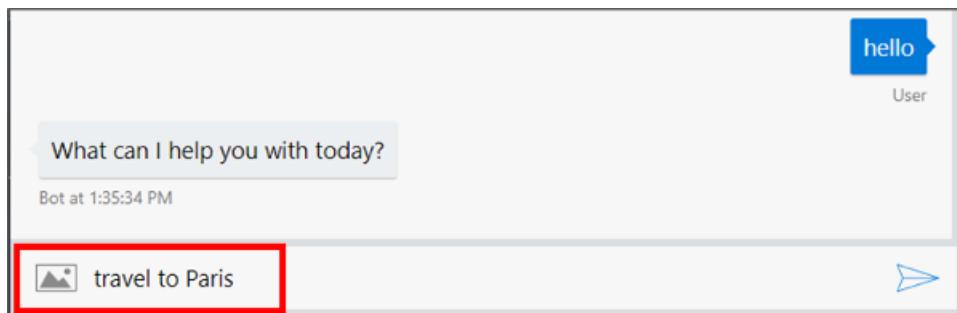
`LuisHelper.cs` [[!code-csharp!luis helper](#)]

現在已針對您的 Bot 設定和連線 LUIS。

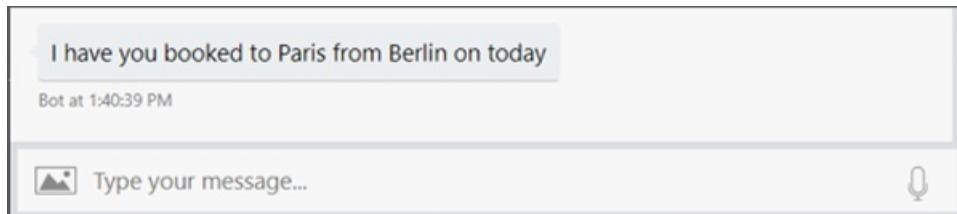
測試 Bot

下載並安裝最新版 [Bot Framework Emulator](#)

1. 在您的電腦本機執行範例。如需相關指示，請參閱 [C# 範例](#) 或 [JS 範例](#) 的讀我檔案。
2. 在模擬器中，輸入訊息，例如「到巴黎旅行」或「從巴黎到柏林」。使用在 FlightBooking.json 檔案中找到的任何語句來訓練「預訂航班」意圖。



如果 LUIS 傳回的最高意圖解析為「預訂航班」，您的 Bot 會詢問其他問題，直到儲存足夠的資訊來建立旅行預訂為止。屆時，會將此預訂資訊傳回給您的使用者。



此時，程式碼 Bot 邏輯會重設，而您可以繼續建立其他預訂。

後續步驟

[使用 QnA Maker 回答問題](#)

使用 QnA Maker 回答問題

2019/5/10 • [Edit Online](#)

適用於：  SDK v4  SDK v3

QnA Maker 會透過您的資料提供對話式的問題和解答層。這可讓您的 Bot 傳送問題給 QnA Maker 並接收回應，而不需要您來剖析和解譯問題的意圖。

在建立您自己的 QnA Maker 服務時，其中一項基本需求是在服務中植入問題和解答。在許多案例中，問題和解答已存在於常見問題集或其他文件的內容中；但有些時候，您可能會想要以更自然的對話方式來自訂問題的解答。

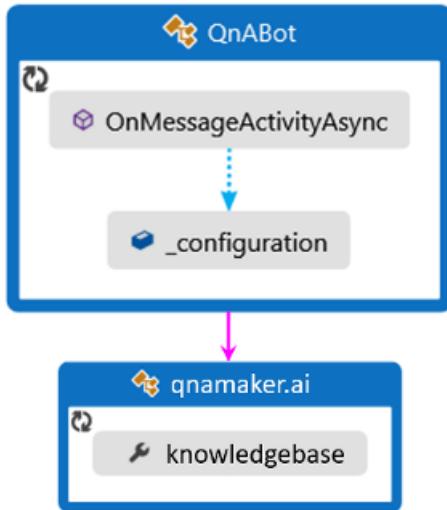
必要條件

- 本文中的程式碼是以 QnA Maker 範例為基礎。您需要從 [CSharp](#) 或 [JavaScript](#) 中取得其副本。
- [QnA Maker](#) 帳戶
- [Bot 基本概念](#)、[QnA Maker](#) 及 [管理 Bot 資源](#)的知識。

關於此範例

為了讓 Bot 可使用 QnA Maker，您必須先在 [QnA Maker](#) 上建立知識庫，我們將在下一節中討論這部分。然後，您的 Bot 可以對其傳送使用者的查詢，讓其在回應中提供問題的最佳答案。

- [C#](#)
- [JavaScript](#)



系統會針對每個收到的使用者輸入呼叫 `OnMessageActivityAsync`。呼叫後，該方法就會在範例程式碼的 `appsetting.json` 檔案內存取儲存的 `_configuration` 資訊，以尋找值來連線到預先設定的 QnA Maker 知識庫。

使用者輸入會傳送至您的知識庫，並向使用者顯示最適合的回傳解答。

建立 QnA Maker 服務及發佈知識庫

第一步是建立 QnA Maker 服務。依照 QnA Maker [文件](#) 中所述的步驟在 Azure 中建立服務。

接下來，您將使用 `smartLightFAQ.tsv` 檔案（位於範例專案的 CognitiveModels 資料夾中）來建立知識庫。用於建立、定型及發佈 QnA Maker [知識庫](#)的步驟會列於 QnA Maker 文件中。當您遵循這些步驟時，請將您的 KB 命名為 `qna`，並使用 `smartLightFAQ.tsv` 檔案來填入 KB。

注意：本文也可用來存取您自己使用者所開發的 QnA Maker 知識庫。

取得值，將您的 Bot 連線到知識庫

1. 在 [QnA Maker](#) 網站上，選取您的知識庫。
2. 開啟您的知識庫，然後選取 [設定]。記錄針對「服務名稱」顯示的值。使用 QnA Maker 入口網站介面時，此值很適合用於尋找您感興趣的知識庫。但不會用於將 Bot 應用程式連線到此知識庫。
3. 向下捲動以尋找 [部署詳細資料] 及記錄下列值：
 - POST /knowledgebases/<Your_Knowledge_Base_Id>/getAnswers
 - 主機: <Your_Hostname>/qnamaker
 - 授權: EndpointKey <Your_Endpoint_Key>

您主機名稱的完整 URL 字串看起來會類似 "https://<>.azure.net/qnamaker"。這三個值會為您的應用程式提供透過 Azure QnA 服務連線到 QnA Maker 知識庫所需的資訊。

更新設定檔

首先，將存取您知識庫所需的資訊（包括主機名稱、端點金鑰和知識庫識別碼 (kbId)）新增至設定檔。這些是您在 QnA Maker 中從知識庫 [設定] 索引標籤儲存的值。

如果您未將其部署於生產環境，可將應用程式識別碼和密碼欄位保留空白。

NOTE

如果您要將 QnA Maker 知識庫的存取權新增到現有 Bot 應用程式中，請務必為 QnA 項目新增具參考價值的標題。此區段內的 "name" 值會提供從您的應用程式存取此資訊所需的金鑰。

- [C#](#)
- [JavaScript](#)

更新 appsettings.json 檔案

```
{  
  "MicrosoftAppId": "",  
  "MicrosoftAppPassword": "",  
  
  "QnA-sample-qna-kbId": "<knowledge-base-id>",  
  "QnA-sample-qna-endpointKey": "<your-endpoint-key>",  
  "QnA-sample-qna-hostname": "<your-hostname>"  
}
```

設定 QnA Maker 執行個體

首先，我們會建立用來存取 QnA Maker 知識庫的物件。

- [C#](#)
- [JavaScript](#)

請確定您已為專案安裝 **Microsoft.Bot.Builder.AI.QnA** NuGet 套件。

在 **QnABot.cs** 的 `OnMessageActivityAsync` 方法中，我們會建立 QnAMaker 執行個體。`QnABot` 類別中也會加入連線資訊名稱（儲存在上述的 `appsettings.json` 中）。如果您已在設定檔中選擇不同的知識庫連線資訊名稱，請務必在此更新名稱，以反映您所選擇的名稱。

Bots/QnABot.cs [!code-csharpqna connection]

從您的 Bot 呼叫 QnA Maker

- [C#](#)
- [JavaScript](#)

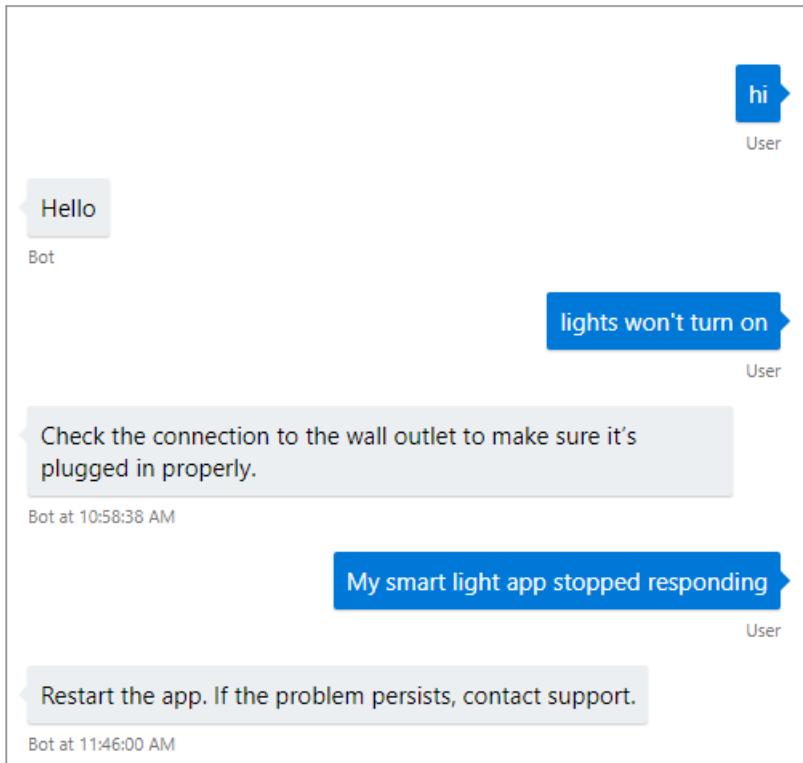
當您的 Bot 需要來自 QnAMaker 的解答時，從您的 Bot 程式碼呼叫 `GetAnswersAsync()`，以根據目前的內容取得適當解答。如果您要存取自己的知識庫，請變更以下的_找不到解答_訊息，為您的使用者提供實用的指示。

QnABot.cs [!code-csharpqna connection]

測試 Bot

在您的電腦本機執行範例。如果您尚未安裝 [Bot Framework Emulator](#)，請進行安裝。如需進一步指示，請參閱 [C#範例](#)或[Javascript 範例](#)的讀我檔案。

啟動模擬器、連線到您的 Bot 並傳送如下所示的訊息。



後續步驟

QnA Maker 可以結合其他認知服務，讓 Bot 具有更強大的功能。分派工具可讓您在 Bot 中結合 QnA 和 Language Understanding (LUIS)。

[使用分派工具結合 LUIS 和 QnA 服務](#)

使用多個 LUIS 和 QnA 模型

2019/5/10 • [Edit Online](#)

適用於： SDK v4 SDK v3

如果 Bot 使用多個 LUIS 模型和 QnA Maker 知識庫 (KB)，您可以使用分派工具來判斷哪一個 LUIS 模型或 QnA Maker KB 最符合使用者輸入。分派工具執行此操作的方式是建立單一 LUIS 應用程式，以將使用者輸入路由至正確的模型。如需有關分派工具的詳細資訊（包括 CLI 命令），請參閱[讀我檔案](#)。

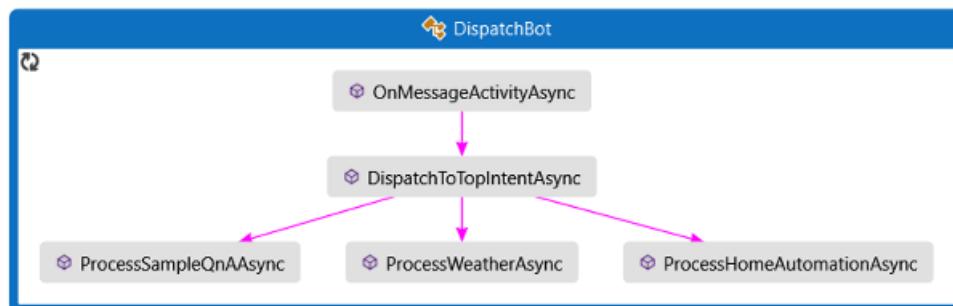
必要條件

- [Bot 基本概念](#)、[LUIS](#) 和 [QnA Maker](#) 的知識。
- [分派工具](#)
- 從 [C# 範例](#) 或 [JS 範例](#) 程式碼存放庫取得一份採用分派的 [NLP](#)。
- 用來發佈 LUIS 應用程式的 [luis.ai](#) 帳戶。
- 用來發佈 QnA 知識庫的 [QnA Maker](#) 帳戶。

關於此範例

此範例會以一組預先定義的 LUIS 和 QnA Maker 應用程式作為基礎。

- [C#](#)
- [JavaScript](#)



系統會針對每個收到的使用者輸入呼叫 `OnMessageActivityAsync`。此模組會尋找評分最高的使用者意圖，並將該結果傳遞至 `DispatchToTopIntentAsync`。接著，`DispatchToTopIntentAsync` 會呼叫適當的應用程式處理常式

- `ProcessSampleQnAAsync` - 適用於 Bot 常見問題集的問題。
- `ProcessWeatherAsync` - 適用於天氣查詢。
- `ProcessHomeAutomationAsync` - 適用於家用光源命令。

處理常式會呼叫 LUIS 或 QnA Maker 服務，並將產生的結果傳回給使用者。

建立 LUIS 應用程式和 QnA KB

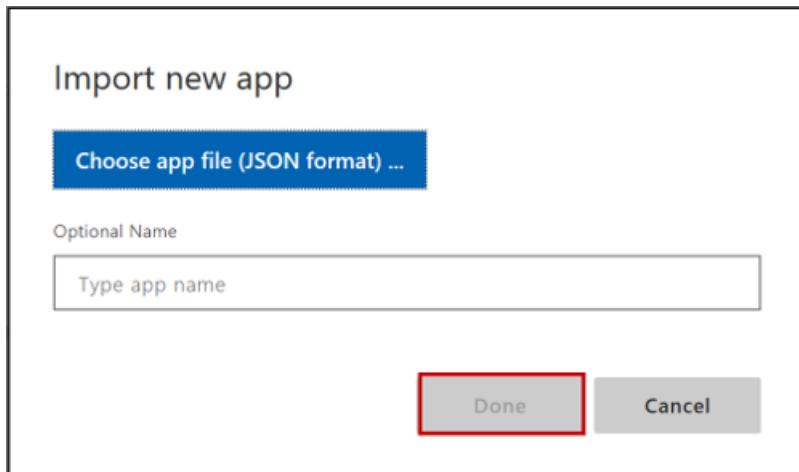
建立分派模型之前，您必須先建立並發佈 LUIS 應用程式和 QnA KB。在本文中，我們將發佈下列模型，這些模型包含在「採用分派的 NLP」範例的 `\CognitiveModels` 資料夾中：

NAME	說明

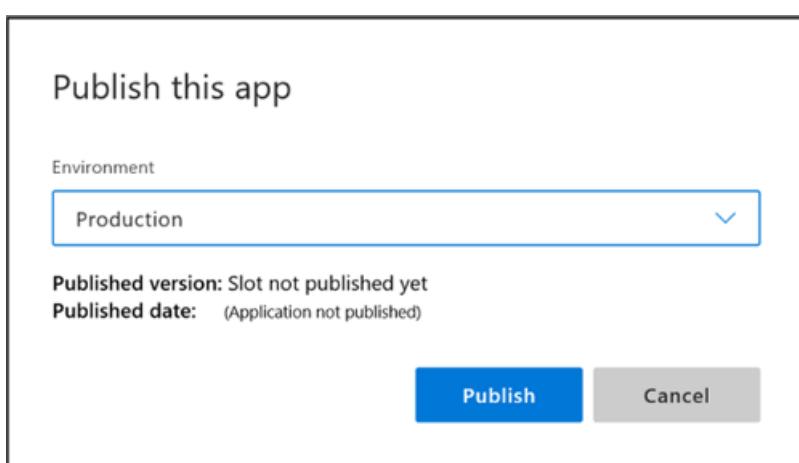
NAME	說明
HomeAutomation	使用相關聯的實體資料，辨識住家自動化意圖的 LUIS 應用程式。
Weather	使用位置資料辨識天氣相關意圖的 LUIS 應用程式。
QnAMaker	提供 Bot 相關簡單問題解答的 QnA Maker 知識庫。

建立 LUIS 應用程式

- 登入 [LUIS Web 入口網站](#)。在 [我的應用程式] 區段之下，選取 [匯入新的應用程式] 索引標籤。下列對話方塊隨即顯示：



- 選取 [選擇應用程式檔案] 按鈕，瀏覽至您範例程式碼的 CognitiveModel 資料夾，然後選取 'HomeAutomation.json' 檔案。將選用名稱欄位保留空白。
- 選取 [完成]。
- 在 LUIS 開啟您的住家自動化應用程式後，請選取 [訓練] 按鈕。這會使用您剛使用 'home-automation.json' 檔案匯入的語句集合，訓練應用程式。
- 訓練完成時，請選取 [發佈] 按鈕。下列對話方塊隨即顯示：



- 選擇 [生產] 環境，然後選取 [發佈] 按鈕。
- 開啟已發佈的新 LUIS 應用程式後，選取 [管理] 索引標籤。在 [應用程式資訊] 頁面中，將 Application ID 的值記錄為 "app-id-for-app"，並將 Display name 的值記錄為 "name-of-app"。在 [金鑰和端點] 頁面中，將 Authoring Key 的值記錄為 "your-luis-authoring-key"，並將 Region 記錄為 "your-region"。稍後在您的 'appsetting.json' 檔案中會用到這些值。

8. 完成後，針對 'Weather.json' 檔案重複上述步驟，以_訓練_和_發佈_您的 LUIS 氣象應用程式和 LUIS 分派應用程式。

建立 QnA Maker KB

設定 QnA Maker KB 的第一個步驟是在 Azure 中設定 QnA Maker 服務。若要這麼做，請遵循[這裡](#)的逐步指示。現在登入 [LUIS Web 入口網站](#)。向下移至步驟 2

STEP 2

Connect your QnA service to your KB.

After you create an Azure QnA service, [refresh this page](#) and then select your Azure service using the options below.

* Microsoft Azure Directory ID
Select tenant

* Azure subscription name
Select subscription

* Azure QnA service
Select service

及選取

1. 您的 Azure AD 帳戶。
2. 您的 Azure 訂用帳戶名稱。
3. 針對您的 QnA Maker 服務建立的名稱。(如果您的 Azure QnA 服務並未一開始就出現在此下拉式清單中，請嘗試重新整理頁面。)

移至步驟 3

STEP 3

Name your KB.

The knowledge base name is for your reference and you can change it at anytime.

* Name
Name your service

提供您的 QnA Maker 知識庫名稱。此範例中，我們會使用 'sample-qna' 名稱。

移至步驟 4

STEP 4

Populate your KB.

Extract question-and-answer pairs from an online FAQ, product manuals, or other files. Supported formats are .tsv, .pdf, .doc, .docx, .xlsx, containing questions and answers in sequence. [Learn more about knowledge base sources](#). Skip this step to add questions and answers manually after creation. The number of sources and file size you can add depends on the QnA service SKU you choose. [Learn more about QnA Maker SKUs](#).

URL

http://

[+ Add URL](#)

File name

[+ Add file](#)

選取 [+ 新增檔案] 選項，瀏覽至您範例程式碼的 CognitiveModel 資料夾，然後選取 'QnAMaker.tsv' 檔案。

有其他選取項目可將「閒聊」個性新增至您的知識庫，但我們的範例不包含此選項。

移至步驟 5

選取 [建立您的 KB]。

從您上傳的檔案中建立知識庫後，選取 [儲存並訓練]，然後在完成時選取 [發佈] 索引標籤，並發佈您的應用程式。

發佈 QnA Maker 應用程式後，選取 [設定] 索引標籤，然後捲動至 [部署詳細資料]。記錄以下來自 Postman 範例 HTTP 要求的值。

```
POST /knowledgebases/<knowledge-base-id>/generateAnswer
Host: <your-hostname> // NOTE - this is a URL.
Authorization: EndpointKey <your-endpoint-key>
```

您主機名稱的完整 URL 字串看起來會類似 "https://< >.azure.net/qnamaker"。

稍後在您的 `appsettings.json` 或 `.env` 檔案中會用到這些值。

請記下 LUIS 應用程式和 QnA Maker 知識庫的名稱和識別碼。也請記下您的 LUIS 撰寫金鑰和認知服務訂用帳戶金鑰。您需要這所有資訊來完成此程序。

建立分派模型

分派工具的 CLI 介面會建立分派給正確服務的模型。

- 開啟命令提示字元或終端機視窗，並將目錄變更為 **CognitiveModels** 目錄
- 請確定您有最新版的 npm 和分派工具。

```
npm i -g npm
npm i -g botdispatch
```

- 使用 `dispatch init` 來初始化分派模型的 `.dispatch` 檔案建立。使用您可辨識的檔案名稱來建立此檔案。

```
dispatch init -n <filename-to-create> --luisAuthoringKey "<your-luis-authoring-key>" --
luisAuthoringRegion <your-region>
```

- 使用 `dispatch add` 將您的 LUIS 應用程式和 QnA Maker 知識庫新增至 `.dispatch` 檔案。

```
dispatch add -t luis -i "<app-id-for-weather-app>" -n "<name-of-weather-app>" -v <app-version-number> -k "<your-luis-authoring-key>" --intentName l_Weather  
dispatch add -t luis -i "<app-id-for-home-automation-app>" -n "<name-of-home-automation-app>" -v <app-version-number> -k "<your-luis-authoring-key>" --intentName l_HomeAutomation  
dispatch add -t qna -i "<knowledge-base-id>" -n "<knowledge-base-name>" -k "<your-cognitive-services-subscription-id>" --intentName q_sample-qna
```

5. 使用 `dispatch create` 從 .dispatch 檔案中產生分派模型。

```
dispatch create
```

6. 使用產生的分派模型 JSON 檔案發佈分派 LUIS 應用程式。

使用分派模型

產生的模型會針對每個應用程式和知識庫定義意圖，也會在語句沒有適當的相符項目時定義_無_意圖。

- `l_HomeAutomation`
- `l_Weather`
- `None`
- `q_sample-qna`

請注意，這些服務必須以正確的名稱發佈，才能讓 Bot 正常運作。

Bot 需要已發佈服務的相關資訊，好讓其可以存取這些服務。

- [C#](#)
- [JavaScript](#)

安裝套件

第一次執行此應用程式之前，請先確認您已安裝數個 NuGet 套件：

Microsoft.Bot.Builder

Microsoft.Bot.Builder.AI.Luis

Microsoft.Bot.Builder.AI.QnA

手動更新 appsettings.json 檔案

建立所有服務應用程式後，每個服務應用程式的資訊都必須新增至 'appsettings.json' 檔案。初始的 [C# 範例程式碼](#) 會包含空白的 appsettings.json 檔案：

appsettings.json [!code-jsonAppSettings]

針對下列所示的每個實體，在這些指示中新增您稍早所記錄的值：

appsettings.json

```
"MicrosoftAppId": "",  
"MicrosoftAppPassword": "",  
  
"QnAKnowledgebaseId": "<knowledge-base-id>",  
"QnAAuthKey": "<your-endpoint-key>",  
"QnAEndpointHostName": "<your-hostname>",  
  
"LuisAppId": "<app-id-for-dispatch-app>",  
"LuisAPIKey": "<your-luis-authoring-key>",  
"LuisAPIHostName": "<your-dispatch-app-region>",
```

完成所有變更時，儲存這個檔案。

從 Bot 連線到服務

若要連線到分派、LUIS 和 QnA Maker 服務，Bot 會從您先前提供的設定中提取資訊。

- [C#](#)
- [JavaScript](#)

在 **BotServices.cs** 中，*appsettings.json* 組態檔中包含的資訊會用於將分派 Bot 連線至 `Dispatch` 和 `SampleQnA` 服務。建構函式會使用您提供的值來連線到這些服務。

BotServices.cs [!code-csharp`ReadConfigurationInfo`]

從 Bot 呼叫服務

對於每個來自您使用者的輸入，Bot 邏輯會針對結合的分派模型來檢查使用者輸入、尋找最高分的傳回意圖，並使用該資訊來為輸入呼叫適當的服務。

- [C#](#)
- [JavaScript](#)

在 **DispatchBot.cs** 檔案中，每當呼叫 `OnMessageActivityAsync` 方法時，我們就會針對分派模型來檢查傳入的使用者訊息。接著，我們會將分派模型的 `topIntent` 和 `recognizerResult` 傳遞至正確的方法，以呼叫服務並傳回結果。

DispatchBot.cs [!code-csharp`OnMessageActivity`]

使用辨識結果

- [C#](#)
- [JavaScript](#)

當模型產生結果時，它會指出哪一項服務最適合處理語句。此 Bot 中的程式碼會將要求路由傳送至對應的服務，然後概述來自所呼叫服務的回應。根據分派傳回的 _意圖_，此程式碼會使用傳回的意圖來路由至正確的 LUIS 模型或 QnA 服務。

DispatchBot.cs [!code-csharp`DispatchToTop`]

如果叫用 `ProcessHomeAutomationAsync` 或 `ProcessWeatherAsync` 方法，則會從 `luisResult.ConnectedServiceResult` 內的分派模型中傳遞結果。接著，指定方法會提供顯示分派模型最高意圖的使用者意見反應，並且以排名列出所有偵測到的意圖和實體。

如果叫用 `q_sample-qna` 方法，該方法會使用 `turnContext` 中包含的使用者輸入，從知識庫產生答案，並將該結果顯示給使用者。

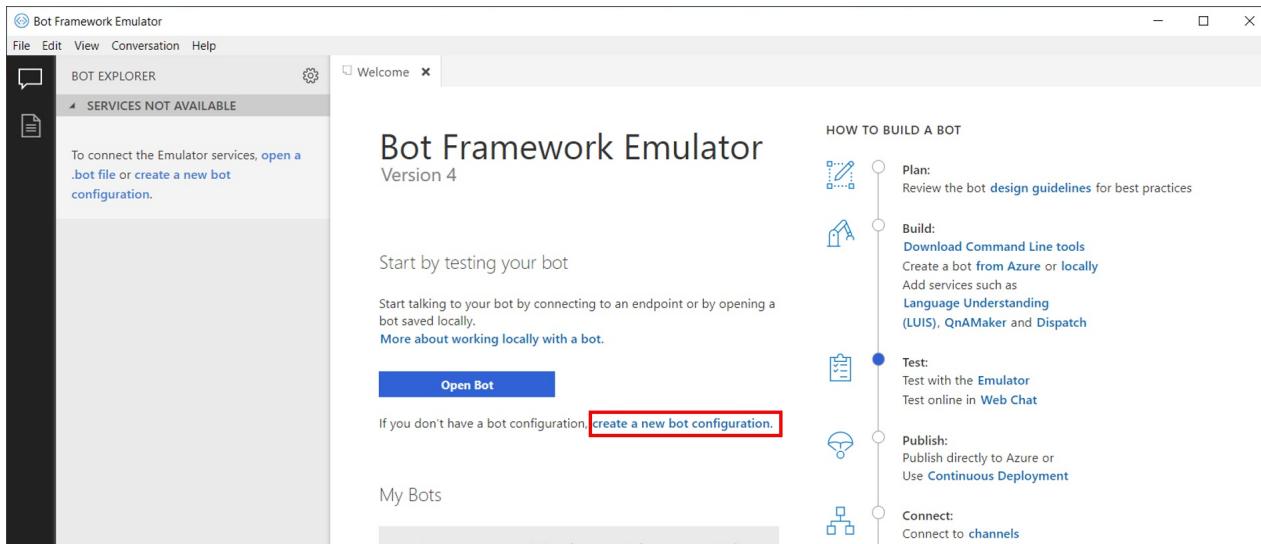
NOTE

如果這是生產應用程式，就會用此應用程式來將選取的 LUIS 方法連線到其指定服務、傳入使用者輸入及處理傳回的 LUIS 意圖和實體資料。

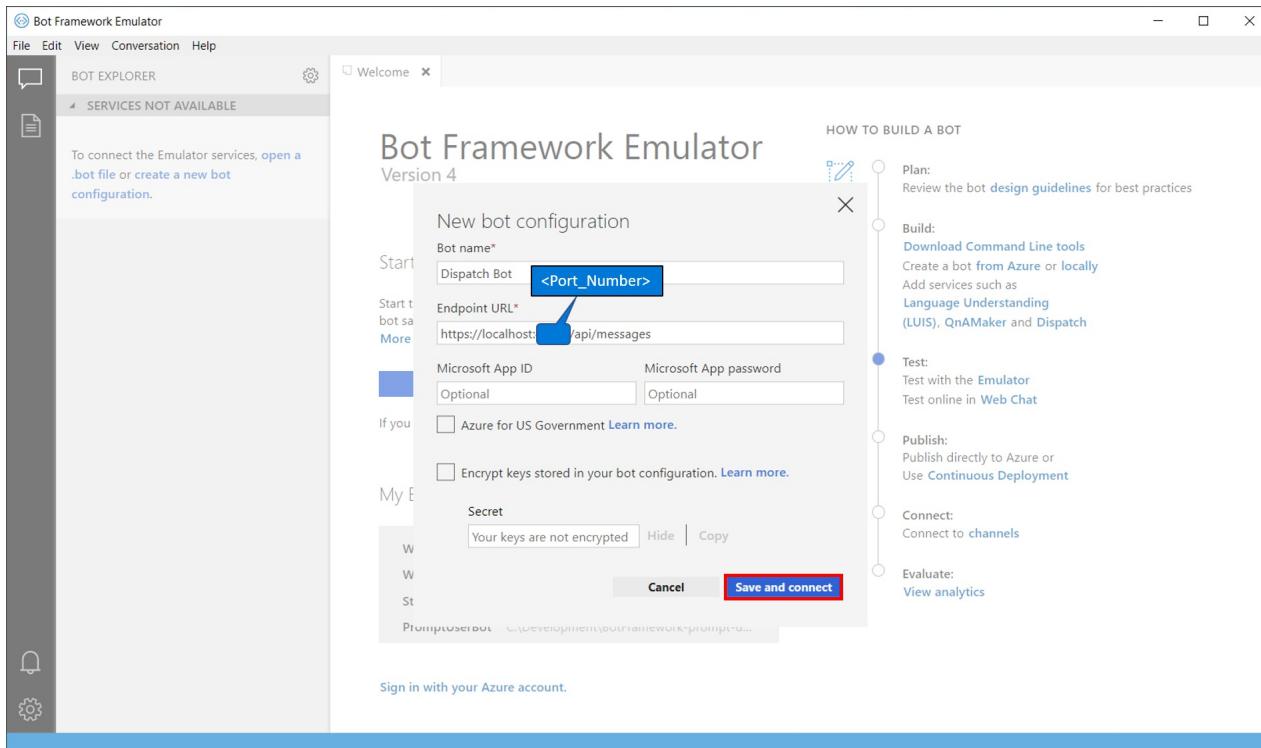
測試 Bot

使用您的開發環境啟動範例程式碼。在由您應用程式開啟的瀏覽器視窗中，記下網址列顯示的 localhost 位址："https://localhost:<Port_Number>" 開啟 Bot Framework Emulator 後，選取下列框起來的藍色文字：

create new bot configuration。



輸入您記下的 localhost 位址，將 '/api/messages' 加到結尾："https://localhost:<Port_Number>/api/messages"



現在，按一下 [Save and connect] 按鈕來存取執行的 Bot。專為 Bot 所建置的服務涵蓋以下一些問題和命令，供您參考：

- QnA Maker
 - hi、good morning
 - what are you、what do you do

- LUIS (住家自動化)

- turn on bedroom light

- turn off bedroom light

- make some coffee

- LUIS (天氣)

- `whats the weather in redmond washington`
- `what's the forecast for london`
- `show me the forecast for nebraska`

其他資訊

當 Bot 正在執行時，移除類似或重疊的語句有可能改善 Bot 的效能。例如，假設在 `Home Automation` LUIS 應用程式中，「開燈」這類要求對應至「TurnOnLights」意圖，但「為什麼無法打開燈？」要求則對應至「無」意圖，因此系統便會將其傳遞至 QnA Maker。使用分派工具結合 LUIS 應用程式和 QnA Maker 服務時，您必須執行下列動作：

- 從原始的 `Home Automation` LUIS 應用程式移除 "None" 意圖，然後將該意圖中的語句新增至發送器應用程式中的 "None" 意圖。
- 如果您未從原始 LUIS 應用程式中移除 "None" 意圖，則必須在 Bot 中新增邏輯，以將符合 "None" 意圖的訊息傳遞至 QnA maker 服務。

上述兩個動作都會減少 Bot 以「找不到答案」訊息回應使用者的次數。

更新或建立新的 LUIS 模型

此範例是以預先設定的 LUIS 模型為基礎。在[這裡](#)可以找到其他資訊，協助您更新此模型，或建立新的 LUIS 模型。

刪除資源

此範例會建立一些應用程式和資源，您可以使用下面所列的步驟將其刪除，但不得刪除「任何其他應用程式或服務」所依賴的資源。

刪除 LUIS 資源：

1. 登入 [luis.ai](#) 入口網站。
2. 移至 [我的應用程式] 頁面。
3. 選取此範例所建立的應用程式。
 - `Home Automation`
 - `Weather`
 - `NLP-With-Dispatch-BotDispatch`
4. 按一下 [刪除]，然後按一下 [確定] 進行確認。

刪除 QnA Maker 資源：

1. 登入 [qnamaker.ai](#) 入口網站。
2. 移至 [我的知識庫] 頁面。
3. 按一下 `Sample QnA` 知識庫的刪除按鈕，然後按一下 [刪除] 進行確認。

最佳做法

若要改善此範例中使用的服務，請參閱 [LUIS](#) 及 [QnA Maker](#) 的最佳做法。

使用分支和迴圈建立進階的交談流程

2019/5/10 • [Edit Online](#)

適用於:  SDK v4  SDK v3

您可以使用對話方塊程式庫來管理簡單和複雜的對話流程。在本文中，我們會說明如何管理可進行分支和迴圈處理的複雜交談。我們也會說明如何在對話塊的不同部分之間傳遞引數。

必要條件

- Bot 基本概念、管理狀態、對話方塊程式庫及如何實作循序對話流程的知識。
- [CSharp](#) 或 [JavaScript](#) 中的複雜對話方塊範例副本。

關於此範例

此範例所代表的 Bot 可讓使用者註冊，以評論清單中最多兩家公司。

`DialogAndWelcomeBot` 會展開 `DialogBot`，以定義不同活動的處理常式及 Bot 的回合處理常式。`DialogBot` 會執行對話方塊：

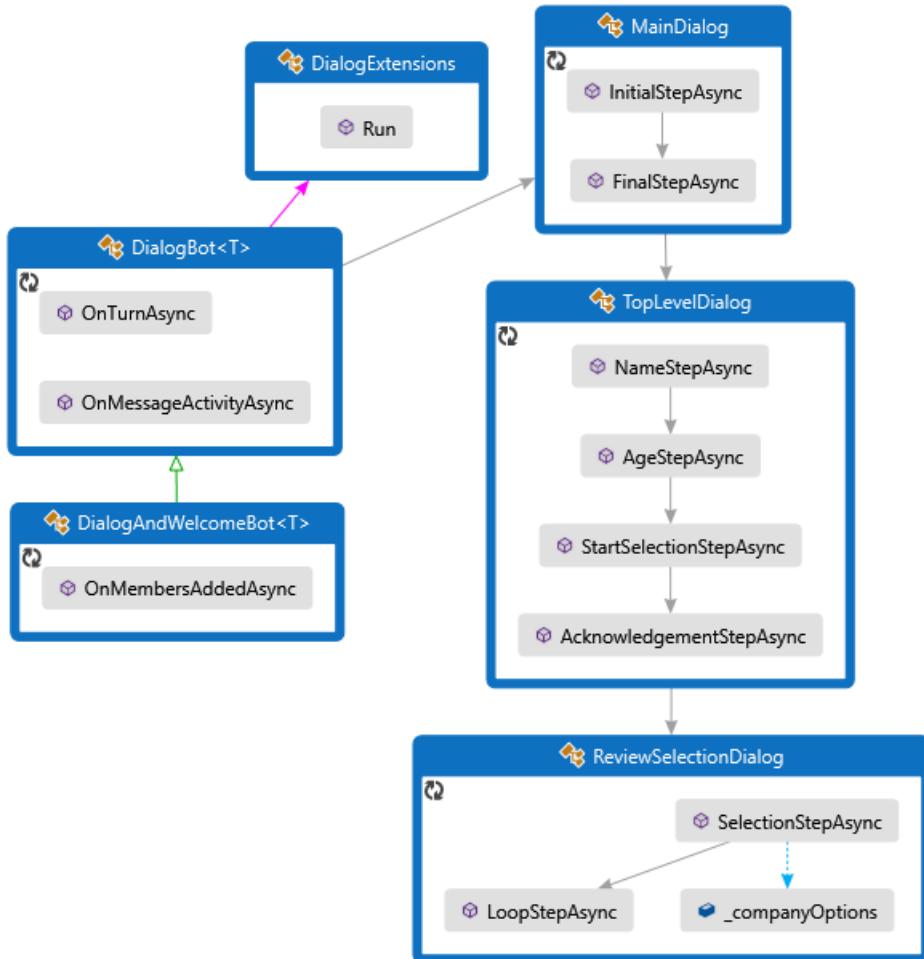
- `DialogBot` 會使用 `run` 方法來開始對話方塊。
- `MainDialog` 是其他兩個對話方塊的父項，只在對話方塊中的特定時間進行呼叫。本文會提供這些對話方塊的詳細資料。

對話方塊分成 `MainDialog`、`TopLevelDialog` 和 `ReviewSelectionDialog` 元件對話，並一起執行下列動作：

- 這些對話方塊會要求使用者的名稱和年齡，然後根據使用者的年齡進行_分支_。
 - 如果使用者太年輕，則不會要求使用者評論任何公司。
 - 如果使用者年紀夠大，就會開始收集使用者的評論喜好。
 - 其可讓使用者選取要評論的公司。
 - 如果使用者選擇了一家公司，這些對話方塊就會以_迴圈_來允許選取第二間公司。
- 並在最後感謝使用者的參與。

其使用兩個瀑布式對話，和一些提示來管理複雜對話。

- [C#](#)
- [JavaScript](#)



若要使用對話方塊，您的專案必須安裝 **Microsoft.Bot.Builder.Dialogs** NuGet 套件。

Startup.cs

我們會在 `Startup` 中為 Bot 註冊服務。這些服務可透過相依性插入來提供給程式碼的其他部分。

- Bot 的基本服務：認證提供者、配接器及 Bot 實作。
- 用於管理狀態的服務：儲存體、使用者狀態及對話狀態。
- Bot 會使用的對話方塊。

```

// Create the credential provider to be used with the Bot Framework Adapter.
services.AddSingleton<ICredentialProvider, ConfigurationCredentialProvider>();

// Create the Bot Framework Adapter with error handling enabled.
services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithErrorHandler>();

// Create the storage we'll be using for User and Conversation state. (Memory is great for testing
purposes.)
services.AddSingleton<IStorage, MemoryStorage>();

// Create the User state. (Used in this bot's Dialog implementation.)
services.AddSingleton<UserState>();

// Create the Conversation state. (Used by the Dialog system itself.)
services.AddSingleton<ConversationState>();

services.AddSingleton<MainDialog>();
// Create the bot as a transient. In this case the ASP Controller is expecting an IBot.
services.AddTransient<IBot, DialogAndWelcomeBot<MainDialog>>();

```

NOTE

記憶體儲存體僅供測試用途，而不適用於生產環境。針對生產 Bot，請務必使用永續性的儲存體類型。

定義用來儲存所收集資訊的類別

- [C#](#)
- [JavaScript](#)

UserProfile.cs

```

/// <summary>Contains information about a user.</summary>
public class UserProfile
{
    public string Name { get; set; }
    public int Age { get; set; }

    // The list of companies the user wants to review.
    public List<string> CompaniesToReview { get; set; } = new List<string>();
}

```

建立要使用的對話方塊

- [C#](#)
- [JavaScript](#)

Dialogs\MainDialog.cs

我們已經定義了元件對話：`MainDialog`，其包含幾個主要步驟，並指示了對話方塊和提示。第一個步驟是呼叫 `TopLevelDialog`（下面會說明）。

```
private async Task<DialogTurnResult> InitialStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    return await stepContext.BeginDialogAsync(nameof(TopLevelDialog), null, cancellationToken);
}

private async Task<DialogTurnResult> FinalStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    var userInfo = (UserProfile)stepContext.Result;

    string status = "You are signed up to review "
        + (userInfo.CompaniesToReview.Count == 0 ? "no companies" : string.Join(" and ", userInfo.CompaniesToReview))
        + ".";

    await stepContext.Context.SendActivityAsync(status);

    var accessor = _userState.CreateProperty<UserProfile>(nameof(UserProfile));
    await accessor.SetAsync(stepContext.Context, userInfo, cancellationToken);

    return await stepContext.EndDialogAsync(null, cancellationToken);
}
```

Dialogs\TopLevelDialog.cs

初始、最上層的對話有四個步驟：

1. 詢問使用者的名稱。
2. 詢問使用者的年齡。
3. 根據使用者的年齡分支。
4. 最後，感謝使用者的參與並傳回所收集的資訊。

在第一個步驟中，我們會清除使用者設定檔，以便對話方塊每次都能從空的設定檔開始。由於最後一個步驟會在結束時傳回資訊，因此 `AcknowledgementStepAsync` 會在結束時將資訊儲存至使用者狀態，然後將該資訊傳回主要對話方塊，以在最後一個步驟中使用。

```

private static async Task<DialogTurnResult> NameStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    // Create an object in which to collect the user's information within the dialog.
    stepContext.Values[UserInfo] = new UserProfile();

    var promptOptions = new PromptOptions { Prompt = MessageFactory.Text("Please enter your name.") };

    // Ask the user to enter their name.
    return await stepContext.PromptAsync(nameof(TextPrompt), promptOptions, cancellationToken);
}

private async Task<DialogTurnResult> AgeStepAsync(WaterfallStepContext stepContext, CancellationToken
cancellationToken)
{
    // Set the user's name to what they entered in response to the name prompt.
    var userProfile = (UserProfile)stepContext.Values[UserInfo];
    userProfile.Name = (string)stepContext.Result;

    var promptOptions = new PromptOptions { Prompt = MessageFactory.Text("Please enter your age.") };

    // Ask the user to enter their age.
    return await stepContext.PromptAsync(nameof(NumberPrompt<int>), promptOptions, cancellationToken);
}

private async Task<DialogTurnResult> StartSelectionStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    // Set the user's age to what they entered in response to the age prompt.
    var userProfile = (UserProfile)stepContext.Values[UserInfo];
    userProfile.Age = (int)stepContext.Result;

    if (userProfile.Age < 25)
    {
        // If they are too young, skip the review selection dialog, and pass an empty list to the next step.
        await stepContext.Context.SendActivityAsync(
            MessageFactory.Text("You must be 25 or older to participate."),
            cancellationToken);
        return await stepContext.NextAsync(new List<string>(), cancellationToken);
    }
    else
    {
        // Otherwise, start the review selection dialog.
        return await stepContext.BeginDialogAsync(nameof(ReviewSelectionDialog), null, cancellationToken);
    }
}

private async Task<DialogTurnResult> AcknowledgementStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    // Set the user's company selection to what they entered in the review-selection dialog.
    var userProfile = (UserProfile)stepContext.Values[UserInfo];
    userProfile.CompaniesToReview = stepContext.Result as List<string> ?? new List<string>();

    // Thank them for participating.
    await stepContext.Context.SendActivityAsync(
        MessageFactory.Text($"Thanks for participating,
{((UserProfile)stepContext.Values[UserInfo]).Name}."),
        cancellationToken);

    // Exit the dialog, returning the collected user information.
    return await stepContext.EndDialogAsync(stepContext.Values[UserInfo], cancellationToken);
}

```

review-selection 對話方塊會從最上層對話方塊的 `StartSelectionStepAsync` 開始，且具有兩個步驟：

1. 要求使用者選擇要評論的公司，或選擇 `done` 表示完成。
2. 視情況重複此對話或結束。

在此設計中，最上層的對話一律優先於堆疊上的 review-selection 對話，而 review-selection 對話可以是最上層對話的子系。

```

private async Task<DialogTurnResult> SelectionStepAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken)
{
    // Continue using the same selection list, if any, from the previous iteration of this dialog.
    var list = stepContext.Options as List<string> ?? new List<string>();
    stepContext.Values[CompaniesSelected] = list;

    // Create a prompt message.
    string message;
    if (list.Count is 0)
    {
        message = $"Please choose a company to review, or `'{DoneOption}` to finish.";
    }
    else
    {
        message = $"You have selected **{list[0]}**. You can review an additional company, " +
            $"or choose `'{DoneOption}` to finish.";
    }

    // Create the list of options to choose from.
    var options = _companyOptions.ToList();
    options.Add(DoneOption);
    if (list.Count > 0)
    {
        options.Remove(list[0]);
    }

    var promptOptions = new PromptOptions
    {
        Prompt = MessageFactory.Text(message),
        RetryPrompt = MessageFactory.Text("Please choose an option from the list."),
        Choices = ChoiceFactory.ToChoices(options),
    };

    // Prompt the user for a choice.
    return await stepContext.PromptAsync(nameof(ChoicePrompt), promptOptions, cancellationToken);
}

private async Task<DialogTurnResult> LoopStepAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken)
{
    // Retrieve their selection list, the choice they made, and whether they chose to finish.
    var list = stepContext.Values[CompaniesSelected] as List<string>;
    var choice = (FoundChoice)stepContext.Result;
    var done = choice.Value == DoneOption;

    if (!done)
    {
        // If they chose a company, add it to the list.
        list.Add(choice.Value);
    }

    if (done || list.Count >= 2)
    {
        // If they're done, exit and return their list.
        return await stepContext.EndDialogAsync(list, cancellationToken);
    }
    else
    {
        // Otherwise, repeat this dialog, passing in the list from this iteration.
        return await stepContext.ReplaceDialogAsync(nameof(ReviewSelectionDialog), list, cancellationToken);
    }
}

```

實作程式碼來管理對話方塊

Bot 的回合處理常式會重複這些對話所定義的一個交談流程。當我們收到來自使用者的訊息時：

1. 繼續進行作用中的對話（如果有的話）。
 - 如果沒有作用中的對話，我們會清除使用者設定檔並開始最上層的對話。
 - 如果作用中對話已完成，我們會收集和儲存所傳回的資訊並顯示狀態訊息。
 - 否則，作用中對話仍為中間程序，而我們目前不需要採取任何其他動作。
2. 儲存交談狀態，以便保存對話狀態的所有更新。
 - [C#](#)
 - [JavaScript](#)

DialogExtensions.cs

在此範例中，我們定義了 `Run` 協助程式方法，我們將用此方法來建立及存取對話方塊內容。由於元件對話會定義內部對話集，因此我們必須建立可讓訊息處理常式程式碼看見的外部對話集，並使用其建立對話方塊內容。

- `dialog` 是 Bot 的主要元件對話。
- `turnContext` 是 Bot 目前的回合內容。

```
public static async Task Run(this Dialog dialog, ITurnContext turnContext,
IStatePropertyAccessor<DialogState> accessor, CancellationToken cancellationToken =
default(CancellationToken))
{
    var dialogSet = new DialogSet(accessor);
    dialogSet.Add(dialog);

    var dialogContext = await dialogSet.CreateContextAsync(turnContext, cancellationToken);
    var results = await dialogContext.ContinueDialogAsync(cancellationToken);
    if (results.Status == DialogTurnStatus.Empty)
    {
        await dialogContext.BeginDialogAsync(dialog.Id, null, cancellationToken);
    }
}
```

Bots\DialogBot.cs

訊息處理常式會呼叫 `Run` 協助程式方法來管理對話方塊，而我們已經覆寫回合處理常式來將任何變更儲存至對話和使用者狀態，這可能會在回合期間發生。基礎 `OnTurnAsync` 會呼叫 `OnMessageActivityAsync` 方法，確保儲存呼叫會在回合結束時執行。

```

public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken =
default(CancellationToken))
{
    await base.OnTurnAsync(turnContext, cancellationToken);

    // Save any state changes that might have occurred during the turn.
    await ConversationState.SaveChangesAsync(turnContext, false, cancellationToken);
    await UserState.SaveChangesAsync(turnContext, false, cancellationToken);
}

protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    Logger.LogInformation("Running dialog with Message Activity.");

    // Run the Dialog with the new message Activity.
    await Dialog.Run(turnContext, ConversationState.CreateProperty<DialogState>(nameof(DialogState)),
cancellationToken);
}

```

Bots\DialogAndWelcome.cs

`DialogAndWelcomeBot` 會展開上述的 `DialogBot`, 以在使用者加入對話時提供歡迎訊息, 並且是 `Startup.cs` 所呼叫的項目。

```

protected override async Task OnMembersAddedAsync(
    IList<ChannelAccount> membersAdded,
    ITurnContext<IConversationUpdateActivity> turnContext,
    CancellationToken cancellationToken)
{
    foreach (var member in membersAdded)
    {
        // Greet anyone that was not the target (recipient) of this message.
        // To learn more about Adaptive Cards, see https://aka.ms/msbot-adaptivecards for more details.
        if (member.Id != turnContext.Activity.Recipient.Id)
        {
            var reply = MessageFactory.Text($"Welcome to Complex Dialog Bot {member.Name}. " +
                "This bot provides a complex conversation, with multiple dialogs. " +
                "Type anything to get started.");
            await turnContext.SendActivityAsync(reply, cancellationToken);
        }
    }
}

```

分支和迴圈

- [C#](#)
- [JavaScript](#)

Dialogs\TopLevelDialog.cs

以下是分支邏輯範例, 來自_最上層_對話方塊的步驟:

```
if (userProfile.Age < 25)
{
    // If they are too young, skip the review selection dialog, and pass an empty list to the next step.
    await stepContext.Context.SendActivityAsync(
        MessageFactory.Text("You must be 25 or older to participate."),
        cancellationToken);
    return await stepContext.NextAsync(new List<string>(), cancellationToken);
}
else
{
    // Otherwise, start the review selection dialog.
    return await stepContext.BeginDialogAsync(nameof(ReviewSelectionDialog), null, cancellationToken);
}
```

Dialogs\ReviewSelectionDialog.cs

以下是迴圈邏輯範例，來自_檢閱選取_對話方塊的步驟：

```
if (done || list.Count >= 2)
{
    // If they're done, exit and return their list.
    return await stepContext.EndDialogAsync(list, cancellationToken);
}
else
{
    // Otherwise, repeat this dialog, passing in the list from this iteration.
    return await stepContext.ReplaceDialogAsync(nameof(ReviewSelectionDialog), list, cancellationToken);
}
```

測試 Bot

1. 如果您尚未安裝 [Bot Framework Emulator](#)，請進行安裝。
2. 在您的電腦本機執行範例。
3. 啟動模擬器、連線到您的 Bot 並傳送如下所示的訊息。

Bot Framework Emulator (V4 PREVIEW) - ComplexDialogBot

File Edit View Conversation Help

BOT EXPLORER ENDPOINT SERVICES

Live Chat (development) x
Start Over Save Transcript As...
http://localhost:3978/api/messages

Welcome to ComplexDialogBot. This bot provides a complex conversation, with multiple dialogs. Type anything to get started.

Bot

User

hi

Please enter your name.

Bot

User

Paul

Please enter your age.

Bot

User

35

Please choose a company to review, or done to finish.

1. Adatum Corporation
2. Contoso Suites
3. Graphic Design Institute
4. Wide World Importers
5. done

Bot

User

1

You have selected **Adatum Corporation**. You can review an additional company, or choose done to finish.

1. Contoso Suites
2. Graphic Design Institute
3. Wide World Importers
4. done

Bot

User

Contoso Suites

Thanks for participating, Paul.

Bot

You are signed up to review Adatum Corporation and Contoso Suites.

Bot at 3:29:50 PM

Type your message...

The screenshot shows a conversation in the Bot Framework Emulator. The user starts by saying 'hi' and the bot asks for their name. The user replies with 'Paul'. The bot then asks for their age, which the user enters as '35'. The bot presents a list of companies for the user to review, with 'Adatum Corporation' selected. The user then selects 'Contoso Suites' from the list. The bot responds with a confirmation message and informs the user they are signed up to review both companies. The interface includes a sidebar for 'BOT EXPLORER', 'ENDPOINT', and 'SERVICES' management.

其他資源

如需實作對話方式的簡介，請參閱[實作循序對話流程](#)，該流程使用單一瀑布式對話和一些提示來建立簡單的互動，以詢問使用者一連串的問題。

Dialogs 程式庫包含提示的基本驗證。您也可以新增自訂憑證。如需詳細資訊，請參閱[使用對話提示收集使用者輸入](#)。

若要簡化對話程式碼並重複使用於多個 Bot，您可以將部分的對話集定義為個別的類別。如需詳細資訊，請參閱[重複使用對話方塊](#)。

後續步驟

[重複使用對話方塊](#)

重複使用對話方塊

2019/5/10 • [Edit Online](#)

適用於:  SDK v4  SDK v3

透過元件對話，您可以建立獨立的對話來處理特定案例，並將大型對話集分割成更多可管理的片段。這些片段都有自己的對話集，而且會避免與本身之外設定的對話集發生任何名稱衝突。

必要條件

- [Bot 基本概念、對話方塊程式庫及如何管理對話](#)的知識。
- [CSharp](#) 或 [JavaScript](#) 中的一份多回合提示範例。

關於範例

在多回合提示範例中，我們會使用瀑布式對話方塊、一些提示和元件對話來建立簡單互動，以詢問使用者一系列的問題。程式碼會使用對話方塊來循環下列步驟：

步驟	提示類型
要求使用者的運輸模式	選擇提示
要求使用者的名稱	文字提示
詢問使用者是否要提供年齡	確認提示
如果他們回答 [是]，則要求年齡	包含驗證的數字提示，只接受大於 0 且小於 150 的年齡。
詢問是否可收集資訊	重複使用確認提示

最後，如果他們回答 [是]，則顯示所收集的資訊；否則，告訴使用者將不會保留其資訊。

實作元件對話

在多回合提示範例中，我們會使用_瀑布式對話方塊_、一些_提示_和_元件對話_來建立簡單互動，以詢問使用者一系列的問題。

元件對話會封裝一個或多個對話。元件對話有內部對話集，新增至內部對話集的對話和提示會有其本身的識別碼，只在元件對話中顯示。

- [C#](#)
- [JavaScript](#)

若要使用對話方塊，請安裝 **Microsoft.Bot.Builder.Dialogs** NuGet 套件。

Dialogs\UserProfileDialog.cs

此處的 `UserProfileDialog` 類別衍生自 `ComponentDialog` 類別。

```
public class UserProfileDialog : ComponentDialog
```

在建構函式中，`AddDialog` 方法會將對話和提示新增至元件對話。您使用此方法新增的第一個項目會設定為初始對話，但您可以藉由明確地設定 `InitialDialogId` 屬性，來加以變更。當您啟動元件對話時，就會啟動其「初始對話」。

```
public UserProfileDialog(UserState userState)
    : base(nameof(UserProfileDialog))
{
    _userProfileAccessor = userState.CreateProperty<UserProfile>("UserProfile");

    // This array defines how the Waterfall will execute.
    var waterfallSteps = new WaterfallStep[]
    {
        TransportStepAsync,
        NameStepAsync,
        NameConfirmStepAsync,
        AgeStepAsync,
        ConfirmStepAsync,
        SummaryStepAsync,
    };

    // Add named dialogs to the DialogSet. These names are saved in the dialog state.
    AddDialog(new WaterfallDialog(nameof(WaterfallDialog), waterfallSteps));
    AddDialog(new TextPrompt(nameof(TextPrompt)));
    AddDialog(new NumberPrompt<int>(nameof(NumberPrompt<int>), AgePromptValidatorAsync));
    AddDialog(new ChoicePrompt(nameof(ChoicePrompt)));
    AddDialog(new ConfirmPrompt(nameof(ConfirmPrompt)));

    // The initial child Dialog to run.
    InitialDialogId = nameof(WaterfallDialog);
}
```

這是瀑布式對話方塊中第一個步驟的實作。

```
private static async Task<DialogTurnResult> TransportStepAsync(WaterfallStepContext stepContext,
    CancellationToken cancellationToken)
{
    // WaterfallStep always finishes with the end of the Waterfall or with another dialog; here it is a
    // Prompt Dialog.
    // Running a prompt here means the next WaterfallStep will be run when the users response is received.
    return await stepContext.PromptAsync(nameof(ChoicePrompt),
        new PromptOptions
        {
            Prompt = MessageFactory.Text("Please enter your mode of transport."),
            Choices = ChoiceFactory.ToChoices(new List<string> { "Car", "Bus", "Bicycle" }),
        }, cancellationToken);
}
```

如需有關如何實作瀑布式對話方塊的詳細資訊，請參閱如何[實作循序對話流程](#)。

在執行階段上，元件對話會維護其自己的對話堆疊。當元件對話啟動時：

- 執行個體會建立並新增至外部對話堆疊
- 其會建立新增至其狀態的內部對話堆疊
- 其會啟動初始對話，並將該對話新增至內部對話堆疊。

在父代內容中，這看起來就像元件是使用中的對話。在元件內部，這看起來就像初始對話是使用中的對話。

實作對話的其餘部分，並將其新增到 Bot

在外部的對話集內（也就是您在其中新增元件對話的對話集），元件對話會有您建立元件時使用的識別碼。在外部集合中，元件看起來就像單一對話，與提示相當類似。

若要使用元件對話，請將其執行個體新增至 Bot 的對話集 - 這是外部對話集。

- [C#](#)
- [JavaScript](#)

DialogExtensions.cs

在範例中，這會使用 [Run](#) 擴充方法來執行，如下所示。

```
public static async Task Run(this Dialog dialog, ITurnContext turnContext,
    IStatePropertyAccessor<DialogState> accessor, CancellationToken cancellationToken =
    default(CancellationToken))
{
    var dialogSet = new DialogSet(accessor);
    dialogSet.Add(dialog);

    var dialogContext = await dialogSet.CreateContextAsync(turnContext, cancellationToken);
    var results = await dialogContext.ContinueDialogAsync(cancellationToken);
    if (results.Status == DialogTurnStatus.Empty)
    {
        await dialogContext.BeginDialogAsync(dialog.Id, null, cancellationToken);
    }
}
```

Bots\DialogBot.cs

[Run](#) 方法是從 Bot 的 [OnMessageActivityAsync](#) 方法中呼叫。

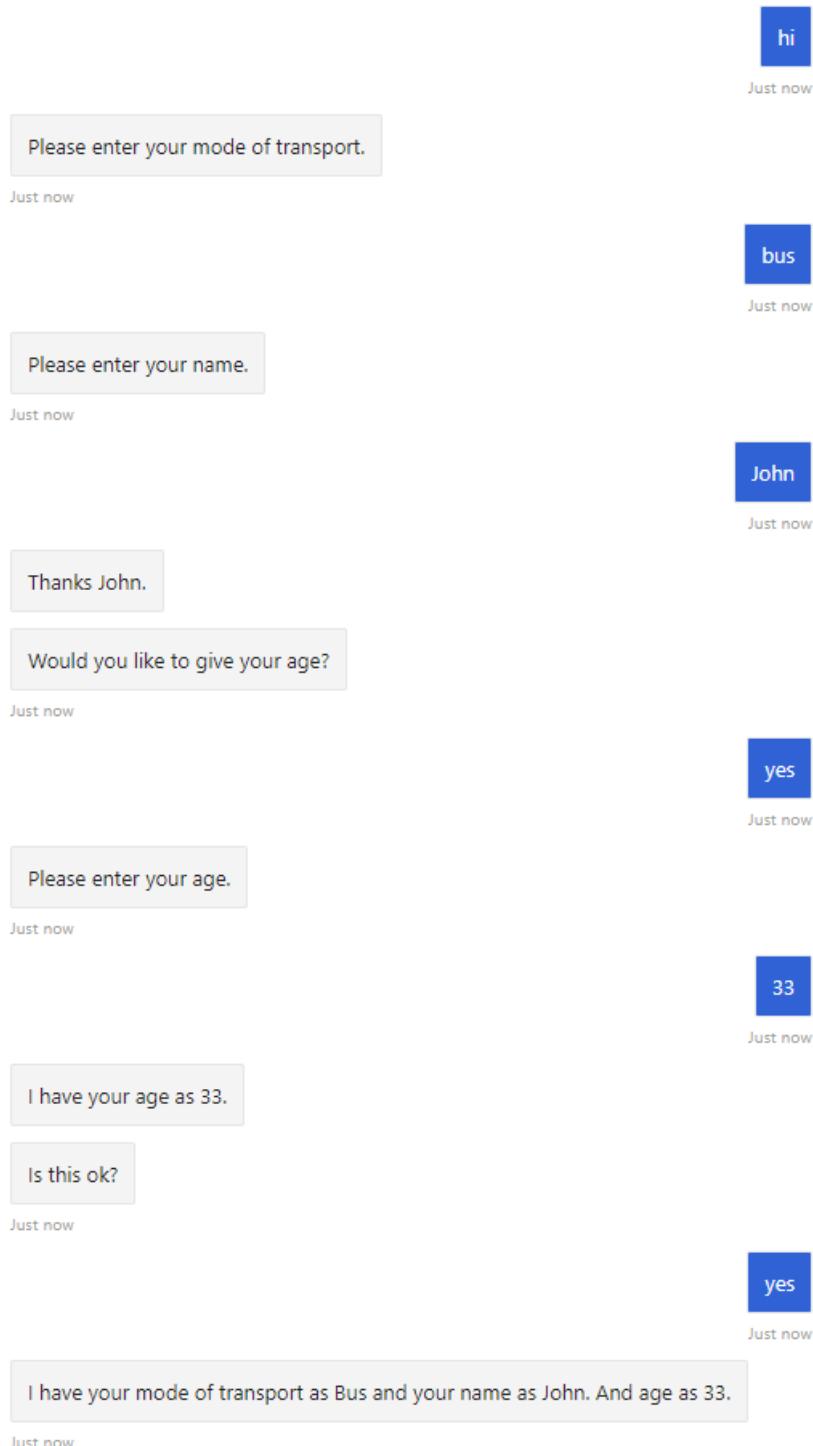
```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
    CancellationToken cancellationToken)
{
    Logger.LogInformation("Running dialog with Message Activity.");

    // Run the Dialog with the new message Activity.
    await Dialog.Run(turnContext, ConversationState.CreateProperty<DialogState>(nameof(DialogState)),
        cancellationToken);
}
```

測試 Bot

1. 如果您尚未安裝 [Bot Framework Emulator](#)，請進行安裝。
2. 在您的電腦本機執行範例。
3. 啟動模擬器、連線到您的 Bot 並傳送如下所示的訊息。

<http://localhost:3978/api/messages>



其他資訊

元件對話的取消作業如何運作

如果您從元件對話的內容呼叫`_取消所有對話_`, 元件對話將會取消其內部堆疊上的所有對話, 然後結束對話, 並將控制項傳回給外部堆疊上的下一個對話。

如果您從外部內容呼叫`_取消所有對話_`, 則元件及外部內容上其餘的對話都會遭到取消。

在 Bot 中管理巢狀元件對話時, 請記住這點。

後續步驟

您可以增強 Bot 以回應其他輸入, 例如 [說明] 或 [取消] 可以中斷交談的一般流程。

處理使用者中斷

處理使用者中斷

2019/5/14 • [Edit Online](#)

適用於: SDK v4 SDK v3

處理中斷是強固 Bot 很重要的層面。使用者不一定會按照所定義的對話流程逐步進行。其可能會嘗試詢問程序中期才會提出的問題，或想要直接取消問題而不想加以完成。在本主題中，我們將會探索一些可供處理 Bot 使用者中斷的常見方式。

必要條件

- Bot 基本概念、管理狀態、對話方塊程式庫 及如何重複使用對話方塊的知識。
- 一份核心的 Bot 範例 (使用 **CSharp** 或 **JavaScript**)。

關於此範例

本文所使用的範例會設計一個預訂航班的 Bot，其會使用對話方塊來向使用者取得航班資訊。使用者可以在與 Bot 對話期間，隨時發出_協助_或_取消_命令來中斷對話。在此，我們會處理兩種中斷：

- **回合層級**: 略過回合層級的處理，但讓對話方塊與所提供的資訊留在堆疊上。在下一個回合時，從我們離開的地方接續進行。
- **對話方塊層級**: 完全取消處理，讓 Bot 可以重新來過一次。

定義和實作中斷邏輯

首先，我們要定義並實作_協助_和_取消_中斷。

- **C#**
- **JavaScript**

若要使用對話方塊，請安裝 **Microsoft.Bot.Builder.Dialogs** NuGet 套件。

Dialogs\CancelAndHelpDialog.cs

一開始，我們會先實作 `CancelAndHelpDialog` 類別來處理使用者中斷。

```
public class CancelAndHelpDialog : ComponentDialog
```

在 `CancelAndHelpDialog` 類別中，`OnBeginDialogAsync` 和 `OnContinueDialogAsync` 方法會呼叫 `InerruptAsync` 方法來檢查使用者是否已中斷一般流程。如果流程已中斷，則會呼叫基底類別方法；，否則會從 `InterruptAsync` 傳回傳回值。

```

protected override async Task<DialogTurnResult> OnBeginDialogAsync(DialogContext innerDc, object options,
CancellationToken cancellationToken = default(CancellationToken))
{
    var result = await InterruptAsync(innerDc, cancellationToken);
    if (result != null)
    {
        return result;
    }

    return await base.OnBeginDialogAsync(innerDc, options, cancellationToken);
}

```

如果使用者輸入「協助」，`InterruptAsync` 方法會傳送一則訊息，然後呼叫 `DialogTurnResult (DialogTurnStatus.Waiting)` 來指出最上層的對話方塊正在等候使用者回應。如此一來，系統便只會在一個回合內中斷對話流程，到下一個回合時，我們就會從離開的地方接續進行。

如果使用者輸入「取消」，則會在其內部對話方塊內容上呼叫 `CancelAllDialogsAsync`，此方法會清除其對話方塊堆疊，並讓其以取消的狀態結束，且不會有結果值。到 `MainDialog` (稍後會顯示) 時，其會顯示預約對話方塊已結束並傳回 Null，情況類似使用者選擇不要確認其預約時。

```

private async Task<DialogTurnResult> InterruptAsync(DialogContext innerDc, CancellationToken
cancellationToken)
{
    if (innerDc.Context.Activity.Type == ActivityTypes.Message)
    {
        var text = innerDc.Context.Activity.Text.ToLowerInvariant();

        switch (text)
        {
            case "help":
            case "?":
                await innerDc.Context.SendActivityAsync($"Show Help...", cancellationToken:
cancellationToken);
                return new DialogTurnResult(DialogTurnStatus.Waiting);

            case "cancel":
            case "quit":
                await innerDc.Context.SendActivityAsync($"Cancelling", cancellationToken: cancellationToken);
                return await innerDc.CancelAllDialogsAsync();
        }
    }

    return null;
}

```

每回合檢查中斷

現在，我們已說明過中斷處理類別的運作方式，接著讓我們回頭看看當 Bot 收到來自使用者的新訊息時，會發生什麼事。

- C#
- JavaScript

Dialogs\MainDialog.cs

有新的訊息活動送達時，Bot 會執行 `MainDialog`。`MainDialog` 會提示使用者其能提供什麼協助。然後，其會藉由呼叫 `BeginDialogAsync` 在 `MainDialog.ActStepAsync` 方法中啟動 `BookingDialog`，如下所示。

```

private async Task<DialogTurnResult> ActStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    // Call LUIS and gather any potential booking details. (Note the TurnContext has the response to the prompt.)
    var bookingDetails = stepContext.Result != null
        ?
        await LuisHelper.ExecuteLuisQuery(Configuration, Logger, stepContext.Context, cancellationToken)
        :
        new BookingDetails();

    // In this sample we only have a single Intent we are concerned with. However, typically a scenario
    // will have multiple different Intents each corresponding to starting a different child Dialog.

    // Run the BookingDialog giving it whatever details we have from the LUIS call, it will fill out the remainder.
    return await stepContext.BeginDialogAsync(nameof(BookingDialog), bookingDetails, cancellationToken);
}

```

接下來，在 `MainDialog` 類別的 `FinalStepAsync` 方法中，預約對話方塊會結束，並認為預約已完成或取消。

```

private async Task<DialogTurnResult> FinalStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    // If the child dialog ("BookingDialog") was cancelled or the user failed to confirm, the Result here will be null.
    if (stepContext.Result != null)
    {
        var result = (BookingDetails)stepContext.Result;

        // Now we have all the booking details call the booking service.

        // If the call to the booking service was successful tell the user.

        var timeProperty = new TimexProperty(result.TravelDate);
        var travelDateMsg = timeProperty.ToNaturalLanguage(DateTime.Now);
        var msg = $"I have you booked to {result.Destination} from {result.Origin} on {travelDateMsg}";
        await stepContext.Context.SendActivityAsync(MessageFactory.Text(msg), cancellationToken);
    }
    else
    {
        await stepContext.Context.SendActivityAsync(MessageFactory.Text("Thank you."), cancellationToken);
    }
    return await stepContext.EndDialogAsync(cancellationToken: cancellationToken);
}

```

這裡不會顯示 `BookingDialog` 中的程式碼，因為其與中斷處理沒有直接關聯。其會用來提示使用者輸入預約詳細資料。您可以在 `Dialogs\BookingDialogs.cs` 中找到該程式碼。

處理未預期的錯誤

接下來，我們要處理可能會發生的任何未處理例外狀況。

- [C#](#)
- [JavaScript](#)

`AdapterWithErrorHandler.cs`

在我們的範例中，配接器的 `OnTurnError` 處理常式會接收 Bot 的回合邏輯所擲回的任何例外狀況。如果有擲回的例外狀況，處理常式就會刪除目前對話的對話狀態，防止 Bot 卡在因為狀態不良而導致的錯誤迴圈中。

```

public class AdapterWithErrorHandler : BotFrameworkHttpAdapter
{
    public AdapterWithErrorHandler(ICredentialProvider credentialProvider, ILogger<BotFrameworkHttpAdapter>
logger, ConversationState conversationState = null)
        : base(credentialProvider)
    {
        OnTurnError = async (turnContext, exception) =>
        {
            // Log any leaked exception from the application.
            logger.LogError($"Exception caught : {exception.Message}");

            // Send a catch-all apology to the user.
            await turnContext.SendActivityAsync("Sorry, it looks like something went wrong.");

            if (conversationState != null)
            {
                try
                {
                    // Delete the conversationState for the current conversation to prevent the
                    // bot from getting stuck in a error-loop caused by being in a bad state.
                    // ConversationState should be thought of as similar to "cookie-state" in a Web pages.
                    await conversationState.DeleteAsync(turnContext);
                }
                catch (Exception e)
                {
                    logger.LogError($"Exception caught on attempting to Delete ConversationState :
{e.Message}");
                }
            }
        };
    }
}

```

註冊伺服器

- C#
- JavaScript

Startup.cs

最後，在 `Startup.cs` 中會建立暫時性的 Bot，且每一回合都會建立新的 Bot 執行個體。

```

// Create the bot as a transient. In this case the ASP Controller is expecting an IBot.
services.AddTransient<IBot, DialogAndWelcomeBot<MainDialog>>();

```

如需參考，以下是呼叫中用來建立上述 Bot 的類別定義。

```

public class MainDialog : ComponentDialog

```

```

public class DialogAndWelcomeBot<T> : DialogBot<T> where T : Dialog

```

```

public class DialogBot<T> : ActivityHandler where T : Dialog

```

測試 Bot

1. 如果您尚未安裝 [Bot Framework Emulator](#)，請進行安裝。

2. 在您的電腦本機執行範例。
3. 啟動模擬器、連線到您的 Bot 並傳送如下所示的訊息。

其他資訊

- [驗證範例](#)會示範如何處理登出，此登出會使用如下所示的類似模式來處理中斷。
- 您應該傳送預設回應，而不是毫無作為，讓使用者臆測究竟發生什麼事。預設回應應該告知使用者 Bot 可以理解哪些命令，讓使用者可以回到正軌。
- 在回合中的任何時點，回合內容的_回應_屬性都會指出 Bot 是否已在這個回合傳送訊息給使用者。回合結束之前，Bot 應該會傳送某些訊息給使用者，即使是簡單的輸入通知也是如此。

直接寫入儲存體

2019/5/14 • [Edit Online](#)

適用於:  SDK v4  SDK v3

您可以將資料直接讀取及寫入至儲存體物件，無需使用中介軟體或內容物件。這種方式適用於 Bot 使用者用來保留交談的資料，或來自 Bot 交談流程以外來源的資料。在此資料儲存體模型中，資料會直接從儲存體讀取，而非使用狀態管理員。本文中的程式碼範例示範如何使用 [記憶體儲存體]、[Cosmos DB]、[Blob 儲存體] 和 [Azure Blob 文字記錄儲存區]，將資料讀取和寫入至儲存體。

必要條件

- 如果您沒有 Azure 訂用帳戶，請在開始前建立[免費帳戶](#)。
- 熟悉本文：本機上針對 [dotnet](#) 或 [nodeJS](#) 建立 Bot。
- 適用於 [C#](#) 或 [nodeJS](#) 和 [yeoman](#) 的 Bot Framework SDK v4 範本。

關於此範例

本文中的範例程式碼是以基本 Echo Bot 的結構開頭，然後藉由新增額外的程式碼（下面提供）來擴充該 Bot 的功能。此擴充程式碼會建立一份清單，以保留其收到的使用者輸入。在每個回合中，系統會將完整的使用者輸入清單回應給使用者。在該回合結束時，包含此輸入清單的資料結構會接著儲存到儲存體。此範例程式碼中新增其他功能時，系統會探索各種類型的儲存體。

記憶體儲存體

Bot Framework SDK 可讓您使用「記憶體內部儲存體」來儲存使用者輸入。記憶體儲存體僅供測試用途，而不適用於生產環境。生產 Bot 最適合使用永續性儲存體類型（例如資料庫儲存體）。務必先將儲存體設定為 Cosmos DB 或 Blob 儲存體，再發佈您的 Bot。

建置基本 Bot

本主題的其餘部分是以 Echo Bot 為基礎。依照建置 [C# EchoBot](#) 或 [JS EchoBot](#) 的快速入門指示，可以在本機建置 Echo Bot 範例程式碼。

- [C#](#)
- [JavaScript](#)

EchoBot.cs

```
using System;
using System.Threading.Tasks;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Schema;
using System.Collections.Generic;
using System.Linq;
using System.Threading;

// Represents a bot saves and echoes back user input.
public class EchoBot : ActivityHandler
{
    // Create local Memory Storage.
    private static readonly MemoryStorage _myStorage = new MemoryStorage();

    // Create cancellation token (used by Async Write operation).
    public CancellationToken cancellationToken { get; private set; }
```

```

// Class for storing a log of utterances (text of messages) as a list.
public class UtteranceLog : IStoreItem
{
    // A list of things that users have said to the bot
    public List<string> UtteranceList { get; } = new List<string>();

    // The number of conversational turns that have occurred
    public int TurnNumber { get; set; } = 0;

    // Create concurrency control where this is used.
    public string ETag { get; set; } = "*";
}

// Echo back user input.
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    // preserve user input.
    var utterance = turnContext.Activity.Text;
    // make empty local logitems list.
    UtteranceLog logItems = null;

    // see if there are previous messages saved in storage.
    try
    {
        string[] utteranceList = { "UtteranceLog" };
        logItems = _myStorage.ReadAsync<UtteranceLog>(utteranceList).Result?.FirstOrDefault().Value;
    }
    catch
    {
        // Inform the user an error occured.
        await turnContext.SendActivityAsync("Sorry, something went wrong reading your stored messages!");
    }

    // If no stored messages were found, create and store a new entry.
    if (logItems is null)
    {
        // add the current utterance to a new object.
        logItems = new UtteranceLog();
        logItems.UtteranceList.Add(utterance);
        // set initial turn counter to 1.
        logItems.TurnNumber++;

        // Show user new user message.
        await turnContext.SendActivityAsync($"{{logItems.TurnNumber}}: The list is now: {string.Join(", ", logItems.UtteranceList)}");

        // Create Dictionary object to hold received user messages.
        var changes = new Dictionary<string, object>();
        {
            changes.Add("UtteranceLog", logItems);
        }
        try
        {
            // Save the user message to your Storage.
            await _myStorage.WriteAsync(changes, cancellationToken);
        }
        catch
        {
            // Inform the user an error occured.
            await turnContext.SendActivityAsync("Sorry, something went wrong storing your message!");
        }
    }
    // Else, our Storage already contained saved user messages, add new one to the list.
    else
    {
        // add new message to list of messages to display.
        logItems.UtteranceList.Add(utterance);
    }
}

```

```
// increment turn counter.  
logItems.TurnNumber++;  
  
// show user new list of saved messages.  
await turnContext.SendActivityAsync($"{logItems.TurnNumber}: The list is now: {string.Join(", ",  
logItems.UtteranceList)}");  
  
// Create Dictionary object to hold new list of messages.  
var changes = new Dictionary<string, object>();  
{  
    changes.Add("UtteranceLog", logItems);  
};  
  
try  
{  
    // Save new list to your Storage.  
    await _myStorage.WriteAsync(changes, cancellationToken);  
}  
catch  
{  
    // Inform the user an error occurred.  
    await turnContext.SendActivityAsync("Sorry, something went wrong storing your message!");  
}  
}  
}  
... // OnMessageActivityAsync( )  
}  
}  
}
```

啟動 Bot

在本機執行您的 Bot。

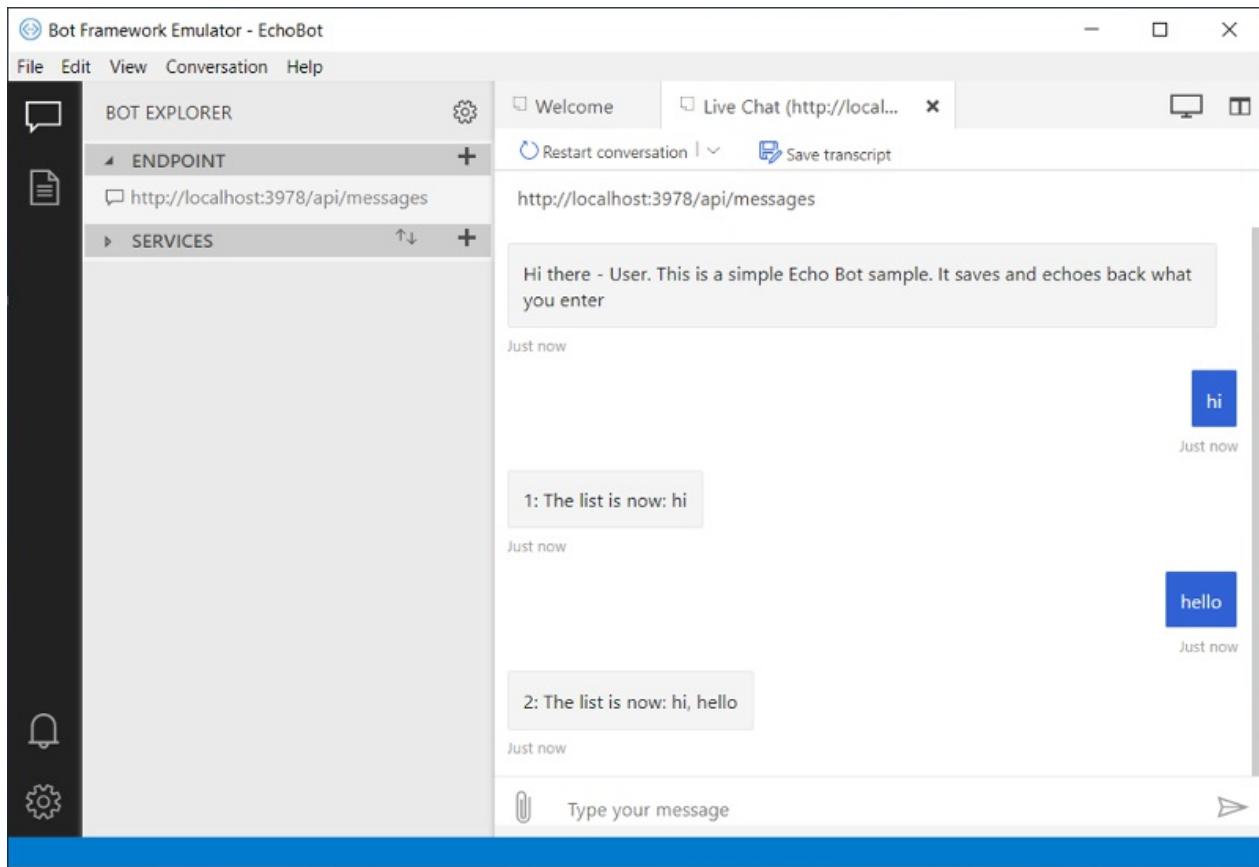
啟動模擬器並且連線至您的 Bot

- 安裝 Bot Framework Emulator 接下來，啟動模擬器，然後在模擬器中連線至您的 Bot：

1. 按一下模擬器 [歡迎使用] 索引標籤中的 [建立新的 Bot 設定] 連結。
2. 以您啟動 Bot 時顯示在網頁上的資訊，填妥欄位以連線到 Bot。

與您的 Bot 互動

將訊息傳送給 Bot。Bot 會列出所收到的訊息。



使用 Cosmos DB

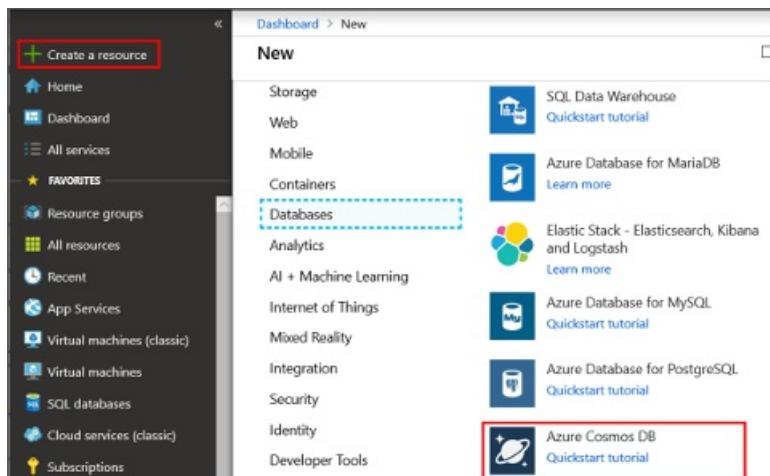
既然您已使用記憶體儲存體，我們會將程式碼更新為使用 Azure Cosmos DB。Cosmos DB 是 Microsoft 全球發行的多模型資料庫。Azure Cosmos DB 可讓您有彈性且獨立地跨任意數目的 Azure 地理區域調整輸送量和儲存體。它利用完整的服務等級協定 (SLA) 提供了輸送量、延遲、可用性和一致性的保證。

設定

若要在 Bot 中使用 Cosmos DB，您需要先設定一些事項，再進入程式碼。如需建立 Cosmos DB 資料庫和應用程式的深入說明，請存取 [Cosmos DB dotnet](#) 或[Cosmos DB nodejs](#) 的文件。

建立資料庫帳戶

- 在新的瀏覽器視窗中，登入 [Azure 入口網站](#)。



- 按一下 [建立資源] > [資料庫] > [Azure Cosmos DB]。

3. 在 [新增帳戶] 頁面上，提供 [訂用帳戶]、[資源群組] 資訊。為您的 [帳戶名稱] 欄位建立唯一的名稱 - 這最終會成為您的資料存取 URL 名稱的一部分。針對 [API]，選取 [Core(SQL)]，並提供附近的 [位置] 來改善資料存取時間。
4. 然後按一下 [檢閱 + 建立]。
5. 驗證成功後，按一下 [建立]。

建立帳戶需要幾分鐘的時間。等候入口網站顯示 恭喜! 已建立您的 Azure Cosmos DB 帳戶 頁面。

新增集合

1. 按一下 [設定] > [新增集合]。[新增集合] 區域會顯示在最右邊，您可能需要向右捲動才能看到它。由於 Cosmos DB 的近期更新，請務必新增單一分割區索引鍵：/id。此索引鍵可避免跨分割區查詢錯誤。

The screenshot shows the Azure SQL API blade. A database named "bot-cosmos-sql-db" is selected. Under it, a collection named "bot-storage" is expanded, revealing sub-items: "Documents", "Scale & Settings", "Stored Procedures", "User Defined Functions", and "Triggers".

2. 新資料庫集合 "bot-cosmos-sql-db" 的集合識別碼為 "bot-storage"。我們會在下面的編碼範例中使用這些值。

The screenshot shows the Azure Cosmos DB Keys blade. On the left, there's a sidebar with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quick start, Data Explorer, and Settings. Under Settings, the "Keys" option is highlighted with a red box. The main area displays the "Read-write Keys" tab. It shows the "URI" as "https://[REDACTED].documents.azure.com:443/" and two key fields: "PRIMARY KEY" (containing "Nfy[REDACTED]") and "SECONDARY KEY" (containing "G[REDACTED]"). Below these are the "PRIMARY CONNECTION STRING" and "SECONDARY CONNECTION STRING", both starting with "AccountEndpoint=https://[REDACTED]/".

3. 您資料庫設定的 [金鑰] 索引標籤中會提供端點 URI 和金鑰。本文稍後設定您的程式碼時，需要使用這些值。

新增組態資訊

我們要新增 Cosmos DB 儲存體的組態資料簡短又簡單，當您的 Bot 變得複雜時，您可以使用相同方法新增其他組態設定。此範例使用上述範例中的 Cosmos DB 資料庫和集合名稱。

- C#
- JavaScript

EchoBot.cs

```
public class EchoBot : ActivityHandler
{
    private const string CosmosServiceEndpoint = "<your-cosmos-db-URI>";
    private const string CosmosDBKey = "<your-cosmos-db-account-key>";
    private const string CosmosDBDatabaseName = "bot-cosmos-sql-db";
    private const string CosmosDBCollectionName = "bot-storage";
    ...
}
```

安裝套件

確定您有 Cosmos DB 所需的套件

- C#
- JavaScript

```
Install-Package Microsoft.Bot.Builder.Azure
```

實作

- [C#](#)
- [JavaScript](#)

下列範例程式碼會使用與上面提供的[記憶體儲存體](#)範例相同的 Bot 程式碼執行。下列程式碼片段示範如何實作 'myStorage' 的 Cosmos DB 儲存體，以取代本機記憶體儲存體。記憶體儲存體已遭到註解排除，並以對 Cosmos DB 的參考取代。

EchoBot.cs

```
using System;
...
using Microsoft.Bot.Builder.Azure;
...
public class EchoBot : ActivityHandler
{
    // Create local Memory Storage - commented out.
    // private static readonly MemoryStorage _myStorage = new MemoryStorage();

    // Replaces Memory Storage with reference to Cosmos DB.
    private static readonly CosmosDbStorage _myStorage = new CosmosDbStorage(new CosmosDbStorageOptions
    {
        AuthKey = CosmosDBKey,
        CollectionId = CosmosDBCollectionName,
        CosmosDBEndpoint = new Uri(CosmosServiceEndpoint),
        DatabaseId = CosmosDBDatabaseName,
    });

    ...
}
```

啟動 Bot

在本機執行您的 Bot。

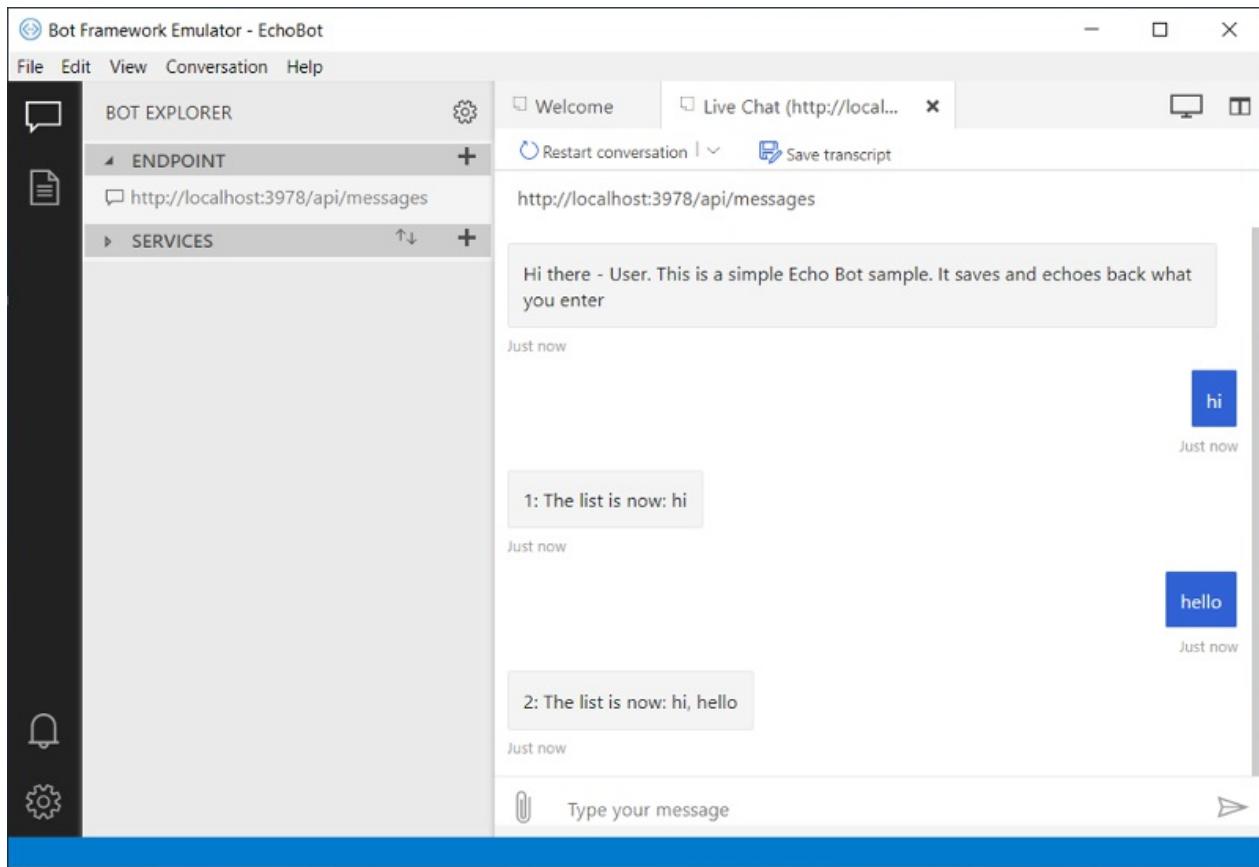
使用 Bot Framework Emulator 測試您的 Bot

現在啟動 Bot Framework Emulator 並連線到 Bot：

1. 按一下模擬器 [歡迎使用] 索引標籤中的 [建立新的 Bot 設定] 連結。
2. 以您啟動 Bot 時顯示在網頁上的資訊，填妥欄位以連線到 Bot。

與您的 Bot 互動

傳送訊息給 Bot，Bot 就會列出所收到的訊息。



檢視資料

在您執行 Bot 並儲存資訊之後，我們可以在 Azure 入口網站的 [資料總管] 索引標籤下檢視所儲存的資料。

The screenshot shows the Azure portal with the 'Cosmos DB' blade open. The left sidebar has 'Data Explorer' selected. The main area shows the 'SQL API' blade with a query result for 'UtteranceLog' documents. One document is shown with its JSON content:

```
1 {
2     "id": "UtteranceLog",
3     "realId": "UtteranceLog",
4     "document": {
5         "UtteranceList": [
6             "Hello",
7             "Testing Cosmos Data Base",
8             "Looks like it works! "
9         ],
10        "_rid": "z28nAIUUGAAQAAAAAAA=",
11        "_self": "dbs/z28nAa=/colls/z28nAIUUGAA/doc",
12        "_etag": "\"6b00ab19-0000-0000-0000-5b565b8b0",
13        "_attachments": "attachments/",
14        "_ts": 1532386187
15    }
16 }
```

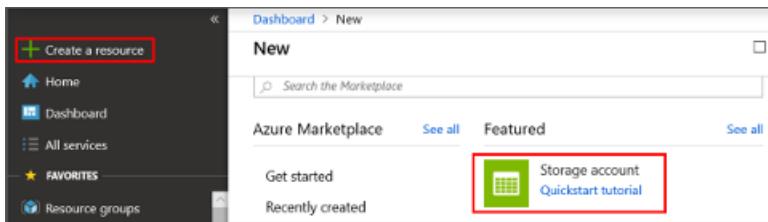
使用 Blob 儲存體

Azure Blob 儲存體是 Microsoft 針對雲端推出的物件儲存體解決方案。Blob 儲存體已針對儲存大量非結構化物件資料 (例如文字或二進位資料) 最佳化。

建立 Blob 儲存體帳戶

若要在 Bot 中使用 Blob 儲存體，您需要先設定一些事項，再進入程式碼。

1. 在新的瀏覽器視窗中，登入 [Azure 入口網站](#)。



2. 按一下 [建立資源] > [儲存體] > [儲存體帳戶 - Blob、檔案、資料表、佇列]。

A screenshot of the 'Create storage account' wizard. The left sidebar shows 'Create a resource' and a list of services. The main form is titled 'Create storage account'. It has sections for 'PROJECT DETAILS' (Subscription: Visual Studio Enterprise with MSDN, Resource group: <Your-Resource-Group-Name>, Create new) and 'INSTANCE DETAILS' (Storage account name: <Your-Account-Name>, Location: West US 2, Performance: Standard selected, Account kind: StorageV2 (general purpose v2)). At the bottom are 'Review + create', 'Previous', and 'Next : Advanced >' buttons.

3. 在 [新增帳戶] 頁面中，輸入儲存體帳戶的 [名稱]，針對 [帳戶種類] 選取 [Blob 儲存體]，提供 [位置]、[資源群組] 和 [訂用帳戶] 資訊。

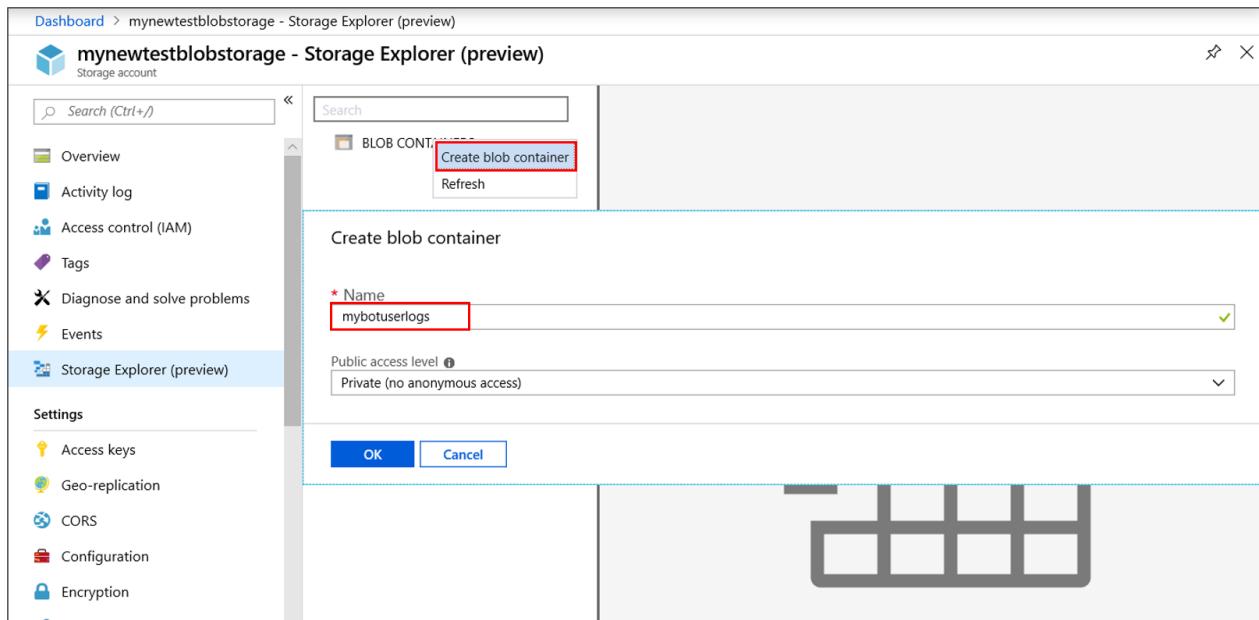
4. 然後按一下 [檢閱 + 建立]。

5. 驗證成功後，按一下 [建立]。

建立 Blob 儲存體容器

建立 Blob 儲存體帳戶之後，開啟此帳戶，做法如下：

1. 選取資源。
2. 現在使用儲存體總管 (預覽)「開啟」



3. 以滑鼠右鍵按一下 [Blob 容器], 選取 [建立 Blob 容器]。
4. 新增名稱。您將使用此名稱作為 "your-blob-storage-container-name" 的值, 以供存取您的 Blob 儲存體帳戶。

新增組態資訊

尋找您為 Bot 設定 Blob 儲存體所需的 Blob 儲存體金鑰, 如上所示:

1. 在 Azure 入口網站中, 開啟 Blob 儲存體帳戶, 然後選取 [設定] > [存取金鑰]。

我們將使用 key1_連接字串_作為 "your-blob-storage-account-string" 值, 以供存取您的 Blob 儲存體帳戶。

安裝套件

如果先前未安裝使用 Cosmos DB, 請安裝下列套件。

- C#
- JavaScript

```
Install-Package Microsoft.Bot.Builder.Azure
```

實作

- C#
- JavaScript

EchoBot.cs

```
using Microsoft.Bot.Builder.Azure;
```

更新可將 "myStorage" 指向現有 Blob 儲存體帳戶的程式碼行。

EchoBot.cs

```
private static readonly AzureBlobStorage _myStorage = new AzureBlobStorage("<your-blob-storage-account-string>", "<your-blob-storage-container-name>");
```

將儲存體設定為指向 Blob 儲存體帳戶後，Bot 程式碼現在會儲存和擷取 Blob 儲存體中的資料。

啟動 Bot

在本機執行您的 Bot。

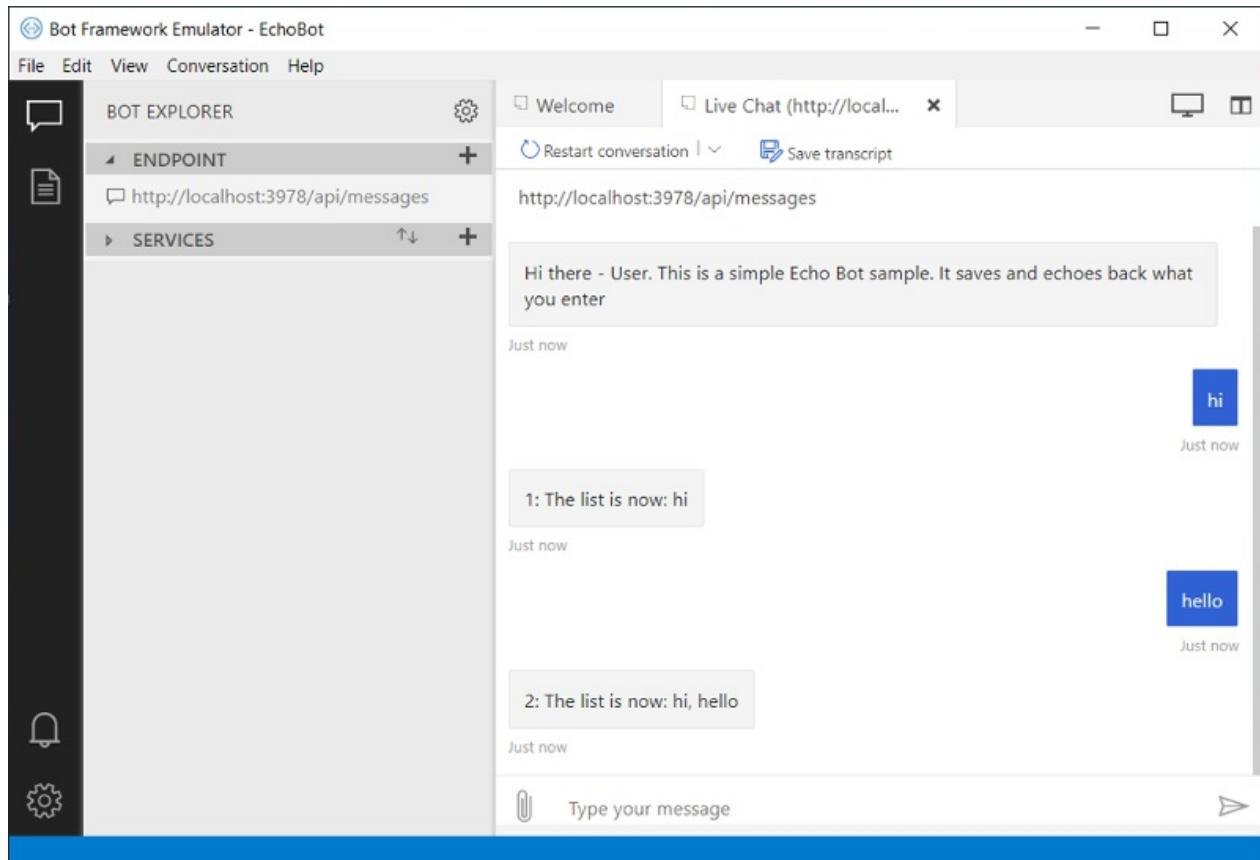
啟動模擬器並且連線至您的 Bot

接下來，請啟動模擬器，然後在模擬器中連線至您的 Bot：

1. 按一下模擬器 [歡迎使用] 索引標籤中的 [建立新的 Bot 設定] 連結。
2. 以您啟動 Bot 時顯示在網頁上的資訊，填妥欄位以連線到 Bot。

與您的 Bot 互動

傳送訊息給 Bot，Bot 就會列出所收到的訊息。



檢視資料

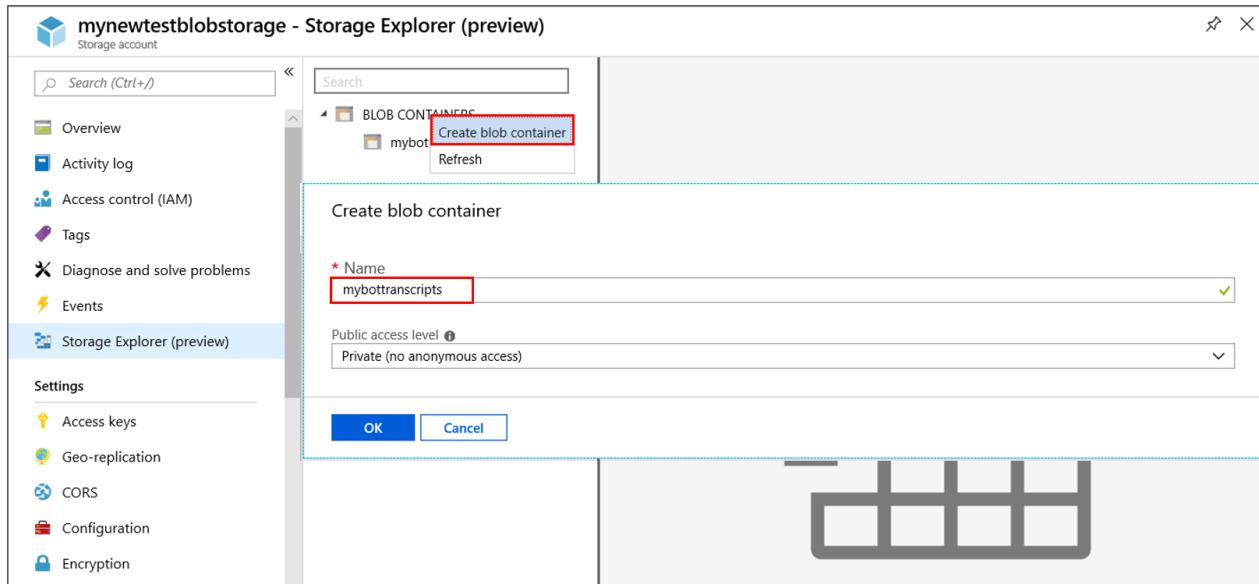
在您執行 Bot 並儲存您的資訊之後，我們可以在 Azure 入口網站的 [儲存體總管] 索引標籤下檢視。

Blob 文字記錄儲存體

Azure Blob 文字記錄儲存體會提供一個特製化儲存體選項，讓您輕鬆地以記錄的文字記錄形式儲存和擷取使用者對話。Azure Blob 文字記錄儲存體特別適合用於自動擷取使用者輸入，以便在對 Bot 效能進行偵錯時檢查。

設定

Azure Blob 文字記錄儲存體可以使用遵循上面「建立 Blob 儲存體帳戶」和「新增組態資訊」小節中詳述的步驟所建立的相同 Blob 儲存體帳戶。我們現在會新增容器來保留我們的文字記錄。



- 開啟 Azure Blob 儲存體帳戶。
- 按一下 [儲存體總管]。
- 以滑鼠右鍵按一下 [Blob 容器]，然後選取 [建立 Blob 容器]。
- 輸入文字記錄容器的名稱，然後選取 [確定]。(我們輸入了 mybottranscripts)

實作

下列程式碼會將文字記錄儲存體指標 `_myTranscripts` 連到新的 Azure Blob 文字記錄儲存體帳戶。若要使用新的容器名稱 建立此連結，請在 Blob 儲存體內建立新容器來保存文字記錄檔。

echoBot.cs

```
using Microsoft.Bot.Builder.Azure;

public class EchoBot : ActivityHandler
{
    ...

    private readonly AzureBlobTranscriptStore _myTranscripts = new AzureBlobTranscriptStore("<your-blob-
transcript-storage-account-string>", "<your-blob-transcript-container-name>");

    ...
}
```

在 Azure blob 文字記錄中儲存使用者對話

在 Blob 容器可用於儲存文字記錄之後，您可以開始保留使用者與 Bot 的對話。這些交談稍後可當作偵錯工具，查看使用者與 Bot 的互動方式。每個模擬器 [重新開始交談] 會啟始建立新的文字記錄交談清單。下列程式碼會在預存文字記錄檔內保留使用者交談輸入。

- 使用 `LogActivityAsync` 可儲存目前的文字記錄。
- 使用 `ListTranscriptsAsync` 可擷取已儲存的文字記錄。在此範例程式碼中，每個預存文字記錄的識別碼會儲存在名為 "storedTranscripts" 的清單中。這份清單稍後用來管理我們保留的預存 Blob 文字記錄數目。

echoBot.cs

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    await _myTranscripts.LogActivityAsync(turnContext.Activity);

    List<string> storedTranscripts = new List<string>();
    PagedResult<Microsoft.Bot.Builder.TranscriptInfo> pagedResult = null;
    var pageSize = 0;
    do
    {
        pagedResult = await _myTranscripts.ListTranscriptsAsync("emulator", pagedResult?.ContinuationToken);
        pageSize = pagedResult.Items.Count();

        // transcript item contains ChannelId, Created, Id.
        // save the channelIds found by "ListTranscriptsAsync" to a local list.
        foreach (var item in pagedResult.Items)
        {
            storedTranscripts.Add(item.Id);
        }
    } while (pagedResult.ContinuationToken != null);

    ...
}
```

管理預存 Blob 文字記錄

雖然預存文字記錄可以作為偵錯工具，但經過一段時間，預存文字記錄可能成長到大於您可保留的數目。以下包含的其他程式碼會使用 `DeleteTranscriptAsync`，從 Blob 文字記錄存放區中移除所有文字記錄（但最後三個擷取的文字記錄項目除外）。

echoBot.cs

```

protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    await _myTranscripts.LogActivityAsync(turnContext.Activity);

    List<string> storedTranscripts = new List<string>();
    PagedResult<Microsoft.Bot.Builder.TranscriptInfo> pagedResult = null;
    var pageSize = 0;
    do
    {
        pagedResult = await _myTranscripts.ListTranscriptsAsync("emulator", pagedResult?.ContinuationToken);
        pageSize = pagedResult.Items.Count();

        // transcript item contains ChannelId, Created, Id.
        // save the channelIds found by "ListTranscriptsAsync" to a local list.
        foreach (var item in pagedResult.Items)
        {
            storedTranscripts.Add(item.Id);
        }
    } while (pagedResult.ContinuationToken != null);

    // Manage the size of your transcript storage.
    for (int i = 0; i < pageSize; i++)
    {
        // Remove older stored transcripts, save just the last three.
        if (i < pageSize - 3)
        {
            string thisTranscriptId = storedTranscripts[i];
            try
            {
                await _myTranscripts.DeleteTranscriptAsync("emulator", thisTranscriptId);
            }
            catch (System.Exception ex)
            {
                await turnContext.SendActivityAsync("Debug Out: DeleteTranscriptAsync had a problem!");
                await turnContext.SendActivityAsync("exception: " + ex.Message);
            }
        }
    }
    ...
}

```

以下連結提供關於 [Azure Blob 文字記錄儲存體](#)的詳細資訊

其他資訊

使用 eTag 管理並行

在 Bot 程式碼範例中，我們會將每個 `IStoreItem` 的 `eTag` 屬性設定為 `*`。存放區物件的 `eTag` (實體標記) 成員在 Cosmos DB 中用來管理並行作業。如果另一個 Bot 執行個體變更了 Bot 所寫入的同一個儲存體中的物件，`eTag` 會告知資料庫該採取什麼動作。

最後寫入者優先 - 允許覆寫

星號 (`*`) 的 `eTag` 屬性值指出最後寫入者優先。建立新的資料存放區時，您可以將屬性的 `eTag` 設定為 `*`，以指出您之前沒有儲存過正在寫入的資料或您想要最後寫入的資料覆寫之前已儲存的任何屬性。如果您的 Bot 沒有並行的問題，針對正在寫入的任何資料，將其 `eTag` 屬性設定為 `*`，以允許覆寫。

維護並行和防止覆寫

將資料儲存到 Cosmos DB 時，如果您想要防止並行存取屬性，以及避免覆寫另一個 Bot 執行個體所做的變更，請對 `eTag` 使用 `*` 以外的值。當 Bot 嘗試儲存狀態資料且 `eTag` 與儲存體中的 `eTag` 值不同時，Bot 會收到含有 `etag conflict key=` 訊息的錯誤回應。

根據預設，每次 Bot 寫入到儲存體物件時，Cosmos DB 存放區會檢查其 `eTag` 屬性是否相同，然後在每次寫入之後將該屬性更新為新的唯一值。如果寫入的 `eTag` 屬性與儲存體中的 `eTag` 不相符，這表示另一個 Bot 或執行緒已變更資料。

例如，假設您想要 Bot 編輯已儲存的附註，但不想覆寫另一個 Bot 執行個體所作的變更。如果另一個 Bot 執行個體已進行編輯，則您想要使用者編輯最新更新的版本。

- [C#](#)
- [JavaScript](#)

首先，建立會實作 `IStoreItem` 的類別。

EchoBot.cs

```
public class Note : IStoreItem
{
    public string Name { get; set; }
    public string Contents { get; set; }
    public string ETag { get; set; }
}
```

接下來，建立儲存體物件來建立初始附註，並將物件新增到存放區。

EchoBot.cs

```
// create a note for the first time, with a non-null, non-* ETag.
var note = new Note { Name = "Shopping List", Contents = "eggs", ETag = "x" };

var changes = Dictionary<string, object>();
{
    changes.Add("Note", note);
};
await NoteStore.WriteAsync(changes, cancellationToken);
```

然後，稍後存取並更新附註，將它的 `eTag` 保持為從存放區讀取的原狀。

EchoBot.cs

```
var note = NoteStore.ReadAsync<Note>("Note").Result?.FirstOrDefault().Value;

if (note != null)
{
    note.Contents += ", bread";
    var changes = new Dictionary<string, object>();
    {
        changes.Add("Note1", note);
    };
    await NoteStore.WriteAsync(changes, cancellationToken);
}
```

如果存放區中的附註在您寫入變更之前已經更新，對 `Write` 的呼叫將會擲回例外狀況。

若要維護並行，一律從儲存體讀取屬性，然後修改所讀取的屬性，如此即可維護 `eTag`。如果您從存放區讀取使用者資料，回應將會包含 `eTag` 屬性。如果您變更資料並將更新的資料寫入存放區，則您的要求應該包含 `eTag` 屬性且它必須指定與您之前讀取相同的值。不過，寫入其 `eTag` 設定為 `*` 的物件，將允許寫入可覆寫任何其他變更。

後續步驟

既然您已經知道如何直接從儲存體讀取和寫入，現在我們來看看如何使用狀態管理員來為您執行這些動作。

使用交談和使用者屬性儲存狀態

透過 Azure Bot 服務將驗證新增至您的 Bot

2019/5/14 • [Edit Online](#)

適用於:  SDK v4  SDK v3

Azure Bot 服務和 v4 SDK 包含全新的 Bot 驗證功能，並提供相關功能，讓您輕鬆開發可對 Azure AD (Azure Active Directory)、GitHub、Uber 等不同識別提供者驗證使用者的 Bot。這些功能也讓部分用戶端無須進行_神奇代碼驗證 (Magic code verification)_，而改善使用者體驗。

在此之前，您的 Bot 必須加入 OAuth 控制器和登入連結、儲存目標用戶端識別碼及密碼，並執行使用者權杖管理作業。Bot 會要求使用者登入網站，繼而產生使用者可用來驗證其身分識別的_神奇代碼_。

現在，Bot 開發人員無須際遇主控 OAuth 控制器或管理權杖的生命週期，這些工作現在全都可由 Azure Bot 服務執行。

這些功能包括：

- 改善通道以支援全新驗證功能 (例如新的 WebChat 和 DirectLineJS 程式庫)，無須再使用 6 位數神奇代碼驗證方式。
- 改善 Azure 入口網站，以針對不同的 OAuth 身分識別提供者新增、刪除和配置連線設定。
- 支援各種現成的身分識別提供者服務，例如：Azure AD (包括 v1 和 v2 端點)、GitHub 等等。
- 更新 C# 和 Node.js Bot Framework SDK，以支援擷取權杖、建立 OAuthCard 及處理 TokenResponse 事件。
- 如何建立能對 Azure AD 進行驗證的 Bot 範例。

您可從本文中的步驟推測如何將這類功能新增至現有的 Bot。下列範例 Bot 將展示新的驗證功能。

NOTE

驗證功能亦可搭配 BotBuilder v3 運作。不過，本文內容僅涵蓋 v4 範例程式碼。

關於此範例

您必須建立 Azure Bot 資源，且必須建立新的 Azure AD (v1 或 v2) 應用程式，讓 Bot 能夠存取 Office 365。Bot 資源會註冊您的 Bot 認證；您需要這些認證來測試驗證功能，即使您在本機執行 Bot 程式碼亦然。

IMPORTANT

每個在 Azure 中註冊的 Bot，都會被指派一個 Azure AD 應用程式。不過，此應用程式會保護通道對 Bot 的存取。每個應用程式都還額外需要一個您要讓 Bot 能夠代表使用者，進行驗證的 AAD 應用程式。

本文說明使用 Azure AD v1 或 v2 權杖連線至 Microsoft Graph 的範例 Bot。此外也說明如何建立及註冊相關聯的 Azure AD 應用程式。而在此程序中，您將使用 [Microsoft/BotBuilder-Samples GitHub](#) 存放庫中的程式碼。本文將說明這些程序。

- 建立 Bot 資源
- 建立 Azure AD 應用程式
- 向 Bot 註冊 Azure AD 應用程式
- 準備 Bot 範例程式碼

完成後，您將有一個在本機執行的 Bot 能夠對 Azure AD 應用程式回應幾項簡單工作，例如，檢查及傳送電子郵件，或者顯示您或管理員的身分。您的 Bot 必須針對 Microsoft.Graph 程式庫使用來自 Azure AD 應用程式的權杖，才能達到上述目的。您不需要發佈 Bot 以測試 OAuth 登入功能；不過，您的 Bot 將需要有效的 Azure 應用程式識別碼

和密碼。

這些驗證功能也可與其他類型的 Bot 搭配使用。不過，本文將使用僅具有註冊功能的 Bot。

網路聊天和 Direct Line 的考量

在網路聊天使用 Azure Bot 服務驗證時，必須考量幾個重要的安全性問題。

1. 防止攻擊者進行模擬而使 Bot 將其誤認為其他人。在網路聊天中，攻擊者可藉由變更他人網路聊天執行個體的使用者識別碼，來模擬其他人。

若要防止此問題發生，請使用難以猜測的使用者識別碼。當您 在 Direct Line 通道中啟用增強式驗證選項時，Azure Bot 服務可偵測並拒絕任何使用者識別碼變更。從 Direct Line 到 Bot 的訊息，一律會使用您用來初始化網路聊天的相同使用者識別碼。請注意，此功能需要以 `d1_` 開頭的使用者識別碼。

2. 請確實登入正確的使用者。使用者有兩個身分識別：通道中的身分識別，和用於識別提供者的身分識別。在網路聊天中，Azure Bot 服務可確保登入程序會在與網路聊天本身相同的瀏覽器工作階段中完成。

若要啟用這項保護，請以 Direct Line 權杖啟動網路聊天，且該權杖必須包含可裝載 Bot 網路聊天用戶端的信任網域清單。然後，在 Direct Line 組態頁面中以靜態方式指定信任網域（原始）清單。

必要條件

- [Bot 基本概念、管理狀態、dialogs 程式庫](#)、如何實作循序交談流程、如何[使用對話提示蒐集使用者輸入](#)，以及如何[重複使用對話](#)的知識。
- Azure 和 OAuth 2.0 開發的知識。
- Visual Studio 2017 或更新版本、Node.js、npm 和 git。
- 以下其中一個範例。

範例	BOTBUILDER 版本	示範
CSharp 或 JavaScript 中的 Bot 驗證	v4	OAuthCard 支援
CSharp 或 JavaScript 中的 Bot 驗證 MSGraph	v4	OAuth 2 的 Microsoft Graph API 支援

在 Azure 上建立 Bot 資源

使用 [Azure 入口網站](#)建立 Bot 通道註冊。

建立和註冊 Azure AD 應用程式

您需要一個可讓 Bot 用來連線至 Microsoft Graph API 的 Azure AD 應用程式。

針對此 Bot，您可使用 Azure AD v1 或 v2 端點。如需 v1 和 v2 端點之間差異的相關資訊，請參閱 [v1 與 v2 比較](#)，以及 [Azure AD v2.0 端點概觀](#)。

建立 Azure AD 應用程式

建立下列步驟建立新的 Azure AD 應用程式。您可以將 v1 或 v2 端點與所建立的應用程式一起使用。

TIP

您必須在具有管理員權限的租用戶中，建立並登錄 Azure AD 應用程式。

- [Azure AD v1](#)
- [Azure AD v2](#)

- 在 Azure 入口網站中開啟 [Azure Active Directory](#) 面板。如果您不在正確的租用戶中，請按一下 [切換目錄] 以切換至正確的租用戶。(如需建立租用戶的指示，請參閱[存取入口網站並建立租用戶](#)。)
- 開啟 [應用程式註冊] 面板。
- 在 [應用程式註冊] 面板中，按一下 [新增應用程式註冊]。
- 填寫必要欄位並建立應用程式註冊。
 - 為您的應用程式命名。
 - 將 [應用程式類型] 設為 [Web 應用程式/API]。
 - 將 [登入 URL] 設為 `https://token.botframework.com/.auth/web/redirect`。
 - 按一下頁面底部的 [新增]。
 - 建立後，即會顯示在 [註冊的應用程式] 窗格中。
 - 記錄 [應用程式識別碼] 值。稍後當您向 Bot 註冊 Azure AD 應用程式時，您將使用此值作為_用戶端識別碼_。
- 按一下 [設定] 以設定應用程式。
- 按一下 [金鑰] 以開啟 [金鑰] 面板。
 - 在 [密碼] 下方建立 `BotLogin` 金鑰。
 - 將其 [持續時間] 設為 [永不過期]。
 - 按一下 [儲存] 並記錄金鑰值。稍後當您向 Bot 註冊 Azure AD 應用程式時，您將使用此值作為_用戶端密碼_。
 - 關閉 [金鑰] 面板。
- 按一下 [必要權限] 以開啟 [必要權限] 面板。
 - 按一下 [新增]。
 - 按一下 [選取 API]，然後選取 [Microsoft Graph]，再按一下 [選取]。
 - 按一下 [選取權限]。選擇您的應用程式將使用的委派權限。

NOTE

只要是標記為 [需要系統管理員] 的權限，則使用者和租用戶管理員皆必須登入，才能將 Bot 隔離在這些權限之外。

選取下列 Microsoft Graph 委派的權限：

- 讀取所有使用者的基本個人資料
 - 讀取使用者的郵件
 - 以使用者身分傳送郵件
 - 登入及讀取使用者設定檔
 - 檢視所有使用者的基本個人資料
 - 檢視使用者的電子郵件地址
- 按一下 [選取]，然後按一下 [完成]。
 - 關閉 [必要權限] 面板。

您現在已完成 Azure AD v1 應用程式設定。

[向 Bot 註冊 Azure AD 應用程式](#)

下一步是向 Bot 註冊您已建立的 Azure AD 應用程式。

- [Azure AD v1](#)
- [Azure AD v2](#)

1. 在 [Azure 入口網站](#) 瀏覽至 Bot 的資源頁面。
2. 按一下 [設定]。
3. 在靠近頁面底部的 [OAuth 連線設定] 下方，按一下 [新增設定]。
4. 填寫表單，如下所示：
 - a. 若為 [名稱] 欄位，請輸入連線的名稱。您將在 Bot 程式碼中使用此名稱。
 - b. 若為 [服務提供者]，請選取 [Azure Active Directory]。選取此項目後，Azure AD 專用的欄位隨即顯示。
 - c. 若為 [用戶端識別碼] 欄位，請輸入您為 Azure AD v1 應用程式記錄的應用程式識別碼。
 - d. 若為 [用戶端密碼] 欄位，請輸入您為應用程式 `BotLogin` 金鑰記錄的金鑰。
 - e. 若為 [授與類型] 欄位，請輸入「`authorization_code`」。
 - f. 若為 [登入 URL] 欄位，請輸入「`https://login.microsoftonline.com`」。
 - g. 若為 [租用戶識別碼] 欄位，請輸入 Azure Active Directory 的租用戶識別碼，例如：`microsoft.com` 或 `common`。
這將會是與可驗證之使用者相關聯的租用戶。若要允許任何人透過 Bot 驗證身分，請使用 `common` 租用戶。
 - h. 針對 [資源 URL]，輸入 `https://graph.microsoft.com/`。
 - i. 請將 [範圍] 欄位保留空白。
5. 按一下 [檔案]。

NOTE

這些值可讓應用程式透過 Microsoft Graph API 存取 Office 365 資料。

測試連線

1. 按一下連線項目以開啟您剛剛建立的連線。
2. 按一下 [服務提供者連線設定] 窗格頂端的 [測試連線]。
3. 第一次將開啟新的瀏覽器索引標籤，其中列出應用程式要求的權限，並提示您接受要求。
4. 按一下 [接受]。
5. 這應會將您重新導向至 [測試對 <your-connection-name> 的連線已成功] 頁面。

您現在可在 Bot 程式碼中使用此連線名稱，以擷取使用者權杖。

準備 Bot 程式碼

您需要 Bot 的應用程式識別碼和密碼來完成此程序。若要擷取您的 Bot 應用程式識別碼和密碼：

1. 在 [Azure 入口網站](#) 中，巡覽至您在其中建立通道註冊 Bot 的資源群組。
2. 開啟 [部署] 窗格，然後開啟 Bot 的部署。
3. 開啟 [輸入] 窗格，並複製 Bot 的 `appId` 和 `appSecret` 值。

- [C#](#)
- [JavaScript](#)

1. 從您想要使用的 GitHub 存放庫複製：[Bot 驗證](#) 或 [Bot 驗證 MSGraph](#)。

2. 遵循 GitHub 讀我檔案頁面上的指示，了解如何執行該特定 Bot。

3. 更新 `appsettings.json`：

- 將 `ConnectionName` 設定為您新增至 Bot 的 OAuth 連線名稱。
- 將 `MicrosoftAppId` 和 `MicrosoftAppPassword` 設定為您 Bot 的應用程式識別碼和應用程式祕密。

視 Bot 祕密中的字元而定，您可能需要讓 XML 逸出該密碼。例如，任何 & 符號都必須編碼為 &

◦

```
{  
  "MicrosoftAppId": "",  
  "MicrosoftAppPassword": "",  
  "ConnectionName": ""  
}
```

如果您不知道如何取得 **Microsoft 應用程式識別碼** 和 **Microsoft 應用程式密碼** 值，您可以：

- [如下所述](#) 建立新密碼
- 從[這裡所述](#)的部署，擷取透過 [Bot 通道註冊] 佈建的 [Microsoft 應用程式識別碼] 和 [Microsoft 應用程式密碼]

NOTE

您現在可將此 Bot 程式碼發佈至 Azure 訂用帳戶（以滑鼠右鍵按一下專案，然後選擇 [發佈]），但此動作在本文的範例中並非必要。在 Azure 入口網站中配置 Bot 時，您必須進行發佈設定，其應使用您所用的應用程式和主控方案。

測試 Bot

1. 如果您尚未安裝 [Bot Framework Emulator](#)，請進行安裝。

2. 在您的電腦本機執行範例。

3. 啟動模擬器、連線到您的 Bot，然後傳送訊息。

- 當您連線到 Bot 時，您必須提供 Bot 的應用程式識別碼和密碼。
- 輸入「`help`」以檢視適用於 Bot 的可用命令清單，以及測試驗證功能。
- 登入後一直到登出前，您都不需要再次提供認證。
- 若要登出並取消驗證，請輸入「`logout`」。

NOTE

Bot 驗證需要使用 Bot 連接器服務。該服務將針對您的 Bot 存取 Bot 通道註冊資訊。

- [Bot 驗證](#)
- [Bot 驗證 MSGraph](#)

在 **Bot 驗證** 範例中，對話的設計訴求是要在使用者登入後擷取使用者權杖。

<http://localhost:3978/api/messages>

The screenshot shows a messaging interface with the following sequence of messages:

- User message: "hi" (A minute ago)
- Bot response: "Please Sign In" (A minute ago)
 - Bot button: "Sign In"
- User message: "You are now logged in."
- User message: "Would you like to view your token?" (A minute ago)
- Bot response: "Yes" (A minute ago)
- User message: "Thank you."
- Bot message: "Here is your token" (A minute ago)
 - Token value: eyJ0eXAiOiJKV1QiLCJub25jZSI6IkFRQUJBQUFBQUFDRWZleFh4amFtUWlzt2VHUTFD6N4lQixLQtuXYyJ08lOrgaYfwd37Aq6XFHqa-5xq3PJ0VdbbaEnc92dRwXBYfoOph6L_bkuNCdOuVpGEYynUnQ0o0cQYVdK-2p76Kfp26ES2dvofTG81cizx2sF5j4WQn6a7HzkX6fMkzzXy0M15480jIBICVz3EQopRI
- User message: "A minute ago"
- User message: "You are now logged in."
- User message: "Would you like to view your token?" (A minute ago)
- Bot response: "No" (A minute ago)
- User message: "A minute ago"
- User message: "Thank you."
- User message: "A minute ago"
- Bot button: "logout" (A minute ago)
- User message: "A minute ago"
- User message: "You have been signed out."

其他資訊

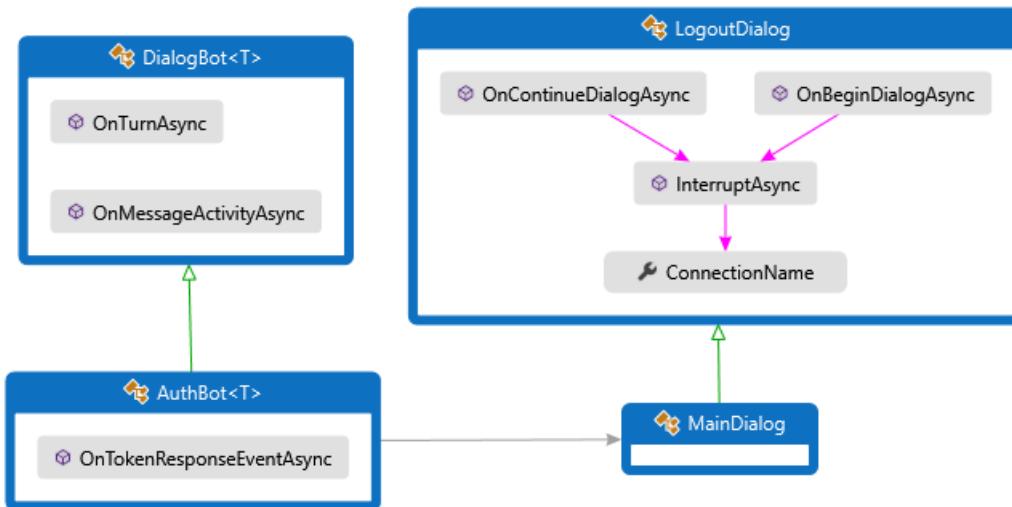
使用者要求 Bot 執行需要使用者登入的工作時，Bot 可使用 `OAuthPrompt` 起始擷取特定連線的權杖。`OAuthPrompt` 會建立權杖擷取流程，其中包含：

1. 查看 Azure Bot Service 是否已經具有可供目前使用者和連線使用的權杖。如果有權杖，則會傳回權杖。
2. 如果 Azure Bot Service 沒有快取的權杖，則會建立 `OAuthCard`，這是使用者可以按一下的登入按鈕。
3. 在使用者按一下 `OAuthCard` 登入按鈕之後，Azure Bot Service 會將使用者的權杖傳直接送給 Bot，或者會向使用者顯示要在聊天視窗中輸入的 6 位數驗證碼。
4. 如果使用者看到驗證碼，則 Bot 會將此驗證碼換成使用者的權杖。

下列各節說明範例如何實作一些常見的驗證工作。

使用 OAuth 提示來登入使用者和取得權杖

- [C#](#)
- [JavaScript](#)



Dialogs\MainDialog.cs

將 OAuth 提示新增至其建構函式中的 **MainDialog**。在此，已從 **appsettings.json** 檔案擷取連線名稱值。

```
AddDialog(new OAuthPrompt(  
    nameof(OAuthPrompt),  
    new OAuthPromptSettings  
    {  
        ConnectionName = ConnectionName,  
        Text = "Please Sign In",  
        Title = "Sign In",  
        Timeout = 300000, // User has 5 minutes to login (1000 * 60 * 5)  
    }));
});
```

在對話步驟中，使用 `BeginDialogAsync` 來啟動 OAuth 提示，其會要求使用者登入。

- 如果使用者已經登入，這會產生一個權杖回應事件，但不會提示使用者。
- 否則會提示使用者登入。Azure Bot Service 會在使用者嘗試登入後傳送權杖回應事件。

```
return await stepContext.BeginDialogAsync(nameof(OAuthPrompt), null, cancellationToken);
```

在下列對話步驟中，檢查上一個步驟的結果中是否有權杖存在。如果不是 Null，則表示使用者登入成功。

```
// Get the token from the previous step. Note that we could also have gotten the  
// token directly from the prompt itself. There is an example of this in the next method.  
var tokenResponse = (TokenResponse)stepContext.Result;  
if (tokenResponse != null)  
{
```

等候 TokenResponseEvent

當您啟動 OAuth 提示時，其會等候權杖回應事件，而且會從中擷取使用者的權杖。

- [C#](#)
- [JavaScript](#)

Bots\AuthBot.cs

AuthBot 衍生自 `ActivityHandler` 並明確處理權杖回應事件活動。在此，我們會持續進行作用中的對話，讓 OAuth 提示得以處理事件和擷取權杖。

```
protected override async Task OnTokenResponseEventAsync(ITurnContext<IEventActivity> turnContext,
CancellationToken cancellationToken)
{
    Logger.LogInformation("Running dialog with Token Response Event Activity.");

    // Run the Dialog with the new Token Response Event Activity.
    await Dialog.Run(turnContext, ConversationState.CreateProperty<DialogState>(nameof(DialogState)),
cancellationToken);
}
```

登出使用者

最佳做法是讓使用者明確登出，而不是依賴連接逾時。

- [C#](#)
- [JavaScript](#)

Dialogs\LogoutDialog.cs

```

protected string ConnectionName { get; }

protected override async Task<DialogTurnResult> OnBeginDialogAsync(DialogContext innerDc, object options,
CancellationToken cancellationToken = default(CancellationToken))
{
    var result = await InterruptAsync(innerDc, cancellationToken);
    if (result != null)
    {
        return result;
    }

    return await base.OnBeginDialogAsync(innerDc, options, cancellationToken);
}

protected override async Task<DialogTurnResult> OnContinueDialogAsync(DialogContext innerDc, CancellationToken
cancellationToken = default(CancellationToken))
{
    var result = await InterruptAsync(innerDc, cancellationToken);
    if (result != null)
    {
        return result;
    }

    return await base.OnContinueDialogAsync(innerDc, cancellationToken);
}

private async Task<DialogTurnResult> InterruptAsync(DialogContext innerDc, CancellationToken cancellationToken
= default(CancellationToken))
{
    if (innerDc.Context.Activity.Type == ActivityTypes.Message)
    {
        var text = innerDc.Context.Activity.Text.ToLowerInvariant();

        if (text == "logout")
        {
            // The bot adapter encapsulates the authentication processes.
            var botAdapter = (BotFrameworkAdapter)innerDc.Context.Adapter;
            await botAdapter.SignOutUserAsync(innerDc.Context, ConnectionName, null, cancellationToken);
            await innerDc.Context.SendActivityAsync(MessageFactory.Text("You have been signed out."),
cancellationToken);
            return await innerDc.CancelAllDialogsAsync(cancellationToken);
        }
    }

    return null;
}

```

進階閱讀

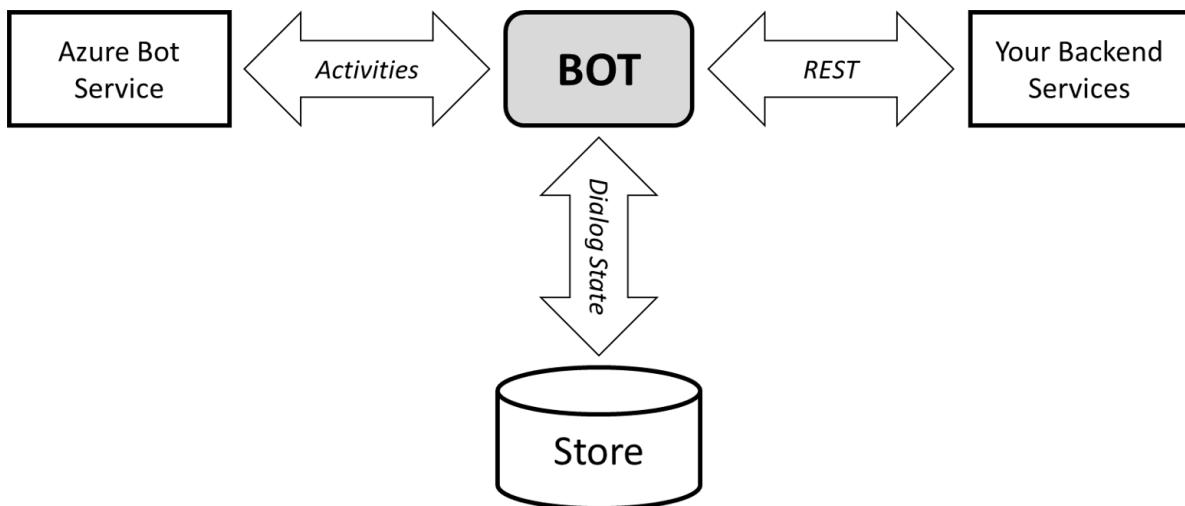
- [Bot Framework 其他資源](#)包含其他支援的連結。
- [Bot Framework SDK](#)存放庫提供更多與 Bot Builder SDK 相關聯的存放庫、範例、工具和規格的詳細資訊。

為您的 Bot 實作自訂儲存體

2019/5/10 • [Edit Online](#)

適用於: SDK v4 SDK v3

Bot 的互動可分成三個方面:首先, 交換活動與 Azure Bot Service, 接著, 利用 Store 載入及儲存對話方塊狀態, 最後是 Bot 完成其作業所需處理的任何其他後端服務。



必要條件

- 本文中所使用的完整範例程式碼皆可在此處找到:[C# 範例](#)。

在本文中, 我們將探索 Bot 與 Azure Bot Service 和 Store 的互動相關語意。

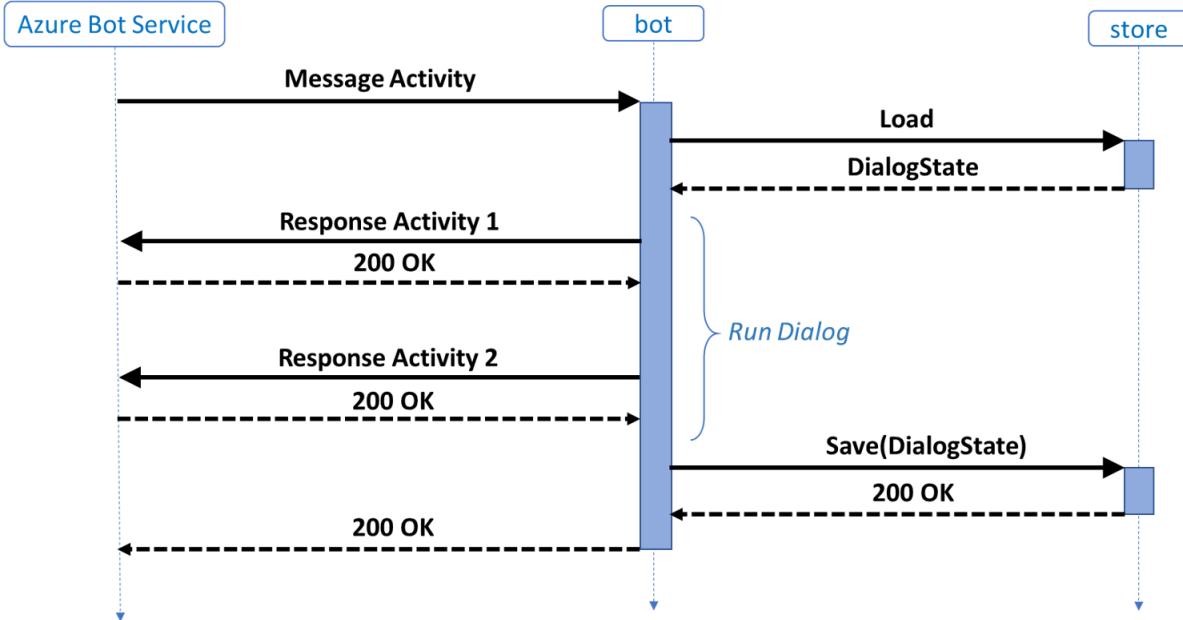
Bot Framework 架構包含預設實作。此實作最可能符合許多應用程式的需求, 只需將這幾項與幾行初始化程式碼連在一起, 即可利用該實作。有許多範例可說明該實作。

不過, 本文目標是要描述當預設實作的語意作用不如在您的應用程式中時, 您可以怎麼做。基本要點為這是一種架構, 而不是具有固定行為的預製應用程式, 換言之, 此架構中許多機制的實作只是預設實作, 並不是唯一的實作。

具體來說, 此架構不會指定活動與 Azure Bot Service 交換之間的關聯性, 以及任何 Bot 狀態的載入和儲存, 該架構只會提供預設值。為了進一步說明這一點, 我們會開發具有不同語意的替代實作。此替代解決方案妥善置於架構中, 甚至可能更適合正在開發的應用程式。這完全視情況而定。

預設 BotFrameworkAdapter 和儲存體提供者的行為

首先, 讓我們檢閱架構套件所隨附的預設實作, 如下列循序圖所示:



在接收活動時，Bot 會載入此對話的對應狀態。然後，其會以此狀態和剛送達的活動，執行對話方塊邏輯。在執行對話方塊的程序中，會建立並立即傳送一或多個輸出活動。當對話方塊處理完成時，Bot 會儲存更新後的狀態，並以新狀態覆寫舊狀態。

下列幾件事可能會伴隨此行為而發生錯誤，值得深思熟慮。

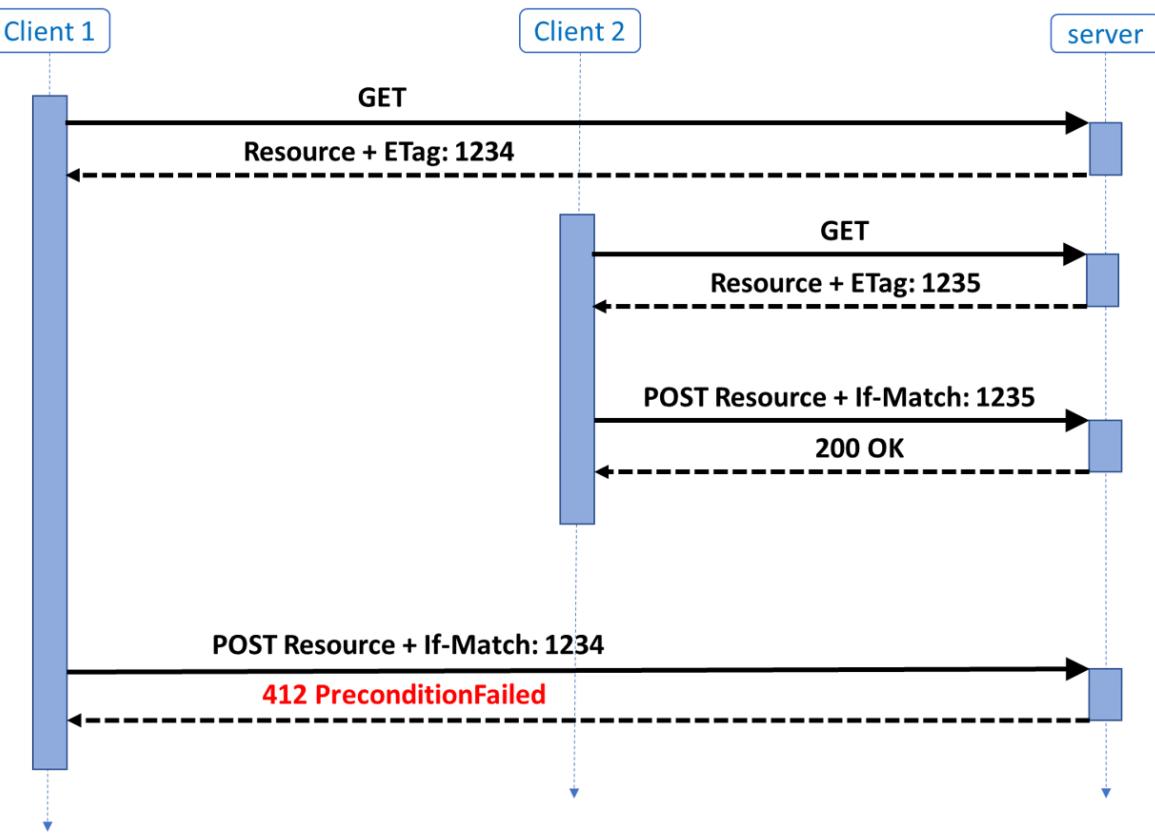
首先，如果儲存作業基於某些原因而失敗，則狀態會隱含地與通道上顯示的狀態不同步，因為看過回應的使用者會認為狀態已移往前，但並非如此。這通常比狀態成功和回應傳訊成功還要糟。這可能會影響對話設計：比方說，對話可能包含與使用者的其他不同多餘確認交換。

其次，如果實作跨多個節點水平擴充部署，則狀態可能會不小心遭到覆寫 - 這特別令人困惑，因為對話方塊可能將活動傳送給含有確認訊息的通道。以訂購披薩的 Bot 為例，如果在詢問使用者要加上哪些配料時，他們加了蘑菇，且毫不猶豫地加了起司，則在有多個執行個體正在執行的水平擴充案例中，可以將後續活動同時傳送到執行 Bot 的不同電腦。發生這種情況時，會出現所謂的「競爭現象」，也就是其中一部電腦可能會覆寫另一部電腦所寫入的狀態。不過，在我們的案例中，因為已經傳送回應，所以使用者收到了已加入蘑菇和起司的確認訊息。不幸的是，當披薩送達時，披薩只包含蘑菇或起司，而非兩者。

開放式鎖定

此解決方法是要引進一些以狀態為主的鎖定。我們將在此使用的特定樣式鎖定稱為開放式鎖定，因為我們會讓每個項目都彷彿是唯一執行中的項目，然後我們會在處理完成後偵測任何並行違規情形。這聽起來可能很複雜，但使用雲端儲存體技術和 Bot 架構中適當的擴充點，卻非常容易辦到。

我們將使用以實體標記標頭 (ETag) 為基礎的標準 HTTP 機制。請務必先了解這項機制，才能了解後面的程式碼。下圖說明此序列。



此圖說明的案例為兩個正在對某些資源執行更新的用戶端。當用戶端發出 GET 要求並從伺服器傳回資源時，其就會伴隨 ETag 標頭。ETag 標頭是不透明的值，代表資源的狀態。如果資源已變更，將會更新 ETag。當用戶端已完成其狀態更新時，它會將狀態 POST 回伺服器，並提出此要求：用戶端附加其先前在先決條件 If-Match 標頭中收到的 ETag 值。如果此 ETag 不符合伺服器上次傳回（在任何回應，傳送給任何用戶端）的值，則先決條件檢查會失敗，發生 412 先決條件失敗。對提出資源已更新的 POST 要求的用戶端而言，此失敗是一項指標。看到此失敗時，用戶端的典型行為會是再次 GET 資源、套用想要的更新，以及 POST 回資源。這第二次 POST 將會成功，當然前提是沒有其他用戶端會更新資源，而若已更新，則用戶端就必須再試一次。

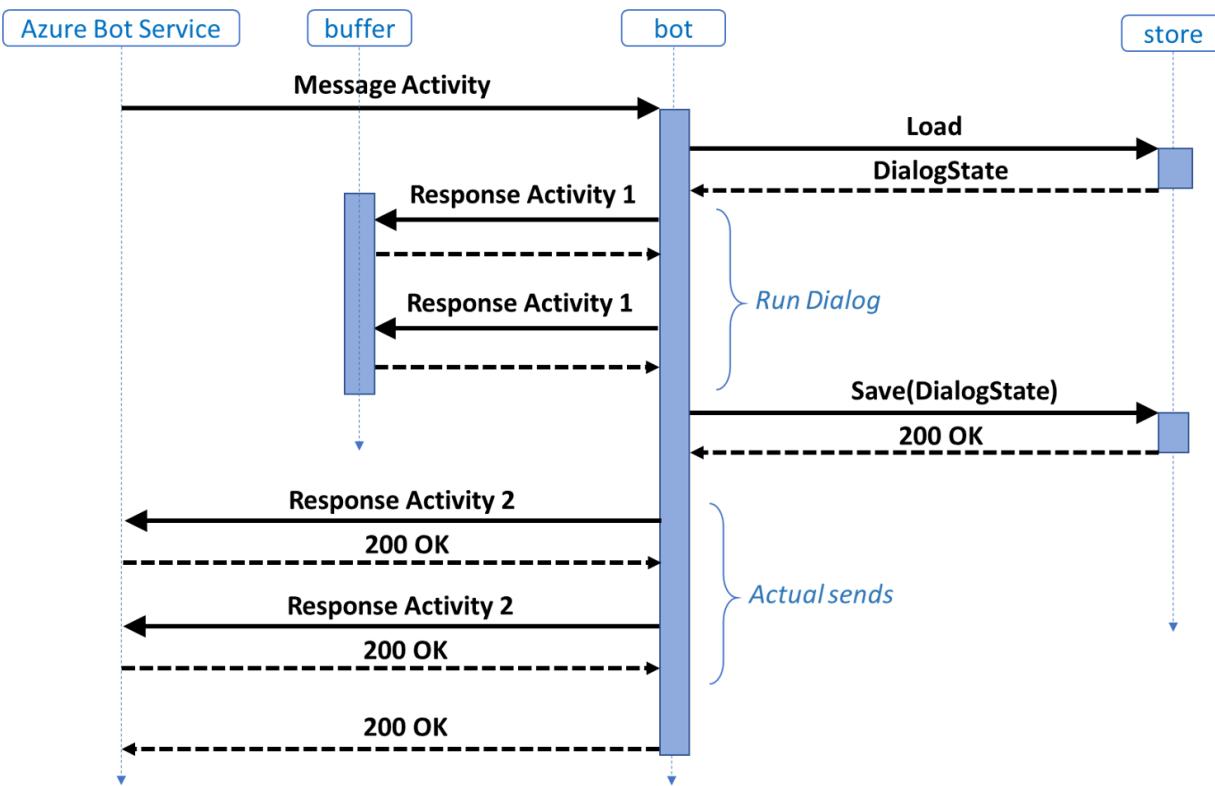
此程序稱為「開放式」，因為取得資源的用戶端會繼續執行其處理，在其他用戶端可以不受限存取資源的意義上，資源本身不會「鎖定」。直到處理完成，才能判斷用戶端之間對於資源狀態應為何的爭論。在分散式系統中，此策略通常比相反的「封閉式」方法更理想。

我們所討論的開放式鎖定機制，假設可以安全地重試程式邏輯，當然此處所要考量的重點是外部服務的呼叫發生什麼事。此處理想的解決方案是這些服務可否具有等冪性。在電腦科學中，若使用相同的輸入參數呼叫等冪作業一次以上，則不會有任何額外的作用。實作 GET、PUT 和 DELETE 的純 HTTP REST 服務符合此描述。此處的推論為直覺式：我們可能會重試處理，所以進行它所需的任何呼叫沒有額外的作用，因為在該重試過程中重新執行呼叫是件好事。基於這項討論，我們會假設我們活在理想的世界中，並顯示在本文開頭的系統圖片右側的後端服務全都是等冪性 HTTP REST 服務，之後我們只會著重於活動的交換。

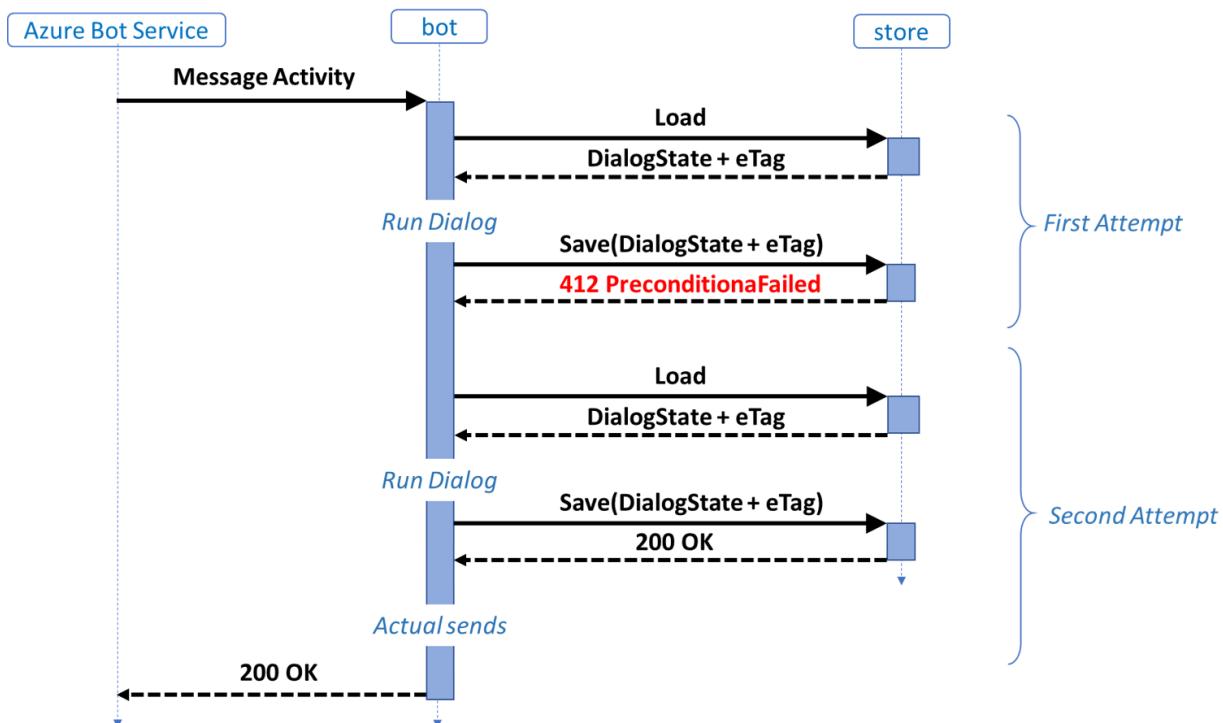
緩衝輸出活動

傳送活動不是等冪性作業，而在端對端案例中也沒有什麼特別意義。總之，活動通常只會攜帶附加至檢視或可能由文字轉換語音代理程式說出的訊息。

傳送活動時，我們想要避免傳送很多次。問題是，開放式鎖定機制要求我們儘可能重新執行我們的邏輯多次。解決方法很簡單：我們必須緩衝處理來自對話方塊的輸出活動，直到確定不會重新執行邏輯為止。直到我們有成功的儲存作業之後。我們正在尋找類似下列的流程：



假設我們可以建置以對話方塊執行為主的重試迴圈，當儲存作業發生先決條件失敗時，我們會有下列行為：



套用此機制並重新瀏覽我們先前的範例，我們就不可能看到有關訂單上比薩配料的錯誤正面確認。事實上，雖然我們可能已將部署水平擴充於多部電腦，但我們已使用開放式鎖定配置，將狀態更新有效地序列化。訂購比薩時，現在甚至可寫入新增項目的確認，以精確反映完整狀態。例如，如果使用者立即輸入「起司」，而在 Bot 有機會回覆「蘑菇」之前，現在有兩個回覆可能是「加起司的披薩」以及「加起司與蘑菇的披薩」。

查看循序圖時，我們發現回覆可能會在儲存作業成功之後遺失，不過，回覆也有可能會在端對端通訊的任何地方遺失。重點在於這不是狀態管理基礎結構可以修正的問題。這需要較高層級的通訊協定，而且有可能牽涉到通道的使用者。比方說，如果 Bot 似乎未能回覆使用者，則可合理預期使用者最後會再試一次或做出一些這類行為。因此，雖然案例偶爾有暫時性中斷很合理，但是預期使用者能夠篩選掉錯誤正面確認，或其他不想要的訊息則相當不合理。

總而言之，在我們新的自訂儲存解決方案中，我們會執行架構中的預設實作不會執行的三件事。首先，我們會使用 ETag 來偵測競爭，接著會在偵測到 ETag 失敗時重試處理，最後會緩衝處理任何輸出活動，直到儲存成功為止。本文的其餘部分說明這三個部分的實作。

實作 ETag 支援

若要支援單元測試，我們會先使用 ETag 支援，為我們新存放區定義介面。擁有介面表示我們可以撰寫兩個版本，一個用於在記憶體中執行的單元測試（不需要叫用網路），另一個用於生產環境。此介面讓使用者很容易運用我們在 ASP.NET 中擁有的相依性插入機制。

此介面包含 Load 和 Save 方法。這兩個方法擁有我們將用於狀態的關鍵。Load 會傳回資料和相關聯的 ETag。而 Save 會接受這些資料。此外，Save 會傳回 bool。此 bool 會指出 ETag 是否有相符且 Save 成功。這無意作為一般錯誤指標，而是作為先決條件失敗的特定指標，我們會將此塑造為傳回碼，而不是例外狀況，因為我們將在重試迴圈的圖形中，撰寫以此為主的控制流量邏輯。

我們希望此最低層級的儲存體項目為插入式，因此我們會確保避免對其提出任何序列化需求，不過我們想要指定內容儲存應為 JSON，如此一來存放區實作即可設定內容類型。在.NET 中，最簡單且最自然的做法是透過引數類型，具體來說，我們會輸入內容引數作為 JObject。在 JavaScript 或 TypeScript 中，這只是一般原生物件。

所產生的介面如下：

IStore.cs [!code-csharp|IStore]

針對 Azure Blob 儲存體實作此介面很簡單。

BlobStore.cs [!code-csharp|BlobStore]

如您所看，Azure Blob 儲存體正在此處進行實際工作。請注意，特定例外狀況的捕捉，以及如何轉化以符合呼叫程式碼的期望。也就是說，對於 Load，我們希望「找不到」例外狀況傳回 null，而對於 Save，「先決條件失敗」例外狀況傳回 bool。

此原始程式碼全都在對應的範例中取得，而該範例會包含記憶體存放區實作。

實作重試迴圈

迴圈的基本形狀直接衍生自循序圖中所示的行為。

接收活動時，我們會為該對話的對應狀態建立索引鍵。我們不會變更活動和對話狀態之間的關聯性，因此我們建立索引鍵的方式完全如同預設狀態實作一樣。

建立適當的索引鍵之後，我們會嘗試 Load 對應的狀態。接著執行 Bot 的對話方塊，然後嘗試 Save。如果 Save 成功，我們會傳送執行對話方塊所產生的輸出活動。否則，我們會返回並重複 Load 前的整個程序。重新進行 Load 讓我們擁有新的 ETag，所以下次 Save 很可能會成功。

所產生的 OnTurn 實作如下所示：

ScaleoutBot.cs [!code-csharp|OnMessageActivity]

請注意，我們已將對話方塊執行塑造為函式呼叫。或許更複雜的實作定義了一個介面，並且讓此相依性得以插入，但是就我們的目的而言，讓對話方塊完全位於靜態函式後面，可強調我們方法的功能性質。一般而言，組織我們的實作，使重要部分得以運作，讓我們能夠在網路上順利實作。

實作輸出活動緩衝處理

下一項需求是我們會緩衝處理輸出活動，直到已執行成功 Save 為止。將會需要自訂 BotAdapter 實作。在此程式碼中，我們會實作抽象的 SendActivity 函式，將活動新增至清單，而非傳送。我們即將主控的對話方塊還是不明白。在此特殊案例中，不支援 UpdateActivity 和 DeleteActivity 作業，所以只要從這些方法擲回「未實作」即可。我們也不在意 SendActivity 的傳回值。在需要傳送活動更新的案例中，某些通道會使用此值，例如，通道中所顯示卡

片上的停用按鈕。特別在需要狀態時，這些訊息交換會變複雜，但這不在本文的討論範圍內。自訂 BotAdapter 的完整實作看起來像這樣：

DialogHostAdapter.cs [!code-csharpDialogHostAdapter]

整合

接下來，只要將各種新項目拼湊在一起，並將其插入現有的架構項目中。主要重試迴圈正好位於 IBot OnTurn 函式中。其會保留自訂 IStore 實作，而為了測試目的，我們讓相依性得以插入。我們已將所有對話方塊主控程式碼放入稱為 DialogHost 的類別中，以便公開單一公用靜態函式。此函式已定義為接受輸入活動和舊狀態，然後傳回所產生的活動和新狀態。

在此函式中所要進行的第一件事，就是建立我們先前介紹的自訂 BotAdapter。接著，我們會建立 DialogSet 和 DialogContext 並執行一般 [繼續] 或 [開始] 流程，以與往常完全相同的方式執行對話方塊。我們尚未涵蓋的唯一項目是自訂存取子需求。這其實是非常簡單的填充碼，可協助將對話方塊狀態傳入對話方塊系統中。存取子會在使用對話方塊系統時使用 ref 語意，因此接下來只需要傳遞控制代碼。為了讓事情變得更清楚，我們限制了使用於 ref 語意的類別範本。

我們對於分層極為小心謹慎，所以將 JsonSerializer 內嵌於我們的主控程式碼，因為當不同的實作可能以不同的方式序列化時，我們不希望其位於插入式儲存層內。

以下是驅動程式程式碼：

DialogHost.cs [!code-csharpDialogHost]

最後，對於自訂存取子，我們只需要實作 Get，因為狀態依 ref 而定：

RefAccessor.cs [!code-csharpRefAccessor]

其他資訊

在 GitHub 上可取得本文中使用的 [C# 範例](#) 程式碼。

將遙測新增至 Bot

2019/5/14 • [Edit Online](#)

適用於:  SDK v4  SDK v3

在 Bot Framework sdk 4.2 版中，已在產品中新增遙測記錄功能。這可讓 Bot 應用程式將事件資料傳送至 Application Insights 之類的服務。第一節會說明這兩種方法，之後則會有更廣泛的遙測功能。

本文件說明如何整合 Bot 與新的遙測功能。

基本遙測選項

基本的 Application insights

設定 Bot 有兩種方法。第一種方法假設您要與 Application Insights 整合。

設定檔包含有關 Bot 在執行時所用外部服務的中繼資料。例如，CosmosDB、Application Insights 和 Language Understanding (LUIS) 服務連線和中繼資料都會儲存在這裡。

如果您想要「貯存」Application Insights，不需要額外的 Application Insights 專屬設定 (例如遙測初始設定式)，只要在初始化期間傳入組態物件 (通常是 `IConfiguration`) 即可。這是最簡單的初始化方法並且會設定 Application Insights，以開始追蹤要求、其他服務的外部呼叫，以及跨服務相互關聯事件。

您必須新增 **Microsoft.Bot.Builder.Integration.ApplicationInsights.Core** NuGet 套件。

- [C#](#)
- [JavaScript](#)

Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    // Add Application Insights - pass in the bot configuration
    services.AddBotApplicationInsights(<your IConfiguration variable name - likely "config">);
    ...

}

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseBotApplicationInsights()
        ...
        .UseDefaultFiles()
        .UseStaticFiles()
        .UseBotFramework();
    ...
}
```

覆寫遙測用戶端

如果您想要自訂 Application Insights 用戶端，或者想要登入完全不同的服務，則必須以不同的方式設定系統。透過 nuget 下載套件 `Microsoft.Bot.Builder.ApplicationInsights`，或使用 npm 安裝 `botbuilder-applicationinsights`。檢測金鑰可於 Azure 入口網站上找到。

修改 Application Insights 組態

```

public void ConfigureServices(IServiceCollection services)
{
    ...
    // Create Application Insight Telemetry Client
    // with custom configuration.
    var telemetryClient = TelemetryClient(myCustomConfiguration)

    // Add Application Insights
    services.AddBotApplicationInsights(new BotTelemetryClient(telemetryClient), "InstrumentationKey");

```

使用自訂遙測 如果您想要將 Bot Framework 所產生的遙測事件記錄到完全不同的系統，請建立自基底介面衍生的新類別並加以設定。

```

public void ConfigureServices(IServiceCollection services)
{
    ...
    // Create my IBotTelemetryClient-based logger
    var myTelemetryClient = MyTelemetryLogger();

    // Add Application Insights
    services.AddSingleton(myTelemetryClient);
    ...
}

```

將自訂記錄新增至您的 Bot

一旦 Bot 設定了新的遙測記錄支援，您即可開始將遙測新增至您的 Bot。BotTelemetryClient (採用 C#，IBotTelemetryClient) 有數種方法可記錄不同類型的事件。選擇適當的事件類型，可讓您利用 Application Insights 現有的報告 (如果您使用 Application Insights 的話)。在一般情況下，通常會使用 TraceEvent。使用 TraceEvent 記錄的資料會落在 Kusto 的 customEvent 資料表中。

如果在 Bot 內使用對話，則每個對話式物件 (包括提示) 都會包含新的 TelemetryClient 屬性。這是可讓您執行記錄的 BotTelemetryClient。這不是只為了方便而已，我們會在本文稍看到如果已設定這個屬性，WaterfallDialogs 將會產生事件。

識別碼和自訂事件

將事件記錄到 Application Insights 時，所產生的事件包含您不必填入的預設屬性。比方說，每個自訂事件 (以 TraceEvent API 產生) 都會包含 user_id 和 session_id 屬性。此外，也會新增 activityId、activityType 和 channelId。

注意：系統不會提供這些值給自訂遙測用戶端。

屬性	類型	詳細資料
user_id	string	ChannelID + From.Id
session_id	string	ConversationID
customDimensions.activityId	string	Bot 活動識別碼
customDimensions.activityType	string	Bot 活動類型
customDimensions.channelId	string	Bot 活動通道識別碼

深入的遙測

4.4 版的 SDK 新增了三個新元件。所有元件會使用 `IBotTelemetryClient` (在 node.js 中則使用 `BotTelemetryClient`) 介面進行記錄，並可使用自訂實作來加以覆寫。

- Bot Framework 中介軟體元件 (`TelemetryLoggerMiddleware`)，會在接收、傳送、更新或刪除訊息時留下記錄。您可以進行覆寫來建立自訂記錄。
- `LuisRecognizer` 類別。您可以透過兩種方式進行覆寫來建立自訂記錄 - 經由叫用 (新增/取代屬性) 或衍生類別。
- `QnAMaker` 類別。您可以透過兩種方式進行覆寫來建立自訂記錄 - 經由叫用 (新增/取代屬性) 或衍生類別。

遙測中介軟體

C#	JAVASCRIPT
<code>Microsoft.Bot.Builder.TelemetryLoggerMiddleware</code>	<code>botbuilder-core</code>

現成可用的使用方式

`TelemetryLoggerMiddleware` 是無須修改即可新增的 Bot Framework 元件，其會執行記錄而能提供會隨附於 Bot Framework SDK 的現成可用報告。

```
var telemetryClient = sp.GetService<IBotTelemetryClient>();
var telemetryLogger = new TelemetryLoggerMiddleware(telemetryClient, logPersonalInformation: true);
options.Middleware.Add(telemetryLogger); // Add to the middleware collection
```

新增屬性

如果您決定要新增其他屬性，就可以衍生 `TelemetryLoggerMiddleware` 類別。例如，如果您想要將屬性 "MyImportantProperty" 新增至 `BotMessageReceived` 事件。當使用者對 Bot 傳送訊息時便會記錄 `BotMessageReceived`。若要新增其他屬性，可透過下列方式來完成：

```
class MyTelemetryMiddleware : TelemetryLoggerMiddleware
{
    ...
    public Task OnReceiveActivityAsync(
        Activity activity,
        CancellationToken cancellation)
    {
        // Fill in the "standard" properties for BotMessageReceived
        // and add our own property.
        var properties = FillReceiveEventProperties(activity,
            new Dictionary<string, string>
            { {"MyImportantProperty", "myImportantValue"} });

        // Use TelemetryClient to log event
        TelemetryClient.TrackEvent(
            TelemetryLoggerConstants.BotMsgReceiveEvent,
            properties);
    }
    ...
}
```

而在啟動時，我們會新增新的類別：

```
var telemetryLogger = new TelemetryLuisRecognizer(telemetryClient, logPersonalInformation: true);
options.Middleware.Add(telemetryLogger); // Add to the middleware collection
```

完全取代屬性/其他事件

如果您決定要完全取代所記錄的屬性，便可以衍生 `TelemetryLoggerMiddleware` 類別 (如上方擴充屬性時)。同樣地，

記錄新事件也會以相同方式來執行。

例如，如果您要完全取代 `BotMessageSend` 屬性並傳送多個事件，下列範例會示範要如何執行：

```
class MyTelemetryMiddleware : TelemetryLoggerMiddleware
{
    ...
    public Task<RecognizerResult> OnLuisRecognizeAsync(
        Activity activity,
        string dialogId = null,
        CancellationToken cancellation)
    {
        // Override properties for BotMsgSendEvent
        var botMsgSendProperties = new Dictionary<string, string>();
        properties.Add("MyImportantProperty", "myImportantValue");
        // Log event
        TelemetryClient.TrackEvent(
            TelemetryLoggerConstants.BotMsgSendEvent,
            botMsgSendProperties);

        // Create second event.
        var secondEventProperties = new Dictionary<string, string>();
        secondEventProperties.Add("activityId",
            activity.Id);
        secondEventProperties.Add("MyImportantProperty",
            "myImportantValue");
        TelemetryClient.TrackEvent(
            "MySecondEvent",
            secondEventProperties);
    }
    ...
}
```

注意：若未記錄標準屬性，將會導致隨附於產品的現成可用報告停止運作。

從遙測中介軟體記錄的事件

[BotMessageSend](#) [BotMessageReceived](#) [BotMessageUpdate](#) [BotMessageDelete](#)

遙測支援 LUIS

C#	JAVASCRIPT
Microsoft.Bot.Builder.AI.Luis	botbuilder-ai

現成可用的使用方式

`LuisRecognizer` 是現有的 Bot Framework 元件，藉由透過 `luisOptions` 傳遞 `IBotTelemetryClient` 介面即可啟用遙測。您可以視需要覆寫所記錄的預設屬性，並記錄新的事件。

在建構 `luisOptions` 期間，必須提供 `IBotTelemetryClient` 物件才能讓此功能運作。

```
var luisOptions = new LuisPredictionOptions(
    ...
    telemetryClient,
    false); // Log personal information flag. Defaults to false.

var client = new LuisRecognizer(luisApp, luisOptions);
```

新增屬性

如果您決定要新增其他屬性，就可以衍生 `LuisRecognizer` 類別。例如，如果您想要將屬性 "MyImportantProperty" 新增至 `LuisResult` 事件。執行 LUIS 預測呼叫時，會記錄 `LuisResult`。若要新增其他屬性，可透過下列方式來完成：

```

class MyLuisRecognizer : LuisRecognizer
{
    ...
    override protected Task OnRecognizerResultAsync(
        RecognizerResult recognizerResult,
        ITurnContext turnContext,
        Dictionary<string, string> properties = null,
        CancellationToken cancellationToken = default(CancellationToken))
    {
        var luisEventProperties = FillLuisEventProperties(result,
            new Dictionary<string, string>
            { {"MyImportantProperty", "myImportantValue"} } );

        TelemetryClient.TrackEvent(
            LuisTelemetryConstants.LuisResultEvent,
            luisEventProperties);
        ...
    }
    ...
}

```

經由叫用來新增屬性

有時候您必須在叫用期間新增其他屬性：

```

var additionalProperties = new Dictionary<string, string>
{
    { "dialogId", "myDialogId" },
    { "conversationInfo", "myConversationInfo" },
};

var result = await recognizer.RecognizeAsync(turnContext,
    additionalProperties,
    CancellationToken.None).ConfigureAwait(false);

```

完全取代屬性/其他事件

如果您決定要完全取代所記錄的屬性，便可以衍生 `LuisRecognizer` 類別（如上方擴充屬性時）。同樣地，記錄新事件也會以相同方式來執行。

例如，如果您要完全取代 `LuisResult` 屬性並傳送多個事件，下列範例會示範要如何執行：

```

class MyLuisRecognizer : LuisRecognizer
{
    ...
    override protected Task OnRecognizerResultAsync(
        RecognizerResult recognizerResult,
        ITurnContext turnContext,
        Dictionary<string, string> properties = null,
        CancellationToken cancellationToken = default(CancellationToken))
    {
        // Override properties for LuisResult event
        var luisProperties = new Dictionary<string, string>();
        properties.Add("MyImportantProperty", "myImportantValue");

        // Log event
        TelemetryClient.TrackEvent(
            LuisTelemetryConstants.LuisResult,
            luisProperties);

        // Create second event.
        var secondEventProperties = new Dictionary<string, string>();
        secondEventProperties.Add("MyImportantProperty2",
            "myImportantValue2");
        TelemetryClient.TrackEvent(
            "MySecondEvent",
            secondEventProperties);
        ...
    }
    ...
}

```

注意：若未記錄標準屬性，將會導致隨附於產品的 Application Insights 現成可用報告停止運作。

從 `TelemetryLuisRecognizer` 記錄的事件

LuisResult

遙測 QnA 辨識器

C#	JAVASCRIPT
<code>Microsoft.Bot.Builder.AI.QnA</code>	<code>botbuilder-ai</code>

現成可用的使用方式

QnAMaker 類別是現有的 Bot Framework 元件，會另外新增兩個建構函式參數來啟用記錄功能，以啟用隨附於 Bot Framework SDK 的現成可用報告。新的 `telemetryClient` 會參考執行記錄的 `IBotTelemetryClient` 介面。

```

var qna = new QnAMaker(endpoint, options, client,
    telemetryClient: telemetryClient,
    logPersonalInformation: true);

```

新增屬性

如果您決定要新增其他屬性，方法有兩種 - 在必須於 QnA 呼叫期間新增屬性以便擷取答案時，或從 `QnAMaker` 類別衍生。

下列範例會示範如何從 `QnAMaker` 類別衍生。此範例會顯示如何將屬性 "MyImportantProperty" 新增至 `QnAMessage` 事件。執行 QnA `GetAnswers` 呼叫時，會記錄 `QnAMessage` 事件。此外，我們還會記錄第二個事件 "MySecondEvent"。

```

class MyQnAMaker : QnAMaker
{
    ...
    protected override Task OnQnaResultsAsync(
        QueryResult[] queryResults,
        ITurnContext turnContext,
        Dictionary<string, string> telemetryProperties = null,
        Dictionary<string, double> telemetryMetrics = null,
        CancellationToken cancellationToken = default(CancellationToken))
    {
        var eventData = await FillQnAEventAsync(queryResults, turnContext, telemetryProperties,
            telemetryMetrics, cancellationToken).ConfigureAwait(false);

        // Add my property
        eventData.Properties.Add("MyImportantProperty", "myImportantValue");

        // Log QnaMessage event
        TelemetryClient.TrackEvent(
            QnATelemetryConstants.QnaMsgEvent,
            eventData.Properties,
            eventData.Metrics
        );

        // Create second event.
        var secondEventProperties = new Dictionary<string, string>();
        secondEventProperties.Add("MyImportantProperty2",
            "myImportantValue2");
        TelemetryClient.TrackEvent(
            "MySecondEvent",
            secondEventProperties);
    }
    ...
}

```

在 `GetAnswersAsync` 期間新增屬性

如果您有需要在執行階段新增的屬性，`GetAnswersAsync` 方法會提供可新增至事件的屬性和（或）計量。

例如，如果您想要將 `dialogId` 新增至事件，則可以透過如下方式來完成：

```

var telemetryProperties = new Dictionary<string, string>
{
    { "dialogId", myDialogId },
};

var results = await qna.GetAnswersAsync(context, opts, telemetryProperties);

```

`QnAMaker` 類別會提供覆寫屬性的功能，包括 `PersonalInformation` 屬性。

完全取代屬性/其他事件

如果您決定要完全取代所記錄的屬性，便可以衍生 `TelemetryQnAMaker` 類別（如上方擴充屬性時）。同樣地，記錄新事件也會以相同方式來執行。

例如，如果您要完全取代 `QnaMessage` 屬性，下列範例會示範要如何執行：

```

class MyLuisRecognizer : TelemetryQnAMaker
{
    ...
    protected override Task OnQnaResultsAsync(
        QueryResult[] queryResults,
        ITurnContext turnContext,
        Dictionary<string, string> telemetryProperties = null,
        Dictionary<string, double> telemetryMetrics = null,
        CancellationToken cancellationToken = default(CancellationToken))
    {
        // Add properties from GetAnswersAsync
        var properties = telemetryProperties ?? new Dictionary<string, string>();
        // GetAnswerAsync properties overrides - don't add if already present.
        properties.TryAdd("MyImportantProperty", "myImportantValue");

        // Log event
        TelemetryClient.TrackEvent(
            QnATelemetryConstants.QnaMsgEvent,
            properties);
    }
    ...
}

```

注意：若未記錄標準屬性，將會導致隨附於產品的現成可用報告停止運作。

從 `TelemetryLuisRecognizer` 記錄的事件

[QnAMessage](#)

WaterfallDialog 事件

除了產生您自己的事件，SDK 內的 `WaterfallDialog` 物件現在也會產生事件。下一節說明從 Bot Framework 內產生的事件。在 `WaterfallDialog` 上設定 `TelemetryClient` 屬性，就會儲存這些事件。

以下範例示範如何修改利用 `WaterfallDialog` 來記錄遙測事件的範例 (BasicBot)。BasicBot 會利用當 `WaterfallDialog` 置於 `ComponentDialog` (`GreetingDialog`) 內時使用的通用樣式。

```

// IBotTelemetryClient is direct injected into our Bot
public BasicBot(BotServices services, UserState userState, ConversationState conversationState,
IBotTelemetryClient telemetryClient)
{
    ...

    // The IBotTelemetryClient passed to the GreetingDialog
    ...
    Dialogs = new DialogSet(_dialogStateAccessor);
    Dialogs.Add(new GreetingDialog(_greetingStateAccessor, telemetryClient));
    ...

    // The IBotTelemetryClient to the WaterfallDialog
    ...
    AddDialog(new WaterfallDialog(ProfileDialog, waterfallSteps) { TelemetryClient = telemetryClient });
    ...
}

```

一旦 `WaterfallDialog` 有已設定的 `IBotTelemetryClient`，它就會開始記錄事件。

Bot Framework Service 所產生的事件

除了 `WaterfallDialog` (可從 Bot 程式碼產生事件)，Bot Framework Channel 服務也會記錄事件。這可協助您診斷通道或整體 Bot 失敗的問題。

CustomEvent: "Activity"

記錄來源：通道服務 在收到訊息時由通道服務記錄。

例外狀況：「Bot 錯誤」

記錄來源：通道服務 在 Bot 呼叫傳回非 2XX Http 回應時由通道記錄。

產生的所有事件

CustomEvent: "WaterfallStart"

當 WaterfallDialog 開始時，就會記錄 `WaterfallStart` 事件。

- `user_id` (從遙測初始設定式)
- `session_id` (從遙測初始設定式)
- `customDimensions.activityId` (從遙測初始設定式)
- `customDimensions.activityType` (從遙測初始設定式)
- `customDimensions.channelId` (從遙測初始設定式)
- `customDimensions.DialogId` (這是傳入您瀑布的 dialogId (字串)。您可以將此視為「瀑布類型」)
- `customDimensions.InstanceID` (每個對話執行個體特有)

CustomEvent: "WaterfallStep"

記錄瀑布式對話中的個別步驟。

- `user_id` (從遙測初始設定式)
- `session_id` (從遙測初始設定式)
- `customDimensions.activityId` (從遙測初始設定式)
- `customDimensions.activityType` (從遙測初始設定式)
- `customDimensions.channelId` (從遙測初始設定式)
- `customDimensions.DialogId` (這是傳入您瀑布的 dialogId (字串)。您可以將此視為「瀑布類型」)
- `customDimensions.StepName` (若為 lambda 則是方法名稱或 `StepXofY`)
- `customDimensions.InstanceID` (每個對話執行個體特有)

CustomEvent: "WaterfallDialogComplete"

記錄於瀑布式對話完成時。

- `user_id` (從遙測初始設定式)
- `session_id` (從遙測初始設定式)
- `customDimensions.activityId` (從遙測初始設定式)
- `customDimensions.activityType` (從遙測初始設定式)
- `customDimensions.channelId` (從遙測初始設定式)
- `customDimensions.DialogId` (這是傳入您瀑布的 dialogId (字串)。您可以將此視為「瀑布類型」)
- `customDimensions.InstanceID` (每個對話執行個體特有)

CustomEvent: "WaterfallDialogCancel"

記錄於瀑布式對話取消時。

- `user_id` (從遙測初始設定式)
- `session_id` (從遙測初始設定式)
- `customDimensions.activityId` (從遙測初始設定式)
- `customDimensions.activityType` (從遙測初始設定式)
- `customDimensions.channelId` (從遙測初始設定式)
- `customDimensions.DialogId` (這是傳入您瀑布的 dialogId (字串)。您可以將此視為「瀑布類型」)

- `customDimensions.StepName` (若為 lambda 則是方法名稱或 `StepXofY`)
- `customDimensions.InstanceID` (每個對話執行個體特有)

CustomEvent: BotMessageReceived

記錄於 Bot 從使用者收到新訊息時。

若未加以覆寫，則會使用 `Microsoft.Bot.Builder.IBotTelemetry.TrackEvent()` 方法從 `Microsoft.Bot.Builder.TelemetryLoggerMiddleware` 記錄此事件。

- 工作階段識別碼
 - 使用 Application Insights 時，會從 `TelemetryBotIdInitializer` 將此屬性記錄為 Application Insights 中所使用的**工作階段識別碼** (`Telemetry.Context.Session.Id`)。
 - 對應至 Bot Framework 通訊協定所定義的**對話識別碼**。
 - 所記錄的屬性名稱是 `session_id`。
- 使用者識別碼
 - 使用 Application Insights 時，會從 `TelemetryBotIdInitializer` 將此屬性記錄為 Application Insights 中所使用的**使用者識別碼** (`Telemetry.Context.User.Id`)。
 - 此屬性的值結合了 Bot Framework 通訊協定所定義的**通道識別碼**和**使用者識別碼** (串連在一起) 屬性。
 - 所記錄的屬性名稱是 `user_id`。
- ActivityID
 - 使用 Application Insights 時，會從 `TelemetryBotIdInitializer` 將此屬性記錄為事件的屬性。
 - 對應至 Bot Framework 通訊協定所定義的**活動識別碼**。
 - 屬性名稱是 `activityId`。
- 通道識別碼
 - 使用 Application Insights 時，會從 `TelemetryBotIdInitializer` 將此屬性記錄為事件的屬性。
 - 對應至 Bot Framework 通訊協定的**通道識別碼**。
 - 所記錄的屬性名稱是 `channelId`。
- ActivityType
 - 使用 Application Insights 時，會從 `TelemetryBotIdInitializer` 將此屬性記錄為事件的屬性。
 - 對應至 Bot Framework 通訊協定的**活動類型**。
 - 所記錄的屬性名稱是 `activityType`。
- 文字
 - 會在 `logPersonalInformation` 屬性設定為 `true` 時**選擇性地**記錄。
 - 對應至 Bot Framework 通訊協定的**活動文字**欄位。
 - 所記錄的屬性名稱是 `text`。
- Speak
 - 會在 `logPersonalInformation` 屬性設定為 `true` 時**選擇性地**記錄。
 - 對應至 Bot Framework 通訊協定的**活動語音**欄位。
 - 所記錄的屬性名稱是 `speak`。
 -
- FromId
 - 對應至 Bot Framework 通訊協定的**來源識別碼**欄位。
 - 所記錄的屬性名稱是 `fromId`。

- FromName
 - 會在 `logPersonalInformation` 屬性設定為 `true` 時選擇性地記錄。
 - 對應至 Bot Framework 通訊協定的來源名稱欄位。
 - 所記錄的屬性名稱是 `fromName`。
- RecipientId
 - 對應至 Bot Framework 通訊協定的來源名稱欄位。
 - 所記錄的屬性名稱是 `fromName`。
- RecipientName
 - 對應至 Bot Framework 通訊協定的來源名稱欄位。
 - 所記錄的屬性名稱是 `fromName`。
- ConversationId
 - 對應至 Bot Framework 通訊協定的來源名稱欄位。
 - 所記錄的屬性名稱是 `fromName`。
- ConversationName
 - 對應至 Bot Framework 通訊協定的來源名稱欄位。
 - 所記錄的屬性名稱是 `fromName`。
- 地區設定
 - 對應至 Bot Framework 通訊協定的來源名稱欄位。
 - 所記錄的屬性名稱是 `fromName`。

CustomEvent: BotMessageSend

記錄來源：TelemetryLoggerMiddleware

記錄於 Bot 傳送訊息時。

- UserID (從遙測初始設定式)
- SessionID (從遙測初始設定式)
- ActivityID (從遙測初始設定式)
- Channel (從遙測初始設定式)
- ActivityType (從遙測初始設定式)
- ReplyToID
- RecipientId
- ConversationName
- 地區設定
- RecipientName (PII 選用)
- 文字 (PII 選用)
- 語音 (PII 選用)

CustomEvent: BotMessageUpdate

記錄來源：TelemetryLoggerMiddleware 記錄於 Bot 更新訊息時 (罕見案例)

- UserID (從遙測初始設定式)
- SessionID (從遙測初始設定式)
- ActivityID (從遙測初始設定式)
- Channel (從遙測初始設定式)
- ActivityType (從遙測初始設定式)
- RecipientId

- ConversationId
- ConversationName
- 地區設定
- 文字 (PII 選用)

CustomEvent: BotMessageDelete

記錄來源：TelemetryLoggerMiddleware 記錄於 Bot 刪除訊息時 (罕見案例)

- UserID (從遙測初始設定式)
- SessionID (從遙測初始設定式)
- ActivityID (從遙測初始設定式)
- Channel (從遙測初始設定式)
- ActivityType (從遙測初始設定式)
- RecipientId
- ConversationId
- ConversationName

CustomEvent: LuisEvent

記錄來源：LuisRecognizer

記錄 LUIS 服務的結果。

- UserID (從遙測初始設定式)
- SessionID (從遙測初始設定式)
- ActivityID (從遙測初始設定式)
- 通道 (從遙測初始設定式)
- ActivityType (從遙測初始設定式)
- ApplicationId
- 意圖
- IntentScore
- Intent2
- IntentScore2
- FromId
- SentimentLabel
- SentimentScore
- Entities (json 形式)
- 問題 (PII 選用)

CustomEvent: QnAMessage

記錄來源：QnAMaker

記錄 QnA Maker 服務的結果。

- UserID (從遙測初始設定式)
- SessionID (從遙測初始設定式)
- ActivityID (從遙測初始設定式)
- 通道 (從遙測初始設定式)
- ActivityType (從遙測初始設定式)
- 使用者名稱 (PII 選用)
- 問題 (PII 選用)

- MatchedQuestion
- QuestionId
- Answer
- 分數
- ArticleFound

查詢資料

使用 Application Insights 時，所有資料都會相互關聯（即使是跨服務）。我們可以查詢成功的請求，以及查看該要求的所有相關事件。

下列查詢會告訴您最近的要求：

```
requests
| where timestamp > ago(3d)
| where resultCode == 200
| order by timestamp desc
| project timestamp, operation_Id, appName
| limit 10
```

從第一個查詢中，選取一些 `operation_Id`，然後尋找更多資訊：

```
let my_operation_id = "<OPERATION_ID>";
let union_all = () {
    union
    (traces | where operation_Id == my_operation_id),
    (customEvents | where operation_Id == my_operation_id),
    (requests | where operation_Id == my_operation_id),
    (dependencies | where operation_Id == my_operation_id),
    (exceptions | where operation_Id == my_operation_id)
};
union_all
| order by timestamp asc
| project itemType, name, performanceBucket
```

這可提供單一要求的時間順序明細，以及每次呼叫的持續時間貯體。

itemType		name		performanceBucket
> request		POST /api/messages		<250ms
> customEvent		BotMessageReceived		
> dependency		POST /luis/v2.0/apps/be386c07-ad4e-4148-a592-a8d2a9c93983		<250ms
> customEvent		BotMessageSend		
> dependency		POST /v3/conversations/b159c157d6564f84a7663142b9312131/activities/b159c157d6564f84a7663142b9312131%7C0000032	<250ms	
> customEvent		Activity		
> customEvent		BotMessageSend		
> dependency		POST /v3/conversations/b159c157d6564f84a7663142b9312131/activities/b159c157d6564f84a7663142b9312131%7C0000032	<250ms	
> customEvent		Activity		
> customEvent		Activity		
> customEvent		BotMessageSend		

注意：“Activity” `customEvent` 事件時間戳記失序，因為這些事件是以非同步方式記錄。

建立儀表板

最簡單的測試方法就是使用 Azure 入口網站的範本部署頁面建立儀表板。

- 按一下 [在編輯器中建置您自己的範本]

- 複製並貼上為了協助您建立儀表板而提供的其中一個 json 檔案：
 - [系統健康情況儀表板](#)
 - [交談健康情況儀表板](#)
- 按一下 [儲存]
- 填入 **Basics**：
 - 訂用帳戶：
 - 資源群組：
 - 位置：
- 填入 **Settings**：
 - 見解元件名稱：<像是 `core672so2hw` >
 - 見解元件資源群組：<像是 `core67` >
 - 儀表板名稱：<像是 `'ConversationHealth'` 或 `SystemHealth` >
- 按一下 `I agree to the terms and conditions stated above`
- 按一下 **Purchase**
- 驗證
 - 按一下 **Resource Groups**
 - 從上面選取您的資源群組 (像是 `core67`)。
 - 如果您沒有看到新的資源，則查看 [部署] 並了解是否有任何失敗。
 - 以下是您通常看到的失敗：

```
{"code":"DeploymentFailed","message":"At least one resource deployment operation failed. Please list deployment operations for details. Please see https://aka.ms/arm-debug for usage details.", "details": [{"code":"BadRequest","message":"{\r\n    \"error\": {\r\n        \"code\": \"InvalidTemplate\", \r\n        \"message\": \"Unable to process template language expressions for resource '/subscriptions/45d8a30e-3363-4e0e-849a-4bb0bbf71a7b/resourceGroups/core67/providers/Microsoft.Portal/dashboards/Bot Analytics Dashboard' at line '34' and column '9'. 'The template parameter 'virtualMachineName' is not found. Please see https://aka.ms/arm-template/#parameters for usage details.'\\r\\n    }\\r\\n}}"]}
```

若要檢視資料，請移至 Azure 入口網站。按一下左邊的 [儀表板]，然後從下拉式清單中選取您建立的儀表板。

其他資源

您可以參考這些實作遙測的範例：

- C#
 - [LUIS 搭配 AppInsights](#)
 - [QnA 搭配 AppInsights](#)
- JS
 - [LUIS 搭配 AppInsights](#)
 - [QnA 搭配 AppInsights](#)

在 Bot 中使用 Direct Line Speech

2019/5/14 • [Edit Online](#)

適用於:  SDK v4  SDK v3

Direct Line Speech 使用 Bot Framework 的新 WebSocket 型串流功能，在 Direct Line Speech 通道與您的 Bot 之間交換訊息。在 Azure 入口網站中啟用 Direct Line Speech 通道後，您必須更新 Bot 以接聽並接受這些 WebSocket 連線。這些指示說明如何執行這項操作。

新增 NuGet 套件

在 Direct Line Speech 預覽版中，有您需要新增至 Bot 的其他 NuGet 套件。這些套件裝載於 myget.org NuGet 摘要上：

1. 在 Visual Studio 中開啟 Bot 的專案。
2. 移至 Bot 專案屬性之下的 [管理 NuGet 套件]。
3. 如果您尚未以它作為來源，請從右上角的 NuGet 摘要設定將 <https://botbuilder.myget.org/F/experimental/api/v3/index.json> 新增為摘要。
4. 選取此 NuGet 來源並新增其中一個 `Microsoft.Bot.Protocol.StreamingExtensions.NetCore` 套件。
5. 接受任何提示，可完成將套件新增至您的專案。

選項 1: 更新 .NET Core Bot 程式碼 (如果 Bot 有 `BotController.cs`)

當您在 Azure 入口網站中使用其中一個範本 (例如 EchoBot) 建立新的 Bot 時，您會收到一個 Bot，其中包含可公開單一 POST 端點的 ASP.NET MVC 控制器。下列指示說明如何將該控制器擴展為同時公開一個端點來接受 WebSocket 串流端點 (GET 端點)。

1. 在您解決方案的 Controllers 資料夾之下開啟 `BotController.cs`
2. 在類別中找到 `PostAsync` 方法，並將其裝飾從 `[HttpPost]` 更新為 `[HttpPost,HttpGet]`：

```
[HttpPost,HttpGet]
public async Task PostAsync()
{
    await _adapter.ProcessAsync(Request, Response, _bot);
}
```

3. 儲存並關閉 `BotController.cs`
4. 在您解決方案的根目錄中開啟 `Startup.cs`。
5. 新增命名空間：

```
using Microsoft.Bot.Protocol.StreamingExtensions.NetCore;
```

6. 在 `ConfigureServices` 方法中，在適當的 `services.AddSingleton` 呼叫中以 `WebSocketEnabledHttpAdapter` 取代使用 `AdapterWithErrorHandler`：

```

public void ConfigureServices(IServiceCollection services)
{
    ...
    // Create the Bot Framework Adapter.
    services.AddSingleton<IBotFrameworkHttpAdapter, WebSocketEnabledHttpAdapter>();

    services.AddTransient<IBot, EchoBot>();

    ...
}

```

- 仍然在 Startup.cs 中，巡覽至 ConfigureServices 方法的底部。在 app.UseMvc(); 呼叫 (這很重要，因為 Use 呼叫的順序關係重大) 之前，新增 app.UseWebSockets();。方法結尾應該類似以下內容：

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    ...
    app.UseDefaultFiles();
    app.UseStaticFiles();
    app.UseWebSockets();
    app.UseMvc();

    ...
}

```

- 您 Bot 程式碼的其餘部分維持不變！

選項 2: 更新 .NET Core Bot 程式碼 (如果 Bot 使用 AddBot 和 UseBotFramework 而非 BotController)

如果您已使用 4.3.2 版之前的 Bot Builder SDK v4 建立 Bot，Bot 可能不包含 BotController，但在 Startup.cs 檔案中改為使用 AddBot() 和 UseBotFramework() 方法來公開 Bot 接收訊息的 POST 端點。若要公開新的串流端點，您必須新增 BotController 及移除 AddBot() 和 UseBotFramework() 方法。這些指示會逐步引導需要進行的變更。

- 藉由新增名為 BotController.cs 的檔案，將新的 MVC 控制器新增至 Bot 專案。將此檔案中新增控制器程式碼：

```

[Route("api/messages")]
[ApiController]
public class BotController : ControllerBase
{
    private readonly IBotFrameworkHttpAdapter _adapter;
    private readonly IBot _bot;

    public BotController(IBotFrameworkHttpAdapter adapter, IBot bot)
    {
        _adapter = adapter;
        _bot = bot;
    }

    [HttpPost,HttpGet]
    public async Task ProcessMessageAsync()
    {
        await _adapter.ProcessAsync(Request, Response, _bot);
    }
}

```

- 在 Startup.cs 檔案中，找出 Configure 方法。移除 UseBotFramework() 行並確定您有這幾行：

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    ...
    app.UseDefaultFiles();
    app.UseStaticFiles();
    app.UseWebSockets();
    app.UseMvc();

    ...
}
```

3. 同時在 Startup.cs 檔案中，找出 ConfigureServices 方法。移除 AddBot() 行並確定您有這幾行：

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    services.AddSingleton<ICredentialProvider, ConfigurationCredentialProvider>();

    services.AddSingleton<IChannelProvider, ConfigurationChannelProvider>();

    // Create the Bot Framework Adapter.
    services.AddSingleton<IBotFrameworkHttpAdapter, WebSocketEnabledHttpAdapter>();

    // Create the bot as a transient. In this case the ASP Controller is expecting an IBot.
    services.AddTransient<IBot, EchoBot>();
}
```

4. 您 Bot 程式碼的其餘部分維持不變！

後續步驟

[將 Bot 連線至 Direct Line Speech \(預覽\)](#)

適用於 .NET 的 Bot Framework SDK

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

適用於 .NET 的 Bot Framework SDK 擁有功能強大的架構，可用於建構 Bot 以處理自由格式互動與功能更強的導向式交談，在這些案例中，使用者可從可能的值加以選取。可以輕鬆利用 C# 來為 .NET 開發人員提供撰寫 Bot 的熟悉方法。

使用 SDK 就可以利用下列 SDK 功能組建 Bot：

- 具備獨立且可組合之對話方塊的強大對話方塊系統
- 內建簡單提示，例如是/否、字串、數字和列舉
- 內建對話方塊，可使用功能強大的 AI 架構，例如 [LUIS](#)
- FormFlow 可自動從 C# 類別產生 Bot，以引導使用者完成對話，並視需要提供說明、瀏覽、釐清和確認

IMPORTANT

已在 2017 年 7 月 31 日的 Bot Framework 安全性通訊協定中實作重大變更。若要避免變更對 Bot 造成負面影響，您必須確定應用程式使用 Bot Framework SDK 3.5 版或更新版本。如果您已使用在 2017 年 1 月 5 日 (Bot Framework SDK 3.5 版發佈日期) 前取得的 SDK 建立 Bot，請務必升級至 Bot Framework SDK 3.5 版或更新版本。

取得 SDK

SDK 可用於 NuGet 套件，以及 [GitHub](#) 的開放原始碼。

IMPORTANT

適用於 .NET 的 Bot Framework SDK 需要 .NET Framework 4.6 或更新版本。若您將 SDK 新增到以較低 .NET Framework 版本為目標的專案，您將必須先將您的專案更新為以 .NET Framework 4.6 為目標。

若要在 Visual Studio 專案中安裝 SDK，請完成下列步驟：

1. 在 [方案總管] 中，以滑鼠右鍵按一下專案名稱，然後選取 [管理 NuGet 套件]。
2. 在 [瀏覽] 索引標籤上的搜尋方塊中輸入 "Microsoft.Bot.Builder"。
3. 選取結果清單中的 [Microsoft.Bot.Builder]，按一下 [安裝]，並接受變更。

取得程式碼範例

此 SDK 包含使用適用於 .NET 的 Bot Framework SDK 功能的[原始碼範例](#)。

後續步驟

藉由檢閱本節中的各個文章，深入了解如何使用適用於 .NET 的 Bot Framework SDK 建置 Bot，從以下開始：

- [快速入門](#)：依照本逐步教學課程中的指示，快速建置及測試簡單的 Bot。
- [重要概念](#)：了解適用於 .NET 的 Bot Framework SDK 重要概念。

如果您有關於適用於 .NET 的 Bot Framework SDK 的問題或建議，請參閱[支援](#)以取得可用資源清單。

適用於 .NET 的 Bot Framework SDK 重要概念

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

本文介紹適用於 .NET 的 Bot Framework SDK 重要概念。

連接器

Bot Framework Connector 提供單一 REST API，讓 Bot 能夠跨多個通道 (例如 Skype、電子郵件、Slack 等) 進行通訊。它可藉由將訊息從 Bot 轉送至通道以及從通道轉送至 Bot，讓 Bot 和使用者之間的通訊更為順暢。

在適用於 .NET 的 Bot Framework SDK 中，[Connector](#) 程式庫可讓您存取 Connector。

活動

連接器會使用[活動](#) (英文) 物件在 Bot 和通道 (使用者) 之間來回傳遞資訊。最常見的活動類型是訊息，但是還有其他活動類型可用來將各種類型的資訊傳達給 Bot 或通道。

如需適用於 .NET 的 Bot Framework SDK 中的活動詳細資訊，請參閱[活動概觀](#)。

對話

當您使用適用於 .NET 的 Bot Framework SDK 建立 Bot 時，您可以使用[對話](#)來建立交談模型及管理[交談流程](#)。對話可由其他對話所組成以便充分地重複使用，而且對話內容會維護[對話的堆疊](#)，這些對話在對話的任何時間點均處於作用中狀態。由對話 (dialog) 組成的對話 (conversation) 可移植到不同電腦，這使得您的 Bot 實作能夠加以調整。

在適用於 .NET 的 Bot Framework SDK 中，[Builder](#) 程式庫可讓您管理對話。

FormFlow

您可以使用適用於 .NET 的 Bot Framework SDK 中的 [FormFlow](#)，簡化建置 Bot 以向使用者收集資訊的程序。例如，接受三明治訂單的 Bot 必須向使用者收集數項資訊，例如麵包類型、配料的選擇、大小等等。根據基本指導方針，FormFlow 可以自動產生管理這類引導式對話 (conversation) 所需的對話 (dialog)。

State

Bot Builder Framework 可讓您的 Bot 儲存和擷取與使用者、對話或特定對話內容中的特定使用者相關聯的狀態資料。狀態資料有許多用途，例如判斷上一個對話最後的內容，或是單純地以名字問候回來的使用者。如果您儲存使用者的喜好設定，下次聊天時就可以使用該資訊來自訂對話。例如，您可能會在有和使用者感興趣之主題相關的新聞文章，或有可用約會時警示使用者。

針對測試和原型設計目的，您可以使用 Bot Builder Framework 的記憶體內部資料儲存體。針對生產環境的 Bot，您可以實作自己的儲存體接器，或使用其中一個 Azure 擴充功能。Azure 擴充功能可讓您將 Bot 的狀態資料儲存於表格儲存體、CosmosDB 或 SQL。本文將示範如何使用記憶體內部儲存體接器來儲存 Bot 的狀態資料。

IMPORTANT

建議您不要將 Bot Framework 狀態服務 API 用於生產環境，因為未來版本可能會淘汰它。建議您更新 Bot 程式碼，以針對測試目的使用記憶體內部儲存體接器，或針對生產環境 Bot 使用其中一個 Azure 擴充功能。

如需使用適用於 .NET 的 Bot Framework SDK 管理狀態的詳細資訊，請參閱[管理狀態資料](#)。

命名慣例

適用於 .NET 的 Bot Framework SDK 程式庫會使用強類型並依照 Pascal 命名法的大小寫慣例。不過，透過網路來回傳輸的 JSON 訊息會使用駝峰式大小寫命名慣例。例如，C# 屬性 **ReplyToId** 會在透過網路傳輸的 JSON 訊息中序列化為 **replyToId**。

安全性

您應該確定 Bot 的端點只能由 Bot Framework Connector 服務加以呼叫。如需有關本主題的詳細資訊，請參閱[保護 Bot](#)。

後續步驟

現在您已了解每個 Bot 背後的概念。您可以使用範本，快速地[使用 Visual Studio 建置 Bot](#)。接下來，將從對話開始，更詳細地研究每個重要概念。

[適用於 .NET 的 Bot Framework SDK 中的對話](#)

活動概觀

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

連接器會使用**活動** (英文) 物件在 Bot 和通道 (使用者) 之間來回傳遞資訊。最常見的活動類型是**訊息**，但是還有其他活動類型可用來將各種類型的資訊傳達給 Bot 或通道。

適用於 .NET 的 Bot Framework SDK 中的活動類型

適用於 .NET 的 Bot Framework SDK 支援下列活動類型。

ACTIVITY.TYPE	介面	說明
message	IMessageActivity	代表 Bot 和使用者之間的通訊。
conversationUpdate	IConversationUpdateActivity	表示 Bot 已新增至對話、其他成員已新增至對話中或從對話中移除，或對話中繼資料已變更。
contactRelationUpdate	IContactRelationUpdateActivity	表示 Bot 已新增至使用者的連絡人清單，或從使用者的連絡人清單中移除。
typing	ITypingActivity	表示使用者或在對話另一端的 Bot 正在編譯回應。
deleteUserData	n/a	表示使用者已要求 Bot 刪除任何可能已儲存的使用者資料。
endOfConversation	IEndOfConversationActivity	表示對話結束。
event	IEventActivity	代表傳送給 Bot 使用者看不到的通訊。
invoke	IInvokeActivity	代表傳送給 Bot 以要求其執行特定作業的通訊。此活動類型由 Microsoft Bot Framework 保留供內部使用。
messageReaction	IMessageReactionActivity	表示使用者已對現有活動做出回應。例如，使用者按了訊息上的「讚」按鈕。

message

您的 Bot 會傳送 **message** 活動以傳達資訊，及接收來自使用者的 **message** 活動。某些訊息可能只包含純文字，而其他訊息可能包含更豐富的內容，例如[要讀出的文字](#)、[建議的動作](#)、[媒體附件](#)、[複合式資訊卡 \(Rich Card\)](#)，和[通道特定資料](#)。如需常用訊息屬性的資訊，請參閱[建立訊息](#)。

conversationUpdate

每當 Bot 已新增至對話、其他成員已新增至對話中或從對話中移除，或對話中繼資料已變更，Bot 就會收到 **conversationUpdate** 活動。

如果成員已新增至對話，活動的 `MembersAdded` 屬性就會包含 `ChannelAccount` 物件陣列以識別新成員。

若要判斷 Bot 是否已新增至對話（亦即成為其中一個新成員），請評估活動的 `Recipient.Id` 值（亦即 Bot 的 ID）是否符合 `MembersAdded` 陣列中任何帳戶的 `Id` 屬性。

如果成員已從對話中移除，`MembersRemoved` 屬性會包含 `ChannelAccount` 物件陣列以識別移除的成員。

TIP

如果您的 Bot 收到 **conversationUpdate** 活動指出使用者已加入對話，您可以選擇傳送歡迎訊息給使用者作為回應。

contactRelationUpdate

每當 Bot 加入使用者的連絡人清單，或從中移除時，Bot 會收到 **contactRelationUpdate** 活動。活動的 `Action` 屬性值（新增 | 移除）會指出 Bot 是否已加入使用者的連絡人清單，或者是否已從中移除。

typing

Bot 會收到 **typing** 活動，表示使用者正在輸入回應。Bot 會傳送 **typing** 活動，向使用者表示 Bot 正在運作以完成要求或編譯回應。

deleteUserData

當使用者要求刪除 Bot 先前為其保存的任何資料，Bot 會收到 **deleteUserData** 活動。如果您的 Bot 收到此類型的活動，就會為提出要求的使用者刪除先前已儲存的任何個人識別資訊（PII）。

endOfConversation

Bot 會收到 **endOfConversation** 活動，表示使用者已結束對話。Bot 會傳送 **endOfConversation** 活動，向使用者表示對話已結束。

事件

Bot 可能會從外部程序或服務收到 **event** 活動，代表想要傳達資訊給您的 Bot，而不向使用者顯示該資訊。**event** 活動的傳送者通常不會預期 Bot 以任何方式確認收到活動。

invoke

Bot 可能會收到 **invoke** 活動，代表執行特定作業的要求。**invoke** 活動的傳送者通常會預期 Bot 透過 HTTP 回應確認收到活動。此活動類型由 Microsoft Bot Framework 保留供內部使用。

messageReaction

當使用者對現有活動做出回應，某些通道會傳送 **messageReaction** 活動給您的 Bot。例如，使用者按了訊息上的「讚」按鈕。**ReplyToId** 屬性會指出使用者回應了哪一個活動。

messageReaction 活動可能對應到通道所定義任何數目的 **messageReactionTypes**。比方說，「讚」或「PlusOne」為通道可能傳送的反應類型。

其他資源

- [傳送及接收活動](#)

- 建立訊息
- 活動類別

建立訊息

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

您的 Bot 會傳送訊息活動以傳達資訊給使用者，同樣也會接收來自使用者的訊息活動。某些訊息可能只包含純文字，而其他訊息可能包含更豐富的內容，例如要讀出的文字、建議的動作、媒體附件、複合式資訊卡 (Rich Card)，和通道特定資料。

本文說明部分的常用訊息屬性。

自訂訊息

若要更充分地掌控訊息的文字格式設定，您可以先使用 [Activity](#) 物件建立自訂訊息並設定所需屬性，再將此物件傳送給使用者。

下列範例說明如何建立自訂 `message` 物件，並設定 `Text`、`TextFormat` 和 `Locale` 屬性。

```
IMessageActivity message = Activity.CreateMessageActivity();
message.Text = "Hello!";
message.TextFormat = "plain";
message.Locale = "en-US";
```

訊息的 `TextFormat` 屬性可用來指定文字格式。`TextFormat` 屬性可以設定為 **plain**、**markdown** 或 **xml**。

`TextFormat` 的預設值為 **markdown**。

附件

訊息物件的 `Attachments` 屬性可用來傳送及接收簡單媒體附件 (影像、音訊、視訊、檔案) 和複合式資訊卡 (Rich Card)。如需詳細資訊，請參閱[將媒體附件新增至訊息](#)和[將複合式資訊卡 \(Rich Card\) 新增至訊息](#)。

實體

訊息的 `Entities` 屬性是開放式 [schema.org](#) 物件的陣列，其允許交換通道與 Bot 之間的通用內容中繼資料。

提及實體

許多通道都支援 Bot 或使用者在交談內容中「提及」某人的功能。若要在訊息中提及使用者，請以 `Mention` 物件填入訊息的 `Entities` 屬性。`Mention` 物件包含下列屬性：

屬性	說明
類型	實體的類型 ("mention")
提及我	<code>ChannelAccount</code> 物件，指出所提及的使用者
文字	<code>Activity.Text</code> 屬性中的文字，表示提及本身 (可以是空白或 null)

此程式碼範例示範如何將 `Mention` 實體新增至 `Entities` 集合。

```
var entity = new Entity();
entity.SetAs(new Mention())
{
    Text = "@johndoe",
    Mentioned = new ChannelAccount()
    {
        Name = "John Doe",
        Id = "UV341235"
    }
});
entities.Add(entity);
```

TIP

嘗試判斷使用者意圖時，Bot 可以忽略提到它的訊息部分。呼叫 `GetMentions` 方法，並評估 `Mention` 回應中傳回的物件。

放置物件

以 `Place` 物件或 `GeoCoordinates` 物件填入訊息的 `Entities` 屬性，即可在訊息中傳達位置相關資訊。

`Place` 物件包含下列屬性：

屬性	說明
類型	實體的類型 ("Place")
位址	說明或 <code>PostalAddress</code> 物件 (未來)
地理區域	<code>GeoCoordinates</code>
HasMap	地圖或 <code>Map</code> 物件的 URL (未來)
Name	位置的名稱

`GeoCoordinates` 物件包含下列屬性：

屬性	說明
類型	實體的類型 ("GeoCoordinates")
Name	位置的名稱
經度	位置的經度 (WGS 84)
緯度	位置的緯度 (WGS 84)
Elevation	位置的提升 (WGS 84)

此程式碼範例示範如何將 `Place` 實體新增至 `Entities` 集合：

```
var entity = new Entity();
entity.SetAs(new Place()
{
    Geo = new GeoCoordinates()
    {
        Latitude = 32.4141,
        Longitude = 43.1123123,
    }
});
entities.Add(entity);
```

取用實體

若要取用實體，請使用 `dynamic` 關鍵字或強型別類別。

此程式碼範例示範如何使用 `dynamic` 關鍵字來處理訊息 `Entities` 屬性內的實體：

```
if (entity.Type == "Place")
{
    dynamic place = entity.Properties;
    if (place.geo.latitude > 34)
        // do something
}
```

此程式碼範例示範如何使用強型別類別來處理訊息 `Entities` 屬性內的實體：

```
if (entity.Type == "Place")
{
    Place place = entity.GetAs<Place>();
    GeoCoordinates geo = place.Geo.ToObject<GeoCoordinates>();
    if (geo.Latitude > 34)
        // do something
}
```

通道資料

訊息活動的 `ChannelData` 屬性可以用來實作通道特有功能。如需詳細資訊，請參閱[實作通道特有功能](#)。

將文字轉換成語音

訊息活動的 `Speak` 屬性可用來指定將由 Bot 在啟用語音的通道上讀出的文字。訊息活動的 `InputHint` 屬性可用來控制用戶端的麥克風和輸入方塊 (如果有的話) 狀態。如需詳細資訊，請參閱[將語音新增至訊息](#)。

建議的動作

訊息活動的 `SuggestedActions` 屬性可用來呈現使用者可點選以提供輸入的按鈕。不同於複合式資訊卡 (Rich Card) 中出現的按鈕 (即使在點選之後，使用者仍可看見並可存取)，出現在建議動作窗格內的按鈕會在使用者進行選取後消失。如需詳細資訊，請參閱[將建議的動作新增至訊息](#)。

後續步驟

Bot 和使用者可以互相傳送訊息。如果訊息較為複雜，Bot 可在訊息中將複合式資訊卡 (Rich Card) 傳送給使用者。複合式資訊卡 (Rich Card) 涵蓋大部分 Bot 常需要的許多簡報與互動案例。

[在訊息中傳送複合式資訊卡 \(Rich Card\)](#)

其他資源

- [活動概觀](#)
- [傳送及接收活動](#)
- [將媒體附件新增至訊息](#)
- [將複合式資訊卡 \(Rich Card\) 新增至訊息](#)
- [將語音新增至訊息](#)
- [將建議的動作新增至訊息](#)
- [實作通道特有功能](#)
- [Activity 類別](#)
- [IMessageActivity 介面](#)

將媒體附件新增至訊息

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

使用者與 Bot 之間的訊息交換可以包含媒體附件 (例如影像、視訊、音訊、檔案等)。[活動](#)物件的 `Attachments` 屬性包含[附件](#)物件的陣列，代表訊息的媒體附件和內含的複合式資訊卡 (Rich Card)。

NOTE

[將複合式資訊卡 \(Rich Card\) 新增至訊息。](#)

新增媒體附件

若要將媒體附件新增至訊息，請建立 `message` 活動的 `Attachment` 物件，並設定 `ContentType`、`ContentUrl` 和 `Name` 屬性。

```
replyMessage.Attachments.Add(new Attachment()
{
    ContentUrl = "https://upload.wikimedia.org/wikipedia/en/a/a6/Bender_Rodriguez.png",
    ContentType = "image/png",
    Name = "Bender_Rodriguez.png"
});
```

如果附件是影像、音訊或視訊，則連接器服務會以可讓[通道](#)在對話內轉譯該附件的方式，將附件資料傳輸至通道。如果附件是檔案，檔案 URL 將會轉譯為對話內的超連結。

其他資源

- [使用頻道偵測器來預覽功能](#)
- [活動概觀](#)
- [建立訊息](#)
- [將複合式資訊卡 \(Rich Card\) 新增至訊息](#)
- [活動類別](#)
- [附件類別](#)

將複合式資訊卡 (Rich Card) 附件新增至訊息

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

使用者與 Bot 之間的訊息交換，可以包含一或多個轉譯為清單或浮動切換的複合式資訊卡。活動物件的 `Attachments` 屬性包含[附件](#)物件的陣列，代表訊息內的複合式資訊卡 (Rich Card) 和媒體附件。

NOTE

如需如何將媒體附件新增至訊息的相關資訊，請參閱[將媒體附件新增至訊息](#)。

複合式資訊卡 (Rich Card) 的類型

Bot Framework 目前支援八種類型的複合式資訊卡 (Rich Card)：

卡片類型	說明
調適型卡片	可包含文字、語音、影像、按鈕和輸入欄位之任意組合的可自訂卡片。請參閱 個別頻道支援 。
動畫卡片	可播放動畫 GIF 或短片的卡片。
音訊卡片	可以播放音訊檔案的卡片。
主圖卡片	通常包含單一大型影像、一或多個按鈕和文字的卡片。
縮圖卡片	通常包含單一縮圖影像、一或多個按鈕和文字的卡片。
收據卡片	讓 Bot 向使用者提供收據的卡片。其中通常包含收據上的項目清單 (稅金和總金額資訊) 和其他文字。
登入卡	可讓 Bot 要求使用者登入的卡片。通常包含文字，以及一或多個使用者可以按一下以起始登入程序的按鈕。
視訊卡片	可播放視訊的卡片。

TIP

若要以清單格式顯示多個複合式資訊卡 (Rich Card)，請將活動的 `AttachmentLayout` 的屬性設定為 [清單]。若要以浮動切換格式顯示多個複合式資訊卡 (Rich Card)，請將活動的 `AttachmentLayout` 的屬性設定為 [浮動切換]。如果通道不支援浮動切換格式，則會以清單格式顯示複合式資訊卡 (Rich Card)，即使 `AttachmentLayout` 屬性指定 [浮動切換] 亦然。

處理複合式資訊卡 (Rich Card) 內的事件

若要處理複合式資訊卡 (Rich Card) 內的事件，請定義 `CardAction` 物件，指定在使用者按一下按鈕或點選卡片

的某區段時所應發生的情況。每個 `CardAction` 物件都包含下列屬性：

屬性	類型	說明
類型	字串	動作的類型 (下表中指定的其中一個值)
標題	字串	按鈕的標題
映像	字串	按鈕的影像 URL
值	字串	執行指定動作類型所需的值

NOTE

調適型卡片內的按鈕不會使用 `CardAction` 物件來建立，而會改用[調適型卡片](#)所定義的結構描述來建立。如需透過範例了解如何將按鈕新增至調適型卡片，請參閱[將調適型卡片新增至訊息](#)。

下表列出 `CardAction.Type` 的有效值，並說明每個類型的 `CardAction.Value` 應有的預期內容：

CARDACTION.TYPE	CARDACTION.VALUE
openUrl	要在內建瀏覽器中開啟的 URL
imBack	要傳送至 Bot 的訊息文字 (來自於按一下按鈕或點選卡片的使用者)。此訊息 (從使用者到 Bot) 會透過裝載對話的用戶端應用程式顯示給所有對話參與者。
postBack	要傳送至 Bot 的訊息文字 (來自於按一下按鈕或點選卡片的使用者)。某些用戶端應用程式可能會在訊息摘要中顯示此文字，讓所有對話參與者都能看見。
call	以下列格式撥打電話的目的地： tel:123123123123
playAudio	待播放音訊的 URL
playVideo	待播放視訊的 URL
showImage	待顯示影像的 URL
downloadFile	待下載檔案的 URL
signin	待起始 OAuth 流程的 URL

將主圖卡新增至訊息

主圖卡通常包含單一大型影像、一或多個按鈕和文字。

下列程式碼範例說明如何建立回覆訊息，且其中須包含三個以浮動切換格式轉譯的主圖卡：

```
Activity replyToConversation = message.CreateReply("Should go to conversation, in carousel format");
replyToConversation.AttachmentLayout = AttachmentLayoutTypes.Carousel;
replyToConversation.Attachments = new List<Attachment>();

Dictionary<string, string> cardContentList = new Dictionary<string, string>();
cardContentList.Add("PigLatin", "https://<ImageUrl1>");
cardContentList.Add("Pork Shoulder", "https://<ImageUrl2>");
cardContentList.Add("Bacon", "https://<ImageUrl3>");

foreach(KeyValuePair<string, string> cardContent in cardContentList)
{
    List<CardImage> cardImages = new List<CardImage>();
    cardImages.Add(new CardImage(url:cardContent.Value ));

    List<CardAction> cardButtons = new List<CardAction>();

    CardAction plButton = new CardAction()
    {
        Value = $"https://en.wikipedia.org/wiki/{cardContent.Key}",
        Type = "openUrl",
        Title = "WikiPedia Page"
    };

    cardButtons.Add(plButton);

    HeroCard plCard = new HeroCard()
    {
        Title = $"I'm a hero card about {cardContent.Key}",
        Subtitle = $"{cardContent.Key} Wikipedia Page",
        Images = cardImages,
        Buttons = cardButtons
    };

    Attachment plAttachment = plCard.ToAttachment();
    replyToConversation.Attachments.Add(plAttachment);
}

var reply = await connector.Conversations.SendToConversationAsync(replyToConversation);
```

將縮圖卡新增至訊息

縮圖卡通常包含單一縮圖影像、一或多個按鈕和文字。

下列程式碼範例說明如何建立回覆訊息，且其中須包含兩個以清單格式轉譯的縮圖卡：

```
Activity replyToConversation = message.CreateReply("Should go to conversation, in list format");
replyToConversation.AttachmentLayout = AttachmentLayoutTypes.List;
replyToConversation.Attachments = new List<Attachment>();

Dictionary<string, string> cardContentList = new Dictionary<string, string>();
cardContentList.Add("PigLatin", "https://<ImageUrl1>");
cardContentList.Add("Pork Shoulder", "https://<ImageUrl2>");

foreach(KeyValuePair<string, string> cardContent in cardContentList)
{
    List<CardImage> cardImages = new List<CardImage>();
    cardImages.Add(new CardImage(url:cardContent.Value ));

    List<CardAction> cardButtons = new List<CardAction>();

    CardAction plButton = new CardAction()
    {
        Value = $"https://en.wikipedia.org/wiki/{cardContent.Key}",
        Type = "openUrl",
        Title = "WikiPedia Page"
    };

    cardButtons.Add(plButton);

    ThumbnailCard plCard = new ThumbnailCard()
    {
        Title = $"I'm a thumbnail card about {cardContent.Key}",
        Subtitle = $"{cardContent.Key} Wikipedia Page",
        Images = cardImages,
        Buttons = cardButtons
    };

    Attachment plAttachment = plCard.ToAttachment();
    replyToConversation.Attachments.Add(plAttachment);
}

var reply = await connector.Conversations.SendToConversationAsync(replyToConversation);
```

將收據卡新增至訊息

收據卡可讓 Bot 向使用者提供收據。其中通常包含收據上的項目清單（稅金和總金額資訊）和其他文字。

下列程式碼範例說明如何建立包含收據卡的回覆訊息：

```

Activity replyToConversation = message.CreateReply("Should go to conversation");
replyToConversation.Attachments = new List<Attachment>();

List<CardImage> cardImages = new List<CardImage>();
cardImages.Add(new CardImage(url: "https://<imageUrl1>" ));

List<CardAction> cardButtons = new List<CardAction>();

CardAction plButton = new CardAction()
{
    Value = $"https://en.wikipedia.org/wiki/PigLatin",
    Type = "openUrl",
    Title = "WikiPedia Page"
};

cardButtons.Add(plButton);

ReceiptItem lineItem1 = new ReceiptItem()
{
    Title = "Pork Shoulder",
    Subtitle = "8 lbs",
    Text = null,
    Image = new CardImage(url: "https://<ImageUrl1>"),
    Price = "16.25",
    Quantity = "1",
    Tap = null
};

ReceiptItem lineItem2 = new ReceiptItem()
{
    Title = "Bacon",
    Subtitle = "5 lbs",
    Text = null,
    Image = new CardImage(url: "https://<ImageUrl2>"),
    Price = "34.50",
    Quantity = "2",
    Tap = null
};

List<ReceiptItem> receiptList = new List<ReceiptItem>();
receiptList.Add(lineItem1);
receiptList.Add(lineItem2);

ReceiptCard plCard = new ReceiptCard()
{
    Title = "I'm a receipt card, isn't this bacon expensive?",
    Buttons = cardButtons,
    Items = receiptList,
    Total = "112.77",
    Tax = "27.52"
};

Attachment plAttachment = plCard.ToAttachment();
replyToConversation.Attachments.Add(plAttachment);

var reply = await connector.Conversations.SendToConversationAsync(replyToConversation);

```

將登入卡新增至訊息

登入卡可讓 Bot 要求使用者登入。其中通常包含文字，以及一或多個可讓使用者點選以起始登入程序的按鈕。

下列程式碼範例說明如何建立包含登入卡的回覆訊息：

```

Activity replyToConversation = message.CreateReply("Should go to conversation");
replyToConversation.Attachments = new List<Attachment>();

List<CardAction> cardButtons = new List<CardAction>();

CardAction plButton = new CardAction()
{
    Value = $"https://<OAuthSignInURL",
    Type = "signin",
    Title = "Connect"
};

cardButtons.Add(plButton);

SigninCard plCard = new SigninCard(title: "You need to authorize me", button: plButton);

Attachment plAttachment = plCard.ToAttachment();
replyToConversation.Attachments.Add(plAttachment);

var reply = await connector.Conversations.SendToConversationAsync(replyToConversation);

```

將調適型卡片新增至訊息

調適型卡片可包含文字、語音、影像、按鈕和輸入欄位的任意組合。調適型卡片會使用[調適型卡片](#)中指定的 JSON 格式來建立，讓您能夠完整控制卡片的內容和格式。

若要使用 .NET 建立調適型卡片，請安裝 [AdaptiveCards](#) NuGet 套件。然後，請利用[調適型卡片](#)網站中的資訊了解調適型卡片的結構描述、探索調適型卡片的元素，並查看 JSON 範例，用以建立具有不同組合和複雜度的卡片。此外，您可以使用互動式視覺化檢視，以設計調適型卡片承載，以及預覽卡片輸出。

下列程式碼範例示範如何建立包含調適型卡片作為行事曆提醒的訊息：

```

Activity replyToConversation = message.CreateReply("Should go to conversation");
replyToConversation.Attachments = new List<Attachment>();

AdaptiveCard card = new AdaptiveCard();

// Specify speech for the card.
card.Speak = "<s>Your meeting about \"Adaptive Card design session\"<brack strength='weak' /> is starting at 12:30pm</s><s>Do you want to snooze <brack strength='weak' /> or do you want to send a late notification to the attendees?</s>";

// Add text to the card.
card.Body.Add(new TextBlock()
{
    Text = "Adaptive Card design session",
    Size = TextSize.Large,
    Weight = TextWeight.Bolder
});

// Add text to the card.
card.Body.Add(new TextBlock()
{
    Text = "Conf Room 112/3377 (10)"
});

// Add text to the card.
card.Body.Add(new TextBlock()
{
    Text = "12:30 PM - 1:30 PM"
});

// Add list of choices to the card.
card.Body.Add(new ChoiceSet()

```

```

{
    Id = "snooze",
    Style = ChoiceInputStyle.Compact,
    Choices = new List<Choice>()
    {
        new Choice() { Title = "5 minutes", Value = "5", IsSelected = true },
        new Choice() { Title = "15 minutes", Value = "15" },
        new Choice() { Title = "30 minutes", Value = "30" }
    }
};

// Add buttons to the card.
card.Actions.Add(new OpenUrlAction()
{
    Url = "http://foo.com",
    Title = "Snooze"
});

card.Actions.Add(new OpenUrlAction()
{
    Url = "http://foo.com",
    Title = "I'll be late"
});

card.Actions.Add(new OpenUrlAction()
{
    Url = "http://foo.com",
    Title = "Dismiss"
});

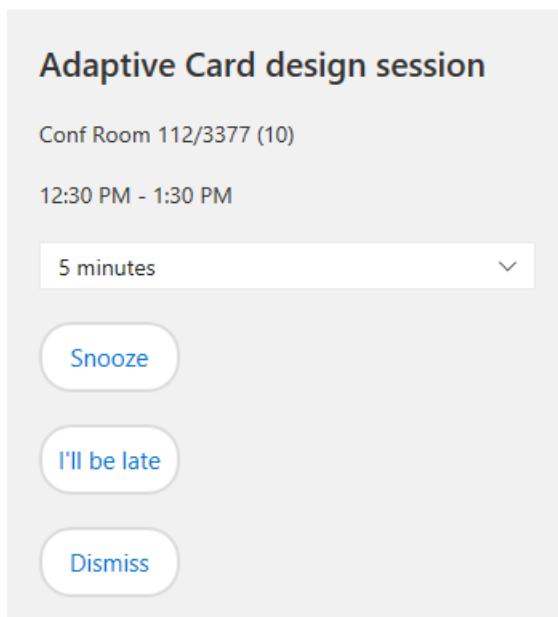
// Create the attachment.
Attachment attachment = new Attachment()
{
    ContentType = AdaptiveCard.ContentType,
    Content = card
};

replyToConversation.Attachments.Add(attachment);

var reply = await connector.Conversations.SendToConversationAsync(replyToConversation);

```

產生的卡片會包含三個文字區塊、一個輸入欄位 (選擇清單) 和三個按鈕：



其他資源

- 使用頻道偵測器來預覽功能
- 調適型卡片
- 活動概觀
- 建立訊息
- 將媒體附件新增至訊息
- 活動類別
- 附件類別

將語音新增至訊息

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

如果您要為具備語音功能的通道 (例如 Cortana) 建置 Bot，您可以建構訊息，其中指定要由 Bot 讀出的文字。您也可以指定[輸入提示](#)，藉由指出您的 Bot 要接受、需要或忽略使用者輸入，來嘗試影響用戶端的麥克風狀態。

指定要由 Bot 讀出的文字

使用適用於 .NET 的 Bot Framework SDK 時，有多種方式可用來指定要由 Bot 在具備語音功能的通道上讀出的文字。您可以設定訊息的 `Speak` 屬性、呼叫 `IDialogContext.SayAsync()` 方法，或在使用內建提示傳送訊息時指定提示選項 `speak` 和 `retrySpeak`。

IMessageActivity.Speak

如果您要建立訊息並設定其個別屬性，您可以設定訊息的 `Speak` 屬性來指定要由您的 Bot 讀出的文字。下列程式碼範例會建立一則訊息，其中指定要顯示的文字和要讀出的文字，並指出 Bot 會[接受使用者輸入](#)。

```
Activity reply = activity.CreateReply("This is the text that will be displayed.");
reply.Speak = "This is the text that will be spoken.";
reply.InputHint = InputHints.AcceptingInput;
await connector.Conversations.ReplyToActivityAsync(reply);
```

IDialogContext.SayAsync()

如果您正在使用對話，您也可以呼叫 `SayAsync()` 方法來建立和傳送訊息，並在訊息內指定要讀出的文字、要顯示的文字和其他選項。下列程式碼範例會建立一則訊息，其中指定要顯示的文字和要讀出的文字。

```
await context.SayAsync(text: "Thank you for your order!", speak: "Thank you for your order!");
```

提示選項

使用任何內建提示時，您可以設定選項 `speak` 和 `retrySpeak`，以指定要由您的 Bot 讀出的文字。下列程式碼範例會建立一則提示，其中指定要顯示的文字、一開始要讀出的文字，以及等待使用者輸入一段時間後要讀出的文字。它會使用 [SSML](#) 格式，來表示應以適度強調的方式讀出 "sure" 這個字。

```
PromptDialog.Confirm(context, AfterResetAsync,
    new PromptOptions<string>(prompt: "Are you sure that you want to cancel this transaction?",
        speak: "Are you <emphasis level=\"moderate\">sure</emphasis> that you want to cancel this
        transaction?",  

        retrySpeak: "Are you <emphasis level=\"moderate\">sure</emphasis> that you want to cancel this
        transaction?"));
```

語音合成標記語言 (SSML)

若要指定要由 Bot 讀出的文字，您可以提供 Bot 格式為語音合成標記語言 (SSML) 的字串。SSML 是以 XML 為基礎的標記語言 (而且必須是有效的 XML)，可讓您控制 Bot 語音，例如語音、速率、磁碟區、發音及音調等各種特性。如需 SSML 的詳細資訊，請參閱[語音合成標記語言參考 \(英文\)](#)。

當提供 SSML 格式化字串時，可能會省略外部的 SSML 包裝函式元素。

輸入提示

當您在啟用語音功能的通道上傳送訊息時，您也可以藉由包含輸入提示以指出 Bot 要接受、需要或忽略使用者輸入，來嘗試影響用戶端的麥克風狀態。如需詳細資訊，請參閱[將輸入提示新增至訊息](#)。

範例程式碼

如需完整範例以了解如何使用適用於 .NET 的 Bot Framework SDK 建立具備語音功能的 Bot，請參閱 GitHub 中的[骰子機技能範例](#)。

其他資源

- [建立訊息](#)
- [將輸入提示新增至訊息](#)
- [語音合成標記語言 \(SSML\) \(英文\)](#)
- [骰子機技能範例 \(GitHub\) \(英文\)](#)
- [Activity 類別 \(英文\)](#)
- [IMessageActivity 介面](#)
- [DialogContext 類別](#)
- [Prompt 類別](#)

將輸入提示新增至訊息

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

您可藉由指定訊息的輸入提示，指出您的 Bot 在訊息傳遞給用戶端之後，會接受、需要或忽略使用者輸入。這可讓用戶端為許多通道設定相應的使用者輸入控制項狀態。例如，如果訊息的輸入提示指出 Bot 會忽略使用者輸入，則用戶端可關閉麥克風並停用輸入方塊，以防止使用者提供輸入。

接受輸入

若要指出 Bot 要被動接受輸入，但不等候使用者回應，請將訊息的輸入提示設定為 `InputHints.AcceptingInput`。在許多通道上，這會啟用用戶端的輸入方塊並關閉麥克風，但使用者仍可存取。例如，如果使用者按住麥克風按鈕，Cortana 會開啟麥克風並接受使用者輸入。下列程式碼範例會建立訊息，指出 Bot 會接受使用者輸入。

```
Activity reply = activity.CreateReply("This is the text that will be displayed.");
reply.Speak = "This is the text that will be spoken.";
reply.InputHint = InputHints.AcceptingInput;
await connector.Conversations.ReplyToActivityAsync(reply);
```

需要輸入

若要指出 Bot 要等候使用者回應，請將訊息的輸入提示設定為 `InputHints.ExpectingInput`。在許多通道上，這會啟用用戶端的輸入方塊並開啟麥克風。下列程式碼範例會建立訊息來指出 Bot 需要使用者輸入。

```
Activity reply = activity.CreateReply("This is the text that will be displayed.");
reply.Speak = "This is the text that will be spoken.";
reply.InputHint = InputHints.ExpectingInput;
await connector.Conversations.ReplyToActivityAsync(reply);
```

忽略輸入

若要指出 Bot 尚未準備好接收使用者輸入，請將訊息的輸入提示設定為 `InputHints.IgnoringInput`。在許多通道上，這會停用用戶端的輸入方塊並關閉麥克風。下列程式碼範例會建立訊息來指出 Bot 會忽略使用者輸入。

```
Activity reply = activity.CreateReply("This is the text that will be displayed.");
reply.Speak = "This is the text that will be spoken.";
reply.InputHint = InputHints.IgnoringInput;
await connector.Conversations.ReplyToActivityAsync(reply);
```

輸入提示的預設值

如果您未設定訊息的輸入提示，Bot Framework SDK 會使用下列邏輯自動為您設定：

- 如果 Bot 傳送提示，訊息的輸入提示將指定 Bot 需要輸入。
- 如果 Bot 傳送單一訊息，訊息的輸入提示將指定 Bot 會接受輸入。

- 如果 Bot 傳送一系列的連續訊息，系列中的所有訊息（但最後一則訊息除外）的輸入提示都將指定 Bot 會忽略輸入，而系列中最後一則訊息的輸入提示將指定 Bot 會接受輸入。

其他資源

- [建立訊息](#)
- [將語音新增至訊息](#)
- [Activity 類別 \(英文\)](#)
- [InputHints 類別](#)

將建議的動作新增至訊息

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

建議動作可讓您的 Bot 顯示可供使用者點選，以提供輸入的按鈕。建議動作會出現在編輯器附近，讓使用者只要點選按鈕即可回答問題或進行選取，而無需使用鍵盤輸入回應，藉以提升使用者體驗。不同於複合式資訊卡 (Rich Card) 中出現的按鈕 (即使在點選之後，使用者仍可看見並可存取)，出現在建議動作窗格內的按鈕會在使用者進行選取後消失。這可以避免使用者在對話中點選過時的按鈕，並簡化 Bot 的開發 (因為您不需要再說明該情境)。

TIP

使用[通道偵測器](#)來查看建議動作的外觀，以及在各種通道上的運作方式。

傳送建議的動作

若要將建議的動作新增至訊息，請將活動的 `SuggestedActions` 屬性設為 `CardAction` 物件清單，這些物件代表要向使用者呈現的按鈕。

此程式碼範例示範如何建立一則訊息，向使用者呈現三種建議的動作：

```
var reply = activity.CreateReply("I have colors in mind, but need your help to choose the best one.");
reply.Type = ActivityTypes.Message;
reply.TextFormat = TextFormatTypes.Plain;

reply.SuggestedActions = new SuggestedActions()
{
    Actions = new List<CardAction>()
    {
        new CardAction(){ Title = "Blue", Type=ActionTypes.ImBack, Value="Blue" },
        new CardAction(){ Title = "Red", Type=ActionTypes.ImBack, Value="Red" },
        new CardAction(){ Title = "Green", Type=ActionTypes.ImBack, Value="Green" }
    }
};
```

當使用者點選其中一個建議的動作時，Bot 會收到來自使用者的訊息，其中包含對應動作的 `Value`。

其他資源

- [使用頻道偵測器來預覽功能](#)
- [活動概觀](#)
- [建立訊息](#)
- [Activity 類別 \(英文\)](#)
- [IMessageActivity 介面](#)
- [CardAction 類別](#)
- [SuggestedActions 類別](#)

傳送及接收活動

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

Bot Framework Connector 提供單一 REST API，讓 Bot 能夠跨多個通道 (例如 Skype、電子郵件、Slack 等) 進行通訊。它可藉由將訊息從 Bot 轉送至通道以及從通道轉送至 Bot，讓 Bot 和使用者之間的通訊更為順暢。

本文說明如何透過適用於 .NET 的 Bot Framework SDK 使用 Connector，在通道上的 Bot 和使用者之間交換資訊。

NOTE

雖然您可以獨佔方式使用本文所述的技術來建構 Bot，但 Bot Framework SDK 會提供額外功能 (例如[對話](#)和 [FormFlow](#))，可以簡化管理交談流程和狀態的程序，並且能夠以更簡單的方式來合併 Language Understanding 之類的認知服務。

建立連接器用戶端

[ConnectorClient](#) 類別包含 Bot 用來與通道上的使用者進行通訊的方法。當您的 Bot 接收到來自 Connector 的 [Activity](#) 物件時，它應該使用針對該活動指定的 [ServiceUrl](#) 來建立連接器用戶端，後續將使用它來產生回應。

```
public async Task<HttpResponseMessage> Post([FromBody]Activity activity)
{
    var connector = new ConnectorClient(new Uri(activity.ServiceUrl));
    . .
}
```

TIP

由於通道的端點可能不穩定，因此，您的 Bot 應該盡可能直接與 Connector 在 [Activity](#) 物件中指定的端點進行通訊 (而不依賴快取的端點)。

如果您的 Bot 需要起始對話，它可以針對指定的通道使用快取的端點 (因為在該案例中，將不會有任何傳入的 [Activity](#) 物件)，但它通常應該會重新整理快取的端點。

建立回覆

Connector 會使用 [Activity](#) 物件，在 Bot 和通道 (使用者) 之間來回傳遞資訊。每個活動都包含可用於將訊息路由傳送至適當目的地的資訊，以及有關訊息建立者 ([From](#) 屬性)、訊息的內容和訊息收件者 ([Recipient](#) 屬性) 的資訊。

當您的 Bot 接收來自 Connector 的活動時，傳入活動的 [Recipient](#) 屬性會在該對話中指定 Bot 的身分識別。由於某些通道 (例如 Slack) 會在將 Bot 新增至對話時為其指派新的身分識別，因此，該 Bot 應一律使用傳入活動的 [Recipient](#) 屬性值作為其回應中 [From](#) 屬性的值。

雖然您可以自行從頭開始建立並初始化傳出的 [Activity](#) 物件，但 Bot Framework SDK 還是會提供更簡單的方法來建立回覆。藉由使用傳入活動的 [createReply](#) 方法，您只需指定回應的訊息文字，而傳出的活動會使用自動填入的 [Recipient](#)、[From](#) 和 [Conversation](#) 屬性來建立。

```
Activity reply = activity.CreateReply($"You sent {activity.Text} which was {length} characters");
```

傳送回覆

一旦建立回覆之後，您就可以藉由呼叫連接器用戶端的 `ReplyToActivity` 方法來傳送它。Connector 將使用適當的通道語意來傳遞回覆。

```
await connector.Conversations.ReplyToActivityAsync(reply);
```

TIP

如果您的 Bot 正在回覆使用者的訊息，請一律使用 `ReplyToActivity` 方法。

傳送 (非回覆) 訊息

如果您的 Bot 是對話的一部分，它可以藉由呼叫 `SendToConversation` 方法，將不屬於直接回覆的訊息傳送給來自使用者的任何訊息。

```
await connector.Conversations.SendToConversationAsync((Activity)newMessage);
```

您可以使用 `CreateReply` 方法來初始化新訊息（這會自動設定訊息的 `Recipient`、`From` 和 `Conversation` 屬性）。或者，您可以使用 `CreateMessageActivity` 方法來建立新訊息，並自行設定所有的屬性值。

NOTE

Bot Framework 對於 Bot 可傳送的訊息數目並無任何限制。不過，大部分的通道會強制執行節流限制，限制 Bot 在短時間內傳送大量的訊息。此外，如果 Bot 連續快速地傳送多則訊息，通道可能不一定會依照正確順序轉譯訊息。

開始對話

您的 Bot 有時可能需要起始與一或多位使用者的對話。您可以呼叫 `CreateDirectConversation` 方法（適用於與單一使用者的私人對話）或 `CreateConversation` 方法（適用於與多位使用者的群組對話）來擷取 `ConversationAccount` 物件，藉以開始對話。然後，建立訊息，並呼叫 `SendToConversation` 方法來傳送它。若要使用 `CreateDirectConversation` 方法或 `CreateConversation` 方法，您必須先使用目標通道的服務 URL（如果您已從上一個訊息中保存它，則可從快取中擷取），來[建立連接器用戶端](#)。

NOTE

並非所有通道都支援群組對話。若要判斷通道是否支援群組對話，請參閱通道的文件。

此程式碼範例使用 `CreateDirectConversation` 方法來建立與單一使用者的私人對話。

```
var userAccount = new ChannelAccount(name: "Larry", id: "@UV357341");
var connector = new ConnectorClient(new Uri(activity.ServiceUrl));
var conversationId = await connector.Conversations.CreateDirectConversationAsync(botAccount, userAccount);

IMessageActivity message = Activity.CreateMessageActivity();
message.From = botAccount;
message.Recipient = userAccount;
message.Conversation = new ConversationAccount(id: conversationId.Id);
message.Text = "Hello, Larry!";
message.Locale = "en-US";
await connector.Conversations.SendToConversationAsync((Activity)message);
```

此程式碼範例使用 [CreateConversation](#) 方法來建立與多位使用者的群組對話。

```
var connector = new ConnectorClient(new Uri(incomingMessage.ServiceUrl));

List<ChannelAccount> participants = new List<ChannelAccount>();
participants.Add(new ChannelAccount("joe@contoso.com", "Joe the Engineer"));
participants.Add(new ChannelAccount("sara@contoso.com", "Sara in Finance"));

ConversationParameters cpMessage = new ConversationParameters(message.Recipient, true, participants, "Quarter
End Discussion");
var conversationId = await connector.Conversations.CreateConversationAsync(cpMessage);

IMessageActivity message = Activity.CreateMessageActivity();
message.From = botAccount;
message.Recipient = new ChannelAccount("lydia@contoso.com", "Lydia the CFO");
message.Conversation = new ConversationAccount(id: conversationId.Id);
message.ChannelId = incomingMessage.ChannelId;
message.Text = "Hello, everyone!";
message.Locale = "en-US";

await connector.Conversations.SendToConversationAsync((Activity)message);
```

其他資源

- [活動概觀](#)
- [建立訊息](#)
- [適用於 .NET 的 Bot Framework SDK 參考](#)
- [Activity 類別 \(英文\)](#)
- [ConnectorClient 類別](#)

實作全域訊息處理常式

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

使用者通常會嘗試使用「說明」、「取消」或「重新開始」等關鍵字，來存取 Bot 內的某些功能。這通常會發生在對話中，而 Bot 這時會預期不同的回應。藉由實作全域訊息處理常式，您可以將 Bot 設計為會以妥善的方式處理這類要求。處理常式會檢查使用者輸入中是否有您指定的關鍵字（例如，「說明」、「取消」或「重新開始」），並做出適當回應。



Do you confirm this order?

Yes

No

Help?



I see you have questions, anything specific you want me to help you with?

NOTE

在全域訊息處理常式中定義邏輯，可讓所有對話方塊都能存取該邏輯。您可以設定個別的對話方塊和提示，安全地忽略關鍵字。

接聽使用者輸入中的關鍵字

下列逐步解說示範如何使用適用於 .NET 的 Bot Framework SDK 來實作全域訊息處理常式。

首先，`Global.asax.cs` 會註冊 `GlobalMessageHandlersBotModule`，實作如下所示。在此範例中，模組會註冊兩個可評分的設定：一個用於管理變更設定的要求 (`SettingsScorable`)，另一個用於管理要取消的要求 (`CancelScoreable`)。

```

public class GlobalMessageHandlersBotModule : Module
{
    protected override void Load(ContainerBuilder builder)
    {
        base.Load(builder);

        builder
            .Register(c => new SettingsScorable(c.Resolve<IDialogTask>()))
            .As<IScorable<IActivity, double>>()
            .InstancePerLifetimeScope();

        builder
            .Register(c => new CancelScorable(c.Resolve<IDialogTask>()))
            .As<IScorable<IActivity, double>>()
            .InstancePerLifetimeScope();
    }
}

```

`CancelScorable` 包含定義觸發程序的 `PrepareAsync` 方法：如果訊息文字是「取消」，將會觸發這個可評分的設定。

```

protected override async Task<string> PrepareAsync(IActivity activity, CancellationToken token)
{
    var message = activity as IMessageActivity;
    if (message != null && !string.IsNullOrWhiteSpace(message.Text))
    {
        if (message.Text.Equals("cancel", StringComparison.InvariantCultureIgnoreCase))
        {
            return message.Text;
        }
    }
    return null;
}

```

收到「取消」要求時，`CancelScoreable` 內的 `PostAsync` 方法會重設對話堆疊。

```

protected override async Task PostAsync(IActivity item, string state, CancellationToken token)
{
    this.task.Reset();
}

```

收到「變更設定」要求時，`SettingsScorable` 內的 `PostAsync` 方法會叫用 `SettingsDialog`（將要求傳遞至該對話），從而將 `SettingsDialog` 新增至對話堆疊的最上方，使其可控制對話。

```

protected override async Task PostAsync(IActivity item, string state, CancellationToken token)
{
    var message = item as IMessageActivity;
    if (message != null)
    {
        var settingsDialog = new SettingsDialog();
        var interruption = settingsDialog.Void<object, IMessageActivity>();
        this.task.Call(interruption, null);
        await this.task.PollAsync(token);
    }
}

```

範例程式碼

如需示範如何使用適用於 .NET 的 Bot Framework SDK 來實作全域訊息處理常式的完整範例，請參閱 GitHub 中的[全域訊息處理常式範例](#)（英文）。

其他資源

- [設計和控制交談流程](#)
- [適用於 .NET 的 Bot Framework SDK 參考](#)
- [全域訊息處理常式範例 \(GitHub\) \(英文\)](#)

攔截訊息

2019/3/21 • • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

Bot Framework SDK 中的**中介軟體**功能可讓 Bot 攔截使用者與 Bot 之間交換的所有訊息。針對每個攔截的訊息，您可以選擇執行如下的動作：將訊息儲存到您所指定的資料存放區，以建立對話記錄，或是以某種方式檢查訊息，並執行程式碼所指定的任何動作。

NOTE

Bot Framework 不會自動儲存交談的詳細資料，因此這麼做可能會擷取 Bot 和使用者不想要與外界共用的個人資訊。如果您的 Bot 儲存了對話的詳細資料，它即應將此情況告知使用者，並說明這些資料的使用方式。

攔截和記錄訊息

下列程式碼範例顯示如何使用適用於 .NET 的 Bot Framework SDK 的**中介軟體**概念，攔截使用者與 Bot 之間交換的訊息。

首先，建立 `DebugActivityLogger` 類別並定義 `LogAsync` 方法，以指定攔截到的每個訊息會執行哪些動作。此範例只會列印每個訊息的部分資訊。

```
public class DebugActivityLogger : IActivityLogger
{
    public async Task LogAsync(IActivity activity)
    {
        Debug.WriteLine($"From:{activity.From.Id} - To:{activity.Recipient.Id} - Message:
{activity.AsMessageActivity()?.Text}");
    }
}
```

然後，將下列程式碼新增至 `Global.asax.cs`。每個使用者與 Bot (任一方向) 之間交換的訊息，將會立即觸發 `DebugActivityLogger` 類別中的 `LogAsync` 方法。

```
public class WebApiApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        var builder = new ContainerBuilder();
        builder.RegisterType<DebugActivityLogger>().AsImplementedInterfaces().InstancePerDependency();
        builder.Update(Conversation.Container);

        GlobalConfiguration.Configure(WebApiConfig.Register);
    }
}
```

雖然此範例只會列印每個訊息的部分資訊，您可以更新 `LogAsync` 方法，以指定您想要針對每個訊息採取的動作。

範例程式碼

如需顯示如何使用適用於 .NET 的 Bot Framework SDK 擋截和記錄訊息的完整範例，請參閱 GitHub 中的[中介軟體範例](#)。

其他資源

- [適用於 .NET 的 Bot Framework SDK 參考](#)
- [中介軟體範例 \(GitHub\)](#)

傳送主動訊息

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

一般而言，Bot 直接傳送給使用者的每個訊息都與使用者先前的輸入相關。在某些情況下，Bot 可能需要將與目前對話主題或使用者最後傳送的訊息不直接相關的訊息傳送給使用者。這些類型的訊息稱為主動訊息。

主動訊息可用於各種情況。如果 Bot 設定了計時器或提醒，它必須在到達該時間時通知使用者。或者，如果 Bot 從外部系統收到通知，它可能需要將該資訊立即傳達給使用者。比方說，如果使用者先前已要求 Bot 監視產品的價格，則 Bot 可以在產品的價格下降了 20% 時對使用者發出警報。或者，如果 Bot 需要一些時間來編譯對使用者問題的回應，它可能會通知使用者已延遲，並且在此同時允許交談繼續進行。當 Bot 完成問題回應的編譯時，它會與使用者分享該資訊。

在 Bot 中實作主動訊息時：

- 請「勿」在短時間內傳送數個主動訊息。某些通道會強制限制 Bot 將訊息傳送給使用者的頻率，如果違反了這些限制，將會停用 Bot。
- 請「勿」將主動訊息傳送給先前未與 Bot 互動或透過其他方式（例如電子郵件或簡訊）與 Bot 連繫的使用者。

主動訊息可能會造成非預期的行為。請考慮下列狀況。



What city are you travelling to?

London



What is your planned date for the trip?



Good news! Your preferred hotel in Las Vegas is offering a discount! Want to book a trip?

Wait, what?



Sorry, this isn't a valid date for your London trip...

Bot you're drunk...

在此範例中，使用者先前已要求 Bot 監視拉斯維加斯旅館的價格。Bot 會啟動一個在過去幾天內持續執行的背景監視工作。在交談中，當背景工作觸發拉斯維加斯旅館折扣的相關通知訊息時，使用者正在預約到倫敦的行程。Bot 會將這項資訊插入交談中，進而產生讓人混淆的使用者體驗。

Bot 應該如何處理這種情況？

- 等待目前旅遊預約完成，然後傳遞通知。這種方式可將混亂情況降到最低，但延遲傳達資訊可能會導致使用者錯過拉斯維加斯旅館的低價機會。
- 取消目前的旅遊預約流程，並立即傳遞通知。這種方式可及時提供資訊，但可能會強制使用者重新開始進行旅遊預約，而讓他們感到沮喪。
- 中斷目前的預約，清楚地將交談主題變更為拉斯維加斯旅館直到使用者回應，然後切換回進行中的旅遊預約並從中斷處繼續執行。這種方式似乎是最佳選擇，但它會對 Bot 開發人員和使用者帶來複雜性。

大多數情況下，您的 Bot 將以某種方式組合使用臨機操作的主動訊息和其他技術來處理這種情況。

主動式訊息的類型

臨機操作主動式訊息是最簡單的主動式訊息類型。Bot 只會在每次觸發時將訊息插入對話，不會顧及使用者目前是否與 Bot 在其他對話主題中，也不會嘗試以任何方式變更對話。

若要更順利地處理通知，請考慮使用其他方式將通知整合到對話流程中，例如在對話狀態中設定旗標，或將通知新增至佇列。

傳送臨機操作的主動訊息

下列程式碼範例示範如何使用適用於 .NET 的 Bot Framework SDK 傳送臨機操作的主動式訊息。

為了能夠將臨機操作的訊息傳送給使用者，Bot 必須先收集並儲存一些來自目前對話的使用者相關資訊。

```
public virtual async Task MessageReceivedAsync(IDialogContext context, IAwaitable<IMessageActivity> result)
{
    var message = await result;

    // Extract data from the user's message that the bot will need later to send an ad hoc message to the
    user.

    // Store the extracted data in a custom class "ConversationStarter" (not shown here).

    ConversationStarter.toId = message.From.Id;
    ConversationStarter.toName = message.From.Name;
    ConversationStarter.fromId = message.Recipient.Id;
    ConversationStarter.fromName = message.Recipient.Name;
    ConversationStarter.serviceUrl = message.ServiceUrl;
    ConversationStarter.channelId = message.ChannelId;
    ConversationStarter.conversationId = message.Conversation.Id;

    // (Save this information somewhere that it can be accessed later, such as in a database.)

    await context.PostAsync("Hello user, good to meet you! I now know your address and can send you
    notifications in the future.");
    context.Wait(MessageReceivedAsync);
}
```

NOTE

為了簡單起見，此範例並未指定如何儲存使用者資料。如何儲存資料並不重要，只要 Bot 稍後可擷取資料即可。

既然資料已儲存，Bot 只需擷取資料、建構臨機操作的主動式訊息，然後加以傳送。

```

// Use the data stored previously to create the required objects.
var userAccount = new ChannelAccount(toId,toName);
var botAccount = new ChannelAccount(fromId, fromName);
var connector = new ConnectorClient(new Uri(serviceUrl));

// Create a new message.
IMessageActivity message = Activity.CreateMessageActivity();
if (!string.IsNullOrEmpty(conversationId) && !string.IsNullOrEmpty(channelId))
{
    // If conversation ID and channel ID was stored previously, use it.
    message.ChannelId = channelId;
}
else
{
    // Conversation ID was not stored previously, so create a conversation.
    // Note: If the user has an existing conversation in a channel, this will likely create a new conversation
    // window.
    conversationId = (await connector.Conversations.CreateDirectConversationAsync( botAccount,
    userAccount)).Id;
}

// Set the address-related properties in the message and send the message.
message.From = botAccount;
message.Recipient = userAccount;
message.Conversation = new ConversationAccount(id: conversationId);
message.Text = "Hello, this is a notification";
message.Locale = "en-us";
await connector.Conversations.SendToConversationAsync((Activity)message);

```

NOTE

如果 Bot 指定先前儲存的對話識別碼，很可能會將訊息傳遞給用戶端上現有對話視窗中的使用者。如果 Bot 產生新的對話識別碼，即會將訊息傳遞給用戶端上新對話視窗中的使用者，但前提是用戶端支援多個對話視窗。

傳送對話方塊式的主動訊息

下列程式碼範例示範如何使用適用於 .NET 的 Bot Framework SDK 傳送對話式主動訊息。

為了能夠將對話型的主動式訊息傳送給使用者，Bot 必須先收集並儲存來自目前對話的相關資訊。

```

public virtual async Task MessageReceivedAsync(IDialogContext context, IAwaitable<IMessageActivity> result)
{
    var message = await result;

    // Store information about this specific point the conversation, so that the bot can resume this
    // conversation later.
    var conversationReference = message.ToConversationReference();
    ConversationStarter.conversationReference = JsonConvert.SerializeObject(conversationReference);

    await context.PostAsync("Greetings, user! I now know how to start a proactive message to you.");
    context.Wait(MessageReceivedAsync);
}

```

傳送訊息時，Bot 會建立新的對話，並將它新增至對話堆疊的最上層。新的對話會取得對話的控制權、傳遞主動式訊息、關閉，然後將控制權傳回給堆疊中的上一個對話。

```

// This will interrupt the conversation and send the user to SurveyDialog, then wait until that's done
public static async Task Resume()
{
    // Recreate the message from the conversation reference that was saved previously.
    var message = JsonConvert.DeserializeObject<ConversationReference>
(conversationReference).GetPostToBotMessage();
    var client = new ConnectorClient(new Uri(message.ServiceUrl));

    // Create a scope that can be used to work with state from bot framework.
    using (var scope = DialogModule.BeginLifetimeScope(Conversation.Container, message))
    {
        var botData = scope.Resolve<IBotData>();
        await botData.LoadAsync(CancellationToken.None);

        // This is our dialog stack.
        var task = scope.Resolve<IDialogTask>();

        // Create the new dialog and add it to the stack.
        var dialog = new SurveyDialog();
        // interrupt the stack. This means that we're stopping whatever conversation that is currently
happening with the user
        // Then adding this stack to run and once it's finished, we will be back to the original conversation
task.Call(dialog.Void<object, IMessageActivity>(), null);

        await task.PollAsync(CancellationToken.None);

        // Flush the dialog stack back to its state store.
        await botData.FlushAsync(CancellationToken.None);
    }
}

```

`SurveyDialog` 會控制對話，直到該對話完成為止。完成其工作時，它會呼叫 `context.Done` 然後關閉，將控制權傳回給上一個對話。

```

[Serializable]
public class SurveyDialog : IDialog<object>
{
    public async Task StartAsync(IDialogContext context)
    {
        await context.PostAsync("Hello, I'm the survey dialog. I'm interrupting your conversation to ask you a
question. Type \"done\" to resume");

        context.Wait(this.MessageReceivedAsync);
    }
    public virtual async Task MessageReceivedAsync(IDialogContext context, IAwaitable<IMessageActivity>
result)
    {
        if ((await result).Text == "done")
        {
            await context.PostAsync("Great, back to the original conversation!");
            context.Done(String.Empty); // Finish this dialog.
        }
        else
        {
            await context.PostAsync("I'm still on the survey until you type \"done\"");
            context.Wait(MessageReceivedAsync); // Not done yet.
        }
    }
}

```

範例程式碼

如需示範如何使用適用於 .NET 的 Bot Framework SDK 傳送主動訊息的完整範例，請參閱 GitHub 中的[主動訊息範例](#)

例。在主動訊息範例中, [simpleSendMessage](#) 示範如何傳送臨機操作的主動訊息, 而 [startNewDialog](#) 示範如何傳送對話方塊式的主動訊息。

其他資源

- [設計和控制交談流程](#)
- [適用於 .NET 的 Bot Framework SDK 參考](#)
- [主動式訊息範例 \(GitHub\)](#)

適用於 .NET 的 Bot Framework SDK 中的對話

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

當您使用適用於 .NET 的 Bot Framework SDK 建立 Bot 時，您可以使用對話來建立交談模型及管理[交談流程](#)。每個對話都是抽象概念，可在實作 `IDialog` 的 C# 類別中封裝本身的狀態。對話可使用其他對話組成以便充分地重複使用，而且對話內容會維護[對話的堆疊](#)，這些對話在交談的任何時間點均處於作用中狀態。

由對話 (dialog) 組成的對話 (conversation) 可移植到不同電腦，這使得您的 Bot 實作能夠加以調整。當您在適用於 .NET 的 Bot Framework SDK 中使用對話時，交談狀態 (對話堆疊和堆疊中每個對話的狀態) 會自動儲存到您選擇的[狀態資料](#)儲存體。這可讓 Bot 的服務程式碼成為無狀態，如同 Web 應用程式不需要在 Web 伺服器記憶體中儲存工作階段狀態一樣。

回應 Bot 範例

請考慮使用這個回應 Bot 範例，此範例描述如何變更在[快速入門教學課程](#)中建立的 Bot，使其使用與使用者交換訊息的對話。

TIP

若要遵循此範例進行操作，請使用[快速入門教學課程](#)中的指示來建立 Bot，然後更新其 `MessagesController.cs` 檔案，如下所述。

MessagesController.cs

在適用於 .NET 的 Bot Framework SDK 中，`Builder` 程式庫可讓您實作對話。若要存取相關類別，請匯入 `Dialogs` 命名空間。

```
using Microsoft.Bot.Builder.Dialogs;
```

接下來，將這個 `EchoDialog` 類別新增至 `MessagesController.cs` 來代表交談。

```
[Serializable]
public class EchoDialog : IDialog<object>
{
    public async Task StartAsync(IDialogContext context)
    {
        context.Wait(MessageReceivedAsync);
    }

    public async Task MessageReceivedAsync(IDialogContext context, IAwaitable<IMessageActivity> argument)
    {
        var message = await argument;
        await context.PostAsync("You said: " + message.Text);
        context.Wait(MessageReceivedAsync);
    }
}
```

然後，透過呼叫 `Conversation.SendAsync` 方法，將 `EchoDialog` 類別連接到 `Post`。

```

public virtual async Task<HttpResponseMessage> Post([FromBody] Activity activity)
{
    // Check if activity is of type message
    if (activity != null && activity.GetActivityType() == ActivityTypes.Message)
    {
        await Conversation.SendAsync(activity, () => new EchoDialog());
    }
    else
    {
        HandleSystemMessage(activity);
    }
    return new HttpResponseMessage(System.Net.HttpStatusCode.Accepted);
}

```

實作詳細資料

`Post` 方法已標記為 `async`，因為 Bot 建立器使用 C# 功能來處理非同步通訊。它會傳回 `Task` 物件，代表負責傳送傳入訊息回覆的工作。如果發生例外狀況，方法所傳回的 `Task` 將會包含例外狀況資訊。

`Conversation.SendAsync` 方法是適用於 .NET 的 Bot Framework SDK 實作對話的關鍵。它會遵循[相依性反轉準則](#)，並執行下列步驟：

1. 具現化必要元件
2. 從 `IBotDataStore` 將交談狀態 (對話堆疊和堆疊中每個對話的狀態) 還原序列化
3. 繼續 Bot 暫停並等候訊息的交談流程
4. 傳送回覆
5. 將已更新的交談狀態序列化，並將其儲存回 `IBotDataStore`

第一次啟動交談時，對話不會包含狀態，因此 `Conversation.SendAsync` 會建構 `EchoDialog`，並呼叫其 `StartAsync` 方法。`StartAsync` 方法會呼叫 `IDialogContext.Wait`，繼續委派以指定在收到新訊息時應該呼叫的方法 (`MessageReceivedAsync`)。

`MessageReceivedAsync` 方法會等候訊息，張貼回應，並等候下一則訊息。每次呼叫 `IDialogContext.Wait` 時，Bot 會進入暫停狀態，因此可以在接收訊息的任何電腦上重新啟動。

使用上述程式碼範例建立的 Bot 會回覆使用者傳送的每則訊息，只需回應前置文字為 'You said:' 的使用者訊息即可。由於 Bot 是使用對話建立的，因此它能進化到支援更複雜的交談，而不需要明確地管理狀態。

以狀態回應 Bot 的範例

下一個範例會新增功能來追蹤對話狀態，以上一個範例為基礎進行建置。當 `EchoDialog` 類別如下列程式碼範例所示更新時，Bot 會回覆使用者傳送的每則訊息，只需回應前置數字 (`count`) 且後接文字 'You said:' 的使用者訊息即可。Bot 將在每次回覆時繼續遞增 `count`，直到使用者選擇重設計數為止。

MessagesController.cs

```

[Serializable]
public class EchoDialog : IDialog<object>
{
    protected int count = 1;

    public async Task StartAsync(IDialogContext context)
    {
        context.Wait(MessageReceivedAsync);
    }

    public virtual async Task MessageReceivedAsync(IDialogContext context, IAwaitable<IMessageActivity> argument)
    {
        var message = await argument;
        if (message.Text == "reset")
        {
            PromptDialog.Confirm(
                context,
                AfterResetAsync,
                "Are you sure you want to reset the count?",
                "Didn't get that!",
                promptStyle: PromptStyle.None);
        }
        else
        {
            await context.PostAsync($"{this.count++}: You said {message.Text}");
            context.Wait(MessageReceivedAsync);
        }
    }

    public async Task AfterResetAsync(IDialogContext context, IAwaitable<bool> argument)
    {
        var confirm = await argument;
        if (confirm)
        {
            this.count = 1;
            await context.PostAsync("Reset count.");
        }
        else
        {
            await context.PostAsync("Did not reset count.");
        }
        context.Wait(MessageReceivedAsync);
    }
}

```

實作詳細資料

如同第一個範例，在收到新訊息時會呼叫 `MessageReceivedAsync` 方法。不過，這次 `MessageReceivedAsync` 方法會在回應之前評估使用者的訊息。如果使用者的訊息是 "reset"，則內建的 `PromptDialog.Confirm` 提示會繁衍子對話，要求使用者確認計數重設。子對話有自己的專用狀態，不會干擾父對話的狀態。當使用者回應提示時，子對話的結果會傳遞至 `AfterResetAsync` 方法，這可將訊息傳送給使用者，以指出計數是否已重設，然後呼叫 `IDialogContext.Wait` 繼續回到下一則訊息的 `MessageReceivedAsync`。

對話內容

傳入每個對話方法的 `IDialogContext` 介面可存取對話所需的服務，用來儲存狀態並與通道進行通訊。

`IDialogContext` 介面包含三個介面：[Internals.IBotData](#)、[Internals.IBotToUser](#) 和 [Internals.IDialogStack](#)。

Internals.IBotData

`Internals.IBotData` 介面可以存取每位使用者、每個交談和連接器所維護之私用交談的狀態資料。每位使用者的狀態資料適用於儲存與特定交談無關的使用者資料，每個交談的資料適用於儲存有關交談的一般資料，而私用交談資料則適用於儲存與特定交談相關的使用者資料。

Internals.IBotToUser

`Internals.IBotToUser` 提供方法將訊息從 Bot 傳送給使用者。訊息可能會內嵌在 Web API 方法呼叫的回應中傳送，或者使用[連接器用戶端](#)直接傳送。透過對話內容傳送和接收訊息可確保 `Internals.IBotData` 狀態會通過連接器。

Internals.IDialogStack

`Internals.IDialogStack` 提供方法來管理[對話堆疊](#)。大部分的情況下，將會為您自動管理對話堆疊。不過，可能會有您想要明確地管理堆疊的情況。例如，您可能想要呼叫子對話並將它新增至對話堆疊的頂端、將目前的對話標示為完成（藉此將它從對話堆疊中移除，並將結果傳回堆疊中較早的對話）、暫停目前的對話直到來自使用者的訊息抵達，或甚至完全重設對話堆疊。

序列化

對話堆疊和所有使用中對話的狀態都會序列化為每位使用者、每個交談的 `IBotDataBag`。序列化的 Blob 會保存在 Bot 從[連接器](#)中傳送和接收的訊息中。若要序列化，`Dialog` 類別必須包含 `[Serializable]` 屬性。`Builder` 程式庫中的所有 `IDialog` 實作都會標示為可序列化。

`Chain` 方法為可用於 LINQ 查詢語法的對話提供 Fluent 介面。LINQ 查詢語法的已編譯形式通常會使用匿名方法。如果這些匿名方法不會參考本機變數的環境，則這些匿名方法沒有狀態，並可透過極簡方式序列化。不過，如果匿名方法會擷取環境中的任何區域變數，產生的結束物件（由編譯器產生）不會標示為可序列化。在此情況下，Bot 建立器會擲回 `ClosureCaptureException` 以找出問題。

若要使用反映來序列化未標示為可序列化的類別，建立器程式庫包含可供您用來註冊 [Autofac](#) 的反映型序列化代理。

```
var builder = new ContainerBuilder();
builder.RegisterType(new DialogModule());
builder.RegisterType(new ReflectionSurrogateModule());
```

對話鏈結

您可以使用 `IDialogStack.Call<R>` 和 `IDialogStack.Done<R>` 以明確管理使用中對話的堆疊，同時也可以使用下列流暢的 `Chain` 方法隱含地管理使用中對話的堆疊。

方法	類型	注意
<code>Chain.Select<T, R></code>	LINQ	支援 LINQ 查詢語法中的 "select" 和 "let"。
<code>Chain.SelectMany<T, C, R></code>	LINQ	支援 LINQ 查詢語法中的後續 "from"。
<code>Chain.Where</code>	LINQ	支援 LINQ 查詢語法中的 "where"。
<code>Chain.From</code>	鏈結	具現化對話的新執行個體。
<code>Chain.Return</code>	鏈結	將常數值傳回到鏈結。
<code>Chain.Do</code>	鏈結	允許鏈結內的副作用。
<code>Chain.ContinueWith<T, R></code>	鏈結	對話的簡單鏈結。
<code>Chain.Unwrap</code>	鏈結	解除包裝對話中巢狀處理的對話。

方法	類型	注意
Chain.DefaultIfException	鏈結	接受來自先前結果的例外狀況，並傳回 default(T)。
Chain.Loop	分支	循環整個對話鏈結。
Chain.Fold	分支	將來自對話列舉的結果摺疊成單一結果。
Chain.Switch<T, R>	分支	支援分支到不同的對話鏈結。
Chain.PostToUser	訊息	將訊息張貼給使用者。
Chain.WaitToBot	訊息	等候將訊息傳送至 Bot。
Chain.PostToChain	訊息	使用來自使用者的訊息開始鏈結。

範例

LINQ 查詢語法會使用 `Chain.Select<T, R>` 方法。

```
var query = from x in new PromptDialog.PromptString(Prompt, Prompt, attempts: 1)
            let w = new string(x.Reverse().ToArray())
            select w;
```

或者，`Chain.SelectMany<T, C, R>` 方法。

```
var query = from x in new PromptDialog.PromptString("p1", "p1", 1)
            from y in new PromptDialog.PromptString("p2", "p2", 1)
            select string.Join(" ", x, y);
```

`Chain.PostToUser<T>` 和 `Chain.WaitToBot<T>` 方法會將來自 Bot 的訊息張貼給使用者，反之亦然。

```
query = query.PostToUser();
```

`Chain.Switch<T, R>` 方法會分支交談對話流程。

```
var logic =
    toBot
    .Switch
    (
        new RegexCase<string>(new Regex("^hello"), (context, text) =>
        {
            return "world!";
        }),
        new Case<string, string>((txt) => txt == "world", (context, text) =>
        {
            return "!";
        }),
        new DefaultCase<string, string>((context, text) =>
        {
            return text;
        })
    );
});
```

如果 `Chain.Switch<T, R>` 傳回巢狀 `IDialog<IDialog<T>>`，則內部 `IDialog<T>` 可以使用 `Chain.Unwrap<T>` 來解除包裝。這可讓交談分支到已鏈結對話的不同路徑，可能的長度不等。此範例示範分支對話的更完整範例，該對話是使用隱含的堆疊管理以更流暢的鏈結樣式所撰寫而成。

```
var joke = Chain
    .PostToChain()
    .Select(m => m.Text)
    .Switch
    (
        Chain.Case
        (
            new Regex("^chicken"),
            (context, text) =>
                Chain
                    .Return("why did the chicken cross the road?")
                    .PostToUser()
                    .WaitToBot()
                    .Select(ignoreUser => "to get to the other side")
            ),
            Chain.Default<string, IDialog<string>>(
                (context, text) =>
                    Chain
                        .Return("why don't you like chicken jokes?")
            )
        )
    )
    .Unwrap()
    .PostToUser().
Loop();
```

後續步驟

對話可管理 Bot 與使用者之間的交談流程。對話會定義與使用者互動的方式。Bot 可以使用堆疊中組織的許多對話，來引導與使用者的交談。在下一節中，請查看當您建立和關閉堆疊中的對話時，對話堆疊如何成長和壓縮。

[使用對話管理交交談流程](#)

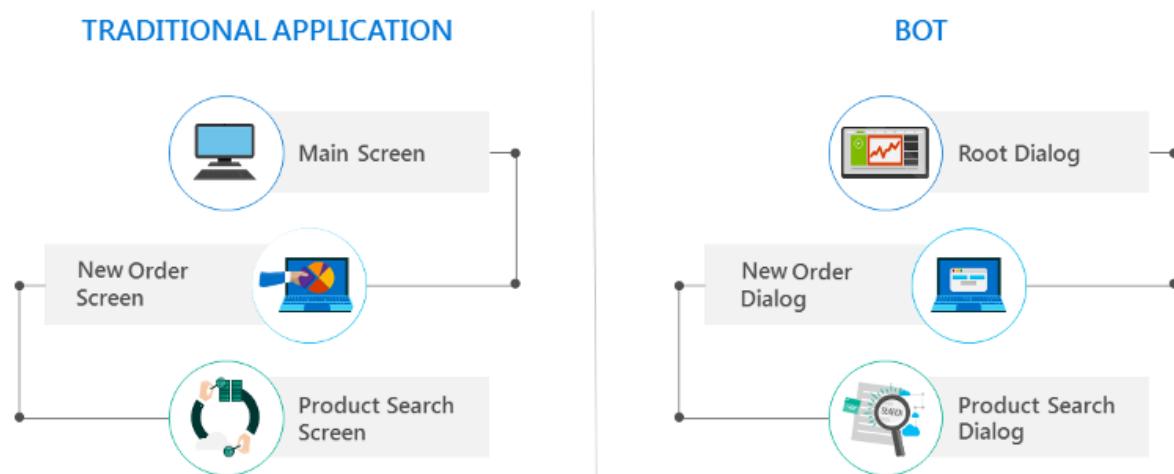
使用對話方塊管理交談流程

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

下圖顯示傳統應用程式的畫面流程與 Bot 對話流程的比較。



在傳統的應用程式中，所有項目都是以主畫面開始。主畫面會叫用新訂單畫面。新訂單畫面會保有持控制權，直到它關閉或叫用其他畫面為止。如果新訂單畫面關閉，使用者就會返回主畫面。

在 bot 中，所有項目都是以根對話方塊開頭。根對話會叫用新訂單對話。此時，新訂單對話會接管對話 (conversation) 並保有控制權，直到它關閉或叫用其他對話 (dialog)為止。如果新訂單對話 (dialog) 關閉，則會將對話 (conversation) 的控制權還給根對話 (dialog)。

本文說明如何使用**對話**和適用於 .NET 的 Bot Framework SDK 建立此交談流程的模型。

叫用根對話方塊

首先，Bot 控制器會叫用「根對話方塊」。下列範例示範如何將基本 HTTP POST 呼叫連線至控制器，然後叫用根對話方塊。

```
public class MessagesController : ApiController
{
    public async Task<HttpResponseMessage> Post([FromBody]Activity activity)
    {
        // Redirect to the root dialog.
        await Conversation.SendAsync(activity, () => new RootDialog());
        ...
    }
}
```

叫用「新訂單」對話方塊

然後，根對話會叫用「新訂單」對話方塊。

```

[Serializable]
public class RootDialog : IDialog<object>
{
    public async Task StartAsync(IDialogContext context)
    {
        // Root dialog initiates and waits for the next message from the user.
        // When a message arrives, call MessageReceivedAsync.
        context.Wait(this.MessageReceivedAsync);
    }

    public virtual async Task MessageReceivedAsync(IDialogContext context, IAwaitable<IMessageActivity> result)
    {
        var message = await result; // We've got a message!
        if (message.Text.ToLower().Contains("order"))
        {
            // User said 'order', so invoke the New Order Dialog and wait for it to finish.
            // Then, call ResumeAfterNewOrderDialog.
            await context.Forward(new NewOrderDialog(), this.ResumeAfterNewOrderDialog, message,
CancellationToken.None);
        }
        // User typed something else; for simplicity, ignore this input and wait for the next message.
        context.Wait(this.MessageReceivedAsync);
    }

    private async Task ResumeAfterNewOrderDialog(IDialogContext context, IAwaitable<string> result)
    {
        // Store the value that NewOrderDialog returned.
        // (At this point, new order dialog has finished and returned some value to use within the root
dialog.)
        var resultFromNewOrder = await result;

        await context.PostAsync($"New order dialog just told me this: {resultFromNewOrder}");

        // Again, wait for the next message from the user.
        context.Wait(this.MessageReceivedAsync);
    }
}

```

對話方塊生命週期

當叫用對話方塊時，它會控制交談流程。每則新訊息都必須由該對話方塊處理，直到關閉或重新導向至另一個對話方塊為止。

在 C# 中，您可以使用 `context.Wait()` 指定要在使用者下次傳送訊息時叫用的回撥。若要關閉對話方塊，並從堆疊移除（藉此將使用者傳回給堆疊中之前的對話方塊），請使用 `context.Done()`。您必須使用 `context.Wait()`、`context.Fail()`、`context.Done()` 或某些重新導向指示詞（如 `context.Forward()` 或 `context.Call()`）來結束每個對話方塊方法。未使用上述其中一種方式結束的對話方塊方法會導致發生錯誤（原因是架構不知道要在使用者下次傳送訊息時採取什麼動作）。

在對話方塊之間傳遞狀態

雖然您可以將狀態儲存在 Bot 狀態中，但也可透過多載對話方塊類別建構函式，在不同的對話方塊間傳遞資料。

```
[Serializable]
public class AgeDialog : IDialog<int>
{
    private string name;

    public AgeDialog(string name)
    {
        this.name = name;
    }
}
```

呼叫對話方塊程式碼，傳入使用者的名稱值。

```
private async Task NameDialogResumeAfter(IDialogContext context, IAwaitable<string> result)
{
    try
    {
        this.name = await result;
        context.Call(new AgeDialog(this.name), this.AgeDialogResumeAfter);
    }
    catch (TooManyAttemptsException)
    {
        await context.PostAsync("I'm sorry, I'm having issues understanding you. Let's try again.");
        await this.SendWelcomeMessageAsync(context);
    }
}
```

範例程式碼

如需示範如何透過在適用於 .NET 的 Bot Framework SDK 中使用對話來管理交談的完整範例，請參閱 GitHub 中的 [Basic Multi-Dialog sample](#) (基本多對話範例)。

其他資源

- [對話方塊](#)
- [設計和控制交談流程](#)
- [Basic Multi-Dialog sample \(GitHub\)](#) (基本多對話方塊範例 (GitHub))
- [適用於 .NET 的 Bot Framework SDK 參考](#)

使用可評分對話的全域訊息處理常式

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

當使用者嘗試在對話中使用「說明」、「取消」或「重新開始」等字，來存取 Bot 內的某些功能，而 Bot 預期不同的回應時。您可以將 Bot 設計為使用可評分對話方塊正常處理這類要求。

可評分對話方塊會監視所有傳入訊息，並判斷訊息是否可用某種方式採取動作。可評分的訊息會依每個可評分對話方塊，獲指派 [0 – 1] 之間的分數。決定分數最高的可評分對話方塊會新增至對話方塊堆疊的頂端，然後將回應傳遞給使用者。可評分對話方塊完成執行之後，對話會從停止的地方繼續進行。

可評分對話方塊能藉由讓您的使用者「中斷」在一般對話方塊中可見的一般對話流程，讓您建立更有彈性的對話。

建立可評分對話方塊

首先，定義新的對話方塊。下列程式碼會使用衍生自 `IDialog<object>` 介面的對話方塊。

```
public class SampleDialog : IDialog<object>
{
    public async Task StartAsync(IDialogContext context)
    {
        await context.PostAsync("This is a Sample Dialog which is Scorable. Reply with anything to return to
the prior prior dialog.");

        context.Wait(this.MessageReceived);
    }

    private async Task MessageReceived(IDialogContext context, IAwaitable<IMessageActivity> result)
    {
        var message = await result;

        if ((message.Text != null) && (message.Text.Trim().Length > 0))
        {
            context.Done<object>(null);
        }
        else
        {
            context.Fail(new Exception("Message was not a string or was an empty string."));
        }
    }
}
```

若要製作可評分對話方塊，請建立繼承自 `ScorableBase` 抽象類別的類別。下列程式碼會示範 `SampleScorable` 類別。

```

using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.Dialogs.Internals;
using Microsoft.Bot.Builder.Internals.Fibers;
using Microsoft.Bot.Builder.Scorables.Internals;

public class SampleScorable : ScorableBase<IActivity, string, double>
{
    private readonly IDialogTask task;

    public SampleScorable(IDialogTask task)
    {
        SetField.NotNull(out this.task, nameof(task), task);
    }
}

```

`ScorableBase` 抽象類別繼承自 `IScorable` 介面。您必須在類別中實作下列 `IScorable` 方法：

- `PrepareAsync` 是可評分執行個體中呼叫的第一個方法。它會接受傳入訊息活動、分析並設定對話方塊的狀態，接著會傳遞至 `IScorable` 介面的所有其他方法。

```

protected override async Task<string> PrepareAsync(IActivity item, CancellationToken token)
{
    // TODO: insert your code here
}

```

- `HasScore` 方法會檢查 `state` 屬性，來判斷可評分對話方塊是否應該提供訊息的分數。如果它傳回 `false`，則可評分對話方塊會忽略該訊息。

```

protected override bool HasScore(IActivity item, string state)
{
    // TODO: insert your code here
}

```

- 只有在 `HasScore` 傳回 `true` 時才會觸發 `GetScore`。您將佈建這個方法中的邏輯，以判斷分數介於 0 - 1 之間的訊息。

```

protected override double GetScore(IActivity item, string state)
{
    // TODO: insert your code here
}

```

- 在 `PostAsync` 方法中，定義要針對可評分類別執行的核心動作。所有的可評分對話方塊都會監視傳入訊息，並根據可評分對話方塊的 `GetScore` 方法，將分數指派給有效的訊息。接著，決定最高分數（介於 0 - 1.0 之間）的可評分類別將會觸發可評分對話方塊的 `PostAsync` 方法。

```

protected override Task PostAsync(IActivity item, string state, CancellationToken token)
{
    //TODO: insert your code here
}

```

- 在評分流程完成之後，會呼叫 `DoneAsync`。使用此方法來處置任何已設定範圍的資源。

```
protected override Task DoneAsync(IActivity item, string state, CancellationToken token)
{
    //TODO: insert your code here
}
```

建立模組來註冊 IScorable 服務

接下來，定義會將 `SampleScorable` 類別註冊為元件的 `Module`。這會佈建 `IScorable` 服務。

```
public class GlobalMessageHandlersBotModule : Module
{
    protected override void Load(ContainerBuilder builder)
    {
        base.Load(builder);

        builder
            .Register(c => new SampleScorable(c.Resolve<IDialogTask>()))
            .As<IScorable<IActivity, double>>()
            .InstancePerLifetimeScope();
    }
}
```

註冊模組

此程序的最後一個步驟是要將 `SampleScorable` 套用至 Bot 的對話容器。這會註冊 Bot Framework 訊息處理管線內的可評分服務。下列程式碼隨即顯示，以在 **Global.asax.cs** 中更新 Bot 應用程式初始化內的

`Conversation.Container`：

```
public class WebApiApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        this.RegisterBotModules();
        GlobalConfiguration.Configure(WebApiConfig.Register);
    }

    private void RegisterBotModules()
    {
        var builder = new ContainerBuilder();
        builder.RegisterModule(new ReflectionSurrogateModule());

        //Register the module within the Conversation container
        builder.RegisterType<GlobalMessageHandlersBotModule>();

        builder.Update(Conversation.Container);
    }
}
```

其他資源

- [全域訊息處理常式範例](#) (英文)
- [簡單可評分 Bot 範例](#) (英文)
- [對話方塊概觀](#)
- [AutoFac](#) (英文)

FormFlow 的基本功能

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

交談方塊功能強大且彈性，處理訂購三明治等引導式交談時可能會很費工。在交談中的每個時間點，接下來有可能會發生許多不同的情況。例如，您可能需要釐清問題、提供協助、返回或顯示進度。使用適用於 .NET 的 Bot Framework SDK 中的 **FormFlow**，即可大幅簡化管理這類引導式交談的程序。

FormFlow 可根據您指定的指導方針，自動產生管理引導式交談時必須使用的交談方塊。雖然使用 FormFlow 會讓您無法建立並管理專屬交談方塊，因而犧牲部分彈性；但使用 FormFlow 設計引導式交談可大幅減少您自己開發 Bot 的時間。此外，您也可使用 FormFlow 產生的對話方塊及其他類型對話方塊組合，建構自己的 Bot。例如，FormFlow 對話方塊可引導使用者完成表單填寫的整個程序，而 [LuisDialog](#) 可評估使用者輸入內容以判斷其意圖。

本文說明如何建立使用 FormFlow 基本功能的 Bot，以向使用者收集資訊。

表單和欄位

若要使用 FormFlow 建立 Bot，您必須指定 Bot 要向使用者收集的資訊。例如，如果 Bot 的目標是取得使用者的三明治訂單，則您定義的表單中必須包含可讓 Bot 填寫訂單之資料的欄位。您可建立 C# 類別，其中包含一或多個公用屬性，代表 Bot 將向使用者收集的資料。每個屬性都必須屬於以下資料類型之一：

- 整數 (sbyte、byte、short、ushort、int、uint、long、ulong)
- 浮點數 (float、double)
- 字串
- Datetime
- 列舉
- 列舉的清單

任何資料類型皆可為 Null，您可在沒有值之欄位的模型中使用。如果表單欄位是以不得為 Null 的列舉屬性為基礎，則列舉中的值 0 即代表 null (即指出該欄位沒有值)，而您應從 1 開始列舉值。FormFlow 會忽略所有其他屬性類型及方法。

針對複雜物件，您必須為最上層 C# 類別建立表單，再為複雜物件建立另一份表單。您可使用一般對話方塊語意，將表單合併在一起。您亦可實作 [Advanced.IField](#) 或使用 [Advanced.Field](#)，並在其中填入字典，藉此直接定義表單。

NOTE

您可使用 C# 類別或 JSON 結構描述定義表單。本文說明如何使用 C# 類別定義表單。如需有關使用 JSON 結構描述的詳細資訊，請參閱[使用 JSON 結構描述定義表單](#)。

範例三明治 Bot

請參考此三明治 Bot 範例，其旨在取得使用者的三明治訂單。

建立表單

`SandwichOrder` 類別可定義表單，而列舉則定義建置三明治的選項。此類別亦包含靜態的 `BuildForm` 方法，其採用 `FormBuilder` 建立表單，並定義簡單的歡迎訊息。

若要使用 FormFlow，您必須先匯入 `Microsoft.Bot.Builder.FormFlow` 命名空間。

```
using Microsoft.Bot.Builder.FormFlow;
using System;
using System.Collections.Generic;

// The SandwichOrder class represents the form that you want to complete
// using information that is collected from the user.
// It must be serializable so the bot can be stateless.
// The order of fields defines the default sequence in which the user is asked questions.

// The enumerations define the valid options for each field in SandwichOrder, and the order
// of the values represents the sequence in which they are presented to the user in a conversation.

namespace Microsoft.Bot.Sample.SimpleSandwichBot
{
    public enum SandwichOptions
    {
        BLT, BlackForestHam, BuffaloChicken, ChickenAndBaconRanchMelt, ColdCutCombo, MeatballMarinara,
        OvenRoastedChicken, RoastBeef, RotisserieStyleChicken, SpicyItalian, SteakAndCheese,
        SweetOnionTeriyaki, Tuna,
        TurkeyBreast, Veggie
    };
    public enum LengthOptions { SixInch, FootLong };
    public enum BreadOptions { NineGrainWheat, NineGrainHoneyOat, Italian, ItalianHerbsAndCheese, Flatbread
    };
    public enum CheeseOptions { American, MontereyCheddar, Pepperjack };
    public enum ToppingOptions
    {
        Avocado, BananaPeppers, Cucumbers, GreenBellPeppers, Jalapenos,
        Lettuce, Olives, Pickles, RedOnion, Spinach, Tomatoes
    };
    public enum SauceOptions
    {
        ChipotleSouthwest, HoneyMustard, LightMayonnaise, RegularMayonnaise,
        Mustard, Oil, Pepper, Ranch, SweetOnion, Vinegar
    };

    [Serializable]
    public class SandwichOrder
    {
        public SandwichOptions? Sandwich;
        public LengthOptions? Length;
        public BreadOptions? Bread;
        public CheeseOptions? Cheese;
        public List<ToppingOptions> Toppings;
        public List<SauceOptions> Sauce;

        public static IForm<SandwichOrder> BuildForm()
        {
            return new FormBuilder<SandwichOrder>()
                .Message("Welcome to the simple sandwich order bot!")
                .Build();
        }
    };
}
```

將表單連接至架構

若要將表單連接至架構，您必須將其新增至控制器。在此範例中，`Conversation.SendAsync` 方法會呼叫靜態的 `MakeRootDialog` 方法，而後者則會呼叫 `FormDialog.FromForm` 方法以建立 `SandwichOrder` 表單。

```

internal static IDialog<SandwichOrder> MakeRootDialog()
{
    return Chain.From(() => FormDialog.FromForm(SandwichOrder.BuildForm));
}

[ResponseType(typeof(void))]
public virtual async Task<HttpResponseMessage> Post([FromBody] Activity activity)
{
    if (activity != null)
    {
        switch (activity.GetActivityType())
        {
            case ActivityTypes.Message:
                await Conversation.SendAsync(activity, MakeRootDialog);
                break;

            case ActivityTypes.ConversationUpdate:
            case ActivityTypes.ContactRelationUpdate:
            case ActivityTypes.Typing:
            case ActivityTypes.DeleteUserData:
            default:
                Trace.TraceError($"Unknown activity type ignored: {activity.GetActivityType()}");
                break;
        }
    }
    ...
}

```

了解其運作方式

只要以 C# 類別定義表單，並將其連接至架構，即可啟用 FormFlow 自動管理 Bot 和使用者之間的交談。下列範例互動示範了 FormFlow 基本功能建立的 Bot 功能。在每個互動中，> 符號皆指出使用者輸入回應的時間點。

顯示第一個提示

此表單會填入 `SandwichOrder.Sandwich` 屬性。表單將自動產生提示「請選取三明治」，而提示中的「三明治」一詞是從屬性名稱 `Sandwich` 衍生而來。`SandwichOptions` 列舉可定義向使用者顯示的選擇，每個列舉值都可自動根據大小寫和底線文字細分變更。

```

Please select a sandwich
1. BLT
2. Black Forest Ham
3. Buffalo Chicken
4. Chicken And Bacon Ranch Melt
5. Cold Cut Combo
6. Meatball Marinara
7. Oven Roasted Chicken
8. Roast Beef
9. Rotisserie Style Chicken
10. Spicy Italian
11. Steak And Cheese
12. Sweet Onion Teriyaki
13. Tuna
14. Turkey Breast
15. Veggie
>

```

提供指引

使用者可在交談中任何時間點輸入「協助」，以取得填寫表單的指引。例如，如果使用者在三明治提示中輸入「協助」，則 Bot 將在回應中提供此指引。

```
> help
* You are filling in the sandwich field. Possible responses:
* You can enter a number 1-15 or words from the descriptions. (BLT, Black Forest Ham, Buffalo Chicken,
Chicken And Bacon Ranch Melt, Cold Cut Combo, Meatball Marinara, Oven Roasted Chicken, Roast Beef,
Rotisserie Style Chicken, Spicy Italian, Steak And Cheese, Sweet Onion Teriyaki, Tuna, Turkey Breast, and
Veggie)
* Back: Go back to the previous question.
* Help: Show the kinds of responses you can enter.
* Quit: Quit the form without completing it.
* Reset: Start over filling in the form. (With defaults from your previous entries.)
* Status: Show your progress in filling in the form so far.
* You can switch to another field by entering its name. (Sandwich, Length, Bread, Cheese, Toppings, and
Sauce).
```

繼續下一個提示

如果使用者在回應第一個三明治提示時輸入「2」, Bot 可接著顯示下個屬性提示, 其由此表單定義:

```
SandwichOrder.Length
```

```
Please select a sandwich
1. BLT
2. Black Forest Ham
3. Buffalo Chicken
4. Chicken And Bacon Ranch Melt
5. Cold Cut Combo
6. Meatball Marinara
7. Oven Roasted Chicken
8. Roast Beef
9. Rotisserie Style Chicken
10. Spicy Italian
11. Steak And Cheese
12. Sweet Onion Teriyaki
13. Tuna
14. Turkey Breast
15. Veggie
> 2
Please select a length (1. Six Inch, 2. Foot Long)
>
```

返回上一個提示

如果使用者在交談中的此時間點輸入「返回」, 則 Bot 將返回上一個提示。提示會顯示使用者目前的選項 (「黑森林火腿」); 使用者可輸入其他數目以修改選項, 或輸入「C」確認選取項。

```
> back
Please select a sandwich(current choice: Black Forest Ham)
1. BLT
2. Black Forest Ham
3. Buffalo Chicken
4. Chicken And Bacon Ranch Melt
5. Cold Cut Combo
6. Meatball Marinara
7. Oven Roasted Chicken
8. Roast Beef
9. Rotisserie Style Chicken
10. Spicy Italian
11. Steak And Cheese
12. Sweet Onion Teriyaki
13. Tuna
14. Turkey Breast
15. Veggie
> c
Please select a length (1. Six Inch, 2. Foot Long)
>
```

釐清使用者輸入值

如果使用者以文字回應 (而非用數字) 並指出選項，則 Bot 可進一步釐清文字，自動詢問使用者其輸入的內容是否包含其他選項。

```
Please select a bread
1. Nine Grain Wheat
2. Nine Grain Honey Oat
3. Italian
4. Italian Herbs And Cheese
5. Flatbread
> nine grain
By "nine grain" bread did you mean (1. Nine Grain Honey Oat, 2. Nine Grain Wheat)
> 1
```

如果使用者輸入值未直接符合任何有效選項，Bot 可自動提示使用者確認。

```
Please select a cheese (1. American, 2. Monterey Cheddar, 3. Pepperjack)
> american
"american" is not a cheese option.
> american smoked
For cheese I understood American. "smoked" is not an option.
```

如果使用者輸入值指出某屬性的多個選擇，而 Bot 無法理解其中任何指定選項，此時就會自動提示使用者以進一步釐清。

```
Please select one or more toppings
1. Banana Peppers
2. Cucumbers
3. Green Bell Peppers
4. Jalapenos
5. Lettuce
6. Olives
7. Pickles
8. Red Onion
9. Spinach
10. Tomatoes
> peppers, lettuce and tomato
By "peppers" toppings did you mean (1. Green Bell Peppers, 2. Banana Peppers)
> 1
```

顯示目前狀態

如果使用者在訂單中的任何時間點輸入「狀態」，則 Bot 的回應將指出已指定和待指定的值。

```
Please select one or more sauce
1. Honey Mustard
2. Light Mayonnaise
3. Regular Mayonnaise
4. Mustard
5. Oil
6. Pepper
7. Ranch
8. Sweet Onion
9. Vinegar
> status
* Sandwich: Black Forest Ham
* Length: Six Inch
* Bread: Nine Grain Honey Oat
* Cheese: American
* Toppings: Lettuce, Tomatoes, and Green Bell Peppers
* Sauce: Unspecified
```

確認選取項目

使用者完成表單後，Bot 便會要求使用者確認其選取項目。

```
Please select one or more sauce
1. Honey Mustard
2. Light Mayonnaise
3. Regular Mayonnaise
4. Mustard
5. Oil
6. Pepper
7. Ranch
8. Sweet Onion
9. Vinegar
> 1
Is this your selection?
* Sandwich: Black Forest Ham
* Length: Six Inch
* Bread: Nine Grain Honey Oat
* Cheese: American
* Toppings: Lettuce, Tomatoes, and Green Bell Peppers
* Sauce: Honey Mustard
>
```

如果使用者在回應中輸入「否」，則 Bot 便會讓使用者修改先前的選取項目。如果使用者在回應中輸入「是」，則系統將視為完成表單，控制項隨即傳回至呼叫對話方塊。

```
Is this your selection?
* Sandwich: Black Forest Ham
* Length: Six Inch
* Bread: Nine Grain Honey Oat
* Cheese: American
* Toppings: Lettuce, Tomatoes, and Green Bell Peppers
* Sauce: Honey Mustard
> no
What do you want to change?
1. Sandwich(Black Forest Ham)
2. Length(Six Inch)
3. Bread(Nine Grain Honey Oat)
4. Cheese(American)
5. Toppings(Lettuce, Tomatoes, and Green Bell Peppers)
6. Sauce(Honey Mustard)
> 2
Please select a length (current choice: Six Inch) (1. Six Inch, 2. Foot Long)
> 2
Is this your selection?
* Sandwich: Black Forest Ham
* Length: Foot Long
* Bread: Nine Grain Honey Oat
* Cheese: American
* Toppings: Lettuce, Tomatoes, and Green Bell Peppers
* Sauce: Honey Mustard
> y
```

處理結束和例外狀況

如果使用者在表單中輸入「結束」，或交談中任何時間點發生例外狀況，則 Bot 必須知道事件發生於哪個步驟、事件發生時的表單狀態，以及事件發生前使用者完成了表單的哪些步驟。表單會透過 `FormCanceledException<T>` 類別傳回此資訊。

此程式碼範例會顯示如何快取例外狀況，並根據發生的事件顯示訊息。

```
internal static IDialog<SandwichOrder> MakeRootDialog()
{
    return Chain.From(() => FormDialog.FromForm(SandwichOrder.BuildLocalizedForm))
        .Do(async (context, order) =>
    {
        try
        {
            var completed = await order;
            // Actually process the sandwich order...
            await context.PostAsync("Processed your order!");
        }
        catch (FormCanceledException<SandwichOrder> e)
        {
            string reply;
            if (e.InnerException == null)
            {
                reply = $"You quit on {e.Last} -- maybe you can finish next time!";
            }
            else
            {
                reply = "Sorry, I've had a short circuit. Please try again.";
            }
            await context.PostAsync(reply);
        }
    });
}
```

總結

本文說明了如何使用 FormFlow 的基本功能建立可執行以下工作的 Bot:

- 自動產生並管理交談
- 提供清楚的指引和協助
- 了解數字和文字項目
- 向使用者提供意見反應，說明目前已理解和待釐清的資訊
- 必要時詢問釐清問題
- 允許使用者在步驟之間往返修改

雖然在某些情況下，基本的 FormFlow 功能已足夠，但您仍應考量在 Bot 中整合其他進階功能的潛在好處。如需詳細資訊，請參閱 [FormFlow 的進階功能](#) 和 [使用 FormBuilder 自訂表單](#)。

範例程式碼

如需示範如何使用適用於 .NET 的 Bot Framework SDK 實作 FormFlow 的完整範例，請參閱 GitHub 中的[多對話 Bot 範例](#) (英文) 和 [Contoso Flowers Bot 範例](#) (英文)。

後續步驟

FormFlow 簡化對話方塊的開發作業。FormFlow 的進階功能可讓您自訂 FormFlow 物件的運作方式。

[FormFlow 的進階功能](#)

其他資源

- [使用 FormBuilder 來自訂表單](#)
- [將表單內容當地語系化](#)
- [使用 JSON 結構描述來定義表單](#)
- [使用模式語言來自訂使用者體驗](#)

FormFlow 的進階功能

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

[FormFlow 的基本功能](#)描述基本的 FormFlow 實作，可提供相當通用的使用者體驗。若要使用 FormFlow 來提供更客製化的使用者體驗，您可以指定初始表單狀態；新增商務邏輯，以管理欄位之間的相互依存性並處理使用者輸入；以及使用屬性來自訂提示、覆寫範本、指定選擇性欄位、比對使用者輸入，並驗證使用者輸入。

指定初始表單狀態和實體

當您啟動 `FormDialog` 時，可以選擇性地傳遞狀態的執行個體。如果您將狀態的執行個體傳入，則 FormFlow 依預設會略過任何欄位中已包含值的步驟；系統將不會提示使用者輸入這些欄位。若要強制表單提示使用者輸入所有欄位（即使欄位已包含初始狀態中的值），當您啟動 `FormDialog` 時，請傳入 `FormOptions.PromptFieldsWithValues`。如果欄位包含一個初始值，則提示會使用該值作為預設值。

您也可以傳入 [LUIS](#) 實體以繫結至狀態。如果 `EntityRecommendation.Type` 是前往 C# 類別中欄位的路徑，則 `EntityRecommendation.Entity` 會通過要繫結至您欄位的辨識器。FormFlow 會略過任何繫結至實體欄位的步驟；系統將不會提示使用者輸入這些欄位。

新增商務邏輯

若要處理表單欄位之間的相互依存性，或在取得或設定欄位值的程序期間套用特定邏輯，您可以指定驗證函式內的商務邏輯。驗證函式可讓您管理狀態，並傳回 `ValidateResult` 物件，當中可能包含：

- 意見反應字串，會描述值無效的原因
- 已轉換的值
- 用於釐清值的一組選擇

此程式碼範例會顯示 `Toppings` 欄位的驗證函式。如果欄位的輸入包含 `ToppingOptions.Everything` 列舉值，則函式可確保 `Toppings` 欄位值包含配料的完整清單。

```

public static IForm<SandwichOrder> BuildForm()
{
    ...
    return new FormBuilder<SandwichOrder>()
        .Message("Welcome to the sandwich order bot!")
        .Field(nameof(Sandwich))
        .Field(nameof(Length))
        .Field(nameof(Bread))
        .Field(nameof(Cheese))
        .Field(nameof(Toppings),
            validate: async (state, value) =>
        {
            var values = ((List<object>)value).OfType<ToppingOptions>();
            var result = new ValidateResult { IsValid = true, Value = values };
            if (values != null && values.Contains(ToppingOptions.Everything))
            {
                result.Value = (from ToppingOptions topping in Enum.GetValues(typeof(ToppingOptions))
                                where topping != ToppingOptions.Everything && !values.Contains(topping)
                                select topping).ToList();
            }
            return result;
        })
        .Message("For sandwich toppings you have selected {Toppings}.")
        ...
        .Build();
}

```

除了驗證函式之外，您還可以新增 [Term](#) 屬性，以符合 "everything" 或 "not" 等使用者運算式。

```

public enum ToppingOptions
{
    // This starts at 1 because 0 is the "no value" value
    [Terms("except", "but", "not", "no", "all", "everything")]
    Everything = 1,
    ...
}

```

此程式碼片段會使用如上所示的驗證函式，來顯示當使用者要求 "everything but Jalapenos" 時，Bot 和使用者之間的互動。

```

Please select one or more toppings (current choice: No Preference)
1. Everything
2. Avocado
3. Banana Peppers
4. Cucumbers
5. Green Bell Peppers
6. Jalapenos
7. Lettuce
8. Olives
9. Pickles
10. Red Onion
11. Spinach
12. Tomatoes
> everything but jalapenos
For sandwich toppings you have selected Avocado, Banana Peppers, Cucumbers, Green Bell Peppers, Lettuce, Olives, Pickles, Red Onion, Spinach, and Tomatoes.

```

FormFlow 屬性

您可以將這些 C# 屬性新增至您的類別，來自訂 FormFlow 對話方塊的行為。

屬性	目的
描述	變更欄位或值在範本或卡片中的顯示方式
數值	限制數值欄位的接受值
選用	將欄位標示為選用
模式	定義規則運算式來驗證字串欄位
提示	定義欄位的提示
範本	定義要用來在提示中產生提示或值的範本
字詞	定義符合欄位或值的輸入字詞

使用 Prompt 屬性來自訂提示

系統會針對表單中的每個欄位自動產生預設提示，但您可以使用 `Prompt` 屬性，指定任何欄位的自訂提示。例如，如果 `SandwichOrder.Sandwich` 欄位的預設提示是「請選取三明治」您可以新增 `Prompt` 屬性來指定該欄位的自訂提示。

```
[Prompt("What kind of {&} would you like? {|||}")]
public SandwichOptions? Sandwich;
```

這個範例會使用 **模式語言**，在執行階段以動態方式將表單資料填入提示：`{&}` 會取代為欄位的描述，而 `{|||}` 會取代為列舉中的選擇清單。

NOTE

根據預設，欄位的描述會產生自欄位的名稱。若要指定欄位的自訂描述，請新增 `Describe` 屬性。

此程式碼片段會顯示上述範例中所指定的自訂提示。

```
What kind of sandwich would you like?
1. BLT
2. Black Forest Ham
3. Buffalo Chicken
4. Chicken And Bacon Ranch Melt
5. Cold Cut Combo
6. Meatball Marinara
7. Oven Roasted Chicken
8. Roast Beef
9. Rotisserie Style Chicken
10. Spicy Italian
11. Steak And Cheese
12. Sweet Onion Teriyaki
13. Tuna
14. Turkey Breast
15. Veggie
>
```

`Prompt` 屬性也可以指定影響表單顯示提示方式的參數。例如，`ChoiceFormat` 參數會決定表單轉譯選擇清單的方式。

```
[Prompt("What kind of {&} would you like? {||}", ChoiceFormat="{1}")]
public SandwichOptions? Sandwich;
```

在此範例中，`ChoiceFormat` 參數的值表示選擇應該顯示為項目符號清單（而不是已編號的清單）。

```
What kind of sandwich would you like?
- BLT
- Black Forest Ham
- Buffalo Chicken
- Chicken And Bacon Ranch Melt
- Cold Cut Combo
- Meatball Marinara
- Oven Roasted Chicken
- Roast Beef
- Rotisserie Style Chicken
- Spicy Italian
- Steak And Cheese
- Sweet Onion Teriyaki
- Tuna
- Turkey Breast
- Veggie
>
```

使用 `Template` 屬性自訂提示

`Prompt` 屬性可讓您自訂單一欄位的提示，而 `Template` 屬性可讓您取代 FormFlow 用來自動產生提示的預設範本。此程式碼範例會使用 `Template` 屬性，重新定義表單處理所有列舉欄位的方式。此屬性會指出使用者只能選取其中一個項目、使用[模式語言](#)來設定提示的文字，並指定表單每一行應該只顯示一個項目。

```
[Template(TemplateUsage.EnumSelectOne, "What kind of {&} would you like on your sandwich? {||}", ChoiceStyle = ChoiceStyleOptions.PerLine)]
public class SandwichOrder
```

此程式碼片段會顯示 `Bread` 欄位和 `Cheese` 欄位的結果提示。

```
What kind of bread would you like on your sandwich?
1. Nine Grain Wheat
2. Nine Grain Honey Oat
3. Italian
4. Italian Herbs And Cheese
5. Flatbread
>
```

```
What kind of cheese would you like on your sandwich?
1. American
2. Monterey Cheddar
3. Pepperjack
>
```

如果您使用 `Template` 屬性來取代 FormFlow 用來產生提示的預設範本，可能需要將一些變化插入表單產生的提示和訊息。若要這樣做，您可以使用[模式語言](#)來定義多個文字字串，每當表單必須顯示提示或訊息時，就會隨機從可用的選項中進行選擇。

此程式碼範例會重新定義 `TemplateUsage.NotUnderstood` 範本，以指定兩個不同變化的訊息。當 Bot 因不了解使用者的輸入而需要通訊時，會從兩個文字字串中隨機選取一個來判斷訊息內容。

```
[Template(TemplateUsage.NotUnderstood, "I do not understand \"{0}\", \"Try again, I don't get \"{0}\")]  
[Template(TemplateUsage.EnumSelectOne, "What kind of {&} would you like on your sandwich? {||}")]  
public class SandwichOrder
```

此程式碼片段會顯示 Bot 與使用者之間產生互動的範例。

```
What size of sandwich do you want? (1. Six Inch, 2. Foot Long)  
> two feet  
I do not understand "two feet".  
> two feet  
Try again, I don't get "two feet"  
>
```

使用 Optional 屬性將欄位指定為選擇性

若要將欄位指定為選擇性，請使用 `optional` 屬性。這個程式碼範例會指定 `Cheese` 欄位是選擇性的。

```
[Optional]  
public CheeseOptions? Cheese;
```

如果欄位是選擇性的，且未指定任何值，則目前的選擇會顯示為「無喜好設定」。

```
What kind of cheese would you like on your sandwich? (current choice: No Preference)  
1. American  
2. Monterey Cheddar  
3. Pepperjack  
>
```

如果欄位是選擇性的，且使用者已指定值，則「無喜好設定」會顯示為清單中的最後一個選擇。

```
What kind of cheese would you like on your sandwich? (current choice: American)  
1. American  
2. Monterey Cheddar  
3. Pepperjack  
4. No Preference  
>
```

使用 Terms 屬性來比對使用者輸入

當使用者將訊息傳送至使用 FormFlow 建置的 Bot 時，Bot 就會比對輸入與字詞清單，以嘗試識別使用者輸入的意義。根據預設，字詞清單是透過將這些步驟套用至欄位或值所產生的：

- 在出現大小寫變更和底線 () 時中斷。
- 產生每個 `n-gram` 直到長度上限。
- 將 "s?" 新增到每個字的結尾 (以支援複數)。

例如，"AngusBeefAndGarlicPizza" 值會產生下列字詞：

- 'angus?'
- 'beefs?'
- 'garlics?'
- 'pizzas?'
- 'angus? beefs?'

- 'garlics? pizzas?'
- 'angus beef and garlic pizza'

若要覆寫此預設行為，並定義字詞清單，用來比對使用者輸入與欄位或欄位中的值，請使用 `Terms` 屬性。例如，您可以使用 `Terms` 屬性（與規則運算式）來說明使用者很可能將 "rotisserie" 這個單字拼錯的事實。

```
[Terms(@"rotis\w* style chicken", MaxPhrase = 3)]
RotisserieStyleChicken, SpicyItalian, SteakAndCheese, SweetOnionTeriyaki, Tuna,...
```

您可以藉由使用 `Terms` 屬性，來增加使用者輸入符合其中一個有效選項的可能性。在此範例中的 `Terms.MaxPhrase` 參數會使 `Language.GenerateTerms` 產生字詞的其他變化。

當使用者拼錯 "Rotisserie" 時，此程式碼片段會顯示 Bot 與使用者之間產生的互動。

```
What kind of sandwich would you like?
1. BLT
2. Black Forest Ham
3. Buffalo Chicken
4. Chicken And Bacon Ranch Melt
5. Cold Cut Combo
6. Meatball Marinara
7. Oven Roasted Chicken
8. Roast Beef
9. Rotisserie Style Chicken
10. Spicy Italian
11. Steak And Cheese
12. Sweet Onion Teriyaki
13. Tuna
14. Turkey Breast
15. Veggie
> rotissary checkin
For sandwich I understood Rotisserie Style Chicken. "checkin" is not an option.
```

使用 Numeric 屬性或 Pattern 屬性來驗證使用者輸入

若要限制數值欄位的允許值範圍，請使用 `Numeric` 屬性。此程式碼範例會使用 `Numeric` 屬性，來指定 `Rating` 欄位的輸入必須是介於 1 到 5 之間的數字。

```
[Numeric(1, 5)]
public double? Rating;
```

若要指定特定欄位值所需的格式，請使用 `Pattern` 屬性。此程式碼範例會使用 `Pattern` 屬性，來指定 `PhoneNumber` 欄位值所需的格式。

```
[Pattern(@"(<Undefined control sequence>\d)?\s*\d{3}(-|\s*)\d{4}")]
public string PhoneNumber;
```

總結

本文已說明如何透過指定初始表單狀態；新增商務邏輯，以管理欄位之間的相互依存性並處理使用者輸入；以及使用屬性來自訂提示、覆寫範本、指定選擇性欄位、比對使用者輸入，並驗證使用者輸入，使用 FormFlow 來提供客製化的使用者體驗。如需使用 FormFlow 來自訂使用者體驗的其他方式相關資訊，請參閱[使用 FormBuilder 來自訂表單](#)。

範例程式碼

如需示範如何使用適用於 .NET 的 Bot Framework SDK 實作 FormFlow 的完整範例，請參閱 GitHub 中的[多對話 Bot 範例](#) (英文) 和 [Contoso Flowers Bot 範例](#) (英文)。

其他資源

- [FormFlow 的基本功能](#)
- [使用 FormBuilder 來自訂表單](#)
- [將表單內容當地語系化](#)
- [使用 JSON 結構描述來定義表單](#)
- [使用模式語言來自訂使用者體驗](#)
- [適用於 .NET 的 Bot Framework SDK 參考](#)

使用 FormBuilder 來自訂表單

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

[FormFlow 的基本功能](#)描述基本的 FormFlow 實作，可提供相當通用的使用者體驗，而 [FormFlow 的進階功能](#)則描述如何使用商務邏輯和屬性來自訂使用者體驗。本文說明如何藉由指定表單執行步驟的序列，並以動態方式定義欄位值、確認和訊息，使用 [FormBuilder](#) 進一步自訂使用者體驗。

以動態方式定義欄位值、確認和訊息

您可以使用 [FormBuilder](#)，以動態方式定義欄位值、確認和訊息。

以動態方式定義欄位值

三明治 Bot 的設計宗旨是將免費飲料或餅乾加入指定長條三明治的任何訂單中，會使用 `Sandwich.Specials` 欄位來儲存可用項目的資料。在此案例中，`Sandwich.Specials` 欄位的值必須根據訂單是否包含長條三明治，以動態方式為每筆訂單進行設定。

`Specials` 欄位已指定為選用，且代表無偏好選擇的文字已指定為「無」。

```
[Optional]
[Template(TemplateUsage.NoPreference, "None")]
public string Specials;
```

此程式碼範例會示範如何以動態方式設定 `Specials` 欄位的值。

```
.Field(new FieldReflector<SandwichOrder>(nameof(Specials))
    .SetType(null)
    .SetActive((state) => state.Length == LengthOptions.FootLong)
    .SetDefine(async (state, field) =>
{
    field
        .AddDescription("cookie", "Free cookie")
        .AddTerms("cookie", "cookie", "free cookie")
        .AddDescription("drink", "Free large drink")
        .AddTerms("drink", "drink", "free drink");
    return true;
}))
```

在此範例中，[Advanced.Field.SetType](#) 方法會指定欄位類型 (`null` 表示列舉欄位)。[Advanced.Field.SetActive](#) 方法可指定只有在三明治長度為 `Length.FootLong` 時，才會啟用欄位。最後，[Advanced.Field.SetDefine](#) 方法會指定可定義欄位的非同步委派。委派已傳遞至目前的狀態物件和以動態方式定義的 [Advanced.Field](#)。此委派會使用欄位的 Fluent 方法，以動態方式定義值。在此範例中，值為字串，且 `AddDescription` 和 `AddTerms` 方法會指定每個值的說明和字詞。

NOTE

若要以動態方式定義欄位值，您可以自己實作 [Advanced.IField](#)，或使用 [Advanced.FieldReflector](#) 類別來簡化程序，如上述範例所示。

以動態方式定義訊息和確認

您也可以使用 [FormBuilder](#) 以動態方式定義訊息和確認。只有在表單中先前的步驟為非使用中或已完成時，才會執行每個訊息和確認。

此程式碼範例會示範動態產生的確認如何計算三明治成本。

```
.Confirm(async (state) =>
{
    var cost = 0.0;
    switch (state.Length)
    {
        case LengthOptions.SixInch: cost = 5.0; break;
        case LengthOptions.FootLong: cost = 6.50; break;
    }
    return new PromptAttribute($"Total for your sandwich is {cost:C2} is that ok?");
})
```

使用 [FormBuilder](#) 來自訂表單

此程式碼範例會使用 [FormBuilder](#) 來定義表單步驟、[驗證選取項目](#)，並以動態方式定義欄位值和確認。根據預設，表單中的步驟會以列示的序列執行。不過，若欄位已包含值，或在已指定明確的導覽時，系統可能會略過相關步驟。

```
public static IForm<SandwichOrder> BuildForm()
{
    OnCompletionAsyncDelegate<SandwichOrder> processOrder = async (context, state) =>
    {
        await context.PostAsync("We are currently processing your sandwich. We will message you the status.");
    };

    return new FormBuilder<SandwichOrder>()
        .Message("Welcome to the sandwich order bot!")
        .Field(nameof(Sandwich))
        .Field(nameof(Length))
        .Field(nameof(Bread))
        .Field(nameof(Cheese))
        .Field(nameof(Toppings),
            validate: async (state, value) =>
            {
                var values = ((List<object>)value).OfType<ToppingOptions>();
                var result = new ValidateResult { IsValid = true, Value = values };
                if (values != null && values.Contains(ToppingOptions.Everything))
                {
                    result.Value = (from ToppingOptions topping in Enum.GetValues(typeof(ToppingOptions))
                        where topping != ToppingOptions.Everything && !values.Contains(topping)
                        select topping).ToList();
                }
                return result;
            })
        .Message("For sandwich toppings you have selected {Toppings}.")
        .Field(nameof(SandwichOrder.Sauces))
        .Field(new FieldReflector<SandwichOrder>(nameof(Specials))
            .SetType(null)
            .SetActive((state) => state.Length == LengthOptions.FootLong)
            .SetDefine(async (state, field) =>
```

```

    {
        field
            .AddDescription("cookie", "Free cookie")
            .AddTerms("cookie", "cookie", "free cookie")
            .AddDescription("drink", "Free large drink")
            .AddTerms("drink", "drink", "free drink");
        return true;
    }))
    .Confirm(async (state) =>
{
    var cost = 0.0;
    switch (state.Length)
    {
        case LengthOptions.SixInch: cost = 5.0; break;
        case LengthOptions.FootLong: cost = 6.50; break;
    }
    return new PromptAttribute($"Total for your sandwich is {cost:C2} is that ok?");
})
.Field(nameof(SandwichOrder.DeliveryAddress),
validate: async (state, response) =>
{
    var result = new ValidateResult { IsValid = true, Value = response };
    var address = (response as string).Trim();
    if (address.Length > 0 && (address[0] < '0' || address[0] > '9'))
    {
        result.Feedback = "Address must start with a number.";
        result.IsValid = false;
    }
    return result;
})
.Field(nameof(SandwichOrder.DeliveryTime), "What time do you want your sandwich delivered? {}")
.Confirm("Do you want to order your {Length} {Sandwich} on {Bread} {&Bread} with {{Cheese} {Toppings} {Sauces}} to be sent to {DeliveryAddress} {?at {DeliveryTime:t}}?")
.AddRemainingFields()
.Message("Thanks for ordering a sandwich!")
.OnCompletion(processOrder)
.Build();
}

```

在此範例中，表單會執行下列步驟：

- 顯示歡迎訊息。
- 填入 `SandwichOrder.Sandwich`。
- 填入 `SandwichOrder.Length`。
- 填入 `SandwichOrder.Bread`。
- 填入 `SandwichOrder.Cheese`。
- 如果使用者選取 `ToppingOptions.Everything`，會填入 `SandwichOrder.Toppings` 並新增遺漏值。- 顯示確認選取配料的訊息。
- 填入 `SandwichOrder.Sauces`。
- **以動態方式定義** `SandwichOrder.Specials` 的欄位值。
- **以動態方式定義** 三明治成本確認。
- 填入 `SandwichOrder.DeliveryAddress` 並**確認**產生的字串。如果地址不是以數字開頭，表單會傳回訊息。
- 以自訂提示填入 `SandwichOrder.DeliveryTime`。
- 確認訂單。
- 新增任何已在類別中定義，但未由 `Field` 明確參考的其餘欄位。(如果範例未呼叫 `AddRemainingFields` 方法，則表單不會包含任何未明確參考的欄位。)
- 顯示感謝訊息。
- 定義 `OnCompletionAsync` 處理常式以處理訂單。

範例程式碼

如需示範如何使用適用於 .NET 的 Bot Framework SDK 實作 FormFlow 的完整範例，請參閱 GitHub 中的 [多對話 Bot 範例 \(英文\)](#) 和 [Contoso Flowers Bot 範例 \(英文\)](#)。

其他資源

- [FormFlow 的基本功能](#)
- [FormFlow 的進階功能](#)
- [將表單內容當地語系化](#)
- [使用 JSON 結構描述來定義表單](#)
- [使用模式語言來自訂使用者體驗](#)
- [適用於 .NET 的 Bot Framework SDK 參考](#)

使用模式語言自訂使用者體驗

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

當您自訂提示或覆寫預設範本時，可以使用模式語言來指定提示的內容和/或格式。

提示與範本

提示會定義要傳送給使用者以要求某部分資訊或要求確認的訊息。您可以使用 [Prompt 屬性](#)或隱含地透過 [IFormBuilder.Field](#)來自訂提示。

表單會使用範本，自動建構提示以及其他項目(例如說明)。您可以使用 [Template 屬性](#)來覆寫類別或欄位的預設範本。

TIP

[FormConfiguration.Templates](#) 類別會定義一組內建範本，以提供如何使用模式語言的良好範例。

模式語言的元素

模式語言會使用大括號 (`{}`) 來識別將在執行階段以實際值取代的元素。下表列出模式語言的元素。

元素	說明
<code>{<format>}</code>	顯示目前欄位 (要套用屬性的欄位) 的值。
<code>{&}</code>	顯示目前欄位的描述 (除非另有指定，否則，這是欄位的名稱)。
<code>{<field><format>}</code>	顯示具名欄位的值。
<code>{&<field>}</code>	顯示具名欄位的描述。
<code>{ }</code>	顯示目前的選項，這可能是欄位的目前值、「無喜好設定」或列舉的值。
<code>{[<field><format>} ...]}</code>	使用 分隔符號 和 LastSeparator 來分隔清單中的個別值，以顯示具名欄位的值清單。
<code>{*}</code>	每一行均顯示一個作用中欄位；每一行都會包含欄位描述和目前的值。
<code>{*filled}</code>	每一行均顯示一個包含實際值的作用中欄位；每一行都會包含欄位描述和目前的值。

元素	說明
{<nth><format>}	一般 C# 格式規範，適用於範本的第 n 個引數。如需可用的引數清單，請參閱 TemplateUsage 。
{?<textOrPatternElement>...}	條件式替代。如果所有參考到的模式元素都有值，則會替代值並使用整個運算式。

對於上方所列的元素：

- <field> 預留位置是您表單類別中的欄位名稱。例如，如果您的類別包含名稱為 `Size` 的欄位，您可以指定 `{Size}` 作為模式元素。
- 模式元素內的省略符號 ("...") 指出元素可能包含多個值。
- <format> 預留位置是一個 C# 格式規範。例如，如果類別包含名稱為 `Rating` 且類型為 `double` 的欄位，則您可以指定 `{Rating:F2}` 作為模式元素來顯示兩位數的精確度。
- <nth> 預留位置會參考範本的第 n 個引數。

Prompt 屬性中的模式語言

這個範例會使用 {&} 元素來顯示 Sandwich 欄位的描述，以及使用 {||} 元素來顯示該欄位的選項清單。

```
[Prompt("What kind of {&} would you like? {||}")]
public SandwichOptions? Sandwich;
```

以下是輸出提示：

```
What kind of sandwich would you like?
1. BLT
2. Black Forest Ham
3. Buffalo Chicken
4. Chicken And Bacon Ranch Melt
5. Cold Cut Combo
6. Meatball Marinara
7. Oven Roasted Chicken
8. Roast Beef
9. Rotisserie Style Chicken
10. Spicy Italian
11. Steak And Cheese
12. Sweet Onion Teriyaki
13. Tuna
14. Turkey Breast
15. Veggie
>
```

格式化參數

提示和範本均支援這些格式化參數。

使用量	說明
AllowDefault	適用於 { } 模式元素。判斷表單是否應該將欄位的目前值顯示為可能的選項。若為 <code>true</code> ，即會將目前的值顯示為可能的值。預設值為 <code>true</code> 。

使用量	說明
ChoiceCase	適用於 <code>{ }</code> 模式元素。判斷是否要將每個選項的文字正規化 (例如, 每個字組的第一個字母是否為大寫)。預設值為 <code>CaseNormalization.None</code> 。如需可能的值, 請參閱 CaseNormalization 。
ChoiceFormat	適用於 <code>{ }</code> 模式元素。判斷是否要將選項清單顯示為編號清單或項目符號清單。如果要顯示為編號清單, 請將 <code>ChoiceFormat</code> 設定為 <code>{0}</code> (預設值)。如果要顯示為項目符號清單, 則將 <code>ChoiceFormat</code> 設定為 <code>{1}</code> 。
ChoiceLastSeparator	適用於 <code>{ }</code> 模式元素。判斷內嵌的選項清單是否要在最後一個選項之前包括分隔符號。
ChoiceParens	適用於 <code>{ }</code> 模式元素。判斷內嵌的選項清單是否要顯示於括號內。若為 <code>true</code> , 選項清單就會顯示於括號內。預設值為 <code>true</code> 。
ChoiceSeparator	適用於 <code>{ }</code> 模式元素。判斷內嵌的選項清單是否要在每個選項 (最後一個選項除外) 之前包括分隔符號。
ChoiceStyle	適用於 <code>{ }</code> 模式元素。判斷選項清單是否要以內嵌方式顯示或每一行顯示。預設值是 <code>ChoiceStyleOptions.Auto</code> , 其會判斷在執行階段, 是否要以內嵌方式或在清單中顯示選項。如需可能的值, 請參閱 ChoiceStyleOptions 。
Feedback	僅適用於提示。判斷表單是否會回應使用者的選擇, 以指出表單已了解該選項。預設值是 <code>FeedbackOptions.Auto</code> , 只有在不了解某部分的使用者輸入時才會加以回應。如需可能的值, 請參閱 FeedbackOptions 。
FieldCase	判斷是否要將欄位描述的文字正規化 (例如, 每個字組的第一個字母是否為大寫)。預設值是 <code>CaseNormalization.Lower</code> , 會將描述轉換成小寫。如需可能的值, 請參閱 CaseNormalization 。
LastSeparator	適用於 <code>{[]}</code> 模式元素。判斷項目陣列是否要在最後一個項目之前包括分隔符號。
Separator	適用於 <code>{[]}</code> 模式元素。判斷項目陣列是否要在陣列中的每個項目 (最後一個項目除外) 之前包括分隔符號。
ValueCase	判斷是否要將欄位值的文字正規化 (例如, 每個字組的第一個字母是否為大寫)。預設值是 <code>CaseNormalization.InitialUpper</code> , 其會將每個字組的第一個字母轉換成大寫。如需可能的值, 請參閱 CaseNormalization 。

具有格式化參數的 Prompt 屬性

這個範例會使用 `ChoiceFormat` 參數, 來指定應該將選項清單顯示為項目符號清單。

```
[Prompt("What kind of {&} would you like? {||}", ChoiceFormat="{1}")]
public SandwichOptions? Sandwich;
```

以下是輸出提示：

```
What kind of sandwich would you like?  
* BLT  
* Black Forest Ham  
* Buffalo Chicken  
* Chicken And Bacon Ranch Melt  
* Cold Cut Combo  
* Meatball Marinara  
* Oven Roasted Chicken  
* Roast Beef  
* Rotisserie Style Chicken  
* Spicy Italian  
* Steak And Cheese  
* Sweet Onion Teriyaki  
* Tuna  
* Turkey Breast  
* Veggie  
>
```

範例程式碼

如需示範如何使用適用於 .NET 的 Bot Framework SDK 實作 FormFlow 的完整範例，請參閱 GitHub 中的[多對話 Bot 範例](#) (英文) 和 [Contoso Flowers Bot 範例](#) (英文)。

其他資源

- [FormFlow 的基本功能](#)
- [FormFlow 的進階功能](#)
- [使用 FormBuilder 自訂表單](#)
- [將表單內容當地語系化](#)
- [使用 JSON 結構描述來定義表單](#)
- [適用於 .NET 的 Bot Framework SDK 參考](#)

將表單內容當地語系化

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

表單的當地語系化語言是由目前執行緒的 `CurrentUICulture` 和 `CurrentCulture` 所決定。根據預設，文化特性是衍生自目前訊息的 [地區設定] 欄位，但是您可以覆寫該預設行為。根據您建構 Bot 的方式，當地語系化資訊可能來自最多三種不同的來源：

- **PromptDialog** 和 **FormFlow** 的內建當地語系化
- 您在表單中針對靜態字串產生的資源檔
- 您使用動態計算欄位、訊息或確認的字串所建立的資源檔

在表單中針對靜態字串產生資源檔

表單中的靜態字串包含表單從 C# 類別中的資訊所產生的字串，以及您指定為提示、範本、訊息或確認的字串。系統不會將從內建範本產生的字串視為靜態字串，因為這些字串已當地語系化。因為表單中有許多字串是自動產生的，所以直接使用一般 C# 資源字串並不可行。相反地，您可以藉由呼叫 `IFormBuilder.SaveResources` 或使用適用於 .NET 的 Bot 建立器 SDK 隨附的 **RView** 工具，為表單中的靜態字串產生資源檔。

使用 `IFormBuilder.SaveResources`

您可以藉由在表單上呼叫 `IFormBuilder.SaveResources`，將字串儲存至 .resx 檔案。

使用 **RView**

或者，您也可以藉由使用適用於 .NET 的 Bot 建立器 SDK 隨附的 **RView** 工具，建立以 .dll 或 .exe 為基礎的資源檔。若要產生 .resx 檔案，請執行 **rview** 並且指定組件，其中包含您的靜態表單建置方法和該方法的路徑。此程式碼片段示範如何使用 **RView** 來產生 `Microsoft.Bot.Sample.AnnotatedSandwichBot.SandwichOrder.resx` 資源檔。

```
rview -g Microsoft.Bot.Sample.AnnotatedSandwichBot.dll  
Microsoft.Bot.Sample.AnnotatedSandwichBot.SandwichOrder.BuildForm
```

此摘要示範部分 .resx 檔案，該檔案是藉由執行此 **rview** 命令而產生的。

```
<data name="Specials_description;VALUE" xml:space="preserve">
<value>Specials</value>
</data>
<data name="DeliveryAddress_description;VALUE" xml:space="preserve">
<value>Delivery Address</value>
</data>
<data name="DeliveryTime_description;VALUE" xml:space="preserve">
<value>Delivery Time</value>
</data>
<data name="PhoneNumber_description;VALUE" xml:space="preserve">
<value>Phone Number</value>
</data>
<data name="Rating_description;VALUE" xml:space="preserve">
<value>your experience today</value>
</data>
<data name="message0;LIST" xml:space="preserve">
<value>Welcome to the sandwich order bot!</value>
</data>
<data name="Sandwich_terms;LIST" xml:space="preserve">
<value>sandwichs?</value>
</data>
```

設定您的專案

產生資源檔之後，請新增至您的專案中，然後藉由完成下列步驟來設定中性語言：

1. 以滑鼠右鍵按一下專案，然後選取 [應用程式]。
2. 按一下 [組件資訊]。
3. 選取與您用來開發 Bot 的語言對應的 [中性語言] 值。

建立您的表單時，`IFormBuilder.Build` 方法會自動尋找資源，其中包含您的表單類型名稱，並且使用這些資源來將表單中的靜態字串當地語系化。

NOTE

使用 `Advanced.Field.SetDefine` 定義的動態計算欄位 (如同[使用動態欄位](#)中所述) 無法透過與靜態欄位相同的方式進行當地語系化，因為動態計算欄位的字串是在表單填入時建構。但是，您可以藉由使用一般 C# 當地語系化機制，將動態計算欄位當地語系化。

將資源檔當地語系化

將資源檔新增至您的專案之後，您可以使用[多語應用程式工具組 \(MAT\)](#) 將資源檔當地語系化。安裝 **MAT**，然後藉由完成下列步驟針對專案加以啟用：

1. 在 Visual Studio 方案總管中選取您的專案。
2. 按一下 [工具]、[多語應用程式工具組]，然後按一下 [啟用]。
3. 以滑鼠右鍵按一下專案，然後選取 [多語應用程式工具組]、[新增翻譯] 以選取翻譯。這樣會建立業界標準的 `XLF` 檔案，您可以自動或手動進行翻譯。

NOTE

雖然這篇文章說明如何使用多語應用程式工具組來將內容當地語系化，但是您可以透過其他各種方式來實作當地語系化。

了解它是如何運作的

這個程式碼範例是在[使用 FormBuilder 來自訂表單](#)其中一個程式碼的基礎上所建置，以便如上所述進行當地語系化。在此範例中，`DynamicSandwich` 類別 (此處未顯示) 包含動態計算欄位、訊息和確認的當地語系化資訊。

```

public static IForm<SandwichOrder> BuildLocalizedForm()
{
    var culture = Thread.CurrentThread.CurrentCulture;
    IForm<SandwichOrder> form;
    if (!_forms.TryGetValue(culture, out form))
    {
        OnCompletionAsyncDelegate<SandwichOrder> processOrder = async (context, state) =>
        {
            await context.PostAsync(DynamicSandwich.Processing);
        };
        // FormBuilder uses the thread culture to automatically switch framework strings and static strings.
        // Dynamically defined fields must do their own localization.
        var builder = new FormBuilder<SandwichOrder>()
            .Message("Welcome to the sandwich order bot!")
            .Field(nameof(Sandwich))
            .Field(nameof(Length))
            .Field(nameof(Bread))
            .Field(nameof(Cheese))
            .Field(nameof(Toppings),
                validate: async (state, value) =>
            {
                var values = ((List<object>)value).OfType<ToppingOptions>();
                var result = new ValidationResult { IsValid = true, Value = values };
                if (values != null && values.Contains(ToppingOptions.Everything))
                {
                    result.Value = (from ToppingOptions topping in
Enum.GetValues(typeof(ToppingOptions))
                        where topping != ToppingOptions.Everything &&
!values.Contains(topping)
                        select topping).ToList();
                }
                return result;
            })
            .Message("For sandwich toppings you have selected {Toppings}.")
            .Field(nameof(SandwichOrder.Sauces))
            .Field(new FieldReflector<SandwichOrder>(nameof(Specials))
                .SetType(null)
                .SetActive((state) => state.Length == LengthOptions.FootLong)
                .SetDefine(async (state, field) =>
            {
                field
                    .AddDescription("cookie", DynamicSandwich.FreeCookie)
                    .AddTerms("cookie", Language.GenerateTerms(DynamicSandwich.FreeCookie, 2))
                    .AddDescription("drink", DynamicSandwich.FreeDrink)
                    .AddTerms("drink", Language.GenerateTerms(DynamicSandwich.FreeDrink, 2));
                return true;
            }))
            .Confirm(async (state) =>
            {
                var cost = 0.0;
                switch (state.Length)
                {
                    case LengthOptions.SixInch: cost = 5.0; break;
                    case LengthOptions.FootLong: cost = 6.50; break;
                }
                return new PromptAttribute(string.Format(DynamicSandwich.Cost, cost) + "{||}");
            })
            .Field(nameof(SandwichOrder.DeliveryAddress),
                validate: async (state, response) =>
            {
                var result = new ValidationResult { IsValid = true, Value = response };
                var address = (response as string).Trim();
                if (address.Length > 0 && address[0] < '0' || address[0] > '9')
                {
                    result.Feedback = DynamicSandwich.BadAddress;
                    result.IsValid = false;
                }
                return result;
            })
    }
}

```

```

        })
        .Field(nameof(SandwichOrder.DeliveryTime), "What time do you want your sandwich delivered?
{|||}")
        .Confirm("Do you want to order your {Length} {Sandwich} on {Bread} {&Bread} with {{Cheese}
{Toppings} {Sauces}}} to be sent to {DeliveryAddress} {?at {DeliveryTime:t}}?")
        .AddRemainingFields()
        .Message("Thanks for ordering a sandwich!")
        .OnCompletion(processOrder);
    builder.Configuration.DefaultPrompt.ChoiceStyle = ChoiceStyleOptions.Auto;
    form = builder.Build();
    _forms[culture] = form;
}
return form;
}

```

當 `CurrentUICulture` 是法文時，此程式碼片段會顯示 Bot 與使用者之間產生的互動。

```

Bienvenue sur le bot d'ordre "sandwich" !
Quel genre de "sandwich" vous souhaitez sur votre "sandwich"?
1. BLT
2. Jambon Forêt Noire
3. Poulet Buffalo
4. Faire fondre le poulet et Bacon Ranch
5. Combo de coupe à froid
6. Boulette de viande Marinara
7. Poulet rôti au four
8. Rôti de boeuf
9. Rotisserie poulet
10. Italienne piquante
11. Bifteck et fromage
12. Oignon doux Teriyaki
13. Thon
14. Poitrine de dinde
15. Veggie
> 2

Quel genre de longueur vous souhaitez sur votre "sandwich"?
1. Six pouces
2. Pied Long
> ?
* Vous renseignez le champ longueur.Réponses possibles:
* Vous pouvez saisir un numéro 1-2 ou des mots de la description. (Six pouces, ou Pied Long)
* Retourner à la question précédente.
* Assistance: Montrez les réponses possibles.
* Abandonner: Abandonner sans finir
* Recommencer remplir le formulaire. (Vos réponses précédentes sont enregistrées.)
* Statut: Montrer le progrès en remplaçant le formulaire jusqu'à présent.
* Vous pouvez passer à un autre champ en entrant son nom. ("Sandwich", Longueur, Pain, Fromage, Nappages,
Sauces, Adresse de remise, Délai de livraison, ou votre expérience aujourd'hui).
Quel genre de longueur vous souhaitez sur votre "sandwich"?
1. Six pouces
2. Pied Long
> 1

Quel genre de pain vous souhaitez sur votre "sandwich"?
1. Neuf grains de blé
2. Neuf grains miel avoine
3. Italien
4. Fromage et herbes italiennes
5. Pain plat
> neuf
Par pain "neuf" vouliez-vous dire (1. Neuf grains miel avoine, ou 2. Neuf grains de blé)

```

範例程式碼

如需示範如何使用適用於 .NET 的 Bot Framework SDK 實作 FormFlow 的完整範例，請參閱 GitHub 中的[多對話 Bot 範例](#) (英文) 和[Contoso Flowers Bot 範例](#) (英文)。

其他資源

- [FormFlow 的基本功能](#)
- [FormFlow 的進階功能](#)
- [使用 FormBuilder 來自訂表單](#)
- [使用 JSON 結構描述來定義表單](#)
- [使用模式語言來自訂使用者體驗](#)
- [適用於 .NET 的 Bot Framework SDK 參考](#)

使用 JSON 結構描述來定義表單

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

如果您透過 FormFlow 建立 Bot 時使用了 [C# 類別](#)來定義表單，表單會衍生自 C# 中類型的靜態定義。或者，您也可以改用 [JSON 結構描述](#) (英文) 來定義表單。使用 JSON 結構描述所定義的表單完全由料驅動；只要更新結構描述即可變更表單 (因而變更 Bot 的行為)。

JSON 結構描述會描述 [JObject](#) (英文) 內的欄位，且包含控制提示、範本和字詞的註釋。若要使用 JSON 結構描述搭配 FormFlow，您必須將 `Microsoft.Bot.Builder.FormFlow.Json` NuGet 套件新增至專案中，並匯入 `Microsoft.Bot.Builder.FormFlow.Json` 命名空間。

標準關鍵字

FormFlow 可支援下列標準 [JSON 結構描述](#) (英文) 關鍵字：

關鍵字	說明
<code>type</code>	會定義欄位包含的資料類型。
<code>列舉</code>	會定義欄位的有效值。
<code>minimum</code>	會定義欄位允許的最小數值 (如 NumericAttribute 中所述)。
<code>maximum</code>	會定義欄位允許的最大數值 (如 NumericAttribute 中所述)。
<code>必要</code>	會定義哪些欄位為必要。
<code>模式</code>	會驗證字串值 (如 PatternAttribute 中所述)。

JSON 結構描述延伸模組

FormFlow 可延伸標準 [JSON 結構描述](#) (英文) 以支援數個額外屬性。

位於結構描述根目錄的其他屬性

屬性	值
<code>OnCompletion</code>	含引數 <code>(IDialogContext context, JObject state)</code> 的 C# 指令碼用來完成表單。
<code>參考</code>	要包含在指令碼中的參考。例如： <code>[assemblyReference, ...]</code> 。路徑應為絕對或相對於目前的目錄。根據預設，指令碼會包含 <code>Microsoft.Bot.Builder.dll</code> 。

屬性	值
匯入	要包含在指令碼中的匯入。例如: <code>[import, ...]</code> 。根據預設, 指令碼會包含 <code>Microsoft.Bot.Builder</code> 、 <code>Microsoft.Bot.Builder.Dialogs</code> 、 <code>Microsoft.Bot.Builder.FormFlow</code> 、 <code>Microsoft.Bot.Builder.FormFlow.Advanced</code> 、 <code>System.Collections.Generic</code> 和 <code>System.Linq</code> 命名空間。

位於結構描述根目錄, 或作為類型屬性同儕節點的其他屬性

屬性	值
範本	<code>{ TemplateUsage: { Patterns: [string, ...], <args> }, ... }</code>
Prompt	<code>{ Patterns:[string, ...] <args> }</code>

若要在 JSON 結構描述中指定範本和提示, 請使用 [TemplateAttribute](#) 和 [PromptAttribute](#) 所定義的相同詞彙。結構描述中的屬性名稱和值, 應符合基礎 C# 列舉中的屬性名稱和值。例如, 此結構描述程式碼片段定義的範本, 會覆寫 `TemplateUsage.NotUnderstood` 範本並指定 `TemplateBaseAttribute.ChoiceStyle` :

```
"Templates":{ "NotUnderstood": { "Patterns": ["I don't get it"], "ChoiceStyle":"Auto"}}
```

作為類型屬性同儕節點的其他屬性

屬性	內容	說明
Datetime	布林	指出欄位是否為 <code>DateTime</code> 欄位。
說明	字串或物件	如 DescribeAttribute 中所述的欄位說明。
詞彙	<code>[string,...]</code>	如 TermsAttribute 中所述, 用於比對欄位值的規則運算式。
MaxPhrase	int	透過 <code>Language.GenerateTerms(string, int)</code> 執行字詞以將其展開。
值	<code>`{ string: {Describe:string</code>	<code>object, Terms:[string, ...], MaxPhrase}, ...`}</code>
Active	script	含引數 <code>(JObject state)->bool</code> 的 C# 指令碼用來測試欄位、訊息或確認是否為作用中。
驗證	script	含引數 <code>(JObject state, object value)->ValidateResult</code> 的 C# 指令碼用來驗證欄位值。

屬性	內容	說明
定義	script	含引數 (JObject state, Field< JObject> field) 的 C# 指令碼用來以動態方式定義欄位。
下一页	script	含引數 (object value, JObject state) 的 C# 指令碼用來在填入欄位後判斷下一個步驟。
先前	`[confirm	message, ...]`
後續	`[confirm	message, ...]`
相依性	[string, ...]	此欄位、訊息或確認所相依的欄位。

透過在 **Before** 屬性或 **After** 屬性值內使用含引數 (JObject state) 的 C# 指令碼，或隨機選取的一組模式搭配選用的範本引數，使用 {Confirm:script|[string, ...], ...templateArgs} 來定義確認。

透過在 **Before** 屬性或 **After** 屬性值內使用含引數 (JObject state) 的 C# 指令碼，或隨機選取的一組模式搭配選用的範本引數，使用 {Message:script|[string, ...] ...templateArgs} 來定義訊息。

指令碼

上述屬性中有數個屬性包含指令碼作為屬性值。指令碼可以是 C# 程式碼的任何程式碼片段，您通常可以在方法的主體中找到。您可以使用 **References** 屬性和/或 **Imports** 屬性來新增參考。特殊的全域變數包括：

變數	說明
選項	指令碼要執行的內部分派。
state	針對所有指令碼繫結的 JObject 表單狀態。
ifield	針對訊息/確認提示建立器以外的所有指令碼， IField< JObject > 允許對目前欄位進行推論。
value	要為 Validate 驗證的物件值。
field	Field< JObject > 允許動態更新 Define 中的欄位。
context	IDialogContext 內容允許將結果張貼在 OnCompletion 中。

透過 JSON 結構描述定義的欄位具有與任何其他欄位相同的功能，可用程式設計方式延伸或覆寫定義。也可用相同方式進行當地語系化。

JSON 結構描述範例

定義表單最簡單的方式是直接在 JSON 結構描述中定義所有項目，包括任何 C# 程式碼。此範例會示範已標註三明治 Bot 的 JSON 結構描述，如[使用 FormBuilder 來自訂表單](#)中所述。

```
"References": [ "Microsoft.Bot.Sample.AnnotatedSandwichBot.dll" ],
"Imports": [ "Microsoft.Bot.Sample.AnnotatedSandwichBot.Resource" ],
"type": "object",
"required": [
    "Sandwich",
    "Length",
    "Ingredients",
    "DeliveryAddress"
],
"Templates": {
    "NotUnderstood": {
        "Patterns": [ "I do not understand \"{0}\".", "Try again, I don't get \"{0}\"." ]
    },
    "EnumSelectOne": {
        "Patterns": [ "What kind of {&} would you like on your sandwich? {||}" ],
        "ChoiceStyle": "Auto"
    }
},
"properties": {
    "Sandwich": {
        "Prompt": { "Patterns": [ "What kind of {&} would you like? {||}" ] },
        "Before": [ { "Message": [ "Welcome to the sandwich order bot!" ] } ],
        "Describe": { "Image": "https://placeholdit.imgix.net/~text?txtsize=16&txt=Sandwich&w=125&h=40&txttrack=0&txtclr=000&txtfont=bold" },
        "type": [
            "string",
            "null"
        ],
        "enum": [
            "BLT",
            "BlackForestHam",
            "BuffaloChicken",
            "ChickenAndBaconRanchMelt",
            "ColdCutCombo",
            "MeatballMarinara",
            "OvenRoastedChicken",
            "RoastBeef",
            "RotisserieStyleChicken",
            "SpicyItalian",
            "SteakAndCheese",
            "SweetOnionTeriyaki",
            "Tuna",
            "TurkeyBreast",
            "Veggie"
        ],
        "Values": {
            "RotisserieStyleChicken": {
                "Terms": [ "rotis\\w* style chicken" ],
                "MaxPhrase": 3
            }
        }
    },
    "Length": {
        "Prompt": {
            "Patterns": [ "What size of sandwich do you want? {||}" ]
        },
        "type": [
            "string",
            "null"
        ],
        "enum": [
            "SixInch",
            "FootLong"
        ]
    },
    "Ingredients": {
        "type": "object",
        "required": [ "Bread" ],
        "properties": {
            "Bread": {
                "type": "string"
            }
        }
    }
}
```

```

properties : [
    "Bread": {
        "type": [
            "string",
            "null"
        ],
        "Describe": {
            "Title": "Sandwich Bot",
            "SubTitle": "Bread Picker"
        },
        "enum": [
            "NineGrainWheat",
            "NineGrainHoneyOat",
            "Italian",
            "ItalianHerbsAndCheese",
            "Flatbread"
        ]
    },
    "Cheese": {
        "type": [
            "string",
            "null"
        ],
        "enum": [
            "American",
            "MontereyCheddar",
            "Pepperjack"
        ]
    },
    "Toppings": {
        "type": "array",
        "items": {
            "type": "integer",
            "enum": [
                "Everything",
                "Avocado",
                "BananaPeppers",
                "Cucumbers",
                "GreenBellPeppers",
                "Jalapenos",
                "Lettuce",
                "Olives",
                "Pickles",
                "RedOnion",
                "Spinach",
                "Tomatoes"
            ],
            "Values": {
                "Everything": { "Terms": [ "except", "but", "not", "no", "all", "everything" ] }
            }
        },
        "Validate": "var values = ((List<object>) value).OfType<string>(); var result = new ValidateResult
{IsValid = true, Value = values} ; if (values != null && values.Contains(\"Everything\")) { result.Value =
(from topping in new string[] { \"Avocado\", \"BananaPeppers\", \"Cucumbers\", \"GreenBellPeppers\",
\"Jalapenos\", \"Lettuce\", \"Olives\", \"Pickles\", \"RedOnion\", \"Spinach\", \"Tomatoes\"} where
!values.Contains(topping) select topping).ToList();} return result;",
        "After": [ { "Message": [ "For sandwich toppings you have selected {Ingredients.Toppings}." ] } ]
    },
    "Sauces": {
        "type": [
            "array",
            "null"
        ],
        "items": {
            "type": "string",
            "enum": [
                "ChipotleSouthwest",
                "HoneyMustard",
                "LightMayonnaise",
                "DijonMustard"
            ]
        }
    }
]

```

```

        "KegularMayonnaise",
        "Mustard",
        "Oil",
        "Pepper",
        "Ranch",
        "SweetOnion",
        "Vinegar"
    ],
}
}
},
},
"Specials": {
    "Templates": {
        "NoPreference": { "Patterns": [ "None" ] }
    },
    "type": [
        "string",
        "null"
    ],
    "Active": "return (string) state[\"Length\"] == \"FootLong\";",
    "Define": "field.setType(null).AddDescription(\"cookie\",
DynamicSandwich.FreeCookie).AddTerms(\"cookie\", Language.GenerateTerms(DynamicSandwich.FreeCookie,
2)).AddDescription(\"drink\", DynamicSandwich.FreeDrink).AddTerms(\"drink\",
Language.GenerateTerms(DynamicSandwich.FreeDrink, 2)); return true;",
    "After": [ { "Confirm": "var cost = 0.0; switch ((string) state[\"Length\"]) { case \"SixInch\": cost =
5.0; break; case \"FootLong\": cost=6.50; break;} return new PromptAttribute($\"Total for your sandwich is
{cost:C2} is that ok?\");" } ]
},
    "DeliveryAddress": {
        "type": [
            "string",
            "null"
        ],
        "Validate": "var result = new ValidateResult{ IsValid = true, Value = value}; var address = (value as
string).Trim(); if (address.Length > 0 && (address[0] < '0' || address[0] > '9')) {result.Feedback =
DynamicSandwich.BadAddress; result.IsValid = false; } return result;"
    },
    "PhoneNumber": {
        "type": [ "string", "null" ],
        "pattern": "(\\(\\d{3}\\))?\\s*\\d{3}(-|\\s*)\\d{4}"
    },
    "DeliveryTime": {
        "Templates": {
            "StatusFormat": {
                "Patterns": [ "{&}:{t}" ],
                "FieldCase": "None"
            }
        },
        "DateTime": true,
        "type": [
            "string",
            "null"
        ],
        "After": [ { "Confirm": [ "Do you want to order your {Length} {Sandwich} on {Ingredients.Bread}
&{Ingredients.Bread} with {{Ingredients.Cheese} {Ingredients.Toppings} {Ingredients.Sauces} to be sent to
{DeliveryAddress} ?at {DeliveryTime}}?" ] } ]
    },
    "Rating": {
        "Describe": "your experience today",
        "type": [
            "number",
            "null"
        ],
        "minimum": 1,
        "maximum": 5,
        "After": [ { "Message": [ "Thanks for ordering your sandwich!" ] } ]
    }
},

```

```
"OnCompletion": "await context.PostAsync(\"We are currently processing your sandwich. We will message you the status.\");"
```

使用 JSON 結構描述實來作 FormFlow

若要使用 JSON 結構描述實來作 FormFlow，請使用 `FormBuilderJson`，這樣可支援與 `FormBuilder` 相同的 Fluent 介面。此程式碼範例會示範如何實作已標註三明治 Bot 的 JSON 結構描述，如[使用 FormBuilder 來自訂表單](#)中所述。

```
public static IForm< JObject> BuildJsonForm()
{
    using (var stream =
Assembly.GetExecutingAssembly().GetManifestResourceStream("Microsoft.Bot.Sample.AnnotatedSandwichBot.AnnotatedSandwich.json"))
    {
        var schema = JObject.Parse(new StreamReader(stream).ReadToEnd());
        return new FormBuilderJson(schema)
            .AddRemainingFields()
            .Build();
    }
    ...
}
```

範例程式碼

如需示範如何使用適用於 .NET 的 Bot Framework SDK 實作 FormFlow 的完整範例，請參閱 GitHub 中的[多對話 Bot 範例](#) (英文) 和 [Contoso Flowers Bot 範例](#) (英文)。

其他資源

- [FormFlow 的基本功能](#)
- [FormFlow 的進階功能](#)
- [使用 FormBuilder 自訂表單](#)
- [將表單內容當地語系化](#)
- [使用模式語言來自訂使用者體驗](#)
- [適用於 .NET 的 Bot Framework SDK 參考](#)

實作通道特定的功能

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

某些通道所提供的功能，無法只透過訊息文字和附件來實作。若要實作通道特定的功能，您可以將原生中繼資料傳遞至活動物件「通道資料」屬性中的通道。例如，Bot 可以使用通道資料屬性，指示 Telegram 傳送貼圖或指示 Office365 傳送電子郵件。

此文章說明如何使用訊息活動的通道資料屬性，來實作此通道特定的功能：

通道	功能
電子郵件	傳送及接收電子郵件，其中包含本文、主旨和重要性中繼資料
Slack	傳送不失真的 Slack 訊息
Facebook	原生傳送 Facebook 通知
Telegram	執行 Telegram 特定動作，例如共用語音備忘或貼圖
Kik	傳送和接收原生 Kik 訊息

NOTE

活動物件的通道資料屬性值是一個 JSON 物件。因此，本文中的範例會說明各種案例中 `channelData` JSON 屬性的預期格式。若要使用 .NET 建立 JSON 物件，請使用 `JObject` (.NET) 類別。

建立自訂的電子郵件訊息

若要建立電子郵件訊息，請將活動物件的通道資料屬性設定為包含這些屬性的 JSON 物件：

屬性	說明
<code>bccRecipients</code>	以分號 (:) 分隔的電子郵件地址字串，用來新增至訊息的 [Bcc] (密件副本) 欄位。
<code>ccRecipients</code>	以分號 (:) 分隔的電子郵件地址字串，用來新增至訊息的 [Cc] (副本) 欄位。
<code>htmlBody</code>	會指定電子郵件訊息本文的 HTML 文件。請參閱通道的文件，以取得受支援 HTML 元素和屬性的資訊。
<code>importance</code>	電子郵件的重要性層級。有效值為高、一般和低。預設值為一般。

屬性	說明
主旨	電子郵件的主旨。請參閱通道的文件，以取得欄位需求的資訊。
toRecipients	以分號 (:) 分隔的電子郵件地址字串，用來新增至訊息的 [收件者] 欄位。

NOTE

Bot 透過電子郵件通道從使用者收到的訊息，可能會包含通道資料屬性，且其中已填入類似上述的 JSON 物件。

此程式碼片段顯示自訂電子郵件訊息的 `channelData` 屬性範例。

```
"channelData": {  
    "type": "message",  
    "locale": "en-US",  
    "channelID": "email",  
    "from": { "id": "mybot@mydomain.com", "name": "My bot"},  
    "recipient": { "id": "joe@otherdomain.com", "name": "Joe Doe"},  
    "conversation": { "id": "123123123123", "topic": "awesome chat" },  
    "channelData":  
    {  
        "htmlBody": "<html><body style = /\"font-family: Calibri; font-size: 11pt;\">This is more than  
awesome.</body></html>",  
        "subject": "Super awesome message subject",  
        "importance": "high",  
        "ccRecipients": "Yasemin@adatum.com;Temel@adventure-works.com"  
    }  
}
```

建立不失真的 Slack 訊息

若要建立不失真的 Slack 訊息，請將活動物件的通道資料屬性設定為 JSON 物件，以指定 [Slack 訊息](#)、[Slack 附件](#) 和/或 [Slack 按鈕](#)。

NOTE

若要在 Slack 訊息中支援按鈕，您必須在[將 Bot 連線](#)至 Slack 通道時啟用 [互動式訊息]。

此程式碼片段顯示自訂 Slack 訊息的 `channelData` 屬性範例。

```

"channelData": {
    "text": "Now back in stock! :tada:",
    "attachments": [
        {
            "title": "The Further Adventures of Slackbot",
            "author_name": "Stanford S. Strickland",
            "author_icon": "https://api.slack.com/img/api/homepage_custom_integrations-2x.png",
            "image_url": "http://i.imgur.com/OJkaVOI.jpg?1"
        },
        {
            "fields": [
                {
                    "title": "Volume",
                    "value": "1",
                    "short": true
                },
                {
                    "title": "Issue",
                    "value": "3",
                    "short": true
                }
            ]
        },
        {
            "title": "Synopsis",
            "text": "After @episod pushed exciting changes to a devious new branch back in Issue 1, Slackbot notifies @don about an unexpected deploy..."
        },
        {
            "fallback": "Would you recommend it to customers?",
            "title": "Would you recommend it to customers?",
            "callback_id": "comic_1234_xyz",
            "color": "#3AA3E3",
            "attachment_type": "default",
            "actions": [
                {
                    "name": "recommend",
                    "text": "Recommend",
                    "type": "button",
                    "value": "recommend"
                },
                {
                    "name": "no",
                    "text": "No",
                    "type": "button",
                    "value": "bad"
                }
            ]
        }
    ]
}

```

當使用者按一下 Slack 訊息中的按鈕時，Bot 會收到回應訊息，其中的通道資料屬性已填入 `payload` JSON 物件。`payload` 物件會指定原始訊息的內容、識別已按下的按鈕，並識別按下按鈕的使用者。

此程式碼片段顯示當使用者按一下 Slack 訊息中的按鈕時，Bot 所收到的訊息中所含有的 `channelData` 屬性範例。

```

"channelData": {
    "payload": {
        "actions": [
            {
                "name": "recommend",
                "value": "yes"
            }
        ],
        ...
    },
    "original_message": "...",
    "response_url": "https://hooks.slack.com/actions/..."
}
}

```

Bot 可以透過正常方式回覆此訊息，也可以將其回應直接張貼到 `payload` 物件的 `response_url` 屬性所指定的端點。如需何時及如何將回應張貼到 `response_url` 的資訊，請參閱 [Slack 按鈕](#)。

您可以使用下列 JSON 來建立動態按鈕。

```

{
    "text": "Would you like to play a game ?",
    "attachments": [
        {
            "text": "Choose a game to play!",
            "fallback": "You are unable to choose a game",
            "callback_id": "wopr_game",
            "color": "#3AA3E3",
            "attachment_type": "default",
            "actions": [
                {
                    "name": "game",
                    "text": "Chess",
                    "type": "button",
                    "value": "chess"
                },
                {
                    "name": "game",
                    "text": "Falken's Maze",
                    "type": "button",
                    "value": "maze"
                },
                {
                    "name": "game",
                    "text": "Thermonuclear War",
                    "style": "danger",
                    "type": "button",
                    "value": "war",
                    "confirm": {
                        "title": "Are you sure?",
                        "text": "Wouldn't you prefer a good game of chess?",
                        "ok_text": "Yes",
                        "dismiss_text": "No"
                    }
                }
            ]
        }
    ]
}

```

若要建立互動式功能表，請使用下列 JSON：

```
{
  "text": "Would you like to play a game ?",
  "response_type": "in_channel",
  "attachments": [
    {
      "text": "Choose a game to play",
      "fallback": "If you could read this message, you'd be choosing something fun to do right now.",
      "color": "#3AA3E3",
      "attachment_type": "default",
      "callback_id": "game_selection",
      "actions": [
        {
          "name": "games_list",
          "text": "Pick a game...",
          "type": "select",
          "options": [
            {
              "text": "Hearts",
              "value": "menu_id_hearts"
            },
            {
              "text": "Bridge",
              "value": "menu_id_bridge"
            },
            {
              "text": "Checkers",
              "value": "menu_id_checkers"
            },
            {
              "text": "Chess",
              "value": "menu_id_chess"
            },
            {
              "text": "Poker",
              "value": "menu_id_poker"
            },
            {
              "text": "Falken's Maze",
              "value": "menu_id_maze"
            },
            {
              "text": "Global Thermonuclear War",
              "value": "menu_id_war"
            }
          ]
        }
      ]
    }
  ]
}
```

建立 Facebook 通知

若要建立 Facebook 通知，請將活動物件的通道資料屬性設定為指定這些屬性的 JSON 物件：

屬性	說明
notification_type	通知類型 (例如 REGULAR , SILENT_PUSH , NO_PUSH)。
attachment	指定影像、視訊或其他多媒體類型的附件，或是回條等樣板化附件。

NOTE

如需 `notification_type` 屬性和 `attachment` 屬性的格式和內容詳細資料, 請參閱 [Facebook API 文件](#)。

此程式碼片段顯示 Facebook 回條附件的 `channelData` 屬性範例。

```
"channelData": {  
    "notification_type": "NO_PUSH",  
    "attachment": {  
        "type": "template"  
        "payload": {  
            "template_type": "receipt",  
            . . .  
        }  
    }  
}
```

建立 Telegram 訊息

若要建立訊息來實作 Telegram 特定動作 (例如, 共用語音備忘或貼圖), 請將活動物件的通道資料屬性設定為會指定這些屬性的 JSON 物件：

屬性	說明
method	要呼叫的 Telegram Bot API 方法。
parameters	所指定方法的參數。

支援下列 Telegram 方法：

- `answerInlineQuery`
- `editMessageCaption`
- `editMessageReplyMarkup`
- `editMessageText`
- `forwardMessage`
- `kickChatMember`
- `sendAudio`
- `sendChatAction`
- `sendContact`
- `sendDocument`
- `sendLocation`
- `sendMessage`
- `sendPhoto`
- `sendSticker`
- `sendVenue`
- `sendVideo`
- `sendVoice`
- `unbanChatMember`

如需這些 Telegram 方法和其參數的詳細資訊, 請參閱 [Telegram Bot API 文件](#)。

NOTE

- `chat_id` 參數通用於所有 Telegram 方法。如果您未指定 `chat_id` 作為參數，架構會為您提供識別碼。
- 不要傳遞內嵌檔案內容，而是應該指定使用 URL 和媒體類型的檔案，如下列範例所示。
- 在 Bot 從 Telegram 通道所收到的每則訊息內，`ChannelData` 屬性會包含 Bot 先前傳送的訊息。

此程式碼片段顯示 `channelData` 屬性範例，此屬性指定單一 Telegram 方法。

```
"channelData": {  
    "method": "sendSticker",  
    "parameters": {  
        "sticker": {  
            "url": "https://domain.com/path/gif",  
            "mediaType": "image/gif",  
        }  
    }  
}
```

此程式碼片段顯示 `channelData` 屬性範例，此屬性指定 Telegram 方法陣列。

```
"channelData": [  
    {  
        "method": "sendSticker",  
        "parameters": {  
            "sticker": {  
                "url": "https://domain.com/path/gif",  
                "mediaType": "image/gif",  
            }  
        }  
    },  
    {  
        "method": "sendMessage",  
        "parameters": {  
            "text": "<b>This message is HTML formatted.</b>",  
            "parse_mode": "HTML"  
        }  
    }  
]
```

建立原生的 Kik 訊息

若要建立原生的 Kik 訊息，請將活動物件的通道資料屬性設定為指定此屬性的 JSON 物件：

屬性	說明
上限	Kik 訊息的陣列。如需 Kik 訊息格式的詳細資訊，請參閱 Kik 訊息格式 。

此程式碼片段顯示原生 Kik 訊息的 `channelData` 屬性範例。

```

"channelData": {
    "messages": [
        {
            "chatId": "c6dd8165...",
            "type": "link",
            "to": "kikhandle",
            "title": "My Webpage",
            "text": "Some text to display",
            "url": "http://botframework.com",
            "picUrl": "http://lorempixel.com/400/200/",
            "attribution": {
                "name": "My App",
                "iconUrl": "http://lorempixel.com/50/50/"
            },
            "noForward": true,
            "kikJsData": {
                "key": "value"
            }
        }
    ]
}

```

建立 LINE 訊息

若要建立會實作 LINE 特有訊息類型 (例如貼圖、範本或 LINE 特有動作類型, 像是開啟手機相機) 的訊息, 請將活動物件的通道資料屬性設定為會指定 LINE 訊息類型和動作類型的 JSON 物件。

屬性	說明
type	LINE 動作/訊息類型名稱

支援這些 LINE 訊息類型：

- 貼圖
- 影像地圖
- 範本 (按鈕、確認、橫向捲軸)
- Flex

您可以在訊息類型 JSON 物件的動作欄位中, 指定這些 LINE 動作：

- Postback
- 訊息
- URI
- Datetimerpicker
- Camera
- Camera roll
- 位置

如需這些 LINE 方法和其參數的詳細資訊, 請參閱 [LINE Bot API 文件](#)。

此程式碼片段會舉例示範 `channelData` 屬性, 此屬性會指定通道訊息類型 `ButtonTemplate` 和 3 個動作類型：`camera`、`cameraRoll`、`Datetimerpicker`。

```
"channelData": {
    "type": "ButtonsTemplate",
    "altText": "This is a buttons template",
    "template": {
        "type": "buttons",
        "thumbnailImageUrl": "https://example.com/bot/images/image.jpg",
        "imageAspectRatio": "rectangle",
        "imageSize": "cover",
        "imageBackgroundColor": "#FFFFFF",
        "title": "Menu",
        "text": "Please select",
        "defaultAction": {
            "type": "uri",
            "label": "View detail",
            "uri": "http://example.com/page/123"
        },
        "actions": [
            {
                "type": "cameraRoll",
                "label": "Camera roll"
            },
            {
                "type": "camera",
                "label": "Camera"
            },
            {
                "type": "datetimepicker",
                "label": "Select date",
                "data": "storeId=12345",
                "mode": "datetime",
                "initial": "2017-12-25t00:00",
                "max": "2018-01-24t23:59",
                "min": "2017-12-25t00:00"
            }
        ]
    }
}
```

其他資源

- 實體和活動類型
- Bot Framework -- 活動結構描述

使用 Cortana 技能建置啟用語音的 Bot

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

適用於 .NET 的 Bot Framework SDK 可藉由將 Bot 連線到 Cortana 通道作為 Cortana 技能，讓您建置啟用語音的 Bot。

TIP

如需有關技能是什麼及其用途的詳細資訊，請參閱 [Cortana 技能套件](#)。

使用 Bot Framework 建立 Cortana 技能不需要太多 Cortana 專屬知識，主要是建置 Bot。與其他您在過去可能已建立的 Bot 可能主要差異在於，Cortana 同時具有視覺與音訊元件。對於視覺元件，Cortana 提供用於轉譯內容（例如卡片）的畫布區域。對於音訊元件，您可以在 Bot 訊息中提供文字或 SSML，Cortana 會讀給使用者聽，讓您的 Bot 擁有語音功能。

NOTE

在許多不同的裝置上皆有提供 Cortana。部分具有螢幕，而其他（如獨立喇叭）可能沒有。您應確認 Bot 能夠處理這兩者。請參閱 [Cortana 特定實體](#)，以了解如何檢查裝置資訊。

將語音新增至您的 Bot

Bot 的語音訊息會表示為語音合成標記語言（SSML）。Bot Framework SDK 可讓您將 SSML 包含在 Bot 的回應中，以在 Bot 顯示的內容之外控制 Bot 所說內容。您也可以指定 Bot 是否接受、預期還是略過使用者輸入，來控制 Cortana 的麥克風狀態。

設定 `IMessageActivity` 物件的 `Speak` 屬性，可指定 Cortana 說出的訊息。如果您指定純文字，Cortana 會決定單字發音的方式。

```
Activity reply = activity.CreateReply("This is the text that Cortana displays.");
reply.Speak = "This is the text that Cortana will say.";
```

如果您想要更充分掌控音調、語氣及強調，請將 `Speak` 屬性格式化為 [語音合成標記語言（SSML）](#)（英文）。

下列程式碼範例會指定應該以適量的強調說出 "text" 這個字：

```
Activity reply = activity.CreateReply("This is the text that will be displayed.");
reply.Speak = "<speak version=\"1.0\" xmlns=\"http://www.w3.org/2001/10/synthesis\" xml:lang=\"en-US\">This is
the <emphasis level=\"moderate\">text</emphasis> that will be spoken.</speak>";
```

`inputHint` 屬性有助於向 Cortana 指出 Bot 是否預期輸入。提示的預設值是 **ExpectingInput**，而其他類型回應的預設值為 **AcceptingInput**。

值	說明
AcceptingInput	您的 Bot 會被動接受輸入，但不會等候回應。如果使用者按住 [麥克風] 按鈕，Cortana 會接受使用者的輸入。
ExpectingInput	指出 Bot 正主動預期使用者的回應。Cortana 會聆聽使用者對麥克風說的話。
IgnoringInput	Cortana 會忽略輸入。如果您的 Bot 主動處理要求，則可能會傳送這項提示，且在完成要求之前，將會忽略來自使用者的輸入。

此範例會示範如何讓 Cortana 知道該預期使用者輸入。麥克風會保持開啟狀態。

```
// Add an InputHint to let Cortana know to expect user input
Activity reply = activity.CreateReply("This is the text that will be displayed.");
reply.Speak = "This is the text that will be spoken.";
reply.InputHint = InputHints.ExpectingInput;
```

在 Cortana 中顯示卡片

除了語音回應，Cortana 也可以顯示卡片附件。Cortana 可支援下列複合式資訊卡 (Rich Card)：

卡片類型	說明
HeroCard	通常包含單一大型影像、一或多個按鈕和文字的卡片。
ThumbnailCard	通常包含單一縮圖影像、一或多個按鈕和文字的卡片。
ReceiptCard	讓 Bot 向使用者提供收據的卡片。通常包含收據上的項目清單 (稅金和總金額資訊) 和其他文字。
SignInCard	可讓 Bot 要求使用者登入的卡片。它通常包含文字，以及一或多個按鈕，使用者可以按一下按鈕來起始登入流程。

請參閱[卡片設計最佳做法](#)，以查看這些卡片在 Cortana 內的樣貌。如需如何在 Bot 中使用複合式資訊卡 (Rich Card) 的範例，請參閱[將多媒體卡片附件新增到郵件](#)。

範例：RollerSkill

下列各節中的程式碼來自擲骰子的範例 Cortana 技能。請從[BotBuilder-Samples 存放庫 \(英文\)](#) 下載 Bot 的完整程式碼。

您可以向 Cortana 說出技能的[叫用名稱](#)以便叫用技能。對於骰子機技能，在您將 Bot 新增至 Cortana 通道並將其註冊為 Cortana 技能後，可以告訴 Cortana「啟動骰子機」或「讓骰子機擲骰子」以進行叫用。

探索程式碼

為了叫用適當的對話方塊，`RootDispatchDialog.cs` 中定義的活動處理常式會使用規則運算式，來比對使用者的輸入。例如，如果使用者說了像是「我想要擲骰子」這樣的語句，就會觸發下列範例中的處理常式。同義字包含在規則運算式中，因此類似的語句將會觸發對話方塊。

```
[RegexPattern("(roll|role|throw|shoot).*(dice|die|dye|bones)")]
[RegexPattern("new game")]
[ScorableGroup(1)]
public async Task NewGame(IDialogContext context, IActivity activity)
{
    context.Call(new CreateGameDialog(), AfterGameCreated);
}
```

`CreateGameDialog` 對話方塊會設定自訂遊戲讓 Bot 玩。它會使用 `PromptDialog` 來詢問使用者想要讓骰子有幾個面，以及應該擲幾個骰子。請注意，用來初始化提示的 `PromptOptions` 物件包含 `speak` 屬性，可供語音版本的提示使用。

```

[Serializable]
public class CreateGameDialog : IDialog<GameData>
{
    public async Task StartAsync(IDialogContext context)
    {
        context.UserData.SetValue<GameData>(Utils.GameDataKey, new GameData());

        var descriptions = new List<string>() { "4 Sides", "6 Sides", "8 Sides", "10 Sides", "12 Sides",
"20 Sides" };
        var choices = new Dictionary<string, IReadOnlyList<string>>()
        {
            { "4", new List<string> { "four", "for", "4 sided", "4 sides" } },
            { "6", new List<string> { "six", "sex", "6 sided", "6 sides" } },
            { "8", new List<string> { "eight", "8 sided", "8 sides" } },
            { "10", new List<string> { "ten", "10 sided", "10 sides" } },
            { "12", new List<string> { "twelve", "12 sided", "12 sides" } },
            { "20", new List<string> { "twenty", "20 sided", "20 sides" } }
        };

        var promptOptions = new PromptOptions<string>(
            Resources.ChooseSides,
            choices: choices,
            descriptions: descriptions,
            speak: SSMLHelper.Speak(Utils.RandomPick(Resources.ChooseSidesSSML))); // spoken prompt

        PromptDialog.Choice(context, this.DiceChoiceReceivedAsync, promptOptions);
    }

    private async Task DiceChoiceReceivedAsync(IDialogContext context, IAwaitable<string> result)
    {
        GameData game;
        if (context.UserData.TryGetValue<GameData>(Utils.GameDataKey, out game))
        {
            int sides;
            if (int.TryParse(await result, out sides))
            {
                game.Sides = sides;
                context.UserData.SetValue<GameData>(Utils.GameDataKey, game);
            }

            var promptText = string.Format(Resources.ChooseCount, sides);

            var promptOption = new PromptOptions<long>(promptText, choices: null, speak:
SSMLHelper.Speak(Utils.RandomPick(Resources.ChooseCountSSML)));

            var prompt = new PromptDialog.PromptInt64(promptOption);
            context.Call<long>(prompt, this.DiceNumberReceivedAsync);
        }
    }

    private async Task DiceNumberReceivedAsync(IDialogContext context, IAwaitable<long> result)
    {
        GameData game;
        if (context.UserData.TryGetValue<GameData>(Utils.GameDataKey, out game))
        {
            game.Count = await result;
            context.UserData.SetValue<GameData>(Utils.GameDataKey, game);
        }

        context.Done(game);
    }
}

```

PlayGameDialog 會藉由在 HeroCard 中顯示結果、並使用 Speak 方法建置要說出的訊息，來呈現這兩個結果。

[Serializable]

```

public class PlayGameDialog : IDialog<object>
{
    private const string RollAgainOptionValue = "roll again";

    private const string NewGameOptionValue = "new game";

    private GameData gameData;

    public PlayGameDialog(GameData gameData)
    {
        this.gameData = gameData;
    }

    public async Task StartAsync(IDialogContext context)
    {
        if (this.gameData == null)
        {
            if (!context.UserData.TryGetValue<GameData>(Utils.GameDataKey, out this.gameData))
            {
                // User started session with "roll again" so let's just send them to
                // the 'CreateGameDialog'
                context.Done<object>(null);
            }
        }

        int total = 0;
        var randomGenerator = new Random();
        var rolls = new List<int>();

        // Generate Rolls
        for (int i = 0; i < this.gameData.Count; i++)
        {
            var roll = randomGenerator.Next(1, this.gameData.Sides);
            total += roll;
            rolls.Add(roll);
        }

        // Format rolls results
        var result = string.Join(" . ", rolls.ToArray());
        bool multiLine = rolls.Count > 5;

        var card = new HeroCard()
        {
            Subtitle = string.Format(
                this.gameData.Count > 1 ? Resources.CardSubtitlePlural : Resources.CardSubtitleSingular,
                this.gameData.Count,
                this.gameData.Sides),
            Buttons = new List<CardAction>()
            {
                new CardAction(ActionTypes.ImBack, "Roll Again", value: RollAgainOptionValue),
                new CardAction(ActionTypes.ImBack, "New Game", value: NewGameOptionValue)
            }
        };

        if (multiLine)
        {
            card.Text = result;
        }
        else
        {
            card.Title = result;
        }

        var message = context.MakeMessage();
        message.Attachments = new List<Attachment>()
        {
            card.ToAttachment()
        };
    }
}

```

```

// Determine bots reaction for speech purposes
string reaction = "normal";

var min = this.gameData.Count;
var max = this.gameData.Count * this.gameData.Sides;
var score = total / max;
if (score == 1)
{
    reaction = "Best";
}
else if (score == 0)
{
    reaction = "Worst";
}
else if (score <= 0.3)
{
    reaction = "Bad";
}
else if (score >= 0.8)
{
    reaction = "Good";
}

// Check for special craps rolls
if (this.gameData.Type == "Craps")
{
    switch (total)
    {
        case 2:
        case 3:
        case 12:
            reaction = "CrapsLose";
            break;
        case 7:
            reaction = "CrapsSeven";
            break;
        case 11:
            reaction = "CrapsEleven";
            break;
        default:
            reaction = "CrapsRetry";
            break;
    }
}

// Build up spoken response
var spoken = string.Empty;
if (this.gameData.Turns == 0)
{
    spoken +=
    Utils.RandomPick(Resources.ResourceManager.GetString($"Start{this.gameData.Type}GameSSML"));
}

spoken += Utils.RandomPick(Resources.ResourceManager.GetString($"{reaction}RollReactionSSML"));

message.Speak = SSMLHelper.Speak(spoken);

// Increment number of turns and store game to roll again
this.gameData.Turns++;
context.UserData.SetValue<GameData>(Utils.GameDataKey, this.gameData);

// Send card and bots reaction to user.
message.InputHint = InputHints.AcceptingInput;
await context.PostAsync(message);

context.Done<object>(null);
}
}

```

後續步驟

如果您的 Bot 是在本機執行或部署在雲端中，可以從 Cortana 叫用。請參閱[測試 Cortana 技能](#)，以了解試用 Cortana 技能所需的步驟。

其他資源

- [Cortana 技能套件](#)
- [將語音新增至訊息](#)
- [SSML 參考](#)
- [Cortana 的語音設計最佳做法](#)
- [Cortana 的卡片設計最佳做法](#)
- [Cortana 開發人員中心](#)
- [Cortana 的測試和偵錯最佳做法](#)
- [適用於 .NET 的 Bot Framework SDK 參考](#)

使用 Skype 進行音訊通話

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

如果您要建置適用於 Skype 的 Bot，您的 Bot 可以透過語音通話來與使用者通訊。當使用者不想或無法藉由輸入、點選或按一下來提供輸入時，語音通話就很實用。

除了語音通話之外，Bot 也可以支援其他使用者控制項 (例如複合式資訊卡或文字)，或者僅透過語音通話進行通訊。

支援音訊通話的 Bot 架構與一般 Bot 的架構非常類似。下列程式碼範例說明如何使用適用於 .NET 的 Bot Framework SDK 啟用，透過 Skype 進行音訊通話的支援。

啟用音訊通話的支援

若要讓 Bot 支援音訊通話，請定義 `CallingController`。

```
[BotAuthentication]
[RoutePrefix("api/calling")]
public class CallingController : ApiController
{
    public CallingController() : base()
    {
        CallingConversation.RegisterCallingBot(callingBotService => new IVRBot(callingBotService));
    }

    [Route("callback")]
    public async Task<HttpResponseMessage> ProcessCallingEventAsync()
    {
        return await CallingConversation.SendAsync(this.Request, CallRequestType.CallingEvent);
    }

    [Route("call")]
    public async Task<HttpResponseMessage> ProcessIncomingCallAsync()
    {
        return await CallingConversation.SendAsync(this.Request, CallRequestType.IncomingCall);
    }
}
```

NOTE

除了支援音訊通話的 `CallingController` 以外，Bot 也可包含 `MessagesController` 以支援訊息。同時使用這兩個選項，可讓使用者選擇其偏好的方式與 Bot 互動。

接聽來電

每當使用者用 Skype 起始與此 Bot 的通話時，就會執行 `ProcessIncomingCallAsync` 工作。建構函式會註冊 `IVRBot` 類別，其中包含為 `incomingCallEvent` 預先定義的處理常式。

工作流程中的第一個動作應先決定 Bot 要接聽還是拒絕來電。此工作流程會指示 Bot 接聽來電，然後播放歡迎訊息。

```

private Task OnIncomingCallReceived(IncomingCallEvent incomingCallEvent)
{
    this.callStateMap[incomingCallEvent.IncomingCall.Id] = new
    CallState(incomingCallEvent.IncomingCall.Participants);

    incomingCallEvent.ResultingWorkflow.Actions = new List<ActionBase>
    {
        new Answer { OperationId = Guid.NewGuid().ToString() },
        GetPromptForText(WelcomeMessage)
    };

    return Task.FromResult(true);
}

```

在 Bot 接聽後

如果 Bot 接聽電話，工作流程中指定的後續動作會指示 **Skype Bot** 通話平台播放提示，錄製音訊、辨識語音，或從撥號鍵台收集數字。工作流程的最後一個動作可能是結束通話。

下列程式碼範例會定義，將在歡迎訊息完成後設定功能表的處理常式。

```

private Task OnPlayPromptCompleted(PlayPromptOutcomeEvent playPromptOutcomeEvent)
{
    var callState = this.callStateMap[playPromptOutcomeEvent.ConversationResult.Id];
    SetupInitialMenu(playPromptOutcomeEvent.ResultingWorkflow);
    return Task.FromResult(true);
}

```

CreateIvrOptions 方法定義會要向使用者顯示的功能表。

```

private static Recognize CreateIvrOptions(string textToBeRead, int numberOptions, bool includeBack)
{
    if (numberOptions > 9)
    {
        throw new Exception("too many options specified");
    }

    var choices = new List<RecognitionOption>();

    for (int i = 1; i <= numberOptions; i++)
    {
        choices.Add(new RecognitionOption { Name = Convert.ToString(i), DtmfVariation = (char)('0' + i) });
    }

    if (includeBack)
    {
        choices.Add(new RecognitionOption { Name = "#", DtmfVariation = '#' });
    }

    var recognize = new Recognize
    {
        OperationId = Guid.NewGuid().ToString(),
        PlayPrompt = GetPromptForText(textToBeRead),
        BargeInAllowed = true,
        Choices = choices
    };

    return recognize;
}

```

RecognitionOption 類別會定義回話和相對應的雙音多頻訊號 (DTMF) 變化。DTMF 可讓使用者以撥號鍵台輸入對

應的數字來應答，而無須實際發話。

`OnRecognizeCompleted` 方法會處理使用者的選取項目，而輸入參數 `recognizeOutcomeEvent` 會包含使用者選取項目的值。

```
private Task OnRecognizeCompleted(RecognizeOutcomeEvent recognizeOutcomeEvent)
{
    var callState = this.callStateMap[recognizeOutcomeEvent.ConversationResult.Id];
    ProcessMainMenuSelection(recognizeOutcomeEvent, callState);
    return Task.FromResult(true);
}
```

支援自然語言

Bot 也可以設計成支援自然語言回應。**Bing 語音 API** 可讓 Bot 辨識使用者實際回答的文字。

```
private async Task OnRecordCompleted(RecordOutcomeEvent recordOutcomeEvent)
{
    recordOutcomeEvent.ResultingWorkflow.Actions = new List<ActionBase>
    {
        GetPromptForText(EndingMessage),
        new Hangup { OperationId = Guid.NewGuid().ToString() }
    };

    // Convert the audio to text
    if (recordOutcomeEvent.RecordOutcome.Outcome == Outcome.Success)
    {
        var record = await recordOutcomeEvent.RecordedContent;
        string text = await this.GetTextFromAudioAsync(record);

        var callState = this.callStateMap[recordOutcomeEvent.ConversationResult.Id];

        await this.SendSTTResultToUser("We detected the following audio: " + text, callState.Participants);
    }

    recordOutcomeEvent.ResultingWorkflow.Links = null;
    this.callStateMap.Remove(recordOutcomeEvent.ConversationResult.Id);
}
```

範例程式碼

如需完整範例以了解如何使用適用於 .NET 的 Bot Framework SDK 支援，透過 Skype 的音訊通話，請參閱 GitHub 中的 [Skype 通話 Bot 範例](#)。

其他資源

- [適用於 .NET 的 Bot Framework SDK 參考](#)
- [Skype 通話 Bot 範例 \(GitHub\)](#)

管理狀態資料

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

Bot Builder Framework 可讓您的 Bot 儲存和擷取與使用者、對話或特定對話內容中的特定使用者相關聯的狀態資料。狀態資料有許多用途，例如判斷上一個對話最後的內容，或是單純地以名字問候回來的使用者。如果您儲存使用者的喜好設定，下次聊天時就可以使用該資訊來自訂對話。例如，您可能會在有和使用者感興趣之主題相關的新聞文章，或有可用約會時警示使用者。

針對測試和原型設計目的，您可以使用 Bot Builder Framework 的記憶體內部資料儲存體。針對生產環境的 Bot，您可以實作自己的儲存體接器，或使用其中一個 Azure 擴充功能。Azure 擴充功能可讓您將 Bot 的狀態資料儲存於表格儲存體、CosmosDB 或 SQL。本文將示範如何使用記憶體內部儲存體接器來儲存 Bot 的狀態資料。

IMPORTANT

建議您不要將 Bot Framework 狀態服務 API 用於生產環境，因為未來版本可能會淘汰它。建議您更新 Bot 程式碼，以針對測試目的使用記憶體內部儲存體接器，或針對生產環境 Bot 使用其中一個 **Azure 擴充功能**。

記憶體內部的資料儲存體

記憶體內部的資料儲存體僅供測試之用。這是可變更的臨時儲存體。每次 Bot 重新啟動時，就會清除資料。若要將記憶體內部儲存體運用於測試用途，請務必：

安裝下列 NuGet 封裝：

- Microsoft.Bot.Builder.Azure
- Autofac.WebApi2

在 **Application_Start** 方法中，建立新記憶體內部儲存體的執行個體，並註冊新的資料存放區：

```
// Global.asax file

var store = new InMemoryDataStore();

Conversation.UpdateContainer(
    builder =>
{
    builder.Register(c => store)
        .Keyed<IBotDataStore<BotData>>(AzureModule.Key_DataStore)
        .AsSelf()
        .SingleInstance();

    builder.Register(c => new CachingBotDataStore(store,
        CachingBotDataStoreConsistencyPolicy
        .ETagBasedConsistency))
        .As<IBotDataStore<BotData>>()
        .AsSelf()
        .InstancePerLifetimeScope();
});

GlobalConfiguration.Configure(WebApiConfig.Register);
```

您可以使用這個方法來設定自己的自訂資料存放區，或使用任一項 *Azure 擴充功能*。

管理自訂資料儲存體

在生產環境中基於效能和安全性理由，您可能要實作自己的資料儲存體，或考量實作下列其中一個資料儲存體選項：

1. [使用 Cosmos DB 管理狀態資料](#)
2. [藉由資料表儲存體管理狀態資料](#)

透過其中任一 *Azure 擴充功能*選項，.Bot Framework SDK for .NET 的資料設定和留存機制會與記憶體內部資料存放區的機制相同。

Bot 狀態方法

下表列出可用於管理狀態資料的方法。

方法	目標範圍	目標
<code>GetUserData</code>	使用者	取得之前儲存供指定通道使用者使用的狀態資料
<code>GetConversationData</code>	交談	取得之前針對指定通道交談儲存的狀態資料
<code>GetPrivateConversationData</code>	使用者和交談	取得之前儲存供指定通道交談內使用者使用的狀態資料
<code>SetUserData</code>	使用者	針對指定通道的使用者儲存的狀態資料

方法	目標範圍	目標
<code>SetConversationData</code>	交談	針對指定通道的交談儲存的狀態資料。 注意：由於 <code>DeleteStateForUser</code> 方法並不會刪除已使用 <code>SetConversationData</code> 方法所儲存的資料，因此您「切勿」使用這個方法來儲存使用者個人識別資訊 (PII)。
<code>SetPrivateConversationData</code>	使用者和交談	針對指定通道的交談中之使用者儲存的狀態資料
<code>DeleteStateForUser</code>	使用者	刪除之前已使用 <code>SetUserData</code> 方法或 <code>SetPrivateConversationData</code> 方法所儲存的使用者狀態資料。 注意：Bot 接收 <code>deleteUserData</code> 或 <code>contactRelationUpdate</code> 類型的活動，指出 Bot 已從使用者的連絡人清單中遭到移除時，Bot 就應呼叫這個方法。

如果您的 Bot 使用 "Set...Data" 方法來儲存狀態資料，日後 Bot 在相同環境中收到的訊息就會包含該資料，這些資料可以使用相應的 "Get...Data" 方法來存取。

適用於管理狀態資料的實用屬性

每個活動物件都包含您要用來管理狀態資料的屬性。

屬性	說明	使用案例
<code>From</code>	可唯一識別通道上的使用者	儲存和擷取與使用者相關聯的狀態資料
<code>Conversation</code>	可唯一識別交談	儲存和擷取與交談相關聯的狀態資料
<code>From</code> 和 <code>Conversation</code>	可唯一識別使用者和交談	儲存和擷取與特定交談內容中與特定使用者相關聯的狀態資料

NOTE

即便您選擇將狀態資料儲存在自己的資料庫，而非選擇使用 Bot Framework 狀態資料存放區，您也可以使用這些屬性值做為索引鍵。

處理並行問題

如果 Bot 的其他執行個體變更了資料，當 Bot 嘗試儲存狀態資料時，可能會收到以下的 HTTP 狀態碼錯誤回應：**412 前置條件失敗**。您可以設計自己的 Bot 來說明這類情形，如下列程式碼範例所示。

```
var builder = new ContainerBuilder();
builder
    .Register(c => new CachingBotDataStore(c.Resolve<ConnectorStore>(),
CachingBotDataStoreConsistencyPolicy.LastWriteWins)
    .As<IBotDataStore<BotData>>()
    .AsSelf()
    .InstancePerLifetimeScope();
builder.Update(Conversation.Container);
```

其他資源

- [Bot Framework 疑難排解指南](#)
- [適用於 .NET 的 Bot Framework SDK 參考](#)

使用適用於 .NET 的 Azure Cosmos DB 管理自訂狀態資料

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

在本文中，您將實作 Azure Cosmos DB 來儲存和管理 Bot 的狀態資料。Bot 所使用的預設連接器狀態服務不適用於生產環境。您應使用 GitHub 上提供的 [Azure 延伸模組](#) (英文)，或使用您選擇的資料儲存體平台來實作自訂狀態用戶端。以下列舉應使用自訂狀態儲存體的若干原因：

- 可獲得更高的狀態 API 輸送量 (更充分掌控效能)
- 可降低地理分佈的延遲
- 可控制資料儲存位置
- 可存取實際的狀態資料
- 儲存超過 32 KB 的資料

必要條件

您需要：

- [Microsoft Azure 帳戶](#)
- [Visual Studio 2015 或更新版本](#)
- [Bot Builder Azure NuGet 套件](#)
- [Autofac Web Api2 NuGet 套件](#) (英文)
- [Bot Framework 模擬器](#) (英文)

建立 Azure 帳戶

如果您沒有 Azure 帳戶，請按一下[這裡](#)註冊免費帳戶。

設定 Azure Cosmos DB 資料庫

1. 在您登入 Azure 入口網站之後，按一下 [新增] 以建立新的 Azure Cosmos DB 資料庫。
2. 按一下 [資料庫]。
3. 找到 **Azure Cosmos DB**，然後按一下 [建立]。
4. 填寫欄位。如果是 [API] 欄位，請選取 [**SQL (DocumentDB)**]。填妥所有欄位後，按一下畫面底部的 [建立] 按鈕來部署新的資料庫。
5. 新資料庫部署完畢後，接著瀏覽至新資料庫。按一下 [存取金鑰] 以尋找金鑰和連接字串。您的 Bot 會使用此資訊來呼叫儲存體服務以儲存狀態資料。

安裝 NuGet 套件

1. 開啟現有的 C# Bot 專案，或使用 Visual Studio 中的 Bot 範本建立新專案。
2. 安裝下列 NuGet 封裝：
 - Microsoft.Bot.Builder.Azure

- Autofac.WebApi2

新增連接字串

在 Web.config 檔案中新增下列項目：

```
<add key="DocumentDbUrl" value="Your DocumentDB URI"/>
<add key="DocumentDbKey" value="Your DocumentDB Key"/>
```

請將值替換為您的 URI 和您在 Azure Cosmos DB 中的主索引鍵。儲存 Web.config 檔案。

修改您的 Bot 程式碼

若要使用 **Azure Cosmos DB** 儲存體，將下列程式碼行新增至 **Application_Start()** 方法中 Bot 的 **Global.asax.cs** 檔案。

```
using System;
using Autofac;
using System.Web.Http;
using System.Configuration;
using Microsoft.Bot.Connector;
using Microsoft.Bot.Builder.Azure;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.Dialogs.Internals;

namespace SampleApp
{
    public class WebApiApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            GlobalConfiguration.Configure(WebApiConfig.Register);
            var uri = new Uri(ConfigurationManager.AppSettings["DocumentDbUrl"]);
            var key = ConfigurationManager.AppSettings["DocumentDbKey"];
            var store = new DocumentDbBotDataStore(uri, key);

            Conversation.UpdateContainer(
                builder =>
                {
                    builder.Register(c => store)
                        .Keyed<IBotDataStore<BotData>>(AzureModule.Key_DataStore)
                        .AsSelf()
                        .SingleInstance();

                    builder.Register(c => new CachingBotDataStore(store,
CachingBotDataStoreConsistencyPolicy.ETagBasedConsistency))
                        .As<IBotDataStore<BotData>>()
                        .AsSelf()
                        .InstancePerLifetimeScope();
                });
        }
    }
}
```

儲存 global.asax.cs 檔案。現在您已準備好使用模擬器測試 Bot。

執行 Bot 應用程式

在 Visual Studio 中執行 Bot，您所新增的程式碼就會在 Azure 中建立自訂的 **botdata** 表格。

將 Bot 連線至模擬器

此時，Bot 正在本機執行。接下來，請啟動模擬器，然後在模擬器中連線至您的 Bot：

1. 將 `http://localhost:port-number/api/messages` 輸入網址列，其中連接埠號碼必須符合應用程式執行所在的瀏覽器中，所顯示的連接埠號碼。您目前可以將 **Microsoft 應用程式識別碼** 和 **Microsoft 應用程式密碼** 欄位留白。稍後[註冊 Bot](#)時，您會取得這些資訊。
2. 按一下 [連接]。
3. 在模擬器中輸入一些訊息以測試您的 Bot。

在 Azure 入口網站上檢視狀態資料

若要檢視狀態資料，請登入 Azure 入口網站並瀏覽至您的資料庫。按一下 [資料總管 (預覽)] 來確認正在儲存的 Bot 狀態資訊。

後續步驟

在本文中，您可以使用 Cosmos DB 來儲存及管理您的 Bot 資料。接下來，了解如何使用對話來模型化對話流程。

[管理對話流程](#)

其他資源

如果您不熟悉上述程式碼中所使用的「控制項反轉」容器和「相依性插入」模式，請造訪 [Autofac](#) 網站以取得資訊。

您也可以從 GitHub 下載[範例](#)，以深入了解如何使用 Cosmos DB 來管理狀態。

使用適用於 .NET 的 Azure 表格儲存體來管理自訂狀態資料

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

在本文中，您將實作 Azure 表格儲存體來儲存和管理 Bot 的狀態資料。Bot 所使用的預設「連接器狀態服務」不適用於生產環境。您應使用 GitHub 上提供的 [Azure 延伸模組](#) (英文)，或使用您選擇的資料儲存體平台來實作自訂狀態用戶端。以下列舉應使用自訂狀態儲存體的若干原因：

- 可獲得更高的狀態 API 輸送量 (更充分掌控效能)
- 可降低地理分佈的延遲
- 可控制資料儲存位置
- 可存取實際的狀態資料
- 儲存超過 32 KB 的資料

必要條件

您需要：

- [Microsoft Azure 帳戶](#)
- [Visual Studio 2015 或更新版本](#)
- [Bot Builder Azure NuGet 套件](#)
- [Autofac Web Api2 NuGet 套件](#) (英文)
- [Bot Framework 模擬器](#) (英文)
- [Azure 儲存體總管](#)

建立 Azure 帳戶

如果您沒有 Azure 帳戶，請按一下[這裡](#)註冊免費帳戶。

設定 Azure 資料表儲存體服務

1. 在您登入 Azure 入口網站之後，請按一下 [新增] 以建立新的 Azure 表格儲存體服務。
2. 搜尋會實作 Azure 表格的儲存體帳戶。
3. 填寫欄位，按一下畫面底部的 [建立] 按鈕來部署新的儲存體服務。部署新的儲存體服務之後，它會顯示可用的功能和選項。
4. 選取左側的 [存取金鑰] 索引標籤，並複製連接字串以供稍後使用。您的 Bot 會使用此連接字串，來呼叫儲存體服務以儲存狀態資料。

安裝 NuGet 套件

1. 開啟現有的 C# Bot 專案，或使用 Visual Studio 中的 C# Bot 範本建立新專案。
2. 安裝下列 NuGet 封裝：
 - Microsoft.Bot.Builder.Azure

- Autofac.WebApi2

新增連接字串

在 Web.config 檔案中新增下列項目：

```
<connectionStrings>
  <add name="StorageConnectionString"
    connectionString="YourConnectionString"/>
</connectionStrings>
```

將 "YourConnectionString" 取代為您稍早儲存的 Azure 表格儲存體連接字串。儲存 Web.config 檔案。

修改您的 Bot 程式碼

在 Global.asax.cs 檔案中新增下列 `using` 陳述式：

```
using Autofac;
using System.Configuration;
using Microsoft.Bot.Connector;
using Microsoft.Bot.Builder.Azure;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.Dialogs.Internals;
```

在 `Application_Start()` 方法中建立 `TableBotDataStore` 類別的執行個體。`TableBotDataStore` 類別會實作 `IBotDataStore<BotData>` 介面。`IBotDataStore` 介面可讓您覆寫預設的連接器狀態服務連線。

```
var store = new
TableBotDataStore(ConfigurationManager.ConnectionStrings["StorageConnectionString"].ConnectionString);
```

註冊服務，如下所示：

```
Conversation.UpdateContainer(
    builder =>
{
    builder.Register(c => store)
        .Keyed<IBotDataStore<BotData>>(AzureModule.Key_DataStore)
        .AsSelf()
        .SingleInstance();

    builder.Register(c => new CachingBotDataStore(store,
        CachingBotDataStoreConsistencyPolicy
        .ETagBasedConsistency))
        .As<IBotDataStore<BotData>>()
        .AsSelf()
        .InstancePerLifetimeScope();

});
```

儲存 Global.asax.cs 檔案。

執行 Bot 應用程式

在 Visual Studio 中執行 Bot，您所新增的程式碼就會在 Azure 中建立自訂的 **botdata** 表格。

將 Bot 連線至模擬器

此時，Bot 正在本機執行。接下來，請啟動模擬器，然後在模擬器中連線至您的 Bot：

1. 將 `http://localhost:port-number/api/messages` 輸入網址列，其中連接埠號碼必須符合應用程式執行所在的瀏覽器中，所顯示的連接埠號碼。您目前可以將 **Microsoft 應用程式識別碼** 和 **Microsoft 應用程式密碼** 欄位留白。稍後[註冊 Bot](#)時，您會取得這些資訊。
2. 按一下 [連接]。
3. 在模擬器中輸入一些訊息以測試您的 Bot。

在 Azure 表格儲存體中檢視資料

若要檢視狀態資料，請開啟**儲存體總管**並使用 Azure 入口網站認證來連線至 Azure，或使用儲存體名稱和儲存體金鑰直接連線至表格，然後瀏覽至您的表格名稱。

後續步驟

在本文中，您會實作 Azure 表格儲存體來儲存及管理 Bot 的資料。接下來，了解如何使用對話方塊將對話流程模型化。

[管理對話流程](#)

其他資源

如果您不熟悉上述程式碼中所使用的「控制項反轉」容器和「相依性插入」模式，請移至 [Autofac](#) (英文) 網站以取得資訊。

您也可以從 GitHub 下載完整的 [Azure 表格儲存體](#) (英文) 範例。

使用 LUIS 辨識意圖和實體

2019/2/28 • [Edit Online](#)

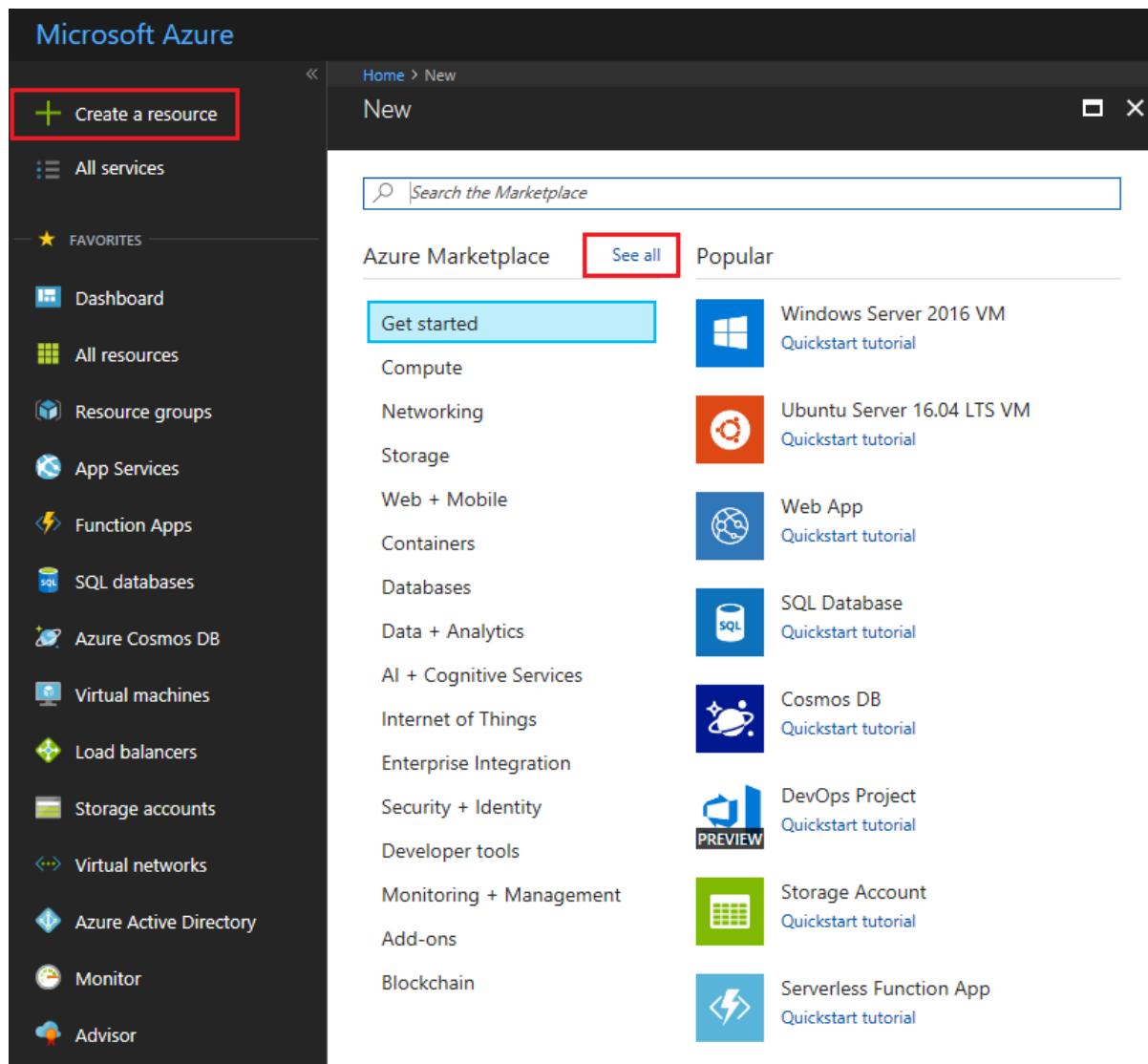
NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

本文使用要用來記筆記的 Bot 範例，來示範 Language Understanding (LUIS) 如何協助您的 Bot 適當地回應自然語言輸入。Bot 會藉由識別使用者的意圖，來偵測他們想要做什麼。此意圖是從語音或文字輸入，或是語句來判斷的。此意圖會將語句對應到 Bot 所採取的動作。例如，記筆記的 Bot 會辨識 `Notes.Create` 意圖，以叫用可用來建立筆記的功能。Bot 可能也需要擷取實體，其為語句中的重要單字。在記筆記的 Bot 範例中，`Notes.Title` 實體會識別每個筆記的標題。

使用 Bot 服務來建立 Language Understanding Bot

- 在 [Azure 入口網站](#) 中，選取功能表刀鋒視窗中的 [建立新資源]，然後按一下 [查看全部]。



- 在搜尋方塊中，搜尋 **Web 應用程式 Bot**。

Home > New > Marketplace > Everything

Marketplace ✎ X Everything

Everything Filter

Search Everything

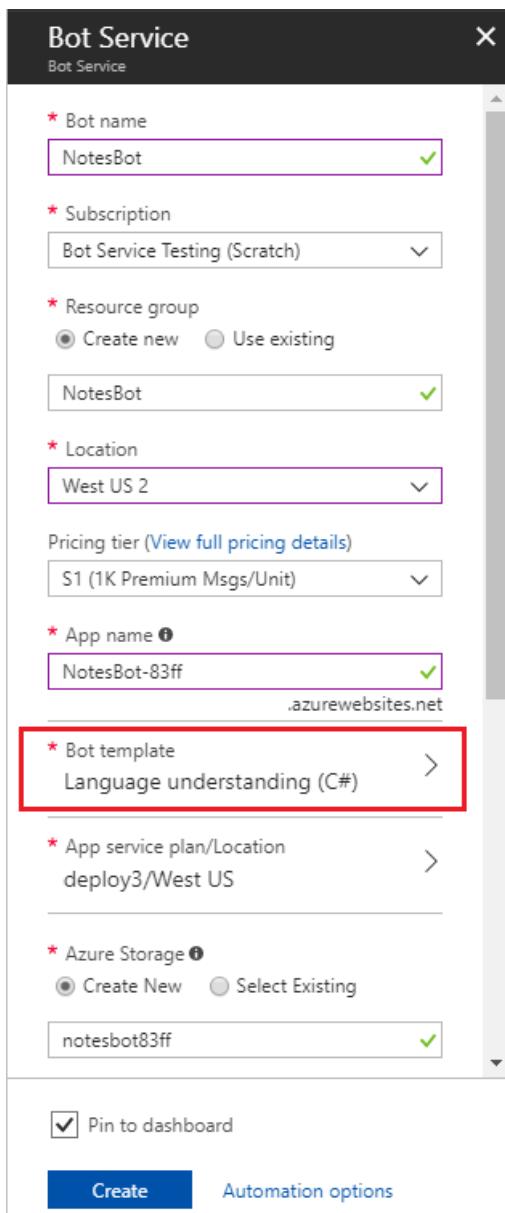
Compute	
Networking	
Storage	
Web + Mobile	
Databases	
Data + Analytics	
AI + Cognitive Services	

 Functions Bot Microsoft

 Bot Channels Registration Microsoft

 Web App Bot Microsoft

3. 在 [Bot 服務] 刀鋒視窗中提供必要資訊，然後按一下 [建立]。這會建立 Bot 服務和 LUIS 應用程式，並將其部署到 Azure。
 - 將 [應用程式名稱] 設定為您 Bot 的名稱。將 Bot 部署到雲端時，此名稱會用來作為子網域 (例如 mynotesbot.azurewebsites.net)。此名稱也會用來作為與您 Bot 相關聯的 LUIS 應用程式名稱。複製它以在稍後用來尋找與 Bot 相關聯的 LUIS 應用程式。
 - 選取訂用帳戶、[資源群組](#)、App Service 方案，以及[位置](#)。
 - 針對 [Bot 範本] 欄位，選取 [Language Understanding (C#)] 範本。



- 選取方塊以確認服務條款。

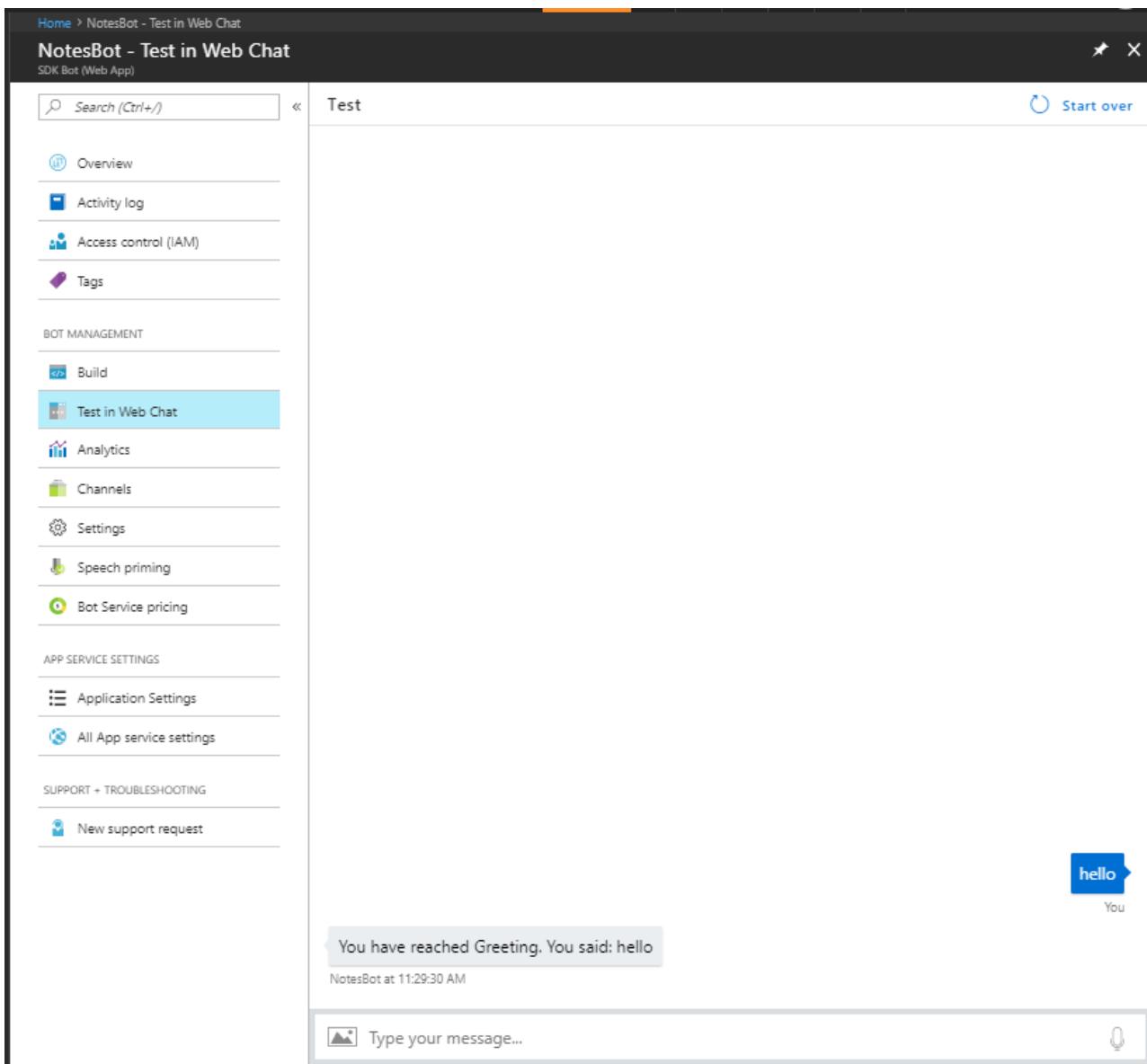
4. 確認已部署 Bot 服務。

- 按一下 [通知] (位於 Azure 入口網站頂端邊緣的鈴鐺圖示)。通知會從 [部署已開始] 變更為 [部署成功]。
- 在通知變更為 [部署成功] 之後，按一下該通知上的 [前往資源]。

測試聊天機器人

勾選 [通知] 來確認已部署 Bot。通知會從 [部署進行中] 變更為 [部署成功]。按一下 [前往資源] 按鈕以開啟 Bot 的資源刀鋒視窗。

註冊 Bot 後，按一下 [在網路聊天中測試] 以開啟 [網路聊天] 窗格。在網路聊天中輸入 "hello"。



Bot 會說出 "You have reached Greeting. You said: hello" 來作為回應。這確認了 Bot 已收到您的訊息，並已將其傳遞給它所建立的預設 LUIS 應用程式。此預設 LUIS 應用程式偵測到 Greeting 意圖。

修改 LUIS 應用程式

使用您用來登入 Azure 的相同帳戶登入 <https://www.luis.ai>。按一下 [My apps] (我的應用程式)。在應用程式清單中，尋找開頭為當您建立 Bot 服務時，於 [Bot 服務] 刀鋒視窗的 [應用程式名稱] 中指定之名稱的應用程式。

LUIS 應用程式會以下列 4 個意圖作為開頭：Cancel、Greeting、Help 及 None。

下列步驟會新增 Note.Create、Note.ReadAloud 和 Note.Delete 意圖：

1. 在頁面左下角按一下 [預先建置的網域]。尋找 **Note** 網域，然後按一下 [新增網域]。
2. 本教學課程不會使用 **Note** 預先建置的網域中包含的所有意圖。在 [意圖] 頁面上，按一下以下每一個意圖名稱，然後按一下 [刪除意圖] 按鈕。
 - Note.ShowNext
 - Note.DeleteNoteItem
 - Note.Confirm
 - Note.Clear
 - Note.CheckOffItem
 - Note.AddToNote

以下為只應保留在 LUIS 應用程式中的意圖：

- Note.ReadAloud
- Note.Create
- Note.Delete
- None
- 說明
- Greeting
- 取消

Name	Utterances
Note.ReadAloud	10
Note.Delete	11
Note.Create	17
None	13
Help	13
Greeting	13
Cancel	14

3. 按一下右上角的 [定型] 按鈕，來將您的應用程式定型。

4. 按一下上方導覽列中的 [發佈]，以開啟 [發佈] 頁面。按一下 [發佈到生產位置] 按鈕。成功發佈之後，複製 [發佈應用程式] 頁面上 [端點] 欄中所顯示的 URL，位於以資源名稱 Starter_Key 開頭的列中。儲存此 URL，以便稍後在您的 Bot 程式碼中使用。URL 的格式類似此範例：

```
https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/xxxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx?subscription-key=xxxxxxxxxxxxxxxxxxxxxxxxxxxx&timezoneOffset=0&verbose=true&q=
```

修改 Bot 程式碼

按一下 [建置]，然後按一下 [開啟線上程式碼編輯器]。

Search (Ctrl+ /)

- [!\[\]\(1a6611e83f73dd4bfdad1be575d1e100_img.jpg\) Overview](#)
- [!\[\]\(b27433aa52a692e2b66682751668f705_img.jpg\) Activity log](#)
- [!\[\]\(9499b494aa947912bd2a7501f226ce80_img.jpg\) Access control \(IAM\)](#)
- [!\[\]\(fe84bc7126d5feb498a162f6089f28d6_img.jpg\) Tags](#)

BOT MANAGEMENT

- [!\[\]\(2084c85b60cdef5e685d24e201ef3fb1_img.jpg\) Build](#)
- [!\[\]\(c7a2b619b496b4e366a481509267c2aa_img.jpg\) Test in Web Chat](#)
- [!\[\]\(d3585aca55e99101c9241b8ed69eff98_img.jpg\) Analytics](#)
- [!\[\]\(cb28f61b122e48c6f9a07c687aa9c008_img.jpg\) Channels](#)
- [!\[\]\(21edc2f5dfc68a6bc55060510f25772c_img.jpg\) Settings](#)
- [!\[\]\(0d6b7630f1cf8d1e1d0b84c4f36f7e6d_img.jpg\) Speech priming](#)
- [!\[\]\(2616db9f51093b1b69e60f816ce6394c_img.jpg\) Bot Service pricing](#)

Choose how to work with your code

Online code editor ⓘ

Make quick changes to your bot code online, run build.cmd in the editor console, and see your changes instantly.

[Open online code editor](#)

Download source code ⓘ

Download your source code and develop locally using your favorite IDE. You can publish your code back to the bot when ready.

[Download zip file](#)

Continuous deployment from source control ⓘ

Step 1: [Download zip file](#)

Step 2: Create a folder/repo for the source files in your preferred service

Step 3: [Configure continuous deployment](#)

在程式碼編輯器中，開啟 `BasicLuisDialog.cs`。它包含下列程式碼，以處理來自 LUIS 應用程式的意圖。

```

using System;
using System.Configuration;
using System.Threading.Tasks;

using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.Luis;
using Microsoft.Bot.Builder.Luis.Models;

namespace Microsoft.Bot.Sample.LuisBot
{
    // For more information about this template visit http://aka.ms/azurebots-csharp-luis
    [Serializable]
    public class BasicLuisDialog : LuisDialog<object>
    {
        public BasicLuisDialog() : base(new LuisService(new LuisModelAttribute(
            ConfigurationManager.AppSettings["LuisAppId"],
            ConfigurationManager.AppSettings["LuisAPIKey"],
            domain: ConfigurationManager.AppSettings["LuisAPIHostName"])))
        {
        }

        [LuisIntent("None")]
        public async Task NoneIntent(IDialogContext context, LuisResult result)
        {
            await this.ShowLuisResult(context, result);
        }

        // Go to https://luis.ai and create a new intent, then train/publish your luis app.
        // Finally replace "Greeting" with the name of your newly created intent in the following handler
        [LuisIntent("Greeting")]
        public async Task GreetingIntent(IDialogContext context, LuisResult result)
        {
            await this.ShowLuisResult(context, result);
        }

        [LuisIntent("Cancel")]
        public async Task CancelIntent(IDialogContext context, LuisResult result)
        {
            await this.ShowLuisResult(context, result);
        }

        [LuisIntent("Help")]
        public async Task HelpIntent(IDialogContext context, LuisResult result)
        {
            await this.ShowLuisResult(context, result);
        }

        private async Task ShowLuisResult(IDialogContext context, LuisResult result)
        {
            await context.PostAsync($"You have reached {result.Intents[0].Intent}. You said: {result.Query}");
            context.Wait(MessageReceived);
        }
    }
}

```

建立用來儲存筆記的類別

在 BasicLuisDialog.cs 中新增下列 `using` 陳述式。

```
using System.Collections.Generic;
```

在建構函式定義之後，於 `BasicLuisDialog` 類別內新增下列程式碼。

```
// Store notes in a dictionary that uses the title as a key
private readonly Dictionary<string, Note> noteByTitle = new Dictionary<string, Note>();

[Serializable]
public sealed class Note : IEquatable<Note>
{
    public string Title { get; set; }
    public string Text { get; set; }

    public override string ToString()
    {
        return $"[{this.Title} : {this.Text}]";
    }

    public bool Equals(Note other)
    {
        return other != null
            && this.Text == other.Text
            && this.Title == other.Title;
    }

    public override bool Equals(object other)
    {
        return Equals(other as Note);
    }

    public override int GetHashCode()
    {
        return this.Title.GetHashCode();
    }
}

// CONSTANTS
// Name of note title entity
public const string Entity_Note_Title = "Note.Title";
// Default note title
public const string DefaultNoteTitle = "default";
```

處理 Note.Create 意圖

若要處理 Note.Create 意圖，請將下列程式碼新增至 `BasicLuisDialog` 類別。

```

private Note noteToCreate;
private string currentTitle;
[LuisIntent("Note.Create")]
public Task NoteCreateIntent(IDialogContext context, LuisResult result)
{
    EntityRecommendation title;
    if (!result.TryFindEntity(Entity_Note_Title, out title))
    {
        // Prompt the user for a note title
        PromptDialog.Text(context, After_TitlePrompt, "What is the title of the note you want to
create?");
    }
    else
    {
        var note = new Note() { Title = title.Entity };
        noteToCreate = this.noteByTitle[note.Title] = note;

        // Prompt the user for what they want to say in the note
        PromptDialog.Text(context, After_TextPrompt, "What do you want to say in your note?");
    }

    return Task.CompletedTask;
}

private async Task After_TitlePrompt(IDialogContext context, IAwaitable<string> result)
{
    EntityRecommendation title;
    // Set the title (used for creation, deletion, and reading)
    currentTitle = await result;
    if (currentTitle != null)
    {
        title = new EntityRecommendation(type: Entity_Note_Title) { Entity = currentTitle };
    }
    else
    {
        // Use the default note title
        title = new EntityRecommendation(type: Entity_Note_Title) { Entity = DefaultNoteTitle };
    }

    // Create a new note object
    var note = new Note() { Title = title.Entity };
    // Add the new note to the list of notes and also save it in order to add text to it later
    noteToCreate = this.noteByTitle[note.Title] = note;

    // Prompt the user for what they want to say in the note
    PromptDialog.Text(context, After_TextPrompt, "What do you want to say in your note?");

}

private async Task After_TextPrompt(IDialogContext context, IAwaitable<string> result)
{
    // Set the text of the note
    noteToCreate.Text = await result;

    await context.PostAsync($"Created note **{this.noteToCreate.Title}** that says \
{this.noteToCreate.Text}\n.");
    context.Wait(MessageReceived);
}

```

處理 Note.ReadAloud 意圖

Bot 可以使用 `Note.ReadAloud` 意圖來顯示筆記的內容，如果未偵測到筆記標題，則會顯示所有筆記的內容。

將下列程式碼貼到 `BasicLuisDialog` 類別中。

```

[LuisIntent("Note.ReadAloud")]
public async Task NoteReadAloudIntent(IDialogContext context, LuisResult result)
{
    Note note;
    if (TryFindNote(result, out note))
    {
        await context.PostAsync($"**{note.Title}**: {note.Text}.");
    }
    else
    {
        // Print out all the notes if no specific note name was detected
        string NoteList = "Here's the list of all notes: \n\n";
        foreach (KeyValuePair<string, Note> entry in noteByTitle)
        {
            Note noteInList = entry.Value;
            NoteList += $"**{noteInList.Title}**: {noteInList.Text}.\n\n";
        }
        await context.PostAsync(NoteList);
    }

    context.Wait(MessageReceived);
}

public bool TryFindNote(string noteTitle, out Note note)
{
    bool foundNote = this.noteByTitle.TryGetValue(noteTitle, out note); // TryGetValue returns false if
no match is found.
    return foundNote;
}

public bool TryFindNote(LuisResult result, out Note note)
{
    note = null;

    string titleToFind;

    EntityRecommendation title;
    if (result.TryFindEntity(Entity_Note_Title, out title))
    {
        titleToFind = title.Entity;
    }
    else
    {
        titleToFind = DefaultNoteTitle;
    }

    return this.noteByTitle.TryGetValue(titleToFind, out note); // TryGetValue returns false if no
match is found.
}

```

處理 Note.Delete 意圖

將下列程式碼貼到 `BasicLuisDialog` 類別中。

```
[LuisIntent("Note.Delete")]
public async Task NoteDeleteIntent(IDialogContext context, LuisResult result)
{
    Note note;
    if (TryFindNote(result, out note))
    {
        this.noteByTitle.Remove(note.Title);
        await context.PostAsync($"Note {note.Title} deleted");
    }
    else
    {
        // Prompt the user for a note title
        PromptDialog.Text(context, After_DeleteTitlePrompt, "What is the title of the note you want to
delete?");
    }
}

private async Task After_DeleteTitlePrompt(IDialogContext context, IAwaitable<string> result)
{
    Note note;
    string titleToDelete = await result;
    bool foundNote = this.noteByTitle.TryGetValue(titleToDelete, out note);

    if (foundNote)
    {
        this.noteByTitle.Remove(note.Title);
        await context.PostAsync($"Note {note.Title} deleted");
    }
    else
    {
        await context.PostAsync($"Did not find note named {titleToDelete}.");
    }

    context.Wait(MessageReceived);
}
```

建置 Bot

在程式碼編輯器中，以滑鼠右鍵按一下 **build.cmd**，然後選擇 [從主控台執行]。

The screenshot shows the App Service Editor interface with the title bar "App Service Editor | notesbot-83ff". On the left is a sidebar with various icons for file operations like copy, search, and refresh. The main area shows a file tree under "WORKING FILES". A context menu is open over the file "BasicLuisDialog.cs" in the "Dialogs" folder. The menu items are:

- Run from Console
- Open to the Side Ctrl+Enter
- Select for Compare
- Copy Ctrl+C
- Rename F2
- Delete

The code editor on the right displays the following C# code:

```
BasicLuisDialog.cs
1 usi
2 usi
3 usi
4
5 usi
6 usi
7 usi
8
9 //
10 usi
11
12 nam
13 {
14
15
16
17
18
```

測試 Bot

在 Azure 入口網站中，按一下 [在網路聊天中測試] 來測試 Bot。請嘗試輸入像是「建立筆記」、「讀取我的筆記」和「刪除筆記」等訊息。

Home > NotesBot - Test in Web Chat

NotesBot - Test in Web Chat

SDK Bot (Web App)

Search (Ctrl+ /) Start over

Test

create a note

What is the title of the note you want to create?

NotesBot

Grocery List

What do you want to say in your note?

NotesBot

Milk and eggs

Created note **Grocery List** that says "Milk and eggs".

NotesBot

create note for me

What is the title of the note you want to create?

NotesBot

Shopping List

What do you want to say in your note?

NotesBot

buy light bulbs

Created note **Shopping List** that says "buy light bulbs".

NotesBot

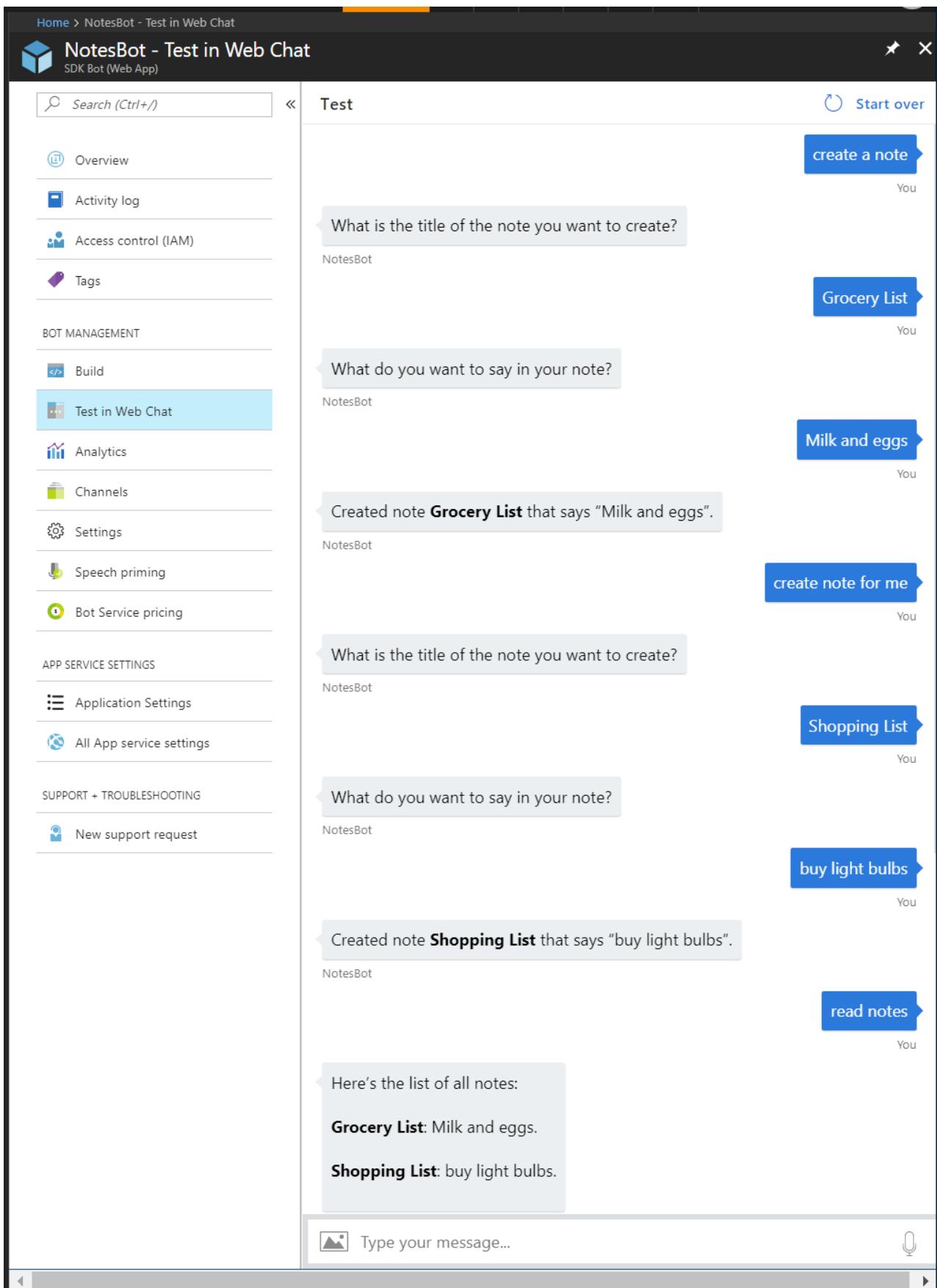
read notes

Here's the list of all notes:

Grocery List: Milk and eggs.

Shopping List: buy light bulbs.

Type your message... Microphone icon



TIP

如果您發現 Bot 並未總是辨識出正確的意圖或實體，請提供更多範例語句來進行 LUIS 應用程式定型，以改善應用程式的效能。您無須對 Bot 程式碼進行任何修改，即可將 LUIS 應用程式重新定型。請參閱[新增範例語句 \(英文\)](#)和[對您的 LUIS 應用程式進行定型和測試 \(英文\)](#)。

TIP

如果您的 Bot 程式碼執行發生問題，請檢查下列各項：

- 您必須建置 Bot。
- 您的 Bot 程式碼會在 LUIS 應用程式中，定義適用於每個意圖的處理常式。

後續步驟

透過試用此 Bot，您可以查看 LUIS 意圖叫用工作的方式。不過，這個簡單範例不允許中斷目前作用中的對話。允許並處理中斷（例如「說明」或「取消」）是一項有彈性的設計，該設計會考量使用者真正執行的動作。深入了解如何使用可評分的對話，如此就可讓您的對話處理中斷。

[使用可評分對話的全域訊息處理常式](#)

其他資源

- [對話方塊](#)
- [使用對話 \(dialogue\) 管理對話 \(conversation\) 流程](#)
- [LUIS](#)
- [適用於 .NET 的 Bot Framework SDK 參考](#)

要求付款

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

如果 Bot 要讓使用者能購買商品，則可在[複合式資訊卡 \(Rich Card\)](#) 中包含一個特別的按鈕，藉此要求付款。本文說明如何使用適用於 .NET 的 Bot Framework SDK 傳送付款要求。

必要條件

您必須先完成這些必要工作，才可以使用適用於 .NET 的 Bot Framework SDK 傳送付款要求。

更新 Web.config

更新 Bot 的 **Web.config** 檔案，以將 `MicrosoftAppId` 和 `MicrosoftAppPassword` 設為在[註冊](#)程序中針對 Bot 所產生的應用程式識別碼和密碼值。

NOTE

若要尋找 Bot 的 **AppID** 和 **AppPassword**，請參閱 [MicrosoftAppID](#) 和 [MicrosoftAppPassword](#)。

建立及設定商家帳戶

1. [如果您還沒有 Stripe 帳戶，請建立並啟用一個帳戶。](#)
2. [使用您的 Microsoft 帳戶登入賣方中心。](#)
3. 在賣方中心內，連結您的帳戶與 Stripe。
4. 在賣方中心內，瀏覽至儀表板，並複製 **MerchantID** 的值。
5. 更新 Bot 的 **Web.config** 檔案，以將 `MerchantId` 設為您從賣方中心儀表板複製的值。

付款程序概觀

付款程序包含三個不同的部分：

1. Bot 傳送付款要求。
2. 使用者登入 Microsoft 帳戶來提供付款、運送和連絡資訊。回呼會傳送至 Bot 以指出 Bot 何時需要執行某些作業 (更新運送地址、更新運送選項、完成付款)。
3. Bot 會處理收到的回呼，包括更新運送地址、更新運送選項和完成付款。

Bot 只需要實作此程序的步驟一和步驟三，步驟二會在 Bot 的內容之外進行。

付款 Bot 範例

[付款 Bot](#) 範例提供使用 .NET 傳送付款要求的 Bot 範例。若要查看作用中的這個範例 Bot，您可以[在網路聊天中試用](#)、[將 Bot 新增為 Skype 連絡人](#)，或下載付款 Bot 範例並使用 Bot Framework Emulator 在本機執行。

NOTE

若要在網路聊天或 Skype 中使用付款 Bot 範例完成端對端付款程序，您必須在您的 Microsoft 帳戶中指定有效的信用卡或金融卡（亦即，美國發卡機構簽發的有效卡片）。我們不會向您的卡片收取費用，也不會驗證卡片的 CVV（信用卡檢查碼），因為付款 Bot 範例是在測試模式中執行（也就是，`LiveMode` 在 `Web.config` 中設定為 `false`）。

本文的後續幾節會在付款 Bot 範例的內容中，說明付款程序的三個部分。

要求付款

Bot 可藉由傳送一則包含**豐富資訊卡附件**的訊息來向使用者要求付款，其中有一個指定 `CardAction.Type` 為「付款」的按鈕。付款 Bot 範例中的這個程式碼片段會建立一則包含主圖卡的訊息，其中有一個 [購買] 按鈕，使用者按一下（或點選）該按鈕即可起始付款程序。

```
var replyMessage = context.MakeMessage();

replyMessage.Attachments = new List<Attachment>();

var displayedItem = await new CatalogService().GetRandomItemAsync();

var cartId = displayedItem.Id.ToString();
context.ConversationData.SetValue(CART_KEY, cartId);
context.ConversationData.SetValue(cartId, context.Activity.From.Id);

var heroCard = new HeroCard
{
    Title = displayedItem.Title,
    Subtitle = $"{displayedItem.Currency} {displayedItem.Price.ToString("F")}",
    Text = displayedItem.Description,
    Images = new List<CardImage>
    {
        new CardImage
        {
            Url = displayedItem.ImageUrl
        }
    },
    Buttons = new List<CardAction>
    {
        new CardAction
        {
            Title = "Buy",
            Type = PaymentRequest.PaymentActionType,
            Value = BuildPaymentRequest(cartId, displayedItem, MethodData)
        }
    }
};

replyMessage.Attachments.Add(heroCard.ToAttachment());

await context.PostAsync(replyMessage);
```

在此範例中，按鈕的類型會指定為 `PaymentRequest.PaymentActionType`，而 Bot Builder 程式庫將其定義為「付款」。此按鈕的值會由 `BuildPaymentRequest` 方法填入，其將傳回 `PaymentRequest` 物件，當中包含支援的付款方式、詳細資料和選項相關資訊。如需有關實作詳情的詳細資訊，請參閱**付款 Bot** 範例中的 `Dialogs/RootDialog.cs`。

此螢幕擷取畫面會顯示由上述程式碼片段所產生的主圖卡（具有 [購買] 按鈕）。

paymentsample X +

← → ⌂ webchat.botframework.com/embed/paymentsample?si= Chat



Scott Gu - Favorite Shirt
USD 1.99
Shiny red, ready to rock on Keynotes

[Buy](#)

paymentsample at 3:33:59 PM

Type your message... 

IMPORTANT

只要使用者可存取 [購買] 按鈕，就可以透過該按鈕起始付款程序。在群組對話的內容中，不可能指定一個按鈕，僅供特定使用者使用。

使用者體驗

使用者按一下 [購買] 按鈕後，系統會將其導向至付款 Web 體驗，以透過其 Microsoft 帳戶提供所有必要的付款、運送及連絡人資訊。

Confirm and Pay

Pay with

Ship to Select shipping address

Shipping options

Email receipt to

Phone number

Total (USD) [Show details](#) \$1.99*

* - Indicates the cost isn't final

Pay

HTTP 回呼

HTTP 回呼將傳送至 Bot，指出其應執行特定作業。每個回呼都是包含下列屬性值的活動：

屬性	值
<code>Activity.Type</code>	叫用
<code>Activity.Name</code>	指出 Bot 應該執行的作業類型 (例如：交貨地址更新、交貨選項更新、付款完成)。
<code>Activity.Value</code>	採用 JSON 格式的要求承載。
<code>Activity.RelatesTo</code>	說明與付款要求相關聯的通道和使用者。

NOTE

`invoke` 是保留給 Microsoft Bot Framework 使用的特別活動類型。`invoke` 活動的傳送者預期 Bot 會藉由傳送 HTTP 回應來認可回呼。

處理回呼

當您的 Bot 接收到回呼時，它應該確認回呼中所指定的資訊是有效的，並藉由傳送 HTTP 回應來確認該回呼。

交貨地址更新和交貨選項更新回呼

接收「交貨地址更新」或「交貨選項更新」回呼，用戶端會在 `Activity.Value` 屬性中將付款詳細資料的目前狀態提供給 Bot。如果您是商家，則應將這些回呼視為靜態，根據輸入付款詳細資料，將會計算某些輸出付款詳細資料，而如果用戶端所提供的輸入狀態因任何理由而無效，則回呼會失敗。如果 Bot 判斷指定的現況資訊有效，則只要傳送 HTTP 狀態碼 `200 OK` 以及未經修改的付款詳細資料。或者，Bot 可以先傳送 HTTP 狀態碼 `200 OK` 以及應套用的更新後付款詳細資料，才能處理訂單。在某些情況下，Bot 可能會判斷更新後的資訊無效，所以無法照現況處理訂單。例如，使用者的交貨地址可能會指定產品供應商不出貨的國家/地區。在此情況下，Bot 可以傳送 HTTP 狀態碼 `200 OK` 和一則訊息，其中填入付款詳細資料物件的錯誤屬性。傳送 `400` 至 `500` 範圍內的任何 HTTP 狀態碼會導致客戶發生一般錯誤。

付款完成回呼

接收「付款完成」回呼時，Bot 會在物件的 `Activity.Value` 屬性中取得一份未經修改的最初付款要求，以及付款回應事件。付款回應物件將包含客戶最終選取的項目以及付款權杖。Bot 應藉此機會根據最初付款要求和客戶最終選取的項目，重新計算最終付款要求。假設系統將客戶的選取項目判定為有效，則 Bot 應確認付款權杖標頭中的金額和貨幣，確保其符合最終付款要求。如果 Bot 決定向客戶收費，則只應收取付款權杖標頭中的金額，因為這是客戶所確認的價格。如果 Bot 預期的值和在「付款完成」回呼中收到的值不相符，則可傳送 HTTP 狀態碼 `200 OK` 並將結果欄位設定為 `failure`，藉此讓付款要求失敗。

除了驗證付款詳細資料以外，Bot 也應該先確認可以履行訂單，再起始付款處理。例如，其可驗證所購買的項目是否仍可在庫存中取得。如果值正確無誤，而且付款處理器已順利向付款權杖收取費用，則 Bot 應該以 HTTP 狀態碼 `200 OK` 回應並將結果欄位設為 `success`，以便付款 Web 體驗顯示付款確認。Bot 收到的付款權杖只能由要求付款的商家使用一次，而且必須提交至 Stripe (Bot Framework 目前唯一支援的付款處理器)。傳送任何 `400` 或 `500` 範圍內的 HTTP 狀態碼會導致客戶發生一般錯誤。

付款 Bot 範例中的 `OnInvoke` 方法會處理 Bot 所接收的回呼。

```

[MethodBind]
[ScorableGroup(1)]
private async Task OnInvoke(IIgnoreActivity invoke, IConnectorClient connectorClient, IStateClient
stateClient, HttpResponseMessage response, CancellationToken token)
{
    MicrosoftAppCredentials.TrustServiceUrl(invoke.RelatesTo.ServiceUrl);

    var jobject = invoke.Value as JObject;
    if (jobject == null)
    {
        throw new ArgumentException("Request payload must be a valid json object.");
    }

    // This is a temporary workaround for the issue that the channelId for "webchat" is mapped to "directline"
    // in the incoming RelatesTo object
    invoke.RelatesTo.ChannelId = (invoke.RelatesTo.ChannelId == "directline") ? "webchat" :
    invoke.RelatesTo.ChannelId;

    if (invoke.RelatesTo.User == null)
    {
        // Bot keeps the userId in context.ConversationData[cartId]
        var conversationData = await stateClient.BotState.GetConversationDataAsync(invoke.RelatesTo.ChannelId,
        invoke.RelatesTo.Conversation.Id, token);
        var cartId = conversationData.GetProperty<string>(RootDialog.CARTKEY);

        if (!string.IsNullOrEmpty(cartId))
        {
            invoke.RelatesTo.User = new ChannelAccount
            {
                Id = conversationData.GetProperty<string>(cartId)
            };
        }
    }

    var updateResponse = default(object);

    switch (invoke.Name)
    {
        case PaymentOperations.UpdateShippingAddressOperationName:
            updateResponse = await ProcessShippingAddressUpdate(jobject.ToObject<PaymentRequestUpdate>(),
            token);
            break;

        case PaymentOperations.UpdateShippingOptionOperationName:
            updateResponse = await ProcessShippingOptionUpdate(jobject.ToObject<PaymentRequestUpdate>(),
            token);
            break;

        case PaymentOperations.PaymentCompleteOperationName:
            updateResponse = await ProcessPaymentComplete(invoke, jobject.ToObject<PaymentRequestComplete>(),
            token);
            break;

        default:
            throw new ArgumentException("Invoke activity name is not a supported request type.");
    }

    response.Content = new ObjectContent<object>(
        updateResponse,
        this.Configuration.Formatters.JsonFormatter,
        JsonMediaTypeFormatter.DefaultMediaType);

    response.StatusCode = HttpStatusCode.OK;
}

```

在此範例中，Bot 會檢查內送事件的 `Name` 屬性，以識別其需要執行的作業類型，然後呼叫適當的方法來處理回呼。

如需有關實作詳情的詳細資訊，請參閱[付款 Bot 範例](#)中的 **Controllers/MessagesControllers.cs**。

測試付款 Bot

若要完整測試要求付款的 Bot，請[設定](#)其在支援 Bot Framework 付款的管道（如網路聊天和 Skype）上執行。或者，您可以使用 [Bot Framework Emulator](#) 在本機測試 Bot。

TIP

當使用者在付款 Web 體驗期間變更資料或按一下 [支付] 時，會傳送回撥給您的 Bot。因此，您可以自行與付款 Web 體驗互動，來測試 Bot 接收和處理回撥的功能。

在[付款 Bot 範例](#)中，**Web.config** 中的 `LiveMode` 組態設定可決定「付款完成」回呼將包含模擬的付款權杖或實際的付款權杖。如果 `LiveMode` 設為 `false`，則標頭會新增至 Bot 的輸出付款要求，以指出 Bot 處於測試模式，而且「付款完成」回呼將包含無法收費的模擬付款權杖。如果 `LiveMode` 設為 `true`，則指出 Bot 處於測試模式的標頭會從 Bot 的輸出付款要求中省略，而且「付款完成」回呼會包含 Bot 將提交至 Stripe 進行付款處理的實際付款權杖。這會是向指定的付款工具收費的真實交易。

其他資源

- [付款 Bot 範例](#)
- [活動概觀](#)
- [將豐富資訊卡新增至訊息](#)
- [在 W3C 進行網路付款](#)
- [適用於 .NET 的 Bot Framework SDK 參考](#)

使用 Azure 搜尋服務建立資料驅動體驗

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

您可以將 [Azure 搜尋服務](#)新增至 Bot，以協助使用者瀏覽大量內容並建立資料驅動的探索體驗。

Azure 搜尋服務是一項 Azure 服務，可提供關鍵字搜尋、內建語言、自訂評分、多面向導覽等等。Azure 搜尋服務也可以對各種來源（包括 Azure SQL DB、DocumentDB、Blob 儲存體和表格儲存體）的內容編製索引。它支援「推送」其他資料來源的編製索引，而且可以開啟 PDF、Office 文件，以及其他包含非結構化資料的格式。收集後，內容就會進入 Azure 搜尋服務索引，然後 Bot 就可加以查詢。

必要條件

在您的 Bot 專案中安裝 [Microsoft.Azure.Search](#) Nuget 套件。

Bot 解決方案需要下列三個 C# 專案。這些專案會為 Bot 與 Azure 搜尋服務提供額外功能。從 [GitHub](#) 分支處理專案或直接下載原始程式碼。

- **Search.Azure** 專案可定義 Azure 服務呼叫。
- **Search.Contracts** 專案可定義一般介面和資料模型來處理資料。
- **Search.Dialogs** 專案包含用來查詢 Azure 搜尋服務的一般 Bot Builder 對話方塊。

進行 Azure 搜尋服務設定

在值欄位中使用自有的 Azure 搜尋服務認證，在專案的 **Web.config** 檔案中設定 Azure 搜尋服務設定。

`AzureSearchClient` 類別中的建構函式將使用這些設定來登錄並將 Bot 繫結至 Azure 服務。

```
<appSettings>
    <add key="SearchDialogsServiceName" value="Azure-Search-Service-Name" /> <!-- replace value field with
    Azure Service Name -->
    <add key="SearchDialogsServiceKey" value="Azure-Search-Service-Primary-Key" /> <!-- replace value field
    with Azure Service Key -->
    <add key="SearchDialogsIndexName" value="Azure-Search-Service-Index" /> <!-- replace value field with your
    Azure Search Index -->
</appSettings>
```

建立搜尋對話方塊

在 Bot 的專案中，建立新的 `AzureSearchDialog` 類別以在 Bot 中呼叫 Azure 服務。這個新類別必須從 **Search.Dialogs** 專案繼承 `SearchDialog` 類別，以處理大部分的苦差事。`GetTopRefiners()` 覆寫可讓使用者縮小/篩選其搜尋結果，而不必從頭開始進行搜尋，並維護搜尋物件的狀態。您可以在 `TopRefiners` 陣列中新增自有的自訂精簡器，讓使用者篩選或縮小其搜尋結果。

```
[Serializable]
public class AzureSearchDialog : SearchDialog
{
    private static readonly string[] TopRefiners = { "refiner1", "refiner2", "refiner3" }; // define your own
    custom refiners

    public AzureSearchDialog(ISearchClient searchClient) : base(searchClient, multipleSelection: true)
    {
    }

    protected override string[] GetTopRefiners()
    {
        return TopRefiners;
    }
}
```

定義回應資料模型

`Search.Contracts` 專案內的 **SearchHit.cs** 類別會定義要從 Azure 搜尋服務回應剖析的相關資料。對 Bot 而言，唯一強制的包含項目為建構函式中的 `PropertyBag` `IDictionary` 告知和建立。您可以在有關 Bot 需求的這個類別中定義所有其他屬性。

```
[Serializable]
public class SearchHit
{
    public SearchHit()
    {
        this.PropertyBag = new Dictionary<string, object>();
    }

    public IDictionary<string, object> PropertyBag { get; set; }

    // customize the fields below as needed
    public string Key { get; set; }

    public string Title { get; set; }

    public string PictureUrl { get; set; }

    public string Description { get; set; }
}
```

Azure 搜尋服務回應之後

成功查詢 Azure 服務時，就必須剖析搜尋結果來擷取相關資料，以便 Bot 向使用者顯示。若要啟用此功能，您必須建立 `SearchResultMapper` 類別。建構函式中所建立的 `GenericSearchResult` 物件可定義清單和字典，以在分別對 Bot 資料模型剖析每個結果之後，分別用來儲存結果和 Facet。

同步處理 `ToSearchHit` 方法中的屬性，以符合 **SearchHit.cs** 中的資料模型。`ToSearchHit` 方法將會執行，並針對在回應中找到每個結果產生新的 `SearchHit`。

```

public class SearchResultMapper : IMapper<DocumentSearchResult, GenericSearchResult>
{
    public GenericSearchResult Map(DocumentSearchResult documentSearchResult)
    {
        var searchResult = new GenericSearchResult();

        searchResult.Results = documentSearchResult.Results.Select(r => ToSearchHit(r)).ToList();
        searchResult.Facets = documentSearchResult.Facets?.ToDictionary(kv => kv.Key, kv => kv.Value.Select(f => ToFacet(f)));
    }

    return searchResult;
}

private static GenericFacet ToFacet(FacetResult facetResult)
{
    return new GenericFacet
    {
        Value = facetResult.Value,
        Count = facetResult.Count.Value
    };
}

private static SearchHit ToSearchHit(SearchResult hit)
{
    return new SearchHit
    {
        // custom properties defined in SearchHit.cs
        Key = (string)hit.Document["id"],
        Title = (string)hit.Document["title"],
        PictureUrl = (string)hit.Document["thumbnail"],
        Description = (string)hit.Document["description"]
    };
}
}

```

剖析並儲存結果之後，仍需要向使用者顯示此資訊。需要管理 `SearchHitStyler` 類別，以配合 `SearchHit` 類別中您的資料模型。例如，`SearchHit.cs` 類別中的 `Title`、`PictureUrl` 和 `Description` 屬性會使用於範例程式碼，以建立新的卡片附件。下列程式碼會針對 `GenericSearchResult` 結果清單中的每個 `SearchHit` 物件，建立新的卡片附件以向使用者顯示。

```
[Serializable]
public class SearchHitStyler : PromptStyler
{
    public override void Apply<T>(ref IMessageActivity message, string prompt, IReadOnlyList<T> options,
IReadOnlyList<string> descriptions = null)
    {
        var hits = options as IList<SearchHit>;
        if (hits != null)
        {
            var cards = hits.Select(h => new ThumbnailCard
            {
                Title = h.Title,
                Images = new[] { new CardImage(h.PictureUrl) },
                Buttons = new[] { new CardAction(ActionTypes.ImBack, "Pick this one", value: h.Key) },
                Text = h.Description
            });

            message.AttachmentLayout = AttachmentLayoutTypes.Carousel;
            message.Attachments = cards.Select(c => c.ToAttachment()).ToList();
            message.Text = prompt;
        }
        else
        {
            base.Apply<T>(ref message, prompt, options, descriptions);
        }
    }
}
```

搜尋結果會向使用者顯示，而且您已成功將 Azure 搜尋服務新增至 Bot。

範例

如需兩個示範如何使用適用於 .NET 的 Bot Framework SDK 支援 Azure 搜尋服務的完整 Bot 範例，請參閱 GitHub 中的[不動產 Bot 範例](#)或[作業清單 Bot 範例](#)。

其他資源

- [Azure 搜尋服務](#)
- [對話概觀](#)

保護 Bot

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

透過 Bot Framework 連接器服務，Bot 可以連線到許多不同的通訊通道 (Skype、SMS、電子郵件等其他項目)。本文將說明如何使用 HTTPS 與 Bot Framework 驗證來保護您的 Bot。

使用 HTTPS 和 Bot Framework 驗證

若要確保您的 Bot 端點僅能透過 Bot Framework [連接器](#)來存取，請將 Bot 的端點設定為只能使用 HTTPS，並[註冊](#)Bot 來取得其 AppID 和密碼，以啟用 Bot Framework 驗證。

設定 Bot 的驗證

在 Bot 的 web.config 檔案中指定 Bot AppID 和密碼。

NOTE

若要尋找 Bot 的 **AppID** 和 **AppPassword**，請參閱 [MicrosoftAppId](#) 和 [MicrosoftAppPassword](#)。

```
<appSettings>
  <add key="MicrosoftAppId" value="_appIdValue_" />
  <add key="MicrosoftAppPassword" value="_passwordValue_" />
</appSettings>
```

然後，在使用適用於 .NET 的 Bot Framework SDK 來建立 Bot 時，使用 `[BotAuthentication]` 屬性來指定驗證認證。

若要使用儲存在 web.config 檔案中的驗證認證，請指定 `[BotAuthentication]` (不含任何參數)。

```
[BotAuthentication]
public class MessagesController : ApiController
{
}
```

若要使用其他值作為驗證認證，請指定 `[BotAuthentication]` 屬性，並傳入那些值。

```
[BotAuthentication(MicrosoftAppId = "_appIdValue_", MicrosoftAppPassword=_passwordValue_")]
public class MessagesController : ApiController
{
}
```

其他資源

- [適用於 .NET 的 Bot Framework SDK](#)

- 適用於 .NET 的 Bot 建立器 SDK 重要概念
- 使用 Bot Framework 註冊 Bot

適用於 Node.js 的 Bot Framework SDK

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

適用於 Node.js 的 Bot Framework SDK 是功能強大、簡單易用的架構，可為 Node.js 開發人員提供熟悉的 Bot 撰寫方式。您可以用它建立各種不同的對話式使用者介面，從簡單的提示到自由形式的對話都在其適用範圍內。

Bot 的對話式邏輯會裝載為 Web 服務。Bot Framework SDK 會使用 [restify](#) 這個受歡迎的 Web 服務建立架構，來建立 Bot 的 Web 伺服器。SDK 也與 [Express](#) 相容，稍做調整即可使用其他 Web 應用程式架構。

使用 SDK 就可以利用下列 SDK 功能：

- 功能強大的系統，用於建立對話以封裝對話式邏輯。
- 內建簡單事物提示（例如，是/否、字串、數字和列舉），以及支援內含影像和附件的訊息和內含按鈕的複合式資訊卡（Rich Card）。
- 內建支援功能強大的 AI 架構，例如 [LUIS](#)。
- 內建辨識器和事件處理常式，可引導使用者完成對話，視需要提供說明、瀏覽、釐清和確認。

開始使用

如果您未曾撰寫過 Bot，請透過逐步指示[使用 Node.js 建立第一個 Bot](#)，讓指示協助您設定專案、安裝 SDK 和執行第一個 Bot。

如果您未曾使用過適用於 Node.js 的 Bot Framework SDK，您可以從重要概念來開始，讓其協助您了解 Bot Framework SDK 的主要元件，請參閱[重要概念](#)。

若要確保 Bot 能應付最常見的使用者案例，請檢閱[設計原則](#)和[瀏覽模式](#)以獲得指導方針。

取得範例

[適用於 Node.js 的 Bot Framework SDK 範例](#)會示範以工作為主的 Bot，示範如何利用適用於 Node.js 的 Bot Framework SDK 功能。您可使用這些範例，協助您快速開始建置具備豐富功能的絕佳 Bot。

後續步驟

[重要概念](#)

其他資源

下列以工作為主的使用說明指南，會示範適用於 Node.js 的 Bot Framework SDK 的各種功能。

- [回應訊息](#)
- [處理使用者動作](#)
- [辨識使用者意圖](#)
- [傳送複合式資訊卡 \(Rich Card\)](#)
- [傳送附件](#)
- [儲存使用者資料](#)

如果您有關於適用於 Node.js 的 Bot Framework SDK 的問題或建議，請參閱[支援](#)以取得可用資源清單。

適用於 Node.js 的 Bot Framework SDK 重要概念

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

本文介紹適用於 Node.js 的 Bot Framework SDK 重要概念。如需 Bot Framework 的簡介，請參閱[Bot Framework 概觀](#)。

連接器

Bot Framework Connector 是一項可將 Bot 連線至多個通道(即 Skype、Facebook、Slack 和 SMS 等用戶端)的服務。此連接器可藉由從 Bot 至通道和從通道至 Bot 的訊息轉送，讓 Bot 和使用者之間的通訊更為順暢。Bot 的邏輯裝載為 Web 服務，並透過連接器服務接收使用者的訊息，而 Bot 的回覆會使用 HTTPS POST 傳送至連接器。

適用於 Node.js 的 Bot Framework SDK 所提供的 [UniversalBot](#) 和 [ChatConnector](#) 類別可用來設定 Bot，使其透過 Bot Framework Connector 來傳送和接收訊息。[UniversalBot](#) 類別會形成 Bot 的思考核心。它負責管理 Bot 與使用者之間的所有對話。[ChatConnector](#) 類別會將您的 Bot 連線至 Bot Framework Connector 服務。如需使用這些類別進行示範的範例，請參閱[使用適用於 Node.js 的 Bot Framework SDK 建立 Bot](#)。

連接器也會將 Bot 傳送至通道的訊息正規化，讓您能夠以跨平台的方式開發 Bot。要將訊息正規化，必須將訊息從 Bot Framework 的結構描述轉換為通道的結構描述。如果通道不支援架構結構描述的所有層面，則連接器會嘗試將訊息轉換為通道支援的格式。例如，如果 Bot 將訊息傳送至 SMS 通道，而訊息中包含具有動作按鈕的卡片，則連接器可能會將卡片轉譯為影像，並將動作包含為訊息文字中的連結。[通道偵測器](#)是一項 Web 工具，會說明連接器在各種通道上轉譯訊息的方式。

要使用 [ChatConnector](#)，您必須在 Bot 內設定 API 端點。使用 Node.js SDK 時，這項作業通常可藉由安裝 [restify](#) Node.js 模組來完成。您也可以使用 [ConsoleConnector](#) 為主控台建立 Bot，而不需要 API 端點。

訊息

訊息中可包含要顯示的文字、要讀出的文字、附件、複合式資訊卡 (Rich Card) 和建議的動作。您可以使用 [session.send](#) 方法來傳送訊息，以回應使用者的訊息。您的 Bot 可視需要呼叫任意次數的 [send](#)，以回應使用者的訊息。如需示範這項功能的範例，請參閱[回應使用者訊息](#)。

如需相關範例，以了解如何傳送含有互動式按鈕，可讓使用者點選以起始動作的圖形化複合式資訊卡，請參閱[將複合式資訊卡 \(Rich Card\) 新增至訊息](#)。如需示範如何傳送和接收附件的範例，請參閱[傳送附件](#)。如需相關範例，以了解如何在訊息中指定您的 Bot 在具備語音功能的通道上要說出的文字，並傳送訊息，請參閱[將語音新增至訊息](#)。如需示範如何傳送建議動作的範例，請參閱[傳送建議的動作](#)。

對話方塊

對話方塊可協助您組織 Bot 中的對話邏輯，而且可作為[設計對話流程](#)的基礎。如需對話的簡介，請參閱[使用對話方塊來管理對話](#)。

動作

您必須將 Bot 設計成具有中斷處理能力，例如，處理對話流程中隨時提出的取消或協助要求。適用於 Node.js 的 Bot Framework SDK 提供全域訊息處理常式，可觸發如取消或叫用其他對話之類的動作。如需如何使用

[triggerAction](#) 處理常式的範例，請參閱[處理使用者動作](#)。

辨識器

當使用者對您的 Bot 提出某項要求時（例如「協助」或「尋找新聞」），Bot 必須了解使用者的要求，並採取適當動作。您可以將 Bot 設計成根據使用者的輸入辨識意圖，並將該意圖與動作產生關聯。

您可以使用 Bot Framework SDK 所提供的內建規則運算式辨識器、呼叫的外部服務（例如 LUIS API），或實作自訂辨識器，以判斷使用者的意圖。如需示範如何將辨識器新增至 Bot 並用它來觸發動作的範例，請參閱[辨識使用者意圖](#)。

儲存狀態

Bot 的設計是否精良，要件之一是能夠追蹤對話的上下文，讓 Bot 記住使用者最後提問這類的事項。使用 Bot Framework SDK 建置的 Bot 會設計為無狀態，以便輕易調整為跨多個計算節點執行。Bot Framework 提供可儲存 Bot 資料的儲存體系統，讓 Bot Web 服務可進行調整。因此，您在一般情況下應避免使用全域變數或函式閉包來儲存狀態。如果這麼做，當您想要相應放大 Bot 時將會出現問題。您應使用 Bot 工作階段物件的下列屬性，來儲存相對於使用者或對話的資料：

- **userData** 會全域儲存使用者在所有對話中的資訊。
- **conversationData** 會全域儲存單一對話的資訊。這項資料會顯示給對話中的所有人，因此在將資料儲存至此屬性時應謹慎操作。依預設會啟用此屬性，但您可以使用 Bot 的 `persistConversationData` 設定將其停用。
- **privateConversationData** 會全域儲存單一交談的資訊，但這會是目前使用者專屬的私人資料。這項資料會跨越所有對話方塊，因此很適合用來儲存您在對話結束後將會清除的暫時性狀態。
- **dialogData** 會保存單一對話方塊執行個體的資訊。這是儲存對話方塊中[瀑布圖](#)步驟之間，暫存資訊所不可或缺的屬性。

如需示範如何使用這些屬性來儲存和擷取資料的範例，請參閱[管理狀態資料](#)。

自然語言理解

Bot 建立器可讓您使用 LUIS，將自然語言理解功能新增至使用 [LuisRecognizer](#) 類別的 Bot。您可以新增對已發佈語言模型進行參考的 **LuisRecognizer** 執行個體，然後再新增會執行動作以回應使用者語句的處理常式。若要了解運作中的 LUIS，請觀看以下長度約 10 分鐘的教學課程：

- [Microsoft LUIS 教學課程](#) (影片)

後續步驟

[對話概觀](#)

適用於 Node.js 的 Bot Framework SDK 中的對話

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

適用於 Node.js 的 Bot Framework SDK 中的對話，可讓您建立交談模型及管理交談流程。Bot 會透過對話與使用者通訊。對話 (conversation) 會被組織為對話 (dialog)。對話可以包含瀑布圖步驟和提示。當使用者與 Bot 互動時，Bot 將會開始和停止對話，以及在對話之間切換，以回應使用者訊息。了解對話的運作方式是成功設計和建立絕佳 Bot 的關鍵。

本文將會介紹對話的概念。在您讀完本文之後，請接著依照[後續步驟](#)一節中的連結深入探索這些概念。

透過對話 (dialog) 進行對話 (conversation)

適用於 Node.js 的 Bot Framework SDK 將交談定義為 Bot 與使用者之間透過一或多個對話所進行的通訊。對話就最基礎層級而言是一個可重複使用的模組，可以執行作業或收集使用者的資訊。您可以將 Bot 的複雜邏輯封裝於可重複使用的對話程式碼中。

對話可以數種方式進行結構化及變更：

- 它可以源自您的[預設對話](#)。
- 它可以從一個對話重新導向到另一個對話。
- 它可以繼續。
- 它可以依循[瀑布圖模式](#)，其能引導使用者進行一系列的步驟，或是以一系列的問題來[提示](#)使用者。
- 它可以使用[動作](#)來接聽會觸發不同對話的單字或片語。

您可以將對話 (conversation) 想像成對話 (dialog) 的父代。在此情況下，對話 (conversation) 會包含[對話 \(dialog\) 堆疊](#)，並維護一組自己的狀態資料，亦即 `conversationData` 和 `privateConversationData`。另一方面，對話 (dialog) 會維護 `dialogData`。如需狀態資料的詳細資訊，請參閱[管理狀態資料](#)。

對話堆疊

Bot 會透過一系列在對話堆疊上維護的對話來與使用者互動。在對話 (conversation) 期間，系統會將對話 (dialog) 推送到堆疊上，或是從其中移除。堆疊的運作方式類似一般的 LIFO 堆疊，這表示最後新增的對話將會第一個完成。完成某個對話之後，系統會將控制權還給堆疊上的前一個對話。

在第一次啟動 Bot 對話 (conversation) 或對話 (conversation) 結束時，對話 (dialog) 堆疊會是空的。此時，當使用者傳送訊息給 Bot 時，Bot 將以[預設對話](#)來回應。

預設對話

在 Bot Framework 3.5 版之前，[根對話](#)是藉由新增名為 `/` 的對話來定義，這導致命名慣例會與 URL 的類似。此命名慣例並不適合用來為對話命名。

NOTE

從 Bot Framework 3.5 版開始，[預設對話](#)會註冊為 `UniversalBot` (英文) 之建構函式中的第二個參數。

下列程式碼片段會示範如何在建立 `UniversalBot` 物件時定義預設對話。

```
var bot = new builder.UniversalBot(connector, [
    //...Default dialog waterfall steps...
]);
```

在對話堆疊是空的，且未透過 LUIS 或其他辨識器觸發任何其他對話時，即會執行預設對話。由於預設對話是 Bot 對使用者的第一個回應，因此預設對話應該要為使用者提供一些內容相關的資訊，例如可用命令的清單或 Bot 可執行項目的概觀。

對話處理常式

對話 (dialog) 處理常式會管理對話 (conversation) 的流程。為了進行對話 (conversation)，對話 (dialog) 處理常式會藉由開始和結束對話 (dialog) 來引導流程。

開始和結束對話

若要開始新的對話 (並將它推送到堆疊上)，請使用 `session.beginDialog()` (英文)。若要結束對話 (並將它從堆疊中移除，然後將控制權還給呼叫對話)，請使用 `session.endDialog()` (英文) 或 `session.endDialogWithResult()` (英文)。

使用瀑布圖和提示

瀑布圖是建立對話流程模型並加以管理的簡單方式。瀑布圖包含一系列的步驟。在每個步驟中，您可以代表使用者完成某個動作，或提示使用者輸入資訊。

瀑布圖是使用由函式集合所組成的對話來實作。每個函式都會定義瀑布圖中的一個步驟。下列程式碼範例會示範一個簡易對話，此對話會使用兩步驟的瀑布圖來提示使用者輸入其名字，並使用該名字問候他們。

```
// Ask the user for their name and greet them by name.
bot.dialog('greetings', [
    function (session) {
        builder.Prompts.text(session, 'Hi! What is your name?');
    },
    function (session, results) {
        session.endDialog(`Hello ${results.response}!`);
    }
]);
```

當 Bot 抵達瀑布圖的結尾，但沒有結束對話時，來自使用者的下一個訊息會從瀑布圖的第一個步驟重新開始該對話。這可能導致使用者覺得自己陷於迴圈中而感到挫折。若要避免此情況，在對話 (conversation) 或對話 (dialog) 抵達結尾時，最佳做法是明確呼叫 `endDialog`、`endDialogWithResult` 或 `endConversation`。

後續步驟

若要更深入探索對話，請務必了解瀑布圖模式的運作方式，以及如何用它來引導使用者完成程序。

[使用瀑布圖定義對話步驟](#)

使用瀑布定義交談步驟

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

交談是使用者與 Bot 之間交換的一系列訊息。當 Bot 的目標是引導使用者完成一系列的步驟時，您可以使用瀑布來定義交談的步驟。

瀑布是特定的**對話**實作，最常用來向使用者收集資訊或引導使用者完成一系列的工作。這些工作會實作為函式陣列，其中第一個函式的結果將作為輸入傳遞至下一個函式，依此類推。每個函式通常代表整個程序的其中一個步驟。在每個步驟中，Bot 會提示使用者提供輸入、等候回應，然後將結果傳遞給下一個步驟。

本文將協助您了解瀑布的運作方式，以及您如何使用它來[管理交談流程](#)。

交談步驟

交談通常涉及使用者與 Bot 之間的數個提示/回應交換。每個提示/回應交換都是交談的逐步前進。您可以使用只有兩個步驟的瀑布建立交談。

例如，請考慮下列 `greetings` 對話。瀑布的第一個步驟會提示輸入使用者名稱，而第二個步驟則使用回應來依名稱歡迎使用者。

```
bot.dialog('greetings', [
    // Step 1
    function (session) {
        builder.Prompts.text(session, 'Hi! What is your name?');
    },
    // Step 2
    function (session, results) {
        session.endDialog(`Hello ${results.response}!`);
    }
]);
```

實現此目的的方式是使用提示。適用於 Node.js 的 Bot Framework SDK 提供許多不同類型的內建**提示**，您可以使用這些提示來要求使用者提供各種類型的資訊。

下列範例程式碼顯示一則對話，此對話會使用提示在含有 4 個步驟的瀑布中向使用者收集各項資訊。

```

var inMemoryStorage = new builder.MemoryBotStorage();

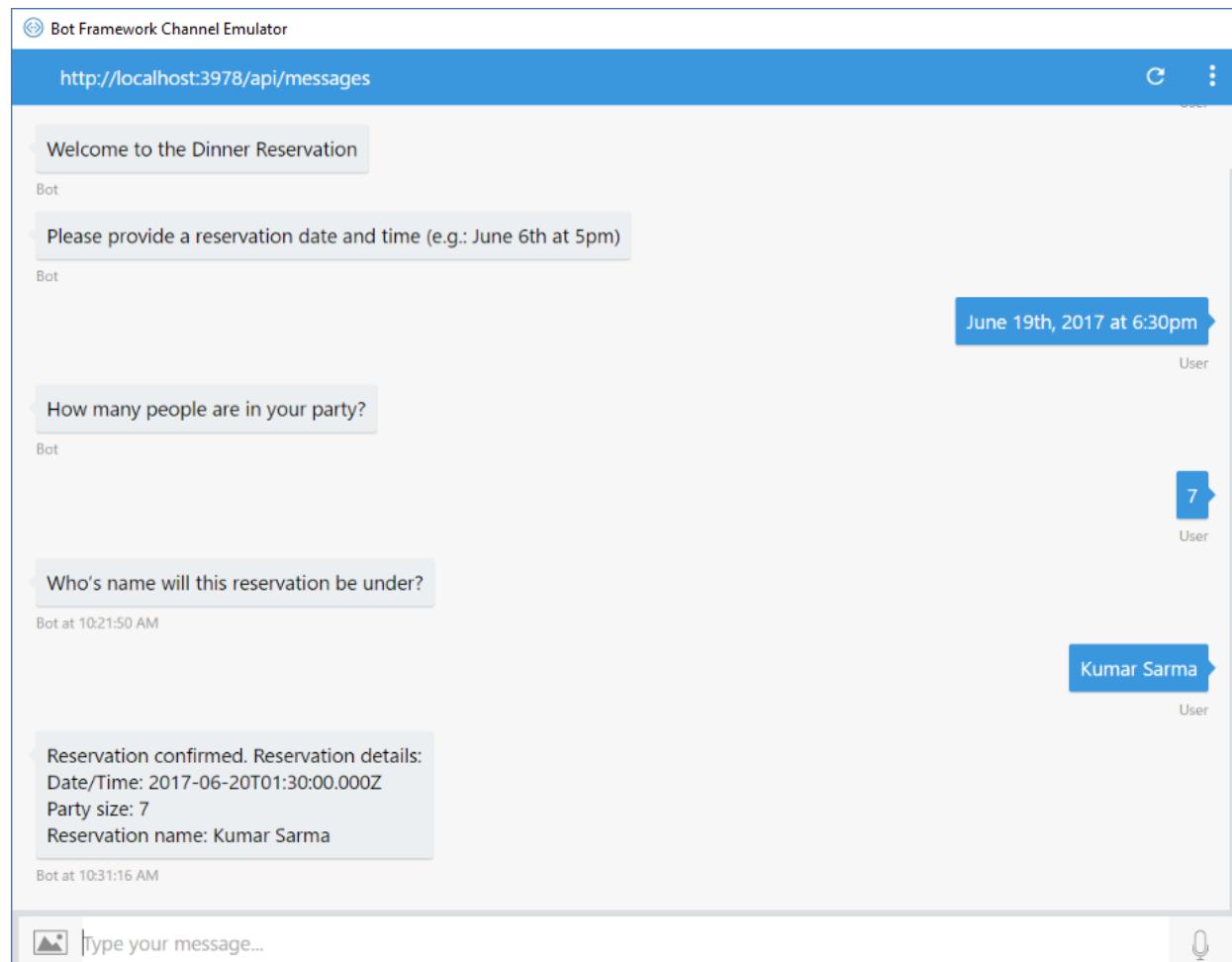
// This is a dinner reservation bot that uses a waterfall technique to prompt users for input.
var bot = new builder.UniversalBot(connector, [
    function (session) {
        session.send("Welcome to the dinner reservation.");
        builder.Prompts.time(session, "Please provide a reservation date and time (e.g.: June 6th at 5pm)");
    },
    function (session, results) {
        session.dialogData.reservationDate = builder.EntityRecognizer.resolveTime([results.response]);
        builder.Prompts.text(session, "How many people are in your party?");
    },
    function (session, results) {
        session.dialogData.partySize = results.response;
        builder.Prompts.text(session, "Whose name will this reservation be under?");
    },
    function (session, results) {
        session.dialogData.reservationName = results.response;

        // Process request and display reservation details
        session.send(`Reservation confirmed. Reservation details: <br/>Date/Time: ${session.dialogData.reservationDate} <br/>Party size: ${session.dialogData.partySize} <br/>Reservation name: ${session.dialogData.reservationName}`);
        session.endDialog();
    }
]).set('storage', inMemoryStorage); // Register in-memory storage

```

在此範例中，預設對話有四個函式，每個各代表瀑布中的一個步驟。每個步驟會提示使用者提供輸入，並將結果傳送到要處理的下一個步驟。此程序會繼續最後一個步驟執行，藉此確認預約並結束對話。

下列螢幕擷取畫面顯示在 [Bot Framework 模擬器](#) 中執行此 Bot 的結果。



使用多個瀑布建立交談

您可以在交談內使用多個瀑布來定義 Bot 所需的任何交談結構。例如，您可能會使用一個瀑布來管理交談流程，並使用其他瀑布向使用者收集資訊。每個瀑布都封裝在一則對話中，可以透過呼叫 `session.beginDialog` 函式來叫用。

下列程式碼範例示範如何在交談中使用多個瀑布。`greetings` 交談內的瀑布會管理交談的流程，而 `askName` 交談內的瀑布則會向使用者收集資訊。

```
bot.dialog('greetings', [
  function (session) {
    session.beginDialog('askName');
  },
  function (session, results) {
    session.endDialog('Hello %s!', results.response);
  }
]);
bot.dialog('askName', [
  function (session) {
    builder.Prompts.text(session, 'Hi! What is your name?');
  },
  function (session, results) {
    session.endDialogWithResult(results);
  }
]);
```

推進瀑布

瀑布會依照陣列中函式的定義順序逐步進行。瀑布內的第一個函式可以接收傳遞給對話的引數。瀑布內的任何函式都可以使用 `next` 函式，以繼續進行下一個步驟，而不提示使用者提供輸入。

下列程式碼範例示範如何在對話中使用 `next` 函式，以逐步引導使用者完成提供其使用者設定檔資訊的程序。在瀑布的每個步驟中，Bot 會提示使用者輸入一項資訊（若必要），而且瀑布的後續步驟會處理使用者的回應（若有）。`ensureProfile` 對話中瀑布的最後一個步驟會結束該對話，並將完成的設定檔資訊傳回給呼叫端對話，該對話接著會將個人化問候語傳送給使用者。

```

var inMemoryStorage = new builder.MemoryBotStorage();

// This bot ensures user's profile is up to date.
var bot = new builder.UniversalBot(connector, [
    function (session) {
        session.beginDialog('ensureProfile', session.userData.profile);
    },
    function (session, results) {
        session.userData.profile = results.response; // Save user profile.
        session.send(`Hello ${session.userData.profile.name}! I love ${session.userData.profile.company}`);
    }
]).set('storage', inMemoryStorage); // Register in-memory storage

bot.dialog('ensureProfile', [
    function (session, args, next) {
        session.dialogData.profile = args || {}; // Set the profile or create the object.
        if (!session.dialogData.profile.name) {
            builder.Prompts.text(session, "What's your name?");
        } else {
            next(); // Skip if we already have this info.
        }
    },
    function (session, results, next) {
        if (results.response) {
            // Save user's name if we asked for it.
            session.dialogData.profile.name = results.response;
        }
        if (!session.dialogData.profile.company) {
            builder.Prompts.text(session, "What company do you work for?");
        } else {
            next(); // Skip if we already have this info.
        }
    },
    function (session, results) {
        if (results.response) {
            // Save company name if we asked for it.
            session.dialogData.profile.company = results.response;
        }
        session.endDialogWithResult({ response: session.dialogData.profile });
    }
]);

```

NOTE

此範例使用兩個不同的資料包來儲存資料：`dialogData` 和 `userData`。如果 Bot 分散在多個計算節點上，則瀑布的每個步驟都可由不同的節點加以處理。如需儲存 Bot 資料的詳細資訊，請參閱[管理狀態資料](#)。

結束瀑布

使用瀑布建立的對話必須明確地結束，否則 Bot 會無限次地重複瀑布。您可以使用下列其中一個方法來結束瀑布：

- `session.endDialog`：如果沒有資料可傳回給呼叫端對話，請使用此方法來結束瀑布。
- `session.endDialogWithResult`：如果有資料可傳回給呼叫端對話，請使用此方法來結束瀑布。傳回的 `response` 引數可以是 JSON 物件，或是任何 JavaScript 基本資料類型。例如：

```

session.endDialogWithResult({
    response: { name: session.dialogData.name, company: session.dialogData.company }
});

```

- `session.endConversation` :如果瀑布的結尾代表交談的結尾, 請使用此方法來結束交談。

作為使用這三種方法其中之一來結束瀑布的替代方式, 您可以將 `endConversationAction` 觸發程序附加到對話中。例如：

```
bot.dialog('dinnerOrder', [
    //...waterfall steps...
    // Last step
    function(session, results){
        if(results.response){
            session.dialogData.room = results.response;
            var msg = `Thank you. Your order will be delivered to room #${session.dialogData.room}`;
            session.endConversation(msg);
        }
    }
])
.endConversationAction(
    "endOrderDinner", "Ok. Goodbye.",
    {
        matches: /^cancel$|^goodbye$/i,
        confirmPrompt: "This will cancel your order. Are you sure?"
    }
);
);
```

後續步驟

使用瀑布, 您可以透過「提示」從使用者收集資訊。讓我們深入了解您可以提示使用者提供輸入的方式。

[提示使用者提供輸入](#)

提示使用者輸入

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

適用於 Node.js 的 Bot Framework SDK 提供一組內建的提示以簡化收集使用者的輸入。

A 提示每當 Bot 需要使用者輸入會使用。您可以透過像是瀑布般連接的提示來使用提示要求使用者輸入一個系列。您可以搭配提示使用[瀑布](#)來協助您在在 Bot [管理對話工作流程](#)。

文章將協助您瞭解提示的運作方式，以及如何使用它們來向使用者收集資訊。

提示與回應

當您需要使用者輸入時，您可以傳送提示，等候使用者回應輸入，然後處理後並回傳回應給使用者。

下列程式碼範例會提示使用者輸入其名稱，並以問候訊息加以回應。

```
bot.dialog('greetings', [
    // Step 1
    function (session) {
        builder.Prompts.text(session, 'Hi! What is your name?');
    },
    // Step 2
    function (session, results) {
        session.endDialog(`Hello ${results.response}!`);
    }
]);
```

使用此基本建構，您可以透過增加 Bot 所要求的提示與回應來建構您的對話工作流程。

提示結果

內建的提示會在傳回使用者的回應中 `results.response` 欄位作為[對話](#)應用。JSON 物件中，回應會回傳 `results.response.entity` 欄位。任何一種[對話的處理常式](#)都可以接收提示的結果。一旦 Bot 收到回應時，它可以使用它或將它藉由呼叫 `session.endDialogWithResult` 方法回傳給呼叫對話方塊。

下列程式碼範例顯示如何使用 `session.endDialogWithResult` 方法來將提示結果回傳呼叫對話方塊。在此範例中，`greetings` 對話方塊使用 `askName` 對話方塊回傳的提示結果依照名稱對用戶進行問候。

```
// Ask the user for their name and greet them by name.
bot.dialog('greetings', [
    function (session) {
        session.beginDialog('askName');
    },
    function (session, results) {
        session.endDialog(`Hello ${results.response}!`);
    }
]);
bot.dialog('askName', [
    function (session) {
        builder.Prompts.text(session, 'Hi! What is your name?');
    },
    function (session, results) {
        session.endDialogWithResult(results);
    }
]);

```

提示類型

適用於 Node.js 的 Bot Framework SDK 包含了許多不同類型的內建提示。

提示類型	說明
Prompts.text	要求使用者輸入文字字串。
Prompts.confirm	要求使用者確認動作。
Prompts.number	要求使用者輸入數字。
Prompts.datetime	要求使用者輸入時間或日期/時間。
Prompts.choice	要求使用者選擇選項清單。
Prompts.attachment	要求使用者上傳圖片或影片。

下列各區段提供關於各類型提示的其他詳細資料。

Prompts.text

使用 [Prompts.text\(\)](#) 方法來要求使用者提供的文字字串。提示會以 [IPromptTextResult](#) 回傳使用者的回應。

```
builder.Prompts.text(session, "What is your name?");
```

Prompts.confirm

使用 [Prompts.confirm\(\)](#) 方法來要求使用者確認動作並回應是/否。提示會以 [IPromptConfirmResult](#) 回傳使用者的回應。

```
builder.Prompts.confirm(session, "Are you sure you wish to cancel your order?");
```

Prompts.number

使用 [Prompts.number\(\)](#) 方法來要求使用者提供數字。提示會以 [IPromptConfirmResult](#) 回傳使用者的回應。

```
builder.Prompts.number(session, "How many would you like to order?");
```

Prompts.time

使用 [Prompts.time\(\)](#) 方法來要求使用者提供時間或是日期/時間。提示會以 [IPromptTimeResult](#) 回傳使用者的回應。此架構使用 [Chrono](#) 資料庫來剖析使用者的回應，並支援（例如「再 5 分鐘」）的相對回應和非相對回應（例如「6 月 6 號下午 2 點」）。

[Results.response](#) 欄位中，它代表使用者的回應，包含指定日期和時間的實體物件。若要解決日期和時間變成 JavaScript [Date](#) 物件，請使用 [EntityRecognizer.resolveTime\(\)](#) 方法。

TIP

使用者輸入的時間會根據 Bot 伺服器的時區時間轉換成 UTC 時間。由於伺服器可能位於與使用者不同的時區，因此，請務必將時區納入考量。若要轉換成使用者當地的日期和時間，可以考慮詢問使用者位於哪個時區。

```
bot.dialog('createAlarm', [
    function (session) {
        session.dialogData.alarm = {};
        builder.Prompts.text(session, "What would you like to name this alarm?");
    },
    function (session, results, next) {
        if (results.response) {
            session.dialogData.name = results.response;
            builder.Prompts.time(session, "What time would you like to set an alarm for?");
        } else {
            next();
        }
    },
    function (session, results) {
        if (results.response) {
            session.dialogData.time = builder.EntityRecognizer.resolveTime([results.response]);
        }

        // Return alarm to caller
        if (session.dialogData.name && session.dialogData.time) {
            session.endDialogWithResult({
                response: { name: session.dialogData.name, time: session.dialogData.time }
            });
        } else {
            session.endDialogWithResult({
                resumed: builder.ResumeReason.notCompleted
            });
        }
    }
]);
```

Prompts.choice

使用 [\[Prompts.choice\(\)\] PromptsChoice](#) 方法來要求使用者從選項清單中選擇。使用者可以透過輸入他們所選擇的選項相關的號碼來傳達其選取項目，或輸入他們所選擇的選項名稱。支援的選項名稱的完整名稱或部分名稱。在提示字元以 [IPromptChoiceResult](#) 回傳使用者的回應。

為指定呈現給使用者的清單樣式，請設定 [IPromptOptions.listStyle](#) 屬性。以下表格顯示 [ListStyle](#) 此屬性的列舉值。

[ListStyle](#) 列舉值如下所示：

索引	NAME	說明
0	None	轉譯任何清單。當列表作為提示的一部份時使用。

索引	NAME	說明
1	內嵌	選項呈現為「1。紅色, 2。綠色或 3。藍色」。
2	list	選項呈現為編號的清單。
3	按鈕	選項會顯示為支援按鈕通道的按鈕。其他通道會將其轉譯為文字。
4	自動	樣式會自動根據通道和選項數量進行選擇。

您可以從這個列舉 `builder` 物件存取，或您可以提供索引選擇 `ListStyle`。例如，下列程式碼片段中的兩個敘述句都完成同樣的工作。

```
// ListStyle passed in as Enum
builder.Prompts.choice(session, "Which color?", "red|green|blue", { listStyle: builder.ListStyle.button });

// ListStyle passed in as index
builder.Prompts.choice(session, "Which color?", "red|green|blue", { listStyle: 3 });
```

若要指定選項的清單，您可以使用直立線符號分隔 (|) 字串、字串陣列或物件對應。

直立線符號分隔的字串：

```
builder.Prompts.choice(session, "Which color?", "red|green|blue");
```

字串陣列：

```
builder.Prompts.choice(session, "Which color?", ["red", "green", "blue"]);
```

物件對應：

```
var salesData = {
    "west": {
        units: 200,
        total: "$6,000"
    },
    "central": {
        units: 100,
        total: "$3,000"
    },
    "east": {
        units: 300,
        total: "$9,000"
    }
};

bot.dialog('getSalesData', [
    function (session) {
        builder.Prompts.choice(session, "Which region would you like sales for?", salesData);
    },
    function (session, results) {
        if (results.response) {
            var region = salesData[results.response.entity];
            session.send(`We sold ${region.units} units for a total of ${region.total}.`);
        } else {
            session.send("OK");
        }
    }
]);

```

Prompts.attachment

使用 [Prompts.attachment\(\)](#) 方法來要求使用者上傳像是影像或影片檔。提示會以 [IPromptAttachmentResult](#) 回傳使用者的回應。

```
builder.Prompts.attachment(session, "Upload a picture for me to transform.");
```

後續步驟

既然您已經知道如何逐步執行透過使用者瀑布圖及提示他們提供資訊，讓我們看看該如何協助您更好管理對話工作流程。

[管理對話工作流程](#)

使用對話方塊管理交談流程

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

管理對話流程是建置 Bot 的必要工作。Bot 必須能夠從容地執行核心工作，並在被中斷的情況下做出適當的因應措施。透過適用於 Node.js 的 Bot Framework SDK，您可以使用對話來管理交談流程。

對話類似於程式中的函式。它通常是設計來執行特定作業，並可視需要隨時叫用。您可以將多個對話 (dialog) 鏈結在一起，以處理幾乎任何您想要讓 Bot 處理的對話 (conversation) 流程。適用於 Node.js 的 Bot Framework SDK 包含[提示](#)和[瀑布圖](#)等內建功能，可協助您管理交談流程。

本文會提供一系列的範例來說明管理簡單和複雜對話 (conversation) 流程的方式，其中您的 Bot 可使用對話 (dialog) 來處理被中斷的情況，並從容地繼續該流程。範例均以下列案例為基礎：

1. 您的 Bot 將協助進行晚餐訂位事宜。
2. 您的 Bot 可以在訂位期間隨時處理 "Help" (說明) 要求。
3. 您的 Bot 可以針對目前的訂位步驟處理內容相關的 "Help" (說明)。
4. 您的 Bot 可以處理多個對話主題。

使用瀑布圖管理對話流程

[瀑布圖](#)是一個對話，可讓 Bot 輕鬆地引導使用者執行一系列的工作。在此範例中，訂位 Bot 會詢問使用者一連串的問題，讓 Bot 能夠處理訂位要求。Bot 將提示使用者提供下列資訊：

1. 訂位日期和時間
2. 用餐人數
3. 訂位者的姓名

下列程式碼範例會示範如何使用瀑布圖來引導使用者完成一系列的提示。

```

var inMemoryStorage = new builder.MemoryBotStorage();

// This is a dinner reservation bot that uses a waterfall technique to prompt users for input.
var bot = new builder.UniversalBot(connector, [
  function (session) {
    session.send("Welcome to the dinner reservation.");
    builder.Prompts.time(session, "Please provide a reservation date and time (e.g.: June 6th at 5pm)");
  },
  function (session, results) {
    session.dialogData.reservationDate = builder.EntityRecognizer.resolveTime([results.response]);
    builder.Prompts.number(session, "How many people are in your party?");
  },
  function (session, results) {
    session.dialogData.partySize = results.response;
    builder.Prompts.text(session, "Whose name will this reservation be under?");
  },
  function (session, results) {
    session.dialogData.reservationName = results.response;

    // Process request and display reservation details
    session.send(`Reservation confirmed. Reservation details: <br/>Date/Time:
${session.dialogData.reservationDate} <br/>Party size: ${session.dialogData.partySize} <br/>Reservation name:
${session.dialogData.reservationName}`);
    session.endDialog();
  }
]).set('storage', inMemoryStorage); // Register in-memory storage

```

這個 Bot 的核心功能會發生在預設對話中。預設對話是在建立 Bot 時定義：

```
var bot = new builder.UniversalBot(connector, [..waterfall steps..]);
```

此外，在這個建立過程中，您可以設定要使用的資料儲存體。例如，若要使用記憶體內部儲存體，您可以使用下列方式設定它：

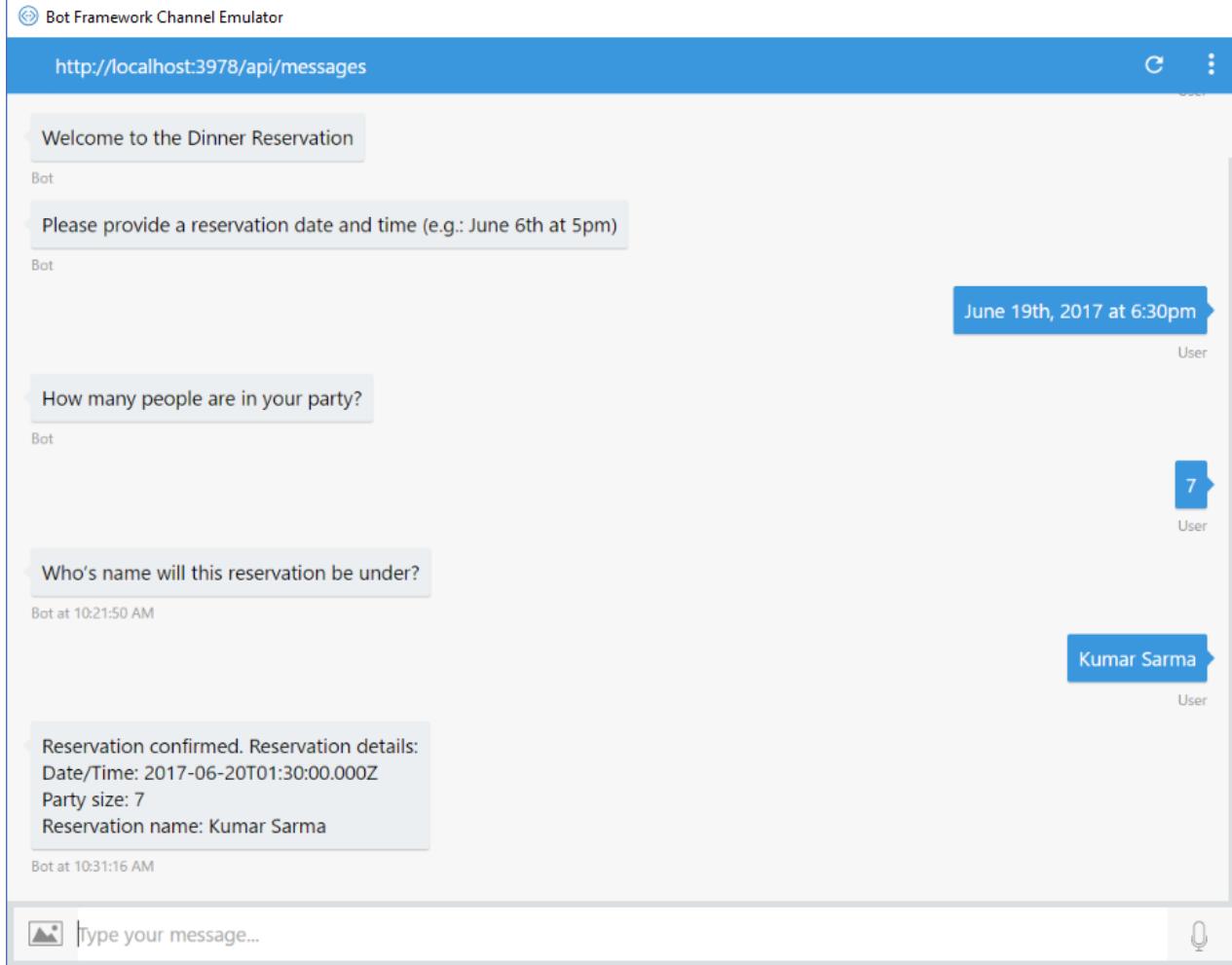
```

var inMemoryStorage = new builder.MemoryBotStorage();
var bot = new builder.UniversalBot(connector, [..waterfall steps..]).set('storage', inMemoryStorage); // 
Register in-memory storage

```

預設對話會建立為定義瀑布圖之步驟的函式陣列。此範例中有四個函式，因此瀑布圖會有四個步驟。每個步驟均會執行單一工作，且其結果會在下一個步驟中加以處理。此程序會繼續進行，直到最後一個步驟為止；該步驟將會確認訂位並結束對話。

下列螢幕擷取畫面會顯示在 [Bot Framework 模擬器](#) 中執行這個 Bot 的結果：



提示使用者輸入

此範例的每個步驟都會使用提示來要求使用者輸入。提示是一種特殊類型的對話，會要求使用者輸入，等候回應，並將該回應傳回至瀑布圖中的下一個步驟。如需可在 Bot 中使用之各種不同類型提示的相關資訊，請參閱[提示使用者輸入](#)。

在此範例中，Bot 會使用 `Prompts.text()` 來要求使用者以文字格式提供自由格式的回應。使用者可以使用任何文字來回應，而 Bot 必須決定要如何處理回應。`Prompts.time()` 會使用 [Chrono](#) (英文) 程式庫，從字串中剖析日期和時間資訊。這可讓您的 Bot 了解更多適用於指定日期與時間的自然語言。例如："June 6th, 2017 at 9pm" (2017 年 6 月 6 日下午 9 點)、"Today at 7:30pm" (今天下午 7:30)、"next monday at 6pm" (下週一下午 6 點) 等等。

TIP

使用者輸入的時間會根據裝載該 Bot 之伺服器的時區轉換成 UTC 時間。由於伺服器可能位於與使用者不同的時區，因此，請務必將時區納入考量。若要將日期和時間轉換成使用者的當地時間，請考慮詢問使用者位於哪個時區。

使用多個對話 (dialog) 管理對話 (conversation) 流程

管理對話 (conversation) 流程的另一個技術，是結合使用瀑布圖和多個對話 (dialog)。瀑布圖可讓您在對話 (dialog) 中鏈結多個函式，而對話 (dialog) 可讓您將對話 (conversation) 細分為可隨時重複使用的較小功能片段。

例如，讓我們以晚餐訂位 Bot 為例。下列程式碼範例會已重新撰寫上一個範例，以使用瀑布圖和多個對話。

```

var inMemoryStorage = new builder.MemoryBotStorage();

// This is a dinner reservation bot that uses multiple dialogs to prompt users for input.
var bot = new builder.UniversalBot(connector, [
    function (session) {
        session.send("Welcome to the dinner reservation.");
        session.beginDialog('askForDateTime');
    },
    function (session, results) {
        session.dialogData.reservationDate = builder.EntityRecognizer.resolveTime([results.response]);
        session.beginDialog('askForPartySize');
    },
    function (session, results) {
        session.dialogData.partySize = results.response;
        session.beginDialog('askForReserverName');
    },
    function (session, results) {
        session.dialogData.reservationName = results.response;

        // Process request and display reservation details
        session.send(`Reservation confirmed. Reservation details: <br/>Date/Time:
${session.dialogData.reservationDate} <br/>Party size: ${session.dialogData.partySize} <br/>Reservation name:
${session.dialogData.reservationName}`);
        session.endDialog();
    }
]).set('storage', inMemoryStorage); // Register in-memory storage

// Dialog to ask for a date and time
bot.dialog('askForDateTime', [
    function (session) {
        builder.Prompts.time(session, "Please provide a reservation date and time (e.g.: June 6th at 5pm)");
    },
    function (session, results) {
        session.endDialogWithResult(results);
    }
]);

// Dialog to ask for number of people in the party
bot.dialog('askForPartySize', [
    function (session) {
        builder.Prompts.text(session, "How many people are in your party?");
    },
    function (session, results) {
        session.endDialogWithResult(results);
    }
]);

// Dialog to ask for the reservation name.
bot.dialog('askForReserverName', [
    function (session) {
        builder.Prompts.text(session, "Who's name will this reservation be under?");
    },
    function (session, results) {
        session.endDialogWithResult(results);
    }
]);

```

執行此 Bot 的結果會與上一個僅使用瀑布圖的 Bot 完全相同。不過，就程式設計方面而言，它們有兩個主要差異：

1. 預設對話 (dialog) 是專門用來管理對話 (conversation) 的流程。
2. 針對對話 (conversation) 之每個步驟的工作，是由個別的對話 (dialog) 所管理。在此範例中，由於 Bot 需要三種資訊，因此它會提示使用者三次。每個提示現在都會包含於自己的對話中。

透過使用此技術，您可以將對話流程從工作邏輯中分隔開來。這可讓不同的對話 (conversation) 流程在必要的情況下重複使用該對話 (dialog)。

回應使用者輸入

在引導使用者完成一系列工作的過程中，如果使用者有問題，或者想要在回答之前要求其他資訊，您該如何處理那些要求？例如，不論使用者處於對話的哪個位置，當使用者輸入 "Help" (說明)、"Support" (支援) 或 "Cancel" (取消) 時，Bot 該如何回應？如果使用者想要取得關於某個步驟的其他資訊，該怎麼辦？如果使用者改變心意，並想要放棄目前的工作以開始全然不同的工作，會發生什麼事？

適用於 Node.js 的 Bot Framework SDK 可讓 Bot 接聽全域內容之內，或是位於目前對話範圍中的區域內容之內的特定輸入。這些輸入稱為[動作](#)，其可讓 Bot 根據 `matches` 子句來接聽使用者輸入。Bot 必須決定該如何回應特定的使用者輸入。

處理全域動作

如果您想要 Bot 能夠在對話中的任何步驟處理動作，請使用 `triggerAction`。觸發程序可讓 Bot 在輸入符合指定字詞時叫用特定的對話。例如，如果您想要支援全域的 "Help" (說明) 選項，您可以建立說明對話，並附加能接聽符合 "Help" (說明) 之輸入的 `triggerAction`。

下列程式碼範例會示範如何將 `triggerAction` 附加到對話，以指定在使用者輸入 "Help" (說明) 時應該叫用對話。

```
// The dialog stack is cleared and this dialog is invoked when the user enters 'help'.
bot.dialog('help', function (session, args, next) {
    session.endDialog("This is a bot that can help you make a dinner reservation. <br/>Please say 'next' to
    continue");
})
.triggerAction({
    matches: /^help$/i,
});
```

根據預設，當 `triggerAction` 執行時，系統會清除對話堆疊，而觸發的對話將會變成新的預設對話。在此範例中，當 `triggerAction` 執行時，系統會清除對話堆疊，接著會將 `help` 對話新增至堆疊以作為新的預設對話。如果這不是您想要的行為，您可以將 `onSelectAction` 選項新增至 `triggerAction`。`onSelectAction` 選項可讓 Bot 在不清除對話堆疊的情況下開始新的對話 (dialog)，這可將對話 (conversation) 暫時重新導向，並於稍後再從相同的位置繼續。

下列程式碼範例會示範如何使用 `onSelectAction` 選項搭配 `triggerAction`，來將 `help` 對話新增至現有的對話堆疊 (且不會清除對話堆疊)。

```
bot.dialog('help', function (session, args, next) {
    session.endDialog("This is a bot that can help you make a dinner reservation. <br/>Please say 'next' to
    continue");
})
.triggerAction({
    matches: /^help$/i,
    onSelectAction: (session, args, next) => {
        // Add the help dialog to the dialog stack
        // (override the default behavior of replacing the stack)
        session.beginDialog(args.action, args);
    }
});
```

在此範例中，`help` 對話 (dialog) 會在使用者輸入 "Help" (說明) 時取得對話 (conversation) 的控制權。由於 `triggerAction` 包含 `onSelectAction` 選項，因此系統會將 `help` 對話推送至現有的對話堆疊，且不會清除該堆疊。當 `help` 對話 (dialog) 結束時，系統就會從對話 (dialog) 堆疊中移除它，而對話 (conversation) 則會從 `help` 命令先前中斷它的步驟繼續。

處理內容相關的動作

在上述範例中，如果使用者在對話 (conversation) 的任何步驟輸入 "Help" (說明)，即會叫用 `help` 對話 (dialog)。在此情況下，該對話只能提供一般的說明指導方針，因為使用者的說明要求並沒有針對任何特定內容。但是，如果

使用者想要針對對話的特定步驟要求說明，該怎麼辦？在該情況下，必須在目前對話的內容內觸發 `help` 對話。

例如，讓我們再以晚餐訂位 Bot 為例。在詢問用餐人數時，如果使用者想要知道用餐人數上限，該怎麼辦？若要處理這種情況，您可以將 `beginDialogAction` 附加至 `askForPartySize` 對話，以接聽使用者輸入 "Help" (說明)。

下列程式碼範例會示範如何使用 `beginDialogAction`，將內容相關的說明附加至對話。

```
// Dialog to ask for number of people in the party
bot.dialog('askForPartySize', [
    function (session) {
        builder.Prompts.text(session, "How many people are in your party?");
    },
    function (session, results) {
        session.endDialogWithResult(results);
    }
])
.beginDialogAction('partySizeHelpAction', 'partySizeHelp', { matches: /^help$/i });

// Context Help dialog for party size
bot.dialog('partySizeHelp', function(session, args, next) {
    var msg = "Party size help: Our restaurant can support party sizes up to 150 members.";
    session.endDialog(msg);
})
```

在此範例中，每當使用者輸入 "Help" (說明) 時，Bot 會將 `partySizeHelp` 對話推送至堆疊。該對話會將說明訊息傳送給使用者，然後結束對話，並將控制權還給 `askForPartySize` 對話，而該對話將會重新提示使用者輸入用餐人數。

請務必注意，只有在使用者位於 `askForPartySize` 對話時，才會執行此內容相關說明。否則，系統將會執行來自 `triggerAction` 的一般說明訊息。換句話說，區域 `matches` 子句的優先順序，一律會優先於全域 `matches` 子句。例如，如果 `beginDialogAction` 與 `help` 相符，系統便不會執行 `triggerAction` 中與 `help` 相符的項目。如需詳細資訊，請參閱[動作優先順序](#)。

變更對話的主題

根據預設，執行 `triggerAction` 會清除對話 (dialog) 堆疊並重設對話 (conversation)，並從指定的對話 (dialog) 開始。當您的 Bot 需要從一個對話主題切換至另一個時 (例如，如果使用者在進行晚餐訂位時，決定要改成使用旅館的送餐服務並直接點晚餐)，通常會更適合使用此行為。

下列範例是建置於上一個範例之上，但會讓 Bot 允許使用者進行晚餐訂位，或是使用送餐服務。在這個 Bot 中，預設對話是一則問候語對話，其會向使用者顯示兩個選項：`Dinner Reservation` 和 `Order Dinner`。

```
var inMemoryStorage = new builder.MemoryBotStorage();

// This bot enables users to either make a dinner reservation or order dinner.
var bot = new builder.UniversalBot(connector, function(session){
    var msg = "Welcome to the reservation bot. Please say `Dinner Reservation` or `Order Dinner`";
    session.send(msg);
}).set('storage', inMemoryStorage); // Register in-memory storage
```

上一個範例之預設對話中的晚餐訂位邏輯，現在已是獨立的 `dinnerReservation` 對話。`dinnerReservation` 的流程會與稍早所討論的多個對話版本保持一致。唯一的差別在於對話具有附加的 `triggerAction`。請注意，在此版本中，`confirmPrompt` 會在叫用新對話 (dialog) 之前，要求使用者確認他們想要變更對話 (conversation) 的主題。在這類案例中，包含 `confirmPrompt` 是一個很好的作法，因為在清除對話 (dialog) 堆疊之後，系統會將使用者導向新的對話 (conversation) 主題，從而放棄先前所進行的對話 (conversation) 主題。

```

// This dialog helps the user make a dinner reservation.
bot.dialog('dinnerReservation', [
    function (session) {
        session.send("Welcome to the dinner reservation.");
        session.beginDialog('askForDateTime');
    },
    function (session, results) {
        session.dialogData.reservationDate = builder.EntityRecognizer.resolveTime([results.response]);
        session.beginDialog('askForPartySize');
    },
    function (session, results) {
        session.dialogData.partySize = results.response;
        session.beginDialog('askForReserverName');
    },
    function (session, results) {
        session.dialogData.reservationName = results.response;

        // Process request and display reservation details
        session.send(`Reservation confirmed. Reservation details: <br/>Date/Time:
${session.dialogData.reservationDate} <br/>Party size: ${session.dialogData.partySize} <br/>Reservation name:
${session.dialogData.reservationName}`);
        session.endDialog();
    }
])
.triggerAction({
    matches: /^dinner reservation$/i,
    confirmPrompt: "This will cancel your current request. Are you sure?"
});

```

第二個對話 (conversation) 主題會使用瀑布圖定義於 `orderDinner` 對話 (dialog) 中。這個對話單純會顯示晚餐菜單，並在使用者點餐之後提示使用者提供房間號碼。`triggerAction` 會附加至該對話 (dialog)，以指定系統應在使用者輸入 "order dinner" (點晚餐) 時叫用它，並在使用者表示想要變更對話 (conversation) 主題時，確保系統會提示使用者確認其選項。

```

// This dialog help the user order dinner to be delivered to their hotel room.
var dinnerMenu = {
    "Potato Salad - $5.99": {
        Description: "Potato Salad",
        Price: 5.99
    },
    "Tuna Sandwich - $6.89": {
        Description: "Tuna Sandwich",
        Price: 6.89
    },
    "Clam Chowder - $4.50": {
        Description: "Clam Chowder",
        Price: 4.50
    }
};

bot.dialog('orderDinner', [
    function(session){
        session.send("Lets order some dinner!");
        builder.Prompts.choice(session, "Dinner menu:", dinnerMenu);
    },
    function (session, results) {
        if (results.response) {
            var order = dinnerMenu[results.response.entity];
            var msg = `You ordered: ${order.Description} for a total of $$${order.Price}.`;
            session.dialogData.order = order;
            session.send(msg);
            builder.Prompts.text(session, "What is your room number?");
        }
    },
    function(session, results){
        if(results.response){
            session.dialogData.room = results.response;
            var msg = `Thank you. Your order will be delivered to room ##${session.dialogData.room}`;
            session.endDialog(msg);
        }
    }
])
.triggerAction({
    matches: /^order dinner$/i,
    confirmPrompt: "This will cancel your order. Are you sure?"
});

```

在使用者開始對話並選取 `Dinner Reservation` 或 `Order Dinner` 之後，他們隨時都可能會改變心意。例如，如果使用者在預訂晚餐過程中輸入「訂購晚餐」，Bot 將藉由說出「這樣將會取消您目前的要求。您確定嗎？」來進行確認。如果使用者輸入 "no" (否)，則會取消該要求，而使用者可以繼續進行晚餐訂位程序。如果使用者輸入 "yes" (是)，Bot 將清除對話 (dialog) 堆疊，並將對話 (conversation) 的控制權轉給 `orderDinner` 對話 (dialog)。

結束對話

在上述範例中，對話是藉由使用 `session.endDialog` 或 `session.endDialogWithResult` 來關閉；這兩者均會結束對話，從堆疊中移除該對話，並將控制權還給呼叫對話。在使用者已抵達對話結尾的情況下，您應該使用 `session.endConversation` 來指出該對話已結束。

`session.endConversation` (英文) 方法會結束對話，並能選擇性地傳送訊息給使用者。例如，上一個範例中的 `orderDinner` 對話 (dialog) 可以使用 `session.endConversation` 來結束對話 (conversation)，如下列程式碼範例所示。

```
bot.dialog('orderDinner', [
    //...waterfall steps...
    // Last step
    function(session, results){
        if(results.response){
            session.dialogData.room = results.response;
            var msg = `Thank you. Your order will be delivered to room #${session.dialogData.room}`;
            session.endConversation(msg);
        }
    }
]);
```

呼叫 `session.endConversation` 將會清除對話 (dialog) 堆疊並重設 `session.conversationData` (英文) 儲存體來結束對話 (conversation)。如需資料儲存體的詳細資訊，請參閱[管理狀態資料](#)。

當使用者完成設計 Bot 以進行的對話流程時，呼叫 `session.endConversation` 是一件合乎邏輯的事。您也可以在使用者於對話期間輸入 "cancel" (取消) 或 "goodbye" (再見) 的情況下，使用 `session.endConversation` 來結束對話。若要這樣做，只需將 `endConversationAction` 附加至對話，並讓此觸發程序接聽符合 "cancel" (取消) 或 "goodbye" (再見) 的輸入。

下列程式碼範例會示範如何將 `endConversationAction` 附加至對話 (dialog)，來在使用者輸入 "cancel" (取消) 或 "goodbye" (再見) 時結束對話 (conversation)。

```
bot.dialog('dinnerOrder', [
    //...waterfall steps...
])
.endConversationAction(
    "endOrderDinner", "Ok. Goodbye.",
    {
        matches: /^cancel$|^goodbye$/i,
        confirmPrompt: "This will cancel your order. Are you sure?"
    }
);
```

由於使用 `session.endConversation` 或 `endConversationAction` 來結束對話 (conversation) 將會清除對話 (dialog) 堆疊，並強制使用者從頭開始，因此您應該包含 `confirmPrompt` 以確保使用者真的想要執行該動作。

後續步驟

在本文中，您已探索管理具有循序本質之對話的方式。如果您想要在對話 (conversation) 中重複某個對話 (dialog) 或使用迴圈模式，該怎麼辦？讓我們來看看如何藉由取代堆疊上的對話來達成該目的。

[取代對話](#)

取代對話方塊

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

當您需要在對話過程中驗證使用者輸入或者重複動作時，取代對話方塊的能力很有用。透過適用於 Node.js 的 Bot Framework SDK，您可藉由使用 `session.replaceDialog` 方法來取代對話。這個方法可讓您結束目前的對話，不用傳回給來電者就可以新的對話加以取代。

建立自訂提示來驗證輸入

適用於 Node.js 的 Bot Framework SDK 包括某些類型提示 (例如 `Prompts.time` 和 `Prompts.choice`) 的輸入驗證。若要驗證您為了回應 `Prompts.text` 所收到的文字輸入，您必須建立自己的驗證邏輯和自訂提示。

如果驗證必須符合您定義的特定值、模式、範圍或準則，建議您驗證輸入。如果輸入驗證失敗，Bot 可以藉由使用 `session.replaceDialog` 方法，再次提示使用者該資訊。

下列程式碼範例示範如何建立自訂提示，來驗證使用者輸入的電話號碼。

```
// This dialog prompts the user for a phone number.  
// It will re-prompt the user if the input does not match a pattern for phone number.  
bot.dialog('phonePrompt', [  
    function (session, args) {  
        if (args && args.reprompt) {  
            builder.Prompts.text(session, "Enter the number using a format of either: '(555) 123-4567' or  
            '555-123-4567' or '5551234567'");  
        } else {  
            builder.Prompts.text(session, "What's your phone number?");  
        }  
    },  
    function (session, results) {  
        var matched = results.response.match(/\d+/g);  
        var number = matched ? matched.join('') : '';  
        if (number.length == 10 || number.length == 11) {  
            session.userData.phoneNumber = number; // Save the number.  
            session.endDialogWithResult({ response: number });  
        } else {  
            // Repeat the dialog  
            session.replaceDialog('phonePrompt', { reprompt: true });  
        }  
    }  
]);
```

在此範例中，系統一開始會提示使用者提供其電話號碼。驗證邏輯會使用規則運算式，該規則運算式會比對使用者輸入的位數範圍。如果輸入包含 10 或 11 位數，則該數字會在回應中傳回。否則，系統會執行 `session.replaceDialog` 方法以重複 `phonePrompt` 對話方塊，提示使用者再次輸入；這次關於預期的輸入格式，會提供更精確的指引。

當您呼叫 `session.replaceDialog` 方法時，您會指定要重複的對話方塊名稱和引數清單。在此範例中，引數清單包含 `{ reprompt: true }`，它會讓 Bot 根據是初始提示還是重新提示，提供不同的提示訊息，但是您可以指定 Bot 可能需要的任何引數。

重複動作

有時候在對話當中您會想要重複對話方塊，讓使用者完成特定動作多次。例如，如果您的 Bot 提供各種服務，您可能會在一開始顯示服務的選單，引導使用者完成要求服務的程序，然後再次顯示服務的選單，藉此讓使用者要求另一個服務。若要達到此目的，您可以使用 `session.replaceDialog` 方法，以再次顯示服務的選單，而不是使用 `'session.endConversation'` 方法來結束對話。

下列範例示範如何使用 `session.replaceDialog` 方法來實作這種案例。首先，定義服務的選單：

```
// Main menu
var menuItems = {
    "Order dinner": {
        item: "orderDinner"
    },
    "Dinner reservation": {
        item: "dinnerReservation"
    },
    "Schedule shuttle": {
        item: "scheduleShuttle"
    },
    "Request wake-up call": {
        item: "wakeUpCall"
    },
}
```

`mainMenu` 對話方塊是由預設對話方塊叫用，因此會在對話開始時向使用者顯示選單。此外，`triggerAction` 會附加至 `mainMenu` 對話，每當使用者輸入「主選單」時呈現選單。當使用者看到選單並選取一個選項時，系統會使用 `session.beginDialog` 方法叫用與使用者選項對應的對話方塊。

```
var inMemoryStorage = new builder.MemoryBotStorage();

// This is a reservation bot that has a menu of offerings.
var bot = new builder.UniversalBot(connector, [
    function(session){
        session.send("Welcome to Contoso Hotel and Resort.");
        session.beginDialog("mainMenu");
    }
]).set('storage', inMemoryStorage); // Register in-memory storage

// Display the main menu and start a new request depending on user input.
bot.dialog("mainMenu", [
    function(session){
        builder.Prompts.choice(session, "Main Menu:", menuItems);
    },
    function(session, results){
        if(results.response){
            session.beginDialog(menuItems[results.response.entity].item);
        }
    }
])
.triggerAction({
    // The user can request this at any time.
    // Once triggered, it clears the stack and prompts the main menu again.
    matches: /^main menu$/i,
    confirmPrompt: "This will cancel your request. Are you sure?"
});
```

在此範例中，如果使用者選擇選項 1 來訂購晚餐送到他們的房間，系統會叫用 `orderDinner` 對話方塊，並且會逐步引導使用者進行訂購晚餐的程序。在此程序結束時，Bot 會確認訂單，並且使用 `session.replaceDialog` 方法再次顯示主選單。

```

// Menu: "Order dinner"
// This dialog allows user to order dinner to be delivered to their hotel room.
bot.dialog('orderDinner', [
    function(session){
        session.send("Lets order some dinner!");
        builder.Prompts.choice(session, "Dinner menu:", dinnerMenu);
    },
    function (session, results) {
        if (results.response) {
            var order = dinnerMenu[results.response.entity];
            var msg = `You ordered: %(Description)s for a total of $$ {order.Price}.`;
            session.dialogData.order = order;
            session.send(msg);
            builder.Prompts.text(session, "What is your room number?");
        }
    },
    function(session, results){
        if(results.response){
            session.dialogData.room = results.response;
            var msg = `Thank you. Your order will be delivered to room # ${results.response}.`;
            session.send(msg);
            session.replaceDialog("mainMenu"); // Display the menu again.
        }
    }
])
.reloadAction(
    "restartOrderDinner", "Ok. Let's start over.",
    {
        matches: /^start over$/i
    }
)
.cancelAction(
    "cancelOrder", "Type 'Main Menu' to continue.",
    {
        matches: /^cancel$/i,
        confirmPrompt: "This will cancel your order. Are you sure?"
    }
);

```

兩個觸發程序會附加至 `orderDinner` 對話方塊，讓使用者在訂購程序期間隨時「重頭開始」或「取消」訂購。

第一個觸發程序是 `reloadAction`，會讓使用者藉由傳送輸入「重頭開始」，再次重頭開始訂購程序。當觸發程序符合「從頭開始」的聲音時，`reloadAction` 會從頭重新啟動對話。

第二個觸發程序是 `cancelAction`，會讓使用者藉由傳送輸入「取消」，完全中止訂購程序。此觸發程序不會自動再次顯示主選單，而是改為傳送訊息，告訴使用者接下來透過說出「輸入 [主選單] 以繼續」的後續步驟。

對話方塊迴圈

在上述範例中，使用者在每次訂購只能選取一個項目。也就是說，如果使用者想要從選單訂購兩個項目，他們必須針對第一個項目完成整個訂購程序，然後針對第二個項目再次重複整個訂購程序。

下列範例示範如何藉由將晚餐選單重構至個別對話方塊中，來改善先前的 Bot。這麼做可讓 Bot 迴圈重複顯示晚餐選單，因此讓使用者可以在單一訂單內選擇多個項目。

首先，將 [結帳] 選項新增至選單中。這個選項可以讓使用者結束項目選取程序，並繼續結帳程序。

```
// The dinner menu
var dinnerMenu = {
    //...other menu items...
    "Check out": {
        Description: "Check out",
        Price: 0 // Order total. Updated as items are added to order.
    }
};
```

接下來，將訂購提示重構至它自己的對話方塊中，讓 Bot 能夠重複選單，並且讓使用者可將多個項目新增至訂單中。

```
// Add dinner items to the list by repeating this dialog until the user says `check out`.
bot.dialog("addDinnerItem", [
    function(session, args){
        if(args && args.reprompt){
            session.send("What else would you like to have for dinner tonight?");
        }
        else{
            // New order
            // Using the conversationData to store the orders
            session.conversationData.orders = new Array();
            session.conversationData.orders.push({
                Description: "Check out",
                Price: 0
            })
        }
        builder.Prompts.choice(session, "Dinner menu:", dinnerMenu);
    },
    function(session, results){
        if(results.response){
            if(results.response.entity.match(/^check out$/i)){
                session.endDialog("Checking out...");
            }
            else {
                var order = dinnerMenu[results.response.entity];
                session.conversationData.orders[0].Price += order.Price; // Add to total.
                var msg = `You ordered: ${order.Description} for a total of $$${order.Price}.`;
                session.send(msg);
                session.conversationData.orders.push(order);
                session.replaceDialog("addDinnerItem", { reprompt: true }); // Repeat dinner menu
            }
        }
    }
])
.reloadAction(
    "restartOrderDinner", "Ok. Let's start over.",
    {
        matches: /^start over$/i
    }
);
```

在此範例中，訂單是儲存在範圍為目前對話 `session.conversationData.orders` 的 Bot 資料存放區中。針對每筆新訂單，變數會以新陣列重新初始化，針對使用者選擇的每個項目，Bot 會將該項目新增至 `orders` 陣列中並且將價格新增至總計，總計會儲存在結帳的 `Price` 變數中。當使用者完成選取訂單的項目時，他們可以說「結帳」，然後繼續訂購程序的其餘部分。

NOTE

如需 Bot 資料儲存體的詳細資訊，請參閱[管理狀態資料](#)。

最後，請更新 `orderDinner` 對話方塊內瀑布圖的第二個步驟，以處理訂單與確認。

```
// Menu: "Order dinner"
// This dialog allows user to order dinner and have it delivered to their room.
bot.dialog('orderDinner', [
    function(session){
        session.send("Lets order some dinner!");
        session.beginDialog("addDinnerItem");
    },
    function (session, results) {
        if (results.response) {
            // Display itemize order with price total.
            for(var i = 1; i < session.conversationData.orders.length; i++){
                session.send(`You ordered: ${session.conversationData.orders[i].Description} for a total of
${session.conversationData.orders[i].Price}.`);
            }
            session.send(`Your total is: ${session.conversationData.orders[0].Price}`);
        }

        // Continue with the check out process.
        builder.Prompts.text(session, "What is your room number?");
    }
},
function(session, results){
    if(results.response){
        session.dialogData.room = results.response;
        var msg = `Thank you. Your order will be delivered to room #${results.response}`;
        session.send(msg);
        session.replaceDialog("mainMenu");
    }
}
])
//...attached triggers...
;
```

取消對話方塊

雖然 `session.replaceDialog` 方法可以用來將「目前」對話方塊取代為新對話方塊，但是它無法用來取代位於對話方塊堆疊底下遠處的對話方塊。若要取代對話方塊堆疊內，除了「目前」對話方塊以外的對話方塊，請改為使用 `session.cancelDialog` 方法。

`session.cancelDialog` 方法可用來結束對話方塊，不論對話方塊在對話方塊堆疊中的哪個位置，並選擇性地在其位置叫用一個新的對話方塊。若要呼叫 `session.cancelDialog` 方法，請指定要取消的對話方塊識別碼，並選擇性地指定要在其位置叫用的對話方塊識別碼。例如，此程式碼片段會取消 `orderDinner` 對話方塊，並且以 `mainMenu` 對話方塊來加以取代：

```
session.cancelDialog('orderDinner', 'mainMenu');
```

當呼叫 `session.cancelDialog` 方法時，對話方塊堆疊會向後搜尋，並且會取消第一個出現的對話方塊，導致該對話方塊及其子系對話方塊（如果有的話）從對話方塊堆疊中移除。接著控制權會交回給呼叫對話方塊，讓該對話方塊可以檢查 `results.resumed` 程式碼是否等於 `ResumeReason.notCompleted`，以偵測取消。

除了在呼叫 `session.cancelDialog` 方法時指定要取消的對話方塊識別碼以外，您也可以改為針對要取消的對話方塊，指定以零為基礎的索引，其中索引代表對話方塊在對話方塊堆疊中的位置。例如，下列程式碼片段會終止目前使用中的對話方塊（索引 = 0），並啟動 `mainMenu` 對話方塊來加以取代。系統會在對話方塊堆疊的 0 位置叫用 `mainMenu` 對話方塊，因此該對話方塊會成為新的預設對話方塊。

```
session.cancelDialog(0, 'mainMenu');
```

請考慮在上方 [對話方塊迴圈](#) 中所討論的範例。當使用者到達項目選取選單時，該對話方塊 (`addDinnerItem`) 是對話方塊堆疊中的第四個對話方塊：`[default dialog, mainMenu, orderDinner, addDinnerItem]`。您要如何讓使用者從 `addDinnerItem` 對話方塊中取消其訂單？如果您將 `cancelAction` 觸發程序附加至 `addDinnerItem` 對話方塊，只會讓使用者返回先前的對話方塊 (`orderDinner`)，這樣會將使用者送回 `addDinnerItem` 對話方塊。

這就是 `session.cancelDialog` 方法很有用的地方。以 [對話方塊迴圈範例](#) 開始，新增「取消訂單」作為晚餐選單內的明確選項。

```
// The dinner menu
var dinnerMenu = {
    //...other menu items...
    "Check out": {
        Description: "Check out",
        Price: 0      // Order total. Updated as items are added to order.
    },
    "Cancel order": { // Cancel the order and back to Main Menu
        Description: "Cancel order",
        Price: 0
    }
};
```

接著，更新 `addDinnerItem` 對話來檢查「取消訂單」要求。如果偵測到「取消」，則使用 `session.cancelDialog` 方法來取消預設對話方塊（亦即位於堆疊索引 0 的對話方塊），並叫用 `mainMenu` 對話方塊來加以取代。

```
// Add dinner items to the list by repeating this dialog until the user says `check out`.
bot.dialog("addDinnerItem", [
    //...waterfall steps...
    // Last step
    function(session, results){
        if(results.response){
            if(results.response.entity.match(/^check out$/i)){
                session.endDialog("Checking out...");
            }
            else if(results.response.entity.match(/^cancel$/i)){
                // Cancel the order and start "mainMenu" dialog.
                session.cancelDialog(0, "mainMenu");
            }
            else {
                //...add item to list and prompt again...
                session.replaceDialog("addDinnerItem", { reprompt: true }); // Repeat dinner menu.
            }
        }
    }
])
//...attached triggers...
;
```

藉由如此使用 `session.cancelDialog` 方法，您就可以實作 Bot 所需的任何對話流程。

後續步驟

如您所見，若要取代堆疊中的對話方塊，可使用各種動作來完成這項工作。動作讓您在管理對話流程方面有絕佳彈性。讓我們進一步看看動作，以及如何更妥善地處理使用者動作。

處理使用者動作

處理使用者動作

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

使用者通常會嘗試使用「help」、「cancel」或「start over」等關鍵字，來存取 Bot 內的某些功能。使用者會在對話中間這麼做，而 Bot 這時會預期不同的回應。藉由實作動作，您可以將 Bot 設計為會以更妥善的方式處理這類要求。處理常式會檢查使用者輸入中是否有您指定的關鍵字（例如，「help」、「cancel」或「start over」），並做出適當回應。



Do you confirm this order?

Yes

No

Help?



I see you have questions, anything specific you want me to help you with?

動作類型

下表列出您可以附加至對話方塊的動作類型。每個動作名稱的連結，會帶您前往可提供更多該動作詳細資料的章節。

動作	影響範圍	說明
triggerAction	全域	將動作繫結至對話，其會清除對話堆疊，並將自身推至堆疊底部。使用 <code>onSelectAction</code> 選項可覆寫這個預設行為。
customAction	全域	將自訂動作繫結至 Bot，其可處理資訊或採取動作，而不會影響對話方塊堆疊。使用 <code>onSelectAction</code> 選項可自訂此動作的功能。
beginDialogAction	內容相關	將動作繫結至對話，其會在觸發時開始另一段對話。系統會將開始中的對話方塊推至堆疊上，並於結束後消失。

動作	影響範圍	說明
reloadAction	內容相關	將動作繫結至對話方塊，其會在觸發時重新載入對話方塊。您可以使用 <code>reloadAction</code> 來處理使用者語句，像是「start over」。
cancelAction	內容相關	將動作繫結至對話方塊，其會在觸發時取消對話方塊。您可以使用 <code>cancelAction</code> 來處理使用者語句，像是「cancel」或「nevermind」。
endConversationAction	內容相關	將動作繫結至對話方塊，其會在觸發時結束與使用者的對話。您可以使用 <code>endConversationAction</code> 來處理使用者語句，像是「goodbye」。

動作優先順序

當 Bot 收到來自使用者的語句時，它會檢查對話方塊堆疊上所有已註冊動作的語句。比對會在堆疊中由上而下進行。如果沒有相符者，則會檢查所有全域動作 `matches` 的選項語句。

若您在不同內容中使用相同命令，動作優先順序就很重要。例如，您可以對 Bot 使用「Help」命令來獲得一般說明。您也可以針對每個工作使用「Help」，但這些「Help」命令會與每個工作的內容相關。如需詳細說明這一點的實用範例，請參閱[回應使用者輸入](#)。

將動作繫結至對話方塊

使用者說出的語句或按下按鈕的動作可以觸發與對話方塊相關聯的動作。如果指定「matches」，此動作將會聽取使用者所說出會觸發動作的字組或片語。`matches` 選項可採用規則運算式或辨識器名稱。若要將動作繫結至按下按鈕，請使用 `CardAction.dialogAction()` 來觸發動作。

動作皆「可鏈結」，以讓您將所需數量的動作繫結至對話方塊。

繫結 triggerAction

若要將 `triggerAction` 繫結至對話方塊，請執行下列動作：

```
// Order dinner.
bot.dialog('orderDinner', [
    //...waterfall steps...
])
// Once triggered, will clear the dialog stack and pushes
// the 'orderDinner' dialog onto the bottom of stack.
.triggerAction({
    matches: /^order dinner$/i
});
```

將 `triggerAction` 繫結至對話方塊便會向 Bot 註冊。一經觸發，`triggerAction` 就會清除對話方塊堆疊，並將已觸發的對話方塊推至堆疊上。此動作最適合用來切換對話主題，或用來允許使用者要求任意的獨立工作。如果您想要覆寫此動作會清除對話方塊堆疊的行為，請將 `onSelectAction` 選項新增至 `triggerAction`。

下列程式碼片段說明如何在不清除對話方塊堆疊的情況下，從全域內容提供一般說明。

```

bot.dialog('help', function (session, args, next) {
    //Send a help message
    session.endDialog("Global help menu.");
})
// Once triggered, will start a new dialog as specified by
// the 'onSelectAction' option.
.triggerAction({
    matches: /help$/i,
    onSelectAction: (session, args, next) => {
        // Add the help dialog to the top of the dialog stack
        // (override the default behavior of replacing the stack)
        session.beginDialog(args.action, args);
    }
});

```

在此情況下，`triggerAction` 會附加至 `help` 對話方塊本身（而不是 `orderDinner` 對話）。`onSelectAction` 選項可讓您在不清除對話堆疊的情況下，啟動這個對話方塊。這可讓您處理全域要求，例如「help」、「about」、「support」等等。請注意，`onSelectAction` 選項會明確地呼叫 `session.beginDialog` 方法來啟動已觸發的對話方塊。已觸發對話方塊的識別碼會透過 `args.action` 提供。請勿將對話方塊識別碼（例如：「help」）手工編碼至這個方法，否則，您可能會遇到執行階段錯誤。如果您想要對 `orderDinner` 工作本身觸發內容相關的說明訊息，則請考慮將 `beginDialogAction` 改為附加到 `orderDinner` 對話方塊。

繫結 `customAction`

不同於其他動作類型，`customAction` 並未定義任何預設動作。您要負責定義這個動作的作用。使用 `customAction` 的好處是您可以選擇處理使用者要求，而不必管理對話方塊堆疊。當 `customAction` 觸發時，`onSelectAction` 選項可以處理要求，而不必將新的對話方塊推至堆疊上。此動作完成後，控制權會交回給堆疊最上方的對話方塊，然後 Bot 便可繼續。

您可以使用 `customAction` 提供一般且快速的動作要求，例如，「What is the temperature outside right now?」、「What time is it right now in Paris?」、「Remind me to buy milk at 5pm today」等等。這些是 Bot 所能執行，且不必管理堆疊的一般動作。

`customAction` 的另一個主要差異是，它會繫結至 Bot 而不是對話方塊。

下列程式碼範例說明如何將 `customAction` 繫結至聽取要求的 `bot`，以設定提醒。

```

bot.customAction({
    matches: /remind|reminder/gi,
    onSelectAction: (session, args, next) => {
        // Set reminder...
        session.send("Reminder is set.");
    }
});

```

繫結 `beginDialogAction`

將 `beginDialogAction` 繫結至對話方塊會向對話註冊動作。這個方法會在觸發時啟動另一個對話方塊。此動作行為類似於呼叫 `beginDialog` 方法。系統會將新的對話方塊推至對話方塊堆疊最上方，因此，新對話不會自動結束目前的工作。當新的對話方塊結束後，目前的工作就會繼續。

下列程式碼片段說明如何將 `beginDialogAction` 繫結至對話方塊。

```

// Order dinner.
bot.dialog('orderDinner', [
    //...waterfall steps...
])
// Once triggered, will start the 'showDinnerCart' dialog.
// Then, the waterfall will resumed from the step that was interrupted.
.beginDialogAction('showCartAction', 'showDinnerCart', {
    matches: /^show cart$/i
});

// Show dinner items in cart
bot.dialog('showDinnerCart', function(session){
    for(var i = 1; i < session.conversationData.orders.length; i++){
        session.send(`You ordered: ${session.conversationData.orders[i].Description} for a total of
${session.conversationData.orders[i].Price}.`);
    }

    // End this dialog
    session.endDialog(`Your total is: ${session.conversationData.orders[0].Price}`);
});

```

如果您需要將額外引數傳遞至新的對話方塊，您可以在動作中新增 `dialogArgs` 選項。

使用上述範例，您就可以將它修改為接受透過 `dialogArgs` 傳遞的引數。

```

// Order dinner.
bot.dialog('orderDinner', [
    //...waterfall steps...
])
// Once triggered, will start the 'showDinnerCart' dialog.
// Then, the waterfall will resumed from the step that was interrupted.
.beginDialogAction('showCartAction', 'showDinnerCart', {
    matches: /^show cart$/i,
    dialogArgs: {
        showTotal: true;
    }
});

// Show dinner items in cart with the option to show total or not.
bot.dialog('showDinnerCart', function(session, args){
    for(var i = 1; i < session.conversationData.orders.length; i++){
        session.send(`You ordered: ${session.conversationData.orders[i].Description} for a total of
${session.conversationData.orders[i].Price}.`);
    }

    if(args && args.showTotal){
        // End this dialog with total.
        session.endDialog(`Your total is: ${session.conversationData.orders[0].Price}`);
    }
    else{
        session.endDialog(); // Ends without a message.
    }
});

```

繫結 `reloadAction`

將 `reloadAction` 繫結至對話方塊便會將其註冊至對話中方塊。將這個動作繫結至對話方塊，會讓對話方塊在觸發動作時重新啟動。觸發此動作類似於呼叫 `replaceDialog` 方法。這適合用來實作邏輯以處理使用者語句（像是「start over」）或建立迴圈。

下列程式碼片段說明如何將 `reloadAction` 繫結至對話方塊。

```
// Order dinner.  
bot.dialog('orderDinner', [  
    //...waterfall steps...  
])  
// Once triggered, will restart the dialog.  
.reloadAction('startOver', 'Ok, starting over.', {  
    matches: /^start over$/i  
});
```

如果您需要將額外引數傳遞至重新載入的對話方塊，您可以在動作中新增 `dialogArgs` 選項。此選項會傳遞至 `args` 參數。重寫上述程式碼範例以便在重新載入動作上收到引數，會看起來像這樣：

```
// Order dinner.  
bot.dialog('orderDinner', [  
    function(session, args, next){  
        if(args && args.isReloaded){  
            // Reload action was triggered.  
        }  
  
        session.send("Lets order some dinner!");  
        builder.Prompts.choice(session, "Dinner menu:", dinnerMenu);  
    }  
    //...other waterfall steps...  
])  
// Once triggered, will restart the dialog.  
.reloadAction('startOver', 'Ok, starting over.', {  
    matches: /^start over$/i,  
    dialogArgs: {  
        isReloaded: true;  
    }  
});
```

繫結 `cancelAction`

繫結 `cancelAction` 會將其註冊至對話方塊中。一經觸發，此動作就會突然結束對話方塊。對話方塊結束後，父對話方塊會以已繼續程式碼 (指出其已 `canceled`) 來繼續進行。這個動作可讓您處理「nevermind」或「cancel」之類的語句。如果您需要改為以程式設計方式取消對話方塊，請參閱[取消對話方塊](#)。如需已繼續程式碼的詳細資訊，請參閱[提示結果](#)。

下列程式碼片段說明如何將 `cancelAction` 繫結至對話方塊。

```
// Order dinner.  
bot.dialog('orderDinner', [  
    //...waterfall steps...  
])  
//Once triggered, will end the dialog.  
.cancelAction('cancelAction', 'Ok, cancel order.', {  
    matches: /^nevermind$|^cancel$|^cancel.*order/i  
});
```

繫結 `endConversationAction`

繫結 `endConversationAction` 會將其註冊至對話方塊中。一經觸發，這個動作就會結束與使用者的對話。觸發此動作類似於呼叫 `endConversation` 方法。交談結束後，適用於 Node.js 的 Bot Framework SDK 會清除對話堆疊和保存的狀態資料。如需所保存狀態資料的詳細資訊，請參閱[管理狀態資料](#)。

下列程式碼片段說明如何將 `endConversationAction` 繫結至對話方塊。

```
// Order dinner.  
bot.dialog('orderDinner', [  
    //...waterfall steps...  
])  
// Once triggered, will end the conversation.  
.endConversationAction('endConversationAction', 'Ok, goodbye!', {  
    matches: /^goodbye$/i  
});
```

確認中斷

上述絕大多數 (雖然不是全部) 的動作都會中斷一般對話流程。許多動作具有干擾性，所以應謹慎處理。例如，`triggerAction`、`cancelAction` 或 `endConversationAction` 會清除對話方塊堆疊。如果使用者不小心觸發上述其中一個動作，他們就必須讓工作重新開始。為了確保使用者真的想要觸發這些動作，您可以將 `confirmPrompt` 選項新增到這些動作。`confirmPrompt` 會詢問使用者是否確定要取消或結束目前的工作。它可讓使用者改變心意，並繼續原本的程序。

下面的程式碼片段顯示 `cancelAction` 與 `confirmPrompt`，可確保使用者真的想要取消訂單程序。

```
// Order dinner.  
bot.dialog('orderDinner', [  
    //...waterfall steps...  
])  
// Confirm before triggering the action.  
// Once triggered, will end the dialog.  
.cancelAction('cancelAction', 'Ok, cancel order.', {  
    matches: /^nevermind$|^cancel$|^cancel.*order/i,  
    confirmPrompt: "Are you sure?"  
});
```

此動作觸發後會詢問使用者「是否確定？」使用者必須回答「是」才能繼續該動作，回答「否」則可取消動作並從原來的地方繼續。

後續步驟

動作可讓您預期使用者的要求，並讓 Bot 妥善處理這些要求。這些動作中有許多會干擾目前的對話流程。如果您想要讓使用者能夠跳出並繼續對話流程，您需要先儲存使用者狀態再跳出。讓我們更加深入地了解如何儲存使用者狀態和管理狀態資料。

[管理狀態資料](#)

建立訊息

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

Bot 和使用者會透過訊息來通訊。Bot 會傳送訊息活動以傳達資訊給使用者，同樣也會接收來自使用者的訊息活動。某些訊息可能只包含純文字，而其他訊息可能包含更豐富的內容，例如要讀出的文字、建議的動作、媒體附件、複合式資訊卡 (Rich Card)，和通道特定資料。

本文說明一些可供您用來增強使用者體驗的常用訊息方法。

預設訊息處理常式

適用於 Node.js 的 Bot Framework SDK 隨附內建於 `session` 物件的預設訊息處理常式。此訊息處理常式可讓您在 Bot 和使用者之間傳送和接收文字訊息。

傳送文字訊息

使用預設訊息處理常式來傳送文字訊息非常簡單，只要呼叫 `session.send` 並傳入字串即可。

下列範例說明如何傳送文字訊息來歡迎使用者。

```
session.send("Good morning.");
```

下列範例說明如何使用 JavaScript 字串範本來傳送文字訊息。

```
var msg = `You ordered: ${order.Description} for a total of $$\{order.Price}\`;  
session.send(msg); //msg: "You ordered: Potato Salad for a total of $5.99."
```

接收文字訊息

當使用者傳送訊息給 Bot 時，Bot 會透過 `session.message` 屬性接收到訊息。

下列範例說明如何存取使用者的訊息。

```
var userMessage = session.message.text;
```

自訂訊息

若要更充分地掌控訊息的文字格式設定，您可以建立自訂 `message` 物件並先設定所需屬性，再將此物件傳送給使用者。

下列範例說明如何建立自訂 `message` 物件，並設定 `text`、`textFormat` 和 `textLocale` 屬性。

```

var customMessage = new builder.Message(session)
    .text("Hello!")
    .textFormat("plain")
    .textLocale("en-us");
session.send(customMessage);

```

若您在範圍內沒有 `session` 物件，則可以使用 `bot.send` 方法將已格式化的訊息傳送給使用者。

訊息的 `textFormat` 屬性可用來指定文字格式。`textFormat` 屬性可以設定為 **plain**、**markdown** 或 **xml**。

`textFormat` 的預設值為 **markdown**。

訊息屬性

`Message` 物件具有內部資料屬性，並使用此屬性來管理所傳送的訊息。您所設定的其他屬性，則會透過此物件向您公開的不同方法。

訊息方法

訊息屬性會透過物件的方法來加以設定並擷取。下表提供您所能呼叫以便設定/取得不同訊息屬性的方法清單。

方法	說明
<code>addAttachment(attachment:AttachmentType)</code>	將附件新增至訊息
<code>addEntity(obj:Object)</code>	將實體新增至訊息。
<code>address(addr:IAddress)</code>	寫下訊息的路由資訊位址。若要傳送主動訊息給使用者，請將訊息的位址儲存在 <code>userData</code> 包中。
<code>attachmentLayout(style:string)</code>	提示用戶端應如何配置多個附件。預設值為 'list'。
<code>attachments(list:AttachmentType)</code>	要傳送給使用者的卡片或影像清單。
<code>compose(prompts:string[], ...args:any[])</code>	撰寫要給使用者的複雜、隨機回覆。
<code>entities(list:Object[])</code>	已傳遞給 Bot 或使用者的結構化物件。
<code>inputHint(hint:string)</code>	傳送給使用者的提示，讓他們知道 Bot 是否需要進一步的輸入。內建提示會自動為外寄訊息填入這個值。
<code>localTimeStamp((optional)time:string)</code>	傳送訊息時的當地時間 (由用戶端或 Bot 設定，例如:2016-09-23T13:07:49.4714686-07:00)。
<code>originalEvent(event:any)</code>	內送訊息通道的原始/原生格式訊息。
<code>sourceEvent(map:ISourceEventMap)</code>	因為外寄訊息可以用來傳遞來源特定的事件資料，例如自訂附件。
<code>speak(ssml:TextType, ...args:any[])</code>	將訊息的發音領域設定為「語音合成標記語言 (SSML)」。會在受支援裝置上向使用者說出此內容。
<code>suggestedActions(suggestions:ISuggestedActions IIIsSuggestedActions)</code>	要傳送給使用者的選擇性建議動作。只會在支援建議動作的通道上顯示建議動作。

方法	說明
<code>summary(text:TextType, ...args:any[])</code>	要在其中顯示為後援以及顯示為訊息內容簡短描述的文字 (例如:近期交談清單)。
<code>text(text:TextType, ...args:any[])</code>	設定訊息文字。
<code>textFormat(style:string)</code>	設定文字格式。預設格式為 markdown 。
<code>textLocale(locale:string)</code>	設定訊息的目標語言。
<code>toMessage()</code>	取得訊息的 JSON。
<code>composePrompt(session:Session, prompts:string[], args?:any[])</code>	將提示陣列結合成單一的當地語系化提示, 然後選擇性地以傳入的引數填滿提示範本位置。
<code>randomPrompt(prompts:TextType)</code>	從傳入的 *prompts 陣列中取得隨機提示。

後續步驟

[傳送和接收附件](#)

傳送和接收附件

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

使用者與 Bot 之間的訊息交換可以包含媒體附件，例如影像、視訊、音訊和檔案等。可傳送的附件類型隨通道而異，但以下是基本類型：

- **媒體和檔案**：您可以傳送影像、音訊和視訊等檔案，方法是將 **contentType** 設定為 [IAttachment 物件](#) 的 MIME 類型，然後在 **contentUrl** 中傳遞檔案的連結。
- **卡**：您可以藉由將 **contentType** 設定為所需的卡類型來傳送一組豐富的視覺化卡，然後再傳遞卡的 JSON。如果您使用其中一個複合式資訊卡 (Rich Card) 建立器類別 (例如 **HeroCard**)，系統會自動為您填入附件。如需這方面的範例，請參閱[傳送複合式資訊卡 \(Rich Card\)](#)。

新增媒體附件

訊息物件預期會是 [IMessage](#) 的執行個體，當您想要包含影像等附件時，最適合以物件形式來傳送訊息給使用者。請使用 [session.send\(\)](#) 方法，以 JSON 物件的形式傳送訊息。

範例

下列範例會檢查使用者是否已傳送附件，如果是，則會回傳附件中包含的任何影像。您可以傳送影像給 Bot，以使用 Bot Framework 模擬器來進行這方面的測試。

```
// Create your bot with a function to receive messages from the user
var bot = new builder.UniversalBot(connector, function (session) {
    var msg = session.message;
    if (msg.attachments && msg.attachments.length > 0) {
        // Echo back attachment
        var attachment = msg.attachments[0];
        session.send({
            text: "You sent:",
            attachments: [
                {
                    contentType: attachment.contentType,
                    contentUrl: attachment.contentUrl,
                    name: attachment.name
                }
            ]
        });
    } else {
        // Echo back users text
        session.send("You said: %s", session.message.text);
    }
});
```

其他資源

- [使用頻道偵測器來預覽功能](#)
- [IMessage](#)

- 傳送複合式資訊卡 (Rich Card)
- `session.send`

傳送主動訊息

2019/3/5 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

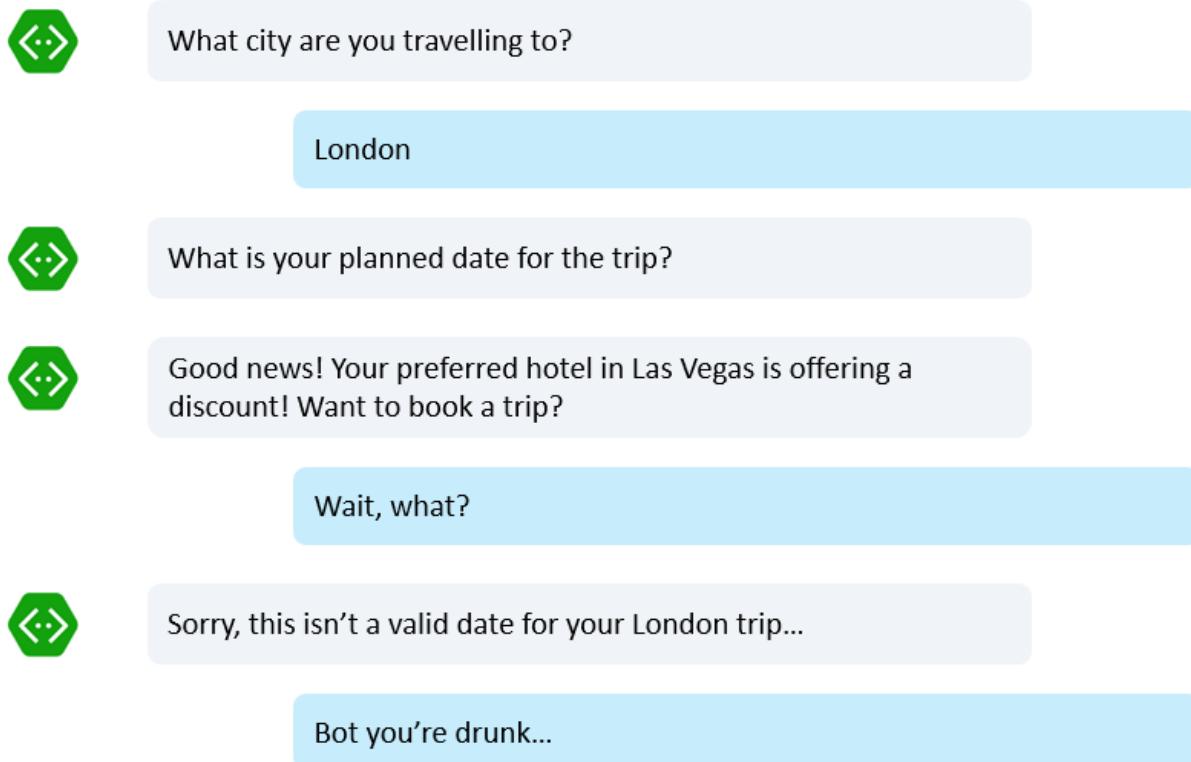
一般而言，Bot 直接傳送給使用者的每個訊息都與使用者先前的輸入相關。在某些情況下，Bot 可能需要將與目前對話主題或使用者最後傳送的訊息不直接相關的訊息傳送給使用者。這些類型的訊息稱為主動訊息。

主動訊息可用於各種情況。如果 Bot 設定了計時器或提醒，它必須在到達該時間時通知使用者。或者，如果 Bot 從外部系統收到通知，它可能需要將該資訊立即傳達給使用者。比方說，如果使用者先前已要求 Bot 監視產品的價格，則 Bot 可以在產品的價格下降了 20% 時對使用者發出警報。或者，如果 Bot 需要一些時間來編譯對使用者問題的回應，它可能會通知使用者已延遲，並且在此同時允許交談繼續進行。當 Bot 完成問題回應的編譯時，它會與使用者分享該資訊。

在 Bot 中實作主動訊息時：

- 請「勿」在短時間內傳送數個主動訊息。某些通道會強制限制 Bot 將訊息傳送給使用者的頻率，如果違反了這些限制，將會停用 Bot。
- 請「勿」將主動訊息傳送給先前未與 Bot 互動或透過其他方式（例如電子郵件或簡訊）與 Bot 連繫的使用者。

主動訊息可能會造成非預期的行為。請考慮下列狀況。



在此範例中，使用者先前已要求 Bot 監視拉斯維加斯旅館的價格。Bot 會啟動一個在過去幾天內持續執行的背景監視工作。在交談中，當背景工作觸發拉斯維加斯旅館折扣的相關通知訊息時，使用者正在預約到倫敦的行程。Bot 會將這項資訊插入交談中，進而產生讓人混淆的使用者體驗。

Bot 應該如何處理這種情況？

- 等待目前旅遊預約完成，然後傳遞通知。這種方式可將混亂情況降到最低，但延遲傳達資訊可能會導致使用者錯過拉斯維加斯旅館的低價機會。
- 取消目前的旅遊預約流程，並立即傳遞通知。這種方式可及時提供資訊，但可能會強制使用者重新開始進行旅遊預約，而讓他們感到沮喪。
- 中斷目前的預約，清楚地將交談主題變更為拉斯維加斯旅館直到使用者回應，然後切換回進行中的旅遊預約並從中斷處繼續執行。這種方式似乎是最佳選擇，但它會對 Bot 開發人員和使用者帶來複雜性。

大多數情況下，您的 Bot 將以某種方式組合使用臨機操作的主動訊息和其他技術來處理這種情況。

主動式訊息的類型

臨機操作主動式訊息是最簡單的主動式訊息類型。Bot 只會在每次觸發時將訊息插入對話，不會顧及使用者目前是否與 Bot 在其他對話主題中，也不會嘗試以任何方式變更對話。

若要更順利地處理通知，請考慮使用其他方式將通知整合到對話流程中，例如在對話狀態中設定旗標，或將通知新增至佇列。

傳送臨機操作的主動訊息

下列程式碼範例示範如何使用適用於 Node.js 的 Bot Framework SDK 傳送臨機操作的主動訊息。

為能將臨機操作的訊息傳送给使用者，Bot 必須先收集並儲存來自目前交談的使用者相關資訊。訊息的 **address** 屬性包含 Bot 稍後需要傳送给使用者之臨機操作訊息的所有資訊。

```
bot.dialog('adhocDialog', function(session, args) {
    var savedAddress = session.message.address;

    // (Save this information somewhere that it can be accessed later, such as in a database, or
    session.userData)
    session.userData.savedAddress = savedAddress;

    var message = 'Hello user, good to meet you! I now know your address and can send you notifications in the
    future.';
    session.send(message);
})
```

NOTE

Bot 可以任何方式儲存使用者資料，只要 Bot 以後能夠存取此資料。

Bot 收集完使用者相關資訊之後，它可以隨時向使用者傳送臨機操作的主動訊息。若要這樣做，它只要擷取之前儲存的使用者資料、建構訊息，然後傳送訊息即可。

```
var inMemoryStorage = new builder.MemoryBotStorage();

var bot = new builder.UniversalBot(connector)
    .set('storage', inMemoryStorage); // Register in-memory storage

function sendProactiveMessage(address) {
    var msg = new builder.Message().address(address);
    msg.text('Hello, this is a notification');
    msg.textLocale('en-US');
    bot.send(msg);
}
```

TIP

您可從非同步的觸發程序 (例如 http 要求、計時器、佇列) 或開發人員選擇的任何其他地方，起始臨機操作主動訊息。

傳送對話方塊式的主動訊息

下列程式碼範例示範如何使用適用於 Node.js 的 Bot Framework SDK 傳送對話式主動訊息。您可在 [startNewDialog](#) 資料夾中找到完整的運作範例。

為能將交談方式式的訊息傳送給使用者，Bot 必須先收集 (並儲存) 來自目前交談的相關資訊。

`session.message.address` 物件包含 Bot 需要傳送給使用者之對話方塊式主動訊息的所有資訊。

```
// proactiveDialog dialog
bot.dialog('proactiveDialog', function (session, args) {

    savedAddress = session.message.address;

    var message = 'Hey there, I\'m going to interrupt our conversation and start a survey in five seconds...';
    session.send(message);

    message = `You can also make me send a message by accessing:
http://localhost:${server.address().port}/api/CustomWebApi`;
    session.send(message);

    setTimeout(() => {
        startProactiveDialog(savedAddress);
    }, 5000);
});
```

傳送訊息時，Bot 會建立新的對話方塊，並將它新增至對話方塊堆疊的最上層。新的交談方塊會控制交談，傳遞主動訊息、關閉，然後將控制權傳回給堆疊中的上一個交談方塊。

```
// initiate a dialog proactively
function startProactiveDialog(address) {
    bot.beginDialog(address, "*:survey");
}
```

NOTE

上述程式碼範例需要自訂的檔案 **botadapter.js**，您可以從 [GitHub](#) 下載。

調查交談方塊會控制交談直到結束。然後，它會關閉 (呼叫 `session.endDialog()`)，再將控制權傳回給之前的對話方塊。

```
// handle the proactive initiated dialog
bot.dialog('survey', function (session, args, next) {
    if (session.message.text === "done") {
        session.send("Great, back to the original conversation");
        session.endDialog();
    } else {
        session.send('Hello, I\'m the survey dialog. I\'m interrupting your conversation to ask you a question.
Type "done" to resume');
    }
});
```

範例程式碼

如需示範如何使用適用於 Node.js 的 Bot Framework SDK 傳送主動訊息的完整範例，請參閱 GitHub 中的[主動訊息範例](#)。在主動訊息範例中，[simpleSendMessage](#) 示範如何傳送臨機操作的主動訊息，而[startNewDialog](#) 示範如何傳送對話方塊式的主動訊息。

其他資源

- [設計交談流程](#)

將複合式資訊卡 (Rich Card) 附件新增至訊息

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

許多頻道 (例如 Skype 和 Facebook) 均支援傳送複合式圖形化資訊卡給使用者，其中具有使用者可按一下來起始某個動作的互動式按鈕。SDK 會提供數個訊息和卡片建立器類別，可用來建立和傳送卡片。Bot Framework 連接器服務將使用頻道的原生結構描述來呈現這些卡片，以支援跨平台通訊。如果頻道不支援卡片 (例如 SMS)，Bot Framework 將盡其所能為使用者呈現合理的體驗。

複合式資訊卡的類型

Bot Framework 目前支援八種類型的複合式資訊卡 (Rich Card)：

卡片類型	說明
調適型卡片	可包含文字、語音、影像、按鈕和輸入欄位之任意組合的可自訂卡片。請參閱 個別頻道支援 。
動畫卡片	可播放動畫 GIF 或短片的卡片。
音訊卡片	可以播放音訊檔案的卡片。
主圖卡片	通常包含單一大型影像、一或多個按鈕和文字的卡片。
縮圖卡片	通常包含單一縮圖影像、一或多個按鈕和文字的卡片。
收據卡片	讓 Bot 向使用者提供收據的卡片。通常包含收據上的項目清單、稅金和總金額資訊，以及其他文字。
登入卡片	可讓 Bot 要求使用者登入的卡片。通常包含文字，以及一或多個使用者可以按一下以起始登入程序的按鈕。
視訊卡片	可播放視訊的卡片。

傳送浮動切換的主圖卡片

下列範例會示範某家虛構 T 恤公司的 Bot，並示範如何在使用者說出 “show shirts” (顯示 T 恤) 時，傳送浮動切換的卡片來回應。

```

// Create your bot with a function to receive messages from the user
// Create bot and default message handler
var bot = new builder.UniversalBot(connector, function (session) {
    session.send("Hi... We sell shirts. Say 'show shirts' to see our products.");
});

// Add dialog to return list of shirts available
bot.dialog('showShirts', function (session) {
    var msg = new builder.Message(session);
    msg.attachmentLayout(builder.AttachmentLayout.carousel)
    msg.attachments([
        new builder.HeroCard(session)
            .title("Classic White T-Shirt")
            .subtitle("100% Soft and Luxurious Cotton")
            .text("Price is $25 and carried in sizes (S, M, L, and XL)")
            .images([builder.CardImage.create(session,
                'http://petersapparel.parseapp.com/img/whiteshirt.png')])
            .buttons([
                builder.CardAction.imBack(session, "buy classic white t-shirt", "Buy")
            ]),
        new builder.HeroCard(session)
            .title("Classic Gray T-Shirt")
            .subtitle("100% Soft and Luxurious Cotton")
            .text("Price is $25 and carried in sizes (S, M, L, and XL)")
            .images([builder.CardImage.create(session,
                'http://petersapparel.parseapp.com/img/grayshirt.png')])
            .buttons([
                builder.CardAction.imBack(session, "buy classic gray t-shirt", "Buy")
            ])
    ]);
    session.send(msg).endDialog();
}).triggerAction({ matches: /^(show|list)/i });

```

這個範例會使用 [Message](#) 類別來建置浮動切換。

浮動切換會由 [HeroCard](#) 類別的清單所組成，其中包含影像、文字，以及會觸發購買該項目的單一按鈕。

按一下 [Buy] (購買) 按鈕就會觸發傳送訊息，因此我們需要新增第二個對話來擷取按鈕點選動作。

處理按鈕輸入

無論在任何時候，只要接收到以 "buy" (購買) 或 "add" (新增) 作為開始，且後面接著包含 "shirt" (T 恤) 單字之內容的訊息時，就會觸發 `buyButtonClick` 對話。對話一開始會使用幾個規則運算式，來尋找符合使用者所詢問之顏色及可選尺寸的 T 恤。這個額外的彈性可讓您同時支援來自使用者的按鈕點選動作，以及自然語言訊息，例如 "please add a large gray shirt to my cart" (請將 L 號的灰色 T 恤新增至我的購物車)。如果顏色是有效的，但是尺寸未知，Bot 就會提示使用者從清單挑選尺寸，然後才會將該項目新增至購物車。一旦 Bot 擁有所需的所有資訊之後，就會將該項目新增至使用 `session.userData` 保存的購物車，並傳送確認訊息給使用者。

```

// Add dialog to handle 'Buy' button click
bot.dialog('buyButtonClick', [
    function (session, args, next) {
        // Get color and optional size from users utterance
        var utterance = args.intent.matched[0];
        var color = /(white|gray)/i.exec(utterance);
        var size = /\b(Extra Large|Large|Medium|Small)\b/i.exec(utterance);
        if (color) {
            // Initialize cart item
            var item = session.dialogData.item = {
                product: "classic " + color[0].toLowerCase() + " t-shirt",
                size: size ? size[0].toLowerCase() : null,
                price: 25.0,
                qty: 1
            };
            if (!item.size) {
                // Prompt for size
                builder.Prompts.choice(session, "What size would you like?", "Small|Medium|Large|Extra Large");
            } else {
                // Skip to next waterfall step
                next();
            }
        } else {
            // Invalid product
            session.send("I'm sorry... That product wasn't found.").endDialog();
        }
    },
    function (session, results) {
        // Save size if prompted
        var item = session.dialogData.item;
        if (results.response) {
            item.size = results.response.entity.toLowerCase();
        }

        // Add to cart
        if (!session.userData.cart) {
            session.userData.cart = [];
        }
        session.userData.cart.push(item);

        // Send confirmation to users
        session.send("A '%(size)s %(product)s' has been added to your cart.", item).endDialog();
    }
]).triggerAction({ matches: /(buy|add)\s.*shirt/i });

```

針對影像下載新增訊息延遲

某些頻道會傾向先下載影像，然後才向使用者顯示訊息，因此如果您傳送一則包含影像的訊息，後面緊接著一則未包含影像的訊息，您有時會看見這兩則訊息的順序會在使用者的摘要中翻轉過來。若要將發生此情況的機會降到最低，您可以試著確保您的影像是來自內容傳遞網路 (CDN)，並避免使用過大的影像。在極端的案例中，您甚至可能需要在含有影像的訊息和後面接著的訊息之間插入 1-2 秒的延遲。若要讓使用者對這個延遲感到更自然，您可以藉由呼叫 **session.sendTyping()** 以在開始延遲之前傳送「正在輸入」的指示。

Bot Framework 會實作批次處理，以試著防止來自 Bot 的多則訊息會以不按照順序的方式顯示。當您的 Bot 傳送多個回覆給使用者時，個別的訊息將會自動分組為批次，並以集合的形式傳遞給使用者，以試圖保留訊息的原始順序。這個自動批次處理會在每次呼叫 **session.send()** 之後，以及起始對 **send()** 的下一個呼叫之前，預設等待 250 毫秒的時間。

訊息批次處理延遲是可設定的。若要停用 SDK 的自動批次處理邏輯，請將預設延遲設定為較大的數字，然後以手動方式呼叫 **sendBatch()**，並搭配會在傳遞批次之後叫用的回呼。

傳送調適型卡片

調適型卡片可包含文字、語音、影像、按鈕和輸入欄位的任意組合。調適型卡片會使用[調適型卡片](#) (英文) 中所指定的 JSON 格式來建立，讓您能夠完整控制卡片的內容和格式。

若要使用 Node.js 建立調適型卡片，請利用[調適型卡片](#) (英文) 網站中的資訊，來了解調適型卡片的結構描述、探索調適型卡片的元素，以及查看可用來建立具有不同組合和複雜度之卡片的 JSON 範例。此外，您可以使用互動式視覺化檢視，來設計調適型卡片承載及預覽卡片輸出。

下列程式碼範例示範如何建立包含調適型卡片作為行事曆提醒的訊息：

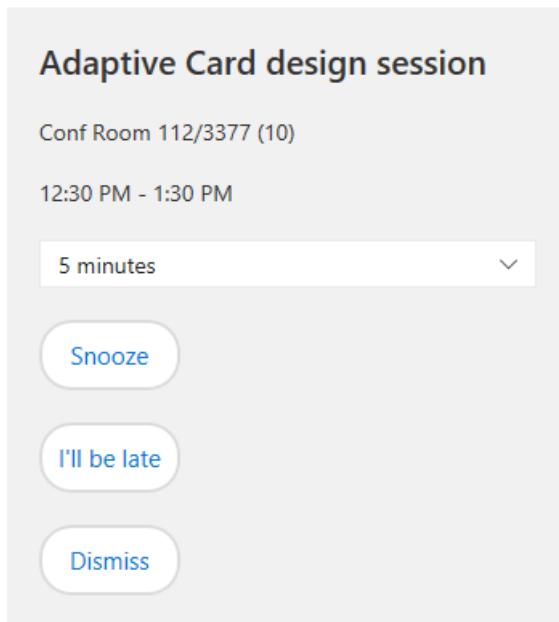
```
var msg = new builder.Message(session)
    .addAttachment({
        contentType: "application/vnd.microsoft.card.adaptive",
        content: {
            type: "AdaptiveCard",
            speak: "<s>Your meeting about \"Adaptive Card design session\"<brack strength='weak' /> is starting at 12:30pm</s><s>Do you want to snooze <brack strength='weak' /> or do you want to send a late notification to the attendees?</s>",
            body: [
                {
                    "type": "TextBlock",
                    "text": "Adaptive Card design session",
                    "size": "large",
                    "weight": "bolder"
                },
                {
                    "type": "TextBlock",
                    "text": "Conf Room 112/3377 (10)"
                },
                {
                    "type": "TextBlock",
                    "text": "12:30 PM - 1:30 PM"
                },
                {
                    "type": "TextBlock",
                    "text": "Snooze for"
                },
                {
                    "type": "Input.ChoiceSet",
                    "id": "snooze",
                    "style": "compact",
                    "choices": [
                        {
                            "title": "5 minutes",
                            "value": "5",
                            "isSelected": true
                        },
                        {
                            "title": "15 minutes",
                            "value": "15"
                        },
                        {
                            "title": "30 minutes",
                            "value": "30"
                        }
                    ]
                },
                ],
                "actions": [
                {
                    "type": "Action.OpenUrl",
                    "method": "POST",
                    "url": "http://foo.com",
                    "title": "Snooze"
                },
                ]
            ]
        }
    })
```

```

    },
    "type": "Action.OpenUrl",
    "method": "POST",
    "url": "http://foo.com",
    "title": "I'll be late"
},
{
    "type": "Action.OpenUrl",
    "method": "POST",
    "url": "http://foo.com",
    "title": "Dismiss"
}
]
}
});
session.send(msg);

```

產生的卡片會包含三個文字區塊、一個輸入欄位 (選擇清單) 和三個按鈕：



其他資源

- [使用頻道偵測器來預覽功能](#)
- [調適型卡片 \(英文\)](#)
- [AnimationCard](#)
- [AudioCard](#)
- [HeroCard](#)
- [ThumbnailCard](#)
- [ReceiptCard](#)
- [SignInCard](#)
- [VideoCard](#)
- [Message](#)
- [如何傳送附件](#)

將語音新增至訊息

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

如果您要為具備語音功能的通道 (例如 Cortana) 建置 Bot，您可以建構訊息，其中指定要由 Bot 讀出的文字。您也可以指定[輸入提示](#)，藉由指出您的 Bot 要接受、需要或忽略使用者輸入，來嘗試影響用戶端的麥克風狀態。

指定要由 Bot 讀出的文字

使用適用於 Node.js 的 Bot Framework SDK 時，有多種方式可用來指定要由 Bot 在具備語音功能的通道上讀出的文字。您可以設定 `IMessage.speak` 屬性，並使用 `session.send()` 方法傳送訊息、使用 `session.say()` 方法 (傳入指定顯示文字、語音文字和選項的參數) 傳送訊息，或使用內建的提示 (指定選項 `speak` 和 `retrySpeak`) 傳送訊息。

`IMessage.speak`

如果您要建立將會使用 `session.send()` 方法來傳送的訊息，請設定 `speak` 屬性以指定要由 Bot 讀出的文字。下列程式碼範例會建立一則訊息，其中指定要讀出的文字，並指出 Bot 會[接受使用者輸入](#)。

```
var msg = new builder.Message(session)
    .speak('This is the text that will be spoken.')
    .inputHint(builder.InputHint.acceptingInput);
session.send(msg).endDialog();
```

`session.say()`

除了使用 `session.send()` 以外，您也可以呼叫 `session.say()` 方法以建立和傳送訊息，並且讓訊息指定要讀出的文字、要顯示的文字和其他選項。此方法定義如下：

```
session.say(displayText: string, speechText: string, options?: object)
```

參數	說明
<code>displayText</code>	要顯示的文字。
<code>speechText</code>	要讀出的文字 (採用純文字或 SSML 格式)。
<code>options</code>	可包含附件或 輸入提示 的 <code>IMessage</code> 物件。

下列程式碼範例會傳送一則訊息，其中指定要顯示的文字和要讀出的文字，並指出 Bot 會[忽略使用者輸入](#)。

```
session.say('Please hold while I calculate a response.',
    'Please hold while I calculate a response.',
    { inputHint: builder.InputHint.ignoringInput }
);
```

提示選項

使用任何內建提示時，您可以設定選項 `speak` 和 `retrySpeak`，以指定要由您的 Bot 讀出的文字。下列程式碼範例

會建立一則提示，其中指定要顯示的文字、一開始要讀出的文字，和等待使用者輸入一段時間後要讀出的文字。它會指出 Bot 需要使用者輸入，並使用 [SSML](#) 格式指定 "sure" 這個字應以適度強調的方式讀出。

```
builder.Prompts.text(session, 'Are you sure that you want to cancel this transaction?', {
    speak: 'Are you <emphasis level="moderate">sure</emphasis> that you want to cancel this transaction?',
    retrySpeak: 'Are you <emphasis level="moderate">sure</emphasis> that you want to cancel this transaction?',
    inputHint: builder.InputHint.expectingInput
});
```

語音合成標記語言 (SSML)

若要指定要由 Bot 讀出的文字，您可以使用純文字字串或格式化為語音合成標記語言 (SSML) 的字串，SSML 是一種以 XML 為基礎的標記語言，可讓您控制 Bot 語音的各種特性，例如聲音、速率、音量、發音、音調等等。如需 SSML 的詳細資訊，請參閱[合成標記語言參考](#)。

TIP

使用 [SSML 程式庫](#)建立格式正確的 SSML。

輸入提示

當您開啟語音功能的通道上傳送訊息時，您也可以藉由包含輸入提示以指出 Bot 要接受、需要或忽略使用者輸入，來嘗試影響用戶端的麥克風狀態。如需詳細資訊，請參閱[將輸入提示新增至訊息](#)。

範例程式碼

如需完整範例以了解如何使用適用於 .NET 的 Bot Framework SDK，來建立具備語音功能的 Bot，請參閱 GitHub 中的[骰子機範例](#)。

其他資源

- [語音合成標記語言 \(SSML\) \(英文\)](#)
- [骰子機範例 \(GitHub\)](#)

將輸入提示新增至訊息

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

您可藉由指定訊息的輸入提示，指出您的 Bot 在訊息傳遞給用戶端之後，會接受、需要或忽略使用者輸入。這可讓用戶端為許多通道設定相應的使用者輸入控制項狀態。例如，如果訊息的輸入提示指出 Bot 會忽略使用者輸入，則用戶端可關閉麥克風並停用輸入方塊，以防止使用者提供輸入。

接受輸入

若要指出 Bot 要被動接受輸入，但不等候使用者回應，請將訊息的輸入提示設定為

`builder.InputHint.acceptingInput`。在許多通道上，這會啟用用戶端的輸入方塊並關閉麥克風，但使用者仍可存取。例如，如果使用者按住麥克風按鈕，Cortana 會開啟麥克風並接受使用者輸入。下列程式碼範例會建立訊息，指出 Bot 會接受使用者輸入。

```
var msg = new builder.Message(session)
    .speak('This is the text that will be spoken.')
    .inputHint(builder.InputHint.acceptingInput);
session.send(msg).endDialog();
```

需要輸入

若要指出 Bot 要等候使用者回應，請將訊息的輸入提示設定為 `builder.InputHint.expectingInput`。在許多通道上，這會啟用用戶端的輸入方塊並開啟麥克風。下列程式碼範例會建立提示，指出 Bot 必須有使用者輸入。

```
builder.Prompts.text(session, 'This is the text that will be displayed.', {
  speak: 'This is the text that will be spoken initially.',
  retrySpeak: 'This is the text that is spoken after waiting a while for user input.',
  inputHint: builder.InputHint.expectingInput
});
```

忽略輸入

若要指出 Bot 尚未準備好接收使用者輸入，請將訊息的輸入提示設定為 `builder.InputHint.ignoringInput`。在許多通道上，這會停用用戶端的輸入方塊並關閉麥克風。下列程式碼範例會使用 `session.say()` 方法傳送訊息，指出 Bot 會忽略使用者輸入。

```
session.say('Please hold while I calculate a response. Thanks!', 
  'Please hold while I calculate a response. Thanks!', 
  { inputHint: builder.InputHint.ignoringInput }
);
```

輸入提示的預設值

如果您未設定訊息的輸入提示，Bot Framework SDK 會使用下列邏輯自動為您設定：

- 如果 Bot 傳送提示，訊息的輸入提示將指定 Bot 需要輸入。
- 如果 Bot 傳送單一訊息，訊息的輸入提示將指定 Bot 會接受輸入。
- 如果 Bot 傳送一系列的連續訊息，系列中的所有訊息（但最後一則訊息除外）的輸入提示都將指定 Bot 會忽略輸入，而系列中最後一則訊息的輸入提示將指定 Bot 會接受輸入。

其他資源

- [將語音新增至訊息](#)

將建議的動作新增至訊息

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

建議動作可讓您的 Bot 顯示可供使用者點選，以提供輸入的按鈕。建議動作會出現在編輯器附近，讓使用者只要點選按鈕即可回答問題或進行選取，而無需使用鍵盤輸入回應，藉以提升使用者體驗。不同於複合式資訊卡 (Rich Card) 中出現的按鈕 (即使在點選之後，使用者仍可看見並可存取)，出現在建議動作窗格內的按鈕會在使用者進行選取後消失。這可以避免使用者在對話中點選過時的按鈕，並簡化 Bot 的開發 (因為您不需要再說明該情境)。

建議的動作範例

若要將建議的動作新增至訊息中，請將訊息的 `suggestedActions` 屬性設為[卡片動作](#)的清單，這些動作可以代表要向使用者呈現的按鈕。

此程式碼範例示範如何傳送一則訊息，向使用者呈現三種建議的動作：

```
var msg = new builder.Message(session)
    .text("Thank you for expressing interest in our premium golf shirt! What color of shirt would you like?")
    .suggestedActions(
        builder.SuggestedActions.create(
            session, [
                builder.CardAction.imBack(session, "productId=1&color=green", "Green"),
                builder.CardAction.imBack(session, "productId=1&color=blue", "Blue"),
                builder.CardAction.imBack(session, "productId=1&color=red", "Red")
            ]
        ));
session.send(msg);
```

當使用者點選其中一個建議的動作時，Bot 會收到來自使用者的訊息，其中包含對應動作的 `value`。

請注意，`imBack` 方法會將 `value` 張貼至您所使用通道的聊天視窗。如果這不是想要的效果，您可以使用 `postBack` 方法；這個方法仍然會將選取項目張貼回您的 Bot，但是不會在聊天視窗中顯示選取項目。部分通道不支援 `postBack`，但是，在這些執行個體中，方法的行為就像是 `imBack`。

其他資源

- [範例](#)
- [IMessage](#)
- [ICardAction](#)
- [session.send](#)

傳送輸入指標

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

使用者期望訊息可收到及時的回應。如果 Bot 正在執行某些長時間執行的工作，例如呼叫伺服器或執行查詢，而未提供使用者 Bot 收到訊息的指示，使用者可能會不耐煩並傳送其他訊息，或以為 Bot 故障了。許多通道都支援傳送輸入指示，以向使用者顯示訊息已收到且正在進行處理。

輸入指標範例

下列範例會示範如何使用 [session.sendTyping\(\)](#) 傳送輸入指示。您可以使用 Bot Framework 模擬器進行測試。

```
// Create bot and default message handler
var bot = new builder.UniversalBot(connector, function (session) {
    session.sendTyping();
    setTimeout(function () {
        session.send("Hello there...");
    }, 3000);
});
```

當插入訊息延遲以避免包含影像的訊息傳送失序時，輸入指標也很實用。

若要進一步了解，請參閱[如何傳送複合式資訊卡 \(Rich Card\)](#)。

其他資源

- [sendTyping](#)

攔截訊息

2019/4/15 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

Bot Framework SDK 中的**中介軟體**功能可讓 Bot 攔截使用者與 Bot 之間交換的所有訊息。針對每個攔截的訊息，您可以選擇執行如下的動作：將訊息儲存到您所指定的資料存放區，以建立對話記錄，或是以某種方式檢查訊息，並執行程式碼所指定的任何動作。

NOTE

Bot Framework 不會自動儲存交談的詳細資料，因此這麼做可能會擷取 Bot 和使用者不想要與外界共用的個人資訊。如果您的 Bot 儲存了對話的詳細資料，它即應將此情況告知使用者，並說明這些資料的使用方式。

範例

下列程式碼範例會示範如何使用適用於 Node.js 的 Bot Framework SDK 中的**中介軟體**概念，來攔截使用者與 Bot 之間交換的訊息。

首先，設定適用於傳入訊息 (`botbuilder`) 和傳出訊息 (`send`) 的處理常式。

```
server.post('/api/messages', connector.listen());
var bot = new builder.UniversalBot(connector);
bot.use({
  botbuilder: function (session, next) {
    myMiddleware.logIncomingMessage(session, next);
  },
  send: function (event, next) {
    myMiddleware.logOutgoingMessage(event, next);
  }
})
```

接著，實作每個處理常式以定義要針對每個被攔截之訊息採取的動作。

```
module.exports = {
  logIncomingMessage: function (session, next) {
    console.log(session.message.text);
    next();
  },
  logOutgoingMessage: function (event, next) {
    console.log(event.text);
    next();
  }
}
```

現在每個輸入訊息 (從使用者到 Bot) 都會觸發 `logIncomingMessage`，而每個輸出訊息 (從 Bot 到使用者) 則都會觸發 `logOutgoingMessage`。在此範例中，Bot 只會列印有關每個訊息的部分資訊，但您可以視需要更新 `logIncomingMessage` 和 `logOutgoingMessage`，以定義您想要針對每個訊息採取的動作。

範例程式碼

如需示範如何使用適用於 Node.js 的 Bot Framework SDK 來攔截和記錄訊息的完整範例，請參閱 GitHub 中的[中介軟體和記錄範例](#) (英文)。

使用 Cortana 技能建置啟用語音的 Bot

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

適用於 Node.js 的 Bot Framework SDK 可藉由將 Bot 連線到 Cortana 通道作為 Cortana 技能，讓您建置啟用語音的 Bot。Cortana 技能可讓您透過 Cortana 提供功能來回應使用者的語音輸入。

TIP

如需有關技能是什麼及其用途的詳細資訊，請參閱 [Cortana 技能套件](#)。

使用 Bot Framework 建立 Cortana 技能不需要太多 Cortana 專屬知識，主要是建置 Bot。與其他您可能已建立的 Bot 主要差異在於，Cortana 同時具有視覺與音訊元件。對於視覺元件，Cortana 提供用於轉譯內容（例如卡片）的畫布區域。對於音訊元件，您可以在 Bot 訊息中提供文字或 SSML，Cortana 會讀給使用者聽，讓您的 Bot 擁有語音功能。

NOTE

在許多不同的裝置上皆有提供 Cortana。部分具有螢幕，而其他（如獨立喇叭）可能沒有。您應確認 Bot 能夠處理這兩者。請參閱 [Cortana 特定實體](#)，以了解如何檢查裝置資訊。

將語音新增至您的 Bot

Bot 的語音訊息會表示為語音合成標記語言 (SSML)。Bot Framework SDK 可讓您將 SSML 包含在 Bot 的回應中，以在 Bot 顯示的內容之外控制 Bot 所說內容。

session.say

您的 Bot 會用 `session.say` 方法來取代 `session.send`，向使用者說話。它包含用於傳送 SSML 輸出的選擇性參數，以及卡片等附件。

該方法的格式如下：

```
session.say(displayText: string, speechText: string, options?: object)
```

參數	說明
<code>displayText</code>	要顯示在 Cortana UI 中的文字訊息。
<code>speechText</code>	Cortana 讀給使用者聽的文字或 SSML。
<code>options</code>	可包含附件或輸入提示的 IMessage 物件。輸入提示會指出 Bot 正在接受、預期或忽略輸入。卡片附件會顯示在 Cortana 畫布中 <code>displayText</code> 資訊的下方。

`inputHint` 屬性有助於向 Cortana 指出 Bot 是否預期輸入。如果您使用內建的提示，這個值會自動設為 `expectingInput` 的預設值。

值	說明
acceptingInput	您的 Bot 會被動接受輸入，但不會等候回應。如果使用者按住 [麥克風] 按鈕，Cortana 會接受使用者的輸入。
expectingInput	指出 Bot 正主動預期使用者的回應。Cortana 會聆聽使用者對麥克風說的話。
	注意：請勿在無周邊裝置（沒有顯示器的裝置）上使用 expectingInput 。請參閱 Cortana 技能套件常見問題集 。
ignoringInput	Cortana 會忽略輸入。如果您的 Bot 主動處理要求，則可能會傳送這項提示，且在完成要求之前，將會忽略來自使用者的輸入。

下列範例會示範 Cortana 如何讀取純文字或 SSML：

```
// Have Cortana read plain text
session.say('This is the text that Cortana displays', 'This is the text that is spoken by Cortana.');

// Have Cortana read SSML
session.say('This is the text that Cortana displays', '<speak version="1.0"
xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">This is the text that is spoken by Cortana.
</speak>');
```

此範例會示範如何讓 Cortana 知道該預期使用者輸入。麥克風會保持開啟狀態。

```
// Add an InputHint to let Cortana know to expect user input
session.say('Hi there', 'Hi, what's your name?', {
    inputHint: builder.InputHint.expectingInput
});
```

提示

除了使用 **session.say()** 方法，您也可以使用 **speak** 和 **retrySpeak** 選項，將文字或 SSML 傳遞給內建的提示。

```
builder.Prompts.text(session, 'text based prompt', {
    speak: 'Cortana reads this out initially',
    retrySpeak: 'This message is repeated by Cortana after waiting a while for user input',
    inputHint: builder.InputHint.expectingInput
});
```

若要向使用者顯示選項清單，請使用 **Prompts.choice**。同義詞選項可在辨識使用者語句時允許更多彈性空間。在 **results.response.entity** 中會傳回值選項。動作選項會指定 Bot 對於選擇所顯示的標籤。

Prompts.choice 可支援序數選擇。這表示使用者可以說「第一個」、「第二個」、「第三個」來選擇清單中的項目。比方說，在下列提示中，如果使用者要求 Cortana 提供「第二個選項」，則提示傳回的值為 8。

```

var choices = [
    { value: '4', action: { title: '4 Sides' }, synonyms: 'four|for|4 sided|4 sides' },
    { value: '8', action: { title: '8 Sides' }, synonyms: 'eight|ate|8 sided|8 sides' },
    { value: '12', action: { title: '12 Sides' }, synonyms: 'twelve|12 sided|12 sides' },
    { value: '20', action: { title: '20 Sides' }, synonyms: 'twenty|20 sided|20 sides' },
];
builder.Prompts.choice(session, 'choose_sides', choices, {
    speak: speak(session, 'choose_sides_ssml') // use helper function to format SSML
});

```

在上一個範例中，提示中 **speak** 屬性的 SSML，會使用儲存在當地語系化提示檔案中的字串，以下列格式進行格式化。

```

{
    "choose_sides": "__Number of Sides__",
    "choose_sides_ssml": [
        "How many sides on the dice?",
        "Pick your poison.",
        "All the standard sizes are supported."
    ]
}

```

協助程式函式接著會建置語音合成標記語言 (SSML) 文件的必要根項目。

```

module.exports.speak = function (template, params, options) {
    options = options || {};
    var output = '<speak xmlns="http://www.w3.org/2001/10/synthesis" ' +
        'version="' + (options.version || '1.0') + '" ' +
        'xml:lang="' + (options.lang || 'en-US') + '"';
    output += module.exports.vsprintf(template, params);
    output += '</speak>';
    return output;
}

```

TIP

您可以在[骰子機範例技能](#) (英文) 中找到小型的公用程式模組 (ssml.js)，用以建置 Bot 的 SSML 型回應。另外還有幾個 [npm](#) (英文) 提供的實用 SSML 程式庫，可讓您輕鬆建立格式化的 SSML。

在 Cortana 中顯示卡片

除了語音回應，Cortana 也可以顯示卡片附件。Cortana 可支援下列多媒體卡片：

- [HeroCard](#)
- [ReceiptCard](#)
- [ThumbnailCard](#)

請參閱[卡片設計最佳做法](#)，以查看這些卡片在 Cortana 內的樣貌。如需如何將複合式資訊卡 (Rich Card) 新增至 Bot 的範例，請參閱[傳送複合式資訊卡 \(Rich Card\)](#)。

下列程式碼會示範如何將語音和 **inputHint** 屬性，新增至包含 Hero 卡片的訊息中。

```
bot.dialog('HelpDialog', function (session) {
    var card = new builder.HeroCard(session)
        .title('help_title')
        .buttons([
            builder.CardAction.imBack(session, 'roll some dice', 'Roll Dice'),
            builder.CardAction.imBack(session, 'play yahtzee', 'Play Yahtzee')
        ]);
    var msg = new builder.Message(session)
        .speak(speak(session, 'I\'m roller, the dice rolling bot. You can say \'roll some dice\''))
        .addAttachment(card)
        .inputHint(builder.InputHint.acceptingInput); // Tell Cortana to accept input
    session.send(msg).endDialog();
}).triggerAction({ matches: '/help/i });

/** This helper function builds the required root element of a Speech Synthesis Markup Language (SSML)
document. */
module.exports.speak = function (template, params, options) {
    options = options || {};
    var output = '<speak xmlns="http://www.w3.org/2001/10/synthesis" ' +
        'version="' + (options.version || '1.0') + '" ' +
        'xml:lang="' + (options.lang || 'en-US') + '"';
    output += module.exports.vsprintf(template, params);
    output += '</speak>';
    return output;
}
```

範例 : RollerSkill

下列各節中的程式碼來自擲骰子的範例 Cortana 技能。請從 [BotBuilder-Samples 存放庫 \(英文\)](#) 下載 Bot 的完整程式碼。

您可以向 Cortana 說出技能的[叫用名稱](#)以便叫用技能。對於骰子機技能，在您將 Bot 新增至 Cortana 通道並將其註冊為 Cortana 技能後，可以告訴 Cortana「啟動骰子機」或「讓骰子機擲骰子」以進行叫用。

探索程式碼

RollerSkill 範例一開始會開啟包含一些按鈕的卡片，來告知使用者哪些選項可供使用。

```
/**  
 * Create your bot with a default message handler that receive messages from the user.  
 * - This function is be called anytime the user's utterance isn't  
 *   recognized by the other handlers in the bot.  
 */  
var bot = new builder.UniversalBot(connector, function (session) {  
    // Just redirect to our 'HelpDialog'.  
    session.replaceDialog('HelpDialog');  
});  
  
//...  
  
bot.dialog('HelpDialog', function (session) {  
    var card = new builder.HeroCard(session)  
        .title('help_title')  
        .buttons([  
            builder.CardAction.imBack(session, 'roll some dice', 'Roll Dice'),  
            builder.CardAction.imBack(session, 'play craps', 'Play Craps')  
        ]);  
    var msg = new builder.Message(session)  
        .speak(speak(session, 'help_ssml'))  
        .addAttachment(card)  
        .inputHint(builder.InputHint.acceptingInput);  
    session.send(msg).endDialog();  
}).triggerAction({ matches: '/help/i });
```

提示使用者輸入

以下對話方塊會設定自訂遊戲讓 Bot 玩。它會詢問使用者想要讓骰子有幾個面，以及應該擲幾個骰子。建置遊戲結構之後，它會將其傳遞至個別的 'PlayGameDialog'。

若要啟動對話方塊，此對話方塊上的 **triggerAction()** 處理常式可讓使用者說出「我想要擲骰子」等內容。它會使用規則運算式來比對使用者的輸入，但您也可以使用 [LUIS 意圖](#)。

```

bot.dialog('CreateGameDialog', [
  function (session) {
    // Initialize game structure.
    // - dialogData gives us temporary storage of this data in between
    //   turns with the user.
    var game = session.dialogData.game = {
      type: 'custom',
      sides: null,
      count: null,
      turns: 0
    };

    var choices = [
      { value: '4', action: { title: '4 Sides' }, synonyms: 'four|for|4 sided|4 sides' },
      { value: '6', action: { title: '6 Sides' }, synonyms: 'six|sex|6 sided|6 sides' },
      { value: '8', action: { title: '8 Sides' }, synonyms: 'eight|8 sided|8 sides' },
      { value: '10', action: { title: '10 Sides' }, synonyms: 'ten|10 sided|10 sides' },
      { value: '12', action: { title: '12 Sides' }, synonyms: 'twelve|12 sided|12 sides' },
      { value: '20', action: { title: '20 Sides' }, synonyms: 'twenty|20 sided|20 sides' },
    ];
    builder.Prompts.choice(session, 'choose_sides', choices, {
      speak: speak(session, 'choose_sides_ssml')
    });
  },
  function (session, results) {
    // Store users input
    // - The response comes back as a find result with index & entity value matched.
    var game = session.dialogData.game;
    game.sides = Number(results.response.entity);

    /**
     * Ask for number of dice.
     */
    var prompt = session.gettext('choose_count', game.sides);
    builder.Prompts.number(session, prompt, {
      speak: speak(session, 'choose_count_ssml'),
      minValue: 1,
      maxValue: 100,
      integerOnly: true
    });
  },
  function (session, results) {
    // Store users input
    // - The response is already a number.
    var game = session.dialogData.game;
    game.count = results.response;

    /**
     * Play the game we just created.
     *
     * replaceDialog() ends the current dialog and start a new
     * one in its place. We can pass arguments to dialogs so we'll pass the
     * 'PlayGameDialog' the game we created.
     */
    session.replaceDialog('PlayGameDialog', { game: game });
  }
]).triggerAction({ matches: [
  /(roll|role|throw|shoot).*(dice|die|dye|bones)/i,
  /new game/i
]}));

```

轉譯結果

這個對話方塊是主要的遊戲迴圈。Bot 會將遊戲結構儲存在 **session.conversationData** 中，因此萬一使用者說「再擲一次」，我們可以再次重擲同一組骰子。

```

bot.dialog('PlayGameDialog', function (session, args) {
    // Get current or new game structure.
    var game = args.game || session.conversationData.game;
    if (game) {
        // Generate rolls
        var total = 0;
        var rolls = [];
        for (var i = 0; i < game.count; i++) {
            var roll = Math.floor(Math.random() * game.sides) + 1;
            if (roll > game.sides) {
                // Accounts for 1 in a million chance random() generated a 1.0
                roll = game.sides;
            }
            total += roll;
            rolls.push(roll);
        }

        // Format roll results
        var results = '';
        var multiLine = rolls.length > 5;
        for (var i = 0; i < rolls.length; i++) {
            if (i > 0) {
                results += ' . ';
            }
            results += rolls[i];
        }

        // Render results using a card
        var card = new builder.HeroCard(session)
            .subtitle(game.count > 1 ? 'card_subtitle_plural' : 'card_subtitle_singular', game)
            .buttons([
                builder.CardAction.imBack(session, 'roll again', 'Roll Again'),
                builder.CardAction.imBack(session, 'new game', 'New Game')
            ]);
        if (multiLine) {
            //card.title('card_title').text('\n\n' + results + '\n\n');
            card.text(results);
        } else {
            card.title(results);
        }
        var msg = new builder.Message(session).addAttachment(card);

        // Determine bots reaction for speech purposes
        var reaction = 'normal';
        var min = game.count;
        var max = game.count * game.sides;
        var score = total/max;
        if (score == 1.0) {
            reaction = 'best';
        } else if (score == 0) {
            reaction = 'worst';
        } else if (score <= 0.3) {
            reaction = 'bad';
        } else if (score >= 0.8) {
            reaction = 'good';
        }

        // Check for special craps rolls
        if (game.type == 'craps') {
            switch (total) {
                case 2:
                case 3:
                case 12:
                    reaction = 'craps_lose';
                    break;
                case 7:
                    reaction = 'craps_seven';
            }
        }
    }
});
```

```
        break;
    case 11:
        reaction = 'craps_eleven';
        break;
    default:
        reaction = 'craps_retry';
        break;
    }
}

// Build up spoken response
var spoken = '';
if (game.turn == 0) {
    spoken += sessiongettext('start_' + game.type + '_game_ssml') + ' ';
}
spoken += sessiongettext(reaction + '_roll_reaction_ssml');
msg.speak(ssml.speak(spoken));

// Increment number of turns and store game to roll again
game.turn++;
session.conversationData.game = game;

/**
 * Send card and bot's reaction to user.
 */

msg.inputHint(builder.InputHint.acceptingInput);
session.send(msg).endDialog();
} else {
    // User started session with "roll again" so let's just send them to
    // the 'CreateGameDialog'
    session.replaceDialog('CreateGameDialog');
}
}).triggerAction({ matches: /(roll|role|throw|shoot) again/i });
});
```

後續步驟

如果您的 Bot 是在本機執行或部署在雲端中，可以從 Cortana 叫用。請參閱[測試 Cortana 技能](#)，以了解試用 Cortana 技能所需的步驟。

其他資源

- [Cortana 技能套件](#)
- [將語音新增至訊息](#)
- [SSML 參考](#)
- [Cortana 的語音設計最佳做法](#)
- [Cortana 的卡片設計最佳做法](#)
- [Cortana 開發人員中心](#)
- [Cortana 的測試和偵錯最佳做法](#)

使用 Skype 支援語音通話

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

Skype 支援稱為「呼叫 Bot」的豐富功能。當啟用時，使用者可以撥打語音電話給 Bot，並使用互動語音回應系統 (IVR) 與其進行互動。適用於 Node.js SDK 的 Bot 建立器包含特殊的呼叫 SDK，開發人員可用來將撥打功能新增至其聊天 Bot。

呼叫 SDK 與[聊天 SDK](#) 非常類似。它們具有類似的類別、會共用通用的建構，且您甚至可以使用聊天 SDK 將訊息傳送給您正在呼叫的使用者。兩個 SDK 是設計為並排執行，雖然它們很類似，但有一些顯著的差異，而您通常應該避免將兩個程式庫中的類別混用。

建立呼叫 Bot

下列範例程式碼顯示呼叫 Bot 的 "Hello World" 外觀，與一般聊天 Bot 類似的程度。

```
var restify = require('restify');
var calling = require('botbuilder-calling');

// Setup Restify Server
var server = restify.createServer();
server.listen(process.env.port || process.env.PORT || 3978, function () {
    console.log(`#${server.name} listening to ${server.url}`);
});

// Create calling bot
var connector = new calling.CallConnector({
    callbackUrl: 'https://<your host>/api/calls',
    appId: '<your bots app id>',
    appPassword: '<your bots app password>'
});
var bot = new calling.UniversalCallBot(connector);
server.post('/api/calls', connector.listen());

// Add root dialog
bot.dialog('/', function (session) {
    session.send('Watson... come here!');
});
```

NOTE

若要尋找您 Bot 的 **AppID** 和 **AppPassword**，請參閱 [MicrosoftAppID](#) 和 [MicrosoftAppPassword](#)。

模擬器目前不支援對呼叫 Bot 進行測試。若要測試您的 Bot，您必須完成發佈您 bot 所需的大部分步驟。您也必須使用 Skype 用戶端來與 Bot 互動。

啟用 Skype 通道

[註冊您的 Bot](#)並啟用 Skype 通道。當您註冊 Bot 時，必須提供傳訊端點。建議您將呼叫 Bot 與聊天 Bot 配對，如此聊天 Bot 的端點就是您一般會放入該欄位的端點。如果您只註冊呼叫 Bot，則可以只將呼叫端點貼入該欄位。

若要啟用的實際呼叫功能，您必須進入 Bot 的 Skype 通道，並開啟呼叫功能。接著您會取得一個欄位，可將您的呼叫端點複製進去。請確定您將 <https://ngrok.com> 連結用於您呼叫端點的主機部分。

在您註冊 Bot 期間，系統將會指派應用程式識別碼和密碼給您，您應該將其貼入您 hello world Bot 的連接器設定。您也需要取得完整的呼叫連結，並貼到 callbackUrl 設定中。

將 Bot 新增至連絡人

在您開發人員入口網站中的 Bot 註冊頁面上，您會在 Bot Skype 通道旁看到 [新增至 Skype] 按鈕。按一下按鈕即可將 Bot 新增至 Skype 中的連絡人清單。一旦您（及取得此加入連結的任何人）這麼做，將能夠與 Bot 進行通訊。

測試 Bot

您可以使用 Skype 用戶端來測試 Bot。當您按一下 Bot 連絡人項目（您可能必須搜尋 Bot 才能看到它）時，應該會注意到呼叫圖示亮起。如果您已將呼叫新增至現有的 Bot，呼叫圖示可能需要幾分鐘的時間才會亮起。

如果您按下呼叫按鈕，應該會撥給您的 Bot，且您應該會聽到它說「Watson... 來這裡！」然後掛斷。

呼叫基本概念

[UniversalCallBot](#) 和 [CallConnector](#) 類別可讓您使用與聊天 Bot 大致相同的方式來撰寫呼叫 Bot。您要將基本上與[聊天對話方塊](#)相同的對話方塊新增至您的 Bot。您可以將[瀑布圖](#)新增到您的 Bot。有一個工作階段物件（[CallSession](#) 類別），其中包含已新增的 [answer\(\)](#)、[hangup\(\)](#)，以及 [reject\(\)](#) 方法，可管理目前的呼叫。一般情況下，您不需要擔心這些呼叫控制方法，因為 CallSession 具有邏輯，可自動管理您的呼叫。如果您採取例如傳送訊息或呼叫內建提示等動作，工作階段會自動接聽電話。如果您呼叫 [endConversation\(\)](#)，它也會自動掛斷/拒絕呼叫，或是會偵測到您已停止詢問呼叫者問題（您未呼叫內建的提示）。

呼叫和聊天 Bot 的另一個差異是，聊天 Bot 通常會將訊息、卡片和鍵盤傳送給使用者，而呼叫 Bot 則會處理「動作」和「結果」。建立組成一或多個[動作的工作流程](#)時，需要 Skype 呼叫 Bot。這在實務上是您不需要太擔心的另一項事情，因為 Bot 建立器呼叫 SDK 幾乎都會為您管理。[CallSession.send\(\)](#) 方法可讓您傳遞動作或字串，從而會變成 [PlayPromptActions](#)。工作階段包含自動批次邏輯，可將多個動作合併成提交給呼叫服務的單一工作流程，以便您可以安全地多次呼叫 [send\(\)](#)。且您應仰賴 SDK 的內建[提示](#)，當使用者在處理所有結果時，向他們收集輸入。

管理狀態資料

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

Bot Builder Framework 可讓您的 Bot 儲存和擷取與使用者、對話或特定對話內容中的特定使用者相關聯的狀態資料。狀態資料有許多用途，例如判斷上一個對話最後的內容，或是單純地以名字問候回來的使用者。如果您儲存使用者的喜好設定，下次聊天時就可以使用該資訊來自訂對話。例如，您可能會在有和使用者感興趣之主題相關的新聞文章，或有可用約會時警報使用者。

針對測試和原型設計目的，您可以使用 Bot Builder Framework 的記憶體內部資料儲存體。針對生產環境的 Bot，您可以實作自己的儲存體接器，或使用其中一個 Azure 擴充功能。Azure 擴充功能可讓您將 Bot 的狀態資料儲存於表格儲存體、CosmosDB 或 SQL。本文將示範如何使用記憶體內部儲存體接器來儲存 Bot 的狀態資料。

IMPORTANT

建議您不要將 Bot Framework 狀態服務 API 用於生產環境，因為未來版本可能會淘汰它。建議您更新 Bot 程式碼，以針對測試目的使用記憶體內部儲存體接器，或針對生產環境 Bot 使用其中一個 Azure 擴充功能。

記憶體內部的資料儲存體

記憶體內部的資料儲存體僅供測試之用。這是可變更的臨時儲存體。每次 Bot 重新啟動時，就會清除資料。若要將記憶體內部的儲存體運用於測試用途，您必須執行兩個動作。首先，建立記憶體內部儲存體的新執行個體：

```
var inMemoryStorage = new builder.MemoryBotStorage();
```

然後，在建立 **UniversalBot** 時，將它設定為 Bot：

```
var inMemoryStorage = new builder.MemoryBotStorage();
var bot = new builder.UniversalBot(connector, [..waterfall steps..])
    .set('storage', inMemoryStorage); // Register in-memory storage
```

您可以使用這個方法來設定自己的自訂資料儲存體，或使用任一個 Azure 擴充功能。

管理自訂資料儲存體

在生產環境中基於效能和安全性理由，您可能要實作自己的資料儲存體，或考量實作下列其中一個資料儲存體選項：

1. [使用 Cosmos DB 管理狀態資料](#)
2. [使用表格儲存體管理狀態資料](#)

利用這其中任一個 Azure 擴充功能選項，透過適用於 Node.js 的 Bot Framework SDK 設定和保存資料的機制，會與記憶體內部的資料儲存體保持一致。

儲存體容器

在適用於 Node.js 的 Bot Framework SDK 中，`session` 物件會公開下列用來儲存狀態資料的屬性。

屬性	目標範圍	說明
<code>userData</code>	使用者	包含儲存於指定通道上以供使用者使用的資料。此資料將跨多個交談進行保存。
<code>privateConversationData</code>	交談	包含儲存於指定通道上特定交談內容中以供使用者使用的資料。此資料是目前使用者私用的，並且僅會針對目前的交談進行保存。當交談結束時或明確呼叫我 <code>endConversation</code> 時，即會清除這個屬性。
<code>conversationData</code>	交談	包含儲存於指定通道上特定交談內容中的資料。此資料會與參與交談的所有使用者共用，並且僅會針對目前的交談進行保存。當交談結束時或明確呼叫我 <code>endConversation</code> 時，即會清除這個屬性。
<code>dialogData</code>	對話	包含僅針對目前對話儲存的資料。每個對話都會針對這個屬性維護自己的複本。從對話堆疊中移除對話時，即會清除這個屬性。

這四個屬性會對應到可用來儲存資料的四個資料儲存體容器。您用來儲存資料的屬性將取決於您所儲存之資料的適當範圍、您所儲存之資料的本質，以及您想要保存資料的時間長度。例如，如果您需要儲存使用者資料，以便跨多個交談使用，請考慮使用 `userData` 屬性。如果您需要將區域變數值暫時儲存於對話範圍內，請考慮使用 `dialogData` 屬性。如果您需要暫時儲存必須能夠跨多個對話存取的資料，請考慮使用 `conversationData` 屬性。

資料持續性

根據預設，會將使用 `userData`、`privateConversationData` 和 `conversationData` 屬性儲存的資料設定為在交談結束之後保存。如果您不想將資料保存於 `userData` 容器中，請將 `persistUserData` 旗標設為 `false`。如果您不想將資料保存於 `conversationData` 容器中，請將 `persistConversationData` 旗標設為 `false`。

```
// Do not persist userData
bot.set(`persistUserData`, false);

// Do not persist conversationData
bot.set(`persistConversationData`, false);
```

NOTE

您無法停用 `privateConversationData` 容器的資料持續性，它一定是持續性的。

設定資料

您可以藉由將簡單的 JavaScript 物件直接儲存至儲存體容器來儲存它們。對於像是 `Date` 的複雜物件，請考慮將它轉換成 `string`。這是因為狀態資料會序列化並儲存為 JSON。下列程式碼範例示範如何儲存基本資

料、陣列、物件對應，以及複雜的 `Date` 物件。

儲存基本資料

```
session.userData.userName = "Kumar Sarma";
session.userData.userAge = 37;
session.userData.hasChildren = true;
```

儲存陣列

```
session.userData.profile = ["Kumar Sarma", "37", "true"];
```

儲存物件對應

```
session.userData.about = {
  "Profile": {
    "Name": "Kumar Sarma",
    "Age": 37,
    "hasChildren": true
  },
  "Job": {
    "Company": "Contoso",
    "StartDate": "June 8th, 2010",
    "Title": "Developer"
  }
}
```

儲存日期與時間

對於複雜的 JavaScript 物件，請先將它轉換為字串，然後再儲存到儲存體容器。

```
var startDate = builder.EntityRecognizer.resolveTime([results.response]);

// Date as string: "2017-08-23T05:00:00.000Z"
session.userData.start = startDate.toISOString();
```

儲存資料

在每個儲存體容器中建立的資料將保留於記憶體中，直到容器儲存為止。適用於 Node.js 的 Bot Framework SDK 會分批將資料傳送至 `ChatConnector` 服務，以便在有要傳送的訊息時加以儲存。若要儲存已存在於儲存體容器中的資料，而不傳送任何訊息，您可以手動呼叫 `save` 方法。如果您未呼叫 `save` 方法，存在於儲存體容器中的資料將會保存以作為批次處理的一部分。

```
session.userData.favoriteColor = "Red";
session.userData.about.job.Title = "Senior Developer";
session.save();
```

取得資料

若要存取儲存於特定儲存體容器中的資料，只需參考對應屬性。下列程式碼範例示範如何存取先前已儲存為基本資料、陣列、物件對應及複雜 `Date` 物件的資料。

存取基本資料

```
var userName = session.userData.userName;
var userAge = session.userData.userAge;
var hasChildren = session.userData.hasChildren;
```

存取陣列

```
var userProfile = session.userData.userProfile;

session.send("User Profile:");
for(int i = 0; i < userProfile.length, i++){
    session.send(userProfile[i]);
}
```

存取物件對應

```
var about = session.userData.about;

session.send("User %s works at %s.", about.Profile.Name, about.Job.Company);
```

存取 Date 物件

以字串形式擷取日期資料，然後將它轉換成 JavaScript 的 Date 物件。

```
// startDate as a JavaScript Date object.
var startDate = new Date(session.userData.start);
```

刪除資料

根據預設，從對話堆疊中移除對話時，即會清除儲存於 `dialogData` 容器中的資料。同樣地，在呼叫 `endConversation` 方法時，會清除儲存於 `conversationData` 容器和 `privateConversationData` 容器中的資料。不過，若要刪除儲存於 `userData` 容器中的資料，您必須明確地清除它。

若要明確地清除儲存於任何儲存體容器中的資料，只需重設容器即可，如下列程式碼範例所示。

```
// Clears data stored in container.
session.userData = {};
session.privateConversationData = {};
session.conversationData = {};
session.dialogData = {};
```

絕對不要將資料容器設定為 `null`，或從 `session` 物件移除它，因為這樣做將在您下次嘗試存取容器時造成錯誤。此外，建議您在手動清除記憶體中的容器之後，手動呼叫 `session.save()`，以清除先前已保存的任何對應資料。

後續步驟

現在，您已經了解如何管理使用者狀態資料，讓我們看看如何使用該資料，更有效地管理交談流程。

管理對話流程

其他資源

- [提示使用者輸入](#)

使用適用於 Node.js 的 Azure Cosmos DB 管理自訂狀態資料

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

在本文中，您將實作 Cosmos DB 儲存體來儲存和管理 Bot 的狀態資料。Bot 所使用的預設連接器狀態服務不適用於生產環境。您應使用 GitHub 上提供的 [Azure 延伸模組](#) (英文)，或使用您選擇的資料儲存體平台來實作自訂狀態用戶端。以下是使用自訂狀態儲存體的一些原因：

- 更高的狀態 API 輸送量 (更充分掌控效能)
- 異地分佈的低延遲
- 掌握資料儲存位置 (例如：美國西部與美國東部)
- 存取實際狀態資料
- 狀態資料資料庫不會與其他 Bot 共用
- 儲存超過 32KB

必要條件

- [Node.js](#)。
- [Bot Framework 模擬器](#) (英文)
- 必須具備 Node.js Bot。如果沒有，請[建立一個 Bot](#)。

建立 Azure 帳戶

如果您沒有 Azure 帳戶，請按一下[這裡](#)註冊免費帳戶。

設定 Azure Cosmos DB 資料庫

1. 在您登入 Azure 入口網站之後，按一下 [新增] 以建立新的 Azure Cosmos DB 資料庫。
2. 按一下 [資料庫]。
3. 找到 **Azure Cosmos DB**，然後按一下 [建立]。
4. 填寫欄位。如果是 [API] 欄位，請選取 [**SQL (DocumentDB)**]。填妥所有欄位後，按一下畫面底部的 [建立] 按鈕來部署新的資料庫。
5. 新資料庫部署完畢後，接著瀏覽至新資料庫。按一下 [存取金鑰] 以尋找金鑰和連接字串。您的 Bot 會使用此資訊來呼叫儲存體服務以儲存狀態資料。

安裝 botbuilder-azure 模組

若要透過命令提示字元安裝 `botbuilder-azure` 模組，請巡覽至 Bot 的目錄，然後執行下列 npm 命令：

```
npm install --save botbuilder-azure
```

修改您的 Bot 程式碼

若要使用 **Azure Cosmos DB** 資料庫，請將下列這幾行程式碼新增至 Bot 的 **app.js** 檔案。

1. 需要新安裝的模組。

```
var azure = require('botbuilder-azure');
```

2. 配置連線設定以連線至 Azure。

```
var documentDbOptions = {
  host: 'Your-Azure-DocumentDB-URI',
  masterKey: 'Your-Azure-DocumentDB-Key',
  database: 'botdocs',
  collection: 'botdata'
};
```

`host` 和 `masterKey` 值可在您資料庫的 [金鑰] 功能表中找到。如果 Azure 資料庫不存在 `database` 和 `collection` 項目於，系統將會為您建立。

3. 使用 `botbuilder-azure` 模組，建立兩個新的物件，以連線至 Azure 資料庫。首先，建立 `DocumentDBClient` 的執行個體，傳入連線的組態設定（定義為上述的 `documentDbOptions`）。接下來，建立傳入 `DocumentDBClient` 物件的 `AzureBotStorage` 執行個體。例如：

```
var docDbClient = new azure.DocumentDbClient(documentDbOptions);

var cosmosStorage = new azure.AzureBotStorage({ gzipData: false }, docDbClient);
```

4. 指定您想要使用自訂的資料庫，而非記憶體內部儲存體。例如：

```
var bot = new builder.UniversalBot(connector, function (session) {
  // ... Bot code ...
})
.set('storage', cosmosStorage);
```

現在您已準備就緒，可以使用模擬器測試 Bot。

執行 Bot 應用程式

透過命令提示字元，巡覽至 Bot 的目錄並使用下列命令執行 Bot：

```
node app.js
```

將 Bot 連線至模擬器

此時，Bot 正在本機執行。啟動模擬器，然後從模擬器連線至您的 Bot：

1. 將 `http://localhost:port-number/api/messages` 鍵入模擬器的網址列，其中連接埠號碼必須符合應用程式執行所在之瀏覽器中顯示的連接埠號碼。您目前可以將 **Microsoft 應用程式識別碼** 和 **Microsoft 應用程式密碼** 欄位留白。稍後註冊 Bot 時，您會取得這些資訊。
2. 按一下 [連接]。
3. 傳送訊息給 Bot，藉此測試您的 Bot。像平常一樣，與 Bot 互動。完成後，請前往儲存體總管並檢視已儲存的狀態資料。

檢視 Azure 入口網站的狀態資料

若要檢視狀態資料，請登入 Azure 入口網站並瀏覽至您的資料庫。按一下 [資料總管 (預覽)] 以確認正在儲存的 Bot 狀態資訊。

後續步驟

現在，您已經可以充分掌握 Bot 的狀態資料，讓我們看看如何使用才能更有效地管理交談流程。

[管理交談流程](#)

使用適用於 Node.js 的 Azure 資料表儲存體管理自訂狀態資料

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

在本文中，您將實作 Azure 資料表儲存體來儲存和管理 Bot 的狀態資料。Bot 所使用的預設「連接器狀態服務」不適用於生產環境。您應使用 GitHub 上提供的 [Azure 延伸模組](#) (英文)，或使用您選擇的資料儲存體平台來實作自訂狀態用戶端。以下是使用自訂狀態儲存體的一些原因：

- 更高的狀態 API 輸送量 (更充分掌控效能)
- 異地分佈的低延遲
- 掌握資料儲存位置 (例如：美國西部與美國東部)
- 存取實際狀態資料
- 狀態資料資料庫不會與其他 Bot 共用
- 儲存超過 32KB

必要條件

- [Node.js](#)。
- [Bot Framework 模擬器](#)。
- 必須具備 Node.js Bot。如果沒有，請[建立一個 Bot](#)。
- [儲存體總管](#)。

建立 Azure 帳戶

如果您沒有 Azure 帳戶，請按一下[這裡](#)註冊免費帳戶。

設定 Azure 資料表儲存體服務

1. 在您登入 Azure 入口網站之後，請按一下 [新增] 以建立新的 Azure 表格儲存體服務。
2. 搜尋實作 Azure 資料表的儲存體帳戶。按一下 [建立] 開始建立儲存體帳戶。
3. 填寫欄位，按一下畫面底部的 [建立] 按鈕來部署新的儲存體服務。
4. 部署新的儲存體服務之後，巡覽至您剛才建立的儲存體帳戶。您會發現它列在 [儲存體帳戶] 刀鋒視窗中。
5. 選取 [存取金鑰]，並複製該金鑰供稍後使用。您的 Bot 會使用 [儲存體帳戶名稱] 和 [金鑰] 來呼叫儲存體服務以儲存狀態資料。

安裝 botbuilder-azure 模組

若要透過命令提示字元安裝 `botbuilder-azure` 模組，請巡覽至 Bot 的目錄，然後執行下列 npm 命令：

```
npm install --save botbuilder-azure
```

修改您的 Bot 程式碼

若要使用 Azure 資料表資料庫，請將下列這幾行程式碼新增至 Bot 的 `app.js` 檔案。

1. 需要新安裝的模組。

```
var azure = require('botbuilder-azure');
```

2. 配置連線設定以連線至 Azure。

```
// Table storage
var tableName = "Table-Name"; // You define
var storageName = "Table-Storage-Name"; // Obtain from Azure Portal
var storageKey = "Azure-Table-Key"; // Obtain from Azure Portal
```

`storageName` 和 `storageKey` 值可在 Azure 資料表的 [存取金鑰] 功能表中找到。如果 `tableName` 不存在於 Azure 資料表中，則會為您建立。

3. 使用 `botbuilder-azure` 模組，建立兩個新的物件以連線至 Azure 資料表。首先，建立傳入連線組態設定的 `AzureTableClient` 執行個體。接下來，建立傳入 `AzureTableClient` 物件的 `AzureBotStorage` 執行個體。例如：

```
var azureTableClient = new azure.AzureTableClient(tableName, storageName, storageKey);

var tableStorage = new azure.AzureBotStorage({gzipData: false}, azureTableClient);
```

4. 指定您想要使用自訂的資料庫，而非記憶體內部儲存體，並將工作階段資訊新增至該資料庫。例如：

```
var bot = new builder.UniversalBot(connector, function (session) {
    // ... Bot code ...

    // capture session user information
    session.userData = {"userId": session.message.user.id, "jobTitle": "Senior Developer"};

    // capture conversation information
    session.conversationData[timestamp.toISOString().replace(/:/g, "-")] = session.message.text;

    // save data
    session.save();
})
.set('storage', tableStorage);
```

現在您已準備就緒，可以使用模擬器測試 Bot。

執行 Bot 應用程式

透過命令提示字元，巡覽至 Bot 的目錄並使用下列命令執行 Bot:

```
node app.js
```

將 Bot 連線至模擬器

此時，Bot 正在本機執行。啟動模擬器，然後從模擬器連線至您的 Bot:

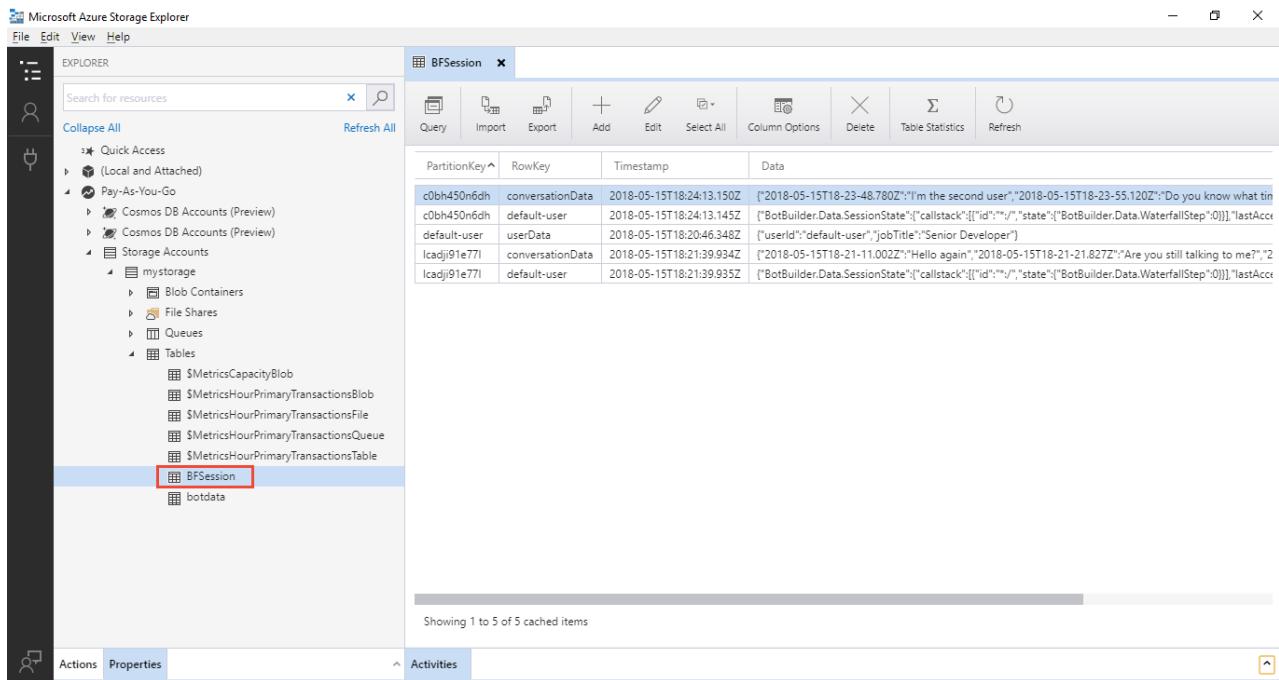
1. 將 `http://localhost:port-number/api/messages` 鍵入模擬器的網址列，其中連接埠號碼必須符合應用程式執

行所在之瀏覽器中顯示的連接埠號碼。您目前可以將 Microsoft 應用程式識別碼和 Microsoft 應用程式密碼欄位留白。稍後註冊 Bot 時，您會取得這些資訊。

2. 按一下 [連接]。
3. 傳送訊息給 Bot，藉此測試您的 Bot。像平常一樣，與 Bot 互動。完成後，請前往 [儲存體總管] 並檢視已儲存的狀態資料。

在 [儲存體總管] 中檢視資料

若要檢視狀態資料，請開啟 [儲存體總管] 並使用 Azure 入口網站認證連線至 Azure，或使用 `storageName` 和 `storageKey` 直接連線至表格，然後巡覽至您的 `tableName`。



The screenshot shows the Microsoft Azure Storage Explorer interface. On the left, the 'EXPLORER' sidebar lists storage accounts and tables under 'mystorage'. A table named 'BFSessions' is selected and highlighted with a red box. The main pane displays the 'BFSessions' table with columns: PartitionKey, RowKey, Timestamp, and Data. Five rows of data are shown:

PartitionKey	RowKey	Timestamp	Data
c0bh450n6dh	conversationData	2018-05-15T18:24:13.150Z	{"2018-05-15T18-23-48.780Z": "I'm the second user", "2018-05-15T18-23-55.120Z": "Do you know what time it is?", "2018-05-15T18-24-12.214Z": "I'm looking for information about the new process."}
c0bh450n6dh	default-user	2018-05-15T18:24:13.145Z	[{"BotBuilder.Data.SessionState": "callstack": [{"id": "", "state": "BotBuilder.Data.WaterfallStep[0]"}], "lastAccessed": "2018-05-15T18-21-11.002Z", "userId": "default-user", "jobTitle": "Senior Developer"}
default-user	userData	2018-05-15T18:20:46.348Z	
lcadj91e77l	conversationData	2018-05-15T18:21:39.934Z	{"2018-05-15T18-21-11.002Z": "Hello again", "2018-05-15T18-21-21.827Z": "Are you still talking to me?", "2018-05-15T18-21-21.827Z": "I'm the second user"}
lcadj91e77l	default-user	2018-05-15T18:21:39.935Z	[{"BotBuilder.Data.SessionState": "callstack": [{"id": "", "state": "BotBuilder.Data.WaterfallStep[0]"}], "lastAccessed": "2018-05-15T18-21-11.002Z", "userId": "default-user", "jobTitle": "Senior Developer"}

[資料] 資料行中交談的一筆記錄如下所示：

```
{  
  "2018-05-15T18-23-48.780Z": "I'm the second user",  
  "2018-05-15T18-23-55.120Z": "Do you know what time it is?",  
  "2018-05-15T18-24-12.214Z": "I'm looking for information about the new process."  
}
```

後續步驟

現在，您已經可以充分掌握 Bot 的狀態資料，讓我們看看如何使用才能更有效地管理交談流程。

管理交談流程

從訊息內容辨識使用者意圖

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

當 Bot 收到來自使用者的訊息時，Bot 可以使用辨識器來檢查訊息並判斷意圖。此意圖提供一個可叫用的對應（從訊息到對話方塊）。本文說明如何使用規則運算式或藉由檢查訊息內容來辨識意圖。例如，Bot 可以使用規則運算式來檢查訊息是否包含 "help" 這個字，並叫用 [說明] 對話方塊。Bot 也可以檢查使用者訊息的屬性，查看使用者已傳送映像而非文字，然後叫用映像處理對話方塊。

NOTE

如需使用 LUIS 辨識意圖的相關資訊，請參閱[使用 LUIS 辨識意圖和實體](#)

使用內建規則運算式辨識器

使用規則運算式 [RegExpRecognizer](#) 來偵測使用者的目的。您可以將多個運算式傳遞至辨識器，以支援多種語言。

TIP

請參閱[支援當地語系化](#)，以取得將 Bot 當地語系化的詳細資訊。

下列程式碼會建立名為 `cancelIntent` 的規則運算式辨識器並將它新增至 Bot。此範例中的辨識器提供 `en_us` 和 `ja_jp` 地區設定的規則運算式。

```
bot.recognizer(new builder.RegExpRecognizer( "CancelIntent", { en_us: /^(cancel|nevermind)/i, ja_jp: /^(キャンセル)/ }));
```

將辨識器新增至 Bot 後，請將 [triggerAction](#) 附加到您希望 Bot 在辨識器偵測到意圖時叫用的對話方塊。使用 [matches](#) 選項來指定意圖名稱，如下列程式碼所示：

```
bot.dialog('CancelDialog', function (session) {
    session.endConversation("Ok, I'm canceling your order.");
}).triggerAction({ matches: 'CancelIntent' });
```

意圖辨識器是全域的，這表示辨識器會針對從使用者接收的每則訊息執行。如果辨識器使用 [triggerAction](#) 偵測到繫結至對話方塊的意圖，則可以觸發中斷目前使用中的對話方塊。允許並處理中斷是一項有彈性的設計，該設計會考量使用者真正執行的動作。

TIP

若要了解 [triggerAction](#) 如何搭配對話方塊運作，請參閱[管理交談流程](#)。若要深入了解您可與已辨識意圖相關聯的各種動作，請參閱[處理使用者動作](#)。

註冊自訂意圖辨識器

您也可以實作自訂辨識器。這個範例會新增一個簡單辨識器，以尋找會說出 'help' 或 'goodbye' 的使用者，但您可以輕易地新增可進行更複雜處理的辨識器，例如辨識使用者何時傳送映像的辨識器。

```
var builder = require('....core/');

// Create bot and default message handler
var connector = new builder.ConsoleConnector().listen();
var bot = new builder.UniversalBot(connector, function (session) {
    session.send("You said: '%s'. Try asking for 'help' or say 'goodbye' to quit", session.message.text);
});

// Install a custom recognizer to look for user saying 'help' or 'goodbye'.
bot.recognizer({
    recognize: function (context, done) {
        var intent = { score: 0.0 };

        if (context.message.text) {
            switch (context.message.text.toLowerCase()) {
                case 'help':
                    intent = { score: 1.0, intent: 'Help' };
                    break;
                case 'goodbye':
                    intent = { score: 1.0, intent: 'Goodbye' };
                    break;
            }
        }
        done(null, intent);
    }
});
});
```

註冊辨識器之後，您即可使用 `matches` 子句來建立的辨識器與動作關聯。

```
// Add a help dialog with a trigger action that is bound to the 'Help' intent
bot.dialog('helpDialog', function (session) {
    session.endDialog("This bot will echo back anything you say. Say 'goodbye' to quit.");
}).triggerAction({ matches: 'Help' });

// Add a global endConversation() action that is bound to the 'Goodbye' intent
bot.endConversationAction('goodbyeAction', "Ok... See you later.", { matches: 'Goodbye' });
```

釐清多個意圖

Bot 可以註冊一個以上的辨識器。請注意，自訂辨識器範例涉及將數值分數指派給每個意圖。這是因為 Bot 可能有一個以上的辨識器，而且 Bot Framework SDK 會提供內建邏輯來釐清由多個辨識器所傳回的意圖。指派給意圖的分數通常介於 0.0 到 1.0，但自訂辨識器可能會定義大於 1.1 的意圖，以確保 Bot Framework SDK 的去除混淆邏輯一律會選擇該意圖。

根據預設，辨識器則會以平行方式執行，但您可以在 [IIntentRecognizerSetOptions](#) 中設定 `recognizeOrder`，所以只要 Bot 找到可提供分數 1.0 的辨識器時，程序就會立即結束。

Bot Framework SDK 包含一個[範例](#)，該範例示範如何藉由實作 [IDisambiguateRouteHandler](#) 來提供自訂去除混淆邏輯。

後續步驟

使用規則運算式和檢查訊息內容的邏輯可能會變複雜，尤其是在 Bot 的對話流程為開放式時。為了協助 Bot 處理來自使用者的各種文字和語音輸入，您可以使用 [LUIS](#) 等意圖辨識服務，將自然語言理解新增到 Bot。

使用 LUIS 辨識意圖和實體

使用 LUIS 辨識意圖和實體

2019/2/28 • [Edit Online](#)

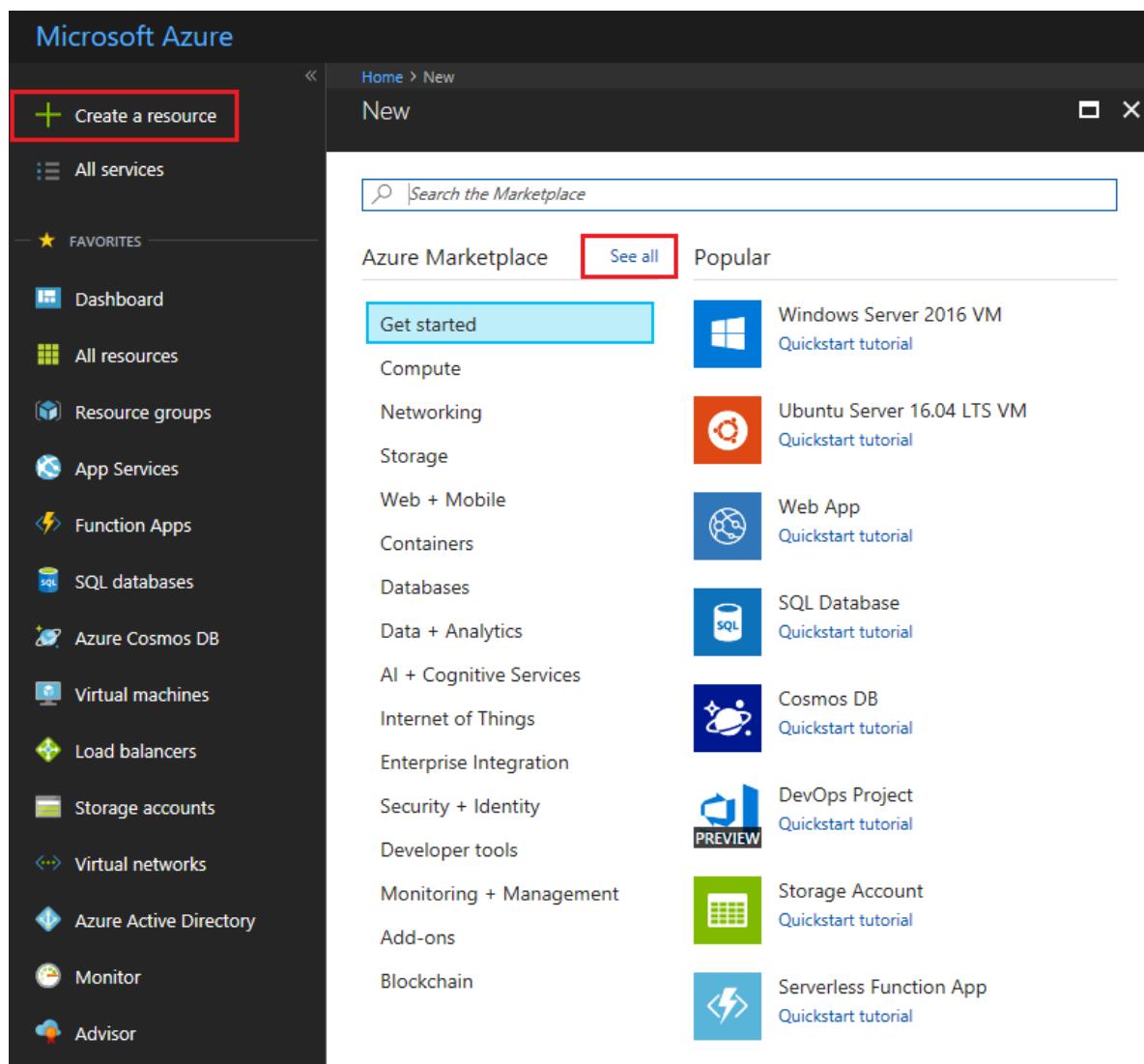
NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

本文使用要用來記筆記的 Bot 範例，來示範 Language Understanding (LUIS) 如何協助您的 Bot 適當地回應自然語言輸入。Bot 會藉由識別使用者的意圖，來偵測他們想要做什麼。此意圖是從語音或文字輸入，或是語句來判斷的。此意圖會將語句對應到 Bot 所採取的動作，例如叫用對話。Bot 可能也需要擷取實體，其為語句中的重要字組。有時，實體必須滿足意圖。在記筆記的 Bot 範例中，`Notes.Title` 實體會識別每個筆記的標題。

使用 Bot 服務來建立 Language Understanding Bot

- 在 [Azure 入口網站](#) 中，選取功能表刀鋒視窗中的 [建立新資源]，然後按一下 [查看全部]。

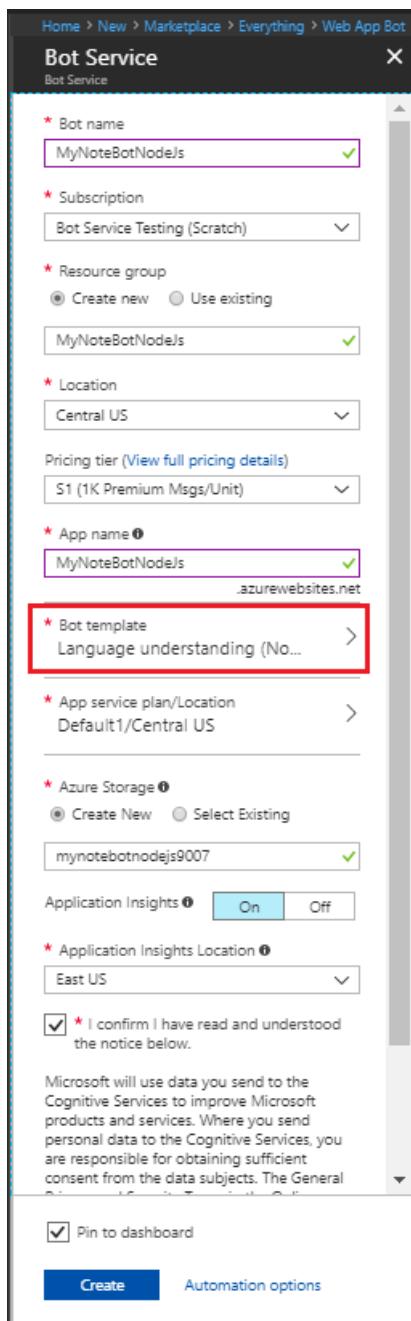


- 在搜尋方塊中，搜尋 **Web 應用程式 Bot**。

The screenshot shows the Azure Marketplace interface. On the left, a sidebar lists categories: Everything, Compute, Networking, Storage, Web + Mobile, Databases, Data + Analytics, and AI + Cognitive Services. The 'Everything' category is selected and highlighted in light blue. On the right, the main pane displays search results for 'Everything'. At the top right of this pane is a 'Filter' icon. Below it is a search bar with the placeholder 'Search Everything'. Three items are listed: 'Functions Bot' (Microsoft), 'Bot Channels Registration' (Microsoft), and 'Web App Bot' (Microsoft). The 'Web App Bot' item is highlighted with a red border.

3. 在 [Bot 服務] 刀鋒視窗中提供必要資訊，然後按一下 [建立]。這會建立 Bot 服務和 LUIS 應用程式，並將其部署到 Azure。

- 將 [應用程式名稱] 設定為您 Bot 的名稱。將 Bot 部署到雲端時，此名稱會用來作為子網域（例如 mynotesbot.azurewebsites.net）。此名稱也會用來作為與您 Bot 相關聯的 LUIS 應用程式名稱。複製它以在稍後用來尋找與 Bot 相關聯的 LUIS 應用程式。
- 選取訂用帳戶、[資源群組](#)、App Service 方案，以及位置。
- 針對 [Bot 範本] 欄位，選取 [Language Understanding (Node.js)] 範本。



- 選取方塊以確認服務條款。

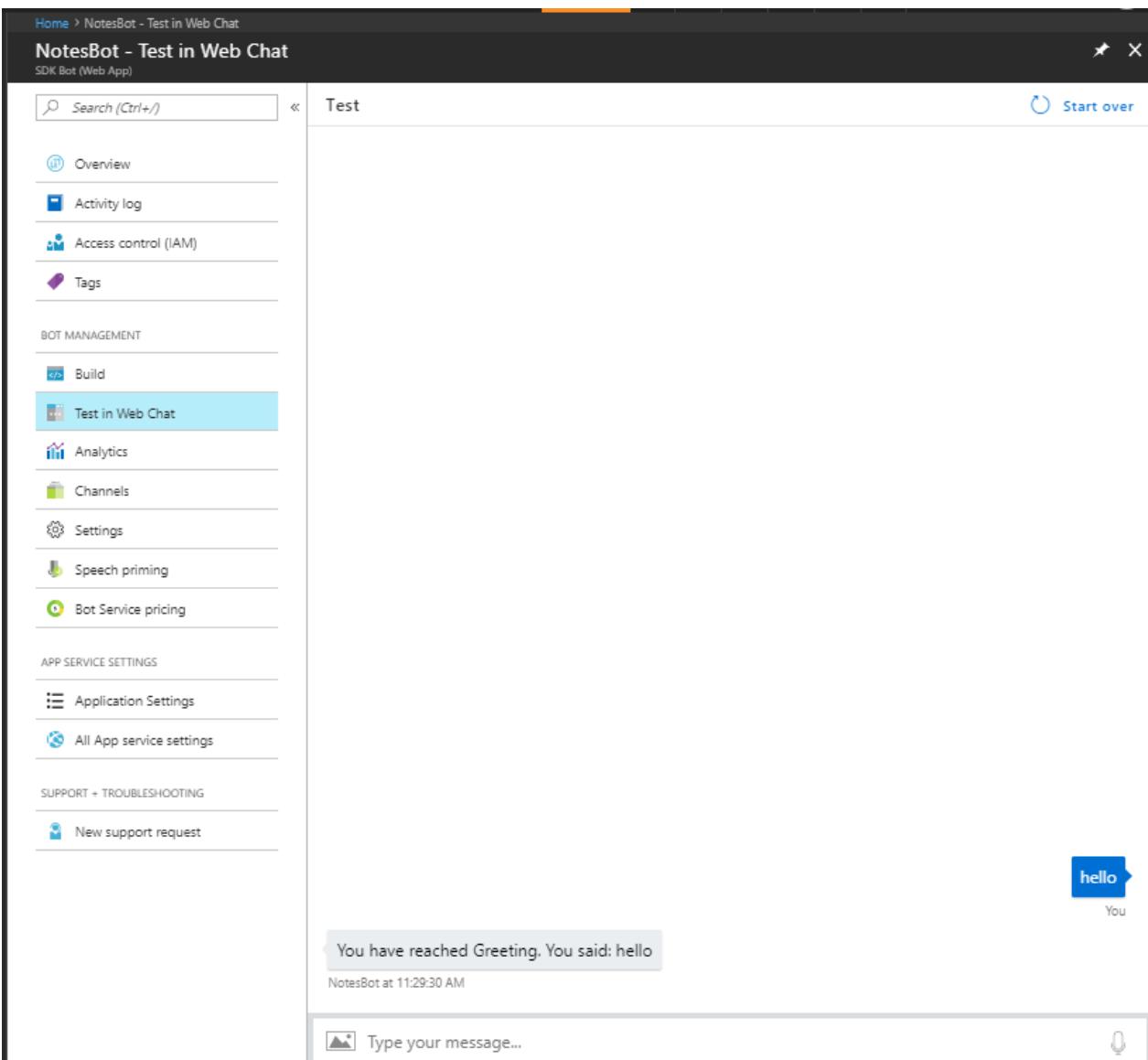
4. 確認已部署 Bot 服務。

- 按一下 [通知] (位於 Azure 入口網站頂端邊緣的鈴鐺圖示)。通知會從 [部署已開始] 變更為 [部署成功]。
- 在通知變更為 [部署成功] 之後，按一下該通知上的 [前往資源]。

測試聊天機器人

勾選 [通知] 來確認已部署 Bot。通知會從 [部署進行中] 變更為 [部署成功]。按一下 [前往資源] 按鈕以開啟 Bot 的資源刀鋒視窗。

註冊 Bot 後，按一下 [在網路聊天中測試] 以開啟 [網路聊天] 窗格。在網路聊天中輸入 "hello"。



Bot 會說出 "You have reached Greeting. You said: hello" 來作為回應。這確認了 Bot 已收到您的訊息，並已將其傳遞給它所建立的預設 LUIS 應用程式。此預設 LUIS 應用程式偵測到 Greeting 意圖。

修改 LUIS 應用程式

使用您用來登入 Azure 的相同帳戶登入 <https://www.luis.ai>。按一下 [My apps] (我的應用程式)。在應用程式清單中，尋找開頭為當您建立 Bot 服務時，於 [Bot 服務] 刀鋒視窗的 [應用程式名稱] 中指定之名稱的應用程式。

LUIS 應用程式會以下列 4 個意圖作為開頭：Cancel、Greeting、Help 及 None。

下列步驟會新增 Note.Create、Note.ReadAloud 和 Note.Delete 意圖：

1. 在頁面左下角按一下 [預先建置的網域]。尋找 **Note** 網域，然後按一下 [新增網域]。
2. 本教學課程不會使用 **Note** 預先建置的網域中包含的所有意圖。在 [意圖] 頁面上，按一下以下每一個意圖名稱，然後按一下 [刪除意圖] 按鈕。

- Note.ShowNext
- Note.DeleteNoteItem
- Note.Confirm
- Note.Clear
- Note.CheckOffItem
- Note.AddToNote

以下為只應保留在 LUIS 應用程式中的意圖：

- Note.ReadAloud
- Note.Create
- Note.Delete
- None
- 說明
- Greeting
- 取消

The screenshot shows the Microsoft LUIS interface. The top navigation bar includes Language Understanding, My apps, Docs, Pricing, Support, About, and Patti Owens. The user is currently working on the MyNotesApp (V 0.1) app. The main menu has DASHBOARD, BUILD (selected), PUBLISH, and SETTINGS. Below the menu, there's a 'Train' button and a 'Test' button with a back arrow. On the left, a sidebar lists App Assets (Intents selected, Entities, Improve app performance (Review user utterances, Phrase lists)). The main content area is titled 'Intents' with a 'Create new intent' button and a search bar. A table lists the intents with their names and utterance counts:

Name	Utterances
Note.ReadAloud	10
Note.Delete	11
Note.Create	17
None	13
Help	13
Greeting	13
Cancel	14

- 按一下右上角的 [定型] 按鈕，來將您的應用程式定型。
- 按一下上方導覽列中的 [發佈]，以開啟 [發佈] 頁面。按一下 [發佈到生產位置] 按鈕。成功發佈之後，即會將 LUIS 應用程式部署到 [發佈應用程式] 頁面上 [端點] 欄中所顯示的 URL，位於以資源名稱 Starter_Key 開頭的列中。URL 的格式類似此範例：

```
https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx?subscription-key=xxxxxxxxxxxxxxxxxxxxxxxxxxxx&timezoneOffset=0&verbose=true&q=
```
- 此 URL 中的應用程式識別碼和訂用帳戶金鑰會與 [App Service 設定] > [ApplicationSettings] > [應用程式設定]**** 中的 LuisAppId 和 LuisAPIKey 相同。

修改 Bot 程式碼

按一下 [建置]，然後按一下 [開啟線上程式碼編輯器]。

The screenshot shows the Microsoft Bot Framework portal interface. On the left, there's a sidebar with various navigation links: Overview, Activity log, Access control (IAM), Tags, BOT MANAGEMENT, Build (which is highlighted with a red box), Test in Web Chat, Analytics, Channels, Settings, Speech priming, and Bot Service pricing. The main content area is titled 'Choose how to work with your code'. It has two sections: 'Online code editor' (with a link to 'Open online code editor' which is also highlighted with a red box) and 'Download source code' (with links to 'Download zip file' and 'Continuous deployment from source control'). Below these are three steps for continuous deployment: Step 1: Download zip file, Step 2: Create a folder/repo for the source files in your preferred service, and Step 3: Configure continuous deployment.

在程式碼編輯器中，開啟 `app.js`。其中包含下列程式碼：

```

var restify = require('restify');
var builder = require('botbuilder');
var botbuilder_azure = require("botbuilder-azure");

// Setup Restify Server
var server = restify.createServer();
server.listen(process.env.port || process.env.PORT || 3978, function () {
    console.log('%s listening to %s', server.name, server.url);
});

// Create chat connector for communicating with the Bot Framework Service
var connector = new builder.ChatConnector({
    appId: process.env.MicrosoftAppId,
    appPassword: process.env.MicrosoftAppPassword,
    openIdMetadata: process.env.BotOpenIdMetadata
});

// Listen for messages from users
server.post('/api/messages', connector.listen());

/*
 * Bot Storage: This is a great spot to register the private state storage for your bot.
 * We provide adapters for Azure Table, CosmosDb, SQL Azure, or you can implement your own!
 * For samples and documentation, see: https://github.com/Microsoft/BotBuilder-Azure
 */
var tableName = 'botdata';
var azureTableClient = new botbuilder_azure.AzureTableClient(tableName, process.env['AzureWebJobsStorage']);
var tableStorage = new botbuilder_azure.AzureBotStorage({ gzipData: false }, azureTableClient);

// Create your bot with a function to receive messages from the user
// This default message handler is invoked if the user's utterance doesn't
// match any intents handled by other dialogs.
var bot = new builder.UniversalBot(connector, function (session, args) {
    session.send('You reached the default message handler. You said \'%s\'., session.message.text);
});

bot.set('storage', tableStorage);

// Make sure you add code to validate these fields
var luisAppId = process.env.LuisAppId;
var luisAPIKey = process.env.LuisAPIKey;
var luisAPIHostName = process.env.LuisAPIHostName || 'westus.api.cognitive.microsoft.com';

```

```

const LuisModelUrl = 'https://' + luisAPIHostName + '/luis/v2.0/apps/' + luisAppId + '?subscription-key=' +
luisAPIKey;

// Create a recognizer that gets intents from LUIS, and add it to the bot
var recognizer = new builder.LuisRecognizer(LuisModelUrl);
bot.recognizer(recognizer);

// Add a dialog for each intent that the LUIS app recognizes.
// See https://docs.microsoft.com/en-us/bot-framework/nodejs/bot-builder-nodejs-recognize-intent-luis
bot.dialog('GreetingDialog',
  (session) => {
    session.send('You reached the Greeting intent. You said \'%s\'.', session.message.text);
    session.endDialog();
  }
).triggerAction({
  matches: 'Greeting'
})

bot.dialog('HelpDialog',
  (session) => {
    session.send('You reached the Help intent. You said \'%s\'.', session.message.text);
    session.endDialog();
  }
).triggerAction({
  matches: 'Help'
})

bot.dialog('CancelDialog',
  (session) => {
    session.send('You reached the Cancel intent. You said \'%s\'.', session.message.text);
    session.endDialog();
  }
).triggerAction({
  matches: 'Cancel'
})

```

TIP

您也可以在 [Notes Bot 範例](#) 中找到本文所述的範例程式碼。

編輯預設訊息處理常式

Bot 具有預設訊息處理常式。編輯它以符合下列內容：

```

// Create your bot with a function to receive messages from the user.
// This default message handler is invoked if the user's utterance doesn't
// match any intents handled by other dialogs.
var bot = new builder.UniversalBot(connector, function (session, args) {
  session.send("Hi... I'm the note bot sample. I can create new notes, read saved notes to you and delete
notes.");

  // If the object for storing notes in session.userData doesn't exist yet, initialize it
  if (!session.userData.notes) {
    session.userData.notes = {};
    console.log("initializing userData.notes in default message handler");
  }
});

```

處理 Note.Create 意圖

複製下列程式碼，並將它貼在 app.js 結尾：

```

// CreateNote dialog
bot.dialog('CreateNote', [
    function (session, args, next) {
        // Resolve and store any Note.Title entity passed from LUIS.
        var intent = args.intent;
        var title = builder.EntityRecognizer.findEntity(intent.entities, 'Note.Title');

        var note = session.dialogData.note = {
            title: title ? title.entity : null,
        };

        // Prompt for title
        if (!note.title) {
            builder.Prompts.text(session, 'What would you like to call your note?');
        } else {
            next();
        }
    },
    function (session, results, next) {
        var note = session.dialogData.note;
        if (results.response) {
            note.title = results.response;
        }

        // Prompt for the text of the note
        if (!note.text) {
            builder.Prompts.text(session, 'What would you like to say in your note?');
        } else {
            next();
        }
    },
    function (session, results) {
        var note = session.dialogData.note;
        if (results.response) {
            note.text = results.response;
        }

        // If the object for storing notes in session.userData doesn't exist yet, initialize it
        if (!session.userData.notes) {
            session.userData.notes = {};
            console.log("initializing session.userData.notes in CreateNote dialog");
        }
        // Save notes in the notes object
        session.userData.notes[note.title] = note;

        // Send confirmation to user
        session.endDialog('Creating note named "%s" with text "%s"',
            note.title, note.text);
    }
]).triggerAction({
    matches: 'Note.Create',
    confirmPrompt: "This will cancel the creation of the note you started. Are you sure?"
}).cancelAction('cancelCreateNote', "Note canceled.", {
    matches: /^(cancel|nevermind)/i,
    confirmPrompt: "Are you sure?"
});

```

語句中的任何實體都會使用 `args` 參數來傳遞給對話。瀑布圖的第一個步驟會呼叫 `EntityRecognizer.findEntity`，從 LUIS 回應中的任何 `Note.Title` 實體取得筆記的標題。如果 LUIS 應用程式未偵測到 `Note.Title` 實體，Bot 會提示使用者輸入筆記的名稱。瀑布圖的第二個步驟會提示您輸入要包含於筆記中的文字。一旦 Bot 擁有筆記的文字之後，第三個步驟就會使用 `session.userData`，利用標題作為索引鍵，將筆記儲存於 `notes` 物件中。如需關於 `session.userData` 的詳細資訊，請參閱[管理狀態資料](#)。

處理 Note.Delete 意圖

Bot 只會針對 `Note.Create` 意圖檢查標題的 `args` 參數。如果未偵測到任何標題，Bot 就會提示使用者。標題可用來查詢要從 `session.userData.notes` 刪除的筆記。

複製下列程式碼，並將它貼在 `app.js` 結尾：

```
// Delete note dialog
bot.dialog('DeleteNote', [
    function (session, args, next) {
        if (noteCount(session.userData.notes) > 0) {
            // Resolve and store any Note.Title entity passed from LUIS.
            var title;
            var intent = args.intent;
            var entity = builder.EntityRecognizer.findEntity(intent.entities, 'Note.Title');
            if (entity) {
                // Verify that the title is in our set of notes.
                title = builder.EntityRecognizer.findBestMatch(session.userData.notes, entity.entity);
            }

            // Prompt for note name
            if (!title) {
                builder.Prompts.choice(session, 'Which note would you like to delete?',
                    session.userData.notes);
            } else {
                next({ response: title });
            }
        } else {
            session.endDialog("No notes to delete.");
        }
    },
    function (session, results) {
        delete session.userData.notes[results.response.entity];
        session.endDialog("Deleted the '%s' note.", results.response.entity);
    }
]).triggerAction({
    matches: 'Note.Delete'
}).cancelAction('cancelDeleteNote', "Ok - canceled note deletion.", {
    matches: /^(cancel|nevermind)/i
});
```

處理 `Note.Delete` 的程式碼會使用 `noteCount` 函式，來判斷 `notes` 物件是否包含筆記。

將 `noteCount` 協助程式函式貼到 `app.js` 結尾。

```
// Helper function to count the number of notes stored in session.userData.notes
function noteCount(notes) {

    var i = 0;
    for (var name in notes) {
        i++;
    }
    return i;
}
```

處理 Note.ReadAloud 意圖

複製下列程式碼並貼到 `app.js` 中的 `Note.Delete` 處理常式之後：

```

// Read note dialog
bot.dialog('ReadNote', [
    function (session, args, next) {
        if (noteCount(session.userData.notes) > 0) {

            // Resolve and store any Note.Title entity passed from LUIS.
            var title;
            var intent = args.intent;
            var entity = builder.EntityRecognizer.findEntity(intent.entities, 'Note.Title');
            if (entity) {
                // Verify it's in our set of notes.
                title = builder.EntityRecognizer.findBestMatch(session.userData.notes, entity.entity);
            }

            // Prompt for note name
            if (!title) {
                builder.Prompts.choice(session, 'Which note would you like to read?',
                session.userData.notes);
            } else {
                next({ response: title });
            }
        } else {
            session.endDialog("No notes to read.");
        }
    },
    function (session, results) {
        session.endDialog("Here's the '%s' note: '%s'.", results.response.entity,
        session.userData.notes[results.response.entity].text);
    }
]).triggerAction({
    matches: 'Note.ReadAloud'
}).cancelAction('cancelReadNote', "Ok.", {
    matches: /^(cancel|nevermind)/i
});

```

`session.userData.notes` 物件會當成第三個引數傳遞給 `builder.Prompts.choice`，如此一來，提示就會向使用者顯示筆記清單。

您現在已新增適用於新意圖的處理常式，`app.js` 的完整程式碼會包含下列內容：

```

var restify = require('restify');
var builder = require('botbuilder');
var botbuilder_azure = require("botbuilder-azure");

// Setup Restify Server
var server = restify.createServer();
server.listen(process.env.port || process.env.PORT || 3978, function () {
    console.log('%s listening to %s', server.name, server.url);
});

// Create chat connector for communicating with the Bot Framework Service
var connector = new builder.ChatConnector({
    appId: process.env.MicrosoftAppId,
    appPassword: process.env.MicrosoftAppPassword,
    openIdMetadata: process.env.BotOpenIdMetadata
});

// Listen for messages from users
server.post('/api/messages', connector.listen());

/*
 * Bot Storage: This is a great spot to register the private state storage for your bot.
 * We provide adapters for Azure Table, CosmosDb, SQL Azure, or you can implement your own!
 * For samples and documentation, see: https://github.com/Microsoft/BotBuilder-Azure
 */

```

```
var tableName = 'botdata';
var azureTableClient = new botbuilder_azure.AzureTableClient(tableName, process.env['AzureWebJobsStorage']);
var tableStorage = new botbuilder_azure.AzureBotStorage({ gzipData: false }, azureTableClient);

// Create your bot with a function to receive messages from the user.
// This default message handler is invoked if the user's utterance doesn't
// match any intents handled by other dialogs.
var bot = new builder.UniversalBot(connector, function (session, args) {
    session.send("Hi... I'm the note bot sample. I can create new notes, read saved notes to you and delete
notes.");

    // If the object for storing notes in session.userData doesn't exist yet, initialize it
    if (!session.userData.notes) {
        session.userData.notes = {};
        console.log("initializing userData.notes in default message handler");
    }
});

bot.set('storage', tableStorage);

// Make sure you add code to validate these fields
var luisAppId = process.env.LuisAppId;
var luisAPIKey = process.env.LuisAPIKey;
var luisAPIHostName = process.env.LuisAPIHostName || 'westus.api.cognitive.microsoft.com';

const LuisModelUrl = 'https://' + luisAPIHostName + '/luis/v2.0/apps/' + luisAppId + '?subscription-key=' +
luisAPIKey;

// Create a recognizer that gets intents from LUIS, and add it to the bot
var recognizer = new builder.LuisRecognizer(LuisModelUrl);
bot.recognizer(recognizer);

// CreateNote dialog
bot.dialog('CreateNote', [
    function (session, args, next) {
        // Resolve and store any Note.Title entity passed from LUIS.
        var intent = args.intent;
        var title = builder.EntityRecognizer.findEntity(intent.entities, 'Note.Title');

        var note = session.dialogData.note = {
            title: title ? title.entity : null,
        };

        // Prompt for title
        if (!note.title) {
            builder.Prompts.text(session, 'What would you like to call your note?');
        } else {
            next();
        }
    },
    function (session, results, next) {
        var note = session.dialogData.note;
        if (results.response) {
            note.title = results.response;
        }

        // Prompt for the text of the note
        if (!note.text) {
            builder.Prompts.text(session, 'What would you like to say in your note?');
        } else {
            next();
        }
    },
    function (session, results) {
        var note = session.dialogData.note;
        if (results.response) {
            note.text = results.response;
        }
    }
]);
```

```

        }

        // If the object for storing notes in session.userData doesn't exist yet, initialize it
        if (!session.userData.notes) {
            session.userData.notes = {};
            console.log("initializing session.userData.notes in CreateNote dialog");
        }
        // Save notes in the notes object
        session.userData.notes[note.title] = note;

        // Send confirmation to user
        session.endDialog('Creating note named "%s" with text "%s"',
            note.title, note.text);
    }
]).triggerAction({
    matches: 'Note.Create',
    confirmPrompt: "This will cancel the creation of the note you started. Are you sure?"
}).cancelAction('cancelCreateNote', "Note canceled.", {
    matches: /^(cancel|nevermind)/i,
    confirmPrompt: "Are you sure?"
});

// Delete note dialog
bot.dialog('DeleteNote', [
    function (session, args, next) {
        if (noteCount(session.userData.notes) > 0) {
            // Resolve and store any Note.Title entity passed from LUIS.
            var title;
            var intent = args.intent;
            var entity = builder.EntityRecognizer.findEntity(intent.entities, 'Note.Title');
            if (entity) {
                // Verify that the title is in our set of notes.
                title = builder.EntityRecognizer.findBestMatch(session.userData.notes, entity.entity);
            }

            // Prompt for note name
            if (!title) {
                builder.Prompts.choice(session, 'Which note would you like to delete?',
                    session.userData.notes);
            } else {
                next({ response: title });
            }
        } else {
            session.endDialog("No notes to delete.");
        }
    },
    function (session, results) {
        delete session.userData.notes[results.response.entity];
        session.endDialog("Deleted the '%s' note.", results.response.entity);
    }
]).triggerAction({
    matches: 'Note.Delete'
}).cancelAction('cancelDeleteNote', "Ok - canceled note deletion.", {
    matches: /^(cancel|nevermind)/i
});

// Read note dialog
bot.dialog('ReadNote', [
    function (session, args, next) {
        if (noteCount(session.userData.notes) > 0) {

            // Resolve and store any Note.Title entity passed from LUIS.
            var title;
            var intent = args.intent;
            var entity = builder.EntityRecognizer.findEntity(intent.entities, 'Note.Title');
            if (entity) {
                // Verify it's in our set of notes.
                title = builder.EntityRecognizer.findBestMatch(session.userData.notes, entity.entity);
            }
        }
    }
]);

```

```
}

    // Prompt for note name
    if (!title) {
        builder.Prompts.choice(session, 'Which note would you like to read?',
session.userData.notes);
    } else {
        next({ response: title });
    }
} else {
    session.endDialog("No notes to read.");
}
},
function (session, results) {
    session.endDialog("Here's the '%s' note: '%s'.", results.response.entity,
session.userData.notes[results.response.entity].text);
}
]).triggerAction({
    matches: 'Note.ReadAloud'
}).cancelAction('cancelReadNote', "Ok.", {
    matches: /^(cancel|nevermind)/i
});
};

// Helper function to count the number of notes stored in session.userData.notes
function noteCount(notes) {

    var i = 0;
    for (var name in notes) {
        i++;
    }
    return i;
}
```

測試 Bot

在 Azure 入口網站中，按一下 [在網路聊天中測試] 來測試 Bot。試著輸入「建立筆記」、「讀取我的筆記」和「刪除筆記」之類的訊息，來叫用您想要新增到其中的意圖。

Home > NotesBot - Test in Web Chat

NotesBot - Test in Web Chat
SDK Bot (Web App)

Search (Ctrl+ /) Start over

« Test create a note

Overview You

Activity log You

Access control (IAM) You

Tags You

BOT MANAGEMENT

Build You

Test in Web Chat You

Analytics You

Channels You

Settings You

Speech priming You

Bot Service pricing You

APP SERVICE SETTINGS

Application Settings You

All App service settings You

SUPPORT + TROUBLESHOOTING

New support request You

What would you like to call your note?

NotesBot

Grocery List You

What do you want to say in your note?

NotesBot

Milk and eggs You

Created note **Grocery List** that says "Milk and eggs".

NotesBot

create note for me You

What would you like to call your note?

NotesBot

Shopping List You

What do you want to say in your note?

NotesBot

buy light bulbs You

Created note **Shopping List** that says "buy light bulbs".

NotesBot

read notes You

Which note would you like to read?

Type your message... Microphone icon

« »

TIP

如果您發現 Bot 並未總是辨識出正確的意圖或實體，請提供更多範例語句來進行 LUIS 應用程式定型，以改善應用程式的效能。您無須對 Bot 程式碼進行任何修改，即可將 LUIS 應用程式重新定型。請參閱[新增範例語句 \(英文\)](#)和[對您的 LUIS 應用程式進行定型和測試 \(英文\)](#)。

後續步驟

試用 Bot 之後，您會看到辨識器可以觸發中斷目前作用中的對話。允許並處理中斷是一項有彈性的設計，該設計會考量使用者真正執行的動作。深入了解您可與已辨識意圖相關聯的各種動作。

處理使用者動作

處理使用者和對話事件

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

本文會示範 Bot 處理事件的方式，這些事件包括使用者加入對話、將 Bot 新增至連絡人清單，或是當 Bot 從對話中移除時會說**再見**等等。

向加入對話的使用者打招呼

每當有成員加入或離開對話時，Bot Framework 會提供 [conversationUpdate](#) 事件來通知您的 Bot。對話成員可以是使用者或 Bot。

下列程式碼片段可讓 Bot 向對話的新成員打招呼，或在 Bot 從對話中移除時說出**再見**。

```
bot.on('conversationUpdate', function (message) {
  if (message.membersAdded && message.membersAdded.length > 0) {
    // Say hello
    var isGroup = message.address.conversation.isGroup;
    var txt = isGroup ? "Hello everyone!" : "Hello...";
    var reply = new builder.Message()
      .address(message.address)
      .text(txt);
    bot.send(reply);
  } else if (message.membersRemoved) {
    // See if bot was removed
    var botId = message.address.bot.id;
    for (var i = 0; i < message.membersRemoved.length; i++) {
      if (message.membersRemoved[i].id === botId) {
        // Say goodbye
        var reply = new builder.Message()
          .address(message.address)
          .text("Goodbye");
        bot.send(reply);
        break;
      }
    }
  }
});
```

認同新增至連絡人清單

[contactRelationUpdate](#) 事件會告訴 Bot 有使用者將您新增至他們的連絡人清單。

```
bot.on('contactRelationUpdate', function (message) {
  if (message.action === 'add') {
    var name = message.user ? message.user.name : null;
    var reply = new builder.Message()
      .address(message.address)
      .text("Hello %s... Thanks for adding me.", name || 'there');
    bot.send(reply);
  }
});
```

新增首次執行對話方塊

由於不是所有通道都支援 **conversationUpdate** 和 **contactRelationUpdate** 事件，若要向加入對話的使用者打招呼，通用的方式是新增首次執行對話方塊。

在下列範例中，我們已新增函式，以在每次有沒看過的使用者出現時觸發對話方塊。您可以藉由提供動作的 **onFindAction** 處理常式，來自訂動作觸發的方式。

```
// Add first run dialog
bot.dialog('firstRun', function (session) {
  session.userData.firstRun = true;
  session.send("Hello...").endDialog();
}).triggerAction({
  onFindAction: function (context, callback) {
    // Only trigger if we've never seen user before
    if (!context.userData.firstRun) {
      // Return a score of 1.1 to ensure the first run dialog wins
      callback(null, 1.1);
    } else {
      callback(null, 0.0);
    }
  }
});
```

您也可以藉由提供 **onSelectAction** 處理常式來自訂動作在觸發後的行為。針對觸發動作，您可以提供 **onInterrupted** 處理常式，以在中斷發生之前加以攔截。如需詳細資訊，請參閱[處理使用者動作](#)

其他資源

- [conversationUpdate](#)
- [contactRelationUpdate](#)

支援當地語系化

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

Bot Builder 包含豐富的當地語系化系統，以建置可使用多種語言與使用者通訊的 Bot。您 Bot 的所有提示都可以使用儲存在您 Bot 目錄結構中的 JSON 檔案來當地語系化。如果您使用的系統如同執行自然語言處理的 LUIS，則您可以使用個別的模型為您 Bot 支援的每種語言設定 [LuisRecognizer](#)，然後 SDK 就會自動選取符合使用者慣用地區設定的模型。

透過提示使用者來決定地區設定

當地語系化您 Bot 的第一個步驟，是新增識別使用者慣用語言的能力。SDK 提供的 [session.preferredLocale\(\)](#) 方法可儲存並擷取每位使用者的此一偏好。下列範例對話方塊會提示使用者輸入其偏好的語言，然後儲存其選擇。

```
bot.dialog('/localePicker', [
  function (session) {
    // Prompt the user to select their preferred locale
    builder.Prompts.choice(session, "What's your preferred language?", 'English|Español|Italiano');
  },
  function (session, results) {
    // Update preferred locale
    var locale;
    switch (results.response.entity) {
      case 'English':
        locale = 'en';
        break;
      case 'Español':
        locale = 'es';
        break;
      case 'Italiano':
        locale = 'it';
        break;
    }
    session.preferredLocale(locale, function (err) {
      if (!err) {
        // Locale files loaded
        session.endDialog(`Your preferred language is now ${results.response.entity}`);
      } else {
        // Problem loading the selected locale
        session.error(err);
      }
    });
  }
]);
```

使用分析決定地區設定

另一個決定使用者地區設定的方法，是使用類似文字分析 API 的服務，根據使用者傳送訊息所用之文字自動偵測使用者的語言。

下列程式碼片段示範如何將這項服務納入您自己的 Bot。

```

var request = require('request');

bot.use({
  receive: function (event, next) {
    if (event.text && !event.textLocale) {
      var options = {
        method: 'POST',
        url: 'https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/languages?numberOfLanguagesToDetect=1',
        body: { documents: [{ id: 'message', text: event.text }] },
        json: true,
        headers: {
          'Ocp-Apim-Subscription-Key': '<YOUR API KEY>'
        }
      };
      request(options, function (error, response, body) {
        if (!error && body) {
          if (body.documents && body.documents.length > 0) {
            var languages = body.documents[0].detectedLanguages;
            if (languages && languages.length > 0) {
              event.textLocale = languages[0].iso6391Name;
            }
          }
        }
        next();
      });
    } else {
      next();
    }
  }
});

```

一旦您將上述程式碼片段新增至您的 Bot，呼叫 `session.preferredLocale()` 就會自動傳回偵測到的語言。

`preferredLocale()` 的搜尋順序如下：

1. 呼叫 `session.preferredLocale()` 來儲存地區設定。此值儲存在 `session.userData['BotBuilder.Data.PreferredLocale']`。
2. 偵測到的地區設定會指派給 `session.message.textLocale`。
3. Bot 的已設定預設地區設定 (例如：英文 ('en'))。

您可以使用其建構函式設定 Bot 的預設地區設定：

```

var bot = new builder.UniversalBot(connector, {
  localizerSettings: {
    defaultLocale: "es"
  }
});

```

當地語系化提示

Bot Framework SDK 的預設當地語系化系統是以檔案為基礎，允許 Bot 使用儲存在磁碟上的 JSON 檔案支援多種語言。根據預設，當地語系化系統會在 `./locale//index.json` 檔案中搜尋 Bot 的提示，是有效的 IETF 語言標記，代表用來尋找提示的慣用地區設定。

下列螢幕擷取畫面顯示支援三種語言的 Bot 目錄結構：英文、義大利文和西班牙文。

```
1  {
2      "greeting": ["Welcome to the localization sample bot!"],
3      "instructions": "This demo will first ask you to select your preferred language.",
4      "locale_prompt": "What's your preferred language?",
5      "locale_updated": "Your preferred language is now set to %s",
6      "text_prompt": "Prompts.text(): enter some text and see what happens!",
7      "number_prompt": "Prompts.number(): enter a number and see what happens!",
8      "listStyle_prompt": "Prompts.choice() supports a range of options",
9      "choice_prompt": "Prompts.choice(): choose an option from the list",
10     "choice_options": "Option A|Option B|Option C",
11     "confirm_prompt": "Prompts.confirm(): enter 'yes' or 'no'",
12     "time_prompt": "Prompts.time(): the built-in time picker",
13     "input_response": "You entered '%s', did I get that right?",
14     "choice_response": "You chose '%s', nice choice!",
15     "time_response": "JSON for time entered: %s",
16     "demo_finished": ["That's the end of the demo.", "See you next time!"]
17 }
```

檔案的結構是訊息識別碼和當地語系化文字字串的簡單 JSON 對應圖。如果值為陣列而非字串，當使用 `session.localizer.gettext()` 撷取該值時會隨機選擇陣列中的一個提示。

如果您在 `session.send()` 呼叫中傳送的是訊息識別碼，而不是特定語言文字，Bot 就會自動擷取訊息的當地語系化版本：

```
var bot = new builder.UniversalBot(connector, [
  function (session) {
    session.send("greeting");
    session.send("instructions");
    session.beginDialog('/localePicker');
  },
  function (session) {
    builder.Prompts.text(session, "text_prompt");
  }
]);
```

在內部，SDK 會呼叫 `session.preferredLocale()` 以取得使用者的慣用地區設定，然後在 `session.localizer.gettext()` 呼叫中使用它，將訊息識別碼對應至其當地語系化的文字字串。您有時候可能需要手動呼叫當地語系化工具。例如，傳送給 `Prompts.choice()` 的列舉值永遠不會自動當地語系化，因此您需要先手動擷取當地語系化清單，再呼叫提示：

```
var options = session.localizer.gettext(session.preferredLocale(), "choice_options");
builder.Prompts.choice(session, "choice_prompt", options);
```

預設的當地語系化工具會跨多個檔案搜尋訊息識別碼，如果它找不到識別碼（或如果未提供當地語系化檔案），它只會傳回識別碼的文字，以使用者不察和選擇性的方式使用當地語系化檔案。檔案搜尋順序如下：

1. 搜尋 `session.preferredLocale()` 所傳回之地區設定下的 `index.json`。
2. 如果地區設定包含選擇性的子標記，例如 `en-US`，則搜尋 `en` 的根標記。
3. 搜尋 Bot 的已設定預設地區設定。

使用命名空間自訂與當地語系化提示

預設的當地語系化工具支援建立提示的命名空間，以避免訊息識別碼之間發生衝突。您的 Bot 可以覆寫已建立命名空間的提示，以自訂或重寫另一個命名空間的提示。您可以利用此功能自訂 SDK 的內建訊息，讓您新增其他語言的支援，或只重寫 SDK 的目前訊息。例如，您可以變更 SDK 的預設錯誤訊息，只要將稱為 **BotBuilder.json** 的檔案新增至您 Bot 的地區設定目錄，然後新增 `default_error` 訊息識別碼項目即可：

The screenshot shows the Visual Studio Code interface with the title "BotBuilder.json - basics-localization - Visual Studio Code". The left sidebar displays the file tree under "EXPLORER". The "OPEN EDITORS" section contains a "BotBuilder.json" file. The "BASICS-LOCALIZATION" section contains a "locale" folder, which further contains "en", "es", and "it" folders. Each of these folders contains a "BotBuilder.json" file, except for "it" which contains an "index.json" file. The "en" folder also contains an "app.js" file. The status bar at the bottom indicates "master*" as the current branch, and the bottom right corner shows "Ln 2, Col 2" with "Spaces: 4", "UTF-8", "CRLF", and "JSON" options.

```
{} BotBuilder.json x
1 {
2     "default_error": "I'm sorry... We encountered a problem with your request."
3 }
```

其他資源

若要深入了解如何當地語系化辨識器，請參閱**辨識意圖**。

使用 backchannel 機制

2019/5/10 • [Edit Online](#)

NOTE

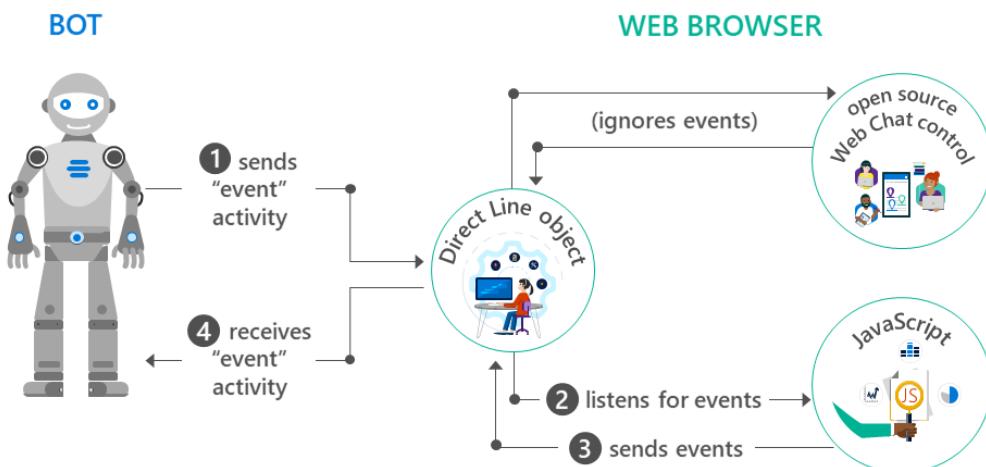
本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

開放原始碼網站聊天控制項使用 [直接線路 API](#) 與 Bot 進行通訊，如此會允許 `activities` 在用戶端與 Bot 之間來回傳送。最常見的活動類型是 `message`，但也有其他類型。比方說，活動類型 `typing` 指明使用者正在輸入或是 bot 正在編譯回覆的工作。

您可以使用 backchannel 機制在用戶端與 Bot 之間交換資訊，而不需要藉由將活動類型設定為 `event` 而將這些資料呈現給使用者。網路聊天控制項會自動忽略 `type="event"` 的任何活動。

逐步解說

開放原始碼網路聊天控制項會使用稱為 [DirectLineJS](#) 的 JavaScript 類別來存取 Direct Line API。控制項可以建立自己 Direct Line 的執行個體，或可與主控頁面共用同一個執行個體。如果控制項與主控頁面共用 Direct Line 的執行個體，則控制項和頁面都能夠傳送及接收活動。下圖顯示網站的高階架構，可使用開放原始碼網路 (聊天) 控制項和 Direct Line API 來支援 Bot 功能。



範例程式碼

在此範例中，Bot 和網頁會使用 backchannel 機制來交換使用者不會察覺到的資訊。Bot 會要求網頁變更其背景色彩，且當使用者按一下網頁上的按鈕時，網頁會通知 Bot。

NOTE

本文中的程式碼片段是來自 [backchannel 範例](#) (英文) 和 [backchannel Bot](#) (英文)。

用戶端程式碼

首先，網頁會建立 **DirectLine** 物件。

```
var botConnection = new BotChat.DirectLine(...);
```

然後，它在建立 WebChat 執行個體時，會共用 **DirectLine** 物件。

```
BotChat.App({
  botConnection: botConnection,
  user: user,
  bot: bot
}, document.getElementById("BotChatGoesHere"));
```

當使用者按一下網頁上的按鈕時，網頁會張貼「活動」類型的通知，以通知 Bot 有人按一下按鈕。

```
const postButtonMessage = () => {
  botConnection
    .postActivity({type: "event", value: "", from: {id: "me"}, name: "buttonClicked"})
    .subscribe(id => console.log("success"));
}
```

TIP

使用 `name` 和 `value` 屬性，來通訊 Bot 要正確解譯及/或回應事件時，可能需要的任何資訊。

最後，網頁也會接聽來自 Bot 的特定事件。在此範例中，網頁會接聽其中 `type="event"` 和 `name="changeBackground"` 的活動。當它收到這種類型的活動時，會將網頁的背景色彩變更為活動所指定的 `value`。

```
botConnection.activity$
  .filter(activity => activity.type === "event" && activity.name === "changeBackground")
  .subscribe(activity => changeBackgroundColor(activity.value))
```

伺服器端程式碼

[Backchannel Bot](#) 會使用 helper 函式來建立事件。

```
var bot = new builder.UniversalBot(connector,
  function (session) {
    var reply = createEvent("changeBackground", session.message.text, session.message.address);
    session.endDialog(reply);
  }
);

const createEvent = (eventName, value, address) => {
  var msg = new builder.Message().address(address);
  msg.data.type = "event";
  msg.data.name = eventName;
  msg.data.value = value;
  return msg;
}
```

同樣地，Bot 也會接聽來自用戶端的事件。在此範例中，如果 Bot 收到包含 `name="buttonClicked"` 的事件，會傳送訊息給使用者，說明「我看到您已按一下按鈕」。

```
bot.on("event", function (event) {
  var msg = new builder.Message().address(event.address);
  msg.data.textLocale = "en-us";
  if (event.name === "buttonClicked") {
    msg.data.text = "I see that you clicked a button.";
  }
  bot.send(msg);
})
```

其他資源

- [Direct Line API](#)
- [Microsoft Bot Framework WebChat 控制項](#)
- [Backchannel 範例](#)
- [Backchannel Bot](#)

要求付款

2019/5/10 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

如果 Bot 要讓使用者能購買商品，則可在[複合式資訊卡 \(Rich Card\)](#) 中包含一個特別的按鈕，藉此要求付款。本文說明如何使用適用於 Node.js 的 Bot Framework SDK 傳送付款要求。

必要條件

您必須先完成這些必要工作，才可以使用適用於 Node.js 的 Bot Framework SDK 傳送付款要求。

註冊並設定 Bot

將 `MicrosoftAppId` 和 `MicrosoftAppPassword` 的 Bot 環境變數，更新為在[註冊](#)過程中針對 Bot 所產生的應用程式識別碼和密碼值。

NOTE

若要尋找您 Bot 的 **AppID** 和 **AppPassword**，請參閱 [MicrosoftAppID 和 MicrosoftAppPassword](#)。

建立及設定商家帳戶

1. [如果您還沒有 Stripe 帳戶，請建立並啟用一個帳戶。](#)
2. [使用您的 Microsoft 帳戶登入賣方中心。](#)
3. 在賣方中心內，連結您的帳戶與 Stripe。
4. 在賣方中心內，瀏覽至儀表板，並複製 **MerchantID** 的值。
5. 將 `PAYMENTS_MERCHANT_ID` 環境變數更新為您從賣方中心儀表板複製的值。

付款程序概觀

付款程序包含三個不同的部分：

1. Bot 傳送付款要求。
2. 使用者登入 Microsoft 帳戶來提供付款、運送和連絡資訊。回呼會傳送至 Bot 以指出 Bot 何時需要執行某些作業 (更新運送地址、更新運送選項、完成付款)。
3. Bot 會處理收到的回呼，包括更新運送地址、更新運送選項和完成付款。

Bot 只需要實作此程序的步驟一和步驟三，步驟二會在 Bot 的內容之外進行。

付款 Bot 範例

[付款 Bot](#) 範例提供使用 Node.js 傳送付款要求的 Bot 範例。若要查看作用中的這個範例 Bot，您可以在[網路聊天中試用](#)、[將它新增為 Skype 連絡人](#)，或下載付款 Bot 範例並使用 Bot Framework Emulator 在本機執行。

NOTE

若要在網路聊天或 Skype 中使用付款 Bot 範例完成端對端付款程序，您必須在您的 Microsoft 帳戶中指定有效的信用卡或金融卡（亦即，美國發卡機構簽發的有效卡片）。我們不會向您的卡片收取費用，也不會驗證卡片的 CVV（信用卡檢查碼），因為付款 Bot 範例是在測試模式中執行（也就是，`PAYMENTS_LIVEMODE` 在 `.env` 中設定為 `false`）。

本文的後續幾節會在付款 Bot 範例的內容中，說明付款程序的三個部分。

要求付款

Bot 可藉由傳送一則包含**複合式資訊卡 (Rich Card)** 的訊息來向使用者要求付款，該卡片中有一個指定 `type` 為「付款」的按鈕。付款 Bot 範例中的這個程式碼片段會建立一則包含 Hero 卡的訊息，該卡片中有一個 [購買] 按鈕，而使用者可以按一下（或點選）此按鈕來起始付款程序。

```
var bot = new builder.UniversalBot(connector, (session) => {

  catalog.getPromotedItem().then(product => {

    // Store userId for later, when reading relatedTo to resume dialog with the receipt.
    var cartId = product.id;
    session.conversationData[CartIdKey] = cartId;
    session.conversationData[cartId] = session.message.address.user.id;

    // Create PaymentRequest obj based on product information.
    var paymentRequest = createPaymentRequest(cartId, product);

    var buyCard = new builder.HeroCard(session)
      .title(product.name)
      .subtitle(util.format('%s %s', product.currency, product.price))
      .text(product.description)
      .images([
        new builder.CardImage(session).url(product.imageUrl)
      ])
      .buttons([
        new builder.CardAction(session)
          .title('Buy')
          .type(payments.PaymentActionType)
          .value(paymentRequest)
      ]);
    session.send(new builder.Message(session)
      .addAttachment(buyCard));
  });
});
```

在此範例中，按鈕的類型會指定為 `payments.PaymentActionType`，而應用程式將其定義為「付款」。此按鈕的值會由 `createPaymentRequest` 函式填入，該函式會傳回 `PaymentRequest` 物件，其中包含支援的付款方式、詳細資料和選項相關資訊。如需有關實作詳細資料的詳細資訊，請參閱付款 Bot 範例中的 `app.js`。

此螢幕擷取畫面會顯示由上述程式碼片段所產生的 Hero 卡（具有 [購買] 按鈕）。

paymentsample +

← → ⌂ | 🔒 webchat.botframework.com/embed/paymentsample?si= Chat



Scott Gu - Favorite Shirt
USD 1.99
Shiny red, ready to rock on Keynotes

[Buy](#)

paymentsample at 3:33:59 PM

Type your message... 

IMPORTANT

只要使用者可存取 [購買] 按鈕，就可以透過該按鈕起始付款程序。在群組對話的內容中，不可能指定一個按鈕，僅供特定使用者使用。

使用者體驗

當使用者按一下 [購買] 按鈕時，系統會將他或她導向至付款 Web 體驗，以透過其 Microsoft 帳戶提供所有必要的付款、交貨和連絡人資訊。

Confirm and Pay

Pay with

Ship to Select shipping address

Shipping options

Email receipt to

Phone number

Total (USD) [Show details](#) \$1.99*

* - Indicates the cost isn't final

Pay

HTTP 回呼

HTTP 回呼將傳送至 Bot, 指出它應該執行特定作業。每個回呼都是包含下列屬性值的事件：

屬性	值
<code>type</code>	叫用
<code>name</code>	指出 Bot 應該執行的作業類型 (例如：交貨地址更新、交貨選項更新、付款完成)。
<code>value</code>	採用 JSON 格式的要求承載。
<code>relatesTo</code>	說明與付款要求相關聯的通道和使用者。

NOTE

`invoke` 是保留給 Microsoft Bot Framework 使用的特別事件類型。`invoke` 事件的傳送者預期 Bot 會藉由傳送 HTTP 回應來認知回呼。

處理回呼

當您的 Bot 接收到回呼時，它應該確認回呼中所指定的資訊是有效的，並藉由傳送 HTTP 回應來確認該回呼。

交貨地址更新和交貨選項更新回呼

接收「交貨地址更新」或「交貨選項更新」回呼，用戶端會在事件的 `value` 屬性中將目前付款狀態的詳細資料提供給 Bot。如果您是商家，則應將這些回呼視為靜態，根據輸入付款詳細資料，將會計算某些輸出付款詳細資料，而如果用戶端所提供的輸入狀態因任何理由而無效，則回呼會失敗。如果 Bot 判斷指定的現況資訊有效，則只要傳送 HTTP 狀態碼 `200 OK` 以及未經修改的付款詳細資料。或者，Bot 可以先傳送 HTTP 狀態碼 `200 OK` 以及應套用的更新後付款詳細資料，才能處理訂單。在某些情況下，Bot 可能會判斷更新後的資訊無效，所以無法照現況處理訂單。例如，使用者的交貨地址可能會指定產品供應商不出貨的國家/地區。在此情況下，Bot 可以傳送 HTTP 狀態碼 `200 OK` 和一則訊息，其中填入付款詳細資料物件的錯誤屬性。傳送任何 `400` 至 `500` 範圍內的 HTTP 狀態碼會導致客戶發生一般錯誤。

付款完成回呼

接收「付款完成」回呼時，Bot 會在物件的 `value` 屬性中取得一份未經修改的最初付款要求，以及付款回應事件。付款回應物件將包含客戶最終選取的項目以及付款代碼 (Token)。Bot 應藉此機會根據最初付款要求和客戶最終選取的項目，重新計算最終付款要求。假設系統將客戶的選取項目判定為有效，則 Bot 應確認付款權杖標頭中的金額和貨幣，確保其符合最終付款要求。如果 Bot 決定向客戶收費，則只應收取付款權杖標頭中的金額，因為這是客戶所確認的價格。如果 Bot 預期的值和在「付款完成」回呼中收到的值不相符，則可傳送 HTTP 狀態碼 `200 OK` 並將結果欄位設定為 `failure`，藉此讓付款要求失敗。

除了驗證付款詳細資料以外，Bot 也應該先確認可以履行訂單，再起始付款處理。例如，其可驗證所購買的項目是否仍可在庫存中取得。如果值正確無誤，而且付款處理器已順利向付款權杖收取費用，則 Bot 應該以 HTTP 狀態碼 `200 OK` 回應並將結果欄位設為 `success`，以便付款 Web 體驗顯示付款確認。Bot 收到的付款權杖只能由要求付款的商家使用一次，而且必須提交至 Stripe (Bot Framework 目前唯一支援的付款處理器)。傳送任何 `400` 至 `500` 範圍內的 HTTP 狀態碼會導致客戶發生一般錯誤。

付款 Bot 範例中的這個程式碼片段會處理 Bot 所接收的回呼。

```
connector.onInvoke((invoke, callback) => {
    console.log('onInvoke', invoke);

    // This is a temporary workaround for the issue that the channelId for "webchat" is mapped to "directline"
    // in the incoming RelatesTo object
    invoke.relatesTo.channelId = invoke.relatesTo.channelId === 'directline' ? 'webchat' :
    invoke.relatesTo.channelId;

    var storageCtx = {
        address: invoke.relatesTo,
        persistConversationData: true,
        conversationId: invoke.relatesTo.conversation.id
    };

    connector.getData(storageCtx, (err, data) => {
        var cartId = data.conversationData[CartIdKey];
        if (!invoke.relatesTo.user && cartId) {
            // Bot keeps the userId in context.ConversationData[cartId]
            var userId = data.conversationData[cartId];
            invoke.relatesTo.useAuth = true;
            invoke.relatesTo.user = { id: userId };
        }

        // Continue based on PaymentRequest event.
        var paymentRequest = null;
        switch (invoke.name) {
            case payments.Operations.UpdateShippingAddressOperation:
            case payments.Operations.UpdateShippingOptionOperation:
                paymentRequest = invoke.value;

                // Validate address AND shipping method (if selected).
                checkout
                    .validateAndCalculateDetails(paymentRequest, paymentRequest.shippingAddress,
                    paymentRequest.shippingOption)
                    .then(updatedPaymentRequest => {
                        // Return new paymentRequest with updated details.
                        callback(null, updatedPaymentRequest, 200);
                    });
        }
    });
});
```

```

        callback(null, { result: "failure" }, 200),
    }).catch(err => {
        // Return error to onInvoke handler.
        callback(err);
        // Send error message back to user.
        bot.beginDialog(invoke.relatesTo, 'checkout_failed', {
            errorMessage: err.message
        });
    });

    break;

case payments.Operations.PaymentCompleteOperation:
    var paymentRequestComplete = invoke.value;
    paymentRequest = paymentRequestComplete.paymentRequest;
    var paymentResponse = paymentRequestComplete.paymentResponse;

    // Validate address AND shipping method.
    checkout
        .validateAndCalculateDetails(paymentRequest, paymentResponse.shippingAddress,
    paymentResponse.shippingOption)
        .then(updatedPaymentRequest =>
            // Process payment.
            checkout
                .processPayment(updatedPaymentRequest, paymentResponse)
                .then(chargeResult => {
                    // Return success.
                    callback(null, { result: "success" }, 200);
                    // Send receipt to user.
                    bot.beginDialog(invoke.relatesTo, 'checkout_receipt', {
                        paymentRequest: updatedPaymentRequest,
                        chargeResult: chargeResult
                    });
                })
        )
        .catch(err => {
            // Return error to onInvoke handler.
            callback(err);
            // Send error message back to user.
            bot.beginDialog(invoke.relatesTo, 'checkout_failed', {
                errorMessage: err.message
            });
        });
    });

    break;
}
);
);

```

在此範例中，Bot 會檢查內送事件的 `name` 屬性，以識別它需要執行的作業類型，然後呼叫適當的方法來處理回呼。如需有關實作詳細資料的詳細資訊，請參閱[付款 Bot](#)範例中的 `app.js`。

測試付款 Bot

若要完整測試要求付款的 Bot，請[設定](#)其在支援 Bot Framework 付款的管道（如網路聊天和 Skype）上執行。或者，您可以使用 [Bot Framework Emulator](#) 在本機測試 Bot。

TIP

當使用者在付款 Web 體驗期間變更資料或按一下 [支付] 時，會傳送回撥給您的 Bot。因此，您可以自行與付款 Web 體驗互動，來測試 Bot 接收和處理回撥的功能。

在[付款 Bot](#)範例中，`.env` 中的 `PAYMENTS_LIVEMODE` 環境變數可決定「付款完成」回呼將包含模擬的付款代碼或實際的

付款代碼。如果 `PAYMENTS_LIVEMODE` 設為 `false`，則標頭會新增至 Bot 的輸出付款要求，以指出 Bot 處於測試模式，而且「付款完成」回呼將包含無法收費的模擬付款代碼。如果 `PAYMENTS_LIVEMODE` 設為 `true`，則指出 Bot 處於測試模式的標頭會從 Bot 的輸出付款要求中省略，而且「付款完成」回呼會包含 Bot 將提交至 Stripe 進行付款處理的實際付款權杖。這會是向指定的付款工具收費的真實交易。

其他資源

- [付款 Bot 範例](#)
- [將複合式資訊卡 \(Rich Card\) 附件新增至訊息](#)
- [在 W3C 進行網路付款](#)

使用 Azure 搜尋服務建立資料驅動體驗

2019/3/7 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

您可以將 [Azure 搜尋服務](#)新增至 Bot，以協助使用者巡覽大量內容，並為 Bot 的使用者建立資料驅動的探索體驗。

Azure 搜尋服務是一項 Azure 服務，可提供關鍵字搜尋、內建語言、自訂評分、多面向導覽等等。Azure 搜尋服務也可以對各種來源（包括 Azure SQL DB、DocumentDB、Blob 儲存體和表格儲存體）的內容編製索引。它支援「推送」其他資料來源的編製索引，而且可以開啟 PDF、Office 文件，以及其他包含非結構化資料的格式。收集後，內容就會進入 Azure 搜尋服務索引，然後 Bot 就可加以查詢。

安裝相依性

在命令提示字元中，巡覽至 Bot 的專案目錄，並使用 Node Package Manager (NPM) 安裝下列模組：

- [bluebird](#)
- [lodash](#)
- [要求](#)

必要條件

下列是必要項目：

- 具備 Azure 訂用帳戶和 Azure 搜尋服務主要金鑰。您可以在 Azure 入口網站中找到這個值。
- 將 [SearchDialogLibrary](#) 程式庫複製到 Bot 的專案目錄。此程式庫包含供使用者進行搜尋，但可以視需要自訂以符合 Bot 的一般對話。
- 將 [SearchProviders](#) 程式庫複製到 Bot 的專案目錄。此程式庫包含建立要求並將其提交至 Azure 搜尋服務所需的所有元件。

連線到 Azure 服務

在 Bot 的主程式檔中（例如：app.js），建立兩個先前已安裝之程式庫的參考路徑。

```
var SearchLibrary = require('./SearchDialogLibrary');
var AzureSearch = require('./SearchProviders/azure-search');
```

將下列程範例程式新增至 Bot。在 `AzureSearch` 物件中，將您自己的 Azure 搜尋服務設定傳入 `.create` 方法。在執行階段，這會將 Bot 繫結至 Azure 搜尋服務，並等待 `Promise` 形式的已完成使用者查詢。

```
// Azure Search
var azureSearchClient = AzureSearch.create('Your-Azure-Search-Service-Name', 'Your-Azure-Search-Primary-Key',
  'Your-Azure-Search-Service-Index');
var ResultsMapper = SearchLibrary.defaultResultsMapper(ToSearchHit);
```

`azureSearchClient` 參考會建立 Azure 搜尋服務模型，該模型則透過 Bot 的 `.env` 設定傳遞 Azure 服務的授權設

定。`ResultsMapper` 會剖析 Azure 回應物件，並對應我們在 `ToSearchHit` 方法中定義的資料。如需這個方法的實作範例，請參閱 [Azure 搜尋服務回應之後](#)。

註冊搜尋程式庫

您可以在 `SearchLibrary` 模組本身中直接自訂搜尋對話方塊。`SearchLibrary` 會執行大部分的繁重工作，包括呼叫 Azure 搜尋服務。

請在 Bot 的主程式檔中新增下列程式碼，以便向 Bot 登錄搜尋對話程式庫。

```
bot.library(SearchLibrary.create({
    multipleSelection: true,
    search: function (query) { return azureSearchClient.search(query).then(ResultsMapper); },
    refiners: ['refiner1', 'refiner2', 'refiner3'], // customize your own refiners
    refineFormatter: function (refiners) {
        return _.zipObject(
            refiners.map(function (r) { return 'By ' + _.capitalize(r); }),
            refiners);
    }
}));
```

`SearchLibrary` 不僅會儲存所有與搜尋相關的對話，還會採用使用者查詢以提交至 Azure 搜尋服務。您必須在 `refiners` 陣列中定義自己的精簡器，以指定您想要讓使用者縮小搜尋結果範圍或篩選搜尋結果的實體。

建立搜尋對話

您可以在需要時選擇建構對話。設定 Azure 搜尋服務對話的唯一需求是透過 `SearchLibrary` 物件叫用 `.begin` 方法，以傳入 Bot Framework SDK 所產生的 `session` 物件。

```
function (session) {
    // Trigger Azure Search dialogs
    SearchLibrary.begin(session);
},
function (session, args) {
    // Process selected search results
    session.send(
        'Search Completed!',
        args.selection.map( )); // format your response
}
```

如需對話的詳細資訊，請參閱 [使用對話來管理交談](#)。

Azure 搜尋服務回應之後

Azure 搜尋服務成功解析之後，您現在需要從回應物件儲存想要的資料，並以對使用者有意義的方式顯示它。

TIP

請考慮包含 `util` 模組。它會協助您進行格式化，並對應來自 Azure 搜尋服務的回應。

在 Bot 的主程式檔中，建立 `ToSearchHit` 方法。這個方法會傳回一個物件，而該物件將格式化 Azure 回應中所需的相關資料。下列程式碼示範如何在 `ToSearchHit` 方法中定義您自己的參數。

```
function ToSearchHit(azureResponse) {
    return {
        // define your own parameters
        key: azureResponse.id,
        title: azureResponse.title,
        description: azureResponse.description,
        imageUrl: azureResponse.thumbnail
    };
}
```

此動作完成之後，您只需要向使用者顯示資料即可。

在 **SearchDialogLibrary** 專案的 **index.js** 檔案中，`searchHitAsCard` 方法會剖析來自 Azure 搜尋服務的每個回應，並建立要向使用者顯示的新卡片物件。您在 Bot 主程式檔之 `ToSearchHit` 方法中所定義的欄位，需要與 `searchHitAsCard` 方法中的屬性進行同步處理。

以下顯示如何及在何處使用您透過 `ToSearchHit` 方法定義的參數，來建置要呈現給使用者的卡片附件 UI。

```
function searchHitAsCard(showSave, searchHit) {
    var buttons = showSave
        ? [new builder.CardAction().type('imBack').title('Save').value(searchHit.key)]
        : [];

    var card = new builder.HeroCard()
        .title(searchHit.title)
        .buttons(buttons);

    if (searchHit.description) {
        card.subtitle(searchHit.description);
    }

    if (searchHit.imageUrl) {
        card.images([new builder.CardImage().url(searchHit.imageUrl)]);
    }

    return card;
}
```

範例程式碼

如需兩個示範如何使用適用於 Node.js 的 Bot Framework SDK 支援 Azure 搜尋服務的完整範例，請參閱 GitHub 中的[不動產 Bot 範例](#)或[作業清單 Bot 範例](#)。

其他資源

- [Azure 搜尋服務](#)
- [Node Util](#)
- [對話](#)

對 Bot 進行偵錯

2019/5/10 • [Edit Online](#)

本文說明如何使用整合式開發環境 (IDE), 例如 Visual Studio 或 Visual Studio Code 和 Bot Framework 模擬器等, 對 Bot 進行偵錯。儘管您可以使用這些方法對任何 Bot 進行本機偵錯, 本文會使用在快速入門中建立的 [C# bot](#) 或 [Javascript bot](#)。

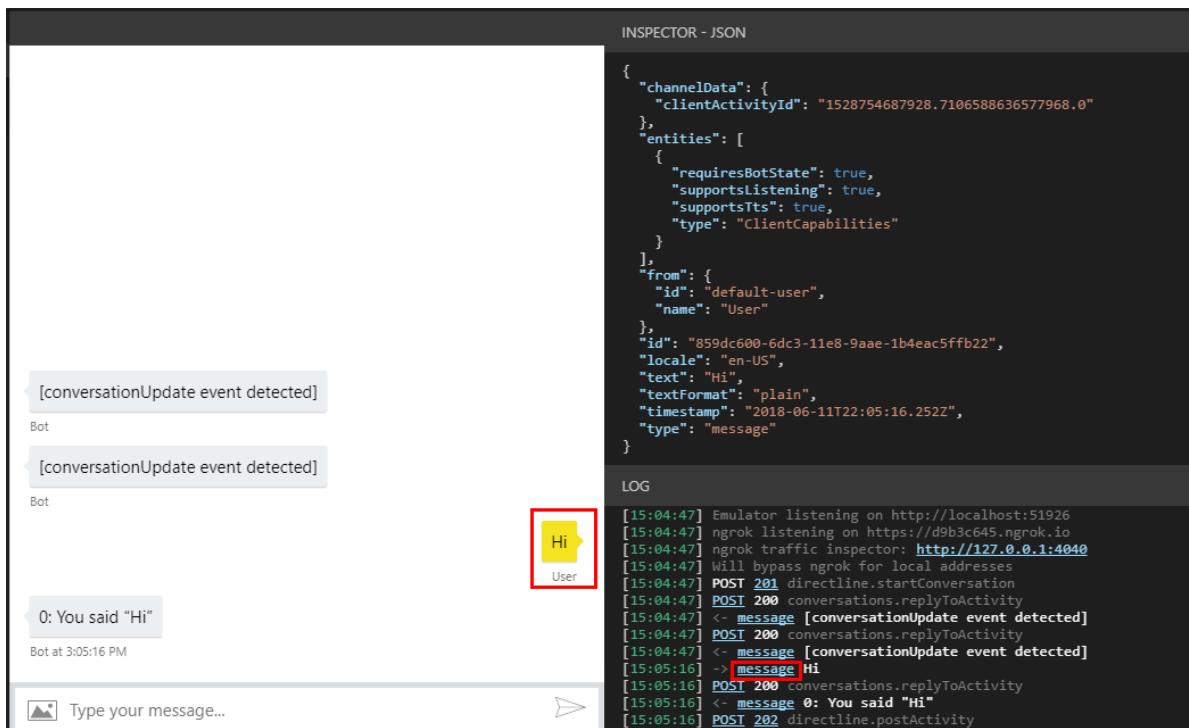
必要條件

- 下載並安裝 [Bot Framework 模擬器](#) (英文)。
- 下載並安裝 [Visual Studio Code](#) 或 [Visual Studio](#) (Community 版本或更新版本)。

使用命令列和模擬器對 JavaScript Bot 進行偵錯

若要使用命令列執行 JavaScript Bot 並使用模擬器測試 Bot, 請執行下列操作：

1. 從命令列將目錄變更至 Bot 專案目錄。
2. 執行命令 **node app.js** 以啟動 Bot。
3. 啟動模擬器, 然後連線到 Bot 的端點 (例如:<http://localhost:3978/api/messages>)。如果這是您第一次執行 Bot, 請按一下 [檔案] > [新的 Bot], 並遵循畫面上的指示。否則, 請按一下 [檔案] > [開啟 Bot] 以開啟現有的 Bot。由於這個 Bot 在本機電腦上執行, 您可以將 **MicrosoftAppId** 和 **MicrosoftAppPassword** 欄位留白。如需詳細資訊, 請參閱[使用模擬器進行偵錯](#)。
4. 在模擬器中, 向 Bot 傳送訊息 (例如:傳送「嗨」訊息)。
5. 使用模擬器視窗右側的 [偵測器] 和 [記錄] 面板來對您的 Bot 進行偵錯。比方說, 按一下任何一個訊息泡泡 (例如:在下方螢幕擷取畫面中的「嗨」訊息泡泡), 在 [偵測器] 面板中會向您顯示該訊息的詳細資料。您可以使用它來檢視要求和回應, 因訊息會在模擬器和 Bot 之間交換。或者, 您也可以按一下 [記錄] 面板中任何連結的文字, 以在 [偵測器] 面板中檢視詳細資料。



使用 Visual Studio Code 的中斷點對 JavaScript Bot 進行偵錯

在 Visual Studio Code 中，您可以設定中斷點，並在偵錯模式中執行 Bot 以逐步執行程式碼。若要在 VS Code 中設定中斷點，請執行下列操作：

1. 啟動 VS Code 並開啟 Bot 專案資料夾。
2. 在功能表列中，按一下 [偵錯] 然後按一下 [開始偵錯]。如果系統提示您選取執行階段引擎來執行程式碼，請選取 **Node.js**。此時，Bot 正在本機執行。
3. 視需要設定中斷點。在 VS Code 中，您可以將滑鼠停留在行號左側的資料行來設定中斷點。會出現一個小紅點。按一下小紅點，即可設定中斷點。如果再按一下，則會移除中斷點。

The screenshot shows the Visual Studio Code interface with the 'app.js' file open. The 'DEBUG' tab is selected. On the left, the 'BREAKPOINTS' sidebar shows a red dot next to the line number 32, indicating a breakpoint is set there. The code itself is a Node.js script for a bot, with the breakpoint located in the 'server.post' handler logic.

```
5  const restify = require('restify');
6
7  // Create server
8  let server = restify.createServer();
9  server.listen(process.env.port || process.env.PORT || 3978, function () {
10    console.log(`${server.name} listening to ${server.url}`);
11 });
12
13 // Create adapter
14 const adapter = new BotFrameworkAdapter({
15   appId: process.env.MicrosoftAppId,
16   appPassword: process.env.MicrosoftAppPassword
17 });
18
19 // Add state middleware
20 const storage = new MemoryStorage();
21 const convoState = new ConversationState(storage);
22 const userState = new UserState(storage);
23 adapter.use(new BotStateSet(convosoState, userState));
24
25 // Listen for incoming requests
26 server.post('/api/messages', (req, res) => {
27   // Route received request to adapter for processing
28   adapter.processActivity(req, res, (context) => {
29     if (context.activity.type === 'message') {
30       const state = convoState.get(context);
31       const count = state.count === undefined ? state.count = 0 : ++state.count;
32       return context.sendActivity(`${count}: You said "${context.activity.text}"`);
33     } else {
34       return context.sendActivity(`[${context.activity.type} event detected]`);
35     }
36   });
37 });
```

4. 啟動 Bot Framework 模擬器並連接到您的 Bot，如上一節所述。
5. 在模擬器中，向 Bot 傳送訊息（例如：傳送「嗨」訊息）。執行會停在您放置中斷點的那一行。

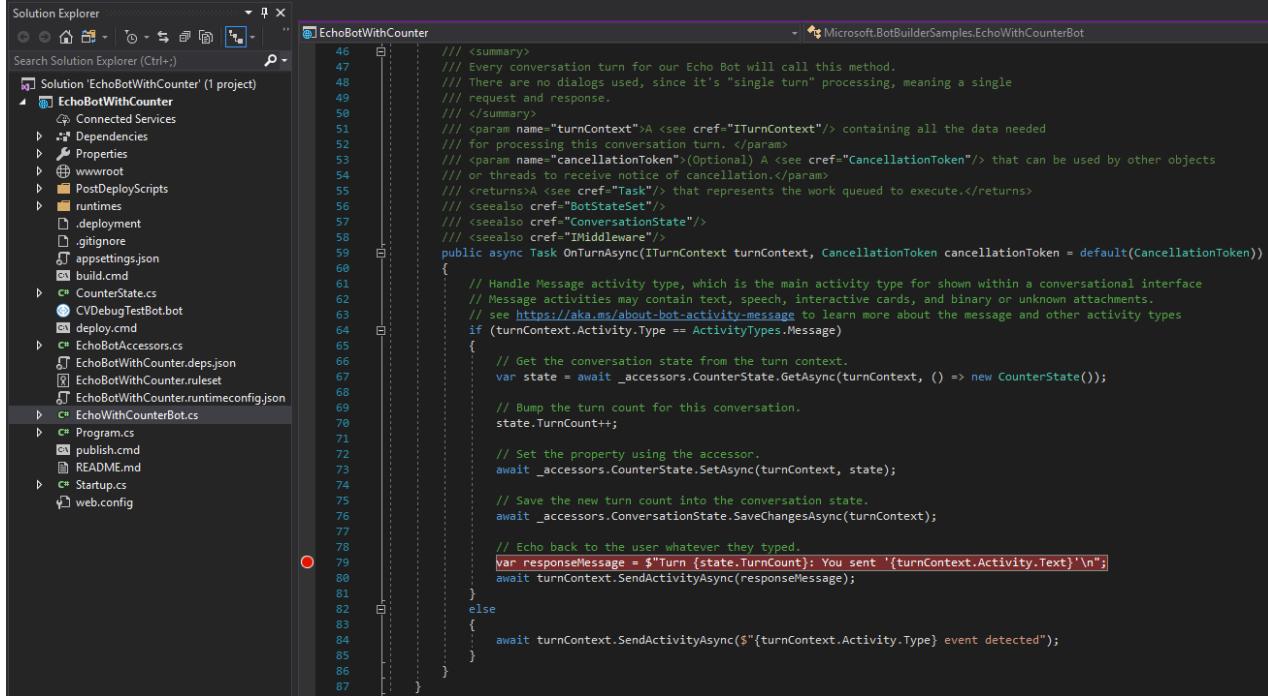
The screenshot shows the Visual Studio Code interface with the 'app.js' file open. The 'DEBUG' tab is selected, and the status bar indicates 'PAUSED ON BREAKPOINT'. The 'BREAKPOINTS' sidebar shows a red dot next to the line number 32. The code is the same as in the previous screenshot, but now the execution has stopped at the breakpoint, as indicated by the yellow highlighting of the line and the pause icon in the status bar.

```
5  const restify = require('restify');
6
7  // Create server
8  let server = restify.createServer();
9  server.listen(process.env.port || process.env.PORT || 3978, function () {
10    console.log(`${server.name} listening to ${server.url}`);
11 });
12
13 // Create adapter
14 const adapter = new BotFrameworkAdapter({
15   appId: process.env.MicrosoftAppId,
16   appPassword: process.env.MicrosoftAppPassword
17 });
18
19 // Add state middleware
20 const storage = new MemoryStorage();
21 const convoState = new ConversationState(storage);
22 const userState = new UserState(storage);
23 adapter.use(new BotStateSet(convosoState, userState));
24
25 // Listen for incoming requests
26 server.post('/api/messages', (req, res) => {
27   // Route received request to adapter for processing
28   adapter.processActivity(req, res, (context) => {
29     if (context.activity.type === 'message') {
30       const state = convoState.get(context);
31       const count = state.count === undefined ? state.count = 0 : ++state.count;
32       return context.sendActivity(`${count}: You said "${context.activity.text}"`);
33     } else {
34       return context.sendActivity(`[${context.activity.type} event detected]`);
35     }
36   });
37 });
```

使用 Visual Studio 的中斷點對 C# Bot 進行偵錯

在 Visual Studio (VS) 中，您可以設定中斷點，並在偵錯模式中執行 Bot 以逐步執行程式碼。若要在 VS 中設定中斷點，請執行下列操作：

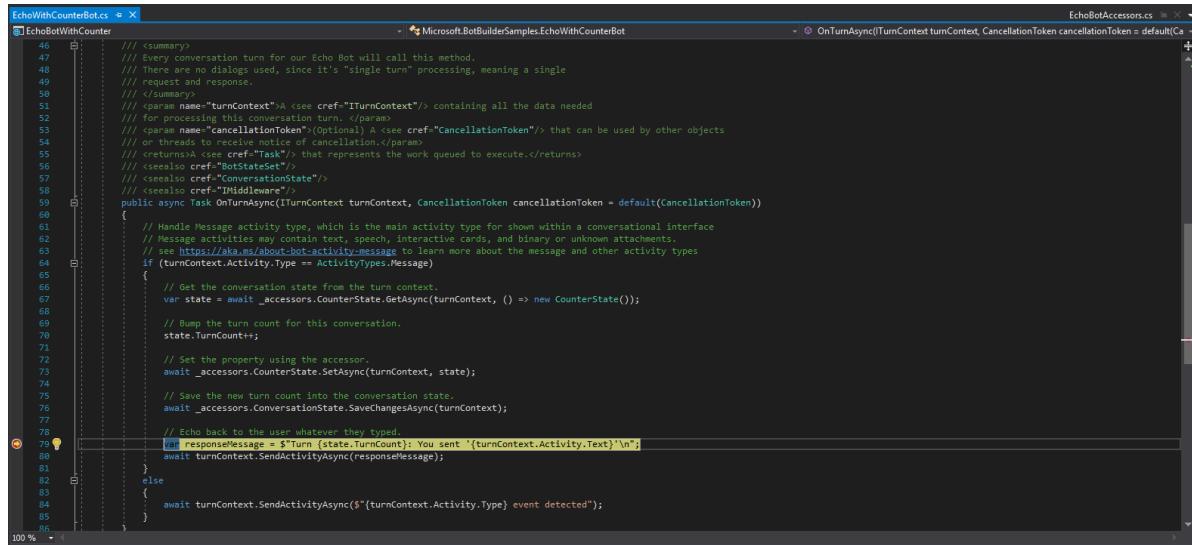
1. 瀏覽至您的 Bot 資料夾，然後開啟 **.sln** 檔案。這會在 VS 中開啟該解決方案。
2. 按一下功能表列中的 [建置]，然後按一下 [建置解決方案]。
3. 在 [方案總管] 中，按一下 **EchoWithCounterBot.cs**。此檔案會定義主要 Bot 邏輯。視需要設定中斷點。在 VS 中，您可以將滑鼠停留在行號左側的資料行來設定中斷點。會出現一個小紅點。按一下小紅點，即可設定中斷點。如果再按一下，則會移除中斷點。
4. 在功能表列中，按一下 [偵錯]，然後按一下 [開始偵錯]。此時，Bot 正在本機執行。



```
46     /// <summary>
47     /// Every conversation turn for our Echo Bot will call this method.
48     /// There are no dialogs used, since it's "single turn" processing, meaning a single
49     /// request and response.
50     /// </summary>
51     /// <param name="turnContext">A <see cref="ITurnContext"/> containing all the data needed
52     /// for processing this conversation turn. </param>
53     /// <param name="cancellationToken">(Optional) A <see cref="CancellationToken"/> that can be used by other objects
54     /// or threads to receive notice of cancellation.</param>
55     /// <returns>A <see cref="Task"/> that represents the work queued to execute.</returns>
56     /// <sealso cref="BotStateSet"/>
57     /// <sealso cref="ConversationState"/>
58     /// <sealso cref="IMiddleware"/>
59     public async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken = default(CancellationToken))
60     {
61         // Handle Message activity type, which is the main activity type for shown within a conversational interface
62         // Message activities may contain text, speech, interactive cards, and binary or unknown attachments.
63         // See https://aka.ms/about-bot-activity-message to learn more about the message and other activity types
64         if (turnContext.Activity.Type == ActivityTypes.Message)
65         {
66             // Get the conversation state from the turn context.
67             var state = await _accessors.CounterState.GetAsync(turnContext, () => new CounterState());
68
69             // Bump the turn count for this conversation.
70             state.TurnCount++;
71
72             // Set the property using the accessor.
73             await _accessors.CounterState.SetAsync(turnContext, state);
74
75             // Save the new turn count into the conversation state.
76             await _accessors.ConversationState.SaveChangesAsync(turnContext);
77
78             // Echo back to the user whatever they typed.
79             var responseMessage = $"Turn {state.TurnCount}: You sent '{turnContext.Activity.Text}'\n";
80             await turnContext.SendActivityAsync(responseMessage);
81         }
82         else
83         {
84             await turnContext.SendActivityAsync($"{turnContext.Activity.Type} event detected");
85         }
86     }
87 }
```

7. 啟動 Bot Framework 模擬器並連接到您的 Bot，如上一節所述。

8. 在模擬器中，向 Bot 傳送訊息（例如：傳送「嗨」訊息）。執行會停在您放置中斷點的那一行。



```
46     /// <summary>
47     /// Every conversation turn for our Echo Bot will call this method.
48     /// There are no dialogs used, since it's "single turn" processing, meaning a single
49     /// request and response.
50     /// </summary>
51     /// <param name="turnContext">A <see cref="ITurnContext"/> containing all the data needed
52     /// for processing this conversation turn. </param>
53     /// <param name="cancellationToken">(Optional) A <see cref="CancellationToken"/> that can be used by other objects
54     /// or threads to receive notice of cancellation.</param>
55     /// <returns>A <see cref="Task"/> that represents the work queued to execute.</returns>
56     /// <sealso cref="BotStateSet"/>
57     /// <sealso cref="ConversationState"/>
58     /// <sealso cref="IMiddleware"/>
59     public async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken = default(CancellationToken))
60     {
61         // Handle Message activity type, which is the main activity type for shown within a conversational interface
62         // Message activities may contain text, speech, interactive cards, and binary or unknown attachments.
63         // See https://aka.ms/about-bot-activity-message to learn more about the message and other activity types
64         if (turnContext.Activity.Type == ActivityTypes.Message)
65         {
66             // Get the conversation state from the turn context.
67             var state = await _accessors.CounterState.GetAsync(turnContext, () => new CounterState());
68
69             // Bump the turn count for this conversation.
70             state.TurnCount++;
71
72             // Set the property using the accessor.
73             await _accessors.CounterState.SetAsync(turnContext, state);
74
75             // Save the new turn count into the conversation state.
76             await _accessors.ConversationState.SaveChangesAsync(turnContext);
77
78             // Echo back to the user whatever they typed.
79             var responseMessage = $"Turn {state.TurnCount}: You sent '{turnContext.Activity.Text}'\n";
80             await turnContext.SendActivityAsync(responseMessage);
81         }
82         else
83         {
84             await turnContext.SendActivityAsync($"{turnContext.Activity.Type} event detected");
85         }
86     }
87 }
```

針對取用方案 C 進行偵錯# Functions Bot

比起一般的 C# 應用程式，Bot Service 中的取用方案無伺服器 C# 環境與 Node.js 更相似，因為它需要執行階段主機，正如同 Node 引擎。在 Azure 中，執行階段是雲端裝載環境的一部分，但您必須在本機桌面上複寫該環境。

必要條件

在對取用方案 C# Bot 進行偵錯前，您必須先完成這些工作。

- 下載 Bot 的原始程式碼 (從 Azure)，如[設定持續部署](#)中所述。
- 下載並安裝 [Bot Framework 模擬器](#) (英文)。
- 安裝 [Azure Functions CLI](#) (英文)。
- 安裝 [DotNet CLI](#) (英文)。

如果您想要在 Visual Studio 2017 中使用中斷點對程式碼進行偵錯，也必須完成下列工作。

- 下載並安裝 [Visual Studio 2017](#) (Community 版本或更新版本)。
- 下載並安裝[命令工作執行器 Visual Studio 擴充功能](#) (英文)。

NOTE

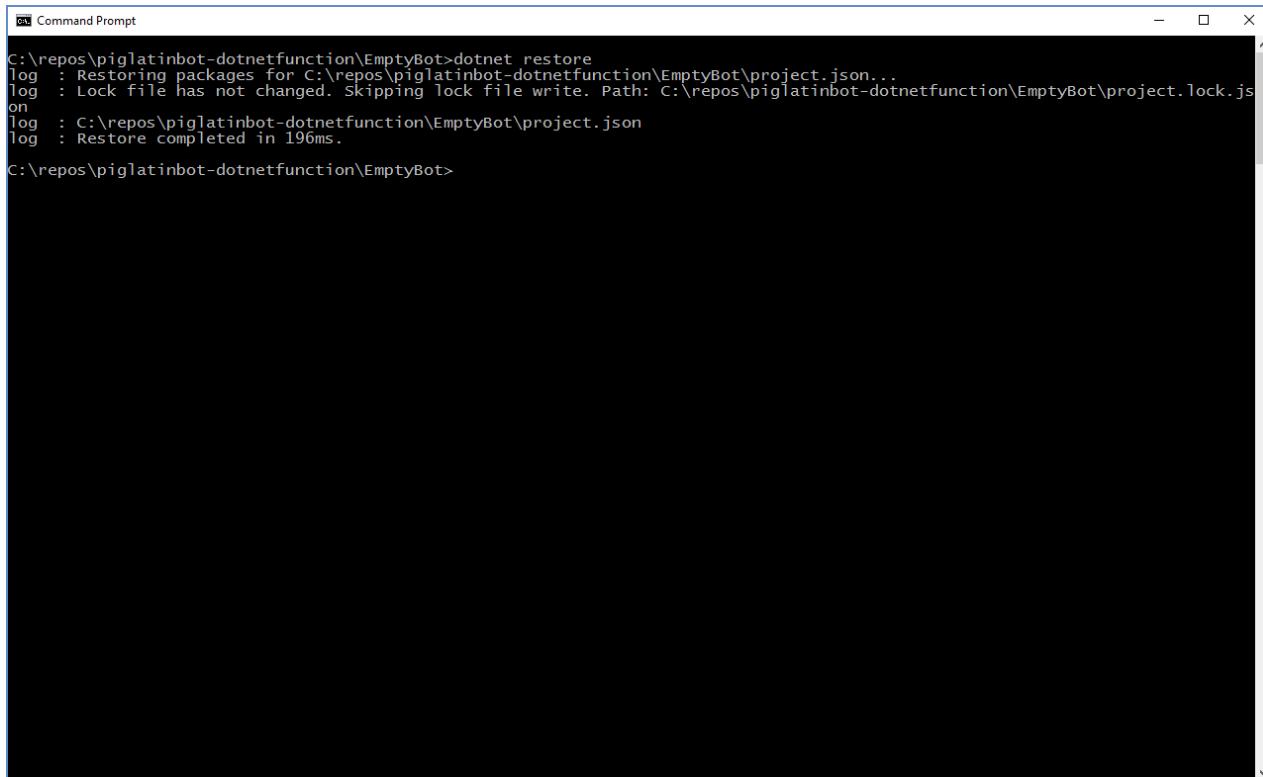
目前不支援 Visual Studio Code。

使用模擬器對取用方案 C# Functions Bot 進行偵錯

在本機對 Bot 進行偵錯最簡單的方法是，啟動 Bot 然後從 Bot Framework 模擬器連線到 Bot。首先，開啟命令提示字元並瀏覽至存放庫中 **project.json** 檔案所在的資料夾。然後，執行命令 `dotnet restore` 以還原 Bot 中所參考的各種套件。

NOTE

Visual Studio 2017 變更了 Visual Studio 處理相依性的方式。Visual Studio 2015 使用 **project.json** 來處理相依性，而 Visual Studio 2017 在 Visual Studio 中載入時則是使用 **.csproj** 模型。如果您使用 Visual Studio 2017，[請將此 .csproj 檔案下載](#)至存放庫中的 **/messages** 資料夾，然後再執行 `dotnet restore` 命令。



```
C:\ repos\piglatinbot-dotnetfunction\EmptyBot>dotnet restore
log : Restoring packages for C:\repos\piglatinbot-dotnetfunction\EmptyBot\project.json...
log : Lock file has not changed. Skipping lock file write. Path: C:\repos\piglatinbot-dotnetfunction\EmptyBot\project.lock.json
log : C:\repos\piglatinbot-dotnetfunction\EmptyBot\project.json
log : Restore completed in 196ms.

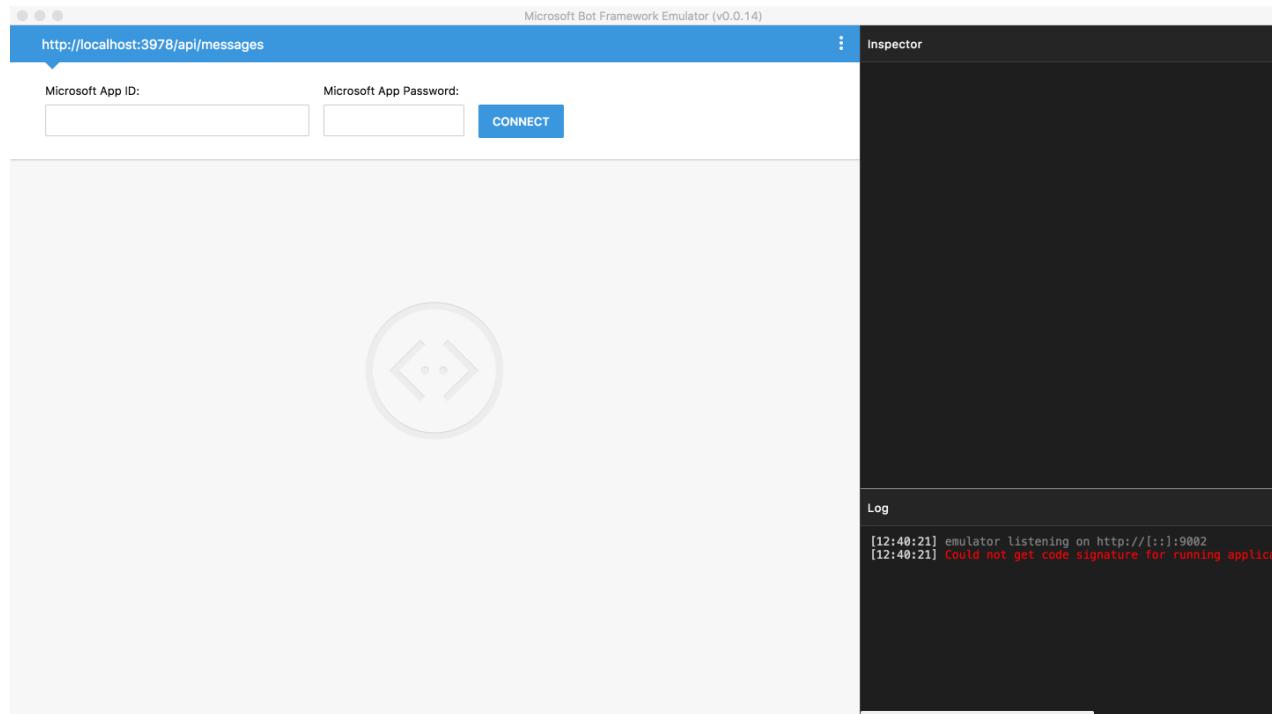
C:\repos\piglatinbot-dotnetfunction\EmptyBot>
```

接下來，執行 `debughost.cmd` 以載入並啟動您的 Bot。

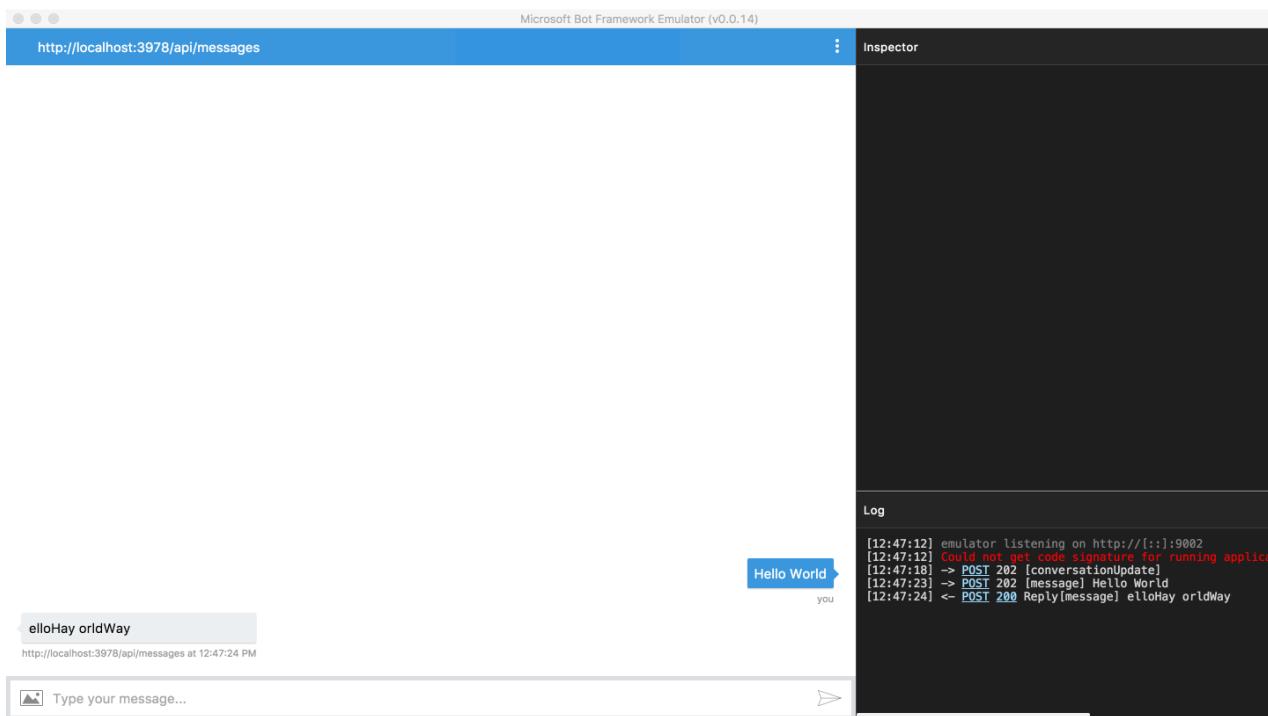
```
C:\ Command Prompt - debughost.cmd
C:\repos\piglatinbot-dotnetfunction\EmptyBot>dotnet restore
log : Restoring packages for C:\repos\piglatinbot-dotnetfunction\EmptyBot\project.json...
log : Lock file has not changed. Skipping lock file write. Path: C:\repos\piglatinbot-dotnetfunction\EmptyBot\project.lock.json
log : C:\repos\piglatinbot-dotnetfunction\EmptyBot\project.json
log : Restore completed in 194ms.

C:\repos\piglatinbot-dotnetfunction\EmptyBot>debughost.cmd
.gitignore already exists. Skipped!
host.json already exists. Skipped!
appsettings.json already exists. Skipped!
Directory already a git repository.
Listening on http://localhost:3978
Hit CTRL-C to exit...
```

此時，Bot 正在本機執行。在主控台視窗中，複製 debughost 正在接聽的端點（此範例中是 <http://localhost:3978>）。然後，啟動 Bot Framework 模擬器，並將端點貼入模擬器的網址列。針對此範例，您也必須將 `/api/messages` 附加至端點。由於您不需要本機偵錯的安全性，您可以將 **Microsoft 應用程式識別碼** 和 **Microsoft 應用程式密鑰** 欄位留白。按一下 [連線]，使用指定端點建立到您 Bot 的連線。



將模擬器連接到 Bot 之後，在位於模擬器視窗底部的文字方塊（亦即左下角會出現輸入您的訊息...）中輸入一些文字，以傳送訊息至 Bot。透過使用模擬器視窗右側的 [記錄] 和 [偵測器] 面板，您可以檢視要求和回應，因訊息會在模擬器和 Bot 之間交換。



此外，您也可以在主控台視窗中檢視記錄詳細資料。

A screenshot of a Windows Command Prompt window titled "Command Prompt - debughost.cmd". The window shows the following output:

```
C:\repos\piglatinbot-dotnetfunction\EmptyBot>dotnet restore
log : Restoring packages for C:\repos\piglatinbot-dotnetfunction\EmptyBot\project.json...
log : Lock file has not changed. Skipping lock file write. Path: C:\repos\piglatinbot-dotnetfunction\EmptyBot\project.lock.json
log : C:\repos\piglatinbot-dotnetfunction\EmptyBot\project.json
log : Restore completed in 194ms.

C:\repos\piglatinbot-dotnetfunction\EmptyBot>debughost.cmd
.gitignore already exists. Skipped!
host.json already exists. Skipped!
appsettings.json already exists. Skipped!
Directory already a git repository.
Listening on http://localhost:3978
Hit CTRL-C to exit.
Found the following functions:
Host.Functions.EmptyBot
Job host started
Executing: 'Functions.EmptyBot' - Reason: 'This function was programmatically called via the host APIs.'
Webhook was triggered!
Webhook was triggered!
ActivityType conversationUpdate, Activity.Text
ActivityType conversationUpdate, Activity.Text
Executed: 'Functions.EmptyBot' (Succeeded)
Executing: 'Functions.EmptyBot' - Reason: 'This function was programmatically called via the host APIs.'
Webhook was triggered!
Webhook was triggered!
ActivityType message, Activity.Text Hello
ActivityType message, Activity.Text Hello
Executed: 'Functions.EmptyBot' (Succeeded)
```

其他資源

- 請參閱[針對一般問題疑難排解](#)以及該區段中的其他疑難排解文章。
- 了解如何[使用模擬器進行偵錯](#)。

後續步驟

使用文字記錄檔進行 Bot 偵錯。

測試和偵錯指導方針

2019/4/26 • [Edit Online](#)

適用於:  SDK v4  SDK v3

Bot 是將許多不同組件整合在一起運作的複雜應用程式。就像任何其他複雜的應用程式，這會導致一些有趣的錯誤 (bug)，或是讓 Bot 產生出乎意料的行為。

針對 Bot 進行測試以及後續的偵錯過程，有時候是很困難的工作。每個開發人員都有自己慣用的方法來完成該工作。我們提供以下的指導方針給您做為建議，可適用於大部分的 Bot。

測試您的 Bot

下面的指導方針以三種不同的等級呈現。每個等級都會增加測試的複雜度和功能，因此建議在您對某個等級的結果感到滿意之後，再移至下一個等級。這樣做可讓您在增加複雜度之前，先隔離出最低等級的問題並予以修正。

只要適用，測試的最佳做法就可涵蓋各種不同的角度。包括安全性、整合、格式不正確的 URL、驗證入侵、HTTP 狀態碼、JSON 承載、Null 值等等。如果您的 Bot 處理對使用者隱私權有影響的任何資訊，這點特別重要。

等級 1: 使用模擬元素

測試的第一個等級是確定應用程式 (在此案例中為 Bot) 的每一個小部分都完全如預期般運作。您可以針對目前未測試的項目使用模擬元素來達成此目的。做為參考，此等級通常可視為單元和整合測試。

使用模擬元素來測試個別區段

盡可能地模擬多個元素，可以更有效地隔離您要測試的部分。模擬元素的對象包括儲存體、配接器、中介軟體、活動管線、通道，以及任何其他非直屬於 Bot 的組件。這也可能會暫時移除某些部分 (例如與您要測試之 Bot 無關的中介軟體)，以便隔離每個部分。不過，如果您要測試中介軟體，建議可以改為模擬 Bot。

模擬元素可以採取數種形式，從以不同的已知物件取代元素，到實作最基本的 Hello World 功能。另外可採取的形式如，假設某元素不是必要元素，則可單純移除元素，或是強制它不執行任何動作。

此等級應演練您 Bot 中的個別方法和函式。您可以透過內建的單元測試 (這也是建議作法) 來測試個別方法，或是透過自己的測試應用程式、測試套件，或使用 IDE 來手動測試個別方法。

使用模擬元素來測試較大的功能

一旦您滿意每個方法的行為之後，請使用這些模擬元素測試 Bot 中更完整的功能。這裡示範幾個層級如何一起運作以便與使用者交談。

有一些工具可供您協助您達成目標。例如，[Azure Bot Framework Emulator](#) 提供與 Bot 通訊的模擬通道。使用模擬器會進入比單純的單元測試及整合測試更複雜的情況，也因此會擴及下一等級的測試。

等級 2: 使用 Direct Line 用戶端

確認您的 Bot 如預期般運作之後，下一步就是將它連線至通道。若要這樣做，您可以將 Bot 部署到預備伺服器，並建立自己的直接線路用戶端以供 Bot 連線。

建立自己的用戶端可讓您定義通道內部的運作方式，並且可特別針對 Bot 回應特定活動交換的方式進行測試。一旦 Bot 連線到您的用戶端後，請執行您的測試以設定 Bot 狀態並驗證功能。如果您的 Bot 會利用如語音之類的功能，使用這些通道可提供驗證該功能的方式。

透過 Azure 入口網站在此處使用模擬器和網路聊天，可提供 Bot 在使用不同通道互動時的執行方式見解。

等級 3: 通道測試

一旦您對 Bot 的獨立效能安心之後，請務必了解在各種將提供 Bot 的通道上，Bot 的運作情況。

達成此目的的方法有許多種，例如分別使用不同的通道與瀏覽器，或使用協力廠商工具（例如使用 Selenium）透過通道互動並擷取 Bot 的回應。

其他測試

不同類型的測試都可透過結合上述的等級，或是不同的角度（例如壓力測試、效能測試或分析 Bot 活動）來完成。Visual Studio 提供達成此目的的本機方法以及 [工具套件](#)，可用來測試您的應用程式，而 [Azure 入口網站](#) 則提供 Bot 執行狀況的見解。

Debugging

針對 Bot 進行偵錯的方式，與針對其他多執行緒應用程式進行偵錯的方式相似，都要使用設定中斷點或是如即時運算視窗的功能。

Bot 遵循事件驅動程式設計架構，如果您還不熟悉，就很難進行說明。讓您的 Bot 處於無狀態、多執行緒以及處理非同步/等候呼叫的概念，都會造成無法預期的錯誤（Bug）。針對 Bot 偵錯的運作方式與針對其他多執行緒應用程式的方式相似，我們將會提供一些實用的建議、工具和資源來協助您。

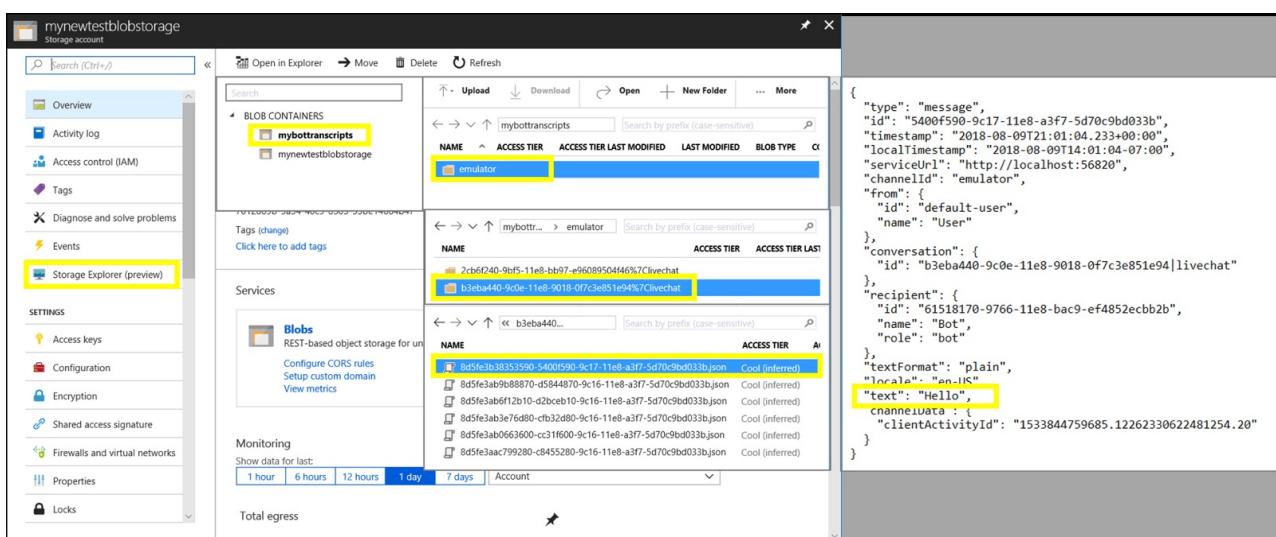
使用模擬器了解 Bot 活動

您的 Bot 除了處理一般的_訊息_活動之外，也會處理不同類型的活動。使用 [模擬器](#) 會顯示那些活動的內容、發生的時機，以及它們所包含的內容。了解那些活動有助於您更有效率地撰寫 Bot 的程式碼，並且可讓您確認 Bot 如預期地傳送和接收活動。

透過文字記錄儲存及擷取使用者互動

Azure Blob 文字記錄儲存體提供特製化資源，您可以在其中 [儲存及擷取文字記錄](#)，內含使用者與您的 bot 之間的互動。

此外，一旦儲存使用者輸入互動，您即可使用 Azure 的「儲存體總管」，手動檢視 Blob 文字記錄存放區內所儲存的文字記錄中包含的資料。下列範例會從 "mynewtestblobstorage" 的設定開啟「儲存體總管」。若要開啟儲存的使用者輸入選取：Blob 容器 > ChannelId > TranscriptId > ConversationId



The screenshot shows the Azure Storage Explorer interface for the storage account "mynewtestblobstorage". On the left, the navigation pane includes "Overview", "Activity log", "Access control (IAM)", "Tags", "Diagnose and solve problems", "Events", and "Storage Explorer (preview)" which is selected. In the main pane, under "BLOB CONTAINERS", the "mybottranscripts" container is selected. Inside this container, there is a single blob file named "emulator". The file's properties show it was last modified on 2018-08-09T21:01:04.233+00:00. The file content is displayed as a JSON object:

```
{  
  "type": "message",  
  "id": "540f5f90-9c17-11e8-a3f7-5d70c9bd033b",  
  "timestamp": "2018-08-09T21:01:04.233+00:00",  
  "localTimestamp": "2018-08-09T14:01:04-07:00",  
  "serviceUrl": "http://localhost:56820",  
  "channelId": "emulator",  
  "from": {  
    "id": "default-user",  
    "name": "User"  
  },  
  "conversation": {  
    "id": "b3eba440-9c0e-11e8-9018-0f7c3e851e94|livechat"  
  },  
  "recipient": {  
    "id": "61518170-9766-11e8-bac9-ef4852ecbb2b",  
    "name": "Bot",  
    "role": "bot"  
  },  
  "textFormat": "plain",  
  "locale": "en-US",  
  "text": "Hello",  
  "channelData": {  
    "clientActivityId": "1533844759685.12262330622481254.20"  
  }  
}
```

這會開啟以 JSON 格式儲存的使用者對話輸入。使用者輸入會與索引鍵 "text:" 一起保存。

中介軟體的運作方式

第一次嘗試使用 [中介軟體](#) 時，可能會覺得不是非常直覺，特別是關於執行的接續或最少運算作法。當執行傳遞給 Bot 邏輯時，中介軟體可透過呼叫 `next()` 委派命令，在回合的前緣或後緣執行。

如果您使用以多個部分組成的中介軟體，則委派可能會將執行傳遞到中介軟體的不同部分（若您的管線導向方式是如此的話）。如需詳細資訊，請參閱 [Bot 中介軟體管線](#) 以澄清概念。

如果沒有呼叫 `next()` 委派，則稱之為 [最少運算路由](#)。當中介軟體滿足目前的活動，並判斷不需要傳遞執行時，就會發生此情況。

了解中介軟體最少運算發生的時機和原因，有助於指出在您管線中哪個中介軟體的部分應先出現。此外，針對由 SDK 或其他開發人員提供的內建中介軟體，請務必了解它們預期會出現的內容。在深入了解內建的中介軟體前，有些人發現先嘗試建立自己的中介軟體進行一些實驗非常有幫助。

了解狀態

持續追蹤狀態是 Bot 中很重要的一環，特別是針對複雜的工作更是如此。一般而言，最佳做法是盡可能地快速處理活動，然後讓處理程序完成，如此狀態才會進入保存狀態。活動可能近乎同時傳送給您的 Bot，而由於非同步架構的關係，可能造成非常令人困惑的錯誤 (Bug)。

因此，最重要的是，務必確定保存的狀態符合您預期的狀態。根據您保存的狀態的位置，適用於 [Cosmos DB](#) 和 [Azure 表格儲存體](#) 的儲存體模擬器有助於您在使用生產環境儲存體之前先驗證該狀態。

如何使用活動處理常式

活動處理常式可能導入另一層次的複雜度，特別是因為每個活動都在獨立的執行緒 (或是背景工作角色，視您的程式設計語言而定) 上執行。根據處理常式執行的內容，這可能會造成目前狀態與預期狀態不符的問題。

內建狀態在回合結束時會被覆寫，不過由該回合產生的任何活動都會獨立執行，與回合管線無關。通常，這對我們沒有影響，但如果活動處理常式變更我們需要寫入以包含該變更的狀態，則狀況又會有所不同。在此情況下，回合管線可能會等候活動完成處理，然後才完成作業，以確保它會記錄該回合的正確狀態。

send activity 方法，以及其處理常式，會造成一個特有的問題。只需在 *on send activities* 處理常式內呼叫 *send activity*，就會導致無限分支的執行緒。針對該問題，有一些方法可以作為因應措施，例如將額外訊息附加到輸出資訊，或寫出到其他位置 (例如主控台)，或是寫出檔案來避免造成 Bot 當機。

其他資源

- [Visual Studio 偵錯](#)
- 針對 Bot Framework 進行[偵錯、追蹤與分析](#) (機器翻譯)
- 針對您不想在生產環境程式碼中納入的方法使用 [ConditionalAttribute](#) (英文)
- 使用如 [Fiddler](#) (英文) 的工具查看網路流量
- [Bot 工具存放庫](#) (英文)
- 有助於進行測試的 Framework，例如 [Moq](#) (英文)
- 針對一般問題進行[疑難排解](#) 以及該區段中的其他疑難排解文章

使用模擬器進行偵錯

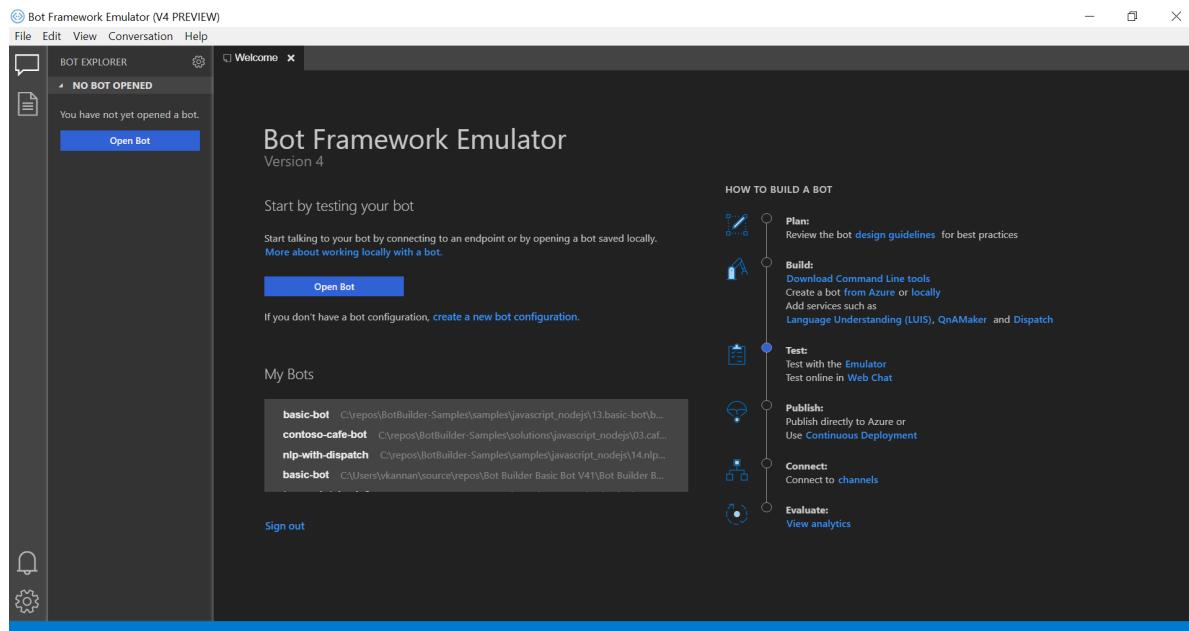
2019/5/10 • [Edit Online](#)

Bot Framework 模擬器是一項桌面應用程式，可讓 Bot 開發人員在本機或從遠端對其 Bot 進行測試和偵錯。使用此模擬器時，您能與您的 Bot 聊天，並檢查您的 Bot 所傳送及接收的訊息。模擬器會顯示在 Web 聊天 UI 中出現的相同訊息，並在您與 Bot 交換訊息時記錄 JSON 要求和回應。將您的 Bot 部署至雲端之前，請先在本機使用模擬器加以測試。即使您尚未使用 Azure Bot Service 建好 Bot 或將其設定為在任何通道上執行，您仍可使用模擬器來測試 Bot。

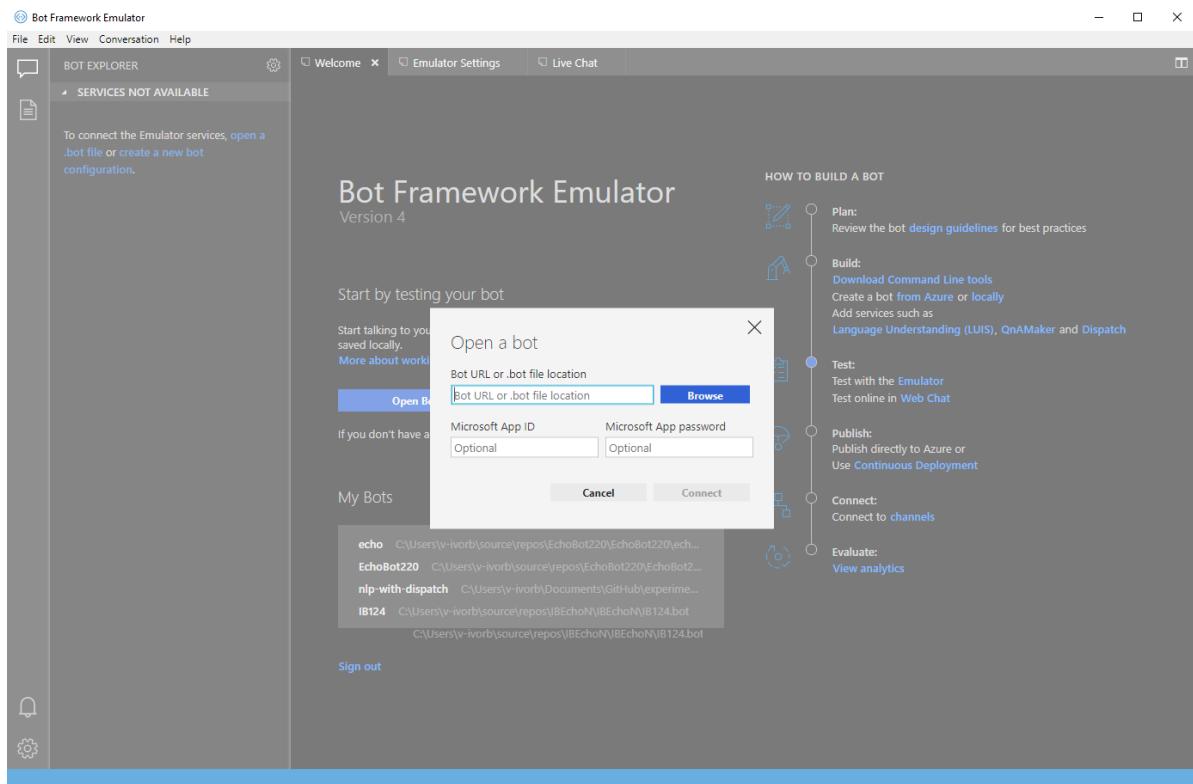
必要條件

- 安裝[模擬器](#)

連線至執行於本機主機的 Bot



若要連線到本機上執行的 Bot，請按一下 [開啟 Bot]，或選取您預先設定的組態檔 (.bot 檔案)。您不需要用組態檔來連線到 Bot，但如果您的 Bot 有組態檔，模擬器仍會與其搭配使用。如果您的 Bot 搭配 Microsoft 帳戶 (MSA) 認證執行，請也輸入這些認證。



使用偵測器檢視詳細訊息活動

傳送訊息給您的 Bot，而 Bot 應該會有所回應。您可以按一下對話視窗內的訊息泡泡，並使用視窗右側的偵測器功能檢查原始 JSON 活動。訊息泡泡經選取後會變成黃色，且活動 JSON 物件將會顯示在聊天視窗左側。JSON 資訊包含主要中繼資料，包括 channelID、活動類型、交談識別碼、文字訊息、端點 URL 等。您可以檢查使用者所傳送的活動，以及 Bot 所回應的活動。

The screenshot shows a detailed view of a conversation in the Bot Framework Emulator. On the left, a transcript pane shows a message from the user 'Please enter your mode of transport.' and three buttons: 'Car', 'Bus', and 'Bicycle'. A response from the bot, 'hi', is shown with a timestamp '5 minutes ago'. On the right, there are two panes: 'INSPECTOR - JSON' which displays the selected JSON activity for the 'hi' message, and 'LOG' which shows the command-line logs for the session.

```

{
  "attachments": [],
  "channelId": "emulator",
  "conversation": {
    "id": "9680fab0-50a2-11e9-90e5-b95b5f203e63|livechat"
  },
  "entities": [],
  "from": {
    "id": "c219a680-4f36-11e9-90e5-b95b5f203e63",
    "name": "Bot",
    "role": "bot"
  },
  "id": "992036a0-50a2-11e9-862b-37a65f22ae7e",
  "inputHint": "acceptingInput",
  "localTimestamp": "2019-03-27T08:11:29-07:00",
  "locale": "",
  "recipient": {
    "id": "7c32a522-2552-4728-92cb-3a2fb7951e0b",
    "name": "User"
  }
}

[08:11:25] POST 201 directline.startConversation
[08:11:25] Emulator listening on http://localhost:61023
[08:11:25] ngrok not configured (only needed when connecting to
remotely hosted bots)
[08:11:25] Connecting to bots hosted remotely
[08:11:25] Edit ngrok settings
[08:11:29] ->message hi
  
```

儲存及載入與 Bot 交談的文字記錄

模擬器中的活動可以儲存為文字記錄。從開啟的即時聊天視窗中，選取 [另存文字記錄]，以儲存文字記錄檔案。您可以隨時使用 [重新開始] 按鈕來清除對話，並重新連線至 Bot。

The screenshot shows a web-based messaging interface. At the top, there are buttons for 'Restart conversation' and 'Save transcript'. Below that is a URL bar with 'http://localhost:3978/api/messages'. The main area displays a conversation transcript:

```

hi
5 minutes ago
Please enter your mode of transport.
5 minutes ago
Car Bus Bicycle

```

Below the transcript is a text input field with a placeholder 'Type your message' and a send button.

To the right of the transcript is an 'INSPECTOR - JSON' panel showing the following JSON data:

```

{
  "attachments": [],
  "channelId": "emulator",
  "conversation": {
    "id": "9680fab0-50a2-11e9-90e5-b95b5f203e63|livechat"
  },
  "entities": [],
  "from": {
    "id": "c219a680-4f36-11e9-90e5-b95b5f203e63",
    "name": "Bot",
    "role": "bot"
  },
  "id": "992036a0-50a2-11e9-862b-37a65f22ae7e",
  "inputHint": "acceptingInput",
  "localTimestamp": "2019-03-27T08:11:29-07:00",
  "locale": "",
  "recipient": {
    "id": "7c32a522-2552-4728-92cb-3a2fb7951e0b",
    "name": "User"
  }
}

```

Below the JSON is a 'LOG' section with the following entries:

- [08:11:25] POST [201](#) directline.startConversation
- [08:11:25] Emulator listening on http://localhost:61023
- [08:11:25] ngrok not configured (only needed when connecting to remotely hosted bots)
- [08:11:25] Connecting to bots hosted remotely
- [08:11:25] Edit ngrok settings
- [08:11:29] ->[message](#) hi

若要載入文字記錄，只需選取 [檔案] > [開啟文字記錄檔案]，再選取文字記錄即可。新的 [文字記錄] 視窗隨即開啟，並將訊息活動轉譯為輸出視窗。

The screenshot shows the 'Bot Framework Emulator' interface. On the left is a sidebar with the following menu items:

- New Bot Configuration...
- Open Bot Configuration...
- Open Recent...
- Open Transcript...** (highlighted with a red box)
- Close Tab
- Sign in with Azure
- Themes
- Exit

The main area displays the 'Bot Framework Emulator' logo and version information. It includes a 'Start by testing your bot' section with a 'Open Bot' button, and a note about creating a new bot configuration if none exists.

新增服務

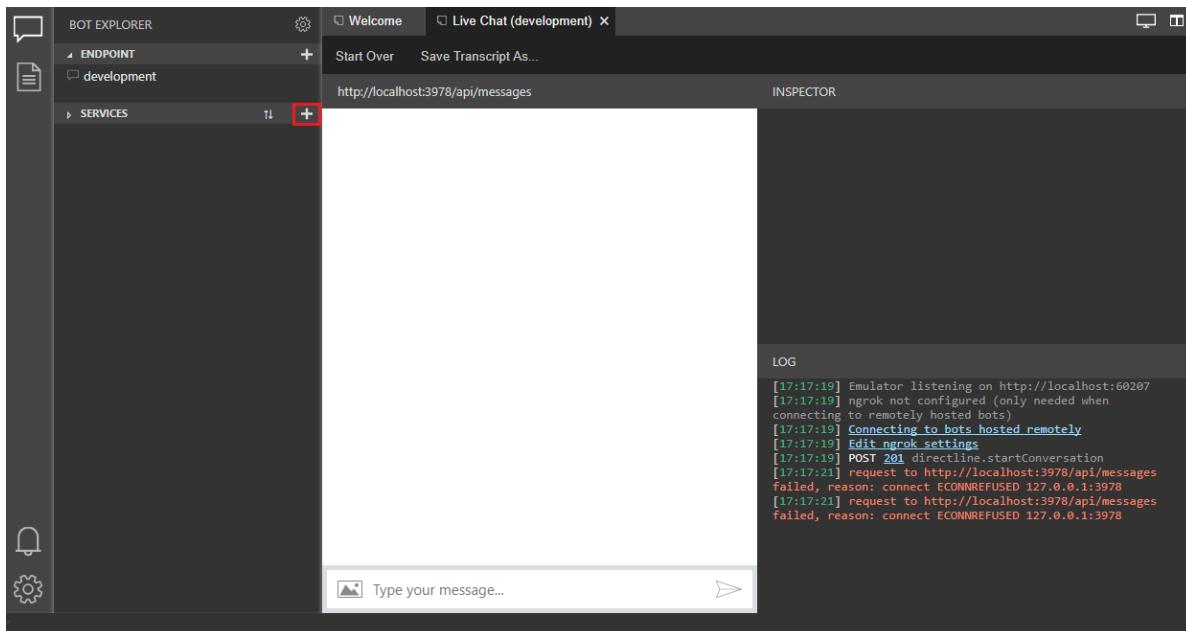
您可以直接從模擬器輕鬆地將 LUIS 應用程式、QnA 知識庫或分派模型新增到 Bot。載入 Bot 後，請選取模擬器視窗最左側的 [服務] 按鈕。您會在 [服務] 功能表下方看到用來新增 LUIS、QnA Maker 和分派的選項。

若要新增服務應用程式，只需按一下 [+] 按鈕，然後選取您想要新增的服務。系統會提示您登入 Azure 入口網站，以將服務新增至 Bot 檔案，並將服務連線到您的 Bot 應用程式。

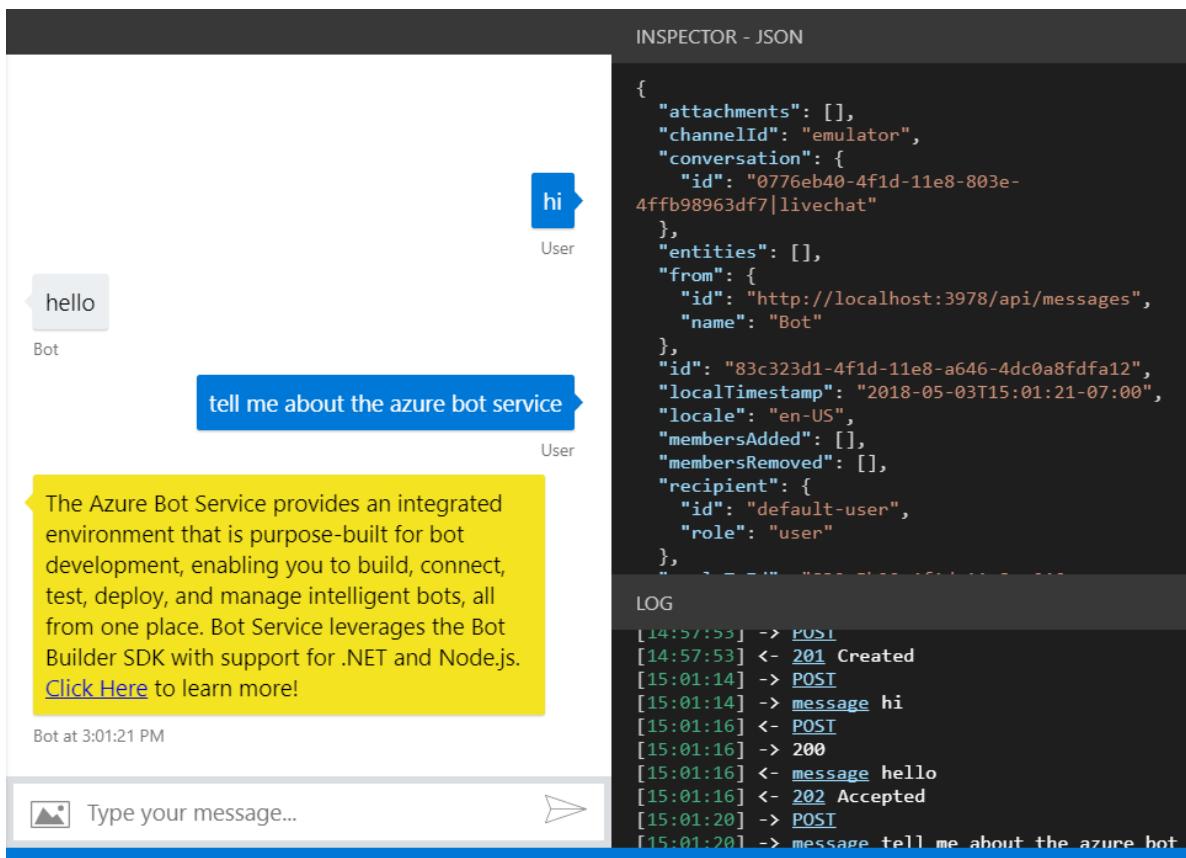
IMPORTANT

只有在您使用 `.bot` 組態檔時，才能新增服務。服務需要個別加入。如需詳細資訊，請參閱[管理 Bot 資源](#)，或針對您要新增的服務，參閱個別的操作說明文章。

如果您不是使用 `.bot` 檔案，左側窗格將不會列出您的服務（即使您的 Bot 有使用服務），而且會顯示服務無法使用。



在任一服務連線時，您都可以返回即時聊天視窗，並確認您的服務已連線並運作中。



檢查服務

透過新的 v4 模擬器，您也可以檢查 LUIS 和 QnA 的 JSON 回應。使用已連接語言服務的 Bot 時，您可以在右下方選取 [記錄] 視窗中的 [追蹤]。這項新工具也提供直接從模擬器更新語言服務的功能。

LOG

```
[11:38:45] POST 200 conversations.replyToActivity
[11:38:45] <- message Hello and welcome to the Luis Sample bot.
[11:39:01] -> message run tests
[11:39:02] POST 200 conversations.replyToActivity
[11:39:02] <- trace Luis Trace
[11:39:02] POST 200 conversations.replyToActivity
[11:39:02] <- message The **top intent** was: **'RunTests'**, with score...
[11:39:02] POST 200 conversations.replyToActivity
[11:39:02] <- message Detail of intents scorings:
[11:39:02] POST 200 conversations.replyToActivity
[11:39:02] <- message * 'RunTests', score 0.9753478 * 'Build', score 0....
[11:39:02] POST 200 directline.postActivity
```

在已連接 LUIS 服務的情況下，您會發現追蹤連結指定了 Luis 追蹤。加以選取後，您會看到 LUIS 服務的原始回應，其中包含意圖、實體及其指定的分數。您也可以選擇重新指派使用者語句的意圖。

LOG

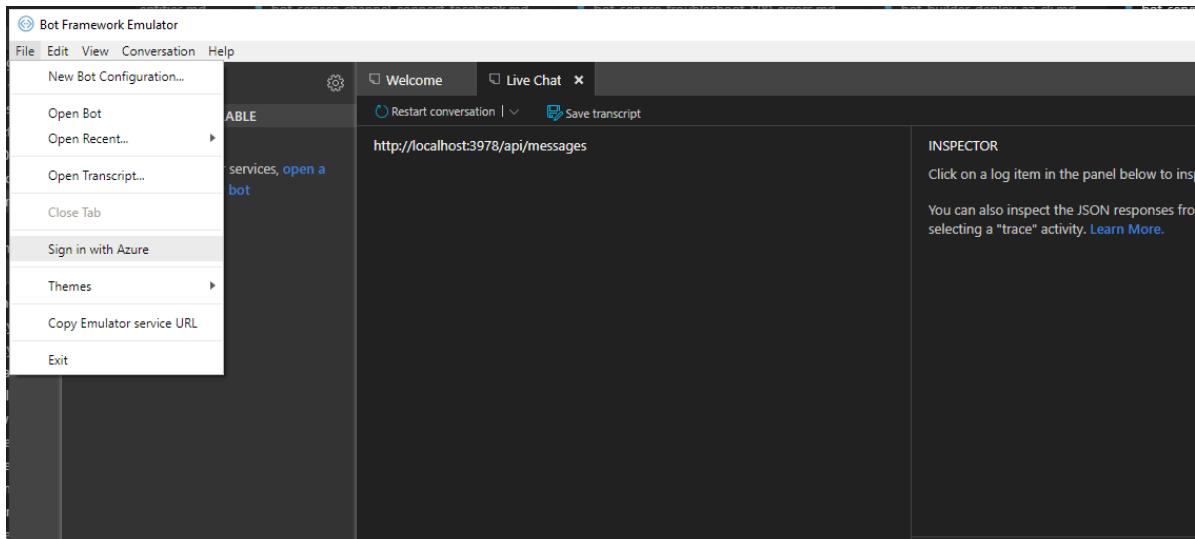
```
[11:22:29] Emulator listening on http://localhost:49863
[11:22:29] ngrok listening on https://899d5e3b.ngrok.io
[11:22:29] ngrok traffic inspector: http://127.0.0.1:4040
[11:22:29] Will bypass ngrok for local addresses
[11:22:29] POST 201 directline.startConversation
[11:22:29] POST 200 conversations.replyToActivity
[11:22:29] <- message Hello and welcome to the QnA Maker Sample bot.
[11:22:36] -> message hi
[11:22:37] POST 200 conversations.replyToActivity
[11:22:37] <- trace QnAMaker Trace
[11:22:37] POST 200 conversations.replyToActivity
[11:22:37] <- message Hello!
[11:22:37] POST 200 directline.postActivity
```

在已連接 QnA 服務的情況下，記錄將會顯示 QnA 追蹤，若加以選取，您將可預覽與該活動相關聯的問答組和信賴分數。在此處，您可以新增答案的替代問題用語。

登入 Azure

您可以使用模擬器來登入您的 Azure 帳戶。這特別適合用來讓您新增及管理 Bot 所依賴的服務。若要深入了解可使用模擬器來管理的服務，請參閱[上述內容](#)。

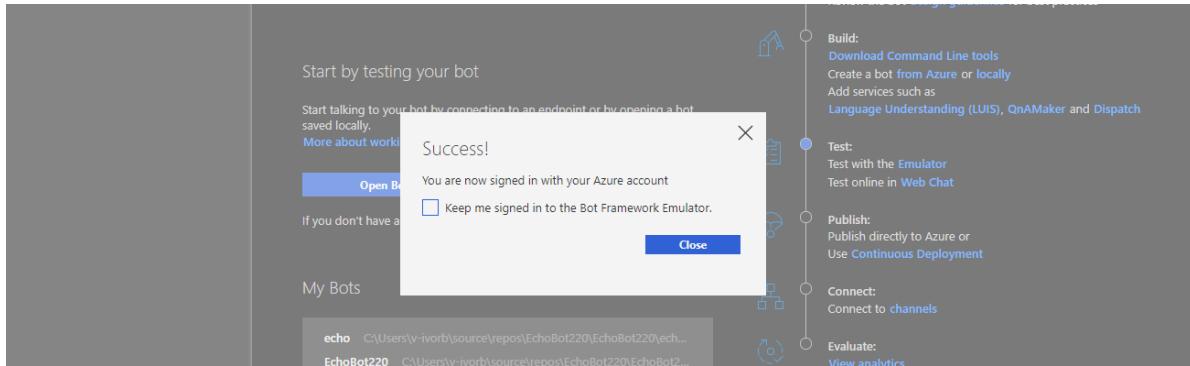
登入



登入

- 您可以按一下 [檔案]-> [使用 Azure 登入]
- 在歡迎畫面上按一下 [使用 Azure 帳戶登入]，您可以選擇性地讓模擬器保留您的登入狀態，以在模擬器

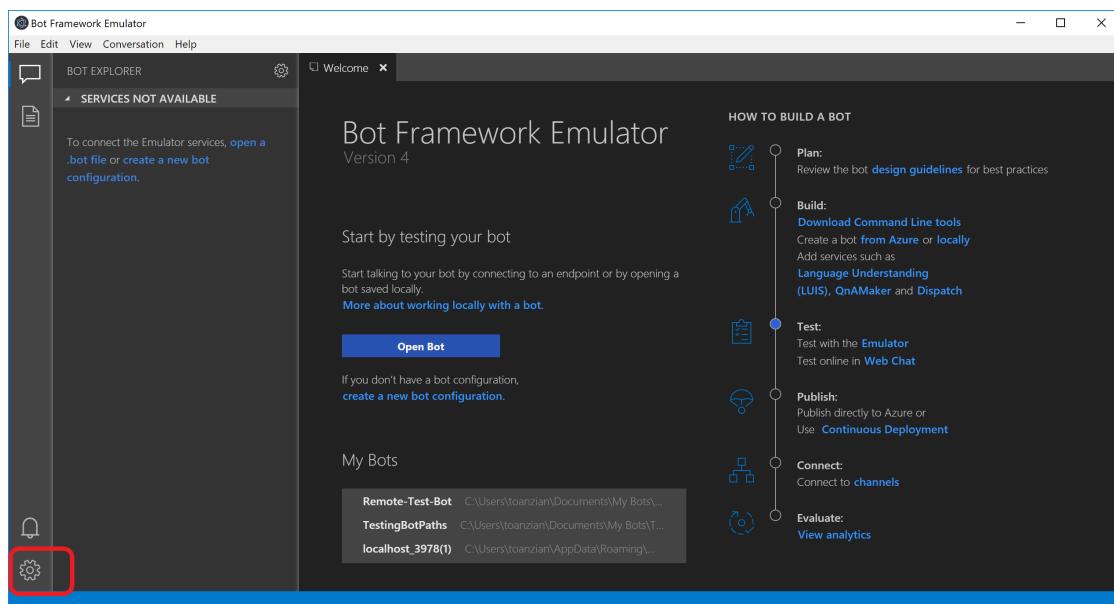
應用程式重新啟動之間保持登入。



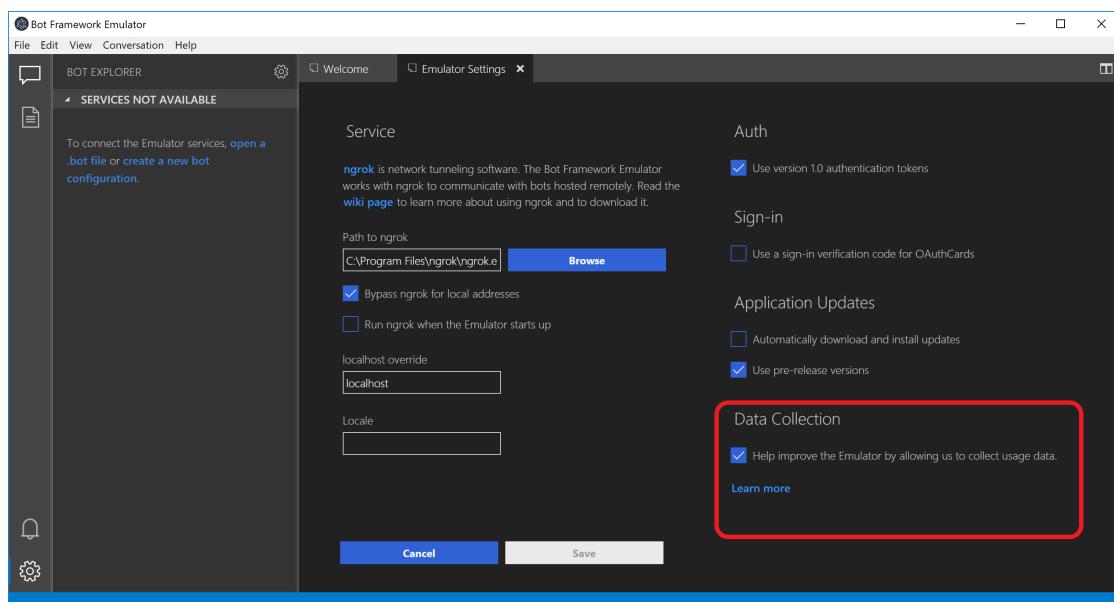
停用資料收集

如果您決定不再允許模擬器收集使用資料，您可以依照下列步驟來輕鬆地停用資料收集：

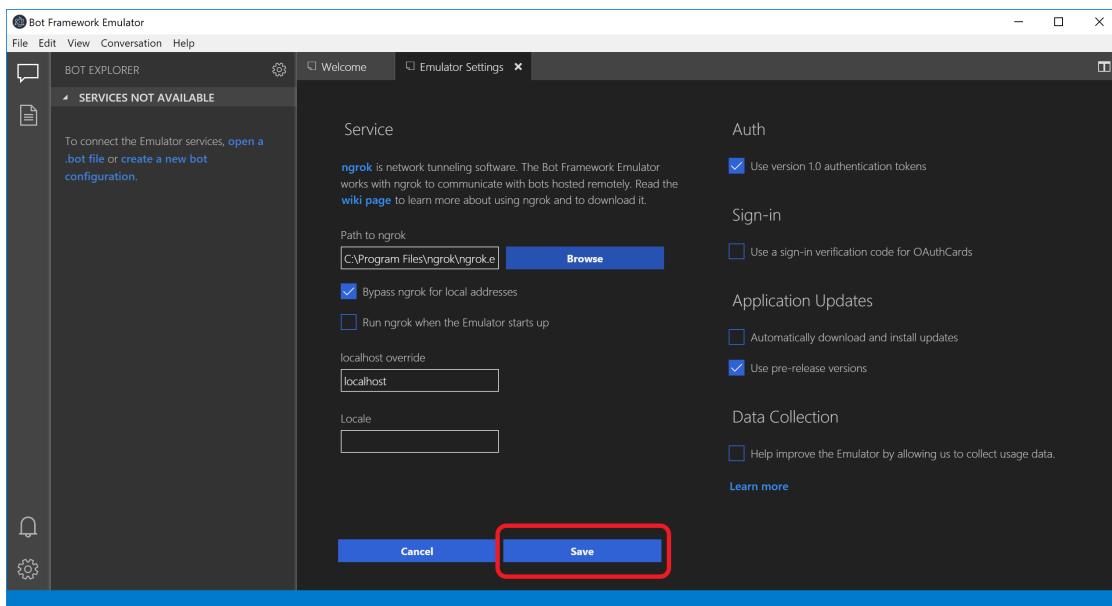
1. 按一下左側瀏覽列中的 [設定] 按鈕 (齒輪圖示)，瀏覽至模擬器的 [設定] 頁面。



2. 在 [資料收集] 區段底下，取消選取標示 [允許我們收集使用資料，以協助改善模擬器] 的核取方塊。



3. 按一下 [儲存] 按鈕。



如果您改變心意，一律可以藉由重新選取核取方塊來將其啟用。

其他資源

Bot Framework 模擬器是開放原始碼。您可以[參與開發](#)，並[提交 Bug 和建議](#)。

如需進行疑難排解，請參閱[針對一般問題疑難排解](#)以及該區段中的其他疑難排解文章。

後續步驟

將對話儲存為文字記錄檔，可讓您快速地對一組互動撰寫草稿及加以重新執行，以用於偵錯。

[使用文字記錄檔進行 Bot 偵錯](#)

使用文字記錄檔進行 Bot 偵錯

2019/5/14 • [Edit Online](#)

適用於： SDK v4 SDK v3

成功進行 Bot 測試和偵錯的關鍵之一，就是能夠記錄並檢查在執行 Bot 時所發生的一些情況。本文討論 Bot 文字記錄檔的建立和使用方式，以提供一組詳細的使用者互動和 Bot 回應以供測試和偵錯。

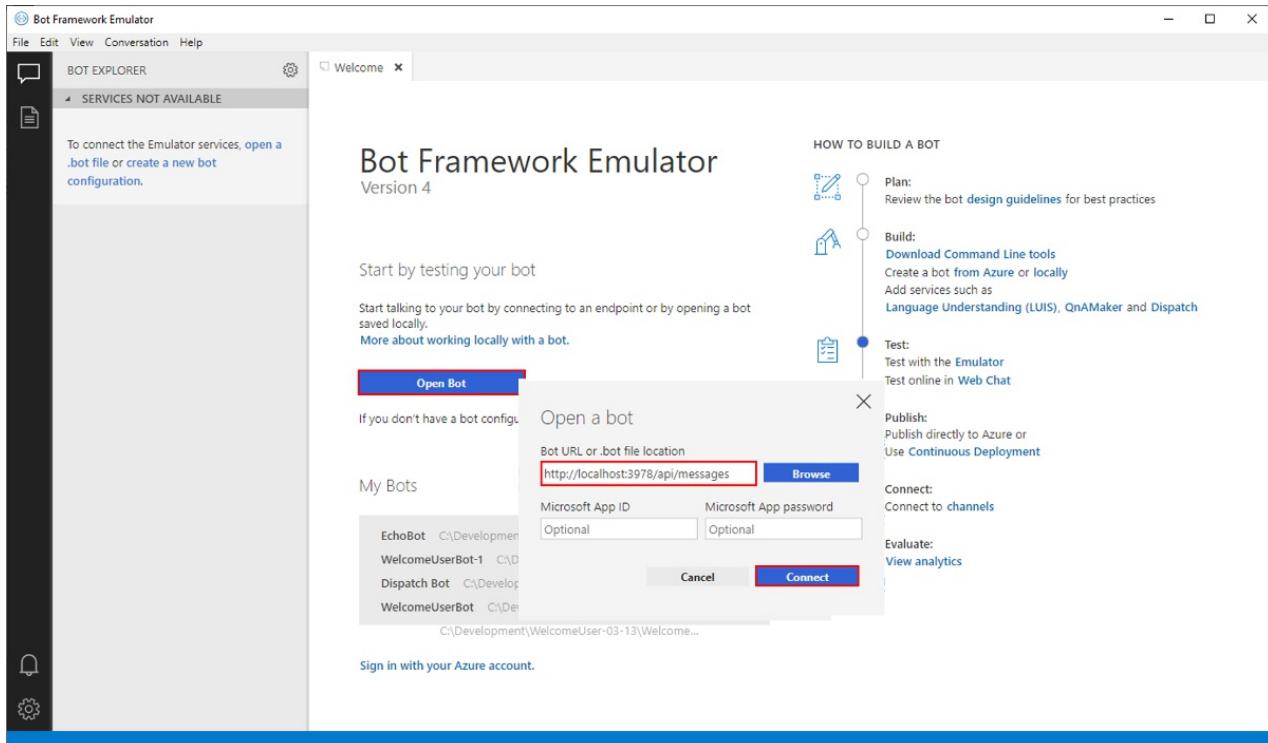
Bot 文字記錄檔

Bot 文字記錄檔是特製化的 JSON 檔案，可保留使用者與 Bot 之間的互動。文字記錄檔不僅保留訊息內容，也會保留互動詳細資料，例如使用者識別碼、通道識別碼、通道類型、通道功能、互動時間等。這些資訊接著可用來協助尋找及解決 Bot 測試或偵錯時的問題。

建立/儲存 Bot 文字記錄檔

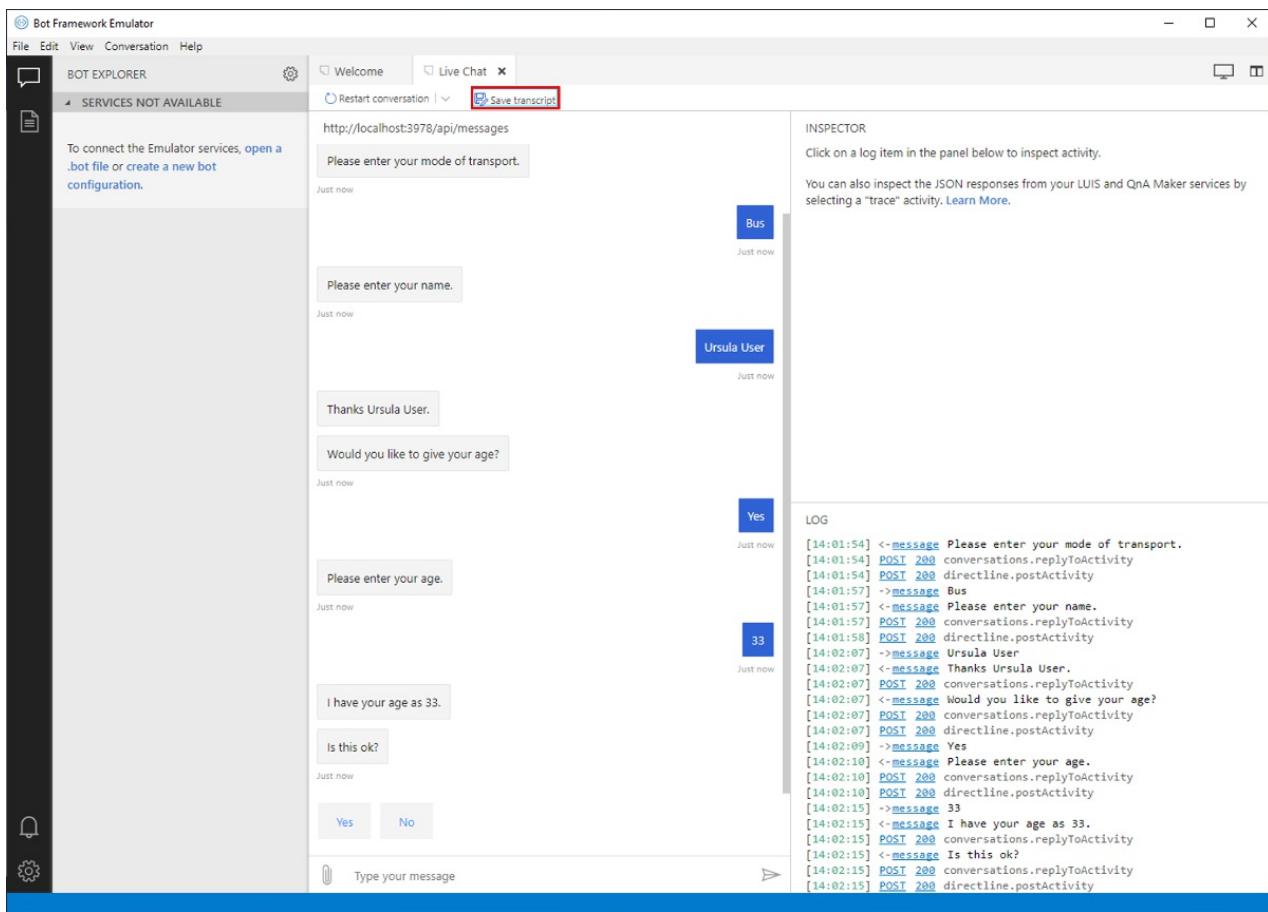
本文說明如何使用 Microsoft 的 [Bot Framework 模擬器](#)建立 Bot 文字記錄檔。文字記錄檔也可透過程式設計方式建立，您可以在[這裡](#)進一步了解該方法。在這篇文章中，我們將使用[多回合提示 Bot](#)的 Bot Framework 範例程式碼，該程式碼會要求使用者的運輸方式、名稱和年齡，但任何可使用 Microsoft 的 Bot Framework Emulator 存取的程式碼都可用來建立文字記錄檔。

若要開始此程序，請確保您想要測試的 Bot 程式碼是在您的開發環境內執行。啟動 Bot Framework Emulator，選取 [開啟 Bot] 按鈕，然後輸入瀏覽器中所示的 *localhost: port* 位址，後面接著 "/api/messages"，如下圖所示。現在按一下 [連線] 按鈕，以將模擬器連線至 Bot。

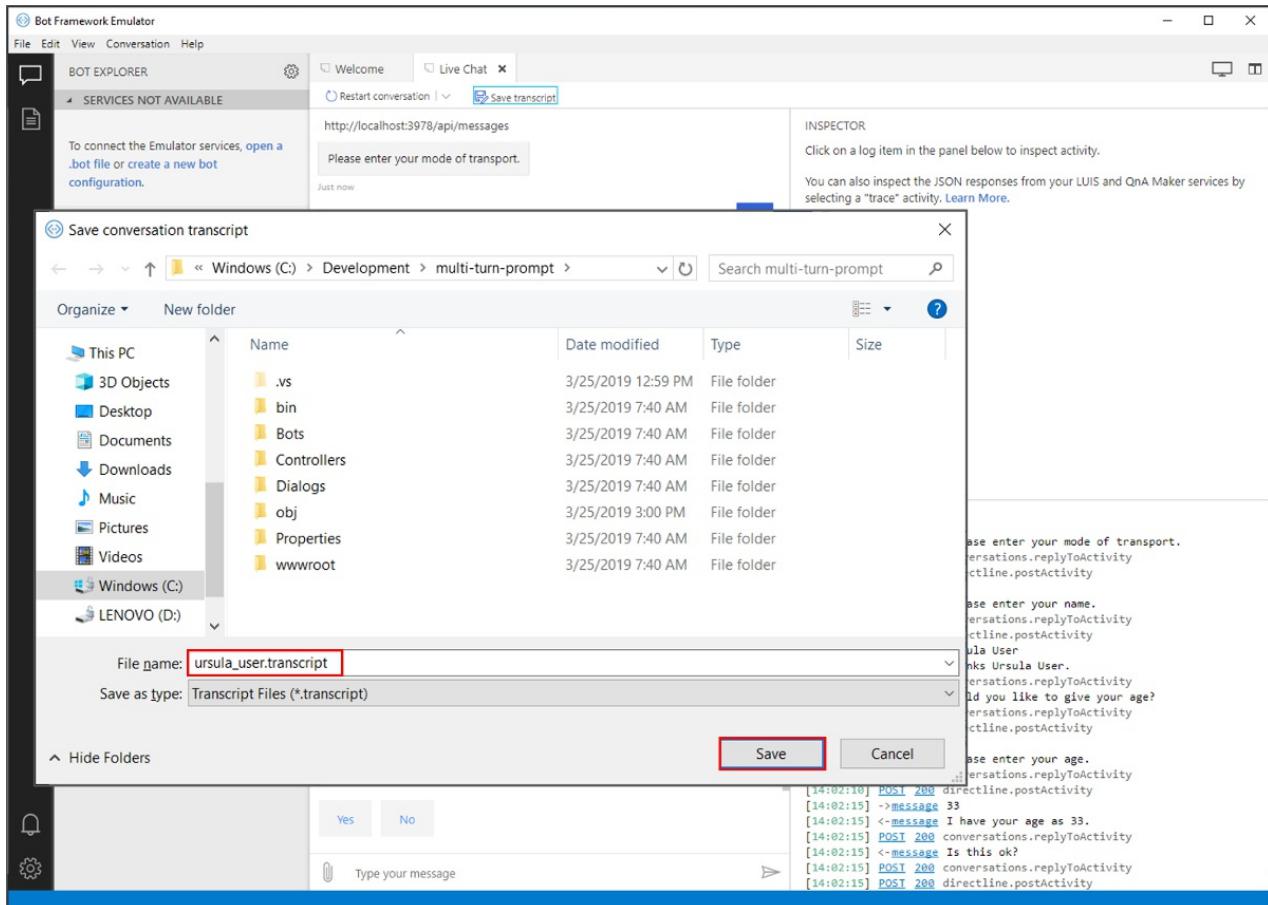


將模擬器連線到執行中的程式碼之後，請將模擬的使用者互動傳送給 Bot，以便測試程式碼。此範例中，我們已傳入使用者的運輸方式、名稱和年齡。在您輸入所有想要保留的使用者互動之後，請使用 Bot Framework 模擬器來建立及儲存含有此對話的文字記錄檔。

在 [即時聊天] 索引標籤 (如下所示) 中，選取 [儲存文字記錄] 按鈕。



選擇文字記錄檔的位置和名稱，然後選取 [儲存] 按鈕。

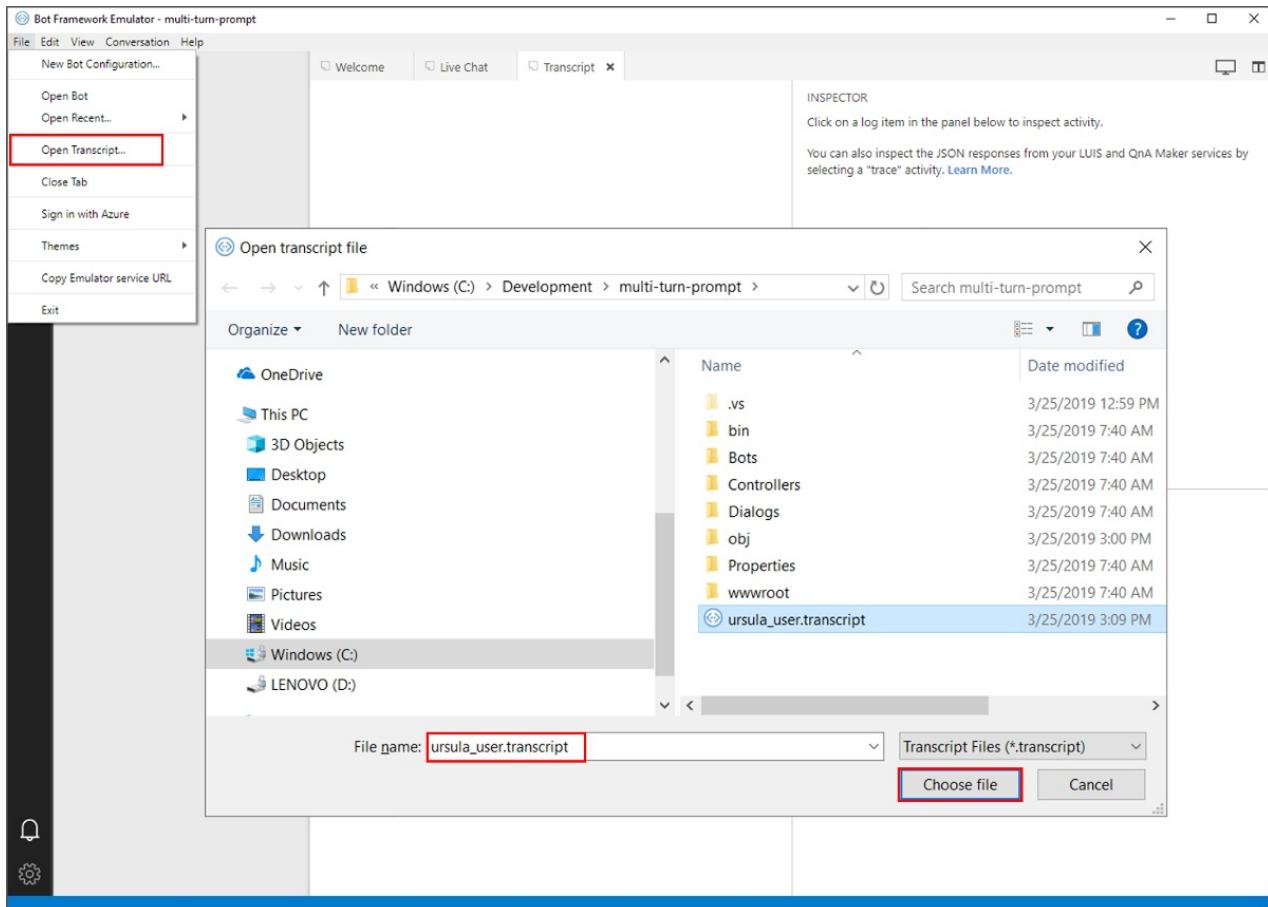


您為了使用模擬器測試程式碼而輸入的所有使用者互動和 Bot 回應，現在已儲存到您可稍後重新載入的文字記錄檔中，以協助進行使用者與 Bot 之間的互動偵錯。

擷取 Bot 文字記錄檔

若要使用 Bot Framework Emulator 擷取 Bot 文字記錄檔，請依序選取模擬器左上角的 [檔案] 和 [開啟文字記錄...]，如下所示。接下來，選取您想要擷取的文字記錄檔。(從模擬器的 [資源] 區段中的 [文字記錄] 清單控制項，也可以存取文字記錄)

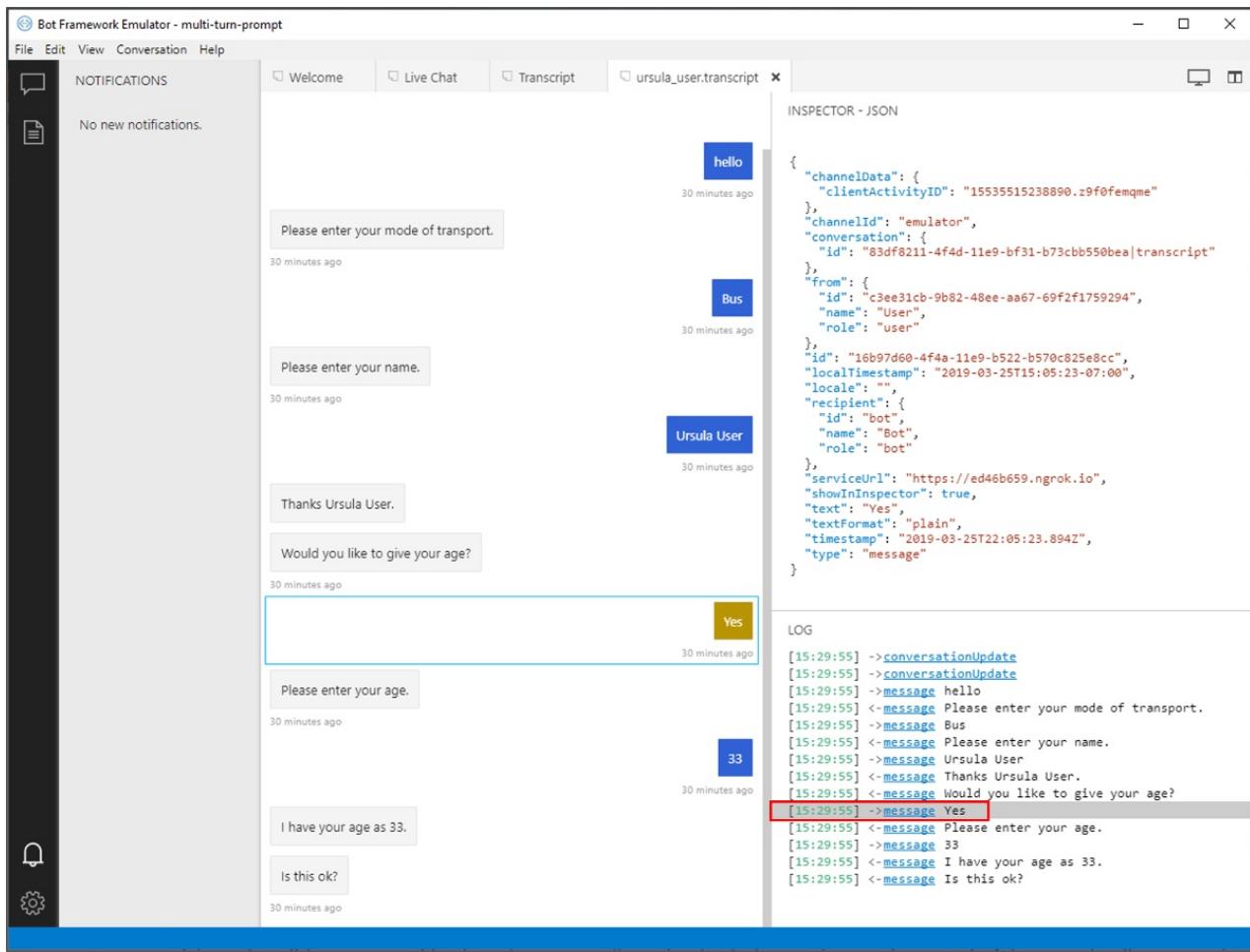
在此範例中，我們會擷取名為 "ursula_user.transcript" 的文字記錄檔。選取文字記錄檔，就會將整個保留的對話自動載入到標題為「文字記錄」的新索引標籤。



使用文字記錄檔進行偵錯

載入文字記錄檔後，您現在即可對在使用者與 Bot 之間擷取的互動進行偵錯。若要這樣做，只要按一下模擬器右下方區域中顯示的 LOG 區段中所記錄的任何事件或活動。在以下所示的範例中，我們選取了使用者傳送 "Hello" 訊息時的第一次反覆運算。當我們執行此動作時，文字記錄檔中關於此特定互動的所有資訊都會以 JSON 格式顯示在模擬器的 INSPECTOR 視窗中。由下而上查看其中一些值，我們會看到：

- 反覆運算類型為 message。
- 訊息的傳送時間。
- 已傳送內含 "Yes" 的純文字。
- 訊息已傳送給 Bot。
- 使用者識別碼和資訊。
- 通道識別碼、功能和資訊。



這麼詳細的資訊可讓您追蹤使用者輸入與 Bot 回應之間的逐步互動，而在 Bot 並未以您預期的方式回應或完全不回應使用者的情況下，這就很適合用於偵錯。擁有這些值以及導致互動失敗的步驟記錄，可讓您逐步執行程式碼、尋找 Bot 未如預期般回應的位置，以及解決這些問題。

使用文字記錄檔搭配 Bot Framework 模擬器，只是可用來協助您進行 Bot 程式碼與使用者互動測試和偵錯的許多工具之一。若要尋找更多 Bot 測試和偵錯的方法，請參閱下面所列的其他資源。

其他資訊

如需其他測試和偵錯資訊，請參閱：

- [Bot 測試和偵錯指導方針](#)
- [使用 Bot Framework 模擬器進行偵錯](#)
- [針對一般問題進行疑難排解](#) 以及該區段中的其他疑難排解文章。
- [Visual Studio 偵錯](#)

測試 Cortana 技能

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於 SDK v3 版本。您可以在[這裡](#)找到最新版本 (v4) 的 SDK 文件。

如果您已經建置使用 Bot Framework SDK Cortana 技能，您可以藉由從 Cortana 叫用來進行測試。下列指示會引導您完成試用 Cortana 技能所需的步驟。

註冊您的機器人

如果您在 Azure 中使用 Bot Service [建立您的 bot](#)，則您的 bot 已註冊，而您可以略過此步驟。

如果您已在其他地方部署您的機器人，或如果您想要在本機測試您的機器人，您必須[註冊](#)您的 bot 將它連接到 Cortana。在註冊過程中，您必須提供您機器人的傳訊端點。如果您選擇要在本機上測試您的機器人，您需要執行如[ngrok](#)的通道軟體，以取得您本機機器人的端點。

使用 ngrok 取得訊息端點

如果您在本機執行 bot，可以透過測試執行通道軟體，例如[ngrok](#) 來取得端點。若要使用 ngrok 從主控台視窗型別取得端點：

```
ngrok.exe http 3979 -host-header="localhost:3979"
```

這會設定並顯示 ngrok 轉送連結，將要求轉送到您於連接埠 3978 上執行的機器人。轉送連結的網址應該看起來像這樣：`https://0d6c4024.ngrok.io`。附加 `/api/messages` 至連結，以形成傳訊端點網址，格式如下：

`https://0d6c4024.ngrok.io/api/messages`。

在 bot 的[設定](#)刀鋒視窗設定區段中，輸入此端點網址。

啟用語音辨識預備

如果您的 Bot 使用 Language Understanding (LUIS) 應用程式，請確定您已為 LUIS 應用程式識別碼與您已註冊的 bot 服務建立關聯。這可協助您的 Bot 識別 LUIS 模型中定義的口語發音。請參閱[語音預備](#)以取得詳細資訊。

新增 Cortana 通道

若要新增 Cortana 做為通道，請依照[將 Bot 連線至 Cortana](#) 中所列的步驟操作。

使用網路聊天控制項進行測試

若要使用 Bot 服務中的整合式網路聊天控制項測試 Bot，請按一下以網路聊天測試並輸入訊息，即可驗證您的 Bot 是否正常運作。

使用模擬器進行測試

若要使用[模擬器](#)測試 Bot，請執行下列動作：

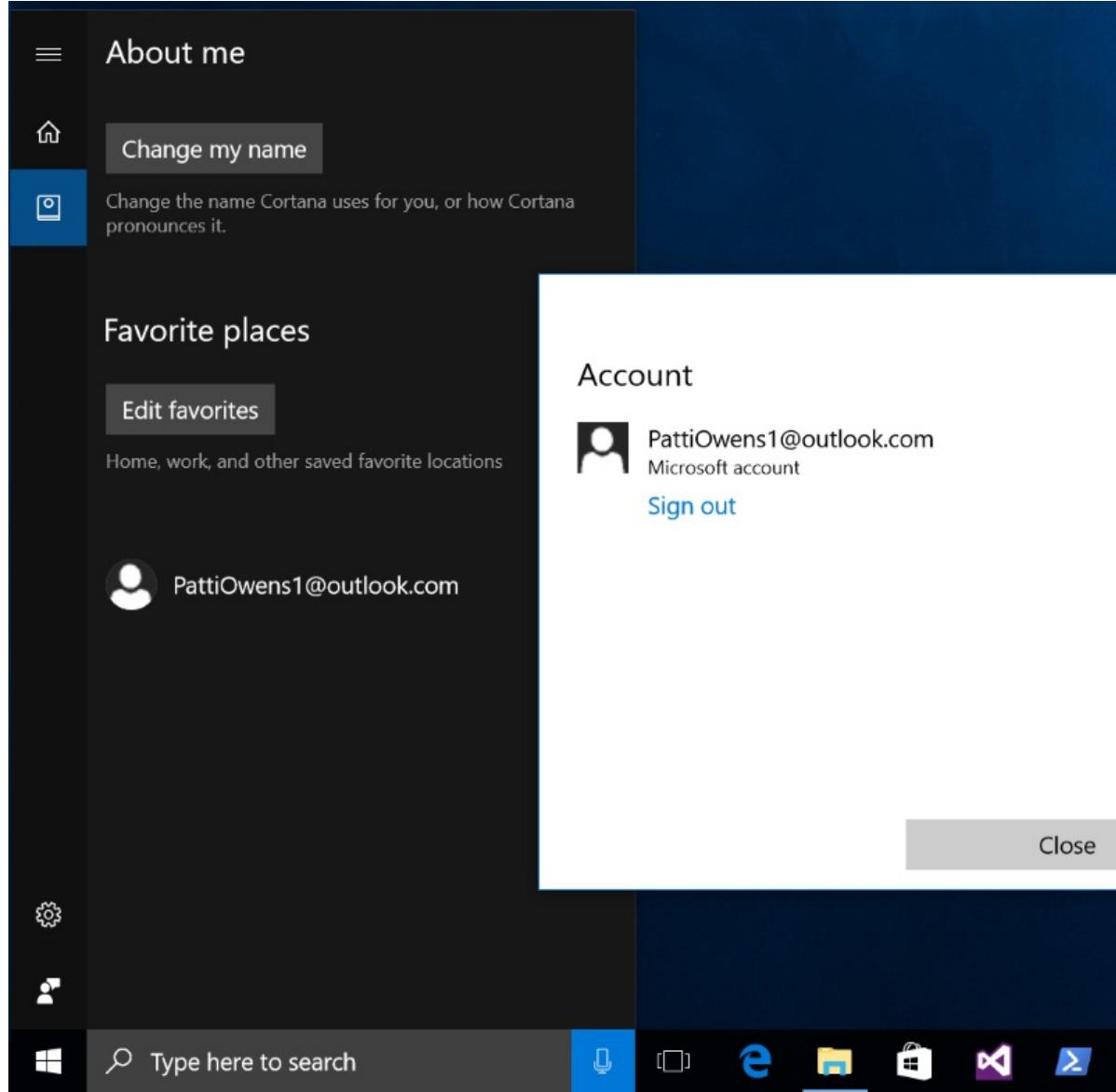
1. 執行 Bot。

2. 開啟模擬器，並填入所需的資訊。若要尋找 Bot 的 **AppID** 和 **AppPassword**，請參閱 [MicrosoftAppID](#) 和 [MicrosoftAppPassword](#)。
3. 按一下連線即可將模擬器連線至 Bot。
4. 輸入訊息，以驗證您的 Bot 是否正常運作。

使用 Cortana 進行測試

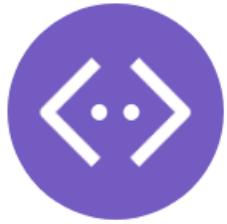
您可以對 Cortana 說出引動片語，以引動您的 Cortana 技能。

1. 開啟 Cortana。
2. 在 Cortana 中開啟「筆記本」，然後按一下關於我即可查看您要為 Cortana 使用的帳戶。請確定您是透過註冊 Bot 所使用的 Microsoft 帳戶登入。



3. 在 Cortana 應用程式或 Windows 的「詢問我任何事」搜尋方塊中按一下麥克風按鈕，然後說出您的 Bot 的**引動片語**。引動片語包含**引動名稱**，可以唯一識別要引動的技能。比方說，如果技能的引動名稱為「北風相片」，適當的引動片語可以包含「詢問北風相片...」或「告訴北風相片...」。

您可以在為 Cortana 設定時指定 Bot 的**引動名稱**。



* Skill icon

[Upload custom icon](#)

32,000 byte maximum size, png only

* Display name

Dice Roller

* Invocation name

Roller

* Long description

This is a bot that rolls dice.

* Short description

Rolls dice.

4. 如果 Cortana 辨識出您的引動片語，您的 Bot 就會在 Cortana 的畫布中啟動。

疑難排解

如果您的 Cortana 技能無法啟動，請檢查下列各項：

- 確定您是透過在 Bot Framework 入口網站中註冊 Bot 所使用的 Microsoft 帳戶登入。
- 按一下在網路聊天中測試來開啟聊天視窗並輸入訊息，即可檢查 Bot 是否正常運作。
- 請檢查您的引動名稱是否符合[指導方針](#)。如果您的引動名稱長度超過三個單字、難以發音或聽起來像其他字，Cortana 可能就難以辨識。
- 如果您的技能使用 LUIS 模型，請務必[啟用語音辨識預備](#)。

請參閱[啟用 Cortana 技能的偵錯](#)以取得其他疑難排解提示以及如何啟用 Cortana 儀表板中技能偵錯的相關資訊。

後續步驟

測試 Cortana 技能，並確認其按照您想要的方式運作後，您可以對一群 beta 版測試人員部署，或公開發佈。如需詳細資訊，請參閱[發佈 Cortana 技能](#)。

其他資源

- [Cortana 技能套件](#)

部署您的 Bot

2019/5/10 • [Edit Online](#)

適用於:  SDK v4  SDK v3

在本文中，我們會示範如何將 Bot 部署至 Azure。在依照步驟執行前閱讀本文很有用，您可完全了解部署 Bot 的相關事項。

必要條件

- 如果您沒有 Azure 訂用帳戶，請先建立[帳戶](#)再開始。
- 您已在本機電腦上開發的 CSharp、JavaScript 或 TypeScript Bot。
- 最新版的[Azure CLI](#)。

1. 準備部署

當您使用 Visual Studio 或 Yeoman 範本建立 Bot 時，所產生的原始程式碼會包含 `deploymentTemplates` 資料夾與 ARM 範本。此處敘述的部署程序會使用 ARM 範本，然後以 Azure CLI 在 Azure 中佈建所需的 Bot 資源。

IMPORTANT

在 Bot Framework SDK 4.3 版本中，我們已不再使用 .bot 檔案，而是改用 appsettings.json 或 .env 檔案來管理資源。若要了解如何將設定從 .bot 檔案遷移至 appsettings.json 或 .env 檔案，請參閱[管理 Bot 資源](#)。

登入 Azure

您已在本機建立及測試 Bot，而現在想要將其部署至 Azure。開啟命令提示字元以登入 Azure 入口網站。

```
az login
```

瀏覽器視窗會隨即開啟，以便您登入。

設定訂用帳戶

設定要使用的預設訂用帳戶。

```
az account set --subscription "<azure-subscription>"
```

如果您不確定哪個訂用帳戶要用於部署 Bot，可以使用 `az account list` 命令，檢視您帳戶的訂用帳戶清單。瀏覽至 Bot 資料夾。

建立應用程式註冊

註冊應用程式意謂著您可以使用 Azure AD 來驗證使用者，以及要求存取使用者資源。在 Azure 中，您的 Bot 需要已註冊的應用程式來允許 Bot 存取 Bot Framework 服務，以傳送和接收已驗證的訊息。若要透過 Azure CLI 建立註冊應用程式，請執行下列命令：

```
az ad app create --display-name "displayName" --password "AtLeastSixteenCharacters_0" --available-to-other-tenants
```

選項	說明
display-name	應用程式的顯示名稱。
password	應用程式密碼，也稱為「用戶端密碼」。密碼必須至少有 16 個字元長，至少包含 1 個大寫或小寫字母，以及至少包含 1 個特殊字元
available-to-other-tenants	應用程式可以從任何 Azure AD 租用戶使用。這必須設為 <code>true</code> ，才能讓您的 Bot 使用 Azure Bot 服務通道。

上述命令會輸出具有 `appId` 索引鍵的 JSON，請儲存此索引鍵的值以用於 ARM 部署，該值會用在 `appId` 參數上。提供的密碼將會用於 `appSecret` 參數。

您可以將 Bot 部署在新的資源群組或現有的資源群組中。選擇最適合您的選項。

- 透過 ARM 範本部署 (使用新資源群組)
- 透過 ARM 範本部署 (使用現有的資源群組)

建立 Azure 資源

您會在 Azure 中建立新的資源群組，然後使用 ARM 範本在其中建立指定資源。在此案例中，我們提供 App Service 方案、Web 應用程式和 Bot 通道註冊。

```
az deployment create --name "<name-of-deployment>" --template-file "template-with-new-rg.json" --location "location-name" --parameters appId="<msa-app-guid>" appSecret="<msa-app-password>" botId="<id-or-name-of-bot>" botSku=F0 newAppServicePlanName="<name-of-app-service-plan>" newWebAppName="<name-of-web-app>" groupName="<new-group-name>" groupLocation="<location>" newAppServicePlanLocation="<location>"
```

選項	說明
name	部署的自訂名稱。
template-file	ARM 範本的路徑。您可以使用專案資料夾 <code>deploymentTemplates</code> 中提供的 <code>template-with-new-rg.json</code> 檔案。
location	位置。值的來源： <code>az account list-locations</code> 。您可以使用 <code>az configure --defaults location=<location></code> 來設定預設位置。
parameters	提供部署參數值。執行 <code>az ad app create</code> 命令時所得的 <code>appId</code> 值。 <code>appSecret</code> 是您在上一個步驟中提供的密碼。 <code>botId</code> 參數應該是全域唯一，而且會用來作為不可變的 Bot 識別碼。此參數也會用來設定 Bot 的顯示名稱，而這是可變動的。 <code>botSku</code> 是定價層，可以是 F0 (免費) 或 S1 (標準)。 <code>newAppServicePlanName</code> 是 App Service 方案的名稱。 <code>newWebAppName</code> 是您要建立的 Web 應用程式名稱。 <code>groupName</code> 是您要建立的 Azure 資源群組名稱。 <code>groupLocation</code> 是 Azure 資源群組的位置。 <code>newAppServicePlanLocation</code> 是 App Service 方案的位置。

擷取或建立必要的 IIS/Kudu 檔案

針對 C# Bot

```
az bot prepare-deploy --lang Csharp --code-dir "." --proj-file-path "MyBot.csproj"
```

您必須提供與 --code-dir 對應的 .csproj 檔案路徑。這可以透過 --proj-file-path 引數來執行。此命令會將 --code-dir 和 --proj-file-path 解析為 "./MyBot.csproj"

針對 JavaScript Bot

```
az bot prepare-deploy --code-dir "." --lang Javascript
```

此命令會擷取 web.config，這是讓 Node.js 應用程式可以在 Azure App Service 上與 IIS 搭配運作的必要項目。請確定 web.config 已儲存到您 Bot 的根目錄。

針對 TypeScript Bot

```
az bot prepare-deploy --code-dir "." --lang Typescript
```

此命令的運作方式類似上述的 JavaScript，但這適用於 Typescript Bot。

手動壓縮程式碼目錄

使用未設定的 [zip 部署 API](#)來部署您 Bot 的程式碼時，Web 應用程式/Kudu 的行為會如下所示：

根據預設，Kudu 會假設來自 zip 檔案的部署都已可供執行，而且部署期間不需要額外的建置步驟，例如 `npm install` 或 `dotnet restore/dotnet publish`。

因此，務必在要部署至 Web 應用程式的 zip 檔案中包含建置程式碼和所有必要相依性，否則您的 Bot 不會如預期般運作。

IMPORTANT

壓縮專案檔之前，請確定您位在正確的資料夾。

- 針對 C# Bot，這是具有 .csproj 檔案的資料夾。
- 針對 JS Bot，這是具有 app.js 或 index.js 檔案的資料夾。

如果您的根資料夾位置不正確，**Bot 將無法在 Azure 入口網站中執行**。

2. 將程式碼部署至 Azure

此時，我們已準備好將程式碼部署至 Azure Web 應用程式。從命令列執行下列命令，即可對 Web 應用程式使用 kudu zip 推送部署來執行部署。

```
az webapp deployment source config-zip --resource-group "<new-group-name>" --name "<name-of-web-app>" --src "code.zip"
```

選項	說明
resource-group	您先前在 Azure 中建立的資源群組名稱。
name	您稍早使用的 Web 應用程式名稱。
src	您所建立的 zip 檔案路徑。

3. 在網路聊天中測試

- 在 Azure 入口網站中，移至 Web 應用程式 Bot 的刀鋒視窗。
- 在 [Bot 管理] 區段中，按一下 [在網路聊天中測試]。Azure Bot Service 會將網路聊天控制項載入，並連線至 Bot。
- 在成功部署後等候幾秒，選擇性地重新啟動您的 Web 應用程式，以清除任何快取。回到您的 Web 應用程式 Bot 刀鋒視窗，使用 Azure 入口網站中提供的網路聊天進行測試。

其他資訊

將 Bot 部署至 Azure 時，需支付您所使用的服務。[計費與成本管理](#)一文協助您了解 Azure 計費方式、監視使用量和成本，以及管理帳戶和訂用帳戶。

後續步驟

[設定持續部署](#)

設定連續部署

2019/5/10 • [Edit Online](#)

適用於：  SDK v4  SDK v3

本文將示範如何設定 Bot 的持續部署。您可以啟用持續部署，自動將來源存放庫中的程式碼變更部署至 Azure。在本主題中，我們將討論如何設定 GitHub 的持續部署。若要了解其他原始檔控制系統的持續部署設定，請參閱此頁面底部的 [其他資源] 區段。

必要條件

- 如果您沒有 Azure 訂用帳戶，請在開始前建立 [免費帳戶](#)。
- 您必須先[將 Bot 部署至 Azure](#)，才能啟用持續部署。

準備您的存放庫

確定您的存放庫根目錄含有專案中的正確檔案。這可讓您從 Azure App Service Kudu 組建伺服器取得自動建置。

執行階段	根目錄檔案
ASP.NET Core	.sln 或 .csproj
Node.js	含啟動指令碼的 server.js、app.js 或 package.json

使用 GitHub 進行持續部署

若要啟用搭配 GitHub 的持續部署，請巡覽至 Azure 入口網站中適用您 Bot 的 **App Service** 頁面。

按一下 [部署中心] > [GitHub] > [授權]。

The screenshot shows the Azure Deployment Center interface. On the left, a sidebar lists various settings like Overview, Activity log, and Deployment slots. The 'Deployment Center' option is highlighted with a red box. The main area displays a four-step process: SOURCE CONTROL, BUILD PROVIDER, CONFIGURE, and SUMMARY. Under SOURCE CONTROL, the 'Github' option is selected and highlighted with a red box. A message says 'Configure continuous integration with a GitHub repo.' Below it, a status message says 'Not Authorized'. At the bottom right of this section is a blue 'Authorize' button, also highlighted with a red box. Other options shown include 'Azure Repos', 'Bitbucket', 'OneDrive', 'Dropbox', and 'External', all with 'Not Authorized' status.

在開啟的瀏覽器視窗中，按一下 [授權 AzureAppService]。

The screenshot shows the GitHub authorization dialog titled 'Authorize Azure App Service'. It features icons for GitHub and Azure. The main text reads: 'Azure App Service by AzureAppService wants to access your [REDACTED] account'. Below this are two dropdown menus: 'Repository webhooks and services' (Admin access) and 'Repositories' (Public and private). At the bottom is a large green 'Authorize AzureAppService' button, which is highlighted with a red box. A note below the button says: 'Authorizing will redirect to https://functions.azure.com'.

在授權 **AzureAppService** 後，返回 Azure 入口網站中的部署中心。

1. 按一下 [繼續]。
2. 選取 [App Service 建置服務]。
3. 按一下 [繼續]。

4. 選取 [組織]、[存放庫] 和 [分支]。
5. 按一下 [繼續]，然後按一下 [完成] 以完成設定。

此時，搭配 GitHub 的持續部署已設定完成。每當您認可來源程式碼存放庫時，變更就會自動部署至 Azure Bot 服務。

停用連續部署

設定 Bot 以進行持續部署時，您可能不會使用線上程式碼編輯器來變更 Bot。如果您想要使用線上程式碼編輯器，可以暫時停用持續部署。

若要停用持續部署，請執行下列作業：

1. 在 [Azure 入口網站](#) 中，請移至 Bot 的 [所有 App Service 設定] 刀鋒視窗，然後按一下 [部署中心]。
2. 按一下 [中斷連線]，停用持續部署。若要重新啟用持續部署，請重複上述適當章節的步驟。

其他資源

- 若要從 BitBucket 與 Azure DevOps Services 啟用連續部署，請參閱[使用 Azure App Service 進行持續部署](#)。

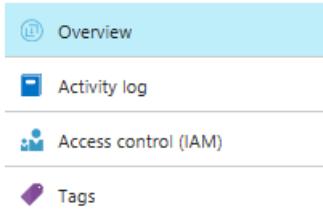
管理 Bot

2019/5/10 • [Edit Online](#)

適用於：  SDK v4  SDK v3

本主題說明如何使用 Azure 入口網站管理 Bot。

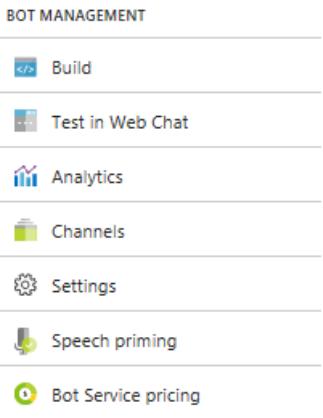
Bot 設定概觀



在 [概觀] 刀鋒視窗中，您可以找到關於 Bot 的高階資訊。例如，您可以看到 Bot 的 [訂用帳戶 ID]、[定價層] 和 [傳訊端點]。

Bot 管理

您可以在 <**BOT 管理**> 一節下找到大部分的 Bot 管理選項。以下是可協助您管理 Bot 的選項清單。



選項	說明
建置	[建置] 索引標籤可提供選項來變更 Bot。此選項不適用於僅限註冊 Bot。
在網絡聊天中測試	使用整合式網路聊天控制項來協助您快速測試 Bot。
分析	如果 Bot 已開啟分析，您可以檢視 Application Insights 針對該 Bot 收集的分析資料。
Channels	設定 Bot 用來與使用者進行通訊的通道。
設定	管理各種 Bot 設定檔設定，例如顯示名稱、分析和傳訊端點。

選項	說明
語音預備	管理 LUIS 應用程式與 Bing 語音服務之間的連線。
Bot 服務定價	管理 Bot 服務的定價層。

App service 設定

APP SERVICE SETTINGS

☰ Application Settings

🌐 All App service settings

[應用程式設定] 刀鋒視窗中包含關於 Bot 的詳細資訊，例如 Bot 的環境、偵錯設定和應用程式設定金鑰。

MicrosoftAppID 和 MicrosoftAppPassword

MicrosoftAppID 和 **MicrosoftAppPassword** 會保存在 Bot 組態檔或 Azure Key Vault 內。若要擷取，請下載您的 Bot 設定或組態檔，或是存取 Azure Key Vault。您可能需要使用識別碼和密碼在本機進行測試。

NOTE

Bot 通道註冊 Bot 服務隨附 *MicrosoftAppID*，但因為沒有應用程式服務與這種類型的服務建立關聯，所以沒有任何 [應用程式設定] 刀鋒視窗可供您查閱 *MicrosoftAppPassword*。若要取得密碼，您必須先產生密碼。若要產生 **Bot 通道註冊** 的密碼，請參閱 [Bot 通道註冊密碼](#)

後續步驟

由於您已在 Azure 入口網站中探索 Bot 服務刀鋒視窗，請了解如何使用線上程式碼編輯器來自訂 Bot。

[使用線上程式碼編輯器](#)

Bot 分析

2019/2/28 • [Edit Online](#)

Analytics 是 [Application Insights](#) 的擴充功能。Application Insights 提供服務層級和檢測資料，例如流量、延遲和整合。Analytics 提供關於使用者、訊息和通道資料的對話層級報告。

檢視針對 Bot 的分析

若要存取 Analytics，請在 Azure 入口網站中開啟 Bot，然後按一下 [Analytics]。

資料太多嗎？您可以對連結到 Bot 的 Application Insights [啟用並設定取樣](#)。這可減少遙測流量和儲存體，同時能夠維持統計上正確的分析。

指定通道

選擇哪個通道會出現在下圖中。請注意，如果 Bot 未在通道上啟用，就不會有來自該通道的任何資料。

The screenshot shows a dropdown menu titled 'Select Channels' with the following options:

- Bing
- Cortana
- Direct Line
- Email
- ...

Below the menu, a date range is displayed: 11/3/2018 - 12/4/2018.

Channel	Channel
<input checked="" type="checkbox"/> Bing	<input checked="" type="checkbox"/> Cortana
<input checked="" type="checkbox"/> Direct Line	<input checked="" type="checkbox"/> Email
<input checked="" type="checkbox"/> Facebook	<input checked="" type="checkbox"/> GroupMe
<input checked="" type="checkbox"/> Kaizala	<input checked="" type="checkbox"/> Kik
<input checked="" type="checkbox"/> Microsoft Teams	<input checked="" type="checkbox"/> Skype
<input checked="" type="checkbox"/> Skype for Business	<input checked="" type="checkbox"/> Slack
<input checked="" type="checkbox"/> Telegram	<input checked="" type="checkbox"/> Twilio (SMS)
<input checked="" type="checkbox"/> Web Chat	<input checked="" type="checkbox"/> WeChat

- 勾選核取方塊以在圖表中包含通道。
- 清除核取方塊以從圖表中移除通道。

指定時間週期

分析僅適用於過去 90 天。當 Application Insights 啟用時，資料收集隨即開始。

The screenshot shows a dropdown menu titled 'Select Time Period' with the following options:

- Bing
- Cortana
- Direct Line
- Email
- ...

Below the menu, a date range is displayed: 11/3/2018 - 12/4/2018.

Time period options:

- Last Hour
- Last Day
- Last Week
- Last Month
- Last 90 Days

按一下下拉式功能表，然後按一下應該要顯示圖表的時間量。請注意，變更整體時間範圍會導致圖表上的時間增量 (X 軸) 據以變更。

總計

在指定時間範圍內的作用中使用者，以及傳送及接收的訊息總數。破折號 -- 表示沒有活動。

保留

「保留」會追蹤多少使用者傳送一則訊息，然後又返回再傳送另一則。圖表是累積的 10 天時間範圍；變更時間範圍

不會影響結果。

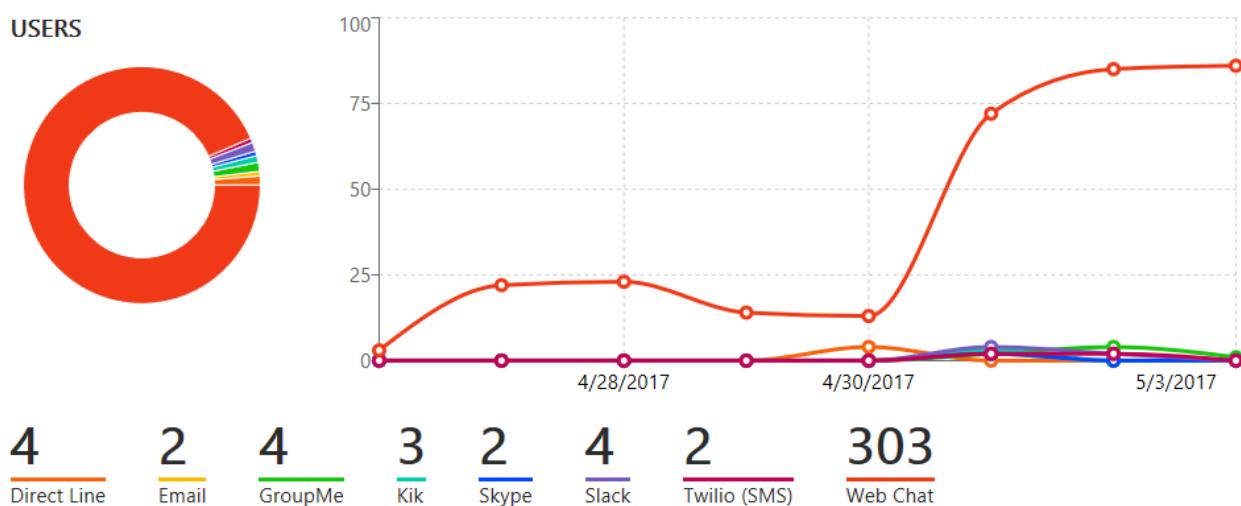
RETENTION - % USERS WHO MESSED AGAIN (LAST 10 DAYS)

Date	Users	Days later									
		1	2	3	4	5	6	7	8	9	10
4/23/2017	0	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
4/24/2017	13	31%	8%	8%	8%	8%	8%	8%	15%	8%	
4/25/2017	43	2%	2%	2%	5%	5%	2%	5%	2%		
4/26/2017	27	4%	4%	4%	4%	4%	4%	4%			
4/27/2017	37	3%	3%	3%	3%	3%	3%				
4/28/2017	22	5%	5%	5%	5%	5%					
4/29/2017	23	9%	9%	9%	9%						
4/30/2017	14	7%	7%	7%							
5/1/2017	17	18%	12%								
5/2/2017	87	9%									

請注意，可能的最近日期是在兩天前；使用者在前天傳送訊息，然後在昨天「返回」。

使用者

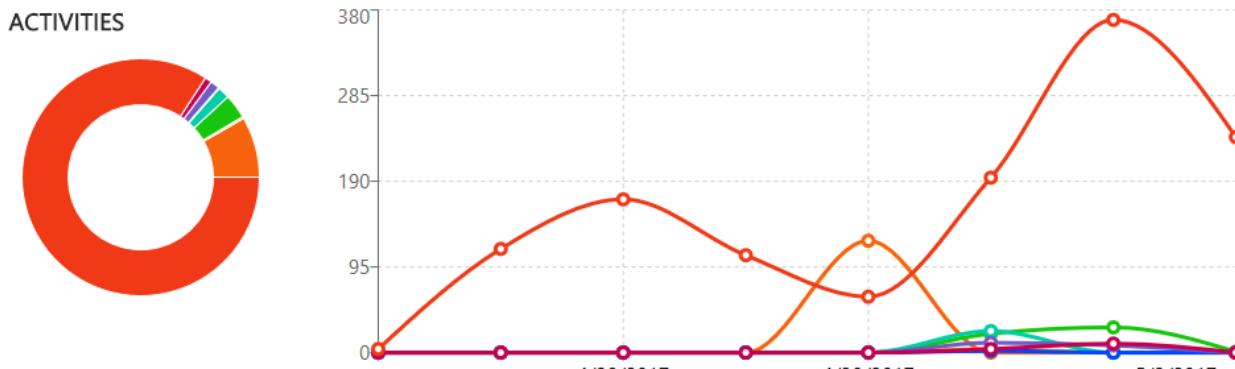
使用者圖表會追蹤有多少使用者在指定時間範圍期間，使用各個通道存取 Bot。



- 百分比圖表會顯示使用者使用各個通道的百分比。
- 折線圖會指出有多少使用者曾經在特定時間存取 Bot。
- 折線圖的圖例會指出哪個色彩表示哪個通道，並且包含指定時間週期期間的使用者總數。

活動

「活動」圖表會追蹤在指定的時間範圍內，使用了哪個通道傳送及接收了多少活動。



- 百分比圖表會顯示透過各個通道通訊的活動百分比。
- 折線圖會指出在指定的時間範圍內，傳送及接收了多少活動。
- 折線圖的圖例會指出哪個線條色彩表示哪個通道，以及指定的時間週期內，在該通道傳送及接收的活動總數。

啟用分析

在啟用並設定 Application Insights 之前，都無法使用 Analytics。Application Insights 一旦啟用，就會開始收集資料。例如，如果 Application Insights 在一週前已針對六個月的 Bot 啟用，它就會收集一週的資料。

NOTE

Analytics 同時需要 Azure 訂用帳戶與 Application Insights 資源。若要存取 Application Insights，請在 [Azure 入口網站](#) 中開啟 Bot，然後按一下 [設定]。

建立 Bot 資源時，您可以新增 Application Insights。

您也可以稍後建立 Application Insights 資源並將其連線到 Bot。

1. 建立 Application Insights 資源。
2. 在儀表板中開啟 Bot。按一下 [設定]，然後向下捲動至 [Analytics] 區段。
3. 輸入資訊以便將 Bot 連線至 Application Insights。所有欄位皆為必填項目。

Analytics

Application Insights Instrumentation key [?](#)

Instrumentation key (Azure Application Insights key)

Application Insights API key [?](#)

API key (User-Generated Application Insights API key)

Application Insights Application ID [?](#)

Application ID (Application Insights Application ID)

如需如何找出這些值的詳細資訊，請參閱 [Application Insights 金鑰](#)。

其他資源

- [Application Insights 金鑰](#)

將 Bot 連線至通道

2019/5/10 • [Edit Online](#)

通道可連接 Bot 與通訊應用程式。您可以將 Bot 設定為連線到您想要提供此 Bot 的通道。透過 Azure 入口網站設定的 Bot Framework Service，可將您的 Bot 連接至這些通道，以利 Bot 和使用者之間的通訊。您可以連線至許多熱門服務，例如 [Cortana](#)、[Facebook Messenger](#)、[Kik](#) 和 [Slack](#)，以及數個其他服務。已為您預先設定[網路聊天](#)通道。除了 Bot Connector 服務所提供的標準通道，您也可以使用 [Direct Line](#) 作為通道，將 Bot 連接到您自己的用戶端應用程式。

藉由將 Bot 傳送給通道的訊息正規化，Bot Framework Service 可讓您以不限通道的方式開發 Bot。這牽涉到從 Bot 架構結構描述轉換成通道的結構描述。不過，如果通道不支援 Bot 架構結構描述的所有層面，則服務會嘗試將訊息轉換成通道支援的格式。比方說，如果 Bot 將訊息傳送至電子郵件通道，而訊息中包含具有動作按鈕的卡片，則連接器可能會將卡片傳送為映像，並在訊息文字中將動作包含為連結。

針對大部分的通道，您必須提供通道設定資訊，才能在通道上執行 Bot。大部分的通道會要求 Bot 要有通道帳戶，Facebook Messenger 等其他通道則會要求 Bot 也要有已向通道註冊的應用程式。

若要設定 Bot 以連線到通道，請完成下列步驟：

1. 登入 [Azure 入口網站](#)。
2. 選取您想要設定的 Bot。
3. 在 [Bot Service] 刀鋒視窗中，按一下 [Bot 管理] 底下的 [通道]。
4. 按一下想要在 Bot 中新增的通道圖示。

The screenshot shows the Azure Bot Service blade. On the left, there's a navigation menu with items like Overview, Activity log, Access control (IAM), Tags, BOT MANAGEMENT (Build, Test in Web Chat, Analytics, Channels, Settings, Speech priming, Bot Service pricing), and APP SERVICE SETTINGS (Application Settings). The 'Channels' item under BOT MANAGEMENT is highlighted with a blue border. The main area is titled 'Connect to channels'. It lists two channels: 'Skype' and 'Web Chat', both marked as 'Running'. There are 'Edit' buttons next to each entry. Below the table, there's a link 'Get bot embed codes' and a placeholder 'Add a featured channel' with three icons: a blue circle, a globe, and a purple square.

Name	Health	Published	
Skype	Running	--	Edit
Web Chat	Running	--	Edit

設定通道後，該通道上的使用者就可以開始使用 Bot。

發佈 Bot

每個通道的發佈程序不同。

Cortana

Bot 會從[儀表板](#)發佈至 Cortana，並且用於提供 Cortana 技術。發佈 Bot 即可加以提交進行檢閱。您可部署 Cortana 技術以供自己使用、部署到小型群組，或發佈至全球。

Skype

Bot 會從**組態頁面**發佈至 Skype。發佈 Bot 即可加以提交進行檢閱。檢閱之前，Bot 受限於 100 個連絡人。已核准的 Bot 沒有連絡人限制，您可選擇將 Bot 包含在 Skype Bot 目錄中。

商務用 Skype

租用戶管理員會向**商務用 Skype Online 租用戶**註冊商務用 Skype Bot。

TIP

若要檢視檢閱狀態，請在 [Bot Framework 入口網站](#)中開啟 Bot，然後按一下 [通道]。如果 Bot 未獲准，則結果會連結至其原因。進行必要的變更之後，重新提供 Bot 進行檢閱。

其他資源

SDK 包含您可用來建置 Bot 的範例。請造訪 [GitHub 上的範例存放庫](#)以查看範例清單。

實作通道特定的功能

2019/4/26 • [Edit Online](#)

NOTE

本主題適用於最新版的 SDK (v4)。您可以在[這裡](#)找到舊版 SDK (v3) 的內容。

某些通道所提供的功能，無法只透過訊息文字和附件來實作。若要實作通道特定的功能，您可以將原生中繼資料傳遞至活動物件「通道資料」屬性中的通道。例如，Bot 可以使用通道資料屬性，指示 Telegram 傳送貼圖或指示 Office365 傳送電子郵件。

此文章說明如何使用訊息活動的通道資料屬性，來實作此通道特定的功能：

通道	功能
電子郵件	傳送及接收電子郵件，其中包含本文、主旨和重要性中繼資料
Slack	傳送不失真的 Slack 訊息
Facebook	原生傳送 Facebook 通知
Telegram	執行 Telegram 特定動作，例如共用語音備忘或貼圖
Kik	傳送和接收原生 Kik 訊息

NOTE

活動物件的通道資料屬性值是一個 JSON 物件。因此，本文中的範例會說明各種案例中 `channelData` JSON 屬性的預期格式。若要使用 .NET 建立 JSON 物件，請使用 `JObject` (.NET) 類別。

建立自訂的電子郵件訊息

若要建立電子郵件訊息，請將活動物件的通道資料屬性設定為包含這些屬性的 JSON 物件：

屬性	說明
<code>bccRecipients</code>	以分號 (:) 分隔的電子郵件地址字串，用來新增至訊息的 [Bcc] (密件副本) 欄位。
<code>ccRecipients</code>	以分號 (:) 分隔的電子郵件地址字串，用來新增至訊息的 [Cc] (副本) 欄位。
<code>htmlBody</code>	會指定電子郵件訊息本文的 HTML 文件。請參閱通道的文件，以取得受支援 HTML 元素和屬性的資訊。
<code>importance</code>	電子郵件的重要性層級。有效值為高、一般和低。預設值為一般。

屬性	說明
主旨	電子郵件的主旨。請參閱通道的文件，以取得欄位需求的資訊。
toRecipients	以分號 (;) 分隔的電子郵件地址字串，用來新增至訊息的 [收件者] 欄位。

NOTE

Bot 透過電子郵件通道從使用者收到的訊息，可能會包含通道資料屬性，且其中已填入類似上述的 JSON 物件。

此程式碼片段顯示自訂電子郵件訊息的 `channelData` 屬性範例。

```
"channelData": {  
    "type": "message",  
    "locale": "en-US",  
    "channelID": "email",  
    "from": { "id": "mybot@mydomain.com", "name": "My bot"},  
    "recipient": { "id": "joe@otherdomain.com", "name": "Joe Doe"},  
    "conversation": { "id": "123123123123", "topic": "awesome chat" },  
    "channelData":  
    {  
        "htmlBody": "<html><body style = /"font-family: Calibri; font-size: 11pt;/" >This is more than  
awesome.</body></html>",  
        "subject": "Super awesome message subject",  
        "importance": "high",  
        "ccRecipients": "Yasemin@adatum.com;Temel@adventure-works.com"  
    }  
}
```

建立不失真的 Slack 訊息

若要建立不失真的 Slack 訊息，請將活動物件的通道資料屬性設定為 JSON 物件，以指定 [Slack 訊息](#)、[Slack 附件](#) 和/或 [Slack 按鈕](#)。

NOTE

若要在 Slack 訊息中支援按鈕，您必須在 [將 Bot 連線至 Slack 通道](#) 時啟用 [互動式訊息]。

此程式碼片段顯示自訂 Slack 訊息的 `channelData` 屬性範例。

```

"channelData": {
    "text": "Now back in stock! :tada:",
    "attachments": [
        {
            "title": "The Further Adventures of Slackbot",
            "author_name": "Stanford S. Strickland",
            "author_icon": "https://api.slack.com/img/api/homepage_custom_integrations-2x.png",
            "image_url": "http://i.imgur.com/OJkaVOI.jpg?1"
        },
        {
            "fields": [
                {
                    "title": "Volume",
                    "value": "1",
                    "short": true
                },
                {
                    "title": "Issue",
                    "value": "3",
                    "short": true
                }
            ]
        },
        {
            "title": "Synopsis",
            "text": "After @episod pushed exciting changes to a devious new branch back in Issue 1, Slackbot notifies @don about an unexpected deploy..."
        },
        {
            "fallback": "Would you recommend it to customers?",
            "title": "Would you recommend it to customers?",
            "callback_id": "comic_1234_xyz",
            "color": "#3AA3E3",
            "attachment_type": "default",
            "actions": [
                {
                    "name": "recommend",
                    "text": "Recommend",
                    "type": "button",
                    "value": "recommend"
                },
                {
                    "name": "no",
                    "text": "No",
                    "type": "button",
                    "value": "bad"
                }
            ]
        }
    ]
}

```

當使用者按一下 Slack 訊息中的按鈕時，Bot 會收到回應訊息，其中的通道資料屬性已填入 `payload` JSON 物件。

`payload` 物件會指定原始訊息的內容、識別已按下的按鈕，並識別按下按鈕的使用者。

此程式碼片段顯示當使用者按一下 Slack 訊息中的按鈕時，Bot 所收到的訊息中所含有的 `channelData` 屬性範例。

```

"channelData": {
    "payload": {
        "actions": [
            {
                "name": "recommend",
                "value": "yes"
            }
        ],
        ...
    },
    "original_message": "...",
    "response_url": "https://hooks.slack.com/actions/..."
}
}

```

Bot 可以透過正常方式回覆此訊息，也可以將其回應直接張貼到 `payload` 物件的 `response_url` 屬性所指定的端點。如需何時及如何將回應張貼到 `response_url` 的資訊，請參閱 [Slack 按鈕](#)。

您可以使用下列 JSON 來建立動態按鈕。

```

{
    "text": "Would you like to play a game ?",
    "attachments": [
        {
            "text": "Choose a game to play!",
            "fallback": "You are unable to choose a game",
            "callback_id": "wopr_game",
            "color": "#3AA3E3",
            "attachment_type": "default",
            "actions": [
                {
                    "name": "game",
                    "text": "Chess",
                    "type": "button",
                    "value": "chess"
                },
                {
                    "name": "game",
                    "text": "Falken's Maze",
                    "type": "button",
                    "value": "maze"
                },
                {
                    "name": "game",
                    "text": "Thermonuclear War",
                    "style": "danger",
                    "type": "button",
                    "value": "war",
                    "confirm": {
                        "title": "Are you sure?",
                        "text": "Wouldn't you prefer a good game of chess?",
                        "ok_text": "Yes",
                        "dismiss_text": "No"
                    }
                }
            ]
        }
    ]
}

```

若要建立互動式功能表，請使用下列 JSON：

```
{
  "text": "Would you like to play a game ?",
  "response_type": "in_channel",
  "attachments": [
    {
      "text": "Choose a game to play",
      "fallback": "If you could read this message, you'd be choosing something fun to do right now.",
      "color": "#3AA3E3",
      "attachment_type": "default",
      "callback_id": "game_selection",
      "actions": [
        {
          "name": "games_list",
          "text": "Pick a game...",
          "type": "select",
          "options": [
            {
              "text": "Hearts",
              "value": "menu_id_hearts"
            },
            {
              "text": "Bridge",
              "value": "menu_id_bridge"
            },
            {
              "text": "Checkers",
              "value": "menu_id_checkers"
            },
            {
              "text": "Chess",
              "value": "menu_id_chess"
            },
            {
              "text": "Poker",
              "value": "menu_id_poker"
            },
            {
              "text": "Falken's Maze",
              "value": "menu_id_maze"
            },
            {
              "text": "Global Thermonuclear War",
              "value": "menu_id_war"
            }
          ]
        }
      ]
    }
  ]
}
```

建立 Facebook 通知

若要建立 Facebook 通知，請將活動物件的通道資料屬性設定為指定這些屬性的 JSON 物件：

屬性	說明
notification_type	通知類型 (例如 REGULAR , SILENT_PUSH , NO_PUSH)。
attachment	指定影像、視訊或其他多媒體類型的附件，或是回條等樣板化附件。

NOTE

如需 `notification_type` 屬性和 `attachment` 屬性的格式和內容詳細資料, 請參閱 [Facebook API 文件](#)。

此程式碼片段顯示 Facebook 回條附件的 `channelData` 屬性範例。

```
"channelData": {  
    "notification_type": "NO_PUSH",  
    "attachment": {  
        "type": "template"  
        "payload": {  
            "template_type": "receipt",  
            . . .  
        }  
    }  
}
```

建立 Telegram 訊息

若要建立訊息來實作 Telegram 特定動作 (例如, 共用語音備忘或貼圖), 請將活動物件的通道資料屬性設定為會指定這些屬性的 JSON 物件：

屬性	說明
method	要呼叫的 Telegram Bot API 方法。
parameters	所指定方法的參數。

支援下列 Telegram 方法：

- `answerInlineQuery`
- `editMessageCaption`
- `editMessageReplyMarkup`
- `editMessageText`
- `forwardMessage`
- `kickChatMember`
- `sendAudio`
- `sendChatAction`
- `sendContact`
- `sendDocument`
- `sendLocation`
- `sendMessage`
- `sendPhoto`
- `sendSticker`
- `sendVenue`
- `sendVideo`
- `sendVoice`
- `unbanChateMember`

如需這些 Telegram 方法和其參數的詳細資訊, 請參閱 [Telegram Bot API 文件](#)。

NOTE

- `chat_id` 參數通用於所有 Telegram 方法。如果您未指定 `chat_id` 作為參數，架構會為您提供識別碼。
- 不要傳遞內嵌檔案內容，而是應該指定使用 URL 和媒體類型的檔案，如下列範例所示。
- 在 Bot 從 Telegram 通道所收到的每則訊息內，`ChannelData` 屬性會包含 Bot 先前傳送的訊息。

此程式碼片段顯示 `channelData` 屬性範例，此屬性指定單一 Telegram 方法。

```
"channelData": {  
    "method": "sendSticker",  
    "parameters": {  
        "sticker": {  
            "url": "https://domain.com/path/gif",  
            "mediaType": "image/gif",  
        }  
    }  
}
```

此程式碼片段顯示 `channelData` 屬性範例，此屬性指定 Telegram 方法陣列。

```
"channelData": [  
    {  
        "method": "sendSticker",  
        "parameters": {  
            "sticker": {  
                "url": "https://domain.com/path/gif",  
                "mediaType": "image/gif",  
            }  
        }  
    },  
    {  
        "method": "sendMessage",  
        "parameters": {  
            "text": "<b>This message is HTML formatted.</b>",  
            "parse_mode": "HTML"  
        }  
    }  
]
```

建立原生的 Kik 訊息

若要建立原生的 Kik 訊息，請將活動物件的通道資料屬性設定為指定此屬性的 JSON 物件：

屬性	說明
上限	Kik 訊息的陣列。如需 Kik 訊息格式的詳細資訊，請參閱 Kik 訊息格式 。

此程式碼片段顯示原生 Kik 訊息的 `channelData` 屬性範例。

```

"channelData": {
  "messages": [
    {
      "chatId": "c6dd8165...",
      "type": "link",
      "to": "kikhandle",
      "title": "My Webpage",
      "text": "Some text to display",
      "url": "http://botframework.com",
      "picUrl": "http://lorempixel.com/400/200/",
      "attribution": {
        "name": "My App",
        "iconUrl": "http://lorempixel.com/50/50/"
      },
      "noForward": true,
      "kikJsData": {
        "key": "value"
      }
    }
  ]
}

```

建立 LINE 訊息

若要建立會實作 LINE 特有訊息類型 (例如貼圖、範本或 LINE 特有動作類型, 像是開啟手機相機) 的訊息, 請將活動物件的通道資料屬性設定為會指定 LINE 訊息類型和動作類型的 JSON 物件。

屬性	說明
type	LINE 動作/訊息類型名稱

支援這些 LINE 訊息類型：

- 貼圖
- 影像地圖
- 範本 (按鈕、確認、橫向捲軸)
- Flex

您可以在訊息類型 JSON 物件的動作欄位中, 指定這些 LINE 動作：

- Postback
- 訊息
- URI
- Datetimerpicker
- Camera
- Camera roll
- 位置

如需這些 LINE 方法和其參數的詳細資訊, 請參閱 [LINE Bot API 文件](#)。

此程式碼片段會舉例示範 `channelData` 屬性, 此屬性會指定通道訊息類型 `ButtonTemplate` 和 3 個動作類型: `camera`、`cameraRoll`、`Datetimerpicker`。

```
"channelData": {  
    "type": "ButtonsTemplate",  
    "altText": "This is a buttons template",  
    "template": {  
        "type": "buttons",  
        "thumbnailImageUrl": "https://example.com/bot/images/image.jpg",  
        "imageAspectRatio": "rectangle",  
        "imageSize": "cover",  
        "imageBackgroundColor": "#FFFFFF",  
        "title": "Menu",  
        "text": "Please select",  
        "defaultAction": {  
            "type": "uri",  
            "label": "View detail",  
            "uri": "http://example.com/page/123"  
        },  
        "actions": [{  
            "type": "cameraRoll",  
            "label": "Camera roll"  
        },  
        {  
            "type": "camera",  
            "label": "Camera"  
        },  
        {  
            "type": "datetimepicker",  
            "label": "Select date",  
            "data": "storeId=12345",  
            "mode": "datetime",  
            "initial": "2017-12-25t00:00",  
            "max": "2018-01-24t23:59",  
            "min": "2017-12-25t00:00"  
        }  
    ]  
}
```

其他資源

- 實體和活動類型
- Bot Framework -- 活動結構描述

將 Bot 連線至 Cortana

2019/5/10 • [Edit Online](#)

Cortana 是具有語音功能的通道，除了文字對話之外，還可以傳送和接收語音訊息。如要將 Bot 連線至 Cortana，應將 Bot 設計為適用於文字和語音。Cortana 技能是可以使用 Cortana 用戶端叫用的 Bot。發佈 Bot 即可將 Bot 加入可用技能的清單。

如要新增 Cortana 通道，請在 [Azure 入口網站](#) 中開啟 Bot，按一下通道刀鋒視窗，然後按一下 **Cortana**。

The screenshot shows the Azure Bot Channels Registration interface. On the left, there's a sidebar with various navigation links: Home, Overview, Activity log, Access control (IAM), Tags, Bot management, Test in Web Chat, Analytics, and Channels. The 'Channels' link is highlighted with a red box. In the main content area, the title 'Connect to channels' is displayed above a table. The table has columns for Name, Health, and Published. It shows one entry: 'Web Chat' with 'Running' status and '--' under Published. There's an 'Edit' button with a pencil icon. Below the table, there's a section titled 'Add a featured channel' with four icons: a blue circle, a globe, Microsoft Teams, and an 'S' logo. The first icon is also highlighted with a red box.

設定 Cortana

將您的 Bot 連線至 Cortana 通道時，Bot 的部分基本資訊會預先填入註冊表格。請仔細檢閱此項資訊。此表格包含下列欄位。

欄位	說明
技能圖示	叫用技能時，圖示就會顯示在 Cortana 畫布上。圖示也可用於搜尋技能（例如 Microsoft 網上商店）。（上限為 32KB，僅限 PNG 檔）。
顯示名稱	Cortana 技能的名稱會顯示於視覺 UI 的頂端。（上限為 30 個字元）
引動名稱	這是使用者叫用技能時說出的名稱。名稱不得超過三個字組，且應易於發音。請參閱 引動名稱指導方針 ，進一步了解如何選擇此名稱。

Default Settings

Skill information



Skill icon

[Upload icon](#)

32 KB max, PNG only

* Display Name

* Invocation Name

Manage user identity through Connected Services

Cortana can manage your skill's user identity and tokens across devices if you add your OAuth 2 identity provider as a Connected Service. To learn more about Connected Services, please refer to the [Connected Services developer docs](#).

Cortana should manage my user's identity

Request user profile data

一般 Bot 資訊

在透過連線服務區段管理使用者身分識別下方按下選項即可啟用。填寫表單。

標有星號 (*) 的欄位為必填項目。Bot 必須先發佈至 Azure, 才能連線至 Cortana。

Manage user identity through Connected Services

Cortana can manage your skill's user identity and tokens across devices if you add your OAuth 2 identity provider as a Connected Service. To learn more about Connected Services, please refer to the [Connected Services developer docs](#).

Cortana should manage my user's identity



When should Cortana prompt for a user to sign in? *

- Sign in at invocation
- Sign in when required [Learn more](#)

* Account name

Enter property value

* Client ID for third-party services

Enter property value

Space-separated list of scopes

List of scopes

* Authorization URL

Enter property value

* Token options

- GET
- POST

* Grant type

- Authorization code
- Implicit

* Token URL

https://

* Client secret/password for third party services

* Client authorization scheme

- HTTP Basic (Recommended)
- Credentials in request body

This skill's Connected Service requires intranet access to authenticate users (leave this unchecked if you are unsure).

Request user profile data

With the user's consent, bots can utilize user information from Cortana's notebook to customize interactions. Request user profile information below.

Data

+ [Add a user profile request](#)

[Back to Channels](#)

[Deploy on Cortana](#)

Cortana 何時應該提示使用者登入

如果您希望 Cortana 讓使用者在叫用您的技能時登入，請選取 [在引動過程登入]。

如果您使用 Bot Service 登入卡片讓使用者登入，請選取 [在必要時登入]。一般而言，只有在使用者使用需要驗證的功能時，才能使用此選項登入。當您的技能傳送包含登入卡附件的訊息時，Cortana 會忽略登入卡，並使用連線帳戶設定執行授權流程。

帳戶名稱

您想要在使用者登入您的技能時顯示的技能名稱。

第三方服務的用戶端識別碼

您的 Bot 應用程式識別碼。當您註冊 Bot 時，就會收到識別碼。

範圍的空格分隔清單

指定服務所需範圍 (請參閱服務的文件)。

授權 URL

設定為 `https://login.microsoftonline.com/common/oauth2/v2.0/authorize`。

權杖選項

選取 [POST]。

授與類型

選取 [授權碼] 以使用程式碼授與流程，或選取 [隱含] 以使用隱含流程。

權杖 URL

將 [授權碼] 授與類型設定為 `https://login.microsoftonline.com/common/oauth2/v2.0/token`。

第三方服務的用戶端密碼

Bot 的密碼。當您註冊 Bot 時，就會收到密碼。

用戶端驗證配置

選取 [HTTP 基本]。

驗證使用者所需的網際網路存取權

讓此屬性保持未核取狀態。

要求使用者設定檔資料 (選用)

Cortana 會提供許多不同類型的使用者設定檔資訊，供您存取並為使用者自訂 Bot。例如，如果技能有使用者名稱和位置的存取權，則可以為技能自訂回覆，例如「嗨，Kamran，祝你在華盛頓州倍有福有愉快的一天」。

按一下 [新增使用者設定檔要求]，然後從下拉式清單中選取您想查看的使用者設定檔資訊。新增用來從 Bot 程式碼存取這項資訊的易記名稱。

在 Cortana 上部署

填妥 Cortana 技能的註冊表單後，請按一下 [在 Cortana 上部署] 以完成連線。這麼做會將您重新導向回您 Bot 的通道刀鋒視窗中，Bot 現應已連線至 Cortana。

此時您的 Bot 已經部署為您帳戶的 Cortana 技能。

後續步驟

- [Cortana 技能套件](#)
- [啟用偵錯](#)
- [發佈 Cortana 技能](#)

關於 Direct Line

2019/5/10 • [Edit Online](#)

Bot Framework Direct Line 通道是能輕鬆將 Bot 整合到您行動應用程式、網頁或其他應用程式的方式。Direct Line 有三種形式：

- Direct Line 服務 – 全域且健全的服務，適用於大部分開發人員
- Direct Line App Service 擴充功能 - 著重於安全性和效能的 Direct Line 功能 (2019 年 5 月起以個人預覽版提供)
- Direct Line Speech – 適用於高效能語音 (2019 年 5 月起以個人預覽版提供)

您可以藉由評估每個供應項目的功能，以及您方案的需求，來選擇最適合您的 Direct Line 供應項目。這些供應項目會隨著時間簡化。

	DIRECT LINE	DIRECT LINE APP SERVICE 擴充功能	DIRECT LINE SPEECH
GA 可用性和 SLA	正式運作	個人預覽版, 無 SLA	個人預覽版, 無 SLA
語音辨識和文字轉語音的效能	標準	標準	高效能
整合式 OAuth	yes	是	否
整合式遙測	yes	是	否
支援舊版網頁瀏覽器	yes	否	否
Bot Framework SDK 支援	所有 v3、v4	需要 v4.5+	需要 v4.5+
用戶端 SDK 支援	JS、C#	JS、C#	C++、C#、Unity
與網路聊天搭配使用	yes	是	否
對話記錄 API	yes	是	否
VNET	否	預覽*	否

* Direct Line App Service 擴充功能可用在 VNET 中，但還不允許限制輸出呼叫。

其他資源

- [將 Bot 連線至 Direct Line](#)
- [將 Bot 連線至 Direct Line Speech](#)

將 Bot 連線至直接線路

2019/2/28 • [Edit Online](#)

您可以使用直接線路通道，讓用戶端應用程式與 Bot 進行通訊。

新增直接線路通道

若要新增直接線路通道，請在 [Azure 入口網站](#) 中開啟 Bot，並按一下 [通道] 刀鋒視窗，然後按一下 [直接線路]。

Connect to channels

Name	Health	Published	
 Skype	Running	--	Edit
 Web Chat	Running	--	Edit

[Get bot embed codes](#)

Add a featured channel



新增網站

接下來，新增網站以代表您想要連接到 bot 的用戶端應用程式。按一下 [新增網站]，輸入網站名稱，然後按一下 [完成]。

Configure Direct Line



+ Add new site Default Site [Edit](#) Disable | 

Default Site Secret keys

管理祕密金鑰

當您建立網站時，Bot Framework 會產生祕密金鑰，用戶端應用程式可藉此來驗證其所發出的直接線路 API 要求，以便與 bot 進行通訊。若要檢視純文字金鑰，按一下 [顯示] 以取得對應的索引鍵。

Configure Direct Line



+ Add new site My New Site Disable |

Default Site

My New Site

Secret keys

XXXXXXXXXXXXXXXXXXXXXX
Show | Regenerate

XXXXXXXXXXXXXXXXXXXXXX
Show | Regenerate

Version
Select which versions of the Direct Line protocol are enabled on this site. More information about these versions can be found in the [Direct Line reference documentation](#).

1.1
 3.0

Enhanced authentication options
For bots using [Azure Bot Service authentication](#), enable tamper-proof user IDs and the ability to specify trusted client hosts. Learn more about enhanced Direct Line [authentication features](#).

複製並安全地儲存顯示的金鑰。然後使用此金鑰來驗證您的用戶端發出的直接線路 API 要求，以便與 Bot 進行通訊。或者，使用直接線路 API [將金鑰交換成權杖](#)，您的用戶端便可用來驗證單一交談範圍內的後續要求。

Configure Direct Line



+ Add new site My New Site Disable |

Default Site

My New Site

Secret keys

OsFg08kJPik.cwA.1G4.BygO0efVi0PCtKJ047nfFc-4j4l0vJ2Mp9mjMUniBH
Show | Regenerate

XXXXXXXXXXXXXXXXXXXXXX
Show | Regenerate

Version
Select which versions of the Direct Line protocol are enabled on this site. More information about these versions can be found in the [Direct Line reference documentation](#).

1.1
 3.0

Enhanced authentication options
For bots using [Azure Bot Service authentication](#), enable tamper-proof user IDs and the ability to specify trusted client hosts. Learn more about enhanced Direct Line [authentication features](#).

配置設定

最後，配置網站的設定。

- 選取您的用戶端應用程式會用來與 Bot 進行通訊的直接線路通訊協定版本。

TIP

如果要在用戶端應用程式與 Bot 之間建立新連線，請使用 直接線路 API 3.0。

完成後，按一下 [完成] 以儲存網站設定。您可以重複此程序，從為您想要連線至 Bot 的每個用戶端應用程式的 [新增網站] 開始。

將 Bot 連線至 Direct Line Speech (預覽)

2019/5/14 • [Edit Online](#)

適用於:  SDK v4  SDK v3

您可以設定 Bot，允許用戶端應用程式透過 Direct Line Speech 通道與其通訊。

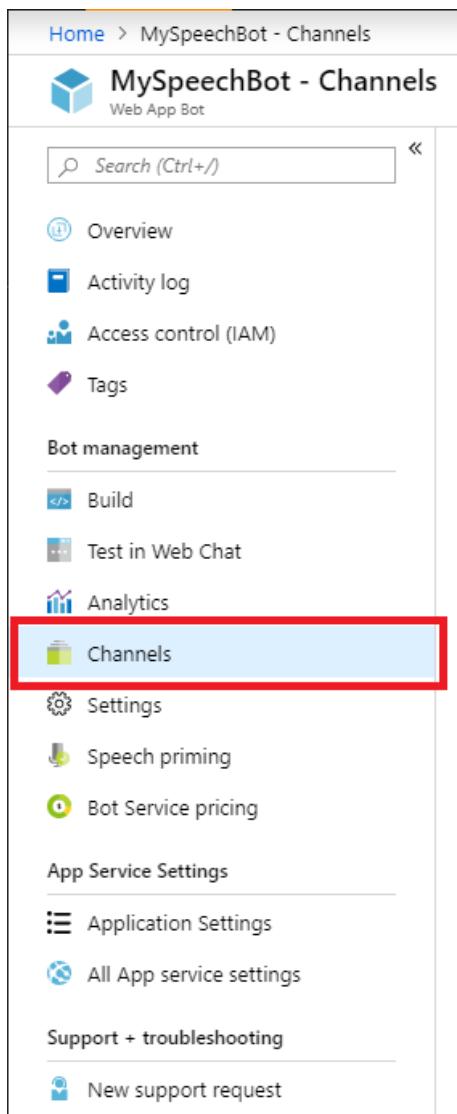
建置好您的 Bot 後，透過 Direct Line Speech 上線，將可使用 [Speech SDK](#) 與用戶端應用程式進行低延遲、高可靠性連線。這些連線最適合用於語音輸入、語音輸出的交談式體驗。如需有關 Direct Line Speech 以及如何建置用戶端應用程式的詳細資訊，請瀏覽 [自訂語音優先虛擬助理](#) 頁面。

註冊 Direct Line Speech 預覽版

Direct Line Speech 目前為預覽版，而且需要在 [Azure 入口網站](#) 中快速註冊。請參閱下面的詳細資料。核准後，您就可以存取通道。

新增 Direct Line Speech 通道

- 若要新增 Direct Line Speech 通道，請先在 [Azure 入口網站](#) 中開啟 Bot，然後按一下設定刀鋒視窗中的 [通道]。



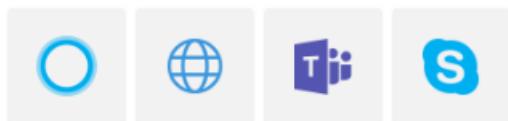
The screenshot shows the Azure Bot Channels page for a bot named 'MySpeechBot'. The left sidebar contains several navigation links: Overview, Activity log, Access control (IAM), Tags, Bot management (Build, Test in Web Chat, Analytics), Channels (which is highlighted with a red box), Settings, Speech priming, and Bot Service pricing. Below these are sections for App Service Settings (Application Settings, All App service settings) and Support + troubleshooting (New support request).

- 在通道選取頁面中，尋找並按一下 **Direct Line Speech** 以選擇通道。

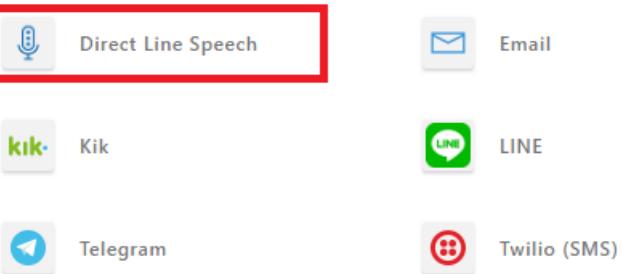
Connect to channels

Name	Health
 Web Chat	Running

Add a featured channel



More channels



3. 如果您尚未獲准存取，就會看到要求存取的頁面。填入必要的資訊，然後按一下 [要求]。將會出現確認頁面。當您的要求核准擋置時，就無法超越此頁面。
4. 一旦獲准存取，就會顯示 Direct Line Speech 的設定頁面。在您檢閱使用規定後，按一下 **Save** 確認您的通道選取項目。



啟用 Bot Framework 通訊協定串流擴充功能

在 Direct Line Speech 通道連線到 Bot 的情況下，您現在需要啟用 Bot Framework 通訊協定串流擴充功能支援，以獲得最佳、低延遲的互動。

1. 如果您還沒這麼做，請在 [Azure 入口網站](#) 中開啟您 Bot 的刀鋒視窗。
2. 按一下 [Bot 管理] 類別之下的 [設定] (在 [通道] 的正下方)。按一下 [啟用串流端點] 核取方塊。

The screenshot shows the Azure Bot Service settings interface. On the left, there's a sidebar with navigation links like Overview, Activity log, Access control (IAM), Tags, Build, Test in Web Chat, Analytics, Channels, Settings (which is selected), Speech priming, and Bot Service pricing. Below these are sections for App Service Settings (Configuration, All App service settings) and Support + troubleshooting (New support request). The main content area is titled 'Bot profile' and contains fields for Icon (with a placeholder 'Upload custom icon'), Display name (marked with a red asterisk), Bot handle, and Description. Under 'Configuration', there are fields for Messaging endpoint and Microsoft App ID (Manage). A checkbox for 'Enable Streaming Endpoint' is highlighted with a red border. At the top right, there are 'Save' and 'Discard' buttons.

3. 按一下頁面頂端的 [儲存]。
4. 在相同刀鋒視窗的 [App Service 設定] 類別之下，按一下 [設定]。

This screenshot shows the 'Configuration' section of the Azure App Service Settings blade. It includes a 'General settings' button, a 'Configuration' button (which is highlighted with a blue background), and an 'All App service settings' button. Below these are sections for 'Support + troubleshooting' (New support request) and 'Analytics'.

5. 按一下 `General settings`，然後選取可啟用 `Web socket` 支援的選項。

 Save  Discard

Application settings General settings * Default documents Path mappings

Stack settings

Stack .NET

.NET Framework version v4.7

Platform settings

Platform 32 Bit

Managed pipeline version Integrated

FTP state All allowed  FTP based

HTTP version 1.1

Web sockets On Off

Always on

 On  Off

 Prevent's your app from being idle

6. 按一下設定頁面頂端的  Save。

7. Bot Framework 通訊協定串流擴充功能現已針對您的 Bot 啟用。您現在已準備好更新 Bot 程式碼並[整合串流擴充功能支援](#)至現有的 Bot 專案。

管理祕密金鑰

用戶端應用程式需要有通道祕密，才能透過 Direct Line Speech 通道連線到您的 Bot。在您儲存通道選取項目後，即可從 Azure 入口網站中的設定 **Direct Line Speech** 頁面擷取這些祕密金鑰。

Configure Direct Line Speech



Secret keys

XXXXXXXXXXXXXXXXXXXXXXXXXXXX
[Show](#) | [Regenerate](#)

XXXXXXXXXXXXXXXXXXXXXXXXXXXX
[Show](#) | [Regenerate](#)

將通訊協定支援新增至您的 Bot

連線 Direct Line Speech 通道並啟用 Bot Framework 通訊協定串流擴充功能支援後，只要將程式碼新增至您的 Bot 來支援最佳化的通訊。遵循[將串流擴充功能支援新增至您的 Bot](#)上的指示，以確保與 Direct Line Speech 完全相容。

已知問題

請注意，此服務目前為預覽版且可能隨時變更，這可能會影響您的 Bot 開發和整體效能。以下是已知問題清單：

1. 此服務目前已部署至 [Azure 區域](#)美國西部 2。我們很快就會推展到其他區域，讓所有客戶都能獲得與其 Bot 進行低延遲語音互動的好處。
2. 即將推出控制欄位 (例如 [serviceUrl](#)) 的次要變更
3. 用來示意交談開始與結束的 [conversationUpdate](#) 和 [endOfCoversation](#) 活動 (常用於產生歡迎訊息)，將會更新來與其他通道達到一致
4. 通道尚未支援 [SigninCard](#)

將 Bot 連結到 Office 365 電子郵件

2019/5/10 • [Edit Online](#)

除了其他[通道](#)之外，Bot 還可透過 Office 365 電子郵件與使用者通訊。當 Bot 設定為存取電子郵件帳戶時，它會在新的電子郵件送達時收到訊息。然後，Bot 可以按照其商務邏輯進行回應。例如，Bot 可以傳送電子郵件回覆，確認收到的電子郵件中有訊息「嗨！感謝您的訂單！我們將立即處理。」

WARNING

建立 "spambot" 違反 [Bot Framework 管理辦法](#)，包括傳送不需要或未經請求的大量電子郵件的 Bot。

設定電子郵件認證

您可以透過在電子郵件通道設定中輸入 Office 365 認證，將 Bot 連結到電子郵件通道。不支援使用取代 AAD 的任何供應商進行同盟驗證。

NOTE

您自己的個人電子郵件帳戶不應該用於 Bot，因為傳送至電子郵件帳戶的每個訊息都將轉送給 Bot。這可能導致 Bot 不當地向寄件者傳送回應。因此，Bot 應該只使用專用的 O365 電子郵件帳戶。

若要新增電子郵件通道，請在 [Azure 入口網站](#) 中開啟 Bot，按一下 [通道] 刀鋒視窗，然後按一下 [電子郵件]。輸入有效的電子郵件認證，然後按一下 [儲存]。

Enter your Email credentials

[How do I connect my bot to Office 365 email?](#)

Email Address

address@email.com

Email Password

Password

電子郵件通道目前僅適用於 Office 365。請注意，目前不支援電子郵件服務。

自訂電子郵件

電子郵件通道支援使用 `channelData` 屬性傳送自訂的屬性，以建立更進階的自訂電子郵件。

屬性	說明
htmlBody	用於訊息本文的 HTML。
主旨	用於訊息的主旨。
importance	用於訊息的重要性旗標： <code>low</code> 、 <code>normal</code> 或 <code>high</code> 。

屬性	說明
toRecipients	以分號 (;) 分隔的電子郵件地址字串，用來新增至訊息的 [收件者] 欄位。
ccRecipients	以分號 (;) 分隔的電子郵件地址字串，用來新增至訊息的 [Cc](副本)欄位。
bccRecipients	以分號 (;) 分隔的電子郵件地址字串，用來新增至訊息的 [Bcc] (密件副本)欄位。

下列範例訊息顯示一個包含這些 `channelData` 屬性的 JSON 檔案。

```
{
  "type": "message",
  "locale": "en-US",
  "channelID": "email",
  "from": { "id": "mybot@mydomain.com", "name": "My bot" },
  "recipient": { "id": "joe@otherdomain.com", "name": "Joe Doe" },
  "conversation": { "id": "123123123123", "topic": "awesome chat" },
  "channelData":
  {
    "htmlBody": "<html><body style = /"font-family: Calibri; font-size: 11pt;/" >This is more than awesome.
</body></html>",
    "subject": "Super awesome message subject",
    "importance": "high",
    "ccRecipients": "Yasemin@adatum.com;Temel@adventure-works.com"
  }
}
```

如需有關使用 `channelData` 的詳細資訊，請參閱 [Node.js 範例](#) 或 [.NET 文件](#)。

如需有關使用 `channelData` 的詳細資訊，請參閱[如何實作通道專屬功能](#)。

其他考量

如果 Bot 未在 15 秒內傳回 200 OK HTTP 狀態碼以回應傳入的電子郵件訊息，電子郵件通道將會嘗試重新傳送訊息，因此 Bot 可能會收到相同的電子郵件訊息活動好幾次。如需詳細資訊，請參閱 **Bot 的運作方式** 中的 [HTTP 詳細資料](#)一節，以及如何[未逾時錯誤進行疑難排解](#)一文。

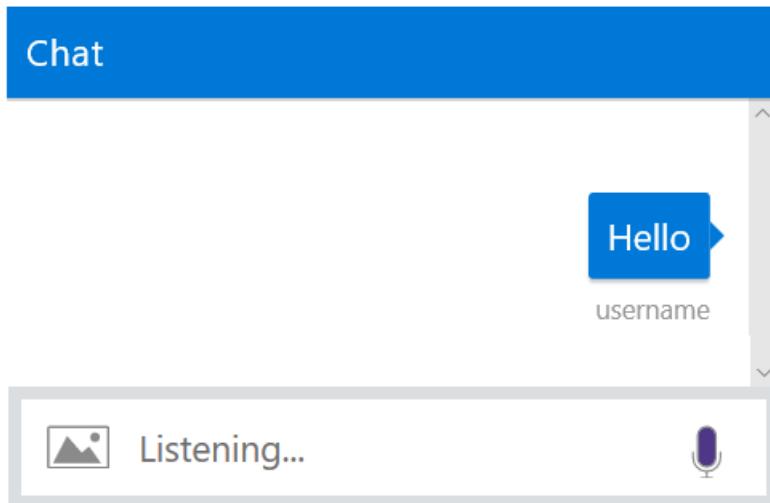
其他資源

- 將 Bot 連結到[通道](#)
- 使用適用於 .NET 的 Bot Framework SDK [實作通道特定功能](#)
- 使用[通道偵測器](#)查看通道如何轉譯 Bot 應用程式的特定功能
- 將 Bot 連結到[通道](#)
- 使用適用於 .NET 的 Bot Framework SDK [實作通道特定功能](#)
- 使用[通道偵測器](#)查看通道如何轉譯 Bot 應用程式的特定功能

啟用網路聊天中的語音

2019/2/28 • [Edit Online](#)

您可以啟用 Web 聊天控制項中的語音介面。使用者可透過使用 Web 聊天控制項中的麥克風，與語音介面進行互動。



如果使用者輸入而不是說出回應，則 Web 聊天將關閉語音功能，並且 Bot 僅提供文字回應而不是大聲朗讀。若要重新啟用語音回應，使用者可以在下一次使用麥克風回應 Bot。如果麥克風正在接受輸入，則它會顯示暗色或填滿。如果它顯示為灰色，則使用者可按一下它加以啟用。

必要條件

在執行範例之前，您需要針對要使用 Web 聊天控制項執行之 Bot 提供直接線路祕密或驗證權杖。

- 如需取得與 Bot 相關聯之直接線路祕密的相關資訊，請參閱[將機器人連線至直接線路](#)。
- 如需交換權杖祕密的詳細資訊，請參閱[產生直接線路權杖](#)。

自訂 Web 聊天的語音

若要在 Web 聊天中啟用語音功能，您需要自訂叫用 Web 聊天控制項的 JavaScript 程式碼。您可以使用下列步驟，在本機嘗試啟用語音的 Web 聊天。

- 下載範例 [index.html](#)。
- 根據要新增的語音支援類型，編輯 `index.html` 中的程式碼。[啟用語音服務](#)中描述了語音實作的類型。
- 啟動 Web 伺服器。一種方法是在 Node.js 命令提示字元中使用 `npm http-server`。
 - 若要全域安裝 `http-server` 以便可以從命令列執行它，請執行下列命令：

```
npm install http-server -g
```

- 若要啟動使用連接埠 8000 的 Web 伺服器，請從包含 `index.html` 的目錄執行下列命令：

```
http-server -p 8000
```

4. 將您的瀏覽器目標設為 `http://localhost:8000/samples?parameters`。例如，

`http://localhost:8000/samples?s=YOURDIRECTLINESECRET` 使用直接線路祕密叫用 Bot。下表說明可以在查詢字串中設定的參數：

參數	說明
s	直接線路祕密。如需取得直接線路祕密的相關資訊，請參閱 將機器人連線至直接線路 。
t	直接線路權杖。如需如何產生此權杖的相關資訊，請參閱 產生直接線路權杖 。
網域	選用。替代直接線路端點的 URL。
WebSocket	選用。設定為 'true' 以使用 WebSocket 來接收訊息。預設值為 <code>false</code> 。
userid	選用。Bot 使用者的識別碼。
username	選用。Bot 使用者的使用者名稱。
botid	選用。Bot 的識別碼。
botname	選用。Bot 的名稱。

啟用語音服務

自訂可讓您以下列任何方式新增語音功能：

- **瀏覽器提供的語音** - 使用網頁瀏覽器內建的語音功能。目前，此功能僅提供 Chrome 瀏覽器使用。
- **建立自訂語音服務** - 您可以建立自己的自訂語音識別和語音合成元件。

瀏覽器提供的語音

下列程式碼具現化瀏覽器隨附的語音辨識器和語音合成元件。並非所有瀏覽器都支援這種新增語音的方法。

NOTE

Google Chrome 支援瀏覽器語音辨識器。但是，在下列情況中，Chrome 可能會封鎖麥克風：

- 如果包含 Web 聊天之頁面的 URL 以 `http://` 開頭而不是 `https://`。
- 如果 URL 是使用 `file://` 通訊協定而不是 `http://localhost:8000` 的本機檔案。

```
const speechOptions = {
    speechRecognizer: new BotChat.Speech.BrowserSpeechRecognizer(),
    speechSynthesizer: new BotChat.Speech.BrowserSpeechSynthesizer()
};
```

自訂語音服務

您也可以提供您自己的自訂語音辨識（實作 `ISpeechRecognizer`）或語音合成（實作 `ISpeechSynthesis`）。

```
const speechOptions = {
    speechRecognizer: new YourOwnSpeechRecognizer(),
    speechSynthesizer: new YourOwnSpeechSynthesizer()
};
```

將語音選項傳遞至 Web 聊天

下列程式碼會將語音選項傳遞至 Web 聊天控制項：

```
BotChat.App({
    bot,
    locale: params['locale'],
    resize: 'detect',
    // sendTyping: true,      // defaults to false. set to true to send 'typing' activities to bot (and other
    users) when user is typing
    speechOptions: speechOptions,
    user,
    directLine: {
        domain: params['domain'],
        secret: params['s'],
        token: params['t'],
        webSocket: params['webSocket'] && params['webSocket'] === 'true' // defaults to true
    }
}, document.getElementById('BotChatGoesHere'));
```

後續步驟

現在您可以透過 Web 聊天啟用語音互動，了解您的 Bot 如何建構語音訊息並調整麥克風的狀態：

- [將語音新增至訊息 \(C#\)](#)
- [將語音新增至訊息 \(Node.js\)](#)

其他資源

- 您可以在 GitHub 上[下載 Web 聊天控制項的原始程式碼](#)。
- [Bing 語音 API 文件](#)提供有關 Bing 語音 API 的詳細資訊。

將 Bot 連線到 Facebook

2019/4/9 • [Edit Online](#)

您的 Bot 可以連線到 Facebook Messenger 和 Facebook Workplace，以便與這兩個平台上的使用者進行通訊。下列教學課程逐步示範如何將 Bot 連線到這兩個通道。

NOTE

Facebook UI 可能會因為您所使用的版本而略有不同。

將 Bot 連線到 Facebook Messenger

若要深入了解如何針對 Facebook Messenger 進行開發，請參閱 [Messenger 平台文件](#)。您可以檢閱 Facebook 的[啟動前指導方針](#)、[快速入門](#)和[設定指南](#)。

若要設定 Bot 使用 Facebook Messenger 進行通訊，請在 Facebook 頁面上啟用 Facebook Messenger，然後將 Bot 連線到應用程式。

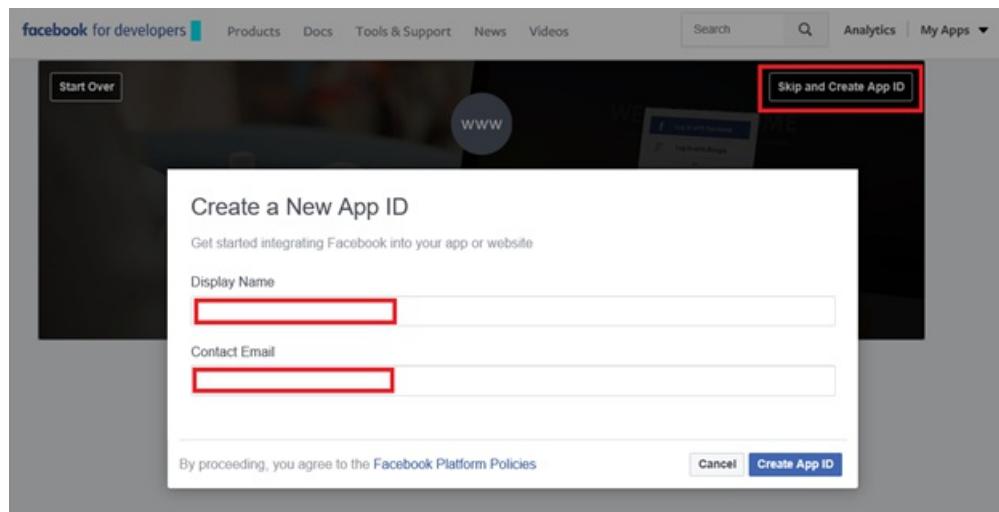
複製頁面識別碼

已透過 Facebook 頁面存取 Bot。請[建立新的 Facebook 頁面](#)或移至現有的頁面。

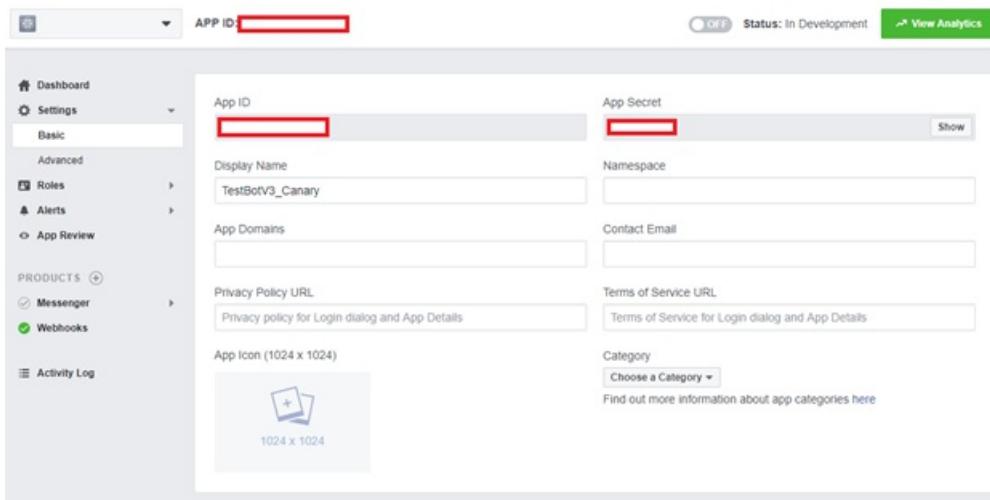
- 請開啟 Facebook 頁面的 [關於] 頁面，然後複製並儲存 [頁面識別碼]。

建立 Facebook 應用程式

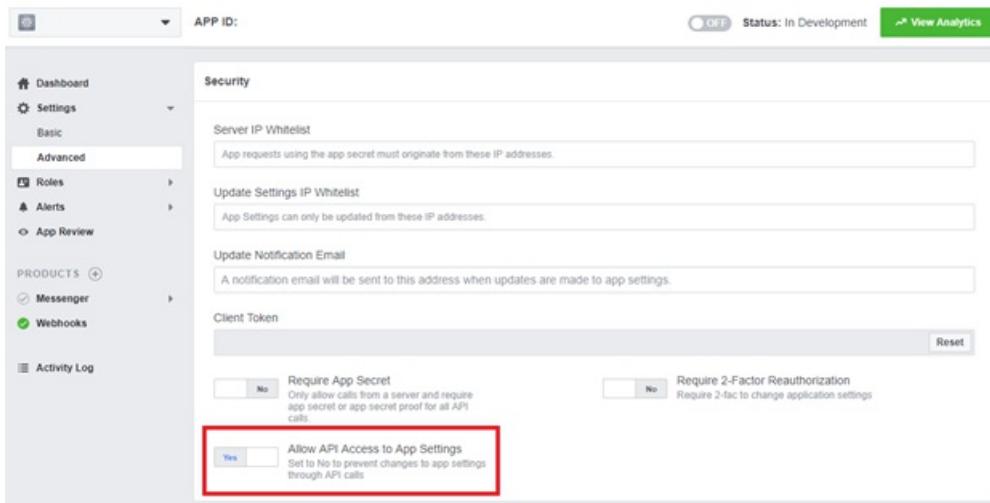
請在頁面上[建立新的 Facebook 應用程式](#)，然後為該應用程式產生應用程式識別碼和應用程式祕密。



- 複製並儲存 [應用程式識別碼] 和 [應用程式祕密]。

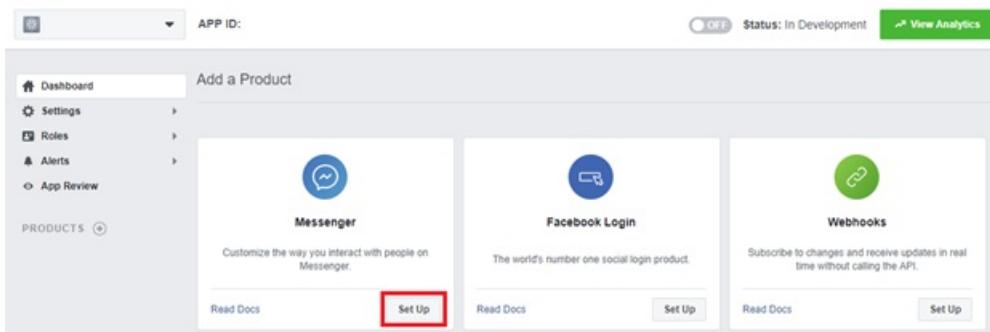


將 [允許 API 存取應用程式設定] 滑桿設為 [是]。



啟用 Messenger

在新的 Facebook 應用程式中啟用 Facebook Messenger。



產生頁面存取權杖

在 Messenger 區段的 [權杖產生] 面板中，選取目標頁面。[頁面存取權杖] 隨即產生。

- 複製並儲存 [頁面存取權杖]。

APP ID: Status: In Development [View Analytics](#)

Dashboard Settings Roles Alerts App Review PRODUCTS + Messenger Settings Activity Log

Token Generation

Page token is required to start using the APIs. This page token will have all messenger permissions even if your app is not approved to use them yet, though in this case you will be able to message only app admins. You can also generate page tokens for the pages you don't own using Facebook Login.

Page **Page Access Token**

Select a Page You must select a Page to generate an access token. Create a new page

Webhooks [Setup Webhooks](#)

To receive messages and other events sent by Messenger users, the app should enable webhooks integration.

啟用 Webhook

按一下 [設定 Webhook], 將訊息事件從 Facebook Messenger 轉送到 Bot。

Webhooks [Setup Webhooks](#)

To receive messages and other events sent by Messenger users, the app should enable webhooks integration.

提供 Webhook 回呼 URL 並確認權杖

在 [Azure 入口網站](#) 中開啟 Bot, 按一下 [通道] 索引標籤, 然後按一下 [Facebook Messenger]。

- 從入口網站複製 [回呼 URL] 和 [確認權杖] 值。

Callback URL and Verify Token for Facebook

What do I do with my Callback URL and Verify token?

Callback URL (Copy and paste in Facebook)

Verify Token (Copy and paste in Facebook)

- 返回 Facebook Messenger 並且貼上 [回呼 URL] 和 [確認權杖] 值。
- 在 [訂用帳戶欄位] 底下, 選取 [message_deliveries]、[messages]、[messaging_options] 及 [messaging_postbacks]。
- 按一下 [確認並儲存]。

APP ID: Status: In Development [View Analytics](#)

Dashboard Settings Roles Alerts App Review PRODUCTS + Messenger Settings Activity Log

Webhooks [Setup Webhooks](#)

New Page Subscription

Callback URL Validation requests and Webhook notifications for this object will be sent to this URL.

Verify Token Token that Facebook will echo back to you as part of callback URL verification.

Subscription Fields

<input checked="" type="checkbox"/> messages	<input checked="" type="checkbox"/> messaging_postbacks	<input checked="" type="checkbox"/> messaging_options
<input checked="" type="checkbox"/> message_deliveries	<input type="checkbox"/> message_reads	<input type="checkbox"/> messaging_payments
<input type="checkbox"/> messaging_pre_checkouts	<input type="checkbox"/> messaging_checkout_updates	<input type="checkbox"/> messaging_account_linking
<input type="checkbox"/> messaging_referrals	<input type="checkbox"/> message_echoes	<input type="checkbox"/> messaging_game_plays
<input type="checkbox"/> standby	<input type="checkbox"/> messaging_handovers	<input type="checkbox"/> messaging_policy_enforcement

[Learn more](#)

[Cancel](#) [Verify and Save](#)

Enables your app to send and receive messages using a Facebook Page.

4. 將 Webhook 訂閱至 Facebook 頁面。

To receive messages and other events sent by Messenger users, the app should enable webhooks integration.

Selected events: messages, messaging_postbacks, messaging_optins, message_deliveries

Select a page to subscribe your webhook to the page events

page page Unsubscribe

✓ Complete

提供 Facebook 證認

在 Azure 入口網站中，貼上先前從 Facebook Messenger 複製的 [Facebook 應用程式識別碼]、[Facebook 應用程式祕密]、[頁面識別碼] 及 [頁面存取權杖] 值。您可以新增額外的頁面識別碼和存取權杖，以便在多個 Facebook 頁面上使用相同的 Bot。

Overview
Activity log
Access control (IAM)
Tags

BOT MANAGEMENT
Build
Test in Web Chat
Analytics
Channels
Settings
Speech priming

Enter your Facebook Messenger credentials
[Step-by-step instructions to add the bot to Facebook Messenger.](#)

Facebook App ID
Facebook app ID can be found in your Facebook account settings

Facebook App Secret
Facebook app secret can be found in your Facebook account settings

Page ID
Page ID can be found in your Facebook account settings

Page Access Token
Access Token can be found in your Facebook account settings

+ Add a new page

提交以供審查

Facebook 需要它基本應用程式設定頁面上的 [隱私權原則 URL] 和 [服務條款 URL]。管理辦法頁面包含第三方資源連結，可以協助建立隱私權原則。使用條款頁面包含範例條款，可以協助建立適當的「服務條款」文件。

Bot 完成之後，Facebook 對於發佈到 Messenger 的應用程式有自己的審查程序。系統會測試 Bot 以確保它符合 Facebook 的平台原則規範。

將應用程式設為公用並發佈至頁面

NOTE

在應用程式發佈之前，它都處於開發模式。外掛程式和 API 功能僅適用於系統管理員、開發人員和測試人員。

審查成功之後，請在 [應用程式審查] 底下的 [應用程式儀表板] 中，將應用程式設為 [公用]。確保與這個 Bot 相關聯的 Facebook 頁面已發佈。狀態會顯示在頁面設定中。

將 Bot 連線到 Facebook Workplace

如需 Facebook Workplace 開發的指導方針，請參閱 [Workplace 說明中心](#) 以深入了解 Facebook Workplace 和 [Workplace 開發人員文件](#)。

若要將 Bot 設定為使用 Facebook Workplace 進行通訊，請建立自訂整合並將 Bot 連線到該整合。

建立 Facebook Workplace Premium 帳戶

遵循 [這裡](#) 的指示，建立 Facebook Workplace Premium 帳戶並將自己設定為系統管理員。請記住，只有 Workplace 的系統管理員可以建立自訂整合。

建立自訂整合

當您建立自訂整合時，只會建立您的 Workplace 社群內可見且已定義權限的應用程式和 'Bot' 類型的頁面。

遵循下列步驟，為您的 Workplace 建立**自訂整合**：

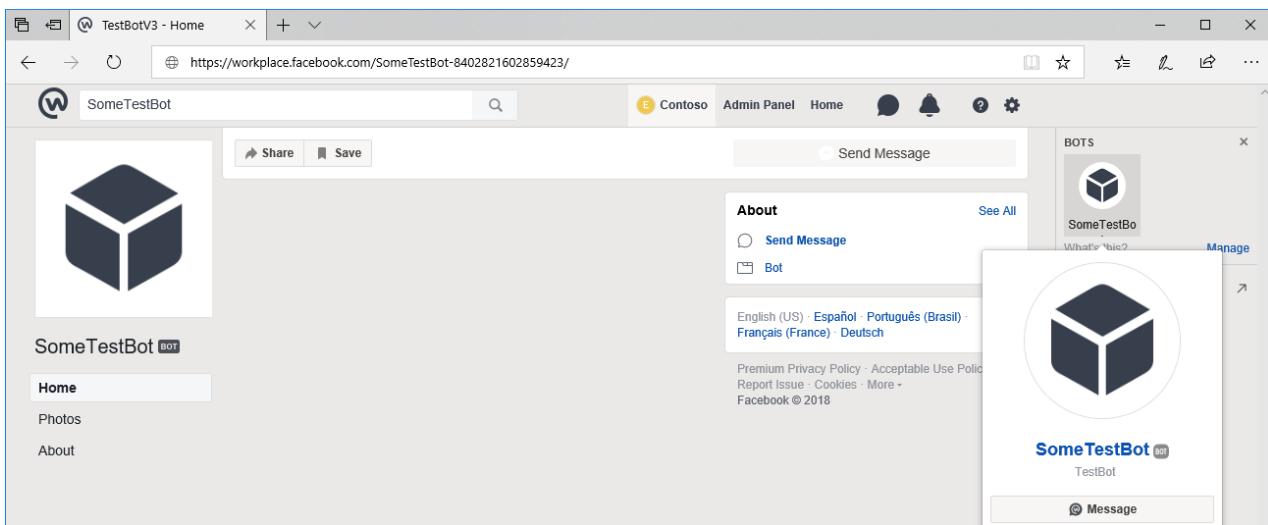
- 在 [管理面板] 中，開啟 [整合] 索引標籤。
- 按一下 [建立自己的自訂應用程式] 按鈕。

The screenshot shows the Workplace Admin Panel. On the left, there's a sidebar with various management options like Setup, Reporting, People, Groups, and Integrations. The Integrations option is highlighted with a red box. The main content area has a heading 'Integrations' with a sub-instruction 'Build integrations that extend the functionality of Workplace using APIs.' Below this, there are sections for 'Added to Workplace' and 'Added By Your Company'. Under 'Added to Workplace', it says 'No Integrations Installed Yet' and provides a link to 'Browse available integrations or create your own custom app.' At the bottom right of this section is a blue button labeled 'Create Custom Integration'. A specific integration named 'Cafe Bot' is listed under 'Custom Integrations', showing its status as 'Enabled'.

- 選擇應用程式的顯示名稱和設定檔圖片。這類資訊會與 'Bot' 類型的頁面共用。
- 將 [允許 API 存取應用程式設定] 設定為 [是]。
- 複製並安全地儲存您所看見的應用程式識別碼、應用程式祕密和應用程式權杖。

The screenshot shows the 'Edit Integration' dialog box over the Admin Panel. The dialog contains fields for 'Name' (set to 'Cafe Bot'), 'Update Logo' (with a placeholder image), 'Description' (set to 'Cafe Bot'), and 'App Enabled' (set to 'Yes'). To the right, there are fields for 'App ID' (37848932759751285) and 'API Version' (v3.1). Below these are fields for 'App Secret' (containing several asterisks) and 'Access Token' (with a 'Reset Access Token' button). A large red box highlights the 'App ID', 'App Secret', and 'Access Token' fields. At the bottom of the dialog is a 'Grant Permissions' section with several checkboxes for managing accounts, impersonating accounts, managing groups, mentioning bots, reading messages, and reading group content. A blue 'Create Custom Integration' button is visible at the bottom right of the dialog.

您現在已經建立好自訂整合。您可以在您的 Workplace 社群中尋找 'Bot' 類型的頁面，如下所示。



提供 Facebook 認證

在 Azure 入口網站中，貼上先前從 Facebook Workplace 複製的 [Facebook 應用程式識別碼]、[Facebook 應用程式祕密] 和 [頁面存取權杖] 值。請使用其 [關於] 頁面上整合名稱後面的數字，而不是傳統的 pageID。類似於將 Bot 連線至 Facebook Messenger，Webhook 可透過 Azure 中顯示的認證來連線。

提交以供審查

如需詳細資訊，請參閱**將 Bot 連線至 Facebook Messenger**一節和 [Workplace 開發人員文件](#)。

將應用程式設為公用並發佈至頁面

如需詳細資訊，請參閱**將 Bot 連線至 Facebook Messenger**一節。

設定 API 版本

如果您收到來自 Facebook 關於特定 Graph API 版本淘汰的通知，請移至 [Facebook 開發人員頁面](#)。瀏覽至 Bot 的 [應用程式設定] 並移至 [設定] > [進階] > [升級 API 版本]，然後將 [升級所有呼叫] 切換為 3.0。

The screenshot shows the Facebook for Developers dashboard with the 'Contoso Bot' selected. In the 'Advanced' settings section, there are two dropdown menus for upgrading API versions. The left dropdown is labeled 'Upgrade All Calls' and the right one is labeled 'Upgrade Calls for App Roles'. Both dropdowns are set to 'v3.0'. A red box highlights the 'Upgrade All Calls' dropdown, and another red box highlights the 'Upgrade Calls for App Roles' dropdown. An arrow points from the text 'Please use the API Upgrade Tool to understand how this might impact your app.' at the top of the page towards the 'Upgrade All Calls' dropdown.

範例程式碼

如需進一步參考，[Facebook-events](#) 範例 Bot 可用來探索 Bot 與 Facebook Messenger 的通訊。

將 Bot 連線至 GroupMe

2019/5/10 • [Edit Online](#)

您可以設定您的 Bot 使用 GroupMe 群組傳訊應用程式與其他人通訊。

TIP

若想了解各種 Bot Framework 功能的外觀及對此通道的運作方式，請[使用通道偵測器](#)。

註冊 GroupMe 帳戶

如果您沒有 GroupMe 帳戶，請註冊[免費新帳戶](#)。

建立 GroupMe 應用程式

為您的 Bot [建立 GroupMe 應用程式](#)。

使用這個回呼 URL: <https://groupme.botframework.com/Home/Login>

Create Application

Application Name

Callback URL

Callback URL must be https, localhost, or a deep link.

Developer Name

Developer Email

Developer Phone Number

Developer Company

Developer Address

I agree to abide by the [Terms of Use](#) and the [Brand Standards](#)

[Save](#)

[Cancel](#)

收集認證

- 在 [重新導向 URL] 欄位中，複製 **client_id=** 之後的值。

2. 複製 [存取權杖] 值。

{YOUR BOT NAME}

Details Settings Delete

Settings

Redirect URL https://oauth.groupme.com/oauth/authorize?client_id=Your Client Id

Callback URL https://groupme.botframework.com/Home/Login

Your Access Token

Use the access token string to authenticate as yourself when making API requests.

Access Token [Your Access Token](#)

提交認證

1. 在 dev.botframework.com, 將您剛剛複製的 **client_id** 值貼到 [用戶端識別碼] 欄位。
2. 將 [存取權杖] 值貼到 [存取權杖] 欄位。
3. 按一下 [檔案]。

Access Token

Access token can be found in your GroupMe account settings

Client ID

Client ID can be found in your GroupMe account settings

將 Bot 連線至 Kik

2019/2/28 • [Edit Online](#)

您可以使用 Kik 傳訊應用程式配置 Bot 以便與其他人通訊。

在手機上安裝 Kik

如果您並未在手機上安裝 Kik，則可以透過手機 App Store 或 [Kik 網站](#)安裝。您需要使用現有的 Kik 使用者帳戶或註冊新帳戶。

The screenshot shows the 'Sign Up' page of the Kik application. At the top, there is a back arrow icon and the text 'Sign Up'. On the left side, there is a circular profile picture placeholder with a 'Set Photo' button. Below it are fields for 'First Name' and 'Last Name'. Further down are fields for 'Username', 'Email', 'Password', 'Birthday', and 'Phone (Recommended)'. At the bottom, there is a checkbox labeled 'By checking this you agree to Kik's [Terms & Privacy Policy](#)' followed by a radio button. A large blue 'Sign Up' button is at the bottom center.

藉由行動電話登入開發人員入口網站

使用行動電話[登入 Kik入口網站](#)。若出現提示：在 "Kik" 中開啟在此頁面嗎？請選取 [開啟]。

Build Your Bot

First there were websites, then there were
apps.

Now, there are bots.



Scan To Create Your Bot

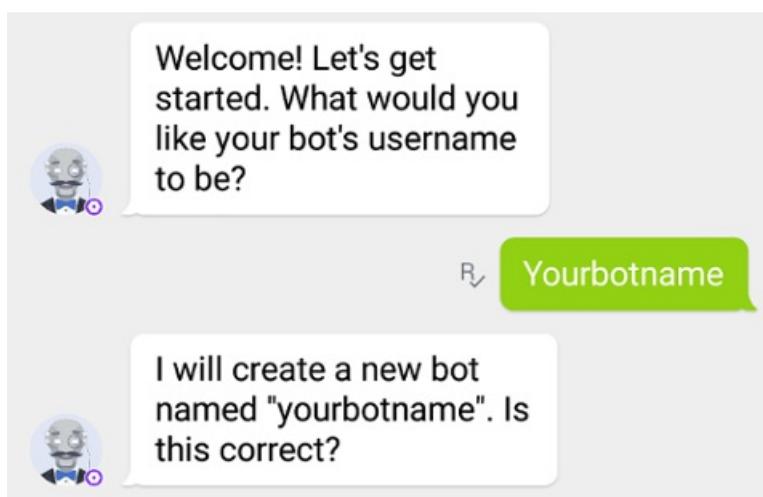
1. Open Kik 2. Scan Code

遵循 Bot 安裝程序

指定您的 Bot 名稱。



← Botsworth



Tap a message



Yes

No

收集認證

在 [設定] 索引標籤中，複製名稱和 API 金鑰。

The screenshot shows the kik Dev dashboard with a purple header containing links for Reporting, Configuration, Bot Shop Settings, Bot Shop, Docs, and a user profile icon.

Account Settings

- Display Name: yourbotname
- Admins:
 - Add an admin...
 - larslidén

API Key

API Key:	3f803309-0000-4000-8000-000000000000	Generated:	14:20 Apr 13, 2016	Regenerate
----------	--------------------------------------	------------	--------------------	------------

提交認證

Bot Name

API Key

Submit Kik Credentials

按一下 [送出 Kik 認證]。

啟用 Bot

勾選 [在 Kik 上啟用此 Bot]。然後按一下 [我已完成配置 Kik]。

當您完成這些步驟時，您的 Bot 即已成功配置為在 Kik 中與使用者進行通訊。

將 Bot 連線至 LINE

2019/3/5 • [Edit Online](#)

您可以設定讓 Bot 透過 LINE 應用程式與人通訊。

登入 LINE 主控台

透過使用 LINE 登入來登入 LINE 帳戶的 [LINE 開發人員主控台](#)。

NOTE

如果您還未[下載 LINE](#), 則請移至您的設定來註冊電子郵件地址。

註冊為開發人員

如果您是第一次登入 LINE 開發人員主控台, 請輸入名稱和電子郵件地址來建立開發人員帳戶。

The screenshot shows the 'Provider List' section of the LINE Developers Console. On the left, there's a sidebar with 'Welcome Username' and 'Providers' (which is expanded). Below 'Providers' is a message 'Has not provider'. Under 'Tools', there's a link. The main area has a heading 'Welcome to LINE Developers Console!' followed by a sub-heading 'Let's develop an app that connects people with people using your development technology and LINE Platform! A provider is a service provider (company / individual), and we begin by creating a provider.' Below this, there are three steps: 'STEP 1' (Create a new provider with a folder icon), 'STEP 2' (Create channels for LINE login and Messaging API with a folder and communication icons), and 'STEP 3' (Finish setting up the application on each channel and develop your own application with a smartphone icon). At the bottom, there's a 'Create New Provider' button.

建立新的提供者

請先建立 Bot 的提供者 (如果您還未設定的話)。提供者是提供應用程式的實體 (個人或公司)。

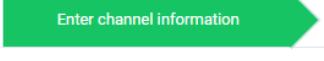
Welcome Username 

Providers 

Has not provider

Tools

Create new provider

Enter channel information 

Confirm 

Done 

Enter name of provider
The provider is the entity (individual or company) that offers the app.

Provider name

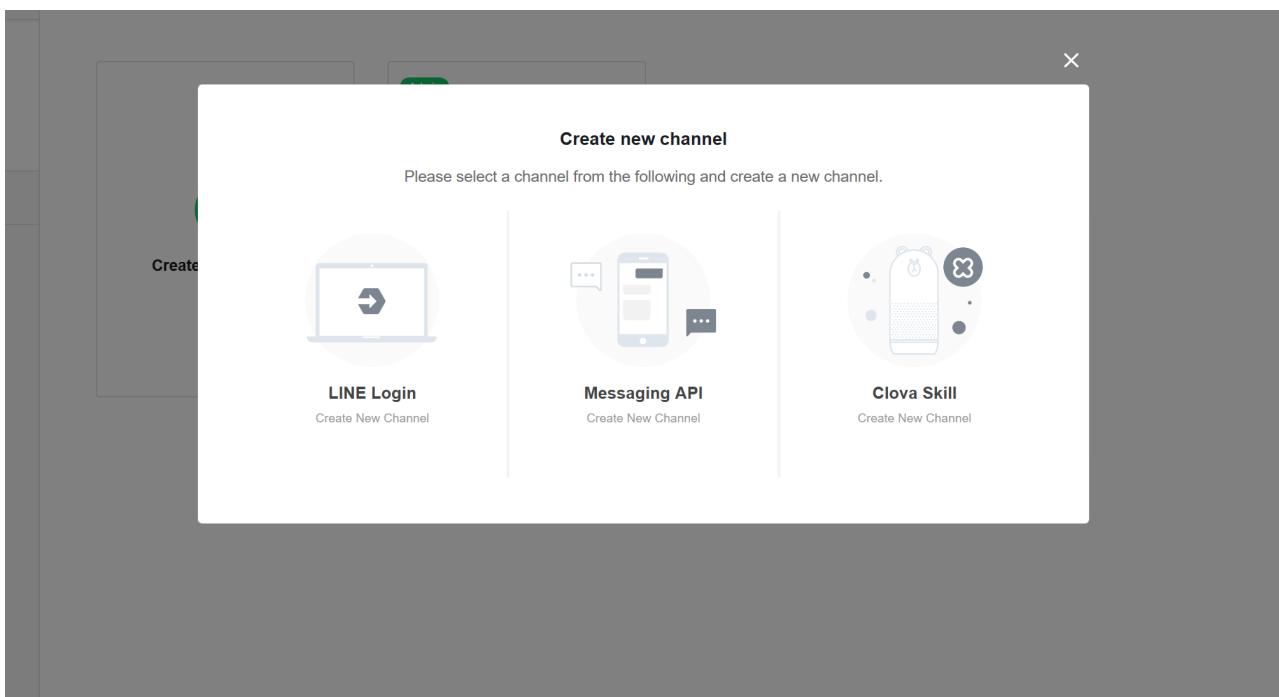
Contoso 

Max: 100 characters

Confirm 

建立傳訊 API 通道

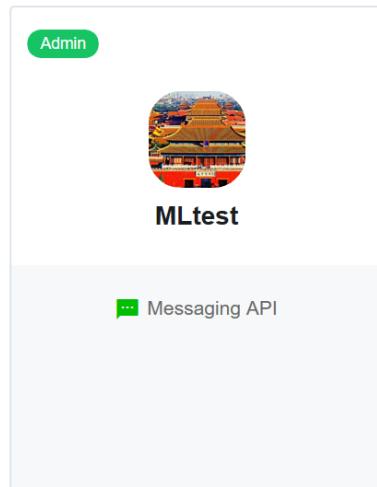
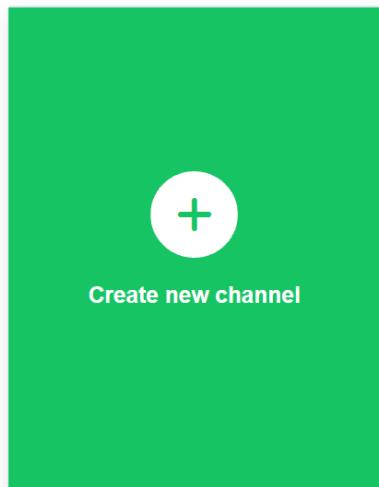
接下來，請建立新的傳訊 API 通道。



按一下綠色方塊來建立新的傳訊 API 通道。



ML

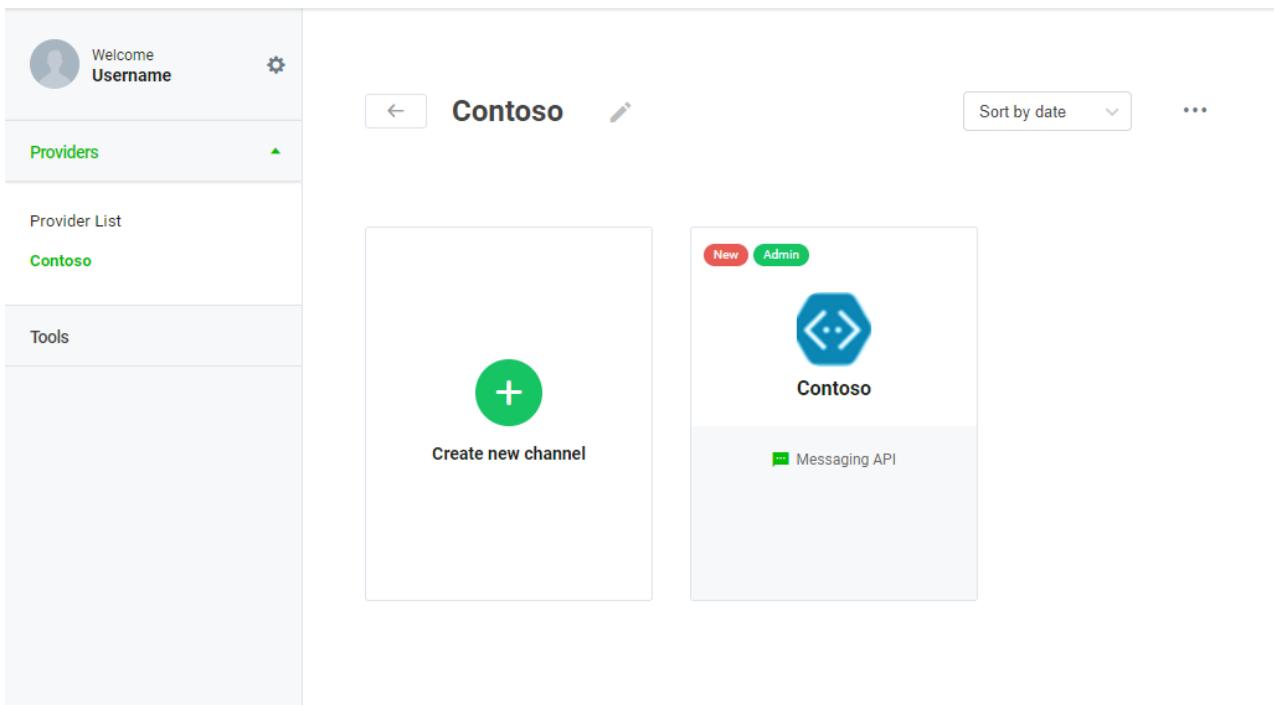


名稱中不得包含 "LINE" 或類似字串。填寫必要欄位，並確認通道的設定。

	<p>App name</p> <p>Enter app name</p> <p>Max: 20 characters</p> <p>Note: The app name cannot be changed for seven days.</p> <hr/> <p>App description</p> <p>Enter app description</p> <p>Max: 500 characters</p> <hr/> <p>Plan</p> <p><input checked="" type="radio"/> Developer Trial A trial plan which lets you create a bot that can send push messages and have up to 50 friends. Note: You cannot upgrade or buy a premium ID for a Developer Trial plan.</p> <p><input type="radio"/> Free A plan which lets you create a bot with an unlimited number of friends. Push messages cannot be sent with this plan. Note: You can upgrade this plan at any time.</p> <hr/> <table border="1"><tr><th style="text-align: left;">Category</th><th style="text-align: left;">Subcategory</th></tr><tr><td>Select category</td><td>Select subcategory</td></tr></table> <hr/> <p>Email address </p> <p>example@line.me</p> <p>Max: 100 characters</p>	Category	Subcategory	Select category	Select subcategory
Category	Subcategory				
Select category	Select subcategory				

從通道設定取得所需值

在確認通道的設定後，系統會將您導向如下所示的頁面。



Welcome Username

Providers

Provider List

Contoso

Tools

Contoso

New Admin

<::>

Contoso

Messaging API

Create new channel

按一下您所建立的通道來存取通道設定，然後向下捲動來找到 [基本資訊] > [通道祕密]。在某處儲存該資訊片刻。
確認 [可用功能] 包括 `PUSH_MESSAGE`。

Channel ID  1968854556

Channel secret  7e53a5215b0455fe6ca72e2926b5091d 

App type  BOT

Plan

For Developer 

To verify your plan after making a change, reload the page.

Available features  REPLY_MESSAGE PUSH_MESSAGE

Email address  bots-line-email@contoso.com 

然後，再往下捲動至 [傳訊設定]。您會在其中看到 [通道存取權杖] 欄位，並有 [核發] 按鈕。按一下該按鈕來取得存取權杖，並先儲存該權杖片刻。

Messaging settings

Channel access token (long-lived)  Pq/2emFE0qD0U8cyu5flvI1ZvHqD9CWLxqnqvwovhILO7ZmlXUGHMWmjsagtqXKsb9zSL08uwQP0432yKXwiliovgrihlhcc6867x+bHESVYMs5Vqvy5wMqD8bIN9iMYrd46srtSKgdB04t89/10/w1cDnyiFU= 

Use webhooks  Enabled 

Webhook URL Requires SSL  line.botframework.com/api/DR7rrK8EV0f  

Allow bot to join group chats  Enabled 

將 LINE 通道連線至 Azure Bot

登入 [Azure 入口網站](#)並尋找您的 Bot，然後按一下 [通道]。

The screenshot shows the Azure portal's 'Channels' page for the 'LineTestBot' bot. On the left, there's a sidebar with various service icons. The main area has a title 'Connect to channels' and a table showing two channels: 'Skype' (Running) and 'Web Chat' (Running). There are 'Edit' buttons for both. Below the table, a link 'Get bot embed codes' is visible. A section titled 'Add a featured channel' shows icons for Skype, Web Chat, and Microsoft Teams. Another section titled 'More channels' lists several other platforms: Twilio (SMS), Email, Facebook, GroupMe, Telegram, Kik, LINE, and Slack.

在其中選取 LINE 通道，然後將前面取得的通道密碼和存取權杖貼到適當欄位中。請務必儲存您所做的變更。

複製 Azure 提供給您的自訂 Webhook URL。

This screenshot shows the 'Configure LINE' step within the Azure portal. It features a 'Configure LINE' header with a LINE icon. Below it, there's a section for 'Enter your LINE credentials' with a 'Step-by-step instructions to add the bot to LINE.' link. Two input fields are shown: 'Channel Secret' containing '25888cd3008b4fd617ee9edd467dbbb4' and 'Channel Access Token' containing '65BEUzf6AOmPhiq3nY4ffhT5KSehQ4f7MUKtAvXV3F1//DeAiBSHAY34B,'. A note below says 'Settings to use in LINE configuration' and 'What do I do with my Callback URL and Verify Token?'. At the bottom, there are 'Cancel', 'Save', and 'Enabled' (which is checked) buttons, along with a 'Delete Channel' button.

設定 LINE 的 Webhook 設定

接下來，回到 LINE 開發人員主控台，將 Azure 提供給您的 Webhook URL 貼到 [訊息設定] > [Webhook URL]，然後按一下 [驗證] 來驗證連線。如果您剛剛才在 Azure 中建立通道，可能需要幾分鐘的時間才會生效。

然後，啟用 [訊息設定] > [使用 Webhook]。

IMPORTANT

在 LINE 開發人員主控台中，您必須先設定 Webhook URL，然後才能設定 [使用 Webhook = Enabled]。具有空白 URL 的第一個啟用 Webhook 不會設定 Enabled 狀態（即使 UI 這麼說也是一樣）。

在新增 Webhook URL 並啟用 Webhook 之後，請務必重新載入此頁面，並確認這些變更是否已正確設定。

Messaging settings

Channel access token (long-lived) [?](#)
Pq/2emFE0qD0U8cyu5fvlV1ZvHqD9CWLxqnqvwovhIL07ZmlXUGHMWmjsgtqXKsb9zSL08uwQP0432yKXwiliovgrihlthcc6867x+bHESVYVm5Vqy5wMqD8bIN9IMYrd46srTSKgdB04t89/10/w1cDrylFU=

[Issue](#)

Use webhooks [?](#)

Enabled Disabled

[Update](#) [Cancel](#)

Webhook URL [Requires SSL](#) [?](#)
line.botframework.com/api/DR7rrK8EV0f

[Verify](#) [Edit](#)

Allow bot to join group chats [?](#)

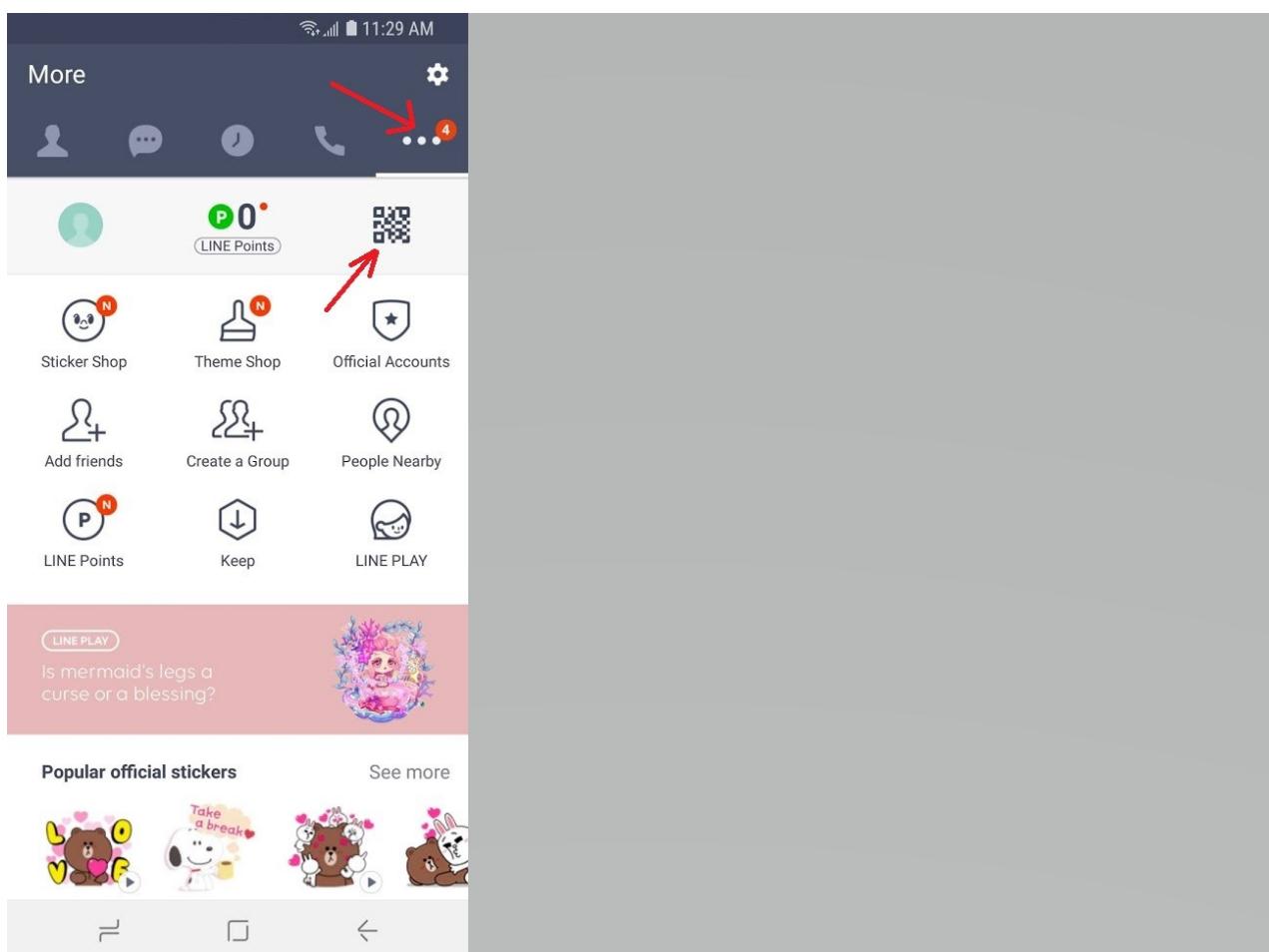
測試 Bot

在完成這些步驟後，您的 Bot 將成功設定為會在 LINE 上與使用者通訊，且已準備好接受測試。

將 Bot 新增至 LINE 行動應用程式

在 LINE 開發人員主控台中，瀏覽至 [設定] 頁面，此時您會看到 Bot 的 QR 代碼。

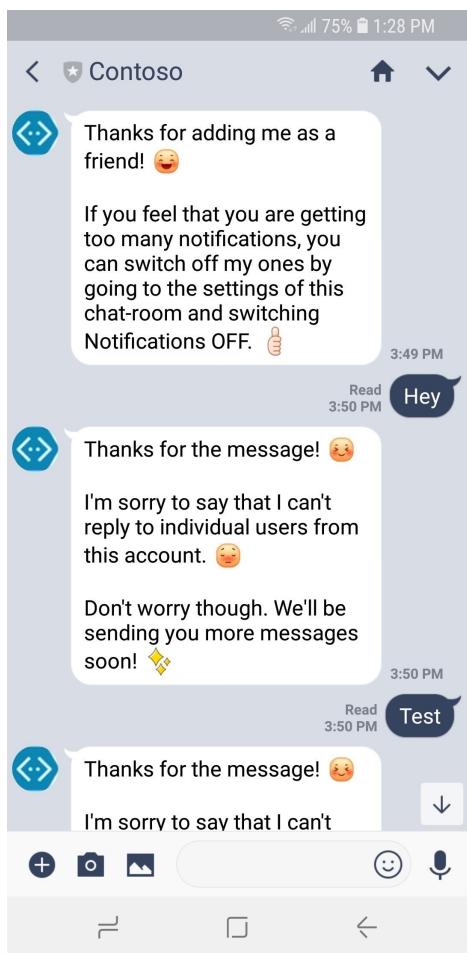
在 LINE 行動應用程式中，移至最右側有三個點 [...] 的瀏覽器索引標籤，然後點選 [QR 代碼] 圖示。



將 QR 代碼讀取器指向開發人員主控台中的 QR 代碼。您現在應該就能與 LINE 行動應用程式中的 Bot 互動，並測試 Bot 了。

自動訊息

當您開始測試 Bot 時，您可能會注意到 Bot 傳送的訊息與您在 `conversationUpdate` 活動中指定的訊息不符。您的對話方塊看起來可能像這樣：



若要避免傳送這些訊息，您必須關閉自動回應訊息。

Using LINE@ features

Message text for LINE@ features are set on the LINE@ Manager.

Auto-reply messages [?](#)

Enabled Disabled

[Update](#) [Cancel](#)

Greeting messages [?](#)

Disabled

[Set message](#) [Edit](#)

QR code of your bot

或者，您也可以選擇保留這些訊息。在此情況下，最好的辦法是按一下 [設定訊息]，並加以編輯。

The screenshot shows the LINE@ Manager interface. On the left, there's a sidebar with a green header 'LINE@ MANAGER'. Below it are sections for 'Compose message', 'Post to Timeline (Home)', and 'Message' (with sub-options like 'Message lists', 'Compose', 'Auto reply message', 'Keyword reply message', and 'Greeting message'). The main area is titled 'Greeting message' and contains a text editor with the placeholder text 'Thanks for adding me as a friend! (happy)'. It also includes a note about notifications and an emoji button. Below the text editor are buttons for 'Text', 'Sticker', 'Photo', 'Coupon', 'Prize drawing page', and 'Polls & Surveys'. A note at the bottom says 'You can send up to five messages at once. Please select the messages you would like to send.'

疑難排解

- 如果 Bot 完全沒有回應任何訊息，請在 Azure 入口網站中瀏覽至 Bot，然後選擇 [在網路聊天中測試]。
 - 如果 Bot 可在其中運作，在 LINE 中卻沒有回應，則請重新載入 LINE 開發人員主控台的頁面，並重複上述的 Webhook 指示。請務必先設定 **Webhook URL** 再啟用 Webhook。
 - 如果 Bot 在網路聊天中無法運作，請針對 Bot 的問題進行偵錯，然後再回來完成 LINE 通道的設定。

將 Bot 連線至 Skype

2019/2/28 • [Edit Online](#)

Skype 可讓您透過立即訊息、電話和視訊通話連接使用者。藉由建置使用者可以透過 Skype 介面探索與互動的 Bot 來擴充這項功能。

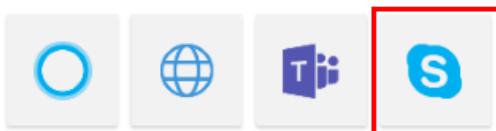
若要新增 Skype 通道，請在 [Azure 入口網站](#) 中開啟 Bot，按一下 [通道] 刀鋒視窗，然後按一下 [Skype]。

Connect to channels

Name	Health	Published	
 Web Chat	Running	--	Edit 

[Get bot embed codes](#)

Add a featured channel



系統會將您引導至 [配置 Skype] 設定頁面。

Configure



[Web control](#) [Messaging](#) [Calling](#) [Groups](#) [Publish](#)

Web control

The Web Control lets you embed the bot in your own website.

[Get embed code](#)

您需要在 [Web 控制項]、[傳訊]、[呼叫]、[群組] 和 [發佈] 中進行設定。讓我們逐一進行設定。

Web 控制項

若要將 Bot 嵌入您的網站，請按一下 [Web 控制項] 區段中的 [取得內嵌程式碼] 按鈕。這會將您導向 Skype for

Developers 頁面。請遵循那裡的指示來取得內嵌程式碼。

訊息

此區段會設定 Bot 如何傳送並接收 Skype 中的訊息。

呼叫

此區段會在 Bot 中配置 Skype 的呼叫功能。您可以指定是否要為 Bot 啟動 [呼叫]，以及啟用後是否要使用 IVR 功能或即時媒體功能。

群組

此區段會設定是否可以將 Bot 加入群組以及其在傳訊群組中的行為，並且也會用來為呼叫 Bot 啟用群組視訊通話。

發佈

此區段會配置 Bot 的發佈設定。所有標示「*」的欄位，皆為必填欄位。

在預覽中的 Bot 限制為 100 名連絡人。如果您需要 100 位以上的連絡人，請提交 Bot 進行審查。按一下 [提交以供審查] 會在接受後自動讓您的 Bot 可在 Skype 中搜尋。如果您的要求未通過核准，則會通知您需要變更才能通過核准的項目。

TIP

如果您想要提交 Bot 進行檢閱，請記住它必須先符合 [Skype 憑證檢查清單](#)，才會獲得接受。

完成設定之後，請按一下 [儲存] 並接受 [服務條款]。Skype 通道現在已新增至您的 Bot。

後續步驟

- [商務用 Skype](#)

將 Bot 連結到商務用 Skype

2019/4/9 • [Edit Online](#)

商務用 Skype Online 可讓您與同事和商務夥伴，透過立即訊息、電話和視訊通話連線。藉由建置使用者可以透過商務用 Skype 介面探索與互動的 Bot 來擴充此功能。

IMPORTANT

Bot Framework 中的商務用 Skype 通道即將在 2019 年 6 月 30 日淘汰。

在該日期之後，沒有任何新的 Bot 能夠新增商務用 Skype 通道。現有的 Bot 將繼續運作，直到 2019 年 10 月 31 日為止。Microsoft Teams 是 Microsoft 建議的通訊工具。了解如何[將 Bot 連線至 Microsoft Teams](#)。

啟用頻道

在 [Azure 入口網站](#)中開啟 Bot，按一下 [頻道] 刀鋒視窗，然後按一下 [商務用 Skype]。現在已啟用 Bot。

將 Bot 與商務用 Skype Online 連結，是由商務用 Skype 租用戶的租用戶系統管理員執行。

後續步驟

- [商務用 Skype Online Bot Framework](#)

將 Bot 連線至 Slack

2019/2/28 • [Edit Online](#)

您可以設定您的 Bot 使用 Slack 傳訊應用程式與其他人通訊。

為您的 Bot 建立 Slack 應用程式

登入 [Slack](#), 然後前往[建立 Slack 應用程式](#)通道。

The screenshot shows the 'Your Apps' section of the Slack developer portal. On the left, there's a sidebar with 'Start here' and 'App features' sections. The main area has a search bar and a table listing one application:

App Name	Workspace	Distribution Status
TestApp	BCCmsft	Not distributed

建立應用程式並指派「開發 Slack 小組」

輸入「應用程式名」稱並選取「開發 Slack 小組」。如果您還不是「開發 Slack 小組」的成員, 請[建立一個或加入一個](#)。

The screenshot shows the 'Create a Slack App' dialog. It has fields for 'App Name' (with placeholder 'e.g. Super Service') and 'Development Slack Workspace' (set to 'Development Slack Workspace'). There's also a note about workspace changes and a terms of service agreement. At the bottom are 'Cancel' and 'Create App' buttons.

按一下 [建立應用程式]。Slack 會建立您的應用程式, 並產生用戶端識別碼和用戶端密碼。

新增重新導向 URL

接下來, 您將新增重新導向 URL。

1. 選取 [OAuth 與權限] 索引標籤。
2. 按一下 [新增重新導向 URL]。
3. 輸入 <https://slack.botframework.com>。
4. 按一下 [新增]。
5. 按一下 [儲存 URL]。

TestApp

OAuth & Permissions

Settings

- Basic Information
- Collaborators
- Install App
- Manage Distribution

Features

- Incoming Webhooks
- Interactive Components
- Slash Commands

OAuth & Permissions

- Event Subscriptions
- Bot Users
- User ID Translation

Slack ❤

- Help
- Contact
- Policies
- Our Blog

OAuth Tokens & Redirect URLs

These OAuth Tokens will be automatically generated when you finish connecting the app to your workspace. You'll use these tokens to authenticate your app.

Install App to Workspace

Redirect URLs

You will need to configure redirect URLs in order to automatically generate the Add to Slack button or to distribute your app. If you pass a URL in an OAuth request, it must (partially) match one of the URLs you enter here. [Learn more.](#)

Redirect URLs

https://slack.botframework.com

Add New Redirect URL

Save URLs

建立 Slack Bot 使用者

新增 Bot 使用者可讓您為 Bot 指派使用者名稱，並選擇是否要一直顯示為上線狀態。

- 選取 [Bot 使用者] 索引標籤。
- 按一下 [新增 Bot 使用者]。

TestApp

Bot User

Settings

- Basic Information
- Collaborators
- Install App
- Manage Distribution

Features

- Incoming Webhooks
- Interactive Components
- Slash Commands
- OAuth & Permissions
- Event Subscriptions

Bot Users

User ID Translation

Slack ❤

- Help
- Contact
- Policies
- Our Blog

You can bundle a bot user with your app to interact with users in a more conversational manner. [Learn more about how bot users work.](#)

Add a Bot User

按一下 [新增 Bot 使用者] 以確認您的設定，並將 [一律讓 Bot 顯示為上線狀態] 設為 [開啟]，然後按一下 [儲存變更]。

TestApp

Bot User

Settings

- Basic Information
- Collaborators
- Install App
- Manage Distribution

Features

- Incoming Webhooks
- Interactive Components
- Slash Commands
- OAuth & Permissions
- Event Subscriptions

Bot Users

- User ID Translation

Slack ❤️

- Help
- Contact
- Policies
- Our Blog

You can bundle a bot user with your app to interact with users in a more conversational manner. Learn more about [how bot users work](#).

Display name

Names must be shorter than 80 characters, and can't use punctuation (other than apostrophes and periods).

Default username

If this username isn't available on any workspace that tries to install it, we will slightly change it to make it work. Usernames must be all lowercase. They cannot be longer than 21 characters and can only contain letters, numbers, periods, hyphens, and underscores.

Always Show My Bot as Online

When this is off, Slack automatically displays whether your bot is online based on usage of the RTM API.

On

Add Bot User

訂閱 Bot 事件

請遵循下列步驟來訂閱六個特定的 Bot 事件。藉由訂閱 Bot 事件，應用程式將會收到所指定 URL 上的使用者活動通知。

TIP

您的 Bot 控制代碼就是您的 Bot 名稱。若要尋找 Bot 的控制代碼，請瀏覽 <https://dev.botframework.com/bots>，然後選擇 Bot 並記錄 Bot 名稱。

1. 選取 [事件訂用帳戶] 索引標籤。
2. 將 [啟用事件] 設定為 [開啟]。
3. 在 [要求 URL] 中輸入 `https://slack.botframework.com/api/Events/{YourBotHandle}`，其中 `{YourBotHandle}` 是您的 Bot 控制代碼 (不含大括號)。本範例中使用的 Bot 控制代碼為 **ContosoBot**。

Contoso Bot

Event Subscriptions

Settings

- Basic Information
- Collaborators
- Install App
- Manage Distribution

Features

- Incoming Webhooks
- Interactive Components
- Slash Commands
- OAuth & Permissions

Event Subscriptions

- Bot Users
- User ID Translation

Slack ❤️

- Help
- Contact
- Policies
- Our Blog

Enable Events

Request URL Verified ✓

https://slack.scratch.botframework.com/api/Events/ContosoBot

Change

We'll send HTTP POST requests to this URL when events occur. As soon as you enter a URL, we'll send a request with a challenge parameter, and your endpoint must respond with the challenge value. [Learn more.](#)

Subscribe to Workspace Events

To subscribe to an event, your app must have access to the related [OAuth permission scope](#).

Event Name	Description	Required Scope
No events added yet.		

Add Workspace Event

4. 在 [訂閱 Bot 事件] 中，按一下 [新增 Bot 使用者事件]。

5. 在事件清單中，選取以下六個事件類型：

- member_joined_channel
- member_left_channel
- message.channels
- message.groups
- message.im
- message.mpim

Subscribe to Bot Events

Bot users can subscribe to events related to the channels and conversations they're part of.

Event Name	Description	
member_joined_channel	A user joined a public or private channel	
member_left_channel	A user left a public or private channel	
message.channels	A message was posted to a channel	
message.groups	A message was posted to a private channel	
message.im	A message was posted in a direct message channel	
message.mpim	A message was posted in a multiparty direct message channel	

Add Bot User Event

6. 按一下 [儲存變更]。

Discard Changes

Save Changes

新增和設定互動式訊息 (選擇性)

如果您的 Bot 將使用 Slack 特有的功能 (例如按鈕), 請遵循下列步驟:

1. 選取 [互動式元件] 索引標籤, 然後按一下 [啟用互動式元件]。
2. 輸入 `https://slack.botframework.com/api/Actions` 作為要求 URL。
3. 按一下 [儲存變更] 按鈕。

The screenshot shows the Slack App Dashboard for a TestApp. The left sidebar has sections for Settings (Basic Information, Collaborators, Install App, Manage Distribution), Features (Incoming Webhooks, Interactive Components, Slash Commands, OAuth & Permissions, Event Subscriptions, Bot Users, User ID Translation), and Slack (Help, Contact, Policies, Our Blog). The 'Interactive Components' section is selected and highlighted with a blue background. The main content area is titled 'Interactive Components'.

Interactivity: A green toggle switch labeled 'On' is shown, with a red box drawn around it. Below it, a text box contains the URL `https://slack.botframework.com/api/Actions`, which is also highlighted with a red box.

Actions: A table header with columns 'Name', 'Description', and 'Callback ID'. Below the table is a button labeled 'Create New Action'.

Message Menus: A text box contains the URL `https://my.app.com/slack/options-load-endpoint`. Below it, a note states: "For message menu actions with `"data_source": "external"` we'll send an HTTP POST request with".

At the bottom right, there are two buttons: 'Discard Changes' and 'Save Changes', with a red box drawn around the 'Save Changes' button.

收集認證

選取 [基本資訊] 索引標籤, 然後捲動至 [應用程式認證] 區段。設定 Slack Bot 所需的用戶端識別碼、用戶端密碼和驗證權杖會隨即顯示。

App Credentials

These credentials allow your app to access the Slack API. They are secret. Please don't share your app credentials with anyone, include them in public code repositories, or store them in insecure ways.

App ID	Date of App Creation
ADAV416JV	October 10, 2018
Client ID	██████████90040641
Client Secret	██████████ <small>Show Regenerate</small>
You'll need to send this secret along with your client ID when making your oauth.access request.	
Signing Secret	██████████ <small>Show Regenerate</small>
Slack signs the requests we send you using this secret. Confirm that each request comes from Slack by verifying its unique signature.	
Verification Token	██████████Slack <small>Regenerate</small>
This deprecated Verification Token can still be used to verify that requests come from Slack, but we strongly recommend using the above, more secure, signing secret instead.	

提交認證

在另一個瀏覽器視窗中，返回 Bot Framework 網站 <https://dev.botframework.com/>。

1. 選取 [我的 Bot]，然後選擇要連線至 Slack 的 Bot。
2. 在 [通道] 區段中，按一下 Slack 圖示。
3. 在 [輸入 Slack 認證] 區段中，將 Slack 網站中的「應用程式認證」貼到適當的欄位。
4. 登陸頁面 URL 是選擇性的。您可以省略或變更此 URL。
5. 按一下 [檔案]。

Search (Ctrl+ /) «

Overview

Activity log

Access control (IAM)

Tags

Bot management

Build

Test in Web Chat

Analytics

Channels

Settings

Speech priming

Bot Service pricing

App Service Settings

Application Settings

All App service settings

Support + troubleshooting

New support request

Enter your Slack credentials

Step-by-step instructions to add the bot to Slack.

Client ID
90040641

Client Secret
3d9b

Verification Token
slack

Landing Page URL (optional)
http://yourbot.example.com/slack_help

Want to add your bot to Slack App Directory?
[Learn how](#)

Cancel **Save** Disabled

請依照指示將 Slack 應用程式的存取權授與您的「開發 Slack 小組」。

啟用 Bot

在 [設定 Slack] 頁面上，確認 [儲存] 按鈕旁的滑桿已設定為 [啟用]。您的 Bot 會設定為可與 Slack 中的使用者通訊。

建立 [新增至 Slack] 按鈕

在本頁的<新增 Slack 按鈕>區段中，Slack 會提供可用來協助 Slack 使用者找到 Bot 的 HTML。若要讓您的 Bot 搭配此 HTML，請使用在 Bot Slack 通道設定中找到的 URL 取代 href 值 (開頭為 `https://`)。請依照下列步驟操作，取得替代 URL。

1. 在 <https://dev.botframework.com/bots> 上，按一下您的 Bot。
2. 按一下 [通道]，以滑鼠右鍵按一下名為 **Slack** 的項目，然後按一下 [複製連結]。此 URL 現在在剪貼簿中。
3. 將剪貼簿中的此 URL 貼到為 [Slack] 按鈕提供的 HTML 中。此 URL 會取代 Slack 為此 Bot 提供的 href 值。

已獲授權的使用者可以按一下 [新增至 Slack] 按鈕 (由此處已修改的 HTML 提供)，來連線到 Slack 上的 Bot。

將 Bot 連線至 Telegram

2019/5/10 • [Edit Online](#)

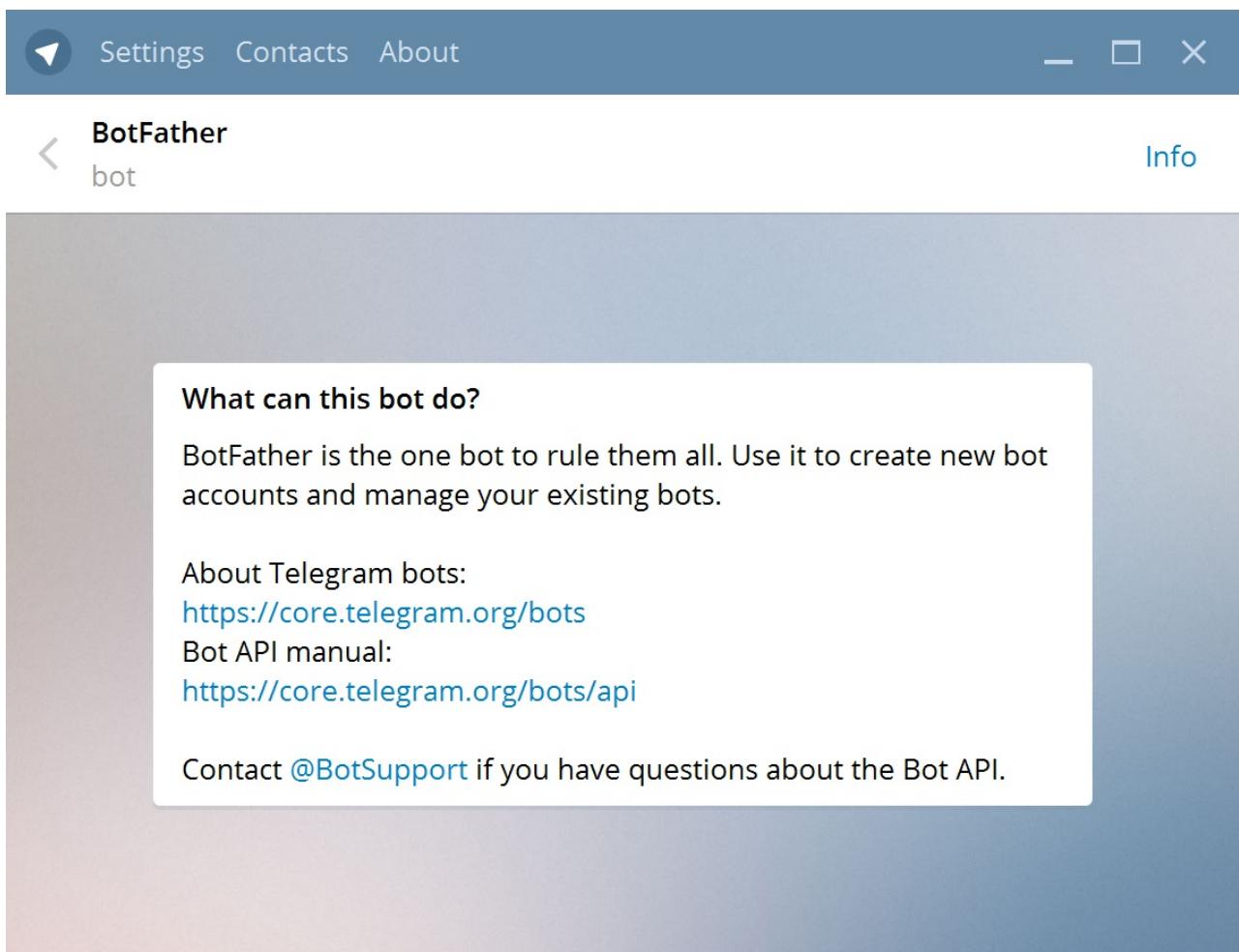
您可以設定您的 Bot 使用 Telegram 傳訊應用程式與其他人通訊。

TIP

若想了解各種 Bot Framework 功能的外觀及對此通道的運作方式，請[使用通道偵測器](#)。

請造訪 Bot Father 以建立新的 Telegram Bot

使用 Bot Father [建立新的 Telegram Bot](#)。



[Start](#)

建立新的 Telegram Bot

若要建立新的 Telegram Bot，請傳送命令 `/newbot`。



Settings Contacts About

- □ ×

BotFather

bot

Info

<https://core.telegram.org/bots>

You can control me by sending these commands:

/newbot - create a new bot
/token - generate authorization token
/revoke - revoke bot access token
/setname - change a bot's name
/setdescription - change bot description
/setabouttext - change bot about info
/setuserpic - change bot profile photo
/setinline - change inline settings
/setinlinefeedback - change inline feedback settings
/setcommands - change bot commands list
/setjoininggroups - can your bot be added to groups?
/setprivacy - what messages does your bot see in groups?
/deletebot - delete a bot



/newbot

create a new bot



/newbot



Send

指定易記的名稱

給 Telegram Bot 一個易記的名稱。



Settings Contacts About

- □ ×

BotFather

bot

Info

/revoke - revoke bot access token
/setname - change a bot's name
/setdescription - change bot description
/setabouttext - change bot about info
/setuserpic - change bot profile photo
/setinline - change inline settings
/setinlinefeedback - change inline feedback settings
/setcommands - change bot commands list
/setjoininggroups - can your bot be added to groups?
/setprivacy - what messages does your bot see in groups?
/deletebot - delete a bot
/cancel - cancel the current operation

1:44 PM

/newbot 1:44 PM ✓

Alright, a new bot. How are we going to call it? Please choose a name for your bot.

1:44 PM



Delightful Bot



Send

指定使用者名稱

給 Telegram Bot 一個唯一的使用者名稱。



Settings Contacts About

- □ ×

BotFather

bot

Info

/setinline - change inline settings
/setinlinefeedback - change inline feedback settings
/setcommands - change bot commands list
/setjoininggroups - can your bot be added to groups?
/setprivacy - what messages does your bot see in groups?
/deletebot - delete a bot
/cancel - cancel the current operation

1:44 PM

/newbot 1:44 PM ✓

Alright, a new bot. How are we going to call it? Please choose a name for your bot.

1:44 PM

Delightful Bot 1:44 PM ✓

Good. Now let's choose a username for your bot. It must end in `bot`. Like this, for example: TetrisBot or tetris_bot.

1:44 PM



DelightfulBot



Send

複製存取權杖

複製 Telegram Bot 的存取權杖。



輸入 Telegram Bot 的存取權杖

將您先前複製的權杖貼至 [存取權杖] 欄位，然後按一下 [提交]。

啟用 Bot

勾選 [在 Telegram 上啟用此 Bot]。然後按一下 [我已完成設定 Telegram]。

當您完成這些步驟時，您的 Bot 即已成功設定為在 Telegram 中與使用者進行通訊。

將 Bot 連線至 Twilio

2019/2/28 • [Edit Online](#)

您可以設定您的 Bot 使用 Twilio 雲端通訊平台與其他人通訊。

登入或建立 Twilio 帳戶來傳送和接收 SMS 訊息

如果您沒有 Twilio 帳戶，請[建立新的帳戶](#)。

建立 TwiML 應用程式

依照指示[建立 TwiML 應用程式](#)。

Create TwiML App

Properties

The screenshot shows the 'Properties' section of a TwiML App creation page. It includes fields for 'Friendly Name' (set to 'My TwiML app'), 'Voice' settings (Request URL: https://sms.botframework.com/api/sms, Method: HTTP POST), and 'Messaging' settings (Request URL: https://sms.botframework.com/api/sms, Method: HTTP POST). At the bottom are 'Save' and 'Cancel' buttons.

在 [屬性] 之下，輸入易記名稱。在本教學課程中，我們以「我的 TwiML 應用程式」為例。語音之下的 [要求 URL] 可以保留空白。在 [傳訊] 底下，[要求 URL] 應該是 `https://sms.botframework.com/api/sms`。

選取或新增電話號碼

請依照[這裡](#)的指示，透過主控台網站新增已驗證的呼叫端識別碼。完成之後，您會在 [管理號碼] 之下的 [作用中號碼] 中看見您已驗證的號碼。

Active Numbers

The screenshot shows the 'Active Numbers' management interface. A specific number (+1 610 609 8967) is selected. The 'CONFIGURATION' section displays the URLs for Voice and Messaging: 'Voice POST: https://demo.twilio.com/welcome/voice/' and 'Messaging POST: https://demo.twilio.com/welcome/sms/reply/'. These URLs are highlighted with a red box.

指定要用於語音和傳訊的應用程式

按一下號碼並移至 [設定]。在 [語音] 和 [傳訊] 之下，將 [設定方式] 設定為 TwiML 應用程式，以及將 [TwiML 應用程式] 設定為 [我的 TwiML 應用程式]。完成之後，按一下 [儲存]。

(610) 609-8967

Configure Calls Log Messages Log Event Log

Properties

FRIENDLY NAME	(610) 609-8967
SID	PNb7228f7f793c08cc22169b60d3202182
PHONE NUMBER	+16106098967
LOCATION	Havertown, PA US
CAPABILITIES	Voice, SMS, MMS

Voice

CONFIGURE WITH	TwiML App
TWIML APP	My TwiML app

Messaging

CONFIGURE WITH	TwiML App
TWIML APP	My TwiML app

Save

Cancel

Release this Number

請返回 [管理號碼]，您會看見 [語音] 和 [傳訊] 的組態都變更為 TwiML 應用程式。

Active Numbers

+ CLICK + TO BUY NEW NUMBER

Number	Voice URL	CAPABILITIES	CONFIGURATION
+1 610 609 8967 Havertown, PA	(610) 609-8967		Voice TwiML App: My TwiML app Messaging TwiML App: My TwiML app

收集認證

請返回 [主控台首頁](#)，您會在專案儀表板上看見您的帳戶 SID 和驗證權杖，如下所示。

Dashboard / [UPGRADE](#)

ProgrammableSMS Dashboard

Project Info

We've customized your dashboard based on the products you selected. Use the product getting started guides to get up and running.
We can't wait to see what you build!

PROJECT NAME	ProgrammableSMS edit
ACCOUNT SID	ACea27a63cc0192f429e7763062440749e
AUTH TOKEN	hide 7b2439a5536ae778724e52c36a35568f
Owner	1 manage
2FA	Disabled edit

[Programmable SMS](#) [Invite Your Team](#)

提交認證

在另一個視窗中，返回 Bot Framework 網站 <https://dev.botframework.com/>。

- 選取 [我的 Bot]，然後選擇要連線至 Twilio 的 Bot。這會將您導向 Azure 入口網站。
- 選取 [Bot 管理] 之下的 [通道]。按一下 Twilio (SMS) 圖示。
- 輸入電話號碼、帳戶 SID，以及您先前記錄的驗證權杖。完成之後，按一下 [儲存]。

 Search (Ctrl+ /)[Overview](#)[Activity log](#)[Access control \(IAM\)](#)[Tags](#)**Bot management**[Build](#)[Test in Web Chat](#)[Analytics](#)[Channels](#)[Settings](#)[Speech priming](#)[Bot Service pricing](#)**App Service Settings**[Application Settings](#)[All App service settings](#)**Support + troubleshooting**[New support request](#)

Enter your Twilio credentials

[Step-by-step instructions to add the bot to Twilio.](#)

Phone Number

+5(555)555-5555

Account Sid

Account SID can be found in your Twilio account settings

Auth Token

Auth token can be found in your Twilio account settings

Twilio (SMS) configured but still not working?

[Look at Twilio logs](#)[Cancel](#)[Save](#) Disabled

當您完成這些步驟時，您的 Bot 即已成功設定為使用 Twilio 與使用者進行通訊。

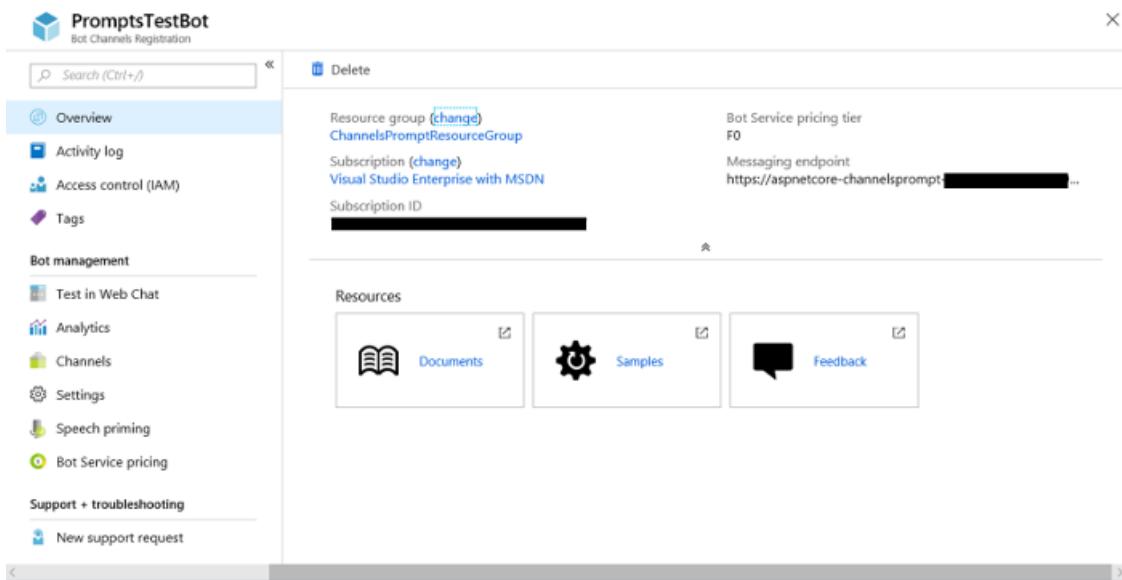
將 Bot 連線至「網路聊天」

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於最新版的 SDK (v4)。您可以在[這裡](#)找到舊版 SDK (v3) 的內容。

當您使用 Bot Service [建立 Bot](#) 時，會自動為您設定網路聊天頻道。網路聊天頻道包含網路聊天控制項，能夠讓您的使用者直接在網頁上與 Bot 進行互動。



Bot Framework 入口網站中的網路聊天頻道，包含您在網頁中內嵌網路聊天控制項所需的所有項目。如需使用網路聊天控制項，您只需要取得 Bot 的祕密金鑰，並在網頁中內嵌控制項即可。

取得您的 bot 紘密金鑰

1. 在 [Azure 入口網站](#) 中開啟您的 Bot，然後按一下 [通道] 刀鋒視窗。
2. 針對 網路聊天頻道，按一下 [編輯]。

Connect to channels

Name	Health	Published	
 Direct Line	Running	--	Edit 
 Web Chat	Running	--	Edit 

[Get bot embed codes](#)

3. 在 [祕密金鑰] 底下，按一下第一個金鑰的 [顯示]。

Configure Web Chat

The screenshot shows the 'Configure Web Chat' page. At the top, there are icons for adding a new site, editing the default site, and disabling it. The 'Default Site' section is selected. Under 'Secret keys', two long strings of characters are displayed in input fields, each with 'Show' and 'Regenerate' buttons. Below this is an 'Embed code' section containing an iframe tag with placeholder values. A 'Copy' button and a link to 'Learn more about advanced customization options for Web Chat' are present. A 'Preview' section explains the purpose of enabling preview mode and includes a checked checkbox for 'Enable Preview'. At the bottom is a blue 'Done' button.

4. 複製 [祕密金鑰] 和 [內嵌程式碼]。

5. 按一下 [完成]。

在您的網站中內嵌網路聊天控制項

您可以使用下列兩個選項之一，在您的網站中內嵌網路聊天控制項。

選項 1 - 保持隱藏您的祕密、將您的祕密與權杖交換，並產生內嵌

如果您可以執行伺服器對伺服器要求，將您的網路聊天祕密與暫存權杖交換，且想要讓其他開發人員難以在他們的網站中內嵌您的 Bot，請使用此選項。雖然使用此選項無法完全防止其他開發人員在其網站中內嵌您的 Bot，但確實會增加難度。

若要將您的祕密與權杖交換並產生內嵌：

1. 請發出 **GET** 要求給 <https://webchat.botframework.com/api/tokens>，並透過 **Authorization** 標頭傳遞您的網路聊天祕密。**Authorization** 標頭會使用 **BotConnector** 配置並包含您的祕密，如下列範例要求中所示。
2. 您 **GET** 要求的回應會包含權杖（以引號括住），可透過轉譯 **iframe** 內的 網路聊天控制項用來開始對話。權杖僅對一個對話有效；若要開始另一個對話，您必須產生新的權杖。
3. 在您從 Bot Framework 入口網站中的網路聊天頻道複製的 **iframe** 內嵌程式碼內（如上方取得您的 bot 紘密金鑰中所述），將 **s=** 參數變更為 **t=**，並以您的權杖來取代 "YOUR_SECRET_HERE"。

NOTE

權杖在到期前會自動更新。

範例 要求

```
requestGET https://webchat.botframework.com/api/tokens  
Authorization: BotConnector YOUR_SECRET_HERE
```

範例 回應

```
"IIbSpLnn8sA.dBB.MQBhAFMAZwBXAHoANGBQAGcAZABKAEcAMwB2ADQASABjAFMAegBuAHYANwA.bbguxy0v0gE.cccJjh-TFDs.ruXQyivVZIcgvosGaFs_4jRj1AyPnDt1wk1HMBb5Fuw"
```

範例 iframe (使用 權杖)

```
<iframe src="https://webchat.botframework.com/embed/YOUR_BOT_ID?t=YOUR_TOKEN_HERE"></iframe>
```

範例 html 程式碼

```
<!DOCTYPE html>  
<html>  
<body>  
    <iframe id="chat" style="width: 400px; height: 400px;" src=''/></iframe>  
</body>  
<script>  
  
    var xhr = new XMLHttpRequest();  
    xhr.open('GET', "https://webchat.botframework.com/api/tokens", true);  
    xhr.setRequestHeader('Authorization', 'BotConnector ' + 'YOUR SECRET HERE');  
    xhr.send();  
    xhr.onreadystatechange = processRequest;  
  
    function processRequest(e) {  
        if (xhr.readyState == 4 && xhr.status == 200) {  
            var response = JSON.parse(xhr.responseText);  
            document.getElementById("chat").src="https://webchat.botframework.com/embed/lucas-direct-line?  
t="+response  
        }  
    }  
  
</script>  
</html>
```

選項 2 - 在您的網站中使用祕密來內嵌網路聊天控制項

如果您想要讓其他開發人員輕鬆地在其網站中內嵌您的 Bot，請使用此選項。

WARNING

如果您使用此選項，其他開發人員只需複製您的內嵌程式碼，即可在其網站中內嵌您的 Bot。

若要藉由指定 `iframe` 標記內的祕密，在您的網站中內嵌 Bot：

1. 請從 Bot Framework 入口網站內的網路聊天頻道，複製 `iframe` 內嵌程式碼 (如上方取得您的 bot 祕密金鑰中所述)。
2. 在內嵌程式碼內，將 "YOUR_SECRET_HERE" 取代為您從相同頁面中複製的祕密金鑰值。

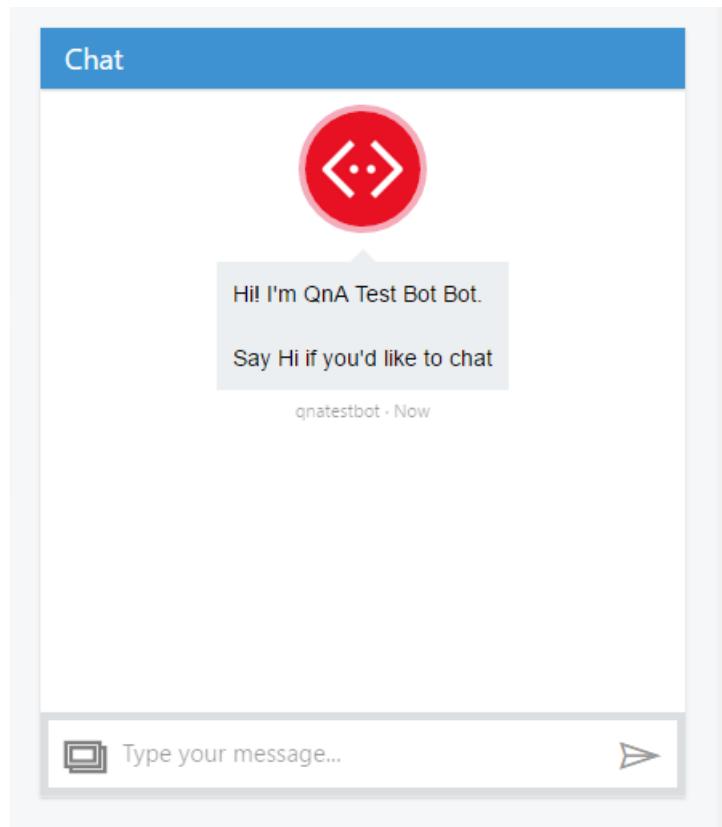
範例 iframe (使用 祕密)

```
<iframe src="https://webchat.botframework.com/embed/YOUR_BOT_ID?s=YOUR_SECRET_HERE"></iframe>
```

設計網路聊天控制項樣式

您可以使用 `iframe` 的 `style` 屬性來指定 `height` 和 `width`，以變更網路聊天控制項的大小。

```
<iframe style="height:480px; width:402px" src="... SEE ABOVE ..."></iframe>
```



其他資源

您可以在 GitHub 上[下載 Web 聊天控制項的原始程式碼](#)。

配置 bot 設定

2019/2/28 • [Edit Online](#)

Bot 設定 (例如顯示名稱、圖示和 Application Insights) 可在 [設定] 刀鋒視窗中檢視及修改。

The screenshot shows the 'st-01 - Settings' blade in the Azure portal. On the left, there's a navigation menu with the following items:

- Overview
- Activity log
- Access control (IAM)
- Tags
- Settings** (highlighted in blue)
- Speech priming
- Bot Service pricing
- Application Settings
- All App service settings
- New support request

The main content area is titled 'Bot profile'. It contains sections for 'Icon' (with a placeholder for 'Upload custom icon'), 'Display name' (set to 'st-01'), 'Bot handle' (set to 'st-01'), 'Configuration' (with 'Messaging endpoint' set to 'https://[REDACTED]/Ddsmd...'), 'Microsoft App ID' (set to '[REDACTED]'), 'Analytics' (with 'Application Insights Instrumentation key' and 'Application Insights API key' fields), and 'Application Insights Application ID' (set to '[REDACTED]'). At the top right, there are 'Save' and 'Discard' buttons.

以下是 [設定] 刀鋒視窗中的欄位清單：

欄位	說明
圖示	自訂圖示，以便透過視覺化方式在頻道中識別您的 Bot，並做為 Skype、Cortana 和其他服務的圖示。此圖示必須採用 PNG 格式，並且不得大於 30K。此值隨時可以變更。
顯示名稱	您的 Bot 在頻道與目錄中的名稱。此值隨時可以變更。35 個字元限制。
Bot 控制代碼	Bot 的唯一識別碼。藉由 Bot 服務建立 Bot 之後就無法變更此值。
傳訊端點	要與您的 Bot 進行通訊的端點。

欄位	說明
Microsoft 應用程式識別碼	Bot 的唯一識別碼。無法變更此值。您可以按一下 [管理] 連結以產生新的密碼。
Application Insights 檢測金鑰	Bot 遙測的唯一索引鍵。如果您想要接收此 Bot 的 Bot 遙測資料，請將您的 Azure Application Insights 金鑰複製到此欄位中。此為選用值。在 Azure 入口網站中建立的 Bot 會自行產生此金鑰。如需此欄位的更多詳細資料，請參閱 Application Insights 金鑰 。
Application Insights API	Bot 分析的唯一金鑰。如果您想要在儀表板中檢視 Bot 相關分析，請將您的 Azure Application Insights API 金鑰複製到此欄位中。此為選用值。如需此欄位的更多詳細資料，請參閱 Application Insights 金鑰 。
Application Insights 應用程式識別碼	Bot 分析的唯一金鑰。如果您想要在儀表板中檢視 bot 相關分析，請將您的 Azure Insights 應用程式識別碼金鑰複製到此欄位中。此為選用值。在 Azure 入口網站中建立的 Bot 會自行產生此金鑰。如需此欄位的更多詳細資料，請參閱 Application Insights 金鑰 。

NOTE

在變更自己的 Bot 的設定後，請按一下刀鋒視窗頂端的 [儲存] 按鈕以儲存新的 Bot 設定。

後續步驟

現在，您已了解如何配置自己的 Bot 服務的設定值，請深入了解如何配置語音預備。

[語音預備](#)

設定語音預備

2019/2/28 • [Edit Online](#)

語音預備會改善您 Bot 中常用的口語單字和片語辨識。對於已啟用語音且使用 網路聊天和 Cortana 通道的 Bot，語音預備會使用 Language Understanding (LUIS) 應用程式中指定的範例，來改善重要單字的語音辨識準確度。

您的 Bot 可能已經與 LUIS 應用程式整合，或您可以選擇建立 LUIS 應用程式，來與語音預備的 Bot 相關聯。LUIS 應用程式包含您預期使用者會對 Bot 說的範例。您需要 Bot 辨識的重要單字都應標示為實體。例如，在下棋 Bot 中，您要確定當使用者說 "Move knight" 時，Bot 不會解讀為 "Move night"。LUIS 應用程式應該包含將 "knight" 標示為實體的範例。

NOTE

若要使用語音預備搭配網路聊天頻道，您必須使用 Bing 語音服務。請參閱[啟用網路聊天頻道中的語音](#)，以了解如何使用 Bing 語音服務。

IMPORTANT

語音預備僅適用於針對 Cortana 通道或網路聊天頻道設定的 Bot。

IMPORTANT

非美國地區的 LUIS 應用程式不支援預備，包括 :eu.luis.ai 和 au.luis.ai

變更您 Bot 所使用的 LUIS 應用程式清單

若要變更 Bing 語音搭配您 Bot 使用的 LUIS 應用程式清單，請執行下列作業：

1. 在 Bot Service 刀鋒視窗上按一下 [語音預備]。可提供給您的 LUIS 應用程式清單隨即顯示。
2. 選取您想要 Bing 語音使用的 LUIS 應用程式。
 - a. 若要在清單中選取 LUIS 應用程式，請將滑鼠停在 LUIS 模型上，直到出現一個核取方塊，然後勾選該核取方塊。
 - b. 若要選取不在清單中的 LUIS 應用程式，請捲動到底部，並將 LUIS 應用程式識別碼 GUID 輸入文字方塊中。
3. 按一下 [儲存]，來儲存與您 Bot 的 Bing 語音相關聯的 LUIS 應用程式清單。

1202sdkcsbasic1 - Speech priming

SDK Bot (Web App)

Search (Ctrl+ /) Save Discard

Overview Activity log Access control (IAM) Tags

BOT MANAGEMENT Build Test in Web Chat Analytics Channels Settings

Speech priming

Bot Service pricing

Improve speech recognition accuracy

Optimize speech recognition by associating your Language Understanding app from luis.ai with Bing Speech, which is part of Cognitive Services offering. Note that Speech Recognition for bots is currently available in the Cortana and Web Chat channels only.

fnechobot-rg-LgCBxqp4IIJ Yes

fnsimplebot-rg-KInN1EoQZLC

Enter a LUIS application ID

其他資源

- 若要深入了解在網路聊天中啟用語音，請參閱[在網路聊天頻道中啟用語音](#)。
- 若要深入了解語音預備，請參閱[Bot Framework 中的語音支援 – 與 Directline、與 Cortana 進行 Web 聊天](#)（英文）。
- 若要深入了解 LUIS 應用程式，請參閱[Language Understanding Intelligent Service](#)。

使用 Bot 服務建立 Bot

2019/2/28 • [Edit Online](#)

如果您的 Bot 已裝載於其他位置，而且您想要使用 Bot 服務與其他通道連線，則您必須使用 Bot 服務註冊您的 Bot。在本主題中，了解如何藉由建立 **Bot 通道註冊** Bot 服務，使用 Bot 服務註冊您的 Bot。

IMPORTANT

如果您的 Bot 未裝載於 Azure 中，您只需要註冊 Bot 即可。如果您透過 Azure 入口網站[建立 Bot](#)，表示您的 Bot 已使用 Bot 服務註冊。

登入 Azure

登入 [Azure 入口網站](#)。

TIP

若您還沒有訂用帳戶，則可以註冊[免費帳戶](#)。

建立 Bot 通道註冊

您需要 **Bot 通道註冊** Bot 服務才能使用 Bot 服務功能。註冊 Bot 可讓您將 Bot 連線至通道。

如要建立 **Bot 通道註冊**，請執行下列操作：

1. 按一下 Azure 入口網站左上角的 [新增] 按鈕，然後選取 [AI + 認知服務] > [Bot 通道註冊]。
2. 隨即開啟含有 **Bot 通道註冊** 相關資訊的新刀鋒視窗。按一下 [建立] 按鈕來開始建立程序。
3. 在 [Bot 服務] 刀鋒視窗中，依照下圖下方表格中說明的內容，提供有關您 Bot 的必要資訊。

Bot Channels Registration

Bot Service

* Bot name ?

* Subscription

* Resource group
 Create new Use existing

* Location

Pricing tier ([View full pricing details](#))

Messaging endpoint

Application Insights ? On Off

* Application Insights Location ?

Microsoft App ID and password ? [>](#)
Auto create App ID and password

Pin to dashboard

Create [Automation options](#)

設定	建議的值	說明
Bot 名稱	您的 Bot 顯示名稱	Bot 在通道和目錄中的顯示名稱。您可以隨時變更此名稱。
訂用帳戶	您的訂用帳戶	選取您要使用的 Azure 訂用帳戶。
資源群組	myResourceGroup	您可以建立新的 資源群組 , 或選擇現有的資源群組。
位置	美國西部	選擇接近您的 Bot 部署位置, 或接近您的 Bot 存取其他服務的位置。
定價層	F0	選取定價層。您可以隨時更新定價層。如需詳細資訊, 請參閱 Bot 服務價格 。
傳訊端點	URL	輸入您的 Bot 傳訊端點的 URL。
Application Insights	另一	決定您要開啟或關閉 Application Insights 。如果您選取 [開啟], 您還必須指定區域位置。

設定	建議的值	說明
Microsoft 應用程式識別碼和密碼	自動建立應用程式識別碼和密碼	如果您需要手動輸入 Microsoft 應用程式識別碼和密碼，請使用此選項。否則，在 Bot 建立流程當中，便會為您建立新的 Microsoft 應用程式識別碼和密碼。

4. 按一下 [建立] 即可建立服務並註冊您的 Bot 的傳訊端點。

勾選 [通知] 來確認已建立的註冊。通知會從 [部署進行中] 變更為 [部署成功]。按一下 [前往資源] 按鈕以開啟 Bot 的資源刀鋒視窗。

Bot 通道密碼

Bot 通道註冊 Bot 服務沒有相關聯的應用程式服務。因此，此 Bot 服務只有 *MicrosoftAppID*。您需要手動產生密碼，然後自行儲存。您必須使用此密碼才能使用 [模擬器](#) 測試 Bot。

若要產生 Microsoft 應用程式密碼，請執行下列操作：

1. 在 [設定] 刀鋒視窗中，按一下 [管理]。這是 **Microsoft 應用程式識別碼** 顯示的連結。此連結會開啟視窗，您可以在其中產生新的密碼。

BOT MANAGEMENT

- Test in Web Chat
- Analytics
- Channels
- Settings**
- Speech priming
- Bot Service pricing

SUPPORT + TROUBLESHOOTING

- New support request

Configuration

Messaging endpoint
https URL

* Microsoft App ID [\(Manage\)](#)

Analytics

Application Insights Instrumentation key

Application Insights API key
API key (User-Generated Application Insights API key)

Application Insights Application ID

2. 按一下 [產生新密碼]。這會產生 Bot 的新密碼。複製此密碼，並儲存至檔案。這是您唯一會看到此密碼的時機。如果您未儲存完整的密碼，您必須重複此程序來建立新的密碼，以便稍後使用。

Properties

Name: [REDACTED]

Application Id: [REDACTED]

Application Secrets

Generate New Password **Generate New Key Pair** Upload Public Key

Type	Password/Public Key
Password	=[REDACTED]

Created: Nov 16, 2017 4:46:57 PM Delete

New password generated

This is the only time when it will be displayed. Please store it securely.

[REDACTED]

Ok

更新 Bot

如果您使用適用於 Node.js 的 Bot Framework SDK，請設定下列環境變數：

- MICROSOFT_APP_ID
- MICROSOFT_APP_PASSWORD

如果您使用適用於 .NET 的 Bot Framework SDK, 請在 web.config 檔案中設定下列索引鍵值：

- MicrosoftAppId
- MicrosoftAppPassword

測試 Bot

現在, 既然您已建立 Bot, 請[在網路聊天中進行測試](#)。輸入訊息, 您的 Bot 應會回應。

後續步驟

本主題中, 您已了解如何使用 Bot 服務註冊裝載的 Bot。下一個步驟是了解如何管理您的 Bot 服務。

[管理 Bot](#)

將 Bot 移轉至 Azure

2019/4/9 • [Edit Online](#)

所有在 [Bot Framework 入口網站](#) 中建立的 **Azure Bot Service (預覽版)** Bot，都必須移轉至 Azure 中的新 Bot Service。此服務已在 2017 年 12 月正式推出 (GA)。

請注意，僅連線至下列通道的註冊 Bot「不需要」移轉：**Teams、Skype 或 Cortana**。例如，連線至 **Facebook** 和 **Skype** 的註冊 Bot 需要移轉，但連線至 **Skype** 和 **Cortana** 註冊 Bot 則不需要移轉。

IMPORTANT

移轉使用 Node.js 建立的 Functions Bot 之前，您必須使用 **Azure Functions Pack** 將 `node_modules` 模組封裝在一起。這麼做可改善移轉期間的效能和 Functions Bot 在移轉之後的執行效能。若要封裝您的模組，請參閱[使用 Funcpack 封裝 Functions Bot](#)。

若要移轉您的 Bot，請執行下列作業：

1. 登入 [Bot Framework 入口網站](#)，然後按一下 [我的 Bot]。
2. 為您要移轉的 Bot 按一下 [移轉] 按鈕。
3. 接受條款，然後按一下 [移轉] 開始進行移轉程序，或按一下 [取消] 以取消此動作。

IMPORTANT

移轉工作正在進行時，請勿離開頁面或重新整理頁面。這麼做會導致移轉工作過早停止，且您將必須重新執行此動作。為了確保移轉順利完成，請等候確認訊息出現。

移轉程序順利完成後，[移轉狀態] 會指出您的 Bot 已移轉，且 [回復移轉] 按鈕在移轉日期後的一週內都可供使用，以備在問題發生時回復。

按一下已移轉的 Bot 名稱，將會在 [Azure 入口網站](#) 中開啟該 Bot。

使用 Funcpack 封裝 Functions Bot

使用 Node.js 建立的 Functions Bot 在移轉之前，必須先使用 [Funcpack](#) 進行封裝。若要以 Funcpack 封裝您的專案，請執行下列步驟：

1. 將您的程式碼[下載](#)到本機 (如果尚未下載)。
2. 將 `packages.json` 中的 npm 套件更新為最新版本，然後執行 `npm install`。
3. 開啟 `messages/index.js`，並將 `module.exports = { default: connector.listen() }` 變更為
`module.exports = connector.listen();`
4. 透過 npm 安裝 Funcpack: `npm install -g azure-functions-pack`
5. 若要封裝 `node_modules` 目錄，請執行下列命令: `funcpack pack ./`
6. 使用 Bot Framework 模擬器執行 Functions Bot，在本機測試您的 Bot。請在[這裡](#)深入了解如何執行 Funcpack Bot。
7. 將您的程式碼上傳回 Azure。請確定 `.funcpack` 目錄已上傳。您不需要上傳 `node_modules` 目錄。
8. 測試您遠端的 Bot，以確定它會如預期回應。
9. 使用上述步驟遷移您的 Bot。

實際移轉狀況

您可以根據所要移轉的 Bot 類型，利用下列清單深入了解實際的運作情況。

- **Web 應用程式 Bot 或 Functions Bot**: 對於這些類型的 Bot，系統會將舊有 Bot 中原始程式碼專案複製到新的 Bot。其他資源（例如 Bot 的儲存體、Application Insights、LUIS 等）則會保持原狀。在這類情況下，新 Bot 中現有資源會具有識別碼/金鑰/密碼的複本。
- **Bot 通道註冊**: 對於此類型的 Bot，移轉程序只會建立新的 **Bot 通道註冊**，並從舊的 Bot 將端點複製過去。
- 無論您要移轉哪些類型的 Bot，移轉程序都不會以任何形式修改現有 Bot 的狀態。這可讓您在必要時安全地進行回復。
- 如果您設定了 [持續部署](#)，您將必須重新設定，原始檔控制才會轉而連線至新的 Bot。

了解移轉之後的 Azure 資源

移轉完成後，您的資源群組將會包含 Bot 運作所需的多項 Azure 資源。資源的類型和數目取決於您所移轉的 Bot 類型。若要深入了解，請參閱以下各節。

註冊 Bot

這是最簡單的類型。Azure 中的資源群組只會包含一項屬於「Bot 通道註冊」的資源。此資源相當於先前位於 Bot Framework 開發人員入口網站中的 Bot 記錄。

NAME ↑↓	TYPE ↑↓
registrationbot-demo	Bot Channels Registration

Web 應用程式 Bot

「Web 應用程式 Bot」移轉將會佈建「Web 應用程式 Bot」類型的 Bot Service 資源，和新的 App Service Web 應用程式（在下列螢幕擷取畫面中以綠色顯示）。先前的 Azure Bot Service（預覽版）Bot 會保留於原處，且可以刪除（在下列螢幕擷取畫面中以紅色顯示）。

NAME ↑↓	TYPE ↑↓
Default1	App Service plan
webappbot-demo	Web App Bot
webappbot-demo	App Service
webappbot-demouj4i2n	Application Insights
webappbot-demo-upgrade	App Service

Functions Bot

Functions Bot 移轉將會佈建 "Functions Bot" 類型的 Bot Service 資源，和新的 App Service Functions 應用程式（在下列螢幕擷取畫面中以綠色顯示）。先前的 Azure Bot Service（預覽版）Bot 會保留於原處，且可以刪除（在下列螢幕擷取畫面中以紅色顯示）。

NAME ↑↓	TYPE ↑↓
functionsbot-demo	Functions Bot
functionsbot-demo	App Service
functionsbotdemotopwm6	Storage account
functionsbot-demotopwm6	Application Insights
functionsbot-demo-upgrade	App Service
WestUSPlan	App Service plan

回復移轉

如果 Bot 在移轉期間或移轉之後發生錯誤，您可以回復移轉。若要回復移轉，請執行下列作業：

1. 登入 [Bot Framework 入口網站](#)，然後按一下 [我的 Bot]。
2. 為您要回復的 Bot 按一下 [回復移轉] 按鈕。此時會出現提示。
3. 按一下 [是，我要回復] 以繼續作業，或按 [取消] 以取消回復動作。

NOTE

回復功能可在移轉後的一週內使用，但只有在移轉後的 Bot 有問題時，才應該使用。

移轉疑難排解/已知問題

我的 node.js/Functions Bot 已成功移轉，但無法回應：

- **檢查端點**
 - 移至 Bot 資源中的 [設定] 刀鋒視窗，並確認 Bot 端點具有 "code" 查詢字串參數及其值。如果沒有，您必須將其加入。
- **在 Kudu 中檢查備份的祕密資料夾**
 - 在某些罕見的情況下，某些備份祕密檔案可能會產生衝突。請移至 Kudu 中的 **home\data\Functions\secrets** 資料夾，並刪除您在該處找到的任何 **host.snapshot** (或 **host.backup**) 檔案。其中應該只有一個 **host.json** 和一個 **messages.json**。最後，請重新啟動 App Service，然後重新嘗試與 Bot 聊天。

若有任何其他問題，請將 CRI 提交至 Azure 支援，或在 [GitHub](#) 中提出問題。

後續步驟

現在，您的 Bot 已移轉，接下來請了解如何從 Azure 入口網站管理您的 Bot。

[管理 Bot](#)

v3 和 v4 .NET SDK 之間的差異

2019/5/10 • [Edit Online](#)

Bot Framework SDK 第 4 版支援與第 3 版相同的基礎 Bot Framework 服務。不過, v4 是舊版 SDK 的重構, 讓開發人員對於其 Bot 有更多的彈性和控制權。SDK 中的主要變更包括:

- Bot 配接器的簡介。此配接器是活動處理堆疊的一部分。
 - 此配接器會處理 Bot Framework 驗證。
 - 此配接器會管理通道和 Bot 回合處理常式之間的傳入和傳出流量, 並封裝對 Bot Framework 連接器的呼叫。
 - 此配接器會初始化每個回合的內容。
 - 如需詳細資料, 請參閱 [Bot 運作方式](#)。
- 重構的狀態管理。
 - 狀態資料不再於 Bot 內自動提供使用。
 - 現在會透過狀態管理物件和屬性存取子管理狀態。
 - 如需詳細資料, 請參閱 [管理狀態](#)。
- 新的 Dialogs 程式庫。
 - v3 對話需要針對新的對話程式庫重新撰寫。
 - 可評分項目已不存在。將控制權交給您的對話之前, 可以檢查「全域」命令。根據您設計 v4 Bot 的方式, 這可能位在訊息處理常式或父代對話方塊中。如需範例, 請參閱如何[處理使用者中斷](#)。
 - 如需詳細資料, 請參閱 [對話方塊程式庫](#)。
- 支援 ASP.NET Core。
 - 建立新 C# Bot 的範本是以 ASP.NET Core 架構為目標。
 - 您仍然可將 ASP.NET 用於 Bot, 但 v4 的重點在於支援 ASP.NET Core 架構。
 - 如需此架構的詳細資訊, 請參閱 [ASP.NET Core 簡介](#)。

活動處理

當您為 Bot 建立配接器時, 您也會提供訊息處理常式委派, 該委派會接收來自通道和使用者的連入活動。此配接器會針對每項接收的活動建立回合內容物件。其會將回合內容物件傳遞給 Bot 的回合處理常式, 然後在回合完成時處置物件。

回合處理常式可接收許多類型的活動。一般情況下, 您只想將「訊息」活動轉送給 Bot 包含的任何對話。如果您從 `ActivityHandler` 衍生 Bot, Bot 的回合處理常式會將所有訊息活動轉送至 `OnMessageActivityAsync`。覆寫此方法來加入訊息處理邏輯。如需活動類型的詳細資訊, 請參閱 [活動結構描述](#)。

處理回合

在處理訊息時, 使用回合內容來取得連入活動的相關資訊, 並將活動傳送給使用者:

若要取得連入活動	取得回合內容的 <code>Activity</code> 屬性。
若要建立活動並傳送給使用者	呼叫回合內容的 <code>SendActivityAsync</code> 方法。 如需詳細資訊, 請參閱 傳送和接收簡訊和將媒體新增至訊息 。

`MessageFactory` 類別會提供一些協助程式方法, 以供建立活動及設定其格式。

可評分項目已不存在

在 Bot 的訊息迴圈中處理這些項目。如需如何透過 v4 對話執行此作業的說明，請參閱如何[處理使用者中斷](#)。

可組合的可評分分派樹狀目錄與可組合的鏈結對話 (例如[_預設例外狀況_](#)) 也都會消失。重新產生這項功能的方法之一，就是在 Bot 的回合處理常式內實作。

狀態管理

在 v3 中，您可以將對話資料儲存在 Bot 狀態服務中，此服務屬於 Bot Framework 所提供的較大型服務套件。不過，該服務已在 2018 年 3 月 31 日淘汰。從 v4 開始，管理狀態的設計考量就像任何 Web 應用程式一樣，並且有許多選項可用。通常，在記憶體和相同程序中進行快取是最簡單的方式；不過，針對生產應用程式，您應該更永久地儲存狀態，例如儲存在 SQL 或 NoSQL 資料庫，或是儲存為 blob。

v4 不會使用 `UserData`、`ConversationData` 及 `PrivateConversationData` 屬性和資料包來管理狀態。如[管理狀態](#)所述，現在會透過狀態管理物件和屬性存取子管理狀態。

v4 會定義 `UserState`、`ConversationState` 和 `PrivateConversationState` 類別，以便管理 Bot 的狀態資料。您需要針對想保存的每個屬性建立狀態屬性存取子，而不只是讀取並寫入至預先定義的資料包。

設定狀態

可能的話，在適用於 .NET Core 的 `Startup.cs` 或在適用於 .NET Framework 的 `Global.asax.cs` 中，應該將狀態設定為單例。

1. 初始化一或多個 `IStorage` 物件。這代表 Bot 資料的備份存放區。v4 SDK 會提供一些[儲存層](#)。您也可以實作自己的儲存層，以連線到不同類型的存放區。
2. 然後，視需要建立並註冊[狀態管理](#)物件。您具有如 v3 可用的相同範圍，而且可以視需要建立其他範圍。
3. 然後，針對 Bot 所需的屬性建立並註冊[狀態屬性存取子](#)。在狀態管理物件內，每個屬性存取子都需要唯一的名稱。

使用狀態

每次建立 Bot 時，您可以使用相依性插入來存取這些狀態 (在 ASP.NET 中，系統會為每個回合建立 Bot 或訊息控制器的新執行個體)。使用狀態屬性存取子來取得及更新屬性，並使用狀態管理物件將任何變更寫入儲存體。了解您應該將並行問題列入考量後，以下說明如何完成一些常見工作。

若要建立狀態屬性存取子	呼叫 <code>BotState.CreateProperty<T></code> 。 <code>BotState</code> 是交談、私人交談和使用者狀態的抽象基底類別。
若要取得目前的屬性值	呼叫 <code>IStatePropertyAccessor<T>.GetAsync</code> 。 如果之前尚未設定任何值，則會使用預設處理站參數來產生值。
若要更新目前快取的屬性值	呼叫 <code>IStatePropertyAccessor<T>.SetAsync</code> 。 這只會更新快取，而不會更新支援儲存層。
若要將狀態變更保存至儲存體	在結束回合處理常式之前，針對狀態已變更的狀態管理物件， 呼叫 <code>BotState.SaveChangesAsync</code> 。

管理並行存取

您的 Bot 可能需要管理狀態並行。如需詳細資訊，請參閱[管理狀態](#)的儲存狀態一節，以及直接寫入儲存體的[使用 eTag 管理並行](#)一節。

對話方塊程式庫

以下是對話的一些主要變更：

- Dialogs 程式庫現在是個別的 NuGet 套件：**Microsoft.Bot.Builder.Dialogs**。
- 對話類別不再必須是可序列化。透過 `DialogState` 狀態屬性存取子管理對話狀態。
 - 現在會保存回合之間的對話狀態屬性，而不是對話物件本身。
- `IDialogContext` 介面會由 `DialogContext` 類別所取代。在回合內，您會建立「對話集」的對話內容。
 - 此對話內容會封裝對話堆疊（舊的堆疊框架）。這項資訊會保存在對話狀態屬性內。
- `IDialog` 介面會由抽象 `Dialog` 類別所取代。

定義對話

雖然所提供的 v3 能夠使用 `IDialog` 介面彈性地實作對話方塊，但這表示您必須針對功能（例如驗證）實作自己的程式碼。在 v4 中，現在有提示類別可自動驗證使用者輸入、將其限制為特定類型（例如整數），並在使用者提供有效輸入之前，反覆提示使用者。一般來說，這表示開發人員可撰寫較少的程式碼。

您現在有一些選項可定義對話：

元件對話，其衍生自 <code>ComponentDialog</code> 類別	可讓您封裝對話程式碼，而不會與外部內容發生命名衝突。請參閱 重複使用對話 。
瀑布式對話，也就是 <code>WaterfallDialog</code> 類別的執行個體	其設計訴求是要搭配提示對話運作，而提示對話會提示使用者輸入並驗證各種類型的使用者輸入。瀑布式對話會為您將大部分的程序自動化，但是對您的對話程式碼強加特定形式，請參閱 循序對話流程 。
自訂對話，其衍生自抽象 <code>Dialog</code> 類別	這讓您的對話運作方式擁有最大彈性，但您也需要深入了解對話堆疊的實作方式。

在 v3 中，您已使用 `FormFlow` 執行一組工作的步驟。在 v4 中，瀑布式對話會取代 FormFlow。當您建立瀑布式對話時，您會在建構函式中定義對話方塊的步驟。執行的步驟順序會完全遵循您宣告的方式，並且會自動地逐一前進。

您也可以使用多個對話方塊來建立複雜的控制流程，請參閱[進階對話流程](#)。

若要存取對話，您需要在「對話集」中放入其執行個體，然後為該集合產生「對話內容」。當您建立對話集時，您需要提供對話狀態屬性存取子。這可讓架構保存回合之間的對話狀態。[管理狀態](#)說明如何在 v4 中管理狀態。

使用對話

以下是 v3 的常見作業清單，以及在瀑布式對話內完成這些作業的方式。請注意，瀑布式對話的每個步驟預計傳回 `DialogTurnResult` 值。若未如此，瀑布可能提早結束。

作業	V3	V4
處理對話的開始	實作 <code>IDialog.StartAsync</code>	讓此項成為瀑布式對話的第一個步驟。
傳送活動	呼叫 <code>IDialogContext.PostAsync</code> 。	呼叫 <code>ITurnContext.SendActivityAsync</code> 。 使用步驟內容的 <code>Context</code> 屬性來取得回合內容。
等候使用者回應	使用 <code>IAwaitable<IMessageActivity></code> 參數並呼叫 <code>IDialogContext.Wait</code> 。	傳回等候 <code>ITurnContext.PromptAsync</code> 開始提示對話。然後在瀑布的下一個步驟中擷取結果。
處理對話的接續	呼叫 <code>IDialogContext.Wait</code> 。	在瀑布式對話中新增其他步驟，或實作 <code>Dialog.ContinueDialogAsync</code>

作業	V3	V4
表示在使用者的下一則訊息前處理結束	呼叫 <code>IDialogContext.Wait</code> 。	傳回 <code>Dialog.EndOfTurn</code> 。
開始子對話	呼叫 <code>IDialogContext.Call</code> 。	傳回等候步驟內容的 <code>BeginDialogAsync</code> 方法。 如果子對話傳回值，該值可透過步驟內容的 <code>Result</code> 屬性用於瀑布的下一個步驟。
以新對話取代目前的對話	呼叫 <code>IDialogContext.Forward</code> 。	傳回等候 <code>ITurnContext.ReplaceDialogAsync</code> 。
表示目前對話已完成的訊號	呼叫 <code>IDialogContext.Done</code> 。	傳回等候步驟內容的 <code>EndDialogAsync</code> 方法。
對話失敗。	呼叫 <code>IDialogContext.Fail</code> 。	擲回要在 Bot 的另一個層級攔截的例外狀況、結束狀態為 <code>Cancelled</code> 的步驟，或呼叫該步驟或對話內容的 <code>CancelAllDialogsAsync</code> 。 請注意，在 v4 中對話內的例外狀況會隨著 C# 堆疊傳播，而不是隨著對話堆疊傳播。

v4 程式碼的其他注意事項：

- v4 中的各種 `Prompt` 衍生類別會以包含兩個步驟的個別對話形式實作使用者提示。請參閱如何實作循序對話流程。
- 使用 `DialogSet.CreateContextAsync` 建立目前回合的對話內容。
- 在對話內，使用 `DialogContext.Context` 屬性來取得目前的回合內容。
- 瀑布步驟具有 `WaterfallStepContext` 參數，其衍生自 `DialogContext`。
- 所有具體的對話和提示類別均衍生自抽象的 `Dialog` 類別。
- 您會在建立元件對話時指派識別碼。對話集中的每個對話都需要被指派該集合內唯一的識別碼。

在對話之間和之內傳遞狀態

Dialogs 程式庫一文的對話狀態、瀑布步驟內容屬性和使用對話章節說明如何在 v4 中管理對話狀態。

IAwaitable 已不存在

若要取得使用者在回合中的活動，請從回合內容中取得。

若要提示使用者並接收提示的結果：

- 將適當的提示執行個體新增到對話集。
- 從瀑布式對話中的步驟呼叫提示。
- 從下一個步驟中步驟內容的 `Result` 屬性擷取結果。

Formflow

在 V3，Formflow 是 C# SDK 的一部分，但不屬於 JavaScript SDK。其不是 v4 SDK 的一部分，但有適用於 C# 的社群版本。

NUGET 套件名稱	社群 GITHUB 存放庫
<code>Bot.Builder.Community.Dialogs.Formflow</code>	BotBuilderCommunity/botbuilder-community-dotnet/libraries/Bot.Builder.Community.Dialogs.FormFlow

其他資源

- [將 .NET SDK v3 bot 遷移至 v4](#)

將 .NET SDK v3 Bot 遷移至 v4

2019/5/10 • [Edit Online](#)

我們在本文中將 v3 [ContosoHelpdeskChatBot](#) 轉換 v4 Bot，而「不需轉換專案類型」。其仍然是 .NET Framework 專案。此轉換可細分成下列步驟：

1. 更新和安裝 NuGet 套件
2. 更新 Global.asax.cs 檔案
3. 更新 MessagesController 類別
4. 轉換您的對話

Bot Framework SDK v4 是以與 SDK v3 相同的基礎 REST API 作為基礎。不過，SDK v4 是舊版 SDK 的重構，讓開發人員對於其 Bot 有更多的彈性和控制權。SDK 中的主要變更包括：

- 透過狀態管理物件和屬性存取子管理狀態。
- 設定回合處理常式並將活動傳遞給它已變更。
- 可評分項目已不存在。將控制權交給您的對話之前，可以檢查回合處理常式中的「全域」命令。
- 與前一版非常不同的新 Dialogs 程式庫。您必須使用元件和瀑布式對話，以及適用於 v4 的 Formflow 對話社群實作，將舊的對話轉換至新的對話系統。

如需特定變更的詳細資訊，請參閱 [v3 和 v4 .NET SDK 之間差異](#)。

更新和安裝 NuGet 套件

1. 將 **Microsoft.Bot.Builder.Azure** 更新為最新的穩定版本。

這將也會更新 **Microsoft.Bot.Builder** 和 **Microsoft.Bot.Connector** 套件，因為它們是相依項目。

2. 刪除 **Microsoft.Bot.Builder.History** 套件。這不是 v4 SDK 的一部分。
3. 新增 **Autofac.WebApi2**

我們將使用它來協助在 ASP.NET 中插入相依性。

4. 新增 **Bot.Builder.Community.Dialogs.Formflow**

這是一個社群程式庫，用於從 v3 Formflow 定義檔建置 v4 對話。它以 **Microsoft.Bot.Builder.Dialogs** 作為其相依性之一，因此系統也會為我們安裝。

如果您在此時建置，則會收到編譯器錯誤。您可以忽略這些錯誤。完成轉換之後，我們就會有工作程式碼。

更新 Global.asax.cs 檔案

某些 Scaffolding 已變更，而且我們必須自行在 v4 中設定部份的**狀態管理**基礎結構。例如，v4 會使用 Bot 配接器來處理驗證並將活動轉送到 Bot 程式碼，而我們會事先宣告我們的狀態屬性。

我們將建立 `DialogState` 的狀態屬性，我們現在需要該屬性才能在 v4 中支援對話。我們將使用相依性插入來取得控制器和 Bot 程式碼的必要資訊。

在 **Global.asax.cs** 中：

1. 更新 `using` 陳述式：

```
using System.Configuration;
using System.Reflection;
using System.Web.Http;
using Autofac;
using Autofac.Integration.WebApi;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Connector.Authentication;
```

2. 從 Application_Start 方法中移除這幾行

```
BotConfig.UpdateConversationContainer();
this.RegisterBotModules();
```

此外，插入這一行。

```
GlobalConfiguration.Configure(BotConfig.Register);
```

3. 移除不再參考的 RegisterBotModules 方法。

4. 以此 BotConfig.Register 方法取代 BotConfig.UpdateConversationContainer 方法，我們將在其中註冊支援相依性插入所需的物件。

NOTE

此 Bot 並未使用「使用者」或「私人交談」狀態。要包含這些項目的程式碼行會在這裡註解排除。

```
public static void Register(HttpConfiguration config)
{
    ContainerBuilder builder = new ContainerBuilder();
    builder.RegisterApiControllers(Assembly.GetExecutingAssembly());

    SimpleCredentialProvider credentialProvider = new SimpleCredentialProvider(
        ConfigurationManager.AppSettings[MicrosoftAppCredentials.MicrosoftAppIdKey],
        ConfigurationManager.AppSettings[MicrosoftAppCredentials.MicrosoftAppPasswordKey]);

    builder.RegisterInstance(credentialProvider).As<ICredentialProvider>();

    // The Memory Storage used here is for local bot debugging only. When the bot
    // is restarted, everything stored in memory will be gone.
    IStorage dataStore = new MemoryStorage();

    // Create Conversation State object.
    // The Conversation State object is where we persist anything at the conversation-scope.
    ConversationState conversationState = new ConversationState(dataStore);
    builder.RegisterInstance(conversationState).As<ConversationState>();

    //var userState = new UserState(dataStore);
    //var privateConversationState = new PrivateConversationState(dataStore);

    // Create the dialog state property acccessor.
    IStatePropertyAccessor<DialogState> dialogStateAccessor
        = conversationState.CreateProperty<DialogState>(nameof(DialogState));
    builder.RegisterInstance(dialogStateAccessor).As<IStatePropertyAccessor<DialogState>>();

   .IContainer container = builder.Build();
    AutofacWebApiDependencyResolver resolver = new AutofacWebApiDependencyResolver(container);
    config.DependencyResolver = resolver;
}
```

更新 MessagesController 類別

這是 v4 中發生回合處理常式的地方，因此需求大幅變更。除了回合處理常式本身，可將大部分的內容視為重複使用文字。在 **Controllers\MessagesController.cs** 檔案中：

1. 更新 `using` 陳述式：

```
using System;
using System.Net;
using System.Net.Http;
using System.Threading;
using System.Threading.Tasks;
using System.Web.Http;
using Bot.Builder.Community.Dialogs.FormFlow;
using ContosoHelpdeskChatBot.Dialogs;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Connector.Authentication;
using Microsoft.Bot.Schema;
```

2. 從類別中移除 `[BotAuthentication]` 屬性。在 v4 中，Bot 的配接器會處理驗證。
3. 新增這些欄位。**ConversationState** 會管理交談的狀態，而需要 **IStatePropertyAccessor<DialogState>** 才能在 v4 中支援對話。

```
private readonly ConversationState _conversationState;
private readonly ICredentialProvider _credentialProvider;
private readonly IStatePropertyAccessor<DialogState> _dialogData;

private readonly DialogSet _dialogs;
```

4. 將建構函式新增至：

- 初始化執行個體欄位。
- 在 ASP.NET 中使用相依性插入來取得參數值。(我們在 **Global.asax.cs** 中註冊了這些類別的執行個體，以支援此功能。)
- 建立並初始化一個對話集，以便從中建立對話內容。(我們必須在 v4 中明確執行這項操作。)

```
public MessagesController(
    ConversationState conversationState,
    ICredentialProvider credentialProvider,
    IStatePropertyAccessor<DialogState> dialogData)
{
    _conversationState = conversationState;
    _dialogData = dialogData;
    _credentialProvider = credentialProvider;

    _dialogs = new DialogSet(dialogData);
    _dialogs.Add(new RootDialog(nameof(RootDialog)));
}
```

5. 取代 **Post** 方法的主體。這是我們建立配接器並使用它來呼叫訊息迴圈的地方(回合處理常式)。我們在每個回合的結尾使用 `SaveChangesAsync`，以儲存 Bot 對狀態所做的任何變更。

```
public async Task<HttpResponseMessage> Post([FromBody]Activity activity)
{
    var botFrameworkAdapter = new BotFrameworkAdapter(_credentialProvider);

    var invokeResponse = await botFrameworkAdapter.ProcessActivityAsync(
        Request.Headers.Authorization?.ToString(),
        activity,
        OnTurnAsync,
        default(CancellationToken));

    if (invokeResponse == null)
    {
        return Request.CreateResponse(HttpStatusCode.OK);
    }
    else
    {
        return Request.CreateResponse(invokeResponse.Status);
    }
}
```

6. 新增 **OnTurnAsync** 方法，其中包含 Bot 的[回合處理常式](#)程式碼。

NOTE

因此 v4 中不存在可評分項目。我們會在繼續任何作用中對話之前，先檢查 Bot 的回合處理常式中來自使用者的 `cancel` 訊息。

```

protected async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken)
{
    // We're only handling message activities in this bot.
    if (turnContext.Activity.Type == ActivityTypes.Message)
    {
        // Create the dialog context for our dialog set.
        DialogContext dc = await _dialogs.CreateContextAsync(turnContext, cancellationToken);

        // Globally interrupt the dialog stack if the user sent 'cancel'.
        if (turnContext.Activity.Text.Equals("cancel", StringComparison.InvariantCultureIgnoreCase))
        {
            Activity reply = turnContext.Activity.CreateReply($"Ok restarting conversation.");
            await turnContext.SendActivityAsync(reply);
            await dc.CancelAllDialogsAsync();
        }
    }

    try
    {
        // Continue the active dialog, if any. If we just cancelled all dialog, the
        // dialog stack will be empty, and this will return DialogTurnResult.Empty.
        DialogTurnResult dialogResult = await dc.ContinueDialogAsync();
        switch (dialogResult.Status)
        {
            case DialogTurnStatus.Empty:
                // There was no active dialog in the dialog stack; start the root dialog.
                await dc.BeginDialogAsync(nameof(RootDialog));
                break;

            case DialogTurnStatus.Complete:
                // The last dialog on the stack completed and the stack is empty.
                await dc.EndDialogAsync();
                break;

            case DialogTurnStatus.Waiting:
            case DialogTurnStatus.Cancelled:
                // The active dialog is waiting for a response from the user, or all
                // dialogs were cancelled and the stack is empty. In either case, we
                // don't need to do anything here.
                break;
        }
    }
    catch (FormCanceledException)
    {
        // One of the dialogs threw an exception to clear the dialog stack.
        await turnContext.SendActivityAsync("Cancelled.");
        await dc.CancelAllDialogsAsync();
        await dc.BeginDialogAsync(nameof(RootDialog));
    }
}
}

```

7. 因為我們只會處理「訊息」活動，所以可刪除 **HandleSystemMessage** 方法。

刪除 **Cancelable** 和 **GlobalMessageHandlersBotModule** 類別

因為 v4 中不存在可評分項目，而且我們已更新回合處理常式以回應 `cancel` 訊息，所以可刪除 **Cancelable** (在 **Dialogs\Cancelable.cs**) 和 **GlobalMessageHandlersBotModule** 類別。

轉換您的對話

我們將對原始對話進行許多變更，以將其遷移至 v4 SDK。現在不用擔心編譯器錯誤。待我們完成轉換後，就會解決這些錯誤。為了不會對原始程式碼進行不必要的修改，在我們完成移轉後，仍會有一些編譯器警告。

我們所有的對話都將衍生自 `ComponentDialog`，而不會實作 v3 的 `IDialog<object>` 介面。

此 Bot 有四個需要我們轉換的對話：

RootDialog	顯示選項並開始其他對話。
InstallAppDialog	處理在電腦上安裝應用程式的要求。
LocalAdminDialog	處理本機電腦系統管理權限的要求。
ResetPasswordDialog	處理重設密碼的要求。

這些對話會收集輸入，但不會在您的電腦上執行上述任何作業。

進行全方案的對話變更

- 針對整個方案，以 `ComponentDialog` 取代所有出現的 `IDialog<object>`。
- 針對整個方案，以 `DialogContext` 取代所有出現的 `IDialogContext`。
- 針對每個對話類別，移除 `[Serializable]` 屬性。

對話內的控制流程和傳訊不再以相同的方式處理，因此我們必須在轉換每個對話時修改這個屬性。

作業	V3 程式碼	V4 程式碼
處理對話的開始	實作 <code>IDialog.StartAsync</code>	讓此項成為瀑布式對話的第一個步驟，或實作 <code>Dialog.BeginDialogAsync</code>
處理對話的接續	呼叫 <code>IDialogContext.Wait</code>	在瀑布式對話中新增其他步驟，或實作 <code>Dialog.ContinueDialogAsync</code>
將訊息傳送給使用者	呼叫 <code>IDialogContext.PostAsync</code>	呼叫 <code>ITurnContext.SendActivityAsync</code>
開始子對話	呼叫 <code>IDialogContext.Call</code>	呼叫 <code>DialogContext.BeginDialogAsync</code>
表示目前對話已完成的訊號	呼叫 <code>IDialogContext.Done</code>	呼叫 <code>DialogContext.EndDialogAsync</code>
取得使用者的輸入	使用 <code>IAwaitable<IMessageActivity></code> 參數	使用瀑布中的提示，或使用 <code>ITurnContext.Activity</code>

v4 程式碼的注意事項：

- 使用 `DialogContext.Context` 屬性來取得目前的回合內容。
- 瀑布步驟具有 `WaterfallStepContext` 參數，其衍生自 `DialogContext`。
- 所有具體的對話和提示類別均衍生自抽象的 `Dialog` 類別。
- 您會在建立元件對話時指派識別碼。對話集中的每個對話都需要被指派該集合內唯一的識別碼。

更新根對話

在此 Bot 中，根對話會提示使用者從一組選項中進行選擇，然後根據該選擇開始進行子對話。接著在交談存留期間執行迴圈。

- 我們可以將主要流程設定為瀑布式對話，這是 v4 SDK 中的新概念。它會依序執行一組固定的步驟。如需詳細資訊，請參閱[實作循序交談流程](#)。
- 提示作業現在會透過提示類別來處理，而這些類別是簡短的子對話，可提示輸入資料、執行最少的處理和驗證，並傳回一個值。如需詳細資訊，請參閱[使用對話提示收集使用者輸入](#)。

在 **Dialogs/RootDialog.cs** 檔案中：

1. 更新 `using` 陳述式：

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.Dialogs.Choices;
```

2. 我們需要將 `HelpdeskOptions` 選項從字串清單轉換為選擇清單。這將搭配選擇提示使用，其將接受選擇號碼（清單中）、選擇值，或任何選擇的同義字作為有效的輸入。

```
private static List<Choice> HelpdeskOptions = new List<Choice>()
{
    new Choice(InstallAppOption) { Synonyms = new List<string>(){ "install" } },
    new Choice(ResetPasswordOption) { Synonyms = new List<string>(){ "password" } },
    new Choice(LocalAdminOption) { Synonyms = new List<string>(){ "admin" } }
};
```

3. 新增建構函式。此程式碼會執行以下動作：

- 每個對話執行個體會在建立時被指派識別碼。對話識別碼是對話要新增到其中的對話集的一部分。還記得 Bot 有一個已在 **MessageController** 類別中初始化的對話集。每個 `ComponentDialog` 都有自己的內部對話集，以及自己的對話識別碼集。
- 它會新增其他對話（包括選擇提示）作為子對話。在此，我們只會使用每個對話識別碼的類別名稱。
- 它會定義包含三個步驟的瀑布式對話。我們會立刻進行實作。
 - 對話會先提示使用者選擇要執行的工作。
 - 然後，開始與該選擇相關聯的子對話。
 - 最後，自行重新開始。
- 瀑布的每個步驟都是一項委派，而我們接下來會實作這些步驟，並從原始對話中取得現有的程式碼。
- 當您啟動元件對話時，就會啟動其「初始對話」。根據預設，這是新增至元件對話的第一個子對話。若要指派不同的子系作為初始對話，您會手動設定元件的 `InitialDialogId` 屬性。

```
public RootDialog(string id)
    : base(id)
{
    AddDialog(new WaterfallDialog("choiceswaterfall", new WaterfallStep[]
    {
        PromptForOptionsAsync,
        ShowChildDialogAsync,
        ResumeAfterAsync,
    }));
    AddDialog(new InstallAppDialog(nameof(InstallAppDialog)));
    AddDialog(new LocalAdminDialog(nameof(LocalAdminDialog)));
    AddDialog(new ResetPasswordDialog(nameof(ResetPasswordDialog)));
    AddDialog(new ChoicePrompt("options"));
}
```

4. 我們可以刪除 **StartAsync** 方法。當元件對話開始時，它會自動開始它「初始」對話。在此情況下，這就是我們在建構函式中定義的瀑布式對話。該對話也會自動在其第一個步驟開始。
5. 我們將會刪除 **MessageReceivedAsync** 和 **ShowOptions** 方法，並以瀑布的第一個步驟取代它們。這兩種方法會先映入使用者的眼簾，並要求他們選擇其中一個可用的選項。
 - 您可以在此看到選擇清單，而系統會提供問候和錯誤訊息作為我們的選擇提示呼叫中的選項。
 - 我們不需要指定要在對話中呼叫的下一個方法，因為瀑布會在選擇提示完成時繼續下一個步驟。

- 選擇提示將會執行迴圈，直到它收到有效的輸入，或取消整個對話堆疊為止。

```
private async Task<DialogTurnResult> PromptForOptionsAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Prompt the user for a response using our choice prompt.
    return await stepContext.PromptAsync(
        "options",
        new PromptOptions()
    {
        Choices = HelpdeskOptions,
        Prompt = MessageFactory.Text(GreetMessage),
        RetryPrompt = MessageFactory.Text(ErrorMessage)
    },
    cancellationToken);
}
```

6. 我們可以使用瀑布的第二個步驟取代 **OnOptionSelected**。我們仍會根據使用者的輸入開始子對話。

- 選擇提示會傳回 `FoundChoice` 值。這會顯示在步驟內容的 `Result` 屬性中。對話堆疊會將所有傳回值視為物件。如果傳回值來自您的其中一個對話，您便知道物件值是何種類型。如需每個提示類型傳回的內容清單，請參閱[提示類型](#)。
- 因為選擇提示不會擲回例外狀況，所以可移除 try-catch 區塊。
- 我們必須新增一項通過，此方法才能一律傳回適當的值。此程式碼應該永遠不會被叫用，但如果叫用，就會讓對話「正常失敗」。

```
private async Task<DialogTurnResult> ShowChildDialogAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // string optionSelected = await userReply;
    string optionSelected = (stepContext.Result as FoundChoice).Value;

    switch (optionSelected)
    {
        case InstallAppOption:
            //context.Call(new InstallAppDialog(), this.ResumeAfterOptionDialog);
            //break;
            return await stepContext.BeginDialogAsync(
                nameof(InstallAppDialog),
                cancellationToken);
        case ResetPasswordOption:
            //context.Call(new ResetPasswordDialog(), this.ResumeAfterOptionDialog);
            //break;
            return await stepContext.BeginDialogAsync(
                nameof(ResetPasswordDialog),
                cancellationToken);
        case LocalAdminOption:
            //context.Call(new LocalAdminDialog(), this.ResumeAfterOptionDialog);
            //break;
            return await stepContext.BeginDialogAsync(
                nameof(LocalAdminDialog),
                cancellationToken);
    }

    // We shouldn't get here, but fail gracefully if we do.
    await stepContext.Context.SendActivityAsync(
        "I don't recognize that option.",
        cancellationToken: cancellationToken);
    // Continue through to the next step without starting a child dialog.
    return await stepContext.NextAsync(cancellationToken: cancellationToken);
}
```

7. 最後，使用瀑布的最後一個步驟取代舊的 **ResumeAfterOptionDialog** 方法。

- 我們會將堆疊上的原始執行個體取代為本身的新執行個體，進而重新開始瀑布式對話，而不是如同我們在原始對話中一樣結束對話並傳回票證號碼。我們可以這麼做，因為原始應用程式一律忽略傳回值（票證號碼），並重新開始進行根對話。

```
private async Task<DialogTurnResult> ResumeAfterAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    try
    {
        //var message = await userReply;
        var message = stepContext.Context.Activity;

        int ticketNumber = new Random().Next(0, 20000);
        //await context.PostAsync($"Thank you for using the Helpdesk Bot. Your ticket number is {ticketNumber}.");
        await stepContext.Context.SendActivityAsync(
            $"Thank you for using the Helpdesk Bot. Your ticket number is {ticketNumber}.",
            cancellationToken: cancellationToken);

        //context.Done(ticketNumber);
    }
    catch (Exception ex)
    {
        // await context.PostAsync($"Failed with message: {ex.Message}");
        await stepContext.Context.SendActivityAsync(
            $"Failed with message: {ex.Message}",
            cancellationToken: cancellationToken);

        // In general resume from task after calling a child dialog is a good place to handle exceptions
        // try catch will capture exceptions from the bot framework awaitable object which is
        // essentially "userReply"
        logger.Error(ex);
    }

    // Replace on the stack the current instance of the waterfall with a new instance,
    // and start from the top.
    return await stepContext.ReplaceDialogAsync(
        "choiceswaterfall",
        cancellationToken: cancellationToken);
}
```

更新安裝應用程式對話

安裝應用程式對話會執行一些邏輯工作，我們會將其設定為包含 4 個步驟的瀑布式對話。如何將現有程式碼分解成瀑布步驟是每個對話的邏輯活動。每個步驟都會註明程式碼來自的原始方法。

1. 要求使用者提供搜尋字串。
2. 查詢資料庫中可能的相符項目。
 - 如果有一項命中，請選取此項目並繼續執行。
 - 如果有多項命中，則會要求使用者選擇一項。
 - 如果沒有命中項目，就會結束對話。
3. 要求使用者提供要安裝應用程式的機器。
4. 將資訊寫入資料庫並傳送確認訊息。

在 **Dialogs/InstallAppDialog.cs** 檔案中：

1. 更新 `using` 陳述式：

```
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using ContosoHelpdeskChatBot.Models;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.Dialogs.Choices;
```

2. 針對我們將用於提示和對話的識別碼，定義常數。這可讓對話程式碼更容易維護，因為要使用的字串會定義於一個地方。

```
// Set up our dialog and prompt IDs as constants.
private const string MainId = "mainDialog";
private const string TextId = "textPrompt";
private const string ChoiceId = "choicePrompt";
```

3. 針對我們將用來追蹤對話狀態的索引鍵，定義常數。

```
// Set up keys for managing collected information.
private const string InstallInfo = "installInfo";
```

4. 新增建構函式並初始化元件的對話集。我們此時會明確地設定 `InitialDialogId` 屬性，這表示主要瀑布式對話不必是您新增至對話集的第一個對話。比方說，如果您想要先新增提示，這可讓您執行這項操作，而不會造成執行階段問題。

```
public InstallAppDialog(string id)
    : base(id)
{
    // Initialize our dialogs and prompts.
    InitialDialogId = MainId;
    AddDialog(new WaterfallDialog(MainId, new WaterfallStep[] {
        GetSearchTermAsync,
        ResolveAppNameAsync,
        GetMachineNameAsync,
        SubmitRequestAsync,
    }));
    AddDialog(new TextPrompt(TextId));
    AddDialog(new ChoicePrompt(ChoiceId));
}
```

5. 我們可以使用瀑布的第一個步驟取代 `StartAsync`。

- 我們不必自行管理狀態，所以會追蹤對話狀態中的安裝應用程式物件。
- 要求使用者輸入資料的訊息會變成提示呼叫中的選項。

```
private async Task<DialogTurnResult> GetSearchTermAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Create an object in dialog state in which to track our collected information.
    stepContext.Values[InstallInfo] = new InstallApp();

    // Ask for the search term.
    return await stepContext.PromptAsync(
        TextId,
        new PromptOptions
        {
            Prompt = MessageFactory.Text("Ok let's get started. What is the name of the application? "),
        },
        cancellationToken);
}
```

6. 我們可以使用瀑布的第二個步驟取代 **appNameAsync** 和 **multipleAppsAsync**。

- 我們現在會取得提示結果，而不只是查看使用者的最後一則訊息。
- 資料庫查詢和 if 陳述式的組織方式與在 **appNameAsync** 中相同。已更新 if 陳述式的每個區塊中的程式碼，可搭配 v4 對話運作。
 - 如果我們有一次命中，我們將會更新對話狀態並繼續進行下一個步驟。
 - 如果您有多項命中，我們將使用選擇提示來要求使用者從選項清單中進行選擇。這表示我們可以刪除 **multipleAppsAsync**。
 - 如果我們沒有命中，我們會結束此對話並傳回 null 紿根對話。

```

private async Task<DialogTurnResult> ResolveAppNameAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Get the result from the text prompt.
    var appname = stepContext.Result as string;

    // Query the database for matches.
    var names = await this.getAppsAsync(appname);

    if (names.Count == 1)
    {
        // Get our tracking information from dialog state and add the app name.
        var install = stepContext.Values[InstallInfo] as InstallApp;
        install.AppName = names.First();

        return await stepContext.NextAsync();
    }
    else if (names.Count > 1)
    {
        // Ask the user to choose from the list of matches.
        return await stepContext.PromptAsync(
            ChoiceId,
            new PromptOptions
            {
                Prompt = MessageFactory.Text("I found the following applications. Please choose one:"),
                Choices = ChoiceFactory.ToChoices(names),
            },
            cancellationToken);
    }
    else
    {
        // If no matches, exit this dialog.
        await stepContext.Context.SendActivityAsync(
            $"Sorry, I did not find any application with the name '{appname}'.",
            cancellationToken: cancellationToken);

        return await stepContext.EndDialogAsync(null, cancellationToken);
    }
}

```

7. **appNameAsync** 也在解析查詢之後，要求使用者提供其機器名稱。我們將在瀑布的下一個步驟中擷取邏輯的該部分。

- 同樣地，在 v4 中，我們必須自行管理狀態。唯一比較麻煩的事，就是我們可以透過上一個步驟中的兩個不同邏輯分支來抵達此步驟。
- 我們會使用與之前相同的文字提示來要求使用者提供機器名稱，只是這次提供不同的選項。

```

private async Task<DialogTurnResult> GetMachineNameAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Get the tracking info. If we don't already have an app name,
    // Then we used the choice prompt to get it in the previous step.
    var install = stepContext.Values[InstallInfo] as InstallApp;
    if (install.AppName is null)
    {
        install.AppName = (stepContext.Result as FoundChoice).Value;
    }

    // We now need the machine name, so prompt for it.
    return await stepContext.PromptAsync(
        TextId,
        new PromptOptions
        {
            Prompt = MessageFactory.Text(
                $"Found {install.AppName}. What is the name of the machine to install application?"),
            cancellationToken);
}

```

8. **machineNameAsync** 中的邏輯會包裝在瀑布的最後一個步驟中。

- 我們可從文字提示結果中擷取機器名稱並更新對話狀態。
- 我們正在移除呼叫以更新資料庫，因為支援的程式碼位於不同的專案中。
- 然後我們會將成功訊息傳送給使用者並結束對話。

```

private async Task<DialogTurnResult> SubmitRequestAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    if (stepContext.Reason != DialogReason.CancelCalled)
    {
        // Get the tracking info and add the machine name.
        var install = stepContext.Values[InstallInfo] as InstallApp;
        install.MachineName = stepContext.Context.Activity.Text;

        //TODO: Save to this information to the database.
    }

    await stepContext.Context.SendActivityAsync(
        $"Great, your request to install {install.AppName} on {install.MachineName} has been
scheduled.",
        cancellationToken: cancellationToken);

    return await stepContext.EndDialogAsync(null, cancellationToken);
}

```

9. 為了模擬資料庫，我已經將 **getAppsAsync** 更新為查詢靜態清單，而不是查詢資料庫。

```

private async Task<List<string>> getAppsAsync(string Name)
{
    List<string> names = new List<string>();

    // Simulate querying the database for applications that match.
    return (from app in AppMsis
            where app.ToLower().Contains(Name.ToLower())
            select app).ToList();
}

// Example list of app names in the database.
private static readonly List<string> AppMsis = new List<string>
{
    "μTorrent 3.5.0.44178",
    "7-Zip 17.1",
    "Ad-Aware 9.0",
    "Adobe AIR 2.5.1.17730",
    "Adobe Flash Player (IE) 28.0.0.105",
    "Adobe Flash Player (Non-IE) 27.0.0.130",
    "Adobe Reader 11.0.14",
    "Adobe Shockwave Player 12.3.1.201",
    "Advanced SystemCare Personal 11.0.3",
    "Auslogics Disk Defrag 3.6",
    "avast! 4 Home Edition 4.8.1351",
    "AVG Anti-Virus Free Edition 9.0.0.698",
    "Bonjour 3.1.0.1",
    "CCleaner 5.24.5839",
    "Chmod Calculator 20132.4",
    "CyberLink PowerDVD 17.0.2101.62",
    "DAEMON Tools Lite 4.46.1.328",
    "FileZilla Client 3.5",
    "Firefox 57.0",
    "Foxit Reader 4.1.1.805",
    "Google Chrome 66.143.49260",
    "Google Earth 7.3.0.3832",
    "Google Toolbar (IE) 7.5.8231.2252",
    "GSpot 2701.0",
    "Internet Explorer 903235.0",
    "iTunes 12.7.0.166",
    "Java Runtime Environment 6 Update 17",
    "K-Lite Codec Pack 12.1",
    "Malwarebytes Anti-Malware 2.2.1.1043",
    "Media Player Classic 6.4.9.0",
    "Microsoft Silverlight 5.1.50907",
    "Mozilla Thunderbird 57.0",
    "Nero Burning ROM 19.1.1005",
    "OpenOffice.org 3.1.1 Build 9420",
    "Opera 12.18.1873",
    "Paint.NET 4.0.19",
    "Picasa 3.9.141.259",
    "QuickTime 7.79.80.95",
    "RealPlayer SP 12.0.0.319",
    "Revo Uninstaller 1.95",
    "Skype 7.40.151",
    "Spybot - Search & Destroy 1.6.2.46",
    "SpywareBlaster 4.6",
    "TuneUp Utilities 2009 14.0.1000.353",
    "Unlocker 1.9.2",
    "VLC media player 1.1.6",
    "Winamp 5.56 Build 2512",
    "Windows Live Messenger 2009 16.4.3528.331",
    "WinPatrol 2010 31.0.2014",
    "WinRAR 5.0",
};


```

在 v3 中，此對話會先映入使用者的眼簾、開始 Formflow 對話，然後將結果儲存至資料庫。這可輕鬆地轉換成包含兩個步驟的瀑布。

1. 更新 `using` 陳述式。請注意，此對話包含 v3 Formflow 對話。在 v4 中，我們可以使用社群 Formflow 程式庫。

```
using System.Threading;
using System.Threading.Tasks;
using Bot.Builder.Community.Dialogs.FormFlow;
using ContosoHelpdeskChatBot.Models;
using Microsoft.Bot.Builder.Dialogs;
```

2. 我們可以移除 `LocalAdmin` 的執行個體屬性，因為可在對話狀態中取得結果。

3. 針對我們將用於對話的識別碼，定義常數。在社群程式庫中，建構的對話一律會設定為對話所產生的類別名稱。

```
// Set up our dialog and prompt IDs as constants.
private const string MainDialog = "mainDialog";
private static string AdminDialog { get; } = nameof(LocalAdminPrompt);
```

4. 新增建構函式並初始化元件的對話集。Formflow 對話會以相同的方式建立。我們只是將它新增至建構函式中元件的對話集。

```
public LocalAdminDialog(string dialogId) : base(dialogId)
{
    InitialDialogId = MainDialog;

    AddDialog(new WaterfallDialog(MainDialog, new WaterfallStep[]
    {
        BeginFormflowAsync,
        SaveResultAsync,
    }));
    AddDialog(FormDialog.FromForm(BuildLocalAdminForm, FormOptions.PromptInStart));
}
```

5. 我們可以使用瀑布的第一個步驟取代 **StartAsync**。我們已經在建構函式中建立 Formflow，而其他兩個陳述式會轉譯為此陳述式。

```
private async Task<DialogTurnResult> BeginFormflowAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    await stepContext.Context.SendActivityAsync("Great I will help you request local machine admin.");

    // Begin the Formflow dialog.
    return await stepContext.BeginDialogAsync(AdminDialog, cancellationToken: cancellationToken);
}
```

6. 我們可以使用瀑布的第二個步驟取代 **ResumeAfterLocalAdminFormDialog**。我們必須從步驟內容中取得傳回值，而不是從執行個體屬性中取得。

```

private async Task<DialogTurnResult> SaveResultAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Get the result from the Formflow dialog when it ends.
    if (stepContext.Reason != DialogReason.CancelCalled)
    {
        var admin = stepContext.Result as LocalAdminPrompt;

        //TODO: Save to this information to the database.
    }

    return await stepContext.EndDialogAsync(null, cancellationToken);
}

```

7. BuildLocalAdminForm 大致維持相同，但我們沒有讓 Formflow 更新執行個體屬性。

```

// Nearly the same as before.
private IForm<LocalAdminPrompt> BuildLocalAdminForm()
{
    //here's an example of how validation can be used in form builder
    return new FormBuilder<LocalAdminPrompt>()
        .Field(nameof(LocalAdminPrompt.MachineName),
            validate: async (state, value) =>
        {
            ValidateResult result = new ValidateResult { IsValid = true, Value = value };
            //add validation here

            //this.admin.MachineName = (string)value;
            return result;
        })
        .Field(nameof(LocalAdminPrompt.AdminDuration),
            validate: async (state, value) =>
        {
            ValidateResult result = new ValidateResult { IsValid = true, Value = value };
            //add validation here

            //this.admin.AdminDuration = Convert.ToInt32((long)value) as int?;
            return result;
        })
        .Build();
}

```

更新重設密碼對話

在 v3 中，此對話會先映入使用者的眼簾、透過密碼授權給使用者、結束或開始 Formflow 對話，然後重設密碼。這仍會轉譯成瀑布。

1. 更新 `using` 陳述式。請注意，此對話包含 v3 Formflow 對話。在 v4 中，我們可以使用社群 Formflow 程式庫。

```

using System;
using System.Threading;
using System.Threading.Tasks;
using Bot.Builder.Community.Dialogs.FormFlow;
using ContosoHelpdeskChatBot.Models;
using Microsoft.Bot.Builder.Dialogs;

```

2. 針對我們將用於對話的識別碼，定義常數。在社群程式庫中，建構的對話一律會設定為對話所產生的類別名稱。

```
// Set up our dialog and prompt IDs as constants.  
private const string MainDialog = "mainDialog";  
private static string ResetDialog { get; } = nameof(ResetPasswordPrompt);
```

3. 新增建構函式並初始化元件的對話集。Formflow 對話會以相同的方式建立。我們只是將它新增至建構函式中元件的對話集。

```
public ResetPasswordDialog(string dialogId) : base(dialogId)  
{  
    InitialDialogId = MainDialog;  
  
    AddDialog(new WaterfallDialog(MainDialog, new WaterfallStep[]  
    {  
        BeginFormflowAsync,  
        ProcessRequestAsync,  
    }));  
    AddDialog(FormDialog.FromForm(BuildResetPasswordForm, FormOptions.PromptInStart));  
}
```

4. 我們可以使用瀑布的第一個步驟取代 **StartAsync**。我們已經在建構函式中建立 Formflow。否則，我們會保留相同的邏輯，只是將 v3 呼叫轉譯成 v4 對等項目。

```
private async Task<DialogTurnResult> BeginFormflowAsync(  
    WaterfallStepContext stepContext,  
    CancellationToken cancellationToken = default(CancellationToken))  
{  
    await stepContext.Context.SendActivityAsync("Alright I will help you create a temp password.");  
  
    // Check the passcode and fail out or begin the Formflow dialog.  
    if (sendPassCode(stepContext))  
    {  
        return await stepContext.BeginDialogAsync(ResetDialog, cancellationToken: cancellationToken);  
    }  
    else  
    {  
        //here we can simply fail the current dialog because we have root dialog handling all exceptions  
        throw new Exception("Failed to send SMS. Make sure email & phone number has been added to  
database.");  
    }  
}
```

5. **sendPassCode** 主要留作練習。已將原始程式碼註解排除，而此方法只會傳回 true。此外，原始 Bot 中並未使用電子郵件地址，所以可再次予以移除。

```

private bool sendPassCode(DialogContext context)
{
    //bool result = false;

    //Recipient Id varies depending on channel
    //refer ChannelAccount class https://docs.botframework.com/en-
    us/csharp/builder/sdkreference/dd/def/class_microsoft_1_1_bot_1_1_connector_1_1_channel_account.html#a0b
    89cf01fdd73cbc00a524dce9e2ad1a
    //as well as Activity class https://docs.botframework.com/en-
    us/csharp/builder/sdkreference/dc/d2f/class_microsoft_1_1_bot_1_1_connector_1_1_activity.html
    //int passcode = new Random().Next(1000, 9999);
    //Int64? smsNumber = 0;
    //string smsMessage = "Your Contoso Pass Code is ";
    //string countryDialPrefix = "+1";

    // TODO: save PassCode to database
    //using (var db = new ContosoHelpdeskContext())
    //{
    //    var reset = db.ResetPasswords.Where(r => r.EmailAddress == email).ToList();
    //    if (reset.Count >= 1)
    //    {
    //        reset.First().PassCode = passcode;
    //        smsNumber = reset.First().MobileNumber;
    //        result = true;
    //    }

    //    db.SaveChanges();
    //}

    // TODO: send passcode to user via SMS.
    //if (result)
    //{
    //    result = Helper.SendSms($"{countryDialPrefix}{smsNumber.ToString()}", $"{smsMessage}
    {passcode}");
    //}

    //return result;
    return true;
}

```

6. **BuildResetPasswordForm** 沒有變更。

7. 我們可以使用瀑布的第二個步驟取代 **ResumeAfterLocalAdminFormDialog**, 並將從步驟內容中取得傳回值。我們已移除原始對話未做任何處理的電子郵件地址，並提供了虛擬結果，而非查詢資料庫。我們會保留相同的邏輯，只是將 v3 呼叫轉譯成 v4 對等項目。

```

private async Task<DialogTurnResult> ProcessRequestAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Get the result from the Formflow dialog when it ends.
    if (stepContext.Result != null)
    {
        var prompt = stepContext.Result as ResetPasswordPrompt;
        int? passcode;

        // TODO: Retrieve the passcode from the database.
        passcode = 1111;

        if (prompt.PassCode == passcode)
        {
            string temppwd = "TempPwd" + new Random().Next(0, 5000);
            await stepContext.Context.SendActivityAsync(
                $"Your temp password is {temppwd}",
                cancellationToken: cancellationToken);
        }
    }

    return await stepContext.EndDialogAsync(null, cancellationToken);
}

```

視需要更新模型

在某些參考 Formflow 程式庫的模型中，我們需要更新 `using` 陳述式。

1. 在 `LocalAdminPrompt` 中，將它們變更如下：

```
using Bot.Builder.Community.Dialogs.FormFlow;
```

2. 在 `ResetPasswordPrompt` 中，將它們變更如下：

```
using Bot.Builder.Community.Dialogs.FormFlow;
using System;
```

更新 Web.config

註解排除 `MicrosoftAppId` 和 `MicrosoftAppPassword` 的組態金鑰。這可讓您在本機進行 Bot 偵錯，而不需將這些值提供給模擬器。

在模擬器中執行並測試您的 Bot

此時，我們應該能夠在 IIS 中本機執行 Bot，然後將它與模擬器連結。

1. 在 IIS 中執行 Bot。
2. 啟動模擬器，然後連線到 Bot 的端點（例如：<http://localhost:3978/api/messages>）。
 - 如果這是您第一次執行 Bot，請按一下 [檔案] > [新的 Bot]，並遵循畫面上的指示。否則，請按一下 [檔案] > [開啟 Bot] 以開啟現有的 Bot。
 - 在組態中再次檢查連接埠設定。例如，如果 Bot 在瀏覽器中開啟至 <http://localhost:3979/>，則在模擬器中，將 Bot 的端點設定為 <http://localhost:3979/api/messages>。
3. 四個對話應該都可以運作，而且您可以在瀑布步驟中設定中斷點，以檢查哪個對話內容和對話狀態是在這些點上。

其他資源

v4 概念性主題:

- Bot 的運作方式
- 管理狀態
- 對話方塊程式庫

v4 作法主題:

- 傳送及接收文字訊息
- 儲存使用者和對話資料
- 實作循序對話流程
- 使用對話方塊提示收集使用者輸入

Bot Framework REST API

2019/2/28 • [Edit Online](#)

Bot Framework REST API 可讓您建置 Bot，以便與 [Bot Framework 入口網站](#) 中設定的通道交換訊息、儲存和擷取狀態資料，以及將自己的用戶端應用程式連線到 Bot。所有 Bot Framework 服務皆透過 HTTPS 使用業界標準的 REST 和 JSON。

建置 Bot

您可以使用 Bot 連接器服務與 Bot Framework 入口網站中設定的通道交換訊息，藉以使用任何程式設計語言建立 Bot。

TIP

Bot Framework 提供可用於在 C# 或 Node.js 中建置 Bot 的用戶端程式庫。若要使用 C# 建置 Bot，請使用[適用於 C# 的 Bot Framework SDK](#)。若要使用 Node.js 建置 Bot，請使用[適用於 Node.js 的 Bot Framework SDK](#)。

若要深入了解如何使用 Bot 連接器服務建置 Bot，請參閱[重要概念](#)。

建置用戶端

您可以使用 Direct Line API，讓自己的用戶端應用程式與 Bot 進行通訊。Direct Line API 可實作使用標準密碼/權杖模式，並提供穩定結構描述的驗證機制，即使您的 Bot 會變更其通訊協定版本也一樣。若要深入了解如何使用 Direct Line API 來啟用用戶端與 Bot 之間的通訊，請參閱[重要概念](#)。

重要概念

2019/2/28 • [Edit Online](#)

您可以使用 Bot 連接器服務和 Bot 狀態服務跨多個頻道 (例如 Skype、電子郵件、Slack 等等) 和使用者通訊。此文
章介紹 Bot 連接器服務和 Bot 狀態服務中的重要概念。

IMPORTANT

建議您不要將 Bot Framework 狀態服務 API 用於生產環境，未來版本可能會將它淘汰。建議更新您的 Bot 程式碼以使用記憶
體內部儲存體進行測試，或將其中一個 Azure 擴充功能用於生產環境 Bot。如需詳細資訊，請參閱 [.NET](#) 或 [Node](#) 實作的〈管
理狀態資料〉主題。

Bot 連接器服務

Bot 連接器服務可讓您的 Bot 和在 [Bot Framework 入口網站](#) (英文) 中所設定的頻道交換訊息。它使用業界標準的
HTTPS 上的 REST 和 JSON，且支援使用 JWT Bearer 權杖驗證。如需如何使用 Bot 連接器服務的詳細資訊，請參
閱 [驗證](#) 和此節中的其餘文章。

活動

Bot 連接器服務透過傳遞 [Activity](#) 物件以在 Bot 和頻道 (使用者) 之間交換資訊。最常見的活動類型是 **message**，
但是還有其他活動類型可用來將各種類型的資訊傳達給 Bot 或頻道。如需 Bot 連接器服務中 Activity 的詳細資訊，
請參閱 [活動概觀](#)。

Bot 狀態服務

Bot 狀態服務可讓您的 Bot 儲存及擷取與使用者、對話或特定對話內容中特定使用者相關聯的狀態資料。它使用業
界標準的 HTTPS 上的 REST 和 JSON，且支援使用 JWT Bearer 權杖驗證。您使用 Bot 狀態服務儲存的任何資料
都會儲存在 Azure 中，並且待用加密。

Bot 狀態服務只有在和 Bot 連接器服務搭配時才有用。也就是說，可以將它用於儲存與對話 (Bot 使用 Bot 連接器
服務進行的對話) 相關的狀態資料。如需如何使用 Bot 狀態服務的詳細資訊，請參閱 [驗證](#) 和 [管理狀態資料](#)。

驗證

Bot 連接器服務和 Bot 狀態服務支援使用 JWT Bearer 權杖進行驗證。如需如何驗證 Bot 傳送至 Bot Framework
連出要求、如何驗證 Bot 從 Bot Framework 接收的連入要求之詳細資訊，請參閱 [驗證](#)。

用戶端程式庫

Bot Framework 提供可用於在 C# 或 Node.js 中建置 Bot 的用戶端程式庫。

- 若要使用 C# 建置 Bot，請使用 [適用於 C# 的 Bot Framework SDK](#)。
- 若要使用 Node.js 建置 Bot，請使用 [適用於 Node.js 的 Bot Framework SDK](#)。

除了建構 Bot 連接器服務和 Bot 狀態服務模型，每個 Bot Framework SDK 也都提供強大的功能，可建置封裝交談
邏輯的對話、簡單項目的內建提示 (如是/否)、字串、數字和列舉、功能強大之 AI 架構 (如 [LUIS](#)) 的支援等等。

NOTE

除了使用 C# SDK 或 Node.js SDK, 您也可以改為使用 [Bot 連接器 Swagger 檔案](#)和 [Bot 狀態 Swagger 檔案](#)以您所選的程式設計語言產生自己用戶端程式庫。

其他資源

透過檢閱本節中的各文章, 深入了解如何使用 Bot 連接器服務和 Bot 狀態服務, 從[驗證](#)開始。如果您遇到問題或有關於 Bot 連接器服務或 Bot 狀態服務的建議, 請參閱[支援](#)以取得可用資源清單。

使用 Bot Connector 服務建立 Bot

2019/2/28 • [Edit Online](#)

Bot Connector 服務可讓您的 Bot 藉由使用產業標準的 REST 和 JSON over HTTPS, 與 [Bot Framework 入口網站](#) 中設定的通道交換訊息。本教學課程會逐步引導您完成從 Bot Framework 取得存取權杖，並且使用 Bot Connector 服務與使用者交換訊息的程序。

取得存取權杖

IMPORTANT

若您尚未這麼做，請務必向 Bot Framework [註冊您的 Bot](#)，以取得其應用程式識別碼和密碼。您將會需要 Bot 的應用程式識別碼和密碼以取得存取權杖。

若要與 Bot Connector 服務通訊，您必須使用下列格式，在每個 API 要求的 `Authorization` 標頭中指定存取權杖：

```
Authorization: Bearer ACCESS_TOKEN
```

您可以藉由發出 API 要求，針對您的 Bot 取得存取權杖。

要求

如需要求存取權杖，以便用來對 Bot Connector 服務驗證要求，請發出下列要求，並將 **MICROSOFT-APP-ID** 和 **MICROSOFT-APP-PASSWORD**，取代為您向 Bot Framework [註冊](#)您的 Bot 時所取得的應用程式識別碼和密碼。

```
POST https://login.microsoftonline.com/botframework.com/oauth2/v2.0/token
Host: login.microsoftonline.com
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&client_id=MICROSOFT-APP-ID&client_secret=MICROSOFT-APP-
PASSWORD&scope=https%3A%2Fapi.botframework.com%2F.default
```

Response

如果要求成功，您會收到 HTTP 200 回應，指定存取權杖及其到期的相關資訊。

```
{
  "token_type": "Bearer",
  "expires_in": 3600,
  "ext_expires_in": 3600,
  "access_token": "eyJhbGciOiJIUzI1Ni..."}
```

TIP

如需 Bot Connector 服務中驗證的詳細資訊，請參閱[驗證](#)。

與使用者交換訊息

對話是使用者與 Bot 之間交換的一系列訊息。

從使用者處接收訊息

當使用者傳送訊息時，Bot Framework Connector 會將要求 POST 到您在註冊 Bot 時指定的端點。要求主體是活動物件。下列範例示範當使用者將簡單的訊息傳送給 Bot 時，Bot 收到的要求主體。

```
{  
    "type": "message",  
    "id": "bf3cc9a2f5de...",  
    "timestamp": "2016-10-19T20:17:52.2891902Z",  
    "serviceUrl": "https://smba.trafficmanager.net/apis",  
    "channelId": "channel's name/id",  
    "from": {  
        "id": "1234abcd",  
        "name": "user's name"  
    },  
    "conversation": {  
        "id": "abcd1234",  
        "name": "conversation's name"  
    },  
    "recipient": {  
        "id": "12345678",  
        "name": "bot's name"  
    },  
    "text": "Haircut on Saturday"  
}
```

回覆使用者的訊息

當您的 Bot 端點收到代表來自使用者訊息的 POST 要求時（亦即 type = 訊息），請使用該要求中的資訊，為回應建立活動物件。

1. 將 conversation 屬性設為使用者訊息中 conversation 屬性的內容。
2. 將 from 屬性設為使用者訊息中 recipient 屬性的內容。
3. 將 recipient 屬性設為使用者訊息中 from 屬性的內容。
4. 適當地設定 text 和 attachments 屬性。

在連入要求中使用 serviceUrl 屬性以識別基底 URI，您的 Bot 應該使用該 URI 來發出其回應。

若要傳送回應，將您的活動物件 POST 到 /v3/conversations/{conversationId}/activities/{activityId}，如下列範例所示。這個要求的主體是活動物件，該物件會提示使用者選取可用約會時間。

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/bf3cc9a2f5de...  
Authorization: Bearer eyJhbGciOiJIUzI1Ni...  
Content-Type: application/json
```

```
{  
    "type": "message",  
    "from": {  
        "id": "12345678",  
        "name": "bot's name"  
    },  
    "conversation": {  
        "id": "abcd1234",  
        "name": "conversation's name"  
    },  
    "recipient": {  
        "id": "1234abcd",  
        "name": "user's name"  
    },  
    "text": "I have these times available:",  
    "replyToId": "bf3cc9a2f5de..."  
}
```

在此範例要求中，`https://smba.trafficmanager.net/apis` 表示基底 URI，和 Bot 所提出要求的基底 URI 可能不同。如需設定基底 URI 的詳細資訊，請參閱 [API 參考](#)。

IMPORTANT

如同這個範例所示，您所傳送每個 API 要求的 `Authorization` 標頭都必須包含字詞 **Bearer**，後面加上您從 [Bot Framework 取得](#)的存取權杖。

若要傳送另一則訊息，讓使用者藉由按一下按鈕來選取可用約會時間，請將另一個要求 `POST` 到相同端點：

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/bf3cc9a2f5de...  
Authorization: Bearer eyJhbGciOiJIUzI1Ni...  
Content-Type: application/json
```

```
{  
    "type": "message",  
    "from": {  
        "id": "12345678",  
        "name": "bot's name"  
    },  
    "conversation": {  
        "id": "abcd1234",  
        "name": "conversation's name"  
    },  
    "recipient": {  
        "id": "1234abcd",  
        "name": "user's name"  
    },  
    "attachmentLayout": "list",  
    "attachments": [  
        {  
            "contentType": "application/vnd.microsoft.card.thumbnail",  
            "content": {  
                "buttons": [  
                    {  
                        "type": "imBack",  
                        "title": "10:30",  
                        "value": "10:30"  
                    },  
                    {  
                        "type": "imBack",  
                        "title": "11:30",  
                        "value": "11:30"  
                    },  
                    {  
                        "type": "openUrl",  
                        "title": "See more",  
                        "value": "http://www.contososalon.com/scheduling"  
                    }  
                ]  
            }  
        ],  
        "replyToId": "bf3cc9a2f5de..."  
    }  
}
```

後續步驟

在本教學課程中，您會從 Bot Framework 取得存取權杖，並且使用 Bot Connector 服務與使用者交換訊息。您可以使用 [Bot Framework 模擬器](#)來測試您的 Bot 並且進行偵錯。如果您想要與其他人共用您的 Bot，您必須將它[設定](#)為在一或多個通道上執行。

API 參考資料

2019/4/4 • • [Edit Online](#)

NOTE

REST API 不等同於 SDK。REST API 是為了支援標準 REST 通訊，而 SDK 則是與 Bot Framework 通訊的慣用方法。

在 Bot Framework 內，Bot 連接器服務可讓 Bot 透過您在 Bot Framework 入口網站中設定的通道和使用者交換訊息；而 Bot 狀態服務則可讓 Bot 儲存和擷取其使用 Bot 連接器服務進行交談時的相關狀態資料。這兩種服務皆透過 HTTPS 使用業界標準的 REST 和 JSON。

IMPORTANT

不建議將 Bot Framework State Service API 用於生產環境，未來版本可能會加以取代。建議更新您的 Bot 程式碼以使用記憶體內部儲存體進行測試，或將其中一個 Azure 擴充功能用於生產環境 Bot。如需詳細資訊，請參閱 [.NET](#) 或 [Node](#) 實作的〈管理狀態資料〉主題。

基底 URI

使用者傳送訊息給 Bot 時，傳入要求將包含一個活動物件，以及一個用於指定端點以接收 Bot 回應的 `serviceUrl` 屬性。若要存取 Bot 連接器服務或 Bot 狀態服務，請將 `serviceUrl` 值當作 API 要求的基底 URI 使用。

例如，假設使用者傳送訊息給您的 Bot 時，Bot 收到了下列活動。

```
{  
  "type": "message",  
  "id": "bf3cc9a2f5de...",  
  "timestamp": "2016-10-19T20:17:52.2891902Z",  
  "serviceUrl": "https://smba.trafficmanager.net/apis",  
  "channelId": "channel's name/id",  
  "from": {  
    "id": "1234abcd",  
    "name": "user's name"  
  },  
  "conversation": {  
    "id": "abcd1234",  
    "name": "conversation's name"  
  },  
  "recipient": {  
    "id": "12345678",  
    "name": "bot's name"  
  },  
  "text": "Haircut on Saturday"  
}
```

使用者訊息內的 `serviceUrl` 屬性指出 Bot 應將回應傳送至端點 `https://smba.trafficmanager.net/apis`；而這就會是 Bot 在該交談內容中發出任何後續要求時所用的基底 URI。如果您的 Bot 需要傳送主動式訊息給使用者，請務必儲存 `serviceUrl` 的值。

下列範例說明由 Bot 發出的要求，該要求乃是用於回應使用者訊息。

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/bf3cc9a2f5de...
Authorization: Bearer eyJhbGciOiJIUzI1Ni...
Content-Type: application/json
```

```
{
  "type": "message",
  "from": {
    "id": "12345678",
    "name": "bot's name"
  },
  "conversation": {
    "id": "abcd1234",
    "name": "conversation's name"
  },
  "recipient": {
    "id": "1234abcd",
    "name": "user's name"
  },
  "text": "I have several times available on Saturday!",
  "replyToId": "bf3cc9a2f5de..."
}
```

headers

要求標頭

除了標準的 HTTP 要求標頭，您發出的每個 API 要求都必須包含指定存取權杖的 `Authorization` 標頭以驗證您的 Bot。使用此格式指定 `Authorization` 標頭：

```
Authorization: Bearer ACCESS_TOKEN
```

如需有關如何取得 Bot 存取權杖的詳細資料，請參閱[驗證 Bot 向 Bot 連接器服務提出的要求](#)。

回應標頭

除了標準的 HTTP 回應標頭，每個回應都將包含 `X-Correlating-OperationId` 標頭。此標頭的值即為對應於 Bot Framework 記錄項目的識別碼，其中包含有關要求的詳細資料。每次您收到錯誤訊息時，都應擷取該標頭的值。如果您無法獨立解決問題，回報時請將此值和相關資訊一併提供給支援小組。

HTTP 狀態碼

與每個回應一併傳回的 [HTTP 狀態碼](#)會指出對應要求的結果。

HTTP 狀態碼	意義
200	要求成功。
201	要求成功。
202	已接受要求進行處理。
204	要求成功，但未傳回任何內容。
400	要求的格式不正確或有其他錯誤。
401	Bot 未獲授權，無法提出要求。

HTTP 狀態碼	意義
403	不允許 Bot 執行要求的作業。
404	找不到要求的資源。
500	發生內部伺服器錯誤。
503	服務無法使用。

Errors

若回應顯示的 HTTP 狀態碼位於 4xx 或 5xx 範圍內，則會在提供錯誤相關資訊的回應本文中加入 [ErrorResponse](#) 物件。如果您收到 4xx 範圍的錯誤訊息，請檢查 [ErrorResponse](#) 物件以找出錯誤原因，並在重新提交要求之前先解決問題。

交談作業

透過這些作業建立交談、傳送訊息 (活動)，以及管理交談內容。

作業	說明
建立交談	建立新交談。
傳送至交談	將活動 (訊息) 傳送至指定交談的結尾處。
回覆活動	將活動 (訊息) 傳送至指定交談，以回覆指定活動。
取得交談成員	取得指定交談的成員。
取得對話分頁成員	取得指定對話的成員 (一次一頁)。
取得活動成員	取得指定交談中指定活動的成員。
更新活動	更新現有作業。
刪除活動	刪除現有活動。
將附件上傳至通道	將附件直接上傳到通道的 Blob 儲存體。

建立交談

建立新交談。

```
POST /v3/conversations
```

要求本文	Conversation 物件
傳回	ResourceResponse 物件

傳送至交談

將活動 (訊息) 傳送至指定交談。系統將根據時間戳記或通道的語意，將活動附加至交談的結尾處。若要回應交談內

的特定訊息，請改為使用[回覆活動](#)。

<code>POST /v3/conversations/{conversationId}/activities</code>

要求本文	Activity 物件
傳回	Identification 物件

回覆活動

將活動 (訊息) 傳送至指定交談，以回覆指定活動。若通道支援，則系統會新增活動以回覆其他活動。若通道不支援巢狀回覆，則此作業將以[傳送至交談](#)的方式進行。

<code>POST /v3/conversations/{conversationId}/activities/{activityId}</code>
--

要求本文	Activity 物件
傳回	Identification 物件

取得交談成員

取得指定交談的成員。

<code>GET /v3/conversations/{conversationId}/members</code>

要求本文	n/a
傳回	ChannelAccount 物件陣列

取得對話分頁成員

取得指定對話的成員 (一次一頁)。

<code>GET /v3/conversations/{conversationId}/pagedmembers</code>
--

要求本文	n/a
傳回	ChannelAccount 物件陣列，以及可用來取得更多值的接續權杖

取得活動成員

取得指定交談中指定活動的成員。

<code>GET /v3/conversations/{conversationId}/activities/{activityId}/members</code>

要求本文	n/a
傳回	ChannelAccount 物件陣列

更新活動

部分通道可讓您編輯現有活動，以反映 Bot 交談的新狀態。例如，使用者按下其中一個按鈕後，您或許會將按鈕從交談訊息中移除。若成功執行此作業，指定交談內的指定活動將會更新。

```
PUT /v3/conversations/{conversationId}/activities/{activityId}
```

要求本文	Activity 物件
傳回	Identification 物件

刪除活動

部分通道可讓您刪除現有活動。若成功執行此作業，指定活動將會從指定交談移除。

```
DELETE /v3/conversations/{conversationId}/activities/{activityId}
```

要求本文	n/a
傳回	用於指出作業結果的 HTTP 狀態碼。回應本文中未指定任何項目。

將附件上傳至通道

將指定交談的附件直接上傳至通道的 Blob 儲存體。如此一來，可讓您將資料儲存在相容的存放區。

```
POST /v3/conversations/{conversationId}/attachments
```

要求本文	AttachmentUpload 物件。
傳回	ResourceResponse 物件。識別碼屬性會指定可搭配 取得附件資訊 作業和 取得附件 作業使用的附件識別碼。

附件作業

利用這些作業擷取附件相關資訊，以及該檔案本身的二進位資料。

作業	說明
取得附件資訊	取得有關指定附件的資訊，包括檔案名稱、檔案類型及可用檢視（例如：原稿或縮圖）。
取得附件	取得指定附件的指定檢視做為二進位內容。

取得附件資訊

取得有關指定附件的資訊，包括檔案名稱、類型及可用檢視（例如：原稿或縮圖）。

```
GET /v3/attachments/{attachmentId}
```

要求本文	n/a
傳回	AttachmentInfo 物件

取得附件

取得指定附件的指定檢視做為二進位內容。

```
GET /v3/attachments/{attachmentId}/views/{viewId}
```

要求本文	n/a
傳回	代表指定附件之指定檢視的二進位內容

狀態作業

利用這些作業儲存和擷取狀態資料。

作業	說明
設定使用者資料	儲存通道上特定使用者的狀態資料。
設定交談資料	儲存某通道上特定交談的狀態資料。
設定私人交談資料	儲存通道上特定交談內容中特定使用者的狀態資料。
取得使用者資料	擷取先前為通道上所有交談的特定使用者儲存的狀態資料。
取得交談資料	擷取先前為通道上的特定交談儲存的狀態資料。
取得私人交談資料	擷取先前為通道上特定交談內容中特定使用者儲存的狀態資料。
刪除使用者狀態	運用 設定使用者資料 作業或 設定私人交談資料 作業，刪除先前為使用者儲存的狀態資料。

設定使用者資料

儲存指定通道上指定使用者的狀態資料。

```
POST /v3/botstate/{channelId}/users/{userId}
```

要求本文	BotData 物件
傳回	BotData 物件

設定交談資料

儲存指定通道上指定交談的狀態資料。

```
POST /v3/botstate/{channelId}/conversations/{conversationId}
```

要求本文	BotData 物件
傳回	BotData 物件

設定私人交談資料

針對指定通道上指定交談內容中的指定使用者儲存狀態資料。

```
POST /v3/botstate/{channelId}/conversations/{conversationId}/users/{userId}
```

要求本文	BotData 物件
傳回	BotData 物件

取得使用者資料

擷取先前針對指定通道上所有交談的指定使用者儲存的狀態資料。

```
GET /v3/botstate/{channelId}/users/{userId}
```

要求本文	n/a
傳回	BotData 物件

取得交談資料

擷取先前針對指定通道的指定交談儲存的狀態資料。

```
GET /v3/botstate/{channelId}/conversations/{conversationId}
```

要求本文	n/a
傳回	BotData 物件

取得私人交談資料

擷取先前針對指定通道的指定交談內容之指定使用者儲存的狀態資料。

```
GET /v3/botstate/{channelId}/conversations/{conversationId}/users/{userId}
```

要求本文	n/a
傳回	BotData 物件

刪除使用者狀態

使用[設定使用者資料](#)作業或[設定私人交談資料](#)作業，刪除先前為指定通道的指定使用者儲存的狀態資料。

```
DELETE /v3/botstate/{channelId}/users/{userId}
```

要求本文	n/a
傳回	字串陣列 (識別碼)

結構描述

結構描述可定義能讓 Bot 用來與使用者通訊的物件及其屬性。

OBJECT	說明
Activity 物件	定義 Bot 和使用者之間交換的訊息。
AnimationCard 物件	定義可播放動畫 GIF 或短片的資訊卡。
Attachment 物件	定義要包含在訊息中的其他資訊。附件可能是媒體檔案 (例如：音訊、影片、影像、檔案) 或豐富資訊卡。
AttachmentData 物件	描述附件資料。
AttachmentInfo 物件	描述附件。
AttachmentView 物件	定義附件檢視。
AttachmentUpload 物件	定義要上傳的附件。
AudioCard 物件	定義可播放音訊檔案的資訊卡。
BotData 物件	定義使用者狀態資料、交談或利用 Bot 狀態服務所儲存的特定交談內容中的使用者。
CardAction 物件	定義要執行的動作。
CardImage 物件	定義要在資訊卡上顯示的影像。
ChannelAccount 物件	定義通道上的 Bot 或使用者帳戶。

OBJECT	說明
Conversation 物件	定義交談，包含交談中的 Bot 及使用者。
ConversationAccount 物件	定義通道中的交談。
ConversationParameters 物件	定義用來建立新交談的參數
ConversationReference 物件	定義交談中的特定要點。
ConversationResourceResponse 物件	包含資源的回應
Entity 物件	定義實體物件。
Error 物件	定義錯誤。
ErrorResponse 物件	定義 HTTP API 回應。
Fact 物件	定義包含事實的機碼值組。
Geocoordinates 物件	定義採用「世界大地坐標系統 (WGS84)」座標的地理位置。
HeroCard 物件	定義具有大型影像、標題、文字及動作按鈕的資訊卡。
Identification 物件	識別資源。
MediaEventValue 物件	媒體事件的增補參數。
MediaUrl 物件	定義媒體檔案來源的 URL。
Mention 物件	定義在交談中提及的使用者或 Bot。
MessageReaction 物件	定義對訊息的回應。
Place 物件	定義在交談中提及的地點。
ReceiptCard 物件	定義內含購買收據的資訊卡。
ReceiptItem 物件	定義收據內的明細項目。
ResourceResponse 物件	定義資源。
SignInCard 物件	定義可讓使用者登入至服務的資訊卡。
SuggestedActions 物件	定義可讓使用者從中選擇的選項。
ThumbnailCard 物件	定義具有縮圖影像、標題、文字和動作按鈕的資訊卡。
ThumbnailUrl 物件	定義影像來源的 URL。
VideoCard 物件	定義可播放影片的資訊卡。

OBJECT	說明
SemanticAction 物件	定義程式設計動作的參考。

活動物件

定義 Bot 和使用者之間交換的訊息。

屬性	類型	說明
action	字串	要套用的動作或是已套用的動作。使用類型屬性判斷動作的內容。例如，在類型是 contactRelationUpdate 的情況下，若使用者將 Bot 新增至其連絡人清單，則 action 屬性的值會是 add ；而若使用者將 Bot 從連絡人清單中移除，其值則為 remove 。
attachments	Attachment[]	Attachment 物件陣列，用於定義要加入訊息的其他資訊。每個附件可能是媒體檔案（例如：音訊、影片、影像、檔案）或豐富資訊卡。
attachmentLayout	字串	包含在訊息中的豐富資訊卡附件的版面配置。下列任一值： carousel 、 list 。如需有關豐富資訊卡附件的詳細資訊，請參閱 將豐富資訊卡附件新增至訊息 。
channelData	物件	包含通道專用內容的物件。某些通道提供的功能，需要使用無法以附件結構描述來呈現的其他資訊。在該等情況下，請將此屬性設定為通道文件中所定義的通道專用內容。如需詳細資訊，請參閱 實作通道專屬功能 。
channelId	字串	可唯一識別通道的識別碼。由通道設定。
交談	ConversationAccount	定義活動所屬交談的 ConversationAccount 物件。
code	字串	指出交談終止原因的代碼。
entities	object[]	代表訊息中提及之實體的物件陣列。此陣列中的物件可能是任何 Schema.org 物件。例如，陣列可能包含 Mention 物件，可識別交談中提及的使用者；而 Place 物件則可識別交談中提及的地點。
from	ChannelAccount	用於指出訊息傳送者的 ChannelAccount 物件。
historyDisclosed	布林值	用於指出是否公開歷程記錄的旗標。預設值為 false 。
id	字串	可唯一識別通道活動的識別碼。

屬性	類型	說明
inputHint	字串	此值指出您的 Bot 在訊息傳遞給用戶端之後是否要接受、等候或忽略使用者輸入。下列任一值： acceptingInput 、 expectingInput 、 ignoringInput 。
locale	字串	訊息內顯示文字所使用之語言的地區設定 (格式為: <language>-<country>)。通道使用此屬性指出使用者語言，如此 Bot 即可以該語言指定顯示字串。預設值為 en-US 。
localTimestamp	字串	訊息傳送時的當地時區日期及時間，以 ISO-8601 格式表示。
membersAdded	ChannelAccount []	代表已加入交談之使用者清單的 ChannelAccount 物件陣列。僅在活動類型為「conversationUpdate」，且使用者已加入交談的情況下才會顯示。
membersRemoved	ChannelAccount []	代表已離開交談之使用者清單的 ChannelAccount 物件陣列。僅在活動類型為「conversationUpdate」，且使用者已離開交談的情況下才會顯示。
name	字串	要叫用之作業的名稱或事件的名稱。
recipient	ChannelAccount	用於指出訊息收件者的 ChannelAccount 物件。
relatesTo	ConversationReference	用於定義交談中特定要點的 ConversationReference 物件。
replyToId	字串	此訊息要回覆之訊息的識別碼。若要回覆使用者傳送的訊息，請將此屬性設為該使用者訊息的識別碼。並非所有通道皆支援執行緒回覆。在這些情況下，通道會忽略此屬性，並使用依照時間排序的語意 (時間戳記)，將訊息附加至交談。
serviceUrl	字串	用於指定通道服務端點的 URL。由通道設定。
speak	字串	要讓 Bot 在支援語音功能的通道上以語音讀出的文字。若要控制 Bot 語音的聲音、速率、音量、發音和音高等各種特性，請以 語音合成標記語言 (SSML) 格式指定此屬性。
suggestedActions	SuggestedActions	用於定義選項以讓使用者從中選擇的 SuggestedActions 物件。
summary	字串	訊息包含的資訊摘要。例如，若是透過電子郵件通道傳送的訊息，此屬性可指定電子郵件訊息的前 50 個字元。

屬性	類型	說明
text	字串	Bot 與使用者之間傳送的訊息文字。請參閱通道文件，以了解加諸於此屬性內容的限制。
textFormat	字串	訊息文字的格式。下列任一值： markdown 、 plain 、 xml 。如需有關文字格式的詳細資料，請參閱 建立訊息 。
timestamp	字串	訊息傳送時的 UTC 時區日期及時間，以 ISO-8601 格式表示。
topicName	字串	活動所屬交談的主題。
type	字串	活動的類型。下列任一值： contactRelationUpdate 、 conversationUpdate 、 deleteUserData 、 message 、 typing 、 event 和 endOfConversation 。如需有關活動類型的詳細資料，請參閱 活動概觀 。
value	物件	開放端點的值。
semanticAction	SemanticAction	SemanticAction 物件，代表程式設計動作的參考。

[回到結構描述資料表](#)

AnimationCard 物件

定義可播放動畫 GIF 或短片的資訊卡。

屬性	類型	說明
autoloop	布林值	用於指出最後一個動畫 GIF 播放完畢後是否要重播動畫 GIF 清單的旗標。將此屬性設為 true 可自動重播動畫；如不重播則設為 false 。預設值為 true 。
autoplay	布林值	用於指出資訊卡顯示時是否要自動播放動畫的旗標。將此屬性設為 true 可自動播放動畫；如不播放則設為 false 。預設值為 true 。
buttons	CardAction[]	可讓使用者執行一或多個動作的 CardAction 物件陣列。此通道可決定您能指定的按鈕數目。
duration	字串	媒體內容的長度，採用 ISO 8601 持續時間格式 。
映像	ThumbnailUrl	可指定要在資訊卡上顯示之影像的 ThumbnailUrl 物件。

屬性	類型	說明
media	MediaUrl	用於指定要播放之動畫 GIF 清單的 MediaUrl 物件陣列。
shareable	布林值	用於指出是否要與他人共用動畫的旗標。將此屬性設為 true 可與他人共用動畫；如不共用則設為 false 。預設值為 true 。
subtitle	字串	顯示在資訊卡標題下方的子標題。
text	字串	顯示在資訊卡標題或子標題下方的描述或提示。
title	字串	資訊卡的標題。
value	物件	此資訊卡的增補參數

[回到結構描述資料表](#)

Attachment 物件

定義要包含在訊息中的其他資訊。附件可能是媒體檔案 (例如：音訊、影片、影像、檔案) 或豐富資訊卡。

屬性	類型	說明

屬性	類型	說明
contentType	字串	<p>附件中內容的媒體類型。若為媒體檔案，請將此屬性設為已知的媒體類型如：image/png、audio/wav 和 video/mp4。若為豐富資訊卡，請將此屬性設為下列任一個廠商特有類型：</p> <ul style="list-style-type: none"> • application/vnd.microsoft.card.adaptive: 豐富卡片可包含文字、語音、影像、按鈕和輸入欄位的任何組合。將內容屬性設為 AdaptiveCard 物件。 • application/vnd.microsoft.card.animation: 可播放動畫的豐富介面卡。將內容屬性設為 AnimationCard 物件。 • application/vnd.microsoft.card.audio: 可播放音訊檔案的豐富介面卡。將內容屬性設為 AudioCard 物件。 • application/vnd.microsoft.card.video: 可播放影片的豐富介面卡。將內容屬性設為 VideoCard 物件。 • application/vnd.microsoft.card.hero: 主圖卡片。將內容屬性設為 HeroCard 物件。 • application/vnd.microsoft.card.thumbnail: 縮圖卡片。將內容屬性設為 ThumbnailCard 物件。 • application/vnd.microsoft.com.card.receipt: 收據卡片。將內容屬性設為 ReceiptCard 物件。 • application/vnd.microsoft.com.card.signin: 使用者登入卡片。將內容屬性設為 SignInCard 物件。
contentUrl	字串	附件內容的 URL。例如，若附件為影像，則將 contentUrl 設為代表該影像位置的 URL。支援的通訊協定：HTTP、HTTPS、檔案和資料。
content	物件	附件的內容。若附件是豐富資訊卡，則將此屬性設為豐富資訊卡物件。此屬性與 contentUrl 屬性互斥。
name	字串	附件的名稱。
thumbnailUrl	字串	若通道支援使用小型的內容或 contentUrl ，通道可使用之縮圖影像的 URL。例如，若您將 contentType 設為 application/word ，並且將 contentUrl 設為 Word 文件的位置，則可加入代表該文件的縮圖影像。通道會顯示縮圖影像而非該文件。只要使用者按一下影像，通道便會開啟該文件。

[回到結構描述資料表](#)

AttachmentData 物件

描述附件資料。

屬性	類型	說明
name	字串	附件的名稱。
originalBase64	字串	附件內容。
thumbnailBase64	字串	附件縮圖內容。
type	字串	附件的內容類型。

AttachmentInfo 物件

描述附件。

屬性	類型	說明
name	字串	附件的名稱。
type	字串	附件的 ContentType。
檢視	AttachmentView[]	表示附件可用檢視的 AttachmentView 物件陣列。

[回到結構描述資料表](#)

AttachmentView 物件

定義附件檢視。

屬性	類型	說明
viewId	字串	檢視識別碼。
大小	number	檔案的大小。

[回到結構描述資料表](#)

AttachmentUpload 物件

定義要上傳的附件。

屬性	類型	說明
type	字串	附件的 ContentType。
name	字串	附件的名稱。
originalBase64	字串	代表檔案原始版本內容的二進位資料。

屬性	類型	說明
thumbnailBase64	字串	代表檔案縮圖版本內容的二進位資料。

[回到結構描述資料表](#)

AudioCard 物件

定義可播放音訊檔案的資訊卡。

屬性	類型	說明
aspect	字串	在 image 屬性中指定之縮圖的外觀比例。有效值為 16:9 和 9:16 。
autoloop	布林值	用於指出最後一個音訊檔案播放完畢後是否要重播音訊清單的旗標。將此屬性設為 true 可自動重播音訊檔案；如不重播則設為 false 。預設值為 true 。
autoplay	布林值	用於指出資訊卡顯示時是否要自動播放音訊的旗標。將此屬性設為 true 可自動播放音訊；如不播放則設為 false 。預設值為 true 。
buttons	CardAction []	可讓使用者執行一或多個動作的 CardAction 物件陣列。此通道可決定您能指定的按鈕數目。
duration	字串	媒體內容的長度，採用 ISO 8601 持續時間格式 。
映像	ThumbnailUrl	可指定要在資訊卡上顯示之影像的 ThumbnailUrl 物件。
media	MediaUrl []	用於指定要播放之音訊檔案清單的 MediaUrl 物件陣列。
shareable	布林值	用於指出是否要與他人共用音訊檔案的旗標。將此屬性設為 true 可與他人共用音訊；如不共用則設為 false 。預設值為 true 。
subtitle	字串	顯示在資訊卡標題下方的子標題。
text	字串	顯示在資訊卡標題或子標題下方的描述或提示。
title	字串	資訊卡的標題。
value	物件	此資訊卡的增補參數

[回到結構描述資料表](#)

BotData 物件

定義使用者狀態資料、交談或利用 Bot 狀態服務所儲存的特定交談內容中的使用者。

屬性	類型	說明
資料	物件	在要求中, 使用 Bot 狀態服務指定要儲存之屬性和值的 JSON 物件。在回應中, 使用 Bot 狀態服務指定已經儲存之屬性和值的 JSON 物件。
eTag	字串	此為實體標記值, 可讓您針對使用 Bot 狀態服務儲存的資料, 控制資料並行存取。如需詳細資訊, 請參閱 管理狀態資料 。

[回到結構描述資料表](#)

CardAction 物件

定義要執行的動作。

屬性	類型	說明
映像	字串	顯示於以下項目之影像的 URL:
text	字串	動作的文字
title	字串	按鈕的文字。僅適用於按鈕的動作。
按鈕。僅適用於按鈕的動作。		
type	字串	要執行之動作的類型。如需有效值的清單, 請參閱 將豐富資訊卡附件新增至訊息 。
value	物件	動作的增補參數。此屬性的值將因動作類型而有所不同。如需詳細資訊, 請參閱 將豐富資訊卡附件新增至訊息 。

[回到結構描述資料表](#)

CardImage 物件

定義要在資訊卡上顯示的影像。

屬性	類型	說明
alt	字串	影像的描述。您應加入描述以支援可存取性。
點選	CardAction	用於指定使用者點選或按一下影像後要執行之動作的 CardAction 物件。
url	字串	影像來源的 URL 或影像的 base64 二進位 (例如: <code>data:image/png;base64,iVBORw0KGgo...</code>)。

[回到結構描述資料表](#)

ChannelAccount 物件

定義通道上的 Bot 或使用者帳戶。

屬性	類型	說明
id	字串	可唯一識別通道上 Bot 或使用者的識別碼。
name	字串	Bot 或使用者的名稱。

[回到結構描述資料表](#)

交談物件

定義交談，包含交談中的 Bot 及使用者。

屬性	類型	說明
Bot	ChannelAccount	用於識別 Bot 的 ChannelAccount 物件。
isGroup	布林值	用於指出此項目是否為群組交談的旗標。設為 true 代表此為群組交談，反之則設為 false 。預設值為 false 。若要開啟群組交談，通道必須支援群組交談功能。
members	ChannelAccount[]	用於識別交談成員的 ChannelAccount 物件陣列。此清單必須包含單一使用者，除非 isGroup 設為 true 。此清單可能包含其他 Bot。
topicName	字串	交談的標題。
activity	活動	在 建立交談 要求中，用於定義要發佈至新交談之第一個訊息的 Activity 物件。

[回到結構描述資料表](#)

ConversationAccount 物件

定義通道中的交談。

屬性	類型	說明
id	字串	可識別交談的識別碼。此識別碼在每個通道均不重複。通道開啟交談時便會設定此識別碼；反之，則由 Bot 將此屬性設為其開啟交談時，Bot 從回應中取回的識別碼（請參閱「 開啟交談 」一節）。

屬性	類型	說明
isGroup	布林值	用於指出活動產生時交談是否要包含兩位以上參與者的旗標。設為 true 代表此為群組交談，反之則設為 false 。預設值為 false 。
name	字串	可用來識別交談的顯示名稱。
conversationType	字串	指出可區別交談類型 (例如：群組或私人) 之通道中的交談類型。

[回到結構描述資料表](#)

ConversationParameters 物件

定義用來建立新交談的參數

屬性	類型	說明
isGroup	布林值	指出此項目是否為群組交談。
Bot	ChannelAccount	交談中 Bot 的位址。
members	array	要新增至交談的成員清單。
topicName	字串	交談的主題標題。僅在通道提供支援時，才可使用此屬性。
activity	活動	(選用) 建立新交談時，以此活動做為交談的初始訊息。
channelData	物件	建立交談時的通道特定酬載。

ConversationReference 物件

定義交談中的特定要點。

屬性	類型	說明
activityId	字串	可唯一識別此物件參照之活動的識別碼。
Bot	ChannelAccount	用於識別此物件參照之交談內 Bot 的 ChannelAccount 物件。
channelId	字串	可唯一識別此物件參照之交談內通道的識別碼。
交談	ConversationAccount	用於定義此物件參照之交談的 ConversationAccount 物件。
serviceUrl	字串	用於指定此物件參照之交談中通道服務端點的 URL。

屬性	類型	說明
user	ChannelAccount	用於指出此物件參照之交談內使用者的 ChannelAccount 物件。

[回到結構描述資料表](#)

ConversationResourceResponse 物件

定義包含資源的回應。

屬性	類型	說明
activityId	字串	活動的識別碼。
id	字串	資源的識別碼。
serviceUrl	字串	服務端點。

錯誤物件

定義錯誤。

屬性	類型	說明
code	字串	錯誤碼。
message	字串	錯誤的描述。

[回到結構描述資料表](#)

實體物件

定義實體物件。

屬性	類型	說明
type	字串	實體類型。通常包含來自 schema.org 的類型。

ErrorResponse 物件

定義 HTTP API 回應。

屬性	類型	說明
error	錯誤	含有錯誤相關資訊的 Error 物件。

[回到結構描述資料表](#)

Fact 物件

定義包含事實的機碼值組。

屬性	類型	說明
key	字串	事實的名稱。例如： 簽入 。此索引鍵在顯示事實的值時做為標籤使用。
value	字串	事實的值。例如： 2016 年 10 月 10 日 。

[回到結構描述資料表](#)

Geocoordinates 物件

定義採用「世界大地坐標系統 (WGS84)」座標的地理位置。

屬性	類型	說明
海拔	number	位置的海拔。
name	字串	位置的名稱。
緯度	number	位置的緯度。
經度	number	位置的經度。
type	字串	此物件的類型。一律設為 GeoCoordinates 。

[回到結構描述資料表](#)

HeroCard 物件

定義具有大型影像、標題、文字及動作按鈕的資訊卡。

屬性	類型	說明
buttons	CardAction[]	可讓使用者執行一或多個動作的 CardAction 物件陣列。此通道可決定您能指定的按鈕數目。
images	CardImage[]	用於指定要在資訊卡上顯示之影像的 CardImage 物件陣列。主圖卡僅包含一個影像。
subtitle	字串	顯示在資訊卡標題下方的子標題。
點選	CardAction	用於指定使用者點選或按一下資訊卡後要執行之動作的 CardAction 物件。此值可與任一按鈕動作相同或選用其他動作。
text	字串	顯示在資訊卡標題或子標題下方的描述或提示。
title	字串	資訊卡的標題。

[回到結構描述資料表](#)

識別物件

識別資源。

屬性	類型	說明
id	字串	可唯一識別資源的識別碼。

[回到結構描述資料表](#)

MediaEventValue 物件

媒體事件的增補參數。

屬性	類型	說明
cardValue	物件	在產生此事件之媒體卡上，於 [值] 欄位中指定的回呼參數。

MediaUrl 物件

定義媒體檔案來源的 URL。

屬性	類型	說明
設定檔	字串	用於描述媒體內容的提示。
url	字串	媒體檔案來源的 URL。

[回到結構描述資料表](#)

Mention 物件

定義在交談中提及的使用者或 Bot。

屬性	類型	說明
提及	ChannelAccount	用於指定提及之使用者或 Bot 的 ChannelAccount 物件。請注意，部分通道（如 Slack）會按交談指派名稱，因此在訊息收件者屬性中提及的 Bot 名稱，可能與您註冊 Bot 時指定的控制代碼不同。不過，這兩者的帳戶識別碼是相同的。
text	字串	在交談中提及的使用者或 Bot。例如，若訊息是「@ColorBot幫我挑一個新顏色」，則此屬性便會設為 @ColorBot。並非所有通道都能設定此屬性。
type	字串	此物件的類型。一律設為提及。

[回到結構描述資料表](#)

MessageReaction 物件

定義對訊息的回應。

屬性	類型	說明
type	字串	反應的類型。

Place 物件

定義在交談中提及的地點。

屬性	類型	說明
address	物件	地點的地址。此屬性可為 <code>string</code> 或類型 <code>PostalAddress</code> 的複雜物件。
地理區域	<code>GeoCoordinates</code>	用於指定地點之地理座標的 <code>GeoCoordinates</code> 物件。
hasMap	物件	對應至地點。此屬性可為 <code>string</code> (<code>URL</code>) 或類型 <code>Map</code> 的複雜物件。
name	字串	地點的名稱。
type	字串	此物件的類型。一律設為地點。

[回到結構描述資料表](#)

ReceiptCard 物件

定義內含購買收據的資訊卡。

屬性	類型	說明
buttons	<code>CardAction[]</code>	可讓使用者執行一或多個動作的 <code>CardAction</code> 物件陣列。此通道可決定您能指定的按鈕數目。
facts	<code>Fact[]</code>	用於指定購買相關資訊的 <code>Fact</code> 物件陣列。例如, 飯店住宿收據的明細清單將列出入住日期和退房日期。此通道可決定您能指定的事實數目。
items	<code>ReceiptItem[]</code>	用於指定已購項目的 <code>ReceiptItem</code> 物件陣列
點選	<code>CardAction</code>	用於指定使用者點選或按一下資訊卡後要執行之動作的 <code>CardAction</code> 物件。此值可與任一按鈕動作相同或選用其他動作。
tax	字串	用於指定該筆購買金額須支付之稅金的貨幣格式字串。
title	字串	顯示在收據頂端的標題。

屬性	類型	說明
total	字串	用於指定購買總額 (包括適用稅金) 的貨幣格式字串。
vat	字串	用於指定該筆購買金額須支付之加值稅 (VAT) 的貨幣格式字串。

[回到結構描述資料表](#)

ReceiptItem 物件

定義收據內的明細項目。

屬性	類型	說明
映像	CardImage	用於指定要顯示於明細項目旁之縮圖影像的 CardImage 物件。
price	字串	用於指定所有已購單位之總價的貨幣格式字串。
quantity	字串	用於指定已購單位數量的數值字串。
subtitle	字串	顯示於明細項目標題下方的子標題。
點選	CardAction	用於指定使用者點選或按一下明細項目後要執行之動作的 CardAction 物件。
text	字串	明細項目的描述。
title	字串	明細項目的標題。

[回到結構描述資料表](#)

ResourceResponse 物件

定義包含資源識別碼的回應。

屬性	類型	說明
id	字串	可唯一識別資源的識別碼。

[回到結構描述資料表](#)

SignInCard 物件

定義可讓使用者登入至服務的資訊卡。

屬性	類型	說明
buttons	CardAction[]	可讓使用者登入服務的 CardAction 物件。此通道可決定您能指定的按鈕數目。

屬性	類型	說明
text	字串	要加入登入卡的描述或提示。

[回到結構描述資料表](#)

SuggestedActions 物件

定義可讓使用者從中選擇的選項。

屬性	類型	說明
actions	CardAction[]	用於定義建議動作的 CardAction 物件。
to	string[]	此字串陣列包含可檢視建議動作之收件者的識別碼。

[回到結構描述資料表](#)

ThumbnailCard 物件

定義具有縮圖影像、標題、文字和動作按鈕的資訊卡。

屬性	類型	說明
buttons	CardAction[]	可讓使用者執行一或多個動作的 CardAction 物件陣列。此通道可決定您能指定的按鈕數目。
images	CardImage[]	用於指定要在資訊卡上顯示之縮圖影像的 CardImage 物件。此通道可決定您能指定的縮圖影像數目。
subtitle	字串	顯示在資訊卡標題下方的子標題。
點選	CardAction	用於指定使用者點選或按一下資訊卡後要執行之動作的 CardAction 物件。此值可與任一按鈕動作相同或選用其他動作。
text	字串	顯示在資訊卡標題或子標題下方的描述或提示。
title	字串	資訊卡的標題。

[回到結構描述資料表](#)

ThumbnailUrl 物件

定義影像來源的 URL。

屬性	類型	說明
alt	字串	影像的描述。您應加入描述以支援可存取性。

屬性	類型	說明
url	字串	影像來源的 URL 或影像的 base64 二進位 (例如: data:image/png;base64,iVBORw0KGgo...)。

[回到結構描述資料表](#)

VideoCard 物件

定義可播放影片的資訊卡。

屬性	類型	說明
aspect	字串	影片的外觀比例 (例如:16:9、4:3)。
autoloop	布林值	用於指出最後一個影片播放完畢後是否要重播影片清單的旗標。將此屬性設為 true 可自動重播影片；如不重播則設為 false 。預設值為 true 。
autoplay	布林值	用於指出資訊卡顯示時是否要自動播放影片的旗標。將此屬性設為 true 可自動播放影片；如不播放則設為 false 。預設值為 true 。
buttons	CardAction []	可讓使用者執行一或多個動作的 CardAction 物件陣列。此通道可決定您能指定的按鈕數目。
duration	字串	媒體內容的長度，採用 ISO 8601 持續時間格式 。
映像	ThumbnailUrl	可指定要在資訊卡上顯示之影像的 ThumbnailUrl 物件。
media	MediaUrl []	用於指定要播放之影片清單的 MediaUrl 物件。
shareable	布林值	用於指出是否要與他人共用影片的旗標。將此屬性設為 true 可與他人共用影片；如不共用則設為 false 。預設值為 true 。
subtitle	字串	顯示在資訊卡標題下方的子標題。
text	字串	顯示在資訊卡標題或子標題下方的描述或提示。
title	字串	資訊卡的標題。
value	物件	此資訊卡的增補參數

[回到結構描述資料表](#)

SemanticAction 物件

定義程式設計動作的參考。

屬性	類型	說明
id	字串	此動作的識別碼
entities	實體	與此動作相關聯的實體

[回到結構描述資料表](#)

驗證

2019/2/28 • [Edit Online](#)

您的 Bot 會透過安全通道 (SSL/TLS) 使用 HTTP 和 Bot 連接器服務通訊。當 Bot 傳送要求至連接器服務時，必須包含連接器可用來驗證該 Bot 之身分識別的資訊。同樣地，當連接器服務傳送要求至 Bot 時，也必須包含 Bot 可用來驗證該服務之身分識別的資訊。本文描述 Bot 和 Bot 連接器服務之間所進行之服務層級驗證的驗證技術和需求。若您準備自行撰寫驗證程式碼，您必須實作本文所述的安全性程序，以使 Bot 能與 Bot 連接器服務交換訊息。

IMPORTANT

若您準備自行撰寫驗證程式碼，請務必正確實作所有安全性程序。透過實作本文中的所有步驟，您將能降低攻擊者讀取傳送至 Bot 的訊息、模擬 Bot 傳送訊息，以及竊取祕密金鑰的風險。

若您準備使用[適用於 .NET 的 Bot Framework SDK](#) 或[適用於 Node.js 的 Bot Framework SDK](#)，便不需要實作本文所述的安全性程序，因為 SDK 會自動實作那些程序。您只需以在[註冊](#)期間為 Bot 取得的應用程式識別碼和密碼設定專案，SDK 便會處理剩餘的部分。

WARNING

在 2016 年 12 月，Bot Framework 安全性通訊協定 v3.1 已針對數個用於權杖產生及驗證的值導入變更。在 2017 年秋末，已推出了 Bot Framework 安全性通訊協定 v3.2，其針對用於權杖產生及驗證的值納入變更。如需詳細資訊，請參閱[安全性通訊協定變更](#)。

驗證技術

一共會使用四個驗證技術來在 Bot 和 Bot 連接器之間建立信任：

TECHNOLOGY	說明
SSL/TLS	會針對所有服務對服務連線使用 SSL/TLS。會使用 X.509v3 憑證來建立所有 HTTPS 服務的身分識別。用戶端應一律檢查服務憑證以確保其為受信任且有效的憑證。(此配置「不會」使用用戶端憑證。)
OAuth 2.0	會使用 OAuth 2.0 登入至 Microsoft 帳戶 (MSA)/AAD v2 登入服務來產生 Bot 可用來傳送訊息的安全權杖。此權杖為服務對服務權杖，且不需要使用者登入。
JSON Web 權杖 (JWT)	會使用 JSON Web 權杖來對在 Bot 來回傳送的權杖進行加密。根據此文章所概述的需求，用戶端應完整驗證其所接收到的所有 JWT 權杖。
OpenID 中繼資料	Bot 連接器服務會發行其用來在已知的靜態端點將其 JWT 權杖簽署至 OpenID 中繼資料的有效權杖清單。

此文章說明如何透過標準 HTTPS 和 JSON 使用這些技術。不需要任何特殊的 SDK，不過適用於 OpenID 等的協助程式可能對您會有幫助。

驗證從 Bot 傳送至 Bot 連接器服務的要求

若要與 Bot 連接器服務通訊，您必須使用下列格式在每個 API 要求的 `Authorization` 標頭中指定存取權杖：

```
Authorization: Bearer ACCESS_TOKEN
```

此圖表示範 Bot 至連接器的驗證步驟：

Bot issues call to Bot Connector



IMPORTANT

若您尚未這麼做，請務必向 Bot Framework [註冊您的 Bot](#)，以取得其應用程式識別碼和密碼。您將需要 Bot 的應用程式識別碼和密碼以要求存取權杖。

步驟 1：從 MSA/AAD v2 登入服務要求存取權杖

若要從 MSA/AAD v2 登入服務要求存取權杖，請發出下列要求，並將 **MICROSOFT-APP-ID** 和 **MICROSOFT-APP-PASSWORD** 取代為您向 Bot Framework [註冊](#)您的 Bot 時所取得的應用程式識別碼和密碼。

```
POST https://login.microsoftonline.com/botframework.com/oauth2/v2.0/token
Host: login.microsoftonline.com
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&client_id=MICROSOFT-APP-ID&client_secret=MICROSOFT-APP-
PASSWORD&scope=https%3A%2F%2Fapi.botframework.com%2F.default
```

步驟 2：從 MSA/AAD v2 登入服務回應取得 JWT 權杖

若您的應用程式已由 MSA/AAD v2 登入服務授權，JSON 回應主體將會指定您的存取權杖、其類型，以及其到期時間（以秒為單位）。

將權杖加入至要求的 `Authorization` 標頭時，您必須使用於此回應中所指定的確切值（亦即不要對權杖值進行逸出或編碼）。存取權杖在到期之後便會失效。若要避免權杖到期影響 Bot 的效能，您可以選擇對權杖進行快取並主動重新整理它。

此範例示範來自 MSA/AAD v2 登入服務的回應：

```
HTTP/1.1 200 OK
...
... (other headers)
```

```
{
  "token_type": "Bearer",
  "expires_in": 3600,
  "ext_expires_in": 3600,
  "access_token": "eyJhbGciOiJIUzI1Ni..."}
```

步驟 3：在要求的 `Authorization` 標頭中指定 JWT 權杖

當您將 API 要求傳送至 Bot 連接器服務時，請使用下列格式在要求的 `Authorization` 標頭中指定存取權杖：

Authorization: Bearer ACCESS_TOKEN

您傳送至 Bot 連接器服務的所有要求，都必須在 `Authorization` 標頭中包含存取權杖。若權杖格式正確、尚未到期且是由 MSA/AAD v2 登入服務所產生，Bot 連接器服務將會授權該要求。系統會執行額外檢查，以確保權杖是屬於傳送要求的 Bot。

下列範例示範如何在要求的 `Authorization` 標頭中指定存取權杖。

```
POST https://smbsa.trafficmanager.net/apis/v3/conversations/12345/activities
Authorization: Bearer eyJhbGciOiJIUzI1Ni...
(JSON-serialized Activity message goes here)
```

IMPORTANT

請僅在傳送至 Bot 連接器服務之要求的 `Authorization` 標頭中指定 JWT 權杖。請「不要」透過不安全的通道傳送權杖，且「不要」將它包含在您傳送至其他服務的 HTTP 要求中。您從 MSA/AAD v2 登入服務取得的 JWT 權杖就像密碼一樣，因此應該謹慎處理。任何擁有該權杖的人都可以使用它來代表您的 Bot 執行作業。

Bot 至連接器：範例 JWT 元件

```
header:
{
  typ: "JWT",
  alg: "RS256",
  x5t: "<SIGNING KEY ID>",
  kid: "<SIGNING KEY ID>"
},
payload:
{
  aud: "https://api.botframework.com",
  iss: "https://sts.windows.net/d6d49420-f39b-4df7-a1dc-d59a935871db/",
  nbf: 1481049243,
  exp: 1481053143,
  appid: "<YOUR MICROSOFT APP ID>",
  ... other fields follow
}
```

NOTE

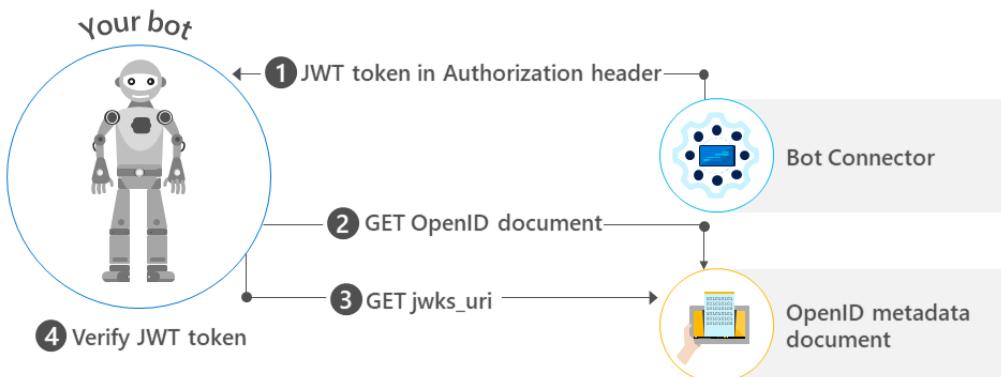
實際的欄位可能會有所不同。請以上述方式建立並驗證所有 JWT 權杖。

驗證從 Bot 連接器服務傳送至 Bot 的要求

當 Bot 連接器服務傳送要求至您的 Bot 時，它會在要求的 `Authorization` 標頭中指定已簽署的 JWT 權杖。您的 Bot 可透過確認該已簽署 JWT 權杖的真確性，以驗證來自 Bot 連接器服務的呼叫。

此圖表示範連接器至 Bot 的驗證步驟：

Bot Connector issues call to bot



步驟 2：取得 OpenID 中繼資料文件

OpenID 中繼資料文件會指定第二份文件的位置，其中會列出 Bot 連接器服務的有效簽署金鑰。若要取得 OpenID 中繼資料文件，請透過 HTTPS 發出此要求：

```
GET https://login.botframework.com/v1/.well-known/openidconfiguration
```

TIP

這是一個您可以硬式編碼至應用程式的靜態 URL。

下列範例示範回應 `GET` 要求所傳回的 OpenID 中繼資料文件。`jwks_uri` 屬性指定包含 Bot 連接器服務之有效簽署金鑰的文件位置。

```
{  
  "issuer": "https://api.botframework.com",  
  "authorization_endpoint": "https://invalid.botframework.com",  
  "jwks_uri": "https://login.botframework.com/v1/.well-known/keys",  
  "id_token_signing_alg_values_supported": [  
    "RS256"  
  ],  
  "token_endpoint_auth_methods_supported": [  
    "private_key_jwt"  
  ]  
}
```

步驟 3：取得有效簽署金鑰的清單

若要取得有效簽署金鑰的清單，請針對由 OpenID 中繼資料文件中的 `jwks_uri` 屬性所指定的 URL，透過 HTTPS 發出 `GET` 要求。例如：

```
GET https://login.botframework.com/v1/.well-known/keys
```

回應主體會以 **JWK 格式** (英文) 指定文件，但也會針對每個金鑰包含額外屬性：`endorsements`。金鑰清單本身相對穩定，並可以長時間進行快取 (在 Bot Framework SDK 中的預設值為 5 天)。

每個金鑰內的 `endorsements` 屬性都包含一或多個簽署字串，您可以使用它們來驗證連入要求之 `Activity` 物件內的 `channelId` 屬性所指定之通道識別碼的真確性。需要進行簽署之通道識別碼的清單，可在每個 Bot 內設定。根據預設，它將會是所有已發佈之通道識別碼的清單，雖然 Bot 開發人員可能會覆寫特定通道識別碼的值。若通道識別碼需要進行簽署：

- 您應該要求搭配該通道識別碼傳送至您 Bot 的任何 `Activity` 物件，都應具有針對該頻道進行簽署的 JWT 權杖。
- 若簽署不存在，您的 Bot 應傳回 **HTTP 403 (禁止)** 狀態碼來拒絕該要求。

步驟 4: 驗證 JWT 權杖

若要驗證由 Bot 連接器服務所傳送之權杖的真確性，您必須從要求的 `Authorization` 標頭擷取該權杖，對它進行剖析並驗證其內容，然後驗證其簽章。

JWT 剖析程式庫可供許多平台使用，且大多數都會針對 JWT 權杖實作安全且可靠的剖析，不過您通常必須設定這些程式庫，以要求權杖的某些特性（其簽發者、對象等）需包含正確的值。剖析權杖時，您必須設定剖析程式庫或自行撰寫驗證，以確保權杖能符合這些需求：

1. 權杖是在 HTTP `Authorization` 標頭中傳送，並具有「持有人」配置。
2. 權杖為符合 [JWT 標準](#)（英文）的有效 JSON。
3. 權杖包含具有 <https://api.botframework.com> 值的「簽發者」宣告。
4. 權杖包含具有與 Bot 的 Microsoft 應用程式識別碼相等之值的「對象」宣告。
5. 權杖仍處於其有效期間之內。業界標準的時鐘誤差為 5 分鐘。
6. 透過使用於在 [步驟 2](#) 中所擷取之 Open ID 中繼資料文件的 `id_token_signing_alg_values_supported` 屬性中所指定的簽署演算法，搭配列於在 [步驟 3](#) 中所擷取之 OpenID 金鑰文件中的金鑰，使權杖具有有效的密碼編譯簽章。
7. 權杖包含 "serviceUrl" 宣告，其值符合連入要求之 `Activity` 物件根目錄中的 `serviceUrl` 屬性。

若權杖不符合上述所有需求，您的 Bot 應傳回 **HTTP 403 (禁止)** 狀態碼來拒絕該要求。

IMPORTANT

這些需求都非常重要，特別是需求 4 和 6。若無法實作上述所有驗證需求，將會使 Bot 處於被攻擊的風險中，並可能導致 Bot 滥露其 JWT 權杖。

實作者不應該公開停用傳送至 Bot 之 JWT 權杖的驗證方法。

連接器至 Bot: 範例 JWT 元件

```
header:  
{  
  typ: "JWT",  
  alg: "RS256",  
  x5t: "<SIGNING KEY ID>",  
  kid: "<SIGNING KEY ID>"  
},  
payload:  
{  
  aud: "<YOU MICROSOFT APP ID>",  
  iss: "https://api.botframework.com",  

```

NOTE

實際的欄位可能會有所不同。請以上述方式建立並驗證所有 JWT 權杖。

驗證從 Bot Framework 模擬器傳送至 Bot 的要求

WARNING

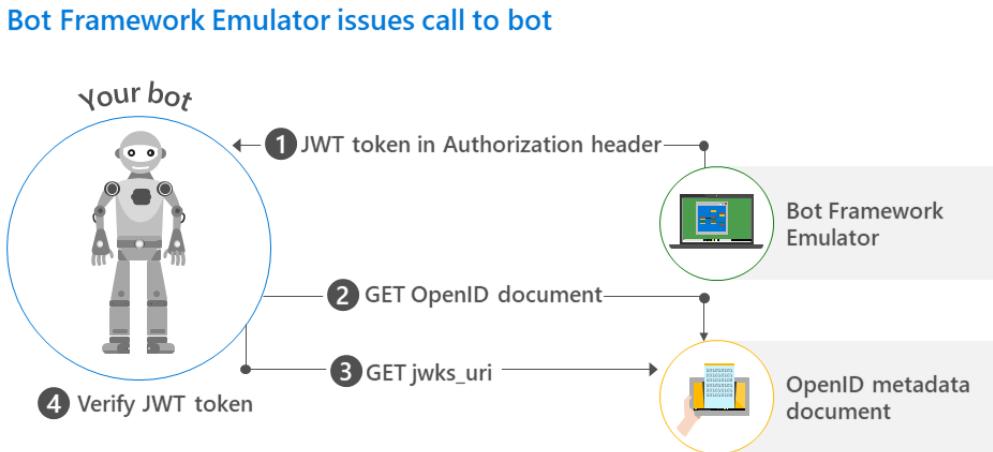
在 2017 年秋末，已推出了 Bot Framework 安全性通訊協定 v3.2。這個新的版本將會在 Bot Framework Emulator 和您的 Bot 之間交換的權杖內包含新的「簽發者」值。為了針對此變更做準備，下列步驟概述檢查 v3.1 和 v3.2 簽發者值的方法。

[Bot Framework Emulator](#) 是可用來測試 Bot 功能的傳統型工具。雖然 Bot Framework Emulator 是使用和上述相同的驗證技術，它並無法模擬真實的 Bot 連接器服務。相反地，它會使用您將模擬器連線至 Bot 時所指定的 Microsoft 應用程式識別碼和 Microsoft 應用程式密碼，來建立與 Bot 所建立的權杖完全相同的權杖。當模擬器傳送要求至您的 Bot 時，它會

在要求的 `Authorization` 標頭中指定 JWT 權杖，基本上便是使用 Bot 自己的認證來驗證要求。

若你要實作驗證程式庫，並想要接受來自 Bot Framework Emulator 的要求，您必須加入這個額外的驗證路徑。路徑的結構類似 [連接器 -> Bot](#) 的驗證路徑，但它會使用 MSA 的 OpenID 文件，而非 Bot 連接器的 OpenID 文件。

此圖表示範模擬器至 Bot 的驗證步驟：



步驟 2：取得 MSA OpenID 中繼資料文件

OpenID 中繼資料文件會指定第二份文件的位置，其中會列出有效簽署金鑰。若要取得 MSA OpenID 中繼資料文件，請透過 HTTPS 發出此要求：

```
GET https://login.microsoftonline.com/botframework.com/v2.0/.well-known/openid-configuration
```

下列範例示範回應 `GET` 要求所傳回的 OpenID 中繼資料文件。`jwks_uri` 屬性會指定包含有效簽署金鑰之文件的位置。

```
{  
    "authorization_endpoint": "https://login.microsoftonline.com/common/oauth2/v2.0/authorize",  
    "token_endpoint": "https://login.microsoftonline.com/common/oauth2/v2.0/token",  
    "token_endpoint_auth_methods_supported": ["client_secret_post", "private_key_jwt"],  
    "jwks_uri": "https://login.microsoftonline.com/common/discovery/v2.0/keys",  
    ...  
}
```

步驟 3：取得有效簽署金鑰的清單

若要取得有效簽署金鑰的清單，請針對由 OpenID 中繼資料文件中的 `jwks_uri` 屬性所指定的 URL，透過 HTTPS 發出 `GET` 要求。例如：

```
GET https://login.microsoftonline.com/common/discovery/v2.0/keys  
Host: login.microsoftonline.com
```

回應主體會以 [JWK 格式](#) (英文) 指定文件。

步驟 4：驗證 JWT 權杖

若要驗證由模擬器所傳送之權杖的真確性，您必須從要求的 `Authorization` 標頭擷取該權杖，對它進行剖析並驗證其內容，然後驗證其簽章。

JWT 剖析程式庫可供許多平台使用，且大多數都會針對 JWT 權杖實作安全且可靠的剖析，不過您通常必須設定這些程式庫，以要求權杖的某些特性（其簽發者、對象等）需包含正確的值。剖析權杖時，您必須設定剖析程式庫或自行撰寫驗證，以確保權杖能符合這些需求：

1. 權杖是在 HTTP `Authorization` 標頭中傳送，並具有「持有人」配置。
2. 權杖為符合 [JWT 標準](#) (英文) 的有效 JSON。

- 權杖包含具有 <https://sts.windows.net/d6d49420-f39b-4df7-a1dc-d59a935871db/> 或 <https://sts.windows.net/f8cdef31-a31e-4b4a-93e4-5f571e91255a/> 值的「簽發者」宣告 (檢查這兩個簽發者值將能確保您會同時檢查安全性通訊協定 v3.1 和 v3.2 的簽發者值)。
- 權杖包含具有與 Bot 的 Microsoft 應用程式識別碼相等之值的「對象」宣告。
- 權杖包含具有與 Bot 的 Microsoft 應用程式識別碼相等之值的「應用程式識別碼」宣告。
- 權杖仍處於其有效期間之內。業界標準的時鐘誤差為 5 分鐘。
- 權杖具有具備於步驟 3 中所擷取的 OpenID 金鑰文件中所列之金鑰的有效密碼編譯簽章。

NOTE

需求 5 為針對模擬器驗證路徑的特定需求。

IMPORTANT

這些需求都非常重要，特別是需求 4 和 7。若無法實作上述所有驗證需求，將會使 Bot 處於被攻擊的風險中，並可能導致 Bot 洩露其 JWT 權杖。

模擬器至 Bot：範例 JWT 元件

```
header:  
{  
  typ: "JWT",  
  alg: "RS256",  
  x5t: "<SIGNING KEY ID>",  
  kid: "<SIGNING KEY ID>"  
},  
payload:  
{  
  aud: "<YOUR MICROSOFT APP ID>",  
  iss: "https://sts.windows.net/d6d49420-f39b-4df7-a1dc-d59a935871db/",  

```

NOTE

實際的欄位可能會有所不同。請以上述方式建立並驗證所有 JWT 權杖。

安全性通訊協定變更

WARNING

針對安全性通訊協定 v3.0 的支援，已在 **2017 年 7 月 31 日** 中止。若您有自行撰寫驗證程式碼 (也就是沒有使用 Bot Framework SDK 來建立 Bot)，您必須更新應用程式以使用下列的 v3.1 值，來升級至安全性通訊協定 v3.1。

Bot 至連接器驗證

OAuth 登入 URL

通訊協定版本	有效值
v3.1 與 v3.2	https://login.microsoftonline.com/botframework.com/oauth2/v2.0/token

OAuth 範圍

通訊協定版本	有效值
v3.1 與 v3.2	<code>https://api.botframework.com/.default</code>

連接器至 Bot 驗證

OpenID 中繼資料文件

通訊協定版本	有效值
v3.1 與 v3.2	<code>https://login.botframework.com/v1/.well-known/openidconfiguration</code>

JWT 簽發者

通訊協定版本	有效值
v3.1 與 v3.2	<code>https://api.botframework.com</code>

模擬器至 Bot 驗證

OAuth 登入 URL

通訊協定版本	有效值
v3.1 與 v3.2	<code>https://login.microsoftonline.com/botframework.com/oauth2/v2.0/token</code>

OAuth 範圍

通訊協定版本	有效值
v3.1 與 v3.2	您 Bot 的 Microsoft 應用程式識別碼 + <code>/.default</code>

JWT 對象

通訊協定版本	有效值
v3.1 與 v3.2	您 Bot 的 Microsoft 應用程式識別碼

JWT 簽發者

通訊協定版本	有效值
v3.1	<code>https://sts.windows.net/d6d49420-f39b-4df7-a1dc-d59a935871db/</code>
v3.2	<code>https://sts.windows.net/f8cdef31-a31e-4b4a-93e4-5f571e91255a/</code>

OpenID 中繼資料文件

通訊協定版本	有效值
v3.1 與 v3.2	<code>https://login.microsoftonline.com/botframework.com/v2.0/.well-known/openid-configuration</code>

其他資源

- 對 Bot Framework 驗證進行疑難排解
- JSON Web 權杖 (JWT) [draft-jones-json-web-token-07](#) (英文)
- JSON Web 簽章 (JWS) [draft-jones-json-web-signature-04](#) (英文)
- JSON Web 金鑰 (JWK) [RFC 7517](#) (英文)

活動概觀

2019/2/28 • [Edit Online](#)

Bot 連接器服務透過傳遞 **Activity** 物件以在 Bot 與通道 (使用者) 之間交換資訊。最常見的活動類型是訊息，但是還有其他活動類型可用來將各種類型的資訊傳達給 Bot 或通道。

Bot 連接器服務中的活動類型

以下為 Bot 連接器服務支援的活動類型。

活動類型	說明
Message	代表 Bot 和使用者之間的通訊。
conversationUpdate	表示 Bot 已新增至對話、其他成員已新增至對話或從對話中移除，或對話中繼資料已變更。
contactRelationUpdate	表示 Bot 已新增至連絡人清單，或從使用者的連絡人清單中移除。
typing	表示使用者或在對話另一端的 Bot 正在編譯回應。
deleteUserData	表示使用者已要求 Bot 刪除任何可能已儲存的使用者資料。
endOfConversation	表示對話結束。

Message

您的 Bot 會傳送 **message** 活動以傳達資訊，及接收來自使用者的 **message** 活動。有些郵件可能是純文字，而其他郵件可能包含更豐富的內容，例如媒體附件、按鈕，以及卡片或是特定通道的資料。如需常用訊息屬性的資訊，請參閱[建立訊息](#)。如需有關如何傳送和接收訊息的資訊，請參閱[傳送和接收訊息](#)。

conversationUpdate

每當 Bot 已加入對話、其他成員已加入或從對話中移除，或對話中繼資料已變更，Bot 會收到 **conversationUpdate** 活動。

如果成員已加入對話，則活動的 `addedMembers` 屬性會識別新成員。如果成員已從對話中移除，則 `removedMembers` 屬性會識別移除的成員。

TIP

如果您的 Bot 收到 **conversationUpdate** 活動指出使用者已加入對話，您可以選擇傳送歡迎訊息給使用者做為回應。

contactRelationUpdate

每當 Bot 加入使用者的連絡人清單，或從中移除時，Bot 會收到 **contactRelationUpdate** 活動。活動的 `action` 屬性值 (新增 | 移除) 會指出 Bot 是否已加入使用者的連絡人清單，或者是否已從中移除。

typing

Bot 會收到 **typing** 活動，表示使用者正在輸入回應。Bot 會傳送 **typing** 活動，向使用者表示 Bot 正在運作以完成要求或編譯回應。

deleteUserData

當使用者要求刪除 Bot 先前為其保存的任何資料，Bot 會收到 **deleteUserData** 活動。如果您的 Bot 收到此類型的活動，就會為提出要求的使用者刪除先前已儲存的任何個人識別資訊 (PII)。

endOfConversation

Bot 會收到 **endOfConversation** 活動，表示使用者已結束對話。Bot 會傳送 **endOfConversation** 活動，向使用者表示對話已結束。

其他資源

- [建立訊息](#)
- [傳送及接收訊息](#)

建立訊息

2019/2/28 • [Edit Online](#)

Bot 會傳送訊息類型的 [Activity](#) 物件以傳達資訊給使用者，同樣也會接收來自使用者的訊息活動。某些訊息可能只包含純文字，而其他訊息可能包含更豐富的內容，例如[要讀出的文字](#)、[建議的動作](#)、[媒體附件](#)、[複合式資訊卡 \(Rich Card\)](#) 和[通道特有資料](#)。本文說明部分的常用訊息屬性。

簡訊和格式設定

使用 **plain**、**markdown** 或 **xml** 可以將訊息格式化。`textFormat` 屬性的預設格式為 **markdown**，以及使用 Markdown 格式設定標準來解譯文字。文字格式支援的層級因通道而異。若要查看您鎖定通道是否支援想使用的功能，請使用[通道偵測器](#)來預覽此功能。

[Activity](#) 物件的 `textFormat` 屬性可以用來指定文字的格式。例如，若要建立只包含純文字的基本訊息，請將 [Activity](#) 物件的 `textFormat` 屬性設定為 **plain**，將 `text` 屬性設定為訊息的內容，以及將 `locale` 屬性設定為寄件者的地區設定。

附件

[Activity](#) 物件的 `attachments` 屬性可用來傳送簡單媒體附件（影像、音訊、視訊、檔案）和複合式資訊卡（Rich Card）。如需詳細資訊，請參閱[將媒體附件新增至訊息](#)和[將複合式資訊卡 \(Rich Card\) 新增至訊息](#)。

實體

[Activity](#) 物件的 `entities` 屬性是開放式 [schema.org](#) 物件的陣列，其允許交換通道與 Bot 之間的通用內容中繼資料。

提及實體

許多通道都支援 Bot 或使用者在交談內容中「提及」某人的功能。若要在訊息中提及使用者，請以 [Mention](#) 物件填入訊息的 `entities` 屬性。

放置實體

若要在訊息中傳達[位置相關資訊](#)，請以 [Place](#) 物件填入訊息的 `entities` 屬性。

通道資料

[Activity](#) 物件的 `channelData` 屬性可以用來實作通道特有功能。如需詳細資訊，請參閱[實作通道特有功能](#)。

將文字轉換成語音

[Activity](#) 物件的 `speak` 屬性可用來指定將由 Bot 在啟用語音的通道上讀出的文字，而 [Activity](#) 物件的 `inputHint` 屬性可用來影響用戶端的麥克風狀態。如需詳細資訊，請參閱[將語音新增至訊息](#)和[將輸入提示新增至訊息](#)。

建議的動作

[Activity](#) 物件的 `suggestedActions` 屬性可用來呈現使用者可點選以提供輸入的按鈕。不同於複合式資訊卡（Rich Card）中出現的按鈕（即使在點選之後，使用者仍可看見並可存取），出現在建議動作窗格內的按鈕會在使用者進行選取後消失。如需詳細資訊，請參閱[將建議的動作新增至訊息](#)。

其他資源

- 使用頻道偵測器來預覽功能
- 活動概觀
- 傳送及接收訊息
- 將媒體附件新增至訊息
- 將複合式資訊卡 (Rich Card) 新增至訊息
- 將語音新增至訊息
- 將輸入提示新增至訊息
- 將建議的動作新增至訊息
- 實作通道特有功能

傳送及接收訊息

2019/2/28 • [Edit Online](#)

Bot Connector 服務可讓 Bot 跨多個通道 (例如 Skype、電子郵件、Slack 等) 進行通訊。它可藉由轉接從 Bot 至通道和從通道至 Bot 的活動，來協助 Bot 和使用者之間的通訊。每個活動都包含資訊，可用於將訊息以及訊息建立者、訊息的內容和訊息收件者的相關資訊，路由傳送至適當的目的地。本文說明如何使用 Bot Connector 服務來交換通道上的 Bot 和使用者之間的訊息活動。

回覆訊息

建立回覆

當使用者將傳送訊息至您的 Bot 時，Bot 會以 **message** 類型的活動物件收到訊息。若要建立使用者訊息的回覆，請建立新的活動物件，並從設定下列屬性開始：

屬性	值
對話	將此屬性設為使用者訊息中 <code>conversation</code> 屬性的內容。
from	將此屬性設為使用者訊息中 <code>recipient</code> 屬性的內容。
地區設定	將此屬性設為使用者訊息中 <code>locale</code> 屬性的內容 (若指定)。
收件者	將此屬性設為使用者訊息中 <code>from</code> 屬性的內容。
replyToId	將此屬性設為使用者訊息中 <code>id</code> 屬性的內容。
type	請將此屬性設定為 message 。

接下來，設定屬性 (該屬性會指定您想要傳達給使用者的資訊)。例如，您可以設定 `text` 屬性來指定要顯示在訊息中的文字、設定 `speak` 屬性來指定您 Bot 要說出的文字，並設定 `attachments` 屬性來指定要包含在訊息中的媒體附件或複合式資訊卡 (Rich Card)。如需常用訊息屬性的詳細資訊，請參閱[建立訊息](#)。

傳送回覆

在連入活動中使用 `serviceUrl` 屬性以識別基底 URI (您的 Bot 應該使用該 URI 來發出其回應)。

若要傳送回覆，請發出此要求：

```
POST /v3/conversations/{conversationId}/activities/{activityId}
```

在此要求 URI 中，將 `{conversationId}` 取代為您 (回覆) 活動內 `conversation` 物件的 `id` 屬性值，並將 `{activityId}` 取代為您 (回覆) 活動內 `replyToId` 屬性的值。將要求的本文設定為您所建立要代表回覆的活動物件。

下列範例會示範將簡單純文字回覆傳送給使用者郵件的要求。在此範例要求中，

`https://smbsa.trafficmanager.net/apis` 表示基底 URI，和 Bot 所提出要求的基底 URI 可能不同。如需設定基底 URI 的詳細資訊，請參閱[API 參考](#)。

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
  "type": "message",
  "from": {
    "id": "12345678",
    "name": "Pepper's News Feed"
  },
  "conversation": {
    "id": "abcd1234",
    "name": "Convo1"
  },
  "recipient": {
    "id": "1234abcd",
    "name": "SteveW"
  },
  "text": "My bot's reply",
  "replyToId": "5d5cdc723"
}
```

傳送 (非回覆) 訊息

您 Bot 所傳送的大部分訊息，會在它從使用者接收到的回覆訊息中。不過，有時候您的 Bot 可能需要將訊息傳送至並非直接回覆給任何使用者訊息的對話。例如，您的 Bot 可能需要啟動新的對話主題，或在對話結尾傳送再見訊息。

若要將訊息傳送到並非直接回覆給任何使用者訊息的對話，請發出此要求：

```
POST /v3/conversations/{conversationId}/activities
```

在此要求 URI 中，將 **{conversationId}** 取代為對話的識別碼。

將要求的本文設定為您所建立要代表回覆的活動物件。

NOTE

Bot Framework 對於 Bot 可傳送的訊息數目並無任何限制。不過，大部分的通道會強制執行節流限制，限制 Bot 在短時間內傳送大量的訊息。此外，如果 Bot 連續快速地傳送多則訊息，通道可能不一定會依照正確順序轉譯訊息。

開始對話

您的 Bot 有時候可能需要起始與一或多位使用者的對話。若要在通道上開始與使用者對話，您的 Bot 必須知道其帳戶資訊，以及該通道上的使用者帳戶資訊。

TIP

如果您的 Bot 在未來可能需要開始與其使用者對話，請快取使用者帳戶資訊、其他相關資訊（例如使用者喜好設定和地區設定）和服務 URL（用以作為「開始對話」要求中的基底 URI）。

若要開始對話，請發出此要求：

```
POST /v3/conversations
```

將要求的本文設定為**對話**物件，該物件會指定您 Bot 的帳戶資訊，和您想要包含在對話中的使用者帳戶資訊。

NOTE

並非所有通道都支援群組對話。請參閱通道的文件，判斷通道是否支援群組對話，並識別通道允許對話中的參與者數目上限。

下列範例顯示開始的對話要求。在此範例要求中，`https://smiba.trafficmanager.net/apis` 表示基底 URI，和 Bot 所提出要求的基底 URI 可能不同。如需設定基底 URI 的詳細資訊，請參閱 [API 參考](#)。

```
POST https://smiba.trafficmanager.net/apis/v3/conversations
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
  "bot": {
    "id": "12345678",
    "name": "bot's name"
  },
  "isGroup": false,
  "members": [
    {
      "id": "1234abcd",
      "name": "recipient's name"
    }
  ],
  "topicName": "News Alert"
}
```

如果對話是以指定的使用者所建立，回應就會包含識別對話的識別碼。

```
{
  "id": "abcd1234"
}
```

然後，您的 Bot 可以使用這個對話識別碼，[將訊息傳送](#)給對話中的使用者。

其他資源

- [活動概觀](#)
- [建立訊息](#)

將媒體附件新增至訊息

2019/5/10 • [Edit Online](#)

Bot 和頻道通常會交換文字字串，但某些頻道也支援交換附件，讓您的 Bot 可將更豐富的訊息傳送給使用者。例如，您的 Bot 可以傳送媒體附件（例如，影像、影片、音訊、檔案）和複合式資訊卡。本文描述如何使用 Bot 連接器服務，將媒體附件新增至訊息。

TIP

若要確認某個通道所支援的附件類型和數目，以及該通道呈現附件的方式，請參閱 [通道偵測器][通道偵測器]。

新增媒體附件

若要將媒體附件新增至訊息，請建立附件物件，設定 `name` 屬性，將 `contentUrl` 屬性設為媒體檔案的 URL，並將 `contentType` 屬性設為適當的媒體類型（例如 `image/png`、`audio/wav`、`video/mp4`）。然後，在表示訊息的活動物件中，於 `attachments` 陣列內指定您的附件物件。

下列範例所顯示的要求會傳送包含文字和單一影像附件的訊息。在此範例要求中，

`https://smba.trafficmanager.net/apis` 表示基底 URI；與 Bot 所提出要求的基底 URI 可能不同。如需設定基底 URI 的詳細資料，請參閱 [API 參考](#)。

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
    "type": "message",
    "from": {
        "id": "12345678",
        "name": "sender's name"
    },
    "conversation": {
        "id": "abcd1234",
        "name": "conversation's name"
    },
    "recipient": {
        "id": "1234abcd",
        "name": "recipient's name"
    },
    "text": "Here's a picture of the duck I was telling you about.",
    "attachments": [
        {
            "contentType": "image/png",
            "contentUrl": "https://aka.ms/DuckOnARock",
            "name": "duck-on-a-rock.jpg"
        }
    ],
    "replyToId": "5d5cdc723"
}
```

對於支援影像內嵌二進位檔的通道，您可以將 `Attachment` 的 `contentUrl` 屬性設定為影像的 base64 二進位檔（例如，`data:image/png;base64,iVBORw0KGgo...`）。該通道會在訊息的文字字串旁顯示影像或影像的 URL。

```
{  
    "type": "message",  
    "from": {  
        "id": "12345678",  
        "name": "sender's name"  
    },  
    "conversation": {  
        "id": "abcd1234",  
        "name": "conversation's name"  
    },  
    "recipient": {  
        "id": "1234abcd",  
        "name": "recipient's name"  
    },  
    "text": "Here's a picture of the duck I was telling you about.",  
    "attachments": [  
        {  
            "contentType": "image/png",  
            "contentUrl": "data:image/png;base64,iVBORw0KGgo...",  
            "name": "duck-on-a-rock.jpg"  
        }  
    ],  
    "replyToId": "5d5cdc723"  
}
```

您可以使用與影像檔上述處理相同的處理序，將影片檔或音訊檔附加到訊息。視通道而定，影片及音訊可以內嵌播放，也可能會顯示為連結。

NOTE

您的 Bot 也可能會收到包含媒體附件的訊息。比方說，如果通道可讓使用者上傳要分析的相片或要儲存的文件，Bot 所收到的訊息可能會包含附件。

新增 AudioCard 附件

新增 [AudioCard](#) 或 [VideoCard](#) 附件等同於新增媒體附件。例如，下列 JSON 顯示如何在媒體附件中新增音訊卡。

```
{
    "type": "message",
    "from": {
        "id": "12345678",
        "name": "sender's name"
    },
    "conversation": {
        "id": "abcd1234",
        "name": "conversation's name"
    },
    "recipient": {
        "id": "1234abcd",
        "name": "recipient's name"
    },
    "attachments": [
        {
            "contentType": "application/vnd.microsoft.card.audio",
            "content": {
                "title": "Allegro in C Major",
                "subtitle": "Allegro Duet",
                "text": "No Image, No Buttons, Autoloop, Autostart, Sharable",
                "duration": "PT2M55S",
                "media": [
                    {
                        "url": "https://contoso.com/media/AllegrofromDuetinCMajor.mp3"
                    }
                ],
                "shareable": true,
                "autoloop": true,
                "autostart": true,
                "value": {
                    // Supplementary parameter for this card
                }
            }
        },
        {
            "replyToId": "5d5cdc723"
        }
    ]
}
```

一旦通道收到此附件，它就會開始播放音訊檔案。比方說，如果使用者按一下 [暫停] 按鈕與音訊互動，通道將使用如下所示的 JSON 傳送回呼給 Bot：

```
{
    ...
    "type": "event",
    "name": "media/pause",
    "value": {
        "url": // URL for media
        "cardValue": {
            // Supplementary parameter for this card
        }
    }
}
```

媒體事件名稱 **media/pause** 會出現在 `activity.name` 欄位中。請參考下方的資料表，以取得所有媒體事件名稱的清單。

EVENT	說明
media/next	用戶端已跳到下一個媒體
media/pause	用戶端已暫停播放媒體

EVENT	說明
media/play	用戶端已開始播放媒體
media/previous	用戶端已跳到上一個媒體
media/resume	用戶端已繼續播放媒體
media/stop	用戶端已停止播放媒體

其他資源

- [建立訊息](#)
- [傳送及接收訊息](#)
- [將複合式資訊卡 \(Rich Card\) 新增至訊息](#)
- [Bot Framework 卡片結構描述](#)

將複合式資訊卡 (Rich Card) 附件新增至訊息

2019/5/10 • [Edit Online](#)

Bot 和頻道通常會交換文字字串，但某些頻道也支援交換附件，讓您的 Bot 可將更豐富的訊息傳送給使用者。例如，您的 Bot 可以傳送複合式資訊卡和媒體附件（例如，影像、視訊、音訊、檔案）。本文說明如何使用 Bot Connector 服務，將複合式資訊卡附件新增至訊息。

NOTE

如需如何將媒體附件新增至訊息的相關資訊，請參閱[將媒體附件新增至訊息](#)。

複合式資訊卡的類型

複合式資訊卡包含標題、描述、連結和影像。訊息可包含多個複合式資訊卡，並以清單格式或浮動切換格式顯示。Bot Framework 目前支援八種類型的複合式資訊卡：

卡片類型	說明
AdaptiveCard	可自訂的卡片，可包含文字、語音、影像、按鈕和輸入欄位的任意組合。請參閱 個別頻道支援 。
AnimationCard	可播放動畫 GIF 或短片的卡片。
AudioCard	可播放音訊檔案的卡片。
HeroCard	通常包含單一大型影像、一或多個按鈕和文字的卡片。
ThumbnailCard	通常包含單一縮圖影像、一或多個按鈕和文字的卡片。
ReceiptCard	讓 Bot 向使用者提供收據的卡片。通常包含收據上的項目清單（稅金和總金額資訊）和其他文字。
SignInCard	可讓 Bot 要求使用者登入的卡片。通常包含文字，以及一或多個可讓使用者點選以起始登入程序的按鈕。
VideoCard	可播放視訊的卡片。

TIP

若要確認某個頻道所支援的複合式資訊卡類型，以及查看該頻道轉譯複合式資訊卡的方式，請參閱[頻道偵測器](#)。如需卡片內容的限制（例如，按鈕數目上限或標題長度上限）的相關資訊，請參閱頻道的文件。

處理複合式資訊卡內的事件

若要處理複合式資訊卡內的事件，請使用 [CardAction](#) 物件，指定在使用者按一下按鈕或點選卡片的某區段時應發生的情況。每個 [CardAction](#) 物件均包含下列屬性：

屬性	類型	說明
type	字串	動作的類型 (下表中指定的其中一個值)
title	字串	按鈕的標題
image	字串	按鈕的影像 URL
value	字串	執行指定動作類型所需的值

NOTE

調適型卡片內的按鈕不會使用 `CardAction` 物件來建立，而會改用[調適型卡片](#)所定義的結構描述來建立。如需示範如何將按鈕新增至調適型卡片的範例，請參閱[將調適型卡片新增至訊息](#)。

下表列出 `CardAction` 物件的 `type` 屬性適用的有效值，並說明每個類型之 `value` 屬性的預期內容：

TYPE	VALUE
openUrl	要在內建瀏覽器中開啟的 URL
imBack	要傳送至 Bot 的訊息文字 (來自於按一下按鈕或點選卡片的使用者)。此訊息 (從使用者到 Bot) 會透過裝載對話的用戶端應用程式顯示給所有對話參與者。
postBack	要傳送至 Bot 的訊息文字 (來自於按一下按鈕或點選卡片的使用者)。某些用戶端應用程式可能會在訊息摘要中顯示此文字，讓所有對話參與者都能看見。
call	以下列格式撥打電話的目的地: <code>tel:123123123123</code>
playAudio	待播放音訊的 URL
playVideo	待播放視訊的 URL
showImage	待顯示影像的 URL
downloadFile	待下載檔案的 URL
signin	待起始 OAuth 流程的 URL

將主圖卡新增至訊息

若要將複合式資訊卡附件新增至訊息，請先建立對應至您想要新增至訊息之[卡片類型](#)的物件。接著建立 `Attachment` 物件、將它的 `contentType` 屬性設定為卡片的媒體類型，並將它的 `content` 屬性設定為您建立來代表卡片的物件。在訊息的 `attachments` 陣列內，指定您的 `Attachment` 物件。

TIP

包含複合式資訊卡附件的訊息通常不會指定 `text`。

某些頻道可讓您將多張複合式資訊卡新增至訊息內的 `attachments` 陣列。此功能在您想要為使用者提供多個選項的情況下很有用。例如，如果您的 Bot 讓使用者能夠預約飯店房間，即可為使用者呈現一份複合式資訊卡清單，以顯示各種類型的可用房間。每張卡片都會包含一張圖片，以及與房間類型相對應的便利設施清單，而使用者可以藉由點選卡片或按一下按鈕來選取房間類型。

TIP

若要以清單格式顯示多張複合式資訊卡，請將 `Activity` 物件的 `attachmentLayout` 屬性設定為 "list"。若要以浮動切換格式顯示多張複合式資訊卡，請將 `Activity` 物件的 `attachmentLayout` 屬性設定為 "carousel"。如果頻道不支援浮動切換格式，則將以清單格式顯示複合式資訊卡，即使 `attachmentLayout` 屬性指定 "carousel" 也一樣。

下列範例所顯示的要求會傳送包含單一主圖卡附件的訊息。在此範例要求中，

`https://smba.trafficmanager.net/apis` 表示基底 URI，您的 Bot 所發出之要求的基底 URI 可能和這個不同。如需設定基底 URI 的詳細資訊，請參閱 [API 參考](#)。

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
    "type": "message",
    "from": {
        "id": "12345678",
        "name": "sender's name"
    },
    "conversation": {
        "id": "abcd1234",
        "name": "conversation's name"
    },
    "recipient": {
        "id": "1234abcd",
        "name": "recipient's name"
    },
    "attachments": [
        {
            "contentType": "application/vnd.microsoft.card.hero",
            "content": {
                "title": "title goes here",
                "subtitle": "subtitle goes here",
                "text": "descriptive text goes here",
                "images": [
                    {
                        "url": "https://aka.ms/DuckOnARock",
                        "alt": "picture of a duck",
                        "tap": {
                            "type": "playAudio",
                            "value": "url to an audio track of a duck call goes here"
                        }
                    }
                ],
                "buttons": [
                    {
                        "type": "playAudio",
                        "title": "Duck Call",
                        "value": "url to an audio track of a duck call goes here"
                    },
                    {
                        "type": "openUrl",
                        "title": "Watch Video",
                        "image": "https://aka.ms/DuckOnARock",
                        "value": "url goes here of the duck in flight"
                    }
                ]
            }
        }
    ],
    "replyToId": "5d5cdc723"
}
```

將調適型卡片新增至訊息

調適型卡片可包含文字、語音、影像、按鈕和輸入欄位的任意組合。調適型卡片會使用[調適型卡片](#)中指定的 JSON 格式來建立，讓您能夠完整控制卡片的內容和格式。

請利用[調適型卡片](#)網站中的資訊，來了解調適型卡片的結構描述、探索調適型卡片的元素，並查看可用以建立具有不同組合和複雜度之卡片的 JSON 範例。此外，您可以使用互動式視覺化檢視，來設計調適型卡片承載及預覽卡片輸出。

下列範例所顯示的要求會傳送包含單一調適型卡片作為行事曆提醒的訊息。在此範例要求中，

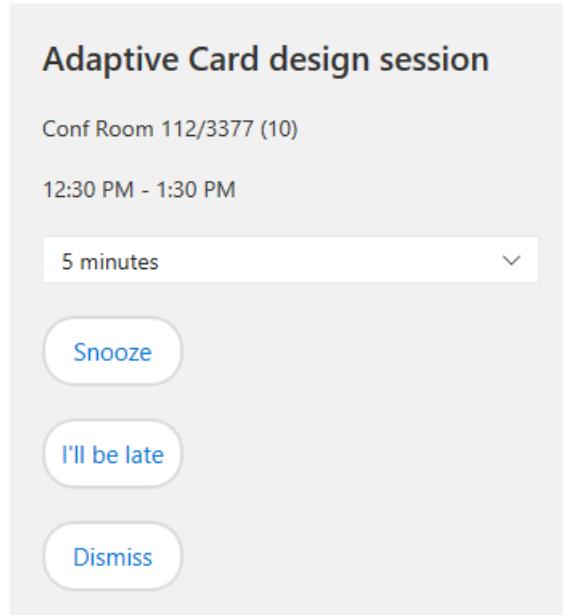
`https://smbs.trafficmanager.net/apis` 表示基底 URI，您的 Bot 所發出之要求的基底 URI 可能和這個不同。如需設定基底 URI 的詳細資訊，請參閱[API 參考](#)。

```
POST https://smaba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
    "type": "message",
    "from": {
        "id": "12345678",
        "name": "sender's name"
    },
    "conversation": {
        "id": "abcd1234",
        "name": "conversation's name"
    },
    "recipient": {
        "id": "1234abcd",
        "name": "recipient's name"
    },
    "attachments": [
        {
            "contentType": "application/vnd.microsoft.card.adaptive",
            "content": {
                "type": "AdaptiveCard",
                "body": [
                    {
                        "type": "TextBlock",
                        "text": "Adaptive Card design session",
                        "size": "large",
                        "weight": "bolder"
                    },
                    {
                        "type": "TextBlock",
                        "text": "Conf Room 112/3377 (10)"
                    },
                    {
                        "type": "TextBlock",
                        "text": "12:30 PM - 1:30 PM"
                    },
                    {
                        "type": "TextBlock",
                        "text": "Snooze for"
                    },
                    {
                        "type": "Input.ChoiceSet",
                        "id": "snooze",
                        "style": "compact",
                        "choices": [
                            {
                                "title": "5 minutes",
                                "value": "5",
                                "isSelected": true
                            },
                            {
                                "title": "15 minutes",
                                "value": "15"
                            },
                            {
                                "title": "30 minutes",
                                "value": "30"
                            }
                        ]
                    },
                    "actions": [
                        {
                            "type": "Action.Http",
                            "method": "POST",
                            "url": "https://smaba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723"
                        }
                    ]
                ]
            }
        }
    ]
}
```

```
        "method": "POST",
        "url": "http://foo.com",
        "title": "Snooze"
    },
    {
        "type": "Action.Http",
        "method": "POST",
        "url": "http://foo.com",
        "title": "I'll be late"
    },
    {
        "type": "Action.Http",
        "method": "POST",
        "url": "http://foo.com",
        "title": "Dismiss"
    }
]
}
],
"replyToId": "5d5cdc723"
}
```

產生的卡片會包含三個文字區塊、一個輸入欄位 (選擇清單) 和三個按鈕：



其他資源

- [建立訊息](#)
- [傳送及接收訊息](#)
- [將媒體附件新增至訊息](#)
- [頻道偵測器](#)
- [調適型卡片](#)

將語音新增至訊息

2019/2/28 • [Edit Online](#)

如果您要為具備語音功能的通道 (例如 Cortana) 建置 Bot，您可以建構訊息，其中指定要由 Bot 讀出的文字。您也可以指定[輸入提示](#)，藉由指出您的 Bot 要接受、需要或忽略使用者輸入，來嘗試影響用戶端的麥克風狀態。

指定要由您的 Bot 讀出的文字

若要指定將由 Bot 在啟用語音通道上讀出的文字，請在代表您訊息的 [Activity](#) 物件中設定 `speak` 屬性。您可以將 `speak` 屬性設定為純文字字串或格式化為[語音合成標記語言 \(SSML\)](#) 的字串，SSML 是一種以 XML 為基礎的標記語言，可讓您控制 Bot 語音的各種特性，例如聲音、速率、音量、發音、音調等等。

下列要求會傳送一則訊息，其中指定要顯示的文字和要讀出的文字，並指出 Bot 需要使用者輸入。它會使用

[SSML](#) 格式來指定 `speak` 屬性，表示應以適度強調的方式讀出 "sure" 這個字。在此範例要求中，

`https://smba.trafficmanager.net/apis` 表示基底 URI，和您提出要求之 Bot 的基底 URI 可能不同。如需設定基底 URI 的詳細資訊，請參閱 [API 參考](#)。

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
    "type": "message",
    "from": {
        "id": "12345678",
        "name": "sender's name"
    },
    "conversation": {
        "id": "abcd1234",
        "name": "conversation's name"
    },
    "recipient": {
        "id": "1234abcd",
        "name": "recipient's name"
    },
    "text": "Are you sure that you want to cancel this transaction?",
    "speak": "Are you <emphasis level='moderate'>sure</emphasis> that you want to cancel this transaction?",
    "inputHint": "expectingInput",
    "replyToId": "5d5cdc723"
}
```

輸入提示

當您在啟用語音功能的通道上傳送訊息時，您也可以藉由包含輸入提示以指出 Bot 要接受、需要或忽略使用者輸入，來嘗試影響用戶端的麥克風狀態。如需詳細資訊，請參閱[將輸入提示新增至訊息](#)。

其他資源

- [建立訊息](#)
- [傳送及接收訊息](#)
- [將輸入提示新增至訊息](#)
- [語音合成標記語言 \(SSML\)](#)

將輸入提示新增至訊息

2019/2/28 • [Edit Online](#)

您可藉由指定訊息的輸入提示，指出您的 Bot 在訊息傳遞給用戶端之後，會接受、需要或忽略使用者輸入。這可讓用戶端為許多通道設定相應的使用者輸入控制項狀態。例如，如果訊息的輸入提示指出 Bot 會忽略使用者輸入，則用戶端可關閉麥克風並停用輸入方塊，以防止使用者提供輸入。

接受輸入

若要指出您的 Bot 是被動接受輸入，但不等候使用者回應，請將表示您訊息之 [Activity](#) 物件內的 `inputHint` 屬性設定為 `acceptingInput`。在許多通道上，這會啓用用戶端的輸入方塊並關閉麥克風，但使用者仍可存取。例如，如果使用者按住 [麥克風] 按鈕，Cortana 會開啟麥克風接受使用者輸入。

下列範例示範傳送訊息並指定 Bot 接受輸入的要求。在此範例要求中，<https://smba.trafficmanager.net/apis> 表示基底 URI，和您提出要求之 Bot 的基底 URI 可能不同。如需設定基底 URI 的詳細資料，請參閱 [API 參考](#)。

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
    "type": "message",
    "from": {
        "id": "12345678",
        "name": "sender's name"
    },
    "conversation": {
        "id": "abcd1234",
        "name": "conversation's name"
    },
    "recipient": {
        "id": "1234abcd",
        "name": "recipient's name"
    },
    "text": "Here's a picture of the house I was telling you about.",
    "inputHint": "acceptingInput",
    "replyToId": "5d5cdc723"
}
```

必須是輸入

若要指出您的 Bot 要等候使用者回應，請將表示您訊息之 [Activity](#) 物件內的 `inputHint` 屬性設定為 `expectingInput`。在許多通道上，這會啓用用戶端的輸入方塊並開啓麥克風。

下列範例示範傳送訊息並指定 Bot 需要輸入的要求。在此範例要求中，<https://smba.trafficmanager.net/apis> 表示基底 URI，和您提出要求之 Bot 的基底 URI 可能不同。如需設定基底 URI 的詳細資料，請參閱 [API 參考](#)。

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{  
    "type": "message",  
    "from": {  
        "id": "12345678",  
        "name": "sender's name"  
    },  
    "conversation": {  
        "id": "abcd1234",  
        "name": "conversation's name"  
    },  
    "recipient": {  
        "id": "1234abcd",  
        "name": "recipient's name"  
    },  
    "text": "What is your favorite color?",  
    "inputHint": "expectingInput",  
    "replyToId": "5d5cdc723"  
}
```

忽略輸入

若要指出您的 Bot 尚未準備接收使用者輸入，請將表示您訊息之 **Activity** 物件內的 `inputHint` 屬性設定為 `ignoringInput`。在許多通道上，這會停用用戶端的輸入方塊並關閉麥克風。

下列範例示範傳送訊息並指定 Bot 忽略輸入的要求。在此範例要求中，<https://smiba.trafficmanager.net/apis> 表示基底 URI，和您提出要求之 Bot 的基底 URI 可能不同。如需設定基底 URI 的詳細資料，請參閱 [API 參考](#)。

```
POST https://smiba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723  
Authorization: Bearer ACCESS_TOKEN  
Content-Type: application/json
```

```
{  
    "type": "message",  
    "from": {  
        "id": "12345678",  
        "name": "sender's name"  
    },  
    "conversation": {  
        "id": "abcd1234",  
        "name": "conversation's name"  
    },  
    "recipient": {  
        "id": "1234abcd",  
        "name": "recipient's name"  
    },  
    "text": "Please hold while I perform the calculation.",  
    "inputHint": "ignoringInput",  
    "replyToId": "5d5cdc723"  
}
```

其他資源

- [建立訊息](#)
- [傳送及接收訊息](#)

將建議的動作新增至訊息

2019/5/10 • [Edit Online](#)

建議動作可讓您的 Bot 顯示可供使用者點選，以提供輸入的按鈕。建議動作會出現在編輯器附近，讓使用者只要點選按鈕即可回答問題或進行選取，而無需使用鍵盤輸入回應，藉以提升使用者體驗。不同於複合式資訊卡 (Rich Card) 中出現的按鈕 (即使在點選之後，使用者仍可看見並可存取)，出現在建議動作窗格內的按鈕會在使用者進行選取後消失。這可以避免使用者在對話中點選過時的按鈕，並簡化 Bot 的開發 (因為您不需要再說明該情境)。

傳送建議的動作

若要將建議的動作新增至訊息，請設定活動的 `suggestedActions` 屬性，以指定 `CardAction` 物件清單，這些物件代表要向使用者呈現的按鈕。

下列要求會傳送一則訊息，向使用者呈現三個建議的動作。在此範例要求中，

`https://smba.trafficmanager.net/apis` 表示基底 URI；與 Bot 所提出要求的基底 URI 可能不同。如需設定基底 URI 的詳細資料，請參閱 [API 參考](#)。

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{  
    "type": "message",  
    "from": {  
        "id": "12345678",  
        "name": "sender's name"  
    },  
    "conversation": {  
        "id": "abcd1234",  
        "name": "conversation's name"  
    },  
    "recipient": {  
        "id": "1234abcd",  
        "name": "recipient's name"  
    },  
    "text": "I have colors in mind, but need your help to choose the best one.",  
    "inputHint": "expectingInput",  
    "suggestedActions": {  
        "actions": [  
            {  
                "type": "imBack",  
                "title": "Blue",  
                "value": "Blue"  
            },  
            {  
                "type": "imBack",  
                "title": "Red",  
                "value": "Red"  
            },  
            {  
                "type": "imBack",  
                "title": "Green",  
                "value": "Green"  
            }  
        ]  
    },  
    "replyToId": "5d5cdc723"  
}
```

當使用者點選其中一個建議的動作時，Bot 會收到來自使用者的訊息，其中包含對應動作的 `value`。

其他資源

- [建立訊息](#)
- [傳送及接收訊息](#)

實作通道特定的功能

2019/5/10 • [Edit Online](#)

某些通道所提供的功能，無法只透過訊息文字和附件來實作。若要實作通道特定的功能，您可以將原生中繼資料傳遞至 **Activity** 物件之 `channelData` 屬性中的通道。例如，Bot 可以使用 `channelData` 屬性，指示 Telegram 傳送貼圖或指示 Office365 傳送電子郵件。

此文章說明如何使用訊息活動的 `channelData` 屬性來實作此通道特定的功能：

通道	功能
電子郵件	傳送及接收電子郵件，其中包含本文、主旨和重要性中繼資料
Slack	傳送不失真的 Slack 訊息
Facebook	原生傳送 Facebook 通知
Telegram	執行 Telegram 特定動作，例如共用語音備忘或貼圖
Kik	傳送和接收原生 Kik 訊息

NOTE

Activity 物件的 `channelData` 屬性值是一個 JSON 物件。JSON 物件的結構將根據正在實作的通道和功能而變化，如下所述。

建立自訂的電子郵件訊息

若要建立電子郵件訊息，請將 **Activity** 物件的 `channelData` 屬性設定為包含這些屬性的 JSON 物件：

屬性	說明
htmlBody	用於訊息本文的 HTML。
主旨	用於訊息的主旨。
importance	用於訊息的重要性旗標： <code>low</code> 、 <code>normal</code> 或 <code>high</code> 。
toRecipients	以分號 (:) 分隔的電子郵件地址字串，用來新增至訊息的 [收件者] 欄位。
ccRecipients	以分號 (:) 分隔的電子郵件地址字串，用來新增至訊息的 [Cc] (副本)欄位。
bccRecipients	以分號 (:) 分隔的電子郵件地址字串，用來新增至訊息的 [Bcc] (密件副本)欄位。

此程式碼片段顯示自訂電子郵件訊息的 `channelData` 屬性範例。

```
"channelData":  
{  
    "htmlBody": "<html><body style = /\"font-family: Calibri; font-size: 11pt;\">This is more than awesome.  
    </body></html>",  
    "subject": "Super awesome message subject",  
    "importance": "high",  
    "ccRecipients": "Yasemin@adatum.com;Temel@adventure-works.com"  
}
```

建立不失真的 Slack 訊息

若要建立不失真的 Slack 訊息，請將 [Activity](#) 物件的 `channelData` 屬性設定為 JSON 物件，以指定 [Slack 訊息](#)、[Slack 附件](#) 和/或 [Slack 按鈕](#)。

NOTE

若要在 Slack 訊息中支援按鈕，您必須在[將 Bot 連線](#)至 Slack 通道時啟用 [互動式訊息]。

此程式碼片段顯示自訂 Slack 訊息的 `channelData` 屬性範例。

```

"channelData": {
    "text": "Now back in stock! :tada:",
    "attachments": [
        {
            "title": "The Further Adventures of Slackbot",
            "author_name": "Stanford S. Strickland",
            "author_icon": "https://api.slack.com/img/api/homepage_custom_integrations-2x.png",
            "image_url": "http://i.imgur.com/OJkaVOI.jpg?1"
        },
        {
            "fields": [
                {
                    "title": "Volume",
                    "value": "1",
                    "short": true
                },
                {
                    "title": "Issue",
                    "value": "3",
                    "short": true
                }
            ]
        },
        {
            "title": "Synopsis",
            "text": "After @episod pushed exciting changes to a devious new branch back in Issue 1, Slackbot notifies @don about an unexpected deploy..."
        },
        {
            "fallback": "Would you recommend it to customers?",
            "title": "Would you recommend it to customers?",
            "callback_id": "comic_1234_xyz",
            "color": "#3AA3E3",
            "attachment_type": "default",
            "actions": [
                {
                    "name": "recommend",
                    "text": "Recommend",
                    "type": "button",
                    "value": "recommend"
                },
                {
                    "name": "no",
                    "text": "No",
                    "type": "button",
                    "value": "bad"
                }
            ]
        }
    ]
}

```

當使用者按一下 Slack 訊息中的按鈕時，Bot 會收到回應訊息，其中的 `channelData` 屬性已填入 `payload` JSON 物件。`payload` 物件會指定原始訊息的內容、識別已按下的按鈕，並識別按下按鈕的使用者。

此程式碼片段顯示當使用者按一下 Slack 訊息中的按鈕時，Bot 所收到的訊息中所含有的 `channelData` 屬性範例。

```

"channelData": {
    "payload": {
        "actions": [
            {
                "name": "recommend",
                "value": "yes"
            }
        ],
        ...
    },
    "original_message": "...",
    "response_url": "https://hooks.slack.com/actions/..."
}
}

```

Bot 可以透過[正常方式](#)回覆此訊息，也可以將其回應直接張貼到 `payload` 物件之 `response_url` 屬性所指定的端點。如需何時及如何將回應張貼到 `response_url` 的資訊，請參閱 [Slack 按鈕](#)。

建立 Facebook 通知

若要建立 Facebook 通知，請將 `Activity` 物件的 `channelData` 屬性設定為指定這些屬性的 JSON 物件：

屬性	說明
<code>notification_type</code>	通知類型 (例如 REGULAR 、 SILENT_PUSH 、 NO_PUSH)。
<code>attachment</code>	指定影像、視訊或其他多媒體類型的附件，或是回條等樣板化附件。

NOTE

如需 `notification_type` 屬性和 `attachment` 屬性的格式和內容詳細資料，請參閱 [Facebook API 文件](#)。

此程式碼片段顯示 Facebook 回條附件的 `channelData` 屬性範例。

```

"channelData": {
    "notification_type": "NO_PUSH",
    "attachment": {
        "type": "template",
        "payload": {
            "template_type": "receipt",
            ...
        }
    }
}

```

建立 Telegram 訊息

若要建立訊息來實作 Telegram 特定動作 (例如，共用語音備忘或貼圖)，請將 `Activity` 物件的 `channelData` 屬性設定為會指定這些屬性的 JSON 物件：

屬性	說明
<code>method</code>	要呼叫的 Telegram Bot API 方法。
<code>parameters</code>	所指定方法的參數。

支援下列 Telegram 方法：

- answerInlineQuery
- editMessageCaption
- editMessageReplyMarkup
- editMessageText
- forwardMessage
- kickChatMember
- sendAudio
- sendChatAction
- sendContact
- sendDocument
- sendLocation
- sendMessage
- sendPhoto
- sendSticker
- sendVenue
- sendVideo
- sendVoice
- unbanChateMember

如需這些 Telegram 方法和其參數的詳細資訊，請參閱 [Telegram Bot API 文件](#)。

NOTE

- `chat_id` 參數通用於所有 Telegram 方法。如果您未指定 `chat_id` 作為參數，架構會為您提供識別碼。
- 不要傳遞內嵌檔案內容，而是應該指定使用 URL 和媒體類型的檔案，如下列範例所示。
- 在 Bot 從 Telegram 通道所收到的每則訊息內，`channelData` 屬性會包含 Bot 先前傳送的訊息。

此程式碼片段顯示 `channelData` 屬性範例，此屬性指定單一 Telegram 方法。

```
"channelData": {  
    "method": "sendSticker",  
    "parameters": {  
        "sticker": {  
            "url": "https://domain.com/path/gif",  
            "mediaType": "image/gif",  
        }  
    }  
}
```

此程式碼片段顯示 `channelData` 屬性範例，此屬性指定 Telegram 方法陣列。

```

"channelData": [
    {
        "method": "sendSticker",
        "parameters": {
            "sticker": {
                "url": "https://domain.com/path/gif",
                "mediaType": "image/gif",
            }
        }
    },
    {
        "method": "sendMessage",
        "parameters": {
            "text": "<b>This message is HTML formatted.</b>",
            "parse_mode": "HTML"
        }
    }
]

```

建立原生的 Kik 訊息

若要建立原生的 Kik 訊息，請將 [Activity](#) 物件的 `channelData` 屬性設定為指定此屬性的 JSON 物件：

屬性	說明
上限	Kik 訊息的陣列。如需 Kik 訊息格式的詳細資訊，請參閱 Kik 訊息格式 。

此程式碼片段顯示原生 Kik 訊息的 `channelData` 屬性範例。

```

"channelData": {
    "messages": [
        {
            "chatId": "c6dd8165...",
            "type": "link",
            "to": "kikhangle",
            "title": "My Webpage",
            "text": "Some text to display",
            "url": "http://botframework.com",
            "picUrl": "http://lorempixel.com/400/200/",
            "attribution": {
                "name": "My App",
                "iconUrl": "http://lorempixel.com/50/50/"
            },
            "noForward": true,
            "kikJsData": {
                "key": "value"
            }
        }
    ]
}

```

其他資源

- [活動概觀](#)
- [建立訊息](#)
- [傳送及接收訊息](#)
- [使用通道偵測器預覽功能](#)

管理狀態資料

2019/2/28 • [Edit Online](#)

Bot 狀態服務可讓您的 Bot 儲存及擷取與使用者、對話或特定對話內容中特定使用者相關聯的狀態資料。您最多可以為頻道上的每個使用者、頻道上的每個對話，以及頻道上對話內容中的每個使用者儲存 32 KB 的資料。狀態資料有許多用途，例如判斷先前離開對話的位置，或單純地以名字向回來的使用者問候。如果您儲存使用者的喜好設定，下次聊天時就可以使用該資訊來自訂對話。例如，您可能會在有使用者感興趣的主題之新聞文章，或有可用約會時通知使用者。

IMPORTANT

建議您不要將 Bot Framework 狀態服務 API 用於生產環境，未來版本可能會將它淘汰。建議更新您的 Bot 程式碼以使用記憶體內部儲存體進行測試，或將其中一個 Azure 擴充功能用於生產環境 Bot。如需詳細資訊，請參閱 [.NET](#) 或 [Node](#) 實作的〈管理狀態資料〉主題。

資料並行

為控制使用 Bot 狀態服務儲存的資料並行，架構針對 `POST` 要求使用實體標記 (Etag)。此架構不會使用標準的 Etag 標頭。相反地，要求和回應的主體使用 `eTag` 屬性指定 Etag。

例如，如果您發出 `GET` 要求以從存放區擷取使用者資料，回應將會包含 `eTag` 屬性。如果您變更資料並發出 `POST` 要求以將更新的資料儲存到存放區，您的要求可以包含 `eTag` 屬性，並為此屬性指定與稍早在 `GET` 回應中收到值相同的值。如您的 `POST` 要求中所指定的 Etag 符合儲存體中目前的值，伺服器就會儲存使用者的資料並回應 **HTTP 200 成功**，然後在回應的主體中指定新的 `eTag` 值。如果您 `POST` 要求中指定的 Etag 不符合存放區中目前的值，伺服器將會回應「**HTTP 412 前置條件失敗**」，以表示自從您上次儲存或擷取之後，存放區中的使用者資料已變更。

TIP

`GET` 回應將一律包含 `eTag` 屬性，但您不需要在 `GET` 要求中指定 `eTag` 屬性。星號 (*) 的 `eTag` 屬性值表示您先前未儲存所指定頻道、使用者與對話之組合的資料。

在 `POST` 要求中指定 `eTag` 屬性是選擇性的。如果並行對於您的 Bot 不是問題，請勿在 `POST` 要求中包含 `eTag` 屬性。

儲存使用者資料

若要儲存特定頻道上使用者的狀態資料，請發出此要求：

```
POST /v3/botstate/{channelId}/users/{userId}
```

在此要求 URI 中，將 `{channelId}` 取代為頻道的識別碼，並將 `{userId}` 取代為頻道上的使用者識別碼。您的 Bot 先前從使用者收到的任何訊息中的 `channelId` 和 `from` 屬性都會包含這些識別碼。您也可以將這些值快取在安全的位置，這樣就可以在未來存取使用者的資料，而不需要從訊息中擷取它們。

將要求的主體設定為 `BotData` 物件，其中 `data` 屬性指定您要儲存的資料。如果您將實體標記用於 **並行控制**，請將 `eTag` 屬性設定為您上次儲存或擷取使用者資料時收到的 Etag (視孰者較新)。如果您不將實體標記用於並行，則請勿在要求中包含 `eTag` 屬性。

NOTE

只要當指定的 Etag 符合伺服器的 Etag, 或未在要求中指定任何 Etag 時, 此 POST 要求才會覆寫存放區中的使用者資料。

要求

下列範例顯示儲存特定頻道上使用者資料的要求。在此範例要求中, `https://smbs.trafficmanager.net/apis` 表示基底 URI, 您的 Bot 所發出之要求的基底 URI 可能和這個不同。如需設定基底 URI 的詳細資料, 請參閱 [API 參考](#)。

```
POST https://smbs.trafficmanager.net/apis/v3/botstate/abcd1234/users/12345678
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
  "data": [
    {
      "trail": "Lake Serene",
      "miles": 8.2,
      "difficulty": "Difficult",
    },
    {
      "trail": "Rainbow Falls",
      "miles": 6.3,
      "difficulty": "Moderate",
    }
  ],
  "eTag": "a1b2c3d4"
}
```

Response

回應將會包含具有新 `eTag` 值的 `BotData` 物件。

取得使用者資料

若要取得先前為特定頻道上使用者儲存的狀態資料, 請發出此要求:

```
GET /v3/botstate/{channelId}/users/{userId}
```

在此要求 URI 中, 將 `{channelId}` 取代為頻道的識別碼, 並將 `{userId}` 取代為頻道上的使用者識別碼。您的 Bot 先前從使用者收到的任何訊息中的 `channelId` 和 `from` 屬性都會包含這些識別碼。您也可以將這些值快取在安全的位置, 這樣就可以在未來存取使用者的資料, 而不需要從訊息中擷取它們。

要求

下列範例顯示取得先前為使用者儲存之資料的要求。在此範例要求中, `https://smbs.trafficmanager.net/apis` 表示基底 URI, 您的 Bot 所發出之要求的基底 URI 可能和這個不同。如需設定基底 URI 的詳細資料, 請參閱 [API 參考](#)。

```
GET https://smbs.trafficmanager.net/apis/v3/botstate/abcd1234/users/12345678
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

Response

回應將會包含 `BotData` 物件, 其中 `data` 屬性包含您先前為使用者儲存的資料, 而 `eTag` 屬性包含您在後續儲存使用者資料的要求中使用的 Etag。如果您先前沒有為使用者儲存資料, 則 `data` 屬性將會是 `null`, 而且 `eTag` 屬

性將會包含星號 (*)。

此範例顯示 GET 要求的回應：

```
{  
  "data": [  
    {  
      "trail": "Lake Serene",  
      "miles": 8.2,  
      "difficulty": "Difficult",  
    },  
    {  
      "trail": "Rainbow Falls",  
      "miles": 6.3,  
      "difficulty": "Moderate",  
    }  
],  
  "eTag": "xyz12345"  
}
```

儲存對話資料

若要儲存特定頻道上對話的狀態資料，請發出此要求：

```
POST /v3/botstate/{channelId}/conversations/{conversationId}
```

在此要求 URI 中，將 {channelId} 取代為頻道的識別碼，並將 {conversationId} 取代為對話的識別碼。您的 Bot 先前在對話內容中傳送或接收的任何訊息內的 channelId 和 conversation 屬性將會包含這些識別碼。您也可以將這些值快取在安全的位置，這樣就可以在未來存取對話的資料，而不需要從訊息中擷取它們。

將要求主體設定為 [BotData](#) 物件，如 [儲存使用者資料](#) 中所述。

IMPORTANT

因為 [刪除使用者資料](#) 作業不會刪除已經使用 [儲存對話資料](#) 作業儲存的資料，請勿使用此作業儲存使用者的個人識別資訊 (PII)。

Response

回應將會包含具有新 eTag 值的 [BotData](#) 物件。

取得對話資料

若要取得先前為特定頻道上對話儲存的狀態資料，請發出此要求：

```
GET /v3/botstate/{channelId}/conversations/{conversationId}
```

在此要求 URI 中，將 {channelId} 取代為頻道的識別碼，並將 {conversationId} 取代為對話的識別碼。您的 Bot 先前在對話內容中傳送或接收的任何訊息內的 channelId 和 conversation 屬性將會包含這些識別碼。您也可以將這些值快取在安全的位置，這樣就可以在未來存取對話的資料，而不需要從訊息中擷取它們。

Response

回應將會包含具有新 eTag 值的 [BotData](#) 物件。

儲存私人對話資料

若要儲存特定對話內容中使用者的狀態資料，請發出此要求：

```
POST /v3/botstate/{channelId}/conversations/{conversationId}/users/{userId}
```

在此要求 URI 中，將 **{channelId}** 取代為頻道的識別碼，將 **{conversationId}** 取代為對話的識別碼，並將 **{userId}** 取代為該頻道上的使用者識別碼。您的 Bot 先前在對話內容中從使用者接收的任何訊息內的 **channelId**、**conversation** 和 **from** 屬性將會包含這些識別碼。您也可以將這些值快取在安全的位置，這樣就可以在未來存取對話的資料，而不需要從訊息中擷取它們。

將要求主體設定為 **BotData** 物件，如 [儲存使用者資料](#) 中所述。

Response

回應將會包含具有新 **eTag** 值的 **BotData** 物件。

取得私人對話資料

若要取得先前為特定對話內容中使用者儲存的狀態資料，請發出此要求：

```
GET /v3/botstate/{channelId}/conversations/{conversationId}/users/{userId}
```

在此要求 URI 中，將 **{channelId}** 取代為頻道的識別碼，將 **{conversationId}** 取代為對話的識別碼，並將 **{userId}** 取代為該頻道上的使用者識別碼。您的 Bot 先前在對話內容中從使用者接收的任何訊息內的 **channelId**、**conversation** 和 **from** 屬性將會包含這些識別碼。您也可以將這些值快取在安全的位置，這樣就可以在未來存取對話的資料，而不需要從訊息中擷取它們。

Response

回應將會包含具有新 **eTag** 值的 **BotData** 物件。

刪除使用者資料

若要刪除特定頻道上使用者的狀態資料，請發出此要求：

```
DELETE /v3/botstate/{channelId}/users/{userId}
```

在此要求 URI 中，將 **{channelId}** 取代為頻道的識別碼，並將 **{userId}** 取代為頻道上的使用者識別碼。您的 Bot 先前從使用者收到的任何訊息中的 **channelId** 和 **from** 屬性都會包含這些識別碼。您也可以將這些值快取在安全的位置，這樣就可以在未來存取使用者的資料，而不需要從訊息中擷取它們。

IMPORTANT

此作業會刪除先前使用 [儲存使用者資料](#) 作業或 [儲存私人對話資料](#) 作業來為使用者儲存的資料。它不會刪除先前使用 [儲存對話資料](#) 作業儲存的資料。因此，請「不要」使用 [儲存對話資料](#) 作業來儲存使用者的個人識別資訊 (PII)。

當您的 Bot 接收 `deleteUserData` 或 `contactRelationUpdate` 類型的 `Activity`，指出 Bot 已從使用者的連絡人清單中被移除時，Bot 應執行 [刪除使用者資料](#) 作業。

其他資源

- [重要概念](#)
- [活動概觀](#)

Direct Line API 3.0 中的重要概念

2019/2/28 • [Edit Online](#)

您可以使用 Direct Line API 啟用您的 Bot 和您自有用戶端應用程式的對話。本文介紹 Direct Line API 3.0 中的重要概念，並提供相關開發人員資源的資訊。

驗證

Direct Line API 3.0 要求可以使用您從 [Bot Framework 入口網站](#) 中的 Direct Line 頻道設定頁面取得的祕密來驗證，或使用您在執行階段取得的權杖來驗證。如需詳細資訊，請參閱[驗證](#)。

開始對話

直接線路對話是由用戶端明確地開啟，且只要 Bot 和用戶端在參與且具有有效認證，對話就會繼續執行。如需詳細資訊，請參閱[開始對話](#)。

傳送訊息

用戶端可藉由使用 Direct Line API 3.0，來透過發出 `HTTP POST` 要求將訊息傳送給您的 Bot。用戶端可能會每個要求都傳送單一訊息。如需詳細資訊，請參閱[將活動傳送到 Bot](#)。

接收訊息

用戶端可以使用 Direct Line API 3.0 透過 `WebSocket` 串流或發出 `HTTP GET` 要求從 Bot 接收訊息。用戶端可以使用這些技術從 `ActivitySet` 的 Bot 一次接收多則訊息。如需詳細資訊，請參閱[接收來自 Bot 的活動](#)。

開發人員資源

用戶端程式庫

Bot Framework 提供用戶端程式庫，可輔助透過 C# 和 Node.js 存取 Direct Line API 3.0。

- 若要在 Visual Studio 專案中使用 .NET 用戶端程式庫，請安裝 [Microsoft.Bot.Connector.DirectLine NuGet 套件](#)。
- 若要使用 Node.js 用戶端程式庫，請使用 [NPM](#) (或[下載來源](#)) 安裝 `botframework-directlinejs` 程式庫。

範例程式碼

[BotBuilder-範例 GitHub](#) 存放庫包含多個範例，說明如何搭配 C# 和 Node.js 使用 Direct Line API 3.0。

範例	語言	說明
Direct Line Bot 範例	C#	Bot 範例和自訂用戶端，會使用 Direct Line API 彼此通訊。
Direct Line Bot 範例 (使用用戶端 WebSocket)	C#	Bot 範例和自訂用戶端，會使用 Direct Line API 和 WebSocket 來彼此通訊。
Direct Line Bot 範例	JavaScript	Bot 範例和自訂用戶端，會使用 Direct Line API 彼此通訊。

範例	語言	說明
Direct Line Bot 範例 (使用用戶端 WebSocket)	JavaScript	Bot 範例和自訂用戶端，會使用 Direct Line API 和 WebSocket 來彼此通訊。

網路聊天控制項

Bot Framework 提供可讓您將使用直接線路的Bot 內嵌到您用戶端應用程式的控制項。如需詳細資訊，請參閱 [Microsoft Bot Framework WebChat 控制項](#)。

Authentication

2019/5/14 • [Edit Online](#)

用戶端可以藉由在 Bot Framework 入口網站使用您從 Direct Line 頻道設定頁面取得的祕密來驗證 Direct Line API 3.0 的要求，或使用您在執行階段取得的權杖來驗證。祕密或權杖應該使用下列格式，指定於每個要求的 `Authorization` 標頭中：

```
Authorization: Bearer SECRET_OR_TOKEN
```

祕密和權杖

Direct Line 祕密是一個主要金鑰，可用來存取屬於相關聯 Bot 的任何對話。祕密也可用來取得權杖。祕密不會過期。

Direct Line 權杖是一個金鑰，可用來存取單一對話。權杖會過期，但可以重新整理。

如果您要建立服務對服務的應用程式，則在 Direct Line API 要求的 `Authorization` 標頭中指定祕密可能是最簡單的方法。如果您要撰寫用戶端在網頁瀏覽器或行動裝置應用程式中執行的應用程式，您可以將交換祕密以取得權杖（這只適用於單一對話，而且除非重新整理，否則將會過期），並在 Direct Line API 要求的 `Authorization` 標頭中指定權杖。選擇最適合您的安全性模型。

NOTE

您的 Direct Line 用戶端認證與您的 Bot 認證不同。這可讓您獨立修訂金鑰，並讓您共用用戶端權杖，而不會洩漏 Bot 的密碼。

取得 Direct Line 祕密

您可以在 [Bot Framework 入口網站](#) 中，透過 Bot 的 Direct Line 頻道設定頁面來[取得 Direct Line 祕密](#)：

Configure Direct Line



+ Add new site

KB_Client_1 [Edit](#) Disable | [Delete](#)

Secret keys

[Hide](#) | [Regenerate](#)

XXXXXXXXXXXXXXXXXXXXXXXXXXXX
[Show](#) | [Regenerate](#)

Version

Select which versions of the Direct Line protocol are enabled on this site. More information about these versions can be found in the [Direct Line reference documentation](#).

產生 Direct Line 權杖

若要產生可用來存取單一對話的 Direct Line 權杖，請先從 [Bot Framework 入口網站](#) 的 Direct Line 頻道設定頁面取得 Direct Line 祕密。然後發出此要求，將您的 Direct Line 祕密交換以取得 Direct Line 權杖：

```
POST https://directline.botframework.com/v3/directline/tokens/generate
Authorization: Bearer SECRET
```

在此要求的 `Authorization` 標頭中，使用您的 Direct Line 祕密值來取代 **SECRET**。

下列程式碼片段提供產生權杖要求和回應的範例。

要求

```
POST https://directline.botframework.com/v3/directline/tokens/generate
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qbOF5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
```

要求承載 (其中包含權杖參數) 是選用，但建議使用的項目。在產生可送回 Direct Line 服務的權杖時，請提供下列承載，讓連線更加安全。藉由包含這些值，Direct Line 可對使用者識別碼和名稱執行額外的安全性驗證，以抑制惡意用戶端竄改這些值。包含這些值也會改善 Direct Line 傳送_交談更新_活動的能力，可讓其在使用者加入交談時立即產生交談更新。若未提供此資訊，使用者必須先傳送內容，Direct Line 才能傳送交談更新。

```
{
  "user": {
    "id": "string",
    "name": "string"
  },
  "trustedOrigins": [
    "string"
  ]
}
```

參數	類型	說明
<code>user.id</code>	字串	選用。權杖內要編碼的使用者通道專屬識別碼。若為 Direct Line 使用者，這必須以 <code>d1_</code> 開頭。您可以針對每次交談建立唯一的使用者識別碼，且基於安全考量，您應該讓此識別碼無法猜測。
<code>user.name</code>	字串	選用。權杖內要編碼的使用者易記顯示名稱。
<code>trustedOrigins</code>	字串陣列	選用。權杖內要內嵌的受信任網域清單。這些是可以裝載 Bot 網路聊天用戶端的網域。這應該符合 Direct Line 設定頁面中您 Bot 的清單。

Response

如果要求成功，回應就會包含對某個對話有效的 `token`，以及表示直到權杖過期前秒數的 `expires_in` 值。若要讓權杖仍然可用，您必須在它過期之前[重新整理權杖](#)。

```
HTTP/1.1 200 OK
[other headers]
```

```
{  
    "conversationId": "abc123",  
    "token":  
        "RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn",  
    "expires_in": 1800  
}
```

產生權杖與開始對話

產生權杖作業 ([POST /v3/directline/tokens/generate](#)) 類似 [開始對話](#) 作業 ([POST /v3/directline/conversations](#)), 這兩個作業都會傳回可用於存取單一對話的 `token`。不過，不同於開始對話作業，產生權杖作業不會開始對話、不會連絡 Bot，也不會建立串流處理的 WebSocket URL。

如果您計畫要將權杖散發給用戶端，並且想要讓它們起始對話，請使用產生權杖作業。如果您想要立即開始對話，請改用 [開始對話](#) 作業。

重新整理 Direct Line 權杖

Direct Line 權杖只要尚未過期，就可以重新整理且不限次數。過期的權杖無法重新整理。若要重新整理 Direct Line 權杖，請發出此要求：

```
POST https://directline.botframework.com/v3/directline/tokens/refresh  
Authorization: Bearer TOKEN_TO_BE_REFRESHED
```

在此要求的 `Authorization` 標頭中，使用您想要重新整理的 Direct Line 權杖來取代 `TOKEN_TO_BE_REFRESHED`。

下列程式碼片段提供重新整理權杖要求和回應的範例。

要求

```
POST https://directline.botframework.com/v3/directline/tokens/refresh  
Authorization: Bearer  
CurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn
```

Response

如果要求成功，回應就會包含對與前一個權杖相同之對話有效的新 `token`，以及表示直到新權杖過期前秒數的 `expires_in` 值。若要讓新權杖仍然可用，您必須在它過期之前 [重新整理權杖](#)。

```
HTTP/1.1 200 OK  
[other headers]
```

```
{  
    "conversationId": "abc123",  
    "token":  
        "RCurR_XV9ZA.cwA.BKA.y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xniaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0",  
    "expires_in": 1800  
}
```

其他資源

- [重要概念](#)
- [將 Bot 連線至 Direct Line](#)

開始對話

2019/2/28 • [Edit Online](#)

Direct Line 對話是由用戶端明確地開啟，且只要 Bot 和用戶端在參與且具有有效認證，對話就可以繼續執行。對話開啟時，Bot 和用戶端可以傳送訊息。可以有超過一個用戶端連線到指定對話，且每個用戶端都可以代表多個使用者參與對話。

開啟新的對話

若要與 Bot 開啟新的對話，請發出此要求：

```
POST https://directline.botframework.com/v3/directline/conversations
Authorization: Bearer SECRET_OR_TOKEN
```

下列程式碼片段提供「開始對話」要求和回應範例。

要求

```
POST https://directline.botframework.com/v3/directline/conversations
Authorization: Bearer
RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn
```

Response

如果要求成功，回應將會包含對話識別碼、權杖、一個表示直到權杖過期前秒數的值，以及用戶端可用來透過 [WebSocket 資料流接收活動](#) 的資料流 URL。

```
HTTP/1.1 201 Created
[other headers]
```

```
{
  "conversationId": "abc123",
  "token": "RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn",
  "expires_in": 1800,
  "streamUrl": "https://directline.botframework.com/v3/directline/conversations/abc123/stream?
t=RCurR_XV9ZA.cwA..."}
```

一般而言，開始對話要求會用來開啟新的對話，而且如果已順利開始新的對話，便會傳回 **HTTP 201** 狀態碼。不過，如果用戶端提交開始對話要求時，在 `Authorization` 標頭中使用了 Direct Line 權杖，該權杖先前已用來藉由「開始對話」作業開始對話，則會傳回 **HTTP 200** 狀態碼，表示這是可接受的要求，但未建立對話（因為已經存在）。

TIP

您有 60 秒的時間可以連線到 WebSocket 資料流 URL。如果無法在這段時間內建立連線，您可以[重新連線到對話](#)，以產生新的資料流 URL。

開始對話與產生權杖

開始對話作業 (`POST /v3/directline/conversations`) 和產生權杖作業 (`POST /v3/directline/tokens/generate`) 類似，兩個作業都會傳回可用於存取單一對話的 `token`。不過，開始對話作業也會開始對話、連絡 Bot，並建立 WebSocket 資料流 URL，而產生權杖作業不會做這些事。

如果您想要立即開始對話，請使用開始對話作業。如果您計畫要將權杖散發給用戶端，並且要它們起始對話，請改為使用產生權杖作業。

其他資源

- [重要概念](#)
- [驗證](#)
- [透過 WebSocket 資料流接收活動](#)
- [重新連線到對話](#)

重新連線到對話

2019/2/28 • [Edit Online](#)

如果用戶端使用 [WebSocket 介面](#)來接收訊息，但卻失去連線，此時可能需要重新連線。在此情況下，用戶端必須產生新的 WebSocket 串流 URL，用它來重新連線到對話。

產生新的 WebSocket 串流 URL

若要產生新的 WebSocket 串流 URL，以用來重新連線到現有對話，請發出此要求：

```
GET https://directline.botframework.com/v3/directline/conversations/{conversationId}?watermark={watermark_value}
Authorization: Bearer SECRET_OR_TOKEN
```

在此要求 URI 中，使用對話識別碼取代 **{conversationId}**，以及使用 watermark 值來取代 **{watermark_value}**（如果有提供 watermark 參數的話）。watermark 是選用參數。如果要求 URI 中有指定 watermark 參數，對話會從浮水印重新執行，以保證沒有任何訊息遺失。如果 watermark 參數從要求 URI 中省略，則只會收到重新連線要求執行之後的訊息。

下列程式碼片段提供重新連線要求和回應的範例。

要求

```
GET https://directline.botframework.com/v3/directline/conversations/abc123?watermark=0000a-42
Authorization: Bearer
RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn
```

Response

如果要求成功，回應會包含對話識別碼、語彙基元和新的 WebSocket 串流 URL。

```
HTTP/1.1 200 OK
[other headers]
```

```
{
  "conversationId": "abc123",
  "token": "RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn",
  "streamUrl": "https://directline.botframework.com/v3/directline/conversations/abc123/stream?watermark=000a-4&t=RCurR_XV9ZA.cwA..."
}
```

重新連線到對話

用戶端必須使用新的 WebSocket 串流 URL，以在 60 秒內 [重新連線到對話](#)。如果無法在這段時間內建立連線，用戶端必須發出另一個重新連線要求，以產生新的串流 URL。

如果您已在 Direct Line 設定中啟用「增強型驗證選項」，則可能會收到 400 "MissingProperty" 錯誤，表示未指定任何使用者識別碼。

其他資源

- 重要概念
- 驗證
- 透過 WebSocket 資料流接收活動

將活動傳送至 Bot

2019/2/28 • [Edit Online](#)

使用 Direct Line 3.0 通訊協定時，用戶端與 Bot 可交換許多不同類型的活動，包括訊息活動、輸入活動，以及 Bot 所支援的自訂活動。對於每個要求，用戶端可傳送一個活動。

傳送活動

若要將活動傳送至 Bot，用戶端必須建立活動物件以定義活動，然後在要求本文中指定活動物件，並對

`https://directline.botframework.com/v3/directline/conversations/{conversationId}/activities` 發出 POST 要求。

下列程式碼片段提供「傳送活動」要求和回應的範例。

要求

```
POST https://directline.botframework.com/v3/directline/conversations/abc123/activities
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
Content-Type: application/json
[other headers]
```

```
{
  "type": "message",
  "from": {
    "id": "user1"
  },
  "text": "hello"
}
```

Response

當活動傳遞至 Bot 時，服務會以反映 Bot 狀態碼的 HTTP 狀態碼來回應。如果 Bot 產生錯誤，系統會將 HTTP 502 回應（「錯誤的閘道」）傳回至用戶端，以回應其「傳送活動」要求。如果 POST 成功，則回應會包含 JSON 承載，指定已傳送至 Bot 的活動識別碼。

```
HTTP/1.1 200 OK
[other headers]
```

```
{
  "id": "0001"
}
```

傳送活動要求/回應的時間總計

將訊息發佈至 Direct Line 交談的時間總計，是以下幾項的總和：

- HTTP 要求從用戶端到 Direct Line 服務的傳輸時間
- Direct Line 中的內部處理時間（通常少於 120 毫秒）
- 從 Direct Line 服務到 Bot 的傳輸時間
- Bot 內的處理時間
- HTTP 回應回到用戶端的傳輸時間

將附件傳送至 Bot

在某些情況下，用戶端可能需要將影像或文件之類的附件傳送至 Bot。用戶端可藉由在它要使用

`POST /v3/directline/conversations/{conversationId}/activities` 傳送的活動物件內為附件指定 URL，或藉由使用

`POST /v3/directline/conversations/{conversationId}/upload` 來上傳附件，將附件傳送至 Bot。

依 URL 傳送附件

若要使用 `POST /v3/directline/conversations/{conversationId}/activities` 傳送隨附於活動物件的一或多個附件，只需將一或多個附件物件包含在活動物件內，並設定每個附件物件的 `contentUrl` 屬性以指定附件的 HTTP、HTTPS 或 `data` URI 即可。

藉由上傳來傳送附件

常常用戶端的裝置上有影像或文件要傳送至 Bot，但卻沒有對應至這些檔案的 URL。在此情況下，用戶端可以發出 `POST /v3/directline/conversations/{conversationId}/upload` 要求，以藉由上傳將附件傳送至 Bot。要求的格式和內容，將取決於用戶端是要傳送單一附件還是傳送多個附件。

藉由上傳來傳送單一附件

若要藉由上傳來傳送單一附件，請發出下列要求：

```
POST https://directline.botframework.com/v3/directline/conversations/{conversationId}/upload?userId={userId}
Authorization: Bearer SECRET_OR_TOKEN
Content-Type: TYPE_OF_ATTACHMENT
Content-Disposition: ATTACHMENT_INFO
[other headers]

[file content]
```

在此要求 URI 中，請將 `{conversationId}` 取代為交談的識別碼，並將 `{userId}` 取代為訊息傳送者的使用者識別碼。`userId` 是必要參數。在要求標頭中，請設定 `Content-Type` 以指定附件的類型，並設定 `Content-Disposition` 以指定附件的檔案名稱。

下列程式碼片段提供「傳送（單一）附件」要求和回應的範例。

要求

```
POST https://directline.botframework.com/v3/directline/conversations/abc123/upload?userId=user1
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
Content-Type: image/jpeg
Content-Disposition: name="file"; filename="badjokeeel.jpg"
[other headers]

[JPEG content]
```

Response

如果要求成功，將會在上傳完成後將訊息活動傳送至 Bot，且用戶端收到的回應將會包含已傳送的活動識別碼。

```
HTTP/1.1 200 OK
[other headers]
```

```
{
  "id": "0003"
}
```

藉由上傳來傳送多個附件

若要藉由上傳來傳送多個附件，請將有多個部分的要求 `POST` 至

`/v3/directline/conversations/{conversationId}/upload` 端點。請將要求的 `Content-Type` 標頭設定為 `multipart/form-data`，並且為每個部分納入 `Content-Type` 標頭和 `Content-Disposition` 標頭，以指定每個附件的類型和檔案名稱。在要求 URI 中，請將 `userId` 參數設定為訊息傳送者的使用者識別碼。

您可以藉由新增一個指定 `Content-Type` 標頭值 `application/vnd.microsoft.activity` 的部分，在要求內納入活動物件。如果要求中包含活動，則承載的其他部分所指定的附件會先新增為該活動的附件，然後才會傳送。如果要求未包含活動，則會建立空的活動，作為指定的附件藉以傳送的容器。

下列程式碼片段提供「傳送 (多個) 附件」要求和回應的範例。在此範例中，要求會傳送包含一些文字和單一影像附件的訊息。在要求中可以新增其他部分，以將多個附件包含在此訊息中。

要求

```
POST https://directline.botframework.com/v3/directline/conversations/abc123/upload?userId=user1
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
Content-Type: multipart/form-data; boundary=----DD4E5147-E865-4652-B662-F223701A8A89
[other headers]

----DD4E5147-E865-4652-B662-F223701A8A89
Content-Type: image/jpeg
Content-Disposition: form-data; name="file"; filename="badjokeeel.jpg"
[other headers]

[JPEG content]

----DD4E5147-E865-4652-B662-F223701A8A89
Content-Type: application/vnd.microsoft.activity
[other headers]

{
  "type": "message",
  "from": {
    "id": "user1"
  },
  "text": "Hey I just IM'd you\n\nand this is crazy\n\nbut here's my webhook\n\nso POST me maybe"
}

----DD4E5147-E865-4652-B662-F223701A8A89
```

Response

如果要求成功，將會在上傳完成後將訊息活動傳送至 Bot，且用戶端收到的回應將會包含已傳送的活動識別碼。

```
HTTP/1.1 200 OK
[other headers]
```

```
{
  "id": "0004"
}
```

其他資源

- [重要概念](#)
- [驗證](#)
- [開始對話](#)
- [重新連線至交談](#)

- 從 Bot 接收活動
- 結束交談

接收來自 Bot 的活動

2019/4/18 • [Edit Online](#)

透過使用直接線路 3.0 通訊協定，用戶端可以透過 `WebSocket` 串流來接收活動，或透過發出 `HTTP GET` 要求來擷取活動。

WebSocket 與 HTTP GET

串流 WebSocket 會有效率地將訊息推送到用戶端，而 GET 介面可讓用戶端明確地要求訊息。雖然 WebSocket 機制因其效率而較常被使用，但 GET 機制對於無法使用 WebSockets 的用戶端而言非常實用。

並非所有活動類型都可透過 WebSocket 和 HTTP GET 這兩者使用。下表說明對於使用直接線路通訊協定的用戶端之各種活動類型的可用性。

活動類型	可用性
Message	HTTP GET 和 WebSocket
typing	只有 WebSocket
conversationUpdate	不透過用戶端傳送/接收
contactRelationUpdate	直接線路中不支援
endOfConversation	HTTP GET 和 WebSocket
所有其他活動類型	HTTP GET 和 WebSocket

透過 WebSocket 串流接收活動

當用戶端傳送 [開始對話](#) 要求以開啟和 Bot 的對話時，服務的回應包含用戶端可以在後續用於透過 WebSocket 連線的 `streamUrl` 屬性。串流 URL 是預先授權的，因此用戶端透過 WebSocket 的連線要求「不」需要 `Authorization` 標頭。

下列範例顯示使用 `streamUrl` 透過 WebSocket 來連線的要求。

```
-- connect to wss://directline.botframework.com --
GET /v3/directline/conversations/abc123/stream?t=RCurR_XV9ZA.cwA...
Upgrade: websocket
Connection: upgrade
[other headers]
```

服務回應狀態碼 HTTP 101 (「正在切換通訊協定」)。

```
HTTP/1.1 101 Switching Protocols
[other headers]
```

接收訊息

透過 WebSocket 連線之後，用戶端可能會收到來自直接線路服務的這些訊息類型：

- 包含 [ActivitySet](#) 的訊息，其中包括一或多個活動和浮水印 (如下所述)。
- 空白訊息，直接線路服務用來確認連線仍然有效。
- 其他類型 (在稍後定義)。這些類型由 JSON 根中的屬性識別。

[ActivitySet](#) 包含由 Bot 及對話中所有使用者所傳送的訊息。下列範例顯示包含單一訊息的 [ActivitySet](#)。

```
{
  "activities": [
    {
      "type": "message",
      "channelId": "directline",
      "conversation": {
        "id": "abc123"
      },
      "id": "abc123|0000",
      "from": {
        "id": "user1"
      },
      "text": "hello"
    }
  ],
  "watermark": "0000a-42"
}
```

處理訊息

用戶端應該持續追蹤收到的每個 [ActivitySet](#) 中的 [watermark](#) 值，這樣就能使用浮水印來保證在連線中斷時不會有任何訊息遺失而需要[重新連線到對話](#)。如果用戶端接收到的 [ActivitySet](#) 中的 [watermark](#) 屬性為 [null](#) 或已遺失，則它應該忽略該值且不覆寫之前收到的浮水印。

用戶端應該忽略從直接線路服務收到的空白訊息。

用戶端可能會傳送空白訊息至直接線路服務以驗證連線能力。直接線路服務將會忽略從用戶端收到的空白訊息。

直接線路服務在某些情況下可能會強制關閉 WebSocket 連線。如果用戶端尚未收到 [endOfConversation](#) 活動，它可能會[產生新的 WebSocket 串流 URL](#)，以用於重新連線到對話。

WebSocket 串流包含即時更新和最近的訊息 (因為已發出透過 WebSocket 連線的呼叫)，但它不包含在最近的 [POST](#) 之前傳送至 [/v3/directline/conversations/{id}](#) 的訊息。若要擷取對話中稍早之前的訊息，請使用下述的 [HTTP GET](#)。

使用 HTTP GET 擷取活動

無法使用 WebSockets 的用戶端可以使用 [HTTP GET](#) 擷取活動。

若要擷取特定交談的訊息，請向 [/v3/directline/conversations/{conversationId}/activities](#) 端點發出 [GET](#) 要求，選擇性指定 [watermark](#) 參數以指出用戶端看到的最新訊息。

下列程式碼片段提供「取得對話活動」的要求和回應範例。「取得對話活動」回應包含 [watermark](#) 做為 [ActivitySet](#) 的屬性。用戶端應隨 [watermark](#) 值向前逐頁查詢可用的活動，直到沒有活動傳回。

要求

```
GET https://directline.botframework.com/v3/directline/conversations/abc123/activities?watermark=0001a-94
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
```

Response

```
HTTP/1.1 200 OK  
[other headers]
```

```
{  
    "activities": [  
        {  
            "type": "message",  
            "channelId": "directline",  
            "conversation": {  
                "id": "abc123"  
            },  
            "id": "abc123|0000",  
            "from": {  
                "id": "user1"  
            },  
            "text": "hello"  
        },  
        {  
            "type": "message",  
            "channelId": "directline",  
            "conversation": {  
                "id": "abc123"  
            },  
            "id": "abc123|0001",  
            "from": {  
                "id": "bot1"  
            },  
            "text": "Nice to see you, user1!"  
        }  
    "watermark": "0001a-95"  
}
```

計時考量

大部分的用戶端都希望建立完整的訊息記錄。雖然直接線路是有潛在計時間隔的多方通訊協定，但通訊協定和服務的設計旨在讓您更容易建置可靠的用戶端。

- 在 WebSocket 串流和「取得對話活動」回應中傳送的 `watermark` 屬性很可靠。只要用戶端逐字重新執行浮水印，就不會錯過任何訊息。
- 當用戶端啟動對話並連線到 WebSocket 串流時，在 POST 之後且在通訊端開啟之前傳送的任何活動，都會在新的活動前重新執行。
- 當用戶端已連線到 WebSocket 串流並發出「取得對話活動」要求（以重新整理記錄）時，活動可能會在兩個頻道之間重複。用戶端應持續追蹤所有已知的活動識別碼，這樣它們就能拒絕重複的活動（如果發生重複）。

使用 `HTTP GET` 輪詢的用戶端應選擇符合其使用目的的輪詢間隔。

- 服務對服務應用程式通常會使用 5 秒或 10 秒的輪詢間隔。
- 面向用戶端的應用程式通常使用 1 秒的輪詢間隔，並在用戶端傳送的每則訊息後，迅速發出單一其他要求（以快速擷取 Bot 的回應）。此延遲最短 300 毫秒，但應根據 Bot 的速度和傳輸時間調整。輪詢在任何長期中都不應頻繁於每秒一次。

其他資源

- [重要概念](#)
- [驗證](#)

- 開始對話
- 重新連線到對話
- 將活動傳送到 Bot
- 結束對話

結束對話

2019/2/28 • [Edit Online](#)

endOfConversation 活動表示通道或 Bot 已結束交談。

NOTE

雖然只有非常少數的通道會傳送 **endOfConversation** 事件，但 Cortana 是唯一接受該事件的通道。其他通道（包括 Direct Line）不會實作這項功能，而會卸除或推進活動；每個通道都可決定如何回應 endOfConversation 活動。如果您要設計 DirectLine 用戶端，您會更新用戶端以做出正確行為，例如，如果 Bot 傳送活動給已經結束的對話，則會產生錯誤。

傳送 endOfConversation 活動

endOfConversation 活動會結束 Bot 和用戶端之間的通訊。在傳送 **endOfConversation** 活動之後，用戶端仍然可以使用 [HTTP GET](#) 來擷取訊息，但用戶端或 Bot 都不能將任何其他訊息傳送至對話。

若要結束對話，只需發出 POST 要求以傳送 **endOfConversation** 活動即可。

要求

```
POST https://directline.botframework.com/v3/directline/conversations/abc123/activities
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
[other headers]
```

```
{
  "type": "endOfConversation",
  "from": {
    "id": "user1"
  }
}
```

Response

如果要求成功，則回應將包含已傳送之活動的識別碼。

```
HTTP/1.1 200 OK
[other headers]
```

```
{
  "id": "0004"
}
```

其他資源

- [重要概念](#)
- [驗證](#)
- [將活動傳送到 Bot](#)

API 參考 - Direct Line API 3.0

2019/4/29 • [Edit Online](#)

使用 Direct Line API 3.0，您就能讓用戶端應用程式與 Bot 進行通訊。Direct Line API 3.0 會透過 HTTPS 使用業界標準的 REST 和 JSON。

基底 URI

若要存取 Direct Line API 3.0，請將此基底 URI 用於所有 API 要求：

`https://directline.botframework.com`

headers

除了標準的 HTTP 要求標頭之外，直接線路 API 要求必須包含指定密碼或權杖的 `Authorization` 標頭，藉此驗證發出要求的用戶端。請使用此格式指定 `Authorization` 標頭：

```
Authorization: Bearer SECRET_OR_TOKEN
```

如需如何取得您的用戶端用來驗證其 Direct Line API 要求之祕密或權杖的詳細資料，請參閱[驗證](#)。

HTTP 狀態碼

與每個回應一併傳回的 [HTTP 狀態碼](#) 會指出對應要求的結果。

HTTP 狀態碼	意義
200	要求成功。
201	要求成功。
202	已接受要求進行處理。
204	要求成功，但未傳回任何內容。
400	要求的格式不正確或其他錯誤。
401	用戶端未獲授權，無法提出要求。此狀態碼出現的原因通常是遺漏 <code>Authorization</code> 標頭或格式不正確。
403	不允許用戶端執行要求的作業。如果要求指定了先前有效但已過期的權杖，在 <code>ErrorResponse</code> 物件內傳回之錯誤的 <code>code</code> 屬性會設定為 <code>TokenExpired</code> 。
404	找不到要求的資源。此狀態碼通常表示要求的 URI 無效。
500	Direct Line 服務內發生內部伺服器錯誤。
502	Bot 無法使用，或傳回錯誤。此為常見錯誤碼。

NOTE

HTTP 狀態碼 101 用於 WebSocket 連線路徑中，雖然這可能由您的 WebSocket 用戶端處理。

Errors

若回應顯示的 HTTP 狀態碼位於 4xx 或 5xx 範圍內，則會在提供錯誤相關資訊的回應本文中加入 **ErrorResponse** 物件。如果您收到 4xx 範圍的錯誤訊息，請檢查 **ErrorResponse** 物件以找出錯誤原因，並在重新提交要求之前先解決問題。

NOTE

ErrorResponse 物件內 `code` 屬性指定的 HTTP 狀態碼和值很穩定。**ErrorResponse** 物件內 `message` 屬性指定的值可能會隨著時間改變。

下列程式碼片段顯示範例要求和產生的錯誤回應。

要求

```
POST https://directline.botframework.com/v3/directline/conversations/abc123/activities  
[detail omitted]
```

Response

```
HTTP/1.1 502 Bad Gateway  
[other headers]
```

```
{  
  "error": {  
    "code": "BotRejectedActivity",  
    "message": "Failed to send activity: bot returned an error"  
  }  
}
```

權杖作業

透過這些作業可建立或重新整理用戶端用於存取單一對話的權杖。

作業	說明
產生權杖	產生新對話的權杖。
重新整理權杖	重新整理權杖。

產生權杖

產生適用於一個對話的權杖。

```
POST /v3/directline/tokens/generate
```

要求本文	n/a
------	-----

傳回	Conversation 物件
----	-----------------

重新整理權杖

重新整理此權杖。

```
POST /v3/directline/tokens/refresh
```

要求本文	n/a
傳回	Conversation 物件

對話作業

使用這些作業可開啟您與 Bot 的對話，並在用戶端與 Bot 之間交換訊息。

作業	說明
開始對話	開啟與 Bot 的新對話。
取得對話資訊	取得現有對話的相關資訊。此作業會產生新的 WebSocket 資料流 URL，用戶端可以用來 重新連線 對話。
取得活動	從 Bot 摄取活動。
傳送活動	將活動傳送到 Bot。
上傳與傳送檔案	上傳檔案後，以附件型式傳送檔案。

開始對話

開啟與 Bot 的新對話。

```
POST /v3/directline/conversations
```

要求本文	n/a
傳回	Conversation 物件

取得對話資訊

取得現有對話的相關資訊，並且產生新的 WebSocket 資料流 URL，用戶端可以用來[重新連線](#)對話。您可以選擇性地在要求 URI 中提供 `watermark` 參數，以指出最近期內用戶端所見到的訊息。

```
GET /v3/directline/conversations/{conversationId}?watermark={watermark_value}
```

要求本文	n/a
傳回	Conversation 物件

取得活動

針對指定對話從 Bot 撈取活動。您可以選擇性地在要求 URI 中提供 `watermark` 參數，以指出最近期內用戶端所見到的訊息。

```
GET /v3/directline/conversations/{conversationId}/activities?watermark={watermark_value}
```

要求本文	n/a
傳回	ActivitySet 物件。回應中包含 <code>watermark</code> 作為 ActivitySet 物件的屬性。用戶端應隨 <code>watermark</code> 值向前逐頁查詢可用的活動，直到沒有活動傳回。

傳送活動

將活動傳送到 Bot。

```
POST /v3/directline/conversations/{conversationId}/activities
```

要求本文	Activity 物件
傳回	ResourceResponse ，其中包含 <code>id</code> 屬性，指定已傳送給 Bot 的活動識別碼。

上傳與傳送檔案

上傳檔案後，以附件型式傳送檔案。設定要求 URI 中的 `userId` 參數以指定正在傳送附件之使用者的識別碼。

```
POST /v3/directline/conversations/{conversationId}/upload?userId={userId}
```

要求本文	若為單一附件，請在要求本文填入檔案內容。若有多個附件，請建立多部分要求本文，包含一個部分是針對每個附件，以及（選擇性）一個部分是針對應該作為指定附件之容器的 Activity 物件。如需詳細資訊，請參閱 將活動傳送到 Bot 。
傳回	ResourceResponse ，其中包含 <code>id</code> 屬性，指定已傳送給 Bot 的活動識別碼。

NOTE

上傳的檔案會在 24 小時後刪除。

結構描述

Direct Line 3.0 結構描述包含 [Bot Framework v3 結構描述](#) 所定義的所有物件，以及 [ActivitySet 物件](#) 和 [Conversation 物件](#)。

ActivitySet 物件

定義一組活動。

屬性	類型	說明
活動	Activity[]	Activity 物件 的陣列。
watermark	字串	一組訊息中，活動的最大浮水印。用戶端可以使用 watermark 值，指出它在 從 Bot 撷取活動 時，或在 產生新的 WebSocket 資料流 URL 時，已看到的最新訊息。

Conversation 物件

定義 Direct Line 對話。

屬性	類型	說明
conversationId	字串	唯一識別指定權杖所適用對話的識別碼。
expires_in	number	權杖到期之前的秒數。
streamUrl	字串	對話的訊息資料流 URL。
token	字串	適用於指定對話的權杖。

活動

針對用戶端透過 Direct Line 從 Bot 收到的每個 [Activity](#)：

- 會保留附件卡片。
- 已上傳附件的 URL 會以私密連結隱藏。
- [channelData](#) 屬性已保留而不修改。

用戶端可能從 Bot [收到](#)多個活動，作為 [ActivitySet](#) 的一部分。

當用戶端透過 Direct Line 傳送 [Activity](#) 紙 Bot：

- [type](#) 屬性會指定它傳送之活動的類型（通常是訊息）。
- [from](#) 屬性必須填入使用者識別碼，由用戶端選擇。
- 附件可能包含現有資源的 URL，或透過 Direct Line 附件端點上傳的 URL。
- [channelData](#) 屬性已保留而不修改。
- 若序列化為 JSON 並且加密，則活動的總大小不得超過 256000 個字元。因此，建議將活動保持在 15 萬以下。如果需要更多資料，請考慮將活動分成多個活動及/或考慮使用附件。

對於每個要求，用戶端可[傳送](#)一個活動。

直接線路 API 1.1 中的重要概念

2019/2/28 • [Edit Online](#)

您可以使用直接線路 API 啟用您的 Bot 和您自有用戶端應用程式的對話。

IMPORTANT

本文介紹直接線路 API 1.1 中的重要概念，並提供相關開發人員資源的資訊。如果要在用戶端應用程式與 Bot 之間建立新連線，請改為使用[直接線路 API 3.0](#)。

驗證

直接線路 API 1.1 要求可以使用您從[Bot Framework 入口網站](#)（英文）中的直接線路頻道設定頁面取得的祕密來驗證，或使用您在執行階段取得的權杖來驗證。如需詳細資訊，請參閱[驗證](#)。

開始對話

直接線路對話是由用戶端明確地開啟，且只要 Bot 和用戶端在參與且具有有效認證，對話就會繼續執行。如需詳細資訊，請參閱[開始對話](#)。

傳送訊息

使用直接線路 API 1.1，用戶端就能透過發出 [HTTP POST](#) 要求將訊息傳送給您的 Bot。用戶端可能會為每個要求都傳送單一訊息。如需詳細資訊，請參閱[將訊息傳送至 Bot](#)。

接收訊息

使用直接線路 API 1.1，用戶端就能透過使用 [HTTP GET](#) 要求來輪詢以接收訊息。用戶端對於每個要求的回應，可能會從 Bot 收到屬於 [MessageSet](#) 一部分的多個訊息。如需詳細資訊，請參閱[接收來自 Bot 的訊息](#)。

開發人員資源

用戶端程式庫

Bot Framework 提供用戶端程式庫，有助於透過 C# 存取直接線路 API 1.1。若要在 Visual Studio 專案中使用用戶端程式庫，請安裝 [Microsoft.Bot.Connector.DirectLine v1.x NuGet 套件](#)（英文）。

除了使用 C# 用戶端程式庫之外，您也可以使用[直接線路 API 1.1 Swagger 檔案](#)以所選的語言產生自己的用戶端程式庫。

Web 聊天控制項

Bot Framework 提供可讓您將使用直接線路的 Bot 內嵌到您用戶端應用程式的控制項。如需詳細資訊，請參閱[Microsoft Bot Framework WebChat 控制項](#)。

驗證

2019/2/28 • [Edit Online](#)

IMPORTANT

本文描述 Direct Line API 1.1 中的驗證。如果要在用戶端應用程式與 Bot 之間建立新連線，請改為使用 [Direct Line API 3.0](#)。

用戶端可以藉由在 Bot Framework 入口網站使用您從 [Direct Line 頻道設定頁面](#)取得的祕密來驗證 Direct Line API 1.1 的要求，或使用您在執行階段取得的權杖來驗證。

祕密或權杖應該指定在每個要求的 `Authorization` 標頭，使用持有人配置或 "BotConnector" 配置。

持有人配置：

```
Authorization: Bearer SECRET_OR_TOKEN
```

BotConnector 配置：

```
Authorization: BotConnector SECRET_OR_TOKEN
```

祕密和權杖

Direct Line 祕密是一個主要金鑰，可用來存取屬於相關聯 Bot 的任何對話。祕密也可用來取得權杖。祕密不會過期。

Direct Line 權杖是一個金鑰，可用來存取單一對話。權杖會過期，但可以重新整理。

如果您要建立服務對服務的應用程式，則在 Direct Line API 要求的 `Authorization` 標頭中指定祕密可能是最簡單的方法。如果您要撰寫用戶端在網頁瀏覽器或行動裝置應用程式中執行的應用程式，您可以將交換祕密以取得權杖（這只適用於單一對話，而且除非重新整理，否則將會過期），並在 Direct Line API 要求的 `Authorization` 標頭中指定權杖。選擇最適合您的安全性模型。

NOTE

您的 Direct Line 用戶端認證與您的 Bot 認證不同。這可讓您獨立修訂金鑰，並讓您共用用戶端權杖，而不會洩漏 Bot 的密碼。

取得 Direct Line 祕密

您可以在 [Bot Framework 入口網站](#)中，透過 Bot 的 Direct Line 頻道設定頁面來[取得 Direct Line 祕密](#)：

Configure Direct Line



+ Add new site KB_Client_1 Disable |

KB_Client_1

Secret keys

Hide | Regenerate

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 Show | Regenerate

Version
Select which versions of the Direct Line protocol are enabled on this site. More information about these versions can be found in the [Direct Line reference documentation](#).

產生 Direct Line 權杖

若要產生可用來存取單一對話的 Direct Line 權杖，請先從 [Bot Framework 入口網站](#) 的 Direct Line 頻道設定頁面取得 Direct Line 祕密。然後發出此要求，將您的 Direct Line 祕密交換以取得 Direct Line 權杖：

```
POST https://directline.botframework.com/api/tokens/conversation  
Authorization: Bearer SECRET
```

在此要求的 `Authorization` 標頭中，使用您的 Direct Line 祕密值來取代 **SECRET**。

下列程式碼片段提供產生權杖要求和回應的範例。

要求

```
POST https://directline.botframework.com/api/tokens/conversation  
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
```

Response

如果要求成功，回應會包含適用於一個對話的權杖。權杖將於 30 分鐘過期。若要讓權杖仍然可用，您必須在它過期之前[重新整理權杖](#)。

```
HTTP/1.1 200 OK  
[other headers]  
"RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn"
```

產生權杖與開始對話

產生權杖作業 (`POST /api/tokens/conversation`) 和[開始對話](#)作業 (`POST /api/conversations`) 類似，兩個作業都會傳回可用於存取單一對話的 `token`。不過，不同於「開始對話」作業，產生權杖作業不會開始對話或連絡 Bot。

如果您計畫要將權杖散發給用戶端，並且要它們起始對話，請使用「產生權杖」作業。如果您想要立即開始對話，請改用[開始對話](#)作業。

重新整理 Direct Line 權杖

Direct Line 權杖的有效期為從產生起算 30 分鐘的時間，而且可以重新整理，時間量不限，只要未過期即可。無法重新整理過期的權杖。若要重新整理 Direct Line 權杖，請發出此要求：

```
POST https://directline.botframework.com/api/tokens/{conversationId}/renew
Authorization: Bearer TOKEN_TO_BE_REFRESHED
```

在此要求的 URI，將 **{conversationId}** 取代為權杖有效的對話識別碼，並在此要求的 **Authorization** 標頭中，將 **TOKEN_TO_BE_REFRESHED** 取代為您要重新整理的 Direct Line 權杖。

下列程式碼片段提供重新整理權杖要求和回應的範例。

要求

```
POST https://directline.botframework.com/api/tokens/abc123/renew
Authorization: Bearer
CurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn
```

Response

如果要求成功，回應會包含適用於與先前權杖相同之對話的新權杖。新權杖將於 30 分鐘過期。若要讓新權杖仍然可用，您必須在它過期之前[重新整理權杖](#)。

```
HTTP/1.1 200 OK
[other headers]

"RCurR_XV9ZA.cwA.BKA.y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xniaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0"
```

其他資源

- [重要概念](#)
- [將 Bot 連線至 Direct Line](#)

開始對話

2019/2/28 • [Edit Online](#)

IMPORTANT

此文章說明如何使用直接線路 API 1.1 開始對話。如果要在用戶端應用程式與 Bot 之間建立新連線，請改為使用[直接線路 API 3.0](#)。

直接線路對話是由用戶端明確地開啟，且只要 Bot 和用戶端在參與且具有有效認證，對話就會繼續執行。對話開啟時，Bot 和用戶端可以傳送訊息。可以有超過一個用戶端連線到指定對話，且每個用戶端都可以代表多個使用者參與對話。

開啟新的對話

若要與 Bot 開啟新的對話，請發出此要求：

```
POST https://directline.botframework.com/api/conversations  
Authorization: Bearer SECRET_OR_TOKEN
```

下列程式碼片段提供「開始對話」要求和回應範例。

要求

```
POST https://directline.botframework.com/api/conversations  
Authorization: Bearer  
RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn
```

Response

如果要求成功，回應將會包含對話識別碼、權杖，以及表示權杖過期前秒數的值。

```
HTTP/1.1 200 OK  
[other headers]
```

```
{  
  "conversationId": "abc123",  
  "token":  
    "RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn",  
  "expires_in": 1800  
}
```

開始對話與產生權杖

「開始對話」作業 (`POST /api/conversations`) 和[取得權杖](#)作業 (`POST /api/tokens/conversation`) 類似，兩個作業都會傳回可用於存取單一對話的 `token`。不過，開始對話作業也會開始對話並連絡 Bot，而產生權杖作業不會做這些事。

如果您想要立即開始對話，請使用「開始對話」作業。如果您計畫要將權杖散發給用戶端，並且要它們起始對話，請改為使用[產生權杖](#)作業。

其他資源

- 重要概念
- 驗證

將訊息傳送至 Bot

2019/2/28 • [Edit Online](#)

IMPORTANT

本文說明如何使用 Direct Line API 1.1，將訊息傳送至 Bot。如果要在用戶端應用程式與 Bot 之間建立新連線，請改為使用 [Direct Line API 3.0](#)。

使用 Direct Line 1.1 通訊協定，用戶端可以與 Bot 交換訊息。這些訊息會轉換成 Bot 支援 (Bot Framework v1 或 Bot Framework v3) 的結構描述。用戶端可能會為每個要求都傳送單一訊息。

傳送訊息

若要將訊息傳送至 Bot，用戶端必須建立 [訊息](#) 物件以定義訊息，然後在要求本文中指定訊息物件，並對

`https://directline.botframework.com/api/conversations/{conversationId}/messages` 發出 `POST` 要求。

下列程式碼片段提供「傳送訊息」要求和回應的範例。

要求

```
POST https://directline.botframework.com/api/conversations/abc123/messages
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrc8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
[other headers]
```

```
{
  "text": "hello",
  "from": "user1"
}
```

Response

當訊息傳遞至 Bot 時，服務會以可反映 Bot 狀態碼的 HTTP 狀態碼來回應。如果 Bot 產生錯誤，系統會將 HTTP 500 回應（「內部伺服器錯誤」）傳回至用戶端，以回應其「傳送訊息」要求。如果 POST 成功，服務就會傳回 HTTP 204 狀態碼。回應的本文中沒有任何傳回的資料。用戶端的訊息和來自 Bot 的任何訊息都，可以透過 [輪詢](#) 取得。

```
HTTP/1.1 204 No Content
[other headers]
```

傳送訊息要求/回應的時間總計

將訊息 POST 至 Direct Line 對話的時間總計，是以下幾項的總和：

- HTTP 要求從用戶端到 Direct Line 服務的傳輸時間
- Direct Line 中的內部處理時間（通常少於 120 毫秒）
- 從 Direct Line 服務到 Bot 的傳輸時間
- Bot 內的處理時間
- HTTP 回應回到用戶端的傳輸時間

將附件傳送至 Bot

在某些情況下，用戶端可能需要將影像或文件之類的附件傳送至 Bot。用戶端可藉由在它要使用

`POST /api/conversations/{conversationId}/messages` 傳送的訊息物件內為附件指定 URL，或藉由使用

`POST /api/conversations/{conversationId}/upload` 來上傳附件，將附件傳送至 Bot。

依 URL 傳送附件

若要使用 `POST /api/conversations/{conversationId}/messages` 傳送一或多個附件作為訊息物件的一部分，請在訊息的 `images` 陣列和/或 `attachments` 陣列內指定附件 URL。

藉由上傳來傳送附件

常常用戶端的裝置上有影像或文件要傳送至 Bot，但卻沒有對應至這些檔案的 URL。在此情況下，用戶端可以發出 `POST /api/conversations/{conversationId}/upload` 要求，以藉由上傳將附件傳送至 Bot。要求的格式和內容，將取決於用戶端是要傳送單一附件還是傳送多個附件。

藉由上傳來傳送單一附件

若要藉由上傳來傳送單一附件，請發出下列要求：

```
POST https://directline.botframework.com/api/conversations/{conversationId}/upload?userId={userId}
Authorization: Bearer SECRET_OR_TOKEN
Content-Type: TYPE_OF_ATTACHMENT
Content-Disposition: ATTACHMENT_INFO
[other headers]

[file content]
```

在此要求 URI 中，請將 `{conversationId}` 取代為對話的識別碼，並將 `{userId}` 取代為訊息傳送者的使用者識別碼。在要求標頭中，請設定 `Content-Type` 以指定附件的類型，並設定 `Content-Disposition` 以指定附件的檔案名稱。

下列程式碼片段提供「傳送(單一)附件」要求和回應的範例。

要求

```
POST https://directline.botframework.com/api/conversations/abc123/upload?userId=user1
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qbOF5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
Content-Type: image/jpeg
Content-Disposition: name="file"; filename="badjokeee1.jpg"
[other headers]

[JPEG content]
```

Response

如果要求成功，會在上傳完成時傳送訊息至 Bot，且服務會傳回 HTTP 204 狀態碼。

```
HTTP/1.1 204 No Content
[other headers]
```

藉由上傳來傳送多個附件

若要藉由上傳來傳送多個附件，請將有多個部分的要求 `POST` 至 `/api/conversations/{conversationId}/upload` 端點。請將要求的 `Content-Type` 標頭設定為 `multipart/form-data`，並且為每個部分納入 `Content-Type` 標頭和 `Content-Disposition` 標頭，以指定每個附件的類型和檔案名稱。在要求 URI 中，請將 `userId` 參數設定為訊息傳送者的使用者識別碼。

您可以藉由新增一個會指定 `Content-Type` 標頭值 `application/vnd.microsoft.bot.message` 的部分，在要求內納入訊息物件。這可讓用戶端自訂包含附件的訊息。如果要求中包含訊息，則承載的其他部分所指定的附件會先新增為該訊息的附件，然後才會傳送。

下列程式碼片段提供「傳送 (多個) 附件」要求和回應的範例。在此範例中，要求會傳送包含一些文字和單一影像附件的訊息。在要求中可以新增其他部分，以將多個附件包含在此訊息中。

要求

```
POST https://directline.botframework.com/api/conversations/abc123/upload?userId=user1
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
Content-Type: multipart/form-data; boundary=----DD4E5147-E865-4652-B662-F223701A8A89
[other headers]

----DD4E5147-E865-4652-B662-F223701A8A89
Content-Type: image/jpeg
Content-Disposition: form-data; name="file"; filename="badjokeeel.jpg"
[other headers]

[JPEG content]

----DD4E5147-E865-4652-B662-F223701A8A89
Content-Type: application/vnd.microsoft.bot.message
[other headers]

{
  "text": "Hey I just IM'd you\n\nand this is crazy\n\nbut here's my webhook\n\nso POST me maybe",
  "from": "user1"
}

----DD4E5147-E865-4652-B662-F223701A8A89
```

Response

如果要求成功，會在上傳完成時傳送訊息至 Bot，且服務會傳回 HTTP 204 狀態碼。

```
HTTP/1.1 204 No Content
[other headers]
```

其他資源

- [重要概念](#)
- [驗證](#)
- [開始對話](#)
- [從 Bot 接收訊息](#)

從 Bot 接收訊息

2019/2/28 • [Edit Online](#)

IMPORTANT

本文描述如何從使用 Direct Line API 1.1 的 Bot 接收訊息。如果要在用戶端應用程式與 Bot 之間建立新連線，請改用 [Direct Line API 3.0](#)。

使用 Direct Line 1.1 通訊協定，用戶端必須輪詢 `HTTP GET` 介面以接收訊息。

使用 HTTP GET 接收訊息

若要擷取特定交談的訊息，請向 `api/conversations/{conversationId}/messages` 端點發出 `GET` 要求，選擇性指定 `watermark` 參數以指出用戶端看到的最新訊息。JSON 回應中會傳回已更新的 `watermark` 值，即使其未包含任何訊息。

下列程式碼片段提供 Get Messages 要求和回應的範例。Get Messages 回應包含作為 `MessageSet` 屬性的 `watermark`。用戶端應持續送出 `watermark` 值來瀏覽整頁的可用訊息，直至不再收到任何訊息。

要求

```
GET https://directline.botframework.com/api/conversations/abc123/messages?watermark=0001a-94
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
```

Response

```
HTTP/1.1 200 OK
[other headers]
```

```
{
  "messages": [
    {
      "conversation": "abc123",
      "id": "abc123|0000",
      "text": "hello",
      "from": "user1"
    },
    {
      "conversation": "abc123",
      "id": "abc123|0001",
      "text": "Nice to see you, user1!",
      "from": "bot1"
    }
  ],
  "watermark": "0001a-95"
}
```

計時考量

雖然 Direct Line 是有潛在計時間隔的多方通訊協定，但通訊協定和服務的設計旨在更容易建置可靠的用戶端。Get Messages 回應中傳送的 `watermark` 屬性是可靠的。只要用戶端逐字重新執行浮水印，就不會錯過任何訊息。

用戶端應該選擇較符合其預計用途的輪詢間隔。

- 服務對服務應用程式通常會使用 5 秒或 10 秒的輪詢間隔。
- 面向用戶端的應用程式通常使用 1 秒的輪詢間隔，並在用戶端傳送的每則訊息後，發出約 300 毫秒的其他要求（以快速擷取 Bot 的回應）。此 300 毫秒的延遲應根據 Bot 的速度和傳輸時間加以調整。

其他資源

- [重要概念](#)
- [驗證](#)
- [開始對話](#)
- [將訊息傳送至 Bot](#)

API 參照 - 直接線路 API 1.1

2019/2/28 • [Edit Online](#)

IMPORTANT

本文包含直接線路 API 1.1 的參照資訊。如果要在用戶端應用程式與 Bot 之間建立新連線，請改為使用[直接線路 API 3.0](#)。

使用直接線路 API 1.1，您就能讓用戶端應用程式與 Bot 進行通訊。直接線路 API 1.1 會透過 HTTPS 使用業界標準的 REST 和 JSON。

基底 URI

若要存取直接線路 API 1.1，請將此基底 URI 使用於所有 API 要求：

```
https://directline.botframework.com
```

headers

除了標準的 HTTP 要求標頭之外，直接線路 API 要求必須包含指定密碼或權杖的 `Authorization` 標頭，藉此驗證發出要求的用戶端。您可以指定讓 `Authorization` 標頭使用「持有人」配置或「BotConnector」配置。

持有人配置：

```
Authorization: Bearer SECRET_OR_TOKEN
```

BotConnector 配置：

```
Authorization: BotConnector SECRET_OR_TOKEN
```

如需如何取得您的用戶端用來驗證其 Direct Line API 要求之祕密或權杖的詳細資料，請參閱[驗證](#)。

HTTP 狀態碼

與每個回應一併傳回的 [HTTP 狀態碼](#) 會指出對應要求的結果。

HTTP 狀態碼	意義
200	要求成功。
204	要求成功，但未傳回任何內容。
400	要求的格式不正確或有其他錯誤。
401	用戶端未獲授權，無法提出要求。此狀態碼出現的原因通常是遺漏 <code>Authorization</code> 標頭或格式不正確。
403	不允許用戶端執行要求的作業。此狀態碼出現的原因通常是 <code>Authorization</code> 標頭指定了無效的權杖或密碼。

HTTP 狀態碼	意義
404	找不到要求的資源。此狀態碼通常表示要求的 URI 無效。
500	Direct Line 服務內發生內部伺服器錯誤
502	Bot 內發生錯誤；Bot 無法使用，或傳回錯誤。此為常見錯誤碼。

權杖作業

透過這些作業可建立或重新整理用戶端用於存取單一對話的權杖。

作業	說明
產生權杖	產生新對話的權杖。
重新整理權杖	重新整理權杖。

產生權杖

產生適用於一個對話的權杖。

```
POST /api/tokens/conversation
```

要求本文	n/a
傳回	代表權杖的字串

重新整理權杖

重新整理此權杖。

```
GET /api/tokens/{conversationId}/renew
```

要求本文	n/a
傳回	代表新權杖的字串

交談作業

使用這些作業可開啟您與 Bot 的交談，並在用戶端與 Bot 之間交換訊息。

作業	說明
開始對話	開啟與 Bot 的新對話。
取得訊息	接收來自 Bot 的訊息。

作業	說明
傳送訊息	將訊息傳送至 Bot。
上傳與傳送檔案	上傳檔案後，以附件型式傳送檔案。

開始對話

開啟與 Bot 的新對話。

```
POST /api/conversations
```

要求本文	n/a
傳回	Conversation 物件

取得訊息

針對指定交談擷取來自 Bot 的訊息。設定 `watermark` 要求 URI 中的參數，以指出最近期內用戶端所見到的訊息。

```
GET /api/conversations/{conversationId}/messages?watermark={watermark_value}
```

要求本文	n/a
傳回	<code>MessageSet</code> 物件。回應中包含 <code>watermark</code> 做為 <code>MessageSet</code> 物件的屬性。用戶端應持續送出 <code>watermark</code> 值來瀏覽整頁的可用訊息，直至不再收到任何訊息。

傳送訊息

將訊息傳送至 Bot。

```
POST /api/conversations/{conversationId}/messages
```

要求本文	訊息物件
傳回	回應的本文中沒有任何傳回的資料。訊息成功傳送時，服務會以 HTTP 204 狀態碼回應。藉由運用 取得訊息 作業，用戶端可能會取得其所傳送的訊息（以及任何 Bot 傳送至用戶端的訊息）。

上傳與傳送檔案

上傳檔案後，以附件型式傳送檔案。設定要求 URI 中的 `userId` 參數以定傳送附件之使用者的識別碼。

```
POST /api/conversations/{conversationId}/upload?userId={userId}
```

要求本文	若為單一附件，請在要求本文填入檔案內容。若有多個附件，請建立多部分要求本文，每個附件均包含其各自的一部份；(選擇性) 亦可採用屬於做為指定附件之容器的訊息物件的單一部分的方式。如需詳細資訊，請參閱 將訊息傳送至 Bot 。
傳回	回應的本文中沒有任何傳回的資料。訊息成功傳送時，服務會以 HTTP 204 狀態碼回應。藉由運用 取得訊息 作業，用戶端可能會取得其所傳送的訊息 (以及任何 Bot 傳送至用戶端的訊息)。

NOTE

上傳的檔案會在 24 小時後刪除。

結構描述

直接線路 1.1 結構描述是 Bot Framework v1 結構描述的簡化版本，其中包含下列物件。

訊息物件

用於定義用戶端與 Bot 之間來回傳送的訊息。

屬性	類型	說明
id	字串	唯一識別由直接路線所指派之訊息的識別碼。
conversationId	字串	可識別交談的識別碼。
created	字串	訊息建立的日期和時間，以 ISO-8601 格式表示。
from	字串	可識別使用者是否為訊息寄件者的識別碼。用戶端在建立訊息時，必須對固定使用者識別碼設定這項屬性。雖然直接線路會在未提供使用者識別碼時指派一個識別碼，不過，如此一來，通常會導致非預期的行為。
text	字串	Bot 與使用者之間傳送的訊息文字。
channelData	物件	包含通道專用內容的物件。某些通道提供的功能，需要使用無法以附件結構描述來呈現的其他資訊。在該等情況下，請將此屬性設定為通道文件中所定義的通道專用內容。此資料會在未經修改的狀態下，在用戶端和 Bot 之間傳送。這個屬性必須設為複雜物件或保留空白。請勿設為字串、數字或其他簡單類型。

屬性	類型	說明
images	string[]	字串陣列，當中包含訊息中所含影像的 URL。某些情況下，此陣列中的字串可能是相對 URL。如果此陣列中的任何字串並未以「http」或「https」開頭，請在字串前加上 <code>https://directline.botframework.com</code> 以構成完整 URL。
attachments	Attachment[]	呈現訊息中所含非影像附件之附件物件陣列。陣列中的每個物件均具有 <code>url</code> 屬性和 <code>contentType</code> 屬性。用戶端從 Bot 接收的訊息中， <code>url</code> 屬性有時會指定相對 URL。對於任何未以「http」或「https」開頭的 <code>url</code> 屬性值，請在字串前加上 <code>https://directline.botframework.com</code> 以構成完整 URL。

下列範例為含有所有可能屬性的訊息物件。大部分情況下，建立一則訊息時，用戶端只需提供 `from` 屬性和至少一個內容屬性（例如 `text`、`images`、`attachments` 或 `channelData`）。

```
{
  "id": "CuvLPID4kDb|00000000000000000004",
  "conversationId": "CuvLPID4kDb",
  "created": "2016-10-28T21:19:51.0357965Z",
  "from": "examplebot",
  "text": "Hello!",
  "channelData": {
    "examplefield": "abc123"
  },
  "images": [
    "/attachments/CuvLPID4kDb/0.jpg?..."
  ],
  "attachments": [
    {
      "url": "https://example.com/example.docx",
      "contentType": "application/vnd.openxmlformats-officedocument.wordprocessingml.document"
    },
    {
      "url": "https://example.com/example.doc",
      "contentType": "application/msword"
    }
  ]
}
```

MessageSet 物件

用於定義一組訊息。

屬性	類型	說明
messages	訊息[]	訊息物件陣列。
watermark	字串	一組訊息中，訊息的最大浮水印。用戶端可以在擷取 Bot 的訊息時，使用 <code>watermark</code> 值指出從 Bot 接收的最新訊息。

Attachment 物件

用於定義非影像附件。

屬性	類型	說明
contentType	字串	附件中內容的媒體類型。
url	字串	附件內容的 URL。

交談物件

定義 Direct Line 對話。

屬性	類型	說明
conversationId	字串	唯一識別指定權杖所適用對話的識別碼。
token	字串	適用於指定對話的權杖。
expires_in	number	權杖到期之前的秒數。

錯誤物件

定義錯誤。

屬性	類型	說明
code	字串	錯誤碼。下列其中一項值： MissingProperty 、 MalformedData 、 NotFound 、 ServiceError 、 Internal 、 InvalidRange 、 NotSupported 、 NotAllowed 、 BadCertificate 。
message	字串	錯誤的描述。
statusCode	number	狀態碼。

ErrorMessage 物件

標準化訊息的錯誤承載。

屬性	類型	說明
error	錯誤	含有錯誤相關資訊的 Error 物件。

實體和活動類型

2019/5/14 • [Edit Online](#)

實體是活動的一部分，會提供關於活動或對話的其他資訊。

NOTE

不同部分的 SDK 會定義個別的實體類別或元素。

- 對於訊息實體，請參閱[實體與活動類型](#) (英文)。
- 針對 LUIS 辨識實體，請參閱[使用 LUIS 辨識意圖和實體](#)。

NOTE

不同部分的 SDK 會定義個別的實體類別或元素。

- 對於訊息實體，請參閱[實體與活動類型](#) (英文)。
- 對於 LUIS 辨識實體，請參閱[擷取實體](#) (英文)。

實體

訊息的 Entities 屬性是開放式 [schema.org](#) 物件的陣列，其允許交換通道與 Bot 之間的通用內容中繼資料。

提及實體

許多通道都支援 Bot 或使用者在交談內容中「提及」某人的功能。若要在訊息中提及使用者，請以 mention 物件填入訊息的實體屬性。提及物件包含下列屬性：

屬性	說明
類型	實體的類型 ("mention")
Mentioned	通道帳戶物件，指出所提及的使用者
文字	activity:text 屬性中的文字，表示 mention 本身 (可以是空白或 null)

此程式碼範例示範如何將 mention 實體新增至實體集合。

- [C#](#)
- [JavaScript](#)

```

var entity = new Entity();
entity.SetAs(new Mention()
{
    Text = "@johndoe",
    Mentioned = new ChannelAccount()
    {
        Name = "John Doe",
        Id = "UV341235"
    }
});
entities.Add(entity);

```

TIP

嘗試判斷使用者意圖時，Bot 可以忽略提到它的訊息部分。呼叫 `GetMentions` 方法，並評估 `Mention` 回應中傳回的物件。

放置物件

以 Place 物件或 GeoCoordinates 物件填入訊息的 entities 屬性，即可在訊息中傳達位置相關資訊。

place 物件包含下列屬性：

屬性	說明
類型	實體的類型 ("Place")
位址	說明或郵寄地址物件 (未來)
地理區域	GeoCoordinates
HasMap	地圖的 URL 或地圖物件 (未來)
Name	位置的名稱

geoCoordinates 物件包含下列屬性：

屬性	說明
類型	實體的類型 ("GeoCoordinates")
Name	位置的名稱
經度	位置的經度 (WGS 84)
經度	位置的緯度 (WGS 84)
Elevation	位置的提升 (WGS 84)

此程式碼範例示範如何將 place 實體新增至實體集合：

- [C#](#)
- [JavaScript](#)

```
var entity = new Entity();
entity.SetAs(new Place())
{
    Geo = new GeoCoordinates()
    {
        Latitude = 32.4141,
        Longitude = 43.1123123,
    }
});
entities.Add(entity);
```

取用實體

- [C#](#)
- [JavaScript](#)

若要取用實體，請使用 `dynamic` 關鍵字或強型別類別。

此程式碼範例示範如何使用 `dynamic` 關鍵字來處理訊息 `Entities` 屬性內的實體：

```
if (entity.Type == "Place")
{
    dynamic place = entity.Properties;
    if (place.geo.latitude > 34)
        // do something
}
```

此程式碼範例示範如何使用強型別類別來處理訊息 `Entities` 屬性內的實體：

```
if (entity.Type == "Place")
{
    Place place = entity.GetAs<Place>();
    GeoCoordinates geo = place.Geo.ToObject<GeoCoordinates>();
    if (geo.Latitude > 34)
        // do something
}
```

活動類型

此程式碼範例示範如何處理 `message` 類型的活動：

- [C#](#)
- [JavaScript](#)

```
if (context.Activity.Type == ActivityTypes.Message){
    // do something
}
```

活動類型有好幾種；活動可以屬於數個會傳遞最常見訊息的不同類型。說明和進一步的資訊可於[活動結構描述頁面](#)中找到。

其他資源

- [Activity 類別 \(英文\)](#)

虛擬助理概觀

2019/5/14 • [Edit Online](#)

概觀

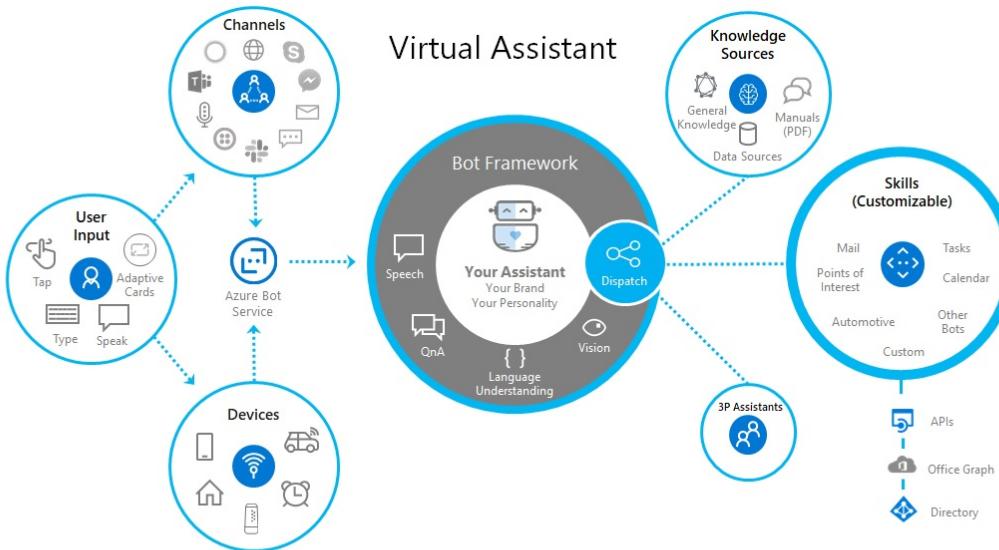
客戶與合作夥伴極需提供專屬於其品牌的交談式助理，並且還需要為他們的使用者量身打造，以及在各種畫布和裝置上提供使用。

開放原始碼的虛擬助理解決方案會延續針對 Bot Framework SDK 開發的 Microsoft 開放原始碼方法，為您提供一組核心功能來完全控制終端使用者的體驗。

此範本納入先前的企業範本，並且將所有最佳做法與透過建置交談式體驗所識別的支援元件結合在一起，大幅簡化新 Bot 專案的建立程序，包括：基本交談式意圖、分派整合、QnA Maker、Application Insights 和自動化部署。

我們深信客戶應該擁有他們的客戶關係和見解，並且加以擴充。因此，任何虛擬小幫手都會透過在 GitHub 上開放原始碼，將使用者體驗的完整控制權提供給客戶和合作夥伴。名稱、語音和特質都可根據組織需求加以變更。虛擬小幫手解決方案簡化了自行建立小幫手的程序，可讓您在幾分鐘內開始使用此功能，然後使用我們的端對端開發工具來擴充。

虛擬小幫手功能的範圍很廣泛，通常會提供一系列功能給使用者。為提升開發人員的生產力，並啟用可重複使用對話式體驗的活躍生態系統，我們會針對可重複使用的對話式技能，提供初始範例給開發人員。這些技術可新增到對話式應用程式，以開啟特定對話體驗，例如尋找景點、與行事曆、工作、電子郵件互動等許多其他案例。技術皆開放為完全自訂，且包含適用於多種語言、對話方塊和程式碼的語言模型。



Azure Platform

Storage

Analytics

Security

Integration

IoT Edge

Microsoft

開始使用

如需詳細資訊，請探索[虛擬助理及技能文件](#)。

產品內容

虛擬助理範本結合了數個我們在建置交談式體驗時發現的最佳做法，並自動整合對 Bot Framework 開發人員非常有幫助的元件。本節涵蓋了一些重要決策的背景知識，以協助說明為什麼範本會如此運作。

虛擬助理範本現在納入先前的企業範本功能，包括多種語言的基礎交談意圖、分派、QnA 和交談見解。目前提供下列小幫手相關功能，並已規劃進一步的功能，我們將與客戶及合作夥伴密切合作，協助提供藍圖資訊。

功能	說明
登入	可讓您的小幫手歡迎使用者及收集初始資訊的範例上線流程。
事件架構	虛擬小幫手的內容中的事件可讓用戶端應用程式主控小幫手(在網頁瀏覽器中或在汽車或喇叭等裝置上)，以交換有關使用者或裝置事件的資訊，同時也能接收事件以執行裝置作業。
連結的帳戶	在語音導向的案例中，使用者透過語音命令輸入其使用者名稱和密碼來支援系統並不切實際。因此，個別的小幫手體驗讓使用者有機會登入，並提供虛擬小幫手擷取權杖以供日後使用的權限。
技術提升	目前有一組廣泛的常見功能需要每位開發人員自行建置。我們的虛擬小幫手解決方案包含新的「技術」功能，僅透過設定就可讓新功能插入虛擬小幫手，並提供技術的驗證機制來要求下游活動的權杖。
景點技術	預覽景點 (Pol) 技術可提供全方位語言模型，以便尋找景點並要求指示。這項技術目前可與 Azure 地圖服務整合。
行事曆技術	預覽行事曆技術，可提供常見行事曆相關活動的全方位語言模型。這項技術目前已整合到 Microsoft Graph (Office 365/Outlook.com) 中，而且很快就會提供 Google API 支援。
電子郵件技術	預覽電子郵件技術，可提供常見電子郵件相關活動的全方位語言模型。這項技術目前已整合到 Microsoft Graph (Office 365/Outlook.com) 中，而且很快就會提供 Google API 支援。
ToDo 技術	預覽 ToDo 技術，可提供常見工作相關活動的全方位語言模型。這項技術目前已整合到 OneNote 中，而且很快就會提供 Microsoft Graph (outlookTask) 支援。
裝置整合	採用調適型卡片的 Azure Bot Service SDK (DirectLine) 和語音 SDK 都能夠輕鬆進行裝置的跨平台整合。已規劃其他裝置整合範例和平台 (包括 Edge)。
測試載入器	除了 Bot Framework 模擬器，還提供了 WebChat 型測試載入器，以便測試更複雜的驗證案例。簡單的主控台型測試載入器會示範交換訊息的方法，以協助塑造不費力的裝置整合。
自動化部署	會自動部署助理所需的所有 Azure 資源：Bot 註冊、Azure App Service、LUIS、QnAMaker、內容審查工具、CosmosDB、Azure 儲存體和 Application Insights。此外，系統會建立、定型及發行適用於所有技術的 LUIS 模型、QnAMaker 和分派模型，以便立即測試。
汽車業語言模型	即將推出汽車業語言模型，其中涵蓋電話、導航及車用功能控制等核心領域

範例案例

虛擬助理橫跨廣泛的產業。以下一些範例案例供您參考。

- 汽車業：藉由將啟用語音的個人助理整合到汽車中，可讓終端使用者執行傳統汽車操作 (例如導航和廣播) 及

攸關生產力的案例，例如在您快遲到時更改會議時間、新增項目到您的工作清單，以及提供更積極的體驗，例如汽車可以在您啟動引擎、回家或啟用定速巡航時提供要完成的工作建議。調適性卡片會在透過隨按即說 (Push-To-Talk) 或喚醒字互動來執行的前端單元和語音整合中進行轉譯。

- **旅遊服務業**: 將啟用語音的個人助理整合到旅館房間裝置，可提供廣泛的住宿相關案例 (例如延長您的住宿時間、要求延後退房、客房服務)，包括門房服務和尋找當地餐廳和景點的功能。若連結您的生產力帳戶 (選用)，可開啟更多個人化體驗，例如建議的提醒呼叫、天氣警告及留宿期間的模式學習。目前您已可以在飯店內體驗到電視的個人化發展。
- **企業**: 藉由將啟用語音和文字的品牌員工助理體驗整合到企業裝置和現有的對話畫布 (例如 Teams、WebChat、Slack) 中，可讓員工管理他們的行事曆、尋找可用的會議室、尋找有特定技術的人員，或執行 HR 相關的作業。

虛擬小幫手原則

您的資料、您的品牌和您的體驗

您可以擁有和控制終端使用者體驗的所有層面。這包括商標、名稱、語音、特質、回應和虛擬人偶。虛擬小幫手的原始碼和支持技術會完全提供給您，讓您可根據需求來進行調整。

虛擬小幫手將會部署在您的 Azure 訂用帳戶內。因此，小幫手所產生的所有資料 (詢問的問題、使用者行為等) 會完全包含在您的 Azure 訂用帳戶中。請參閱[認知服務的 Azure 可信任雲端和信任中心的 Azure 區段](#)，以更具體地取得詳細資訊。

撰寫一次，隨處內嵌

虛擬小幫手會利用 Microsoft 的對話式 AI 平台，因此可以透過任何 Bot Framework 通道呈現 – 例如 WebChat、FaceBook Messenger、Skype 等。

此外，透過 [Direct Line](#) 通道，我們可以將體驗內嵌至桌面和行動應用程式，包括汽車、喇叭、鬧鐘等裝置。

企業級解決方案

虛擬小幫手解決方案以 Azure Bot 服務、Language Understanding 認知服務、整合語音及一整組支援 Azure 的元件作為基礎，這表示您可以從 [Azure 全域基礎結構](#) 中獲益，包括 ISO 27018、HIPPA、PCI DSS、SOC 1、2 和 3 認證。

此外，Language Understanding 支援由 LUIS 認知服務提供，可支援一整組[此處所列](#)的語言。[翻譯工具認知服務](#)提供額外的機器翻譯功能，可進一步擴大虛擬小幫手的適用範圍。

整合與情境感知

虛擬小幫手可以整合到您的裝置和生態系統中，讓您感受到真正的整合和智能體驗。此情境感知功能可讓您開發更智慧型的體驗，並進一步提供別人沒有的個人化設定。

第三方小幫手整合

虛擬小幫手可讓您提供自己獨有的經驗，但也可移交給終端使用者選擇的數位助理，以處理特定類型的問題。

彈性整合

我們的虛擬小幫手架構具有彈性，可與您在裝置型語音或自然語言處理功能上所做的投資進行整合，當然也可以與您現有的後端系統和 API 整合。

調適性卡片

[調適性卡片](#) 可讓虛擬小幫手傳回使用者體驗元素 (例如卡片、影像、按鈕) 及文字回應。如果裝置或對話畫布上有一個畫面，這些調適性卡片可以跨各種支援使用者體驗的裝置與平台進行轉譯 (如果適用)。您可以在[此處](#)找到調適性卡片的範例，而[此處](#)的文件中有轉譯選項的相關資訊。

技術

除了基本的小幫手功能，還有一組廣泛的常見功能需要每位開發人員自行建置。生產力是絕佳範例，每個組織可能都需要建立語言模型 (LUIS)、對話 (程式碼)、整合 (程式碼) 和語言生成 (回應)，以啟用受歡迎的行事曆、工作或電子郵件體驗。

這會因為需要支援大量語言而讓後續作業變得更加複雜，並產生每個組織建置自有小幫手所需的大量工作。

我們的虛擬小幫手解決方案包含新的「技術」功能，僅透過設定就可讓新功能插入自訂小幫手。

每項技術（語言模型、對話、整合程式碼和語言生成）的各方面都可完全由開發人員自訂，因為與虛擬小幫手有關的完整原始程式碼皆可在 GitHub 取得。

開始使用

在此 GitHub 存放庫中可取得虛擬助理解決方案，而虛擬助理小組會定期更新該解決方案。在相同的存放庫中可取得更詳細的文件，而且可以透過 GitHub 意見反應機制直接提供問題/意見反應。

虛擬助理 - 範本大綱

2019/5/14 • [Edit Online](#)

NOTE

本主題適用於 SDK 的 v4 版本。

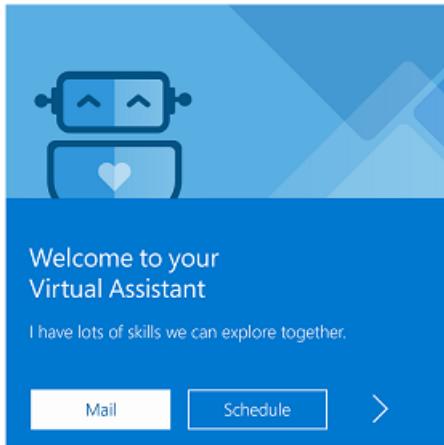
虛擬助理範本結合了數個我們在建置交談式體驗時發現的最佳做法，並自動整合對 Bot Framework 開發人員非常有幫助的元件。本節涵蓋了一些重要決策的背景知識，以協助說明為什麼範本會如此運作。

功能	說明
簡介	在開始對話時，採用的簡介訊息
輸入指標	在對話期間自動化視覺輸入指標，並針對長時間執行的作業重複此動作
基本 LUIS 模型	支援常見的意圖，例如取消、說明、呈報等等
基本對話方塊	擷取基本使用者資訊，以及取消和說明意圖等中斷邏輯的對話方塊流程
基本回應	基本意圖和對話方塊的文字及語音回應
常見問題集	與 QnA Maker 整合以回答知識庫的一般問題
閒聊	專業人員閒聊模型，以提供常用查詢的標準解答（ 了解更多 ）
發送器	整合式分派模型，用來識別指定的語句應該由 LUIS 或 QnA Maker 處理。
語言支援	提供英文、法文、義大利文、德文、西班牙文和中文
文字記錄	Azure 儲存體中儲存的所有對話的文字記錄
遙測	Application Insights 整合，以收集所有對話的遙測資料
分析	Power BI 儀表板範例，可讓您深入了解對話式體驗。
自動化部署	使用 Azure ARM 範本輕鬆部署所有上述服務。

簡介卡片

許多對話式體驗的主要問題是使用者不知道如何開始，這會導致 Bot 無法有效地回答許多常見問題。第一印象非常重要！簡介卡片提供了向終端使用者介紹 Bot 功能的機會，並且提供一些建議的初始問題給使用者，以作為使用此功能的起點。這也是讓 Bot 展現特質的絕佳機會。

系統會提供簡單的簡介卡片，作為您可視需要適應的標準，而當使用者完成上線對話（由簡介卡片上的 [開始使用] 按鈕所觸發）時，後續互動時會顯示傳回的使用者卡片。



基本 Language Understanding (LUIS) 意圖

每個 Bot 都應該能處理基礎等級的對話式語言理解。例如，問候語就是每個 Bot 應能輕鬆處理的基本項目。一般而言，開發人員需要建立這些基本意圖，並提供用於入門的初始訓練資料。虛擬助理範本提供 LU 範例檔案來協助您入門，同時避免在每次使用專案時都必須建立這些檔案，並確保有現成的基礎功能可用。

LU 檔案會以英文、中文、法文、義大利文、德文、西班牙文提供下列意圖。

意圖	範例語句
取消	取消、沒關係
呈報	我可以洽詢人員嗎？
FinishTask	完成、全部完成
GoBack	返回
說明	可以請您提供協助嗎？
Repeat	可以再說一次嗎？
SelectAny	任一
SelectItem	第一個
SelectNone	以上皆非
ShowNext	顯示較多
ShowPrevious	顯示上一個
StartOver	<i>restart</i>
Stop	<i>stop</i>

LU 格式類似於 MarkDown，可讓您輕鬆進行修改與原始檔控制。[LuDown](#) 工具則可用來將 .LU 檔案轉換成 LUIS 模型，然後透過入口網站或相關聯的 [LUIS](#) CLI (命令列) 工具將其發行至您的 LUIS 訂用帳戶。

遙測

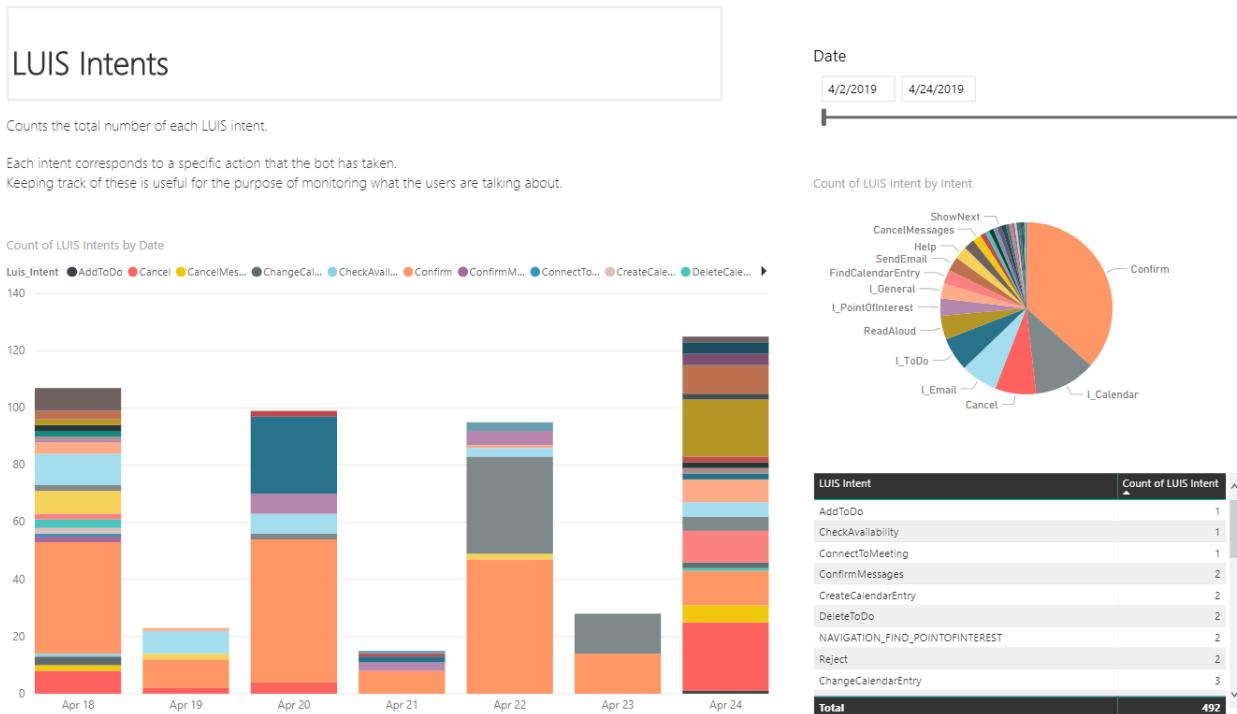
經過證實，提供您 Bot 的業務開發見解相當有價值。此見解可協助您了解業務開發的層級、使用者所使用的 Bot 功能（意圖），以及人們問哪些問題時 Bot 無法回答 - 將 Bot 的知識缺口標示出來，這可能可以透過 QnA Maker 之類的文件來解決。

整合 Application Insights 可提供現成的重要作業/技術見解，但這也可用來擷取特定的 Bot 相關事件 - 傳送和接收的訊息及 LUIS 和 QnA Maker 作業。

Bot 層級的遙測在本質上與技術和作業方面的遙測相連結，因此可讓您檢查 Bot 如何回答指定使用者的問題，反之亦然。

中介軟體元件結合 QnA Maker 和 LuisRecognizer SDK 類別的包裝函式類別，以提供簡潔的方式來收集一組一致的事件。接著，這些一致的事件可交由 Application Insights 工具和 PowerBI 之類的工具來使用。

範例 PowerBI 儀表板是 Bot Framework 解決方案 github 存放庫的一部分，可立即與每個虛擬助理範本搭配運作方式。如需詳細資訊，請參閱[分析](#)一節。



發送器

在第一波交談方式體驗中，產生良好效果的主要設計模式就是利用 Language Understanding (LUIS) 和 QnA Maker。LUIS 可透過 Bot 為終端使用者執行的工作來訓練，而 QnA Maker 可透過較廣泛的知識來訓練。

所有傳入的語句（問題）都會路由至 LUIS 以進行分析。如果無法識別指定語句的意圖，則會將其標示為「無」意圖。接著會使用 QnA Maker 來嘗試為終端使用者尋找回答。

儘管此模式運作良好，還是有可能在兩個主要案例中遇到問題。

- 如果 LUIS 模型和 QnA Maker 中的語句在部分時間上稍微重疊，這可能會導致異常行為，其中 LUIS 可能會在問題應已導向 QnA Maker 時，嘗試處理該問題。
- 如果有兩個或兩個以上的 LUIS 模型，Bot 必須叫用每一個模型，並執行某種形式的意圖評估比較，以識別傳送指定語句的目的地。因為模型之間沒有通用的基準分數比較，無法有效運作會導致不良的使用者體驗。

[發送器](#)為此提供了簡潔的解決方案，方法是從每個設定的 LUIS 模型擷取語句，以及從 QnA Maker 擷取問題，然後建立中央分派 LUIS 模型。

這可讓 Bot 快速找出應處理指定語句的 LUIS 模型或元件，並確保系統會認為 QnA Maker 資料在意圖處理的最上層，而不是跟之前一樣視為 None 意圖。

此分派工具也會啟用評估，並在部署之前，醒目提示 LUIS 模型之間的混淆和重疊及已標示的 QnA Maker 知識庫問題。

發送器會使用於每個專案 (使用範本所建立) 的核心。分派模型會在 `MainDialog` 類別內使用，可識別目標是 LUIS 模型或 QnA。在 LUIS 的案例中，次要 LUIS 模型會受到叫用，並如往常般傳回意圖和實體。發送器也用於中斷偵測。

The screenshot shows the Microsoft Bot Framework Composer interface. The top navigation bar includes the app name "djvalcbuild1en_Dispatch (V Dispatch)" and a "DASHBOARD" button. The left sidebar has sections for "App Assets" (expanded), "Intents" (selected and highlighted in blue), "Entities", "Improve app performance" (expanded), "Review endpoint utterances", "Phrase lists", and "Patterns". The main content area is titled "Intents" with a question mark icon. It features two buttons: "+ Create new intent" and "+ Add prebuilt domain intent". Below these are three rows of intent data, each with a checkbox, a name, and a count of labeled utterances:

Name	Labeled Utterances
None	3
I_general	448
q_chitchat	633
q_faq	66

QnA Maker

[QnA Maker](#) 可讓非開發人員以問題和答案配對的格式來策展通用知識。此知識可以從常見問題集資料來源和產品手冊中匯入，也可在 [QnAMaker](#) 入口網站內以互動方式匯入。

在 CognitiveModels 的 QnA 資料夾內會以 [LU](#) 檔案格式提供兩個範例 QnA Maker 模型，一個用於常見問題集，一個用於閒聊。[LuDown](#) 接著會用來當作部署指令碼的一部分，以建立 QnA Maker JSON 檔案，然後 [QnA Maker CLI](#) (命令列) 工具會使用此檔案來將項目發行至 QnA Maker 知識庫。

Knowledge base

 Search the knowledge base X

96 QnA pairs

View: 10 1 2 3 4**Question**^ Source: custom editorial

What's your age X Are you young X When were you born X What age are you X Age doesn't really apply to me.
Are you old X How old are you X How long ago were you born X +

Be my friend X Can we be friends X Will you be my best friend X BFFs forever X Certainly.
I want to be your friend. X +

You're right. X That was right X That was correct X That's accurate X Excellent.
Accurate X That's right X Yup, that's true X That's true X Correct X
Yes, that's right X Yes, that's true X +

Ha X Haha X Hahaha X LOL X I'm cracking up X ROFL X + Glad you're pleased!



Good evening X Evening X Good evening to you X + Good evening.

內容仲裁

內容審查工具為選用元件，可偵測潛在的不雅內容，並協助檢查個人識別資訊 (PII)。將此功能整合到 Bot 會很有幫助，這可讓 Bot 針對不雅內容或當使用者共用 PII 資訊時，做出適當反應。比方說，Bot 可能會道歉並移交給人類，或在偵測到 PII 資訊時，不儲存遙測記錄。

系統會提供中介軟體元件，以透過 TurnState 物件上的 `TextModeratorResult` 將該畫面轉換成文字並顯示。

後續步驟

請參閱[使用者入門](#)以了解如何建立和部署虛擬助理。

其他資源

虛擬助理範本的完整原始程式碼位於 [GitHub](#)。

虛擬主題 - 技能概觀

2019/5/14 • [Edit Online](#)

NOTE

本主題適用於 SDK 的 v4 版本。

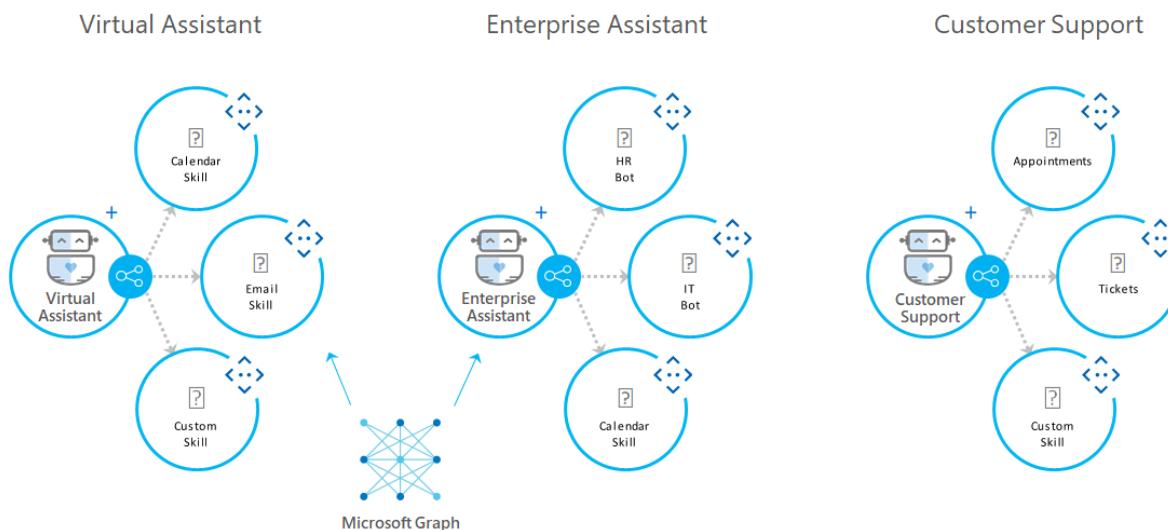
概觀

開發人員藉由將可重複使用的交談式功能 (稱為技能) 拼接在一起，即可撰寫交談式體驗。

在企業內，可以建立父代 Bot，該 Bot 結合了不同小組所擁有的多個子 Bot，或更廣泛地運用其他開發人員所提供的通用功能。透過預覽技能，開發人員可以建立新的 Bot (通常透過虛擬助理範本)，及以利用一項合併所有分派和組態變更的命令列作業新增/移除技能。

技能本身就是可從遠端叫用的 Bot，而且有技能開發人員範本 (.NET、TS) 可供加速新技能的建立。

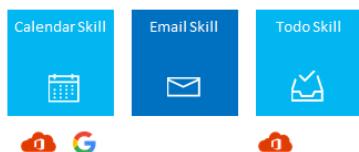
技能的主要設計目標是要維護一致的活動通訊協定，並確保開發體驗盡可能貼近任何一般 V4 SDK Bot。



Bot Framework 技能

此時，我們已提供下列 Bot Framework 技能，這些技能由 Microsoft Graph 提供技能支援並以多種語言提供。

Productivity



Industry



- Open-Source Re-usable Skills (GitHub repo)

- Skills include Language Models, Dialogs, Responses and integration where appropriate.

- Leverages Adaptive Cards throughout and customer can change design, branding and functionality as required.

Experimental



Custom



- Experimental are sample implementations, limited language models, stubbed integration, etc.

- Speech friendly.

NAME	說明
行事曆技能	將行事曆功能新增到您的助理。由 Microsoft Graph 和 Google 提供技術支援。
電子郵件技能	將電子郵件功能新增到您的助理。由 Microsoft Graph 和 Google 提供技術支援。
待辦事項技能	將工作管理功能新增到您的助理。由 Microsoft Graph 提供技術支援。
景點技能	尋找景點及指示。由 Azure 地圖服務和 FourSquare 提供技術支援。
汽車技能	產業垂直整合技能，用於展示和啟用汽車功能控制。
實驗性技能	新聞、餐廳訂位和天氣。

開始使用

請參閱[使用者入門](#)，以了解如何運用現有的技能並增長自己的技能。

Bot Framework 常見問題集

2019/3/5 • [Edit Online](#)

本文包含關於一些 Bot Framework 常見問題集的解答。

背景與可用性

Microsoft 為何要開發 Bot Framework？

在我們使用對話使用者介面 (CUI) 時，目前只有少數開發人員具備必要的專業知識和工具，可以建立新的對話體驗；或是讓現有的應用程式和服務，具備使用者可以享受的對話介面。我們建立了 Bot Framework，讓開發人員更容易建置良好的 Bot 並且與使用者連線，無論是在哪裡進行對話（包括 Microsoft 的主要通道）。

什麼是 v4 SDK？

Bot Framework v4 SDK 是根據意見反映而建置，並且參考了先前的 Bot Framework SDK。它引入了正確層級的抽象概念，同時採用了豐富的 Bot 建置組塊元件化功能。您可以從簡單的 Bot 開始，並且使用模組化和可延伸的架構，讓您的 Bot 更細緻。您可以在 GitHub 上找到 SDK 的[常見問題集](#)。

Bot Framework SDK 第 3 版存留期支援

SDK V3 Bot 會繼續執行並受 Azure Bot Service 支援。自 Bot Framework SDK V4 發行以來，如同其他架構，我們會以安全性、高優先順序錯誤修正及連接器 / 通訊協定層更新，繼續支援 SDK V3。客戶可預期 2019 年繼續有 v3 支援。

Microsoft 對於支援現有 V3 Bot 的計畫為何？我的 V3 Bot 發生什麼事？我的 V3 Bot 會停止運作嗎？

SDK V3 Bot 會繼續執行並受 Azure Bot Service 支援。自 Bot Framework SDK V4 發行以來，如同其他架構，我們會以安全性、高優先順序錯誤修正及連接器 / 通訊協定層更新，繼續支援 SDK V3。客戶可預期 2019 年繼續有 v3 支援。

- Azure Bot Service 與 Bot Framework V3 兩者都是 GA 產品且受到完全支援。基礎 Bot Framework 通訊協定和連接器程式庫維持不變，而且在 V3 與 V4 SDK 之間共用。
- 2019 年會繼續支援使用 Bot Framework (BotBuilder) V3 SDK 所建立的 Bot。
- 客戶可以繼續使用 Azure 入口網站或 Azure CLI 工具建立 V3 Bot。

寫入至 REST 和 Bot Framework 通訊協定 3.1 的 Bot 發生什麼事？

- Azure Bot Service 與 Bot Framework V3 兩者都是 GA 產品且受到完全支援。
- Bot Framework 通訊協定維持不變，而且在 V3 與 V4 SDK 之間共用。

V3 SDK 會有更多更新、額外開發，或只有錯誤修正？

- 我們會以次要增強功能（主要在連接器層），以及安全性和高優先順序的錯誤修正來更新 V3。
- 我們會根據錯誤修正和/或必要的通訊協定變更，視需要每年發行 V3 更新兩次。
- 目前的計畫是將 V3 的次要和修補版本發佈至 NuGet 和 NPM，以供我們的 C# 和 JavaScript SDK 使用。

為何 V4 與 V3 無法回溯相容？

- 在通訊協定層級，交談式應用程式（也就是您的 Bot）與不同通道之間的通訊會使用 Bot Framework 活動通訊協定（V3 與 V4 之間的通訊使用相同的通訊協定）。相同的基礎 Azure Bot Service (AZURE BOT SERVICE) 基礎結構同時支援 V3 和 V4 Bot。
- Bot Framework SDK V4 使用模組化且可擴充的 SDK 架構，提供以交談式為主的開發體驗，讓開發人員能建立強大且複雜的聊天應用程式。V4 擴充式設計是以客戶意見反應為基礎，這意味著 SDK V3 對話模型和基本項目過於狹隘並限制擴充性。

什麼是一般移轉策略？我有 V3 Bot，如何將其遷移至 V4/ 是否可以將 V3 Bot 遷移至 V4？

- 若要了解如何將 V3 Bot 遷移至 V4，請參閱 [v3 和 v4 .NET SDK 之間的差異](#)。
- 在此階段，使用 SDK V3 至 SDK V4 建立的 Bot 移轉說明會以文件和範例形式呈現。我們目前不打算在 SDK V4 中提供任何 SDK V3 相容性層，來允許 V3 建置要在 V4 Bot 內運作的 Bot。
- 如果您在生產環境中已經有 Bot Framework SDK V3 Bot，別擔心，其會繼續按現狀針對可預見的未來運作。
- Bot Framework SDK V4 是非常成功的 V3 SDK 進化版本。V4 為主要版本發行，其中包含防止 V3 Bot 在較新 V4 SDK 上執行的重大變更。

我應該使用 V3 或 V4 建置新的 Bot？

- 在新的交談式體驗中，我們建議您使用 Bot Framework SDK V4 啟動新的 Bot。
- 如果您已熟悉 Bot Framework SDK V3，您應該花點時間了解新的 [Bot Framework SDK V4](#) 所提供的新版本和功能。
- 如果您在生產環境中已經有 Bot Framework SDK V3 Bot，別擔心，其會繼續按現狀針對可預見的未來運作。
- 您可以透過 Azure 入口網站和 Azure 命令列，建立 Bot Framework SDK V4 及舊版 V3 Bot。

聲道

何時會將更多的對話體驗新增至 Bot Framework？

我們計劃持續改進 Bot Framework，包括額外的通道，但是目前尚無法提供排程。

如果您希望將特定通道新增至架構，請[讓我們知道](#)。

我有想要能夠使用 Bot Framework 設定的通訊通道。可以與 Microsoft 合作完成嗎？

我們沒有一般機制可以讓開發人員將新通道新增至 Bot Framework，但是您可以透過[直接線路 API](#) 將您的 Bot 連線到應用程式。如果您是通訊通道的開發人員，並且想要與我們配合，在 Bot Framework 中啟用您的通道，[歡迎與我們聯繫](#)。

如果我想要建立適用於 Skype 的 Bot，應該使用哪些工具和服務？

Bot Framework 的設計訴求，是為 Skype 和其他許多通道建置、連線及部署高品質、回應靈敏、高效能和可調整的 Bot。SDK 可以用來建立文字/簡訊、影像、按鈕和支援卡片的 Bot (涵蓋了現今對話體驗的大多數 Bot 互動)，以及 Skype 特定的 Bot 互動，例如豐富的音訊和視訊體驗。

如果您已經有絕佳的 Bot，並且想要觸及 Skype 對象，您的 Bot 可以透過適用於 REST API 的 Bot Framework (前提是具有可存取網際網路的 REST 端點)，輕易地連線到 Skype (或任何受支援的通道)。

安全性和隱私權

向 Bot Framework 註冊的 Bot 是否會收集個人資訊？如果是的話，我要如何確定資料是安全的？隱私權呢？

每個 Bot 都是它自己的服務，這些服務的開發人員必須根據其開發人員管理辦法，提供服務條款及隱私權聲明。您可以從 Bot 目錄中的 Bot 卡片，存取這項資訊。

為了提供 I/O 服務，Bot Framework 會從您使用的聊天服務，將您的訊息和訊息內容 (包括您的識別碼) 傳輸到 Bot。

可以在自己的伺服器上裝載 Bot 嗎？

是。您的 Bot 可以裝載在網際網路上的任何位置。在您自己的伺服器上、在 Azure 中，或在任何其他資料中心。唯一的需求是 Bot 必須公開可公開存取的 HTTPS 端點。

如何從服務中禁止或移除 Bot？

使用者可以透過目錄中 Bot 的連絡人卡片，回報行為異常的 Bot。開發人員必須遵守 Microsoft 的服務條款，才能參與服務。

我需要將哪個特定 URL 列入公司防火牆的白名單中，才能存取 Bot Framework 服務？

如果您有輸出防火牆會封鎖 Bot 對網際網路的流量，您必須將該防火牆中的下列 URL 列入白名單：

- login.botframework.com (Bot 驗證)
- login.microsoftonline.com (Bot 驗證)
- westus.api.cognitive.microsoft.com (適用於 Luis.ai NLP 整合)
- state.botframework.com (適用於原型設計的 Bot 狀態儲存體)
- cortanabfchanneleastus.azurewebsites.net (Cortana 通道)
- cortanabfchannelwestus.azurewebsites.net (Cortana 通道)
- *.botframework.com (通道)

我可以封鎖所有對 Bot 的流量 (來自 Bot 連接器服務的流量除外) 嗎？

沒有。這種將 IP 位址或 DNS 列入白名單的做法並不切實際。Bot Framework Connector Service 裝載於全球的 Azure 資料中心，而 Azure IP 清單會不斷地變更。將某些 IP 位址列入白名單可能會運作一天，並隨著 Azure IP 位址變更而造成隔天中斷。

何者可讓我的 Bot 保持安全，以便用戶端模擬 Bot Framework Connector Service？

1. 每個 Bot 要求隨附的安全性權杖內都有 ServiceUrl 編碼，這表示即使攻擊者可以存取權杖，也無法將對話重新導向新的 ServiceUrl。這是由 SDK 的所有實作強制執行，並記載於我們的驗證[參考](#)資料中。
2. 如果傳入權杖遺漏或格式不正確，Bot Framework SDK 不會在回應中產生權杖。如果 Bot 的設定不正確，這會限制可以造成多少損害。
3. 在 Bot 內，您可以手動檢查權杖中提供的 ServiceUrl。這可讓 Bot 在發生服務拓撲變更時更容易受損，所以這是可行的，但不建議這麼做。

請注意，這些都是從 Bot 到網際網路的輸出連線。Bot Framework Connector Service 沒有將用來與 Bot 對話的 IP 位址或 DNS 名稱清單。不支援將輸入 IP 位址列入白名單。

速率限制

什麼是速率限制？

Bot Framework 服務必須保護自己本身及其客戶免於不當呼叫模式 (例如拒絕服務的攻擊) 的影響，因此沒有單一 Bot 會對其他 Bot 的效能造成不良影響。為了實現這種保護，我們已在所有端點上新增了速率限制 (也稱為節流)。藉由強制執行速率限制，我們可以限制 Bot 進行特定呼叫的頻率。例如：在啟用了速率限制的情況下，如果 Bot 想要張貼大量的活動，則必須在一段期間內將它們分散。請注意，速率限制的目的不是要限制 Bot 的總量。限制的設計目的是防止未遵循人類對話模式的對話基礎結構濫用。

如何得知我是否受到影響？

即使數量大，您也不太會遇到速率限制。大部分速率限制只有在大量傳送活動 (來自 Bot 或來自用戶端)、極大的負載測試，或發生錯誤時才會發生。當要求已節流時，會傳回 HTTP 429 (太多要求) 回應以及「稍候再試」標頭，指出重試要求成功之前要等待的時間量 (以秒為單位)。您可以藉由透過 Azure Application Insights 啟用 Bot 的分析，來收集這項資訊。或者，您可以在 Bot 中新增程式碼來記錄訊息。

速率限制如何發生？

它會在符合下列情形時發生：

- Bot 傳送訊息頻率過高
- Bot 的用戶端傳送訊息頻率過高
- 直接線路用戶端要求新的 Web Socket 頻率過高

速率限制有哪些？

我們會持續微調速率限制，使其在保護我們的服務和使用者的同時，盡可能寬鬆。因為閾值偶爾會變更，所以目前我們不會發佈數字。如果您受到速率限制的影響，歡迎透過 bf-reports@microsoft.com 與我們連絡。

相關服務

Bot Framework 與認知服務有何關聯？

Bot Framework 和 [認知服務](#)都是在 Microsoft Build 2016 中引進的新功能，將會在正式上市時整合至 Cortana Intelligence Suite。這兩項服務都是根據多年的研究所建置，並且用於熱門的 Microsoft 產品中。這些功能與 ‘Cortana Intelligence’ 合併，可以讓每個組織利用資料、雲端和智慧的能力，建置專屬的智慧系統，以便開啟新的機會、增加營運速度並且成為業界的領先者。

什麼是 Cortana Intelligence？

Cortana Intelligence 是完全受控的巨量資料、進階分析及智慧套件，您可用來將資料轉換成可採取的智慧行動。它是一個全方位套件，將多年研究所得的技術和 Microsoft 的創新融會在一起（範圍橫跨進階分析、機器學習、巨量資料儲存體和雲端中的處理）以及：

- 可讓您收集、管理及儲存您的所有資料，這些資料會隨著時間，以可調整且安全的方式，順暢又符合成本效益地成長。
- 提供由雲端驅動的簡易且可操作的分析，讓您可以針對大部分嚴苛的問題進行預測、規定以及讓決策自動化。
- 透過認知服務和代理程式啟用智慧系統，讓您能以更關聯式及自然的方式看到、聽到、詮譯及了解世界周遭。

有了 Cortana Intelligence，我們希望可以協助我們的企業客戶開啟新的機會、增加營運速度並且成為業界的領先者。

什麼是直接線路通道？

直接線路是一種 REST API，可讓您將 Bot 新增至您的服務、行動裝置應用程式或網頁。

您可以使用任何語言針對直接線路 API 撰寫用戶端。只要對[直接線路通訊協定](#)編寫程式碼、在直接線路設定頁面中產生祕密，然後從程式碼所在的任何位置與您的 Bot 對話即可。

直接線路適用於：

- iOS、Android、Windows Phone 及其他平台上的行動裝置應用程式
- Windows、OSX 等等平台上的傳統型應用程式
- 在其中您需要比[可內嵌網路聊天通道](#)提供更多自訂的網頁
- 服務對服務應用程式

Bot Framework 其他資源

2019/5/10 • [Edit Online](#)

這些資源針對使用 Bot Framework 開發 Bot, 提供了其他資訊和支援。

IMPORTANT

請使用這些資源的其中一個取得支援, 而不是在此文章上發表評論。我們不會監視此文章的支援要求。

支援類型	連絡人
社群支援	可以使用 <code>botframework</code> 標記在 Stack Overflow 上張貼問題。請注意, Stack Overflow 有一些指導方針, 例如要求必須使用描述性的標題、完整且簡潔的問題陳述以及重現問題的足夠詳細資料。功能要求或過於廣泛的問題都是偏離主題的; 如需詳細資訊, 新的使用者應瀏覽 Stack Overflow 說明中心 。
社群聊天群組	Gitter.IM
使用 Bot	透過發行者電子郵件連絡 Bot 的開發人員
Bot Framework SDK 問題/建議	使用 GitHub 存放庫 上的 [問題] 索引標籤
Azure 幫助及支援	Azure 說明 + 支援
文件問題	將問題提交到 Bot Framework 文件 GitHub 存放庫。
文件更新	按一下文章上的 [編輯] 連結, 並向 Bot Framework 文件 GitHub 存放庫 提交提取要求。
檢舉不當使用	請透過 bf-reports@microsoft.com 與我們連絡

依通路分類的活動

2019/4/18 • [Edit Online](#)

下表顯示哪些事件 (線上的活動) 會來自於哪些通道。

這是資料表的索引鍵：

符號	意義
:white_check_mark:	Bot 應會收到此活動
:x:	Bot 應永遠不會收到此活動
:white_large_square:	目前未決定 Bot 是否會收到此活動

可以將活動有意義地分割成不同的類別。每個類別中會有一個可能的活動資料表。

對話

\	CORTANA	DIRECTLINE	DIRECTLINE (網路聊天)	電子郵件	FACEBOOK	GROUPE	KIK	TEAMS	SLACK	SKYPE	商務用SKYPE	TELEGRAM	TWILIO
訊息	:white_check_mark:	:white_check_mark:	:white_check_mark:	:white_check_mark:	:white_check_mark:	:white_check_mark:	:white_check_mark:	:white_check_mark:	:white_check_mark:	:white_check_mark:	:white_check_mark:	:white_check_mark:	:white_check_mark:
MessageReaction	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:white_check_mark:	:x:	:x:	:x:	:x:	:x:

- 所有的通道均傳送訊息活動。
- 使用對話方塊時，通常應一律將訊息活動傳遞到對話方塊上。
- 即使 MessageReaction 是對話很重要的一部分，上述規則可能不適用於 MessageReaction。
- 邏輯上有兩種類型的 MessageReaction：新增和移除

TIP

「訊息反應」就像是之前意見中的「讚」。可能會隨機發生，因此可以視為與按鈕類似。此活動目前是由 Teams 通道傳送。

歡迎使用

\	COR TAN A	DIRE CT LINE	DIRE CT LINE (網路聊天)	電子郵件	FACE BOO K	GRO UPM E	KIK	TEA MS	SLAC K	SKYP E	商務用 SKYPE	TELE GRAM	TWIL IO
Con vers ationUp date	:white_ch eck_mark:	:white_ch eck_mark:	:white_ch eck_mark:	:x:	:white_lar ge_squar e:	:white_ch eck_mark:	:white_ch eck_mark:	:white_ch eck_mark:	:white_ch eck_mark:	:x:	:x:	:white_ch eck_mark:	:x:
ContactRelati onUp date	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:white_ch eck_mark:	:white_ch eck_mark:	:x:	:x:

- 通道通常會傳送 ConversationUpdate 活動。
- 邏輯上有兩種類型的 MessageReaction: 新增和移除
- 如此很容易讓人假設透過連接 ConversationUpdate.Added 即可簡單地實作 Bot「歡迎使用」行為，而這麼做有時的確可行。
- 不過，這是簡化的方法，為了產生可靠的「歡迎使用」行為，Bot 實作也可能需要使用狀態。

應用程式擴充性

\	COR TAN A	DIRE CT LINE	DIRE CT LINE (網路聊天)	電子郵件	FACE BOO K	GRO UPM E	KIK	TEA MS	SLAC K	SKYP E	商務用 SKYPE	TELE GRAM	TWIL IO
Event.*	:white_lar ge_squar e:	:white_ch eck_mark:	:white_ch eck_mark:	:white_lar ge_squar e:									
Event.CreateConversation	:white_lar ge_squar e:												
Event.ContinueConversation	:white_lar ge_squar e:												

- 事件活動是 Direct Line (aka Web Chat) 中的一個擴充性機制。
- 同時擁有用戶端和伺服器的應用程式，可以選擇使用此事件活動透過服務來打開自己事件的通道。

Microsoft Teams

\	CORTANA	DIRECTLINE	DIRECTLINE (網路聊天)	電子郵件	FACEBOOK	GROUPE	KIK	TEAMS	SLACK	SKYPE	商務用SKYPE	TELEGRAM	TWILIO
Invoke.TeamsVerification	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:white_check_mark:	:x:	:x:	:x:	:x:	:x:
Invoke.ComposeResponse	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:white_check_mark:	:x:	:x:	:x:	:x:	:x:

- 除了許多其他類型的活動外，Microsoft Teams 還定義了一些特定叫用活動的小組。
- 叫用活動是專屬於應用程式，而不是用戶端定義的內容。
- 活動的「叫用」特定子類型，沒有通用概念。
- 叫用是 Bot 上目前唯一會觸發要求-回覆行為的活動。

這非常重要：若使用 OAuth 提示的對話方塊，則 Invoke.TeamsVerification 活動必須轉寄給對話方塊。

訊息更新

\	CORTANA	DIRECTLINE	DIRECTLINE (網路聊天)	電子郵件	FACEBOOK	GROUPE	KIK	TEAMS	SLACK	SKYPE	商務用SKYPE	TELEGRAM	TWILIO
MessageUpdate	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:white_check_mark:	:white_largے_quarے:	:x:	:x:	:x:	:x:
MessageDelete	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:white_check_mark:	:white_largے_quarے:	:x:	:x:	:x:	:x:

- 訊息更新目前由 Teams 提供支援。

OAuth

\	CORTANA	DIRECTLINE	DIRECTLINE (網路聊天)	電子郵件	FACEBOOK	GROUPE	KIK	TEAMS	SLACK	SKYPE	商務用SKYPE	TELEGRAM	TWILIO
EventTokenResponse	:white_lar_ge_s_quar_e:	:white_ch_eck_mark:	:white_ch_eck_mark:	:x:	:white_lar_ge_s_quar_e:	:white_lar_ge_s_quar_e:	:white_lar_ge_s_quar_e:	:x:	:white_lar_ge_s_quar_e:	:white_lar_ge_s_quar_e:	:white_lar_ge_s_quar_e:	:white_lar_ge_s_quar_e:	:white_lar_ge_s_quar_e:

這非常重要：若使用 OAuth 提示的對話方塊，則 Event.TokenResponse Activity 活動必須轉寄給對話方塊。

未分類

\	CORTANA	DIRECTLINE	DIRECTLINE (網路聊天)	電子郵件	FACEBOOK	GROUPE	KIK	TEAMS	SLACK	SKYPE	商務用SKYPE	TELEGRAM	TWILIO
EndOfConversation	:x:	:white_ch_eck_mark:	:white_ch_eck_mark:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:
InstallationUpdate	:x:	:white_ch_eck_mark:	:white_ch_eck_mark:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:
輸入	:x:	:white_ch_eck_mark:	:white_ch_eck_mark:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:
Han doff	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:

不使用 (包括付款特定叫用)

- DeleteUserData
- Invoke.PaymentRequest
- Invoke.Address
- Ping

每個通道支援活動的摘要

Cortana

- 訊息
- ConversationUpdate

- *Event.TokenResponse*
- *EndOfConversation* (在視窗關閉時?)

Direct Line

- 訊息
- ConversationUpdate
- Event.TokenResponse
- Event.*
- *Event.CreateConversation*
- *Event.ContinueConversation*

電子郵件

- 訊息

Facebook

- 訊息
- *Event.TokenResponse*

GroupMe

- 訊息
- ConversationUpdate
- *Event.TokenResponse*

Kik

- 訊息
- ConversationUpdate
- *Event.TokenResponse*

Teams

- 訊息
- ConversationUpdate
- MessageReaction
- MessageUpdate
- MessageDelete
- Invoke.TeamsVerification
- Invoke.ComposeResponse

Slack

- 訊息
- ConversationUpdate
- *Event.TokenResponse*

Skype

- 訊息
- ContactRelationUpdate
- *Event.TokenResponse*

商務用 Skype

- 訊息
- ContactRelationUpdate
- *Event.TokenResponse*

Telegram

- 訊息
- ConversationUpdate
- *Event.TokenResponse*

Twilio

- 訊息

所有通道的所有活動摘要表

\	CORTANA	DIRECTLINE	DIRECTLINE (網路聊天)	電子郵件	FACEBOOK	GROUPE	KIK	TEAMS	SLACK	SKYPE	商務用SKYPE	TELEGRAM	TWILIO
訊息	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:
Message Reaction	:	:	:	:	:	:	:	:white_e_ch_eck_mark:	:	:	:	:	:
ConversationUpdate	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:	:	:white_e_lar_ge_squar_e:	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:	:white_e_ch_eck_mark:	:	:	:white_e_ch_eck_mark:	:
ContactRelationUpdate	:	:	:	:	:	:	:	:	:	:white_e_ch_eck_mark:	:	:	:

\	COR TAN A	DIRE CT LINE	DIRE CT LINE (網路 聊天)	電子 郵件	FACE BOO K	GRO UPM E	KIK	TEA MS	SLAC K	SKYP E	商務 用 SKYP E	TELE GRA M	TWIL IO
End OfC onve rsati on	:x:	:whit e_ch eck_ mar k:	:whit e_ch eck_ mar k:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:
Insta llatio nUp date	:x:	:whit e_ch eck_ mar k:	:whit e_ch eck_ mar k:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:
輸入	:x:	:whit e_ch eck_ mar k:	:whit e_ch eck_ mar k:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:
Han doff	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:	:x:

網路聊天

網路聊天將傳送：

- "message":包含「文字」和/或「附件」
- "event":包含「名稱」和「值」(如 JSON/字串)
- "typing":如果使用者設定選項，亦即 "sendTypingIndicator" 網路聊天不會傳送 "contactRelationUpdate"。而網路聊天不支援 "messageReaction"，未明確要求我們支援這項功能。

根據預設，網路聊天會轉譯：

- "message":會根據活動中的選項轉譯為浮動切換或堆疊
- "typing":將轉譯 5 秒並隱藏，或直到下一個活動開始
- "conversationUpdate":會隱藏
- "event":會隱藏
- 其他：會顯示警告方塊 (我們從未在生產環境中看到)。您可以將此轉譯管線修改為新增、移除或取代任何自訂轉譯。

您可以使用網路聊天傳送任何活動類型和承載，我們不會以文件記錄也不會建議這項功能。您應該改為使用 "event" 活動。

Bot Framework 中的識別碼欄位

2019/5/10 • [Edit Online](#)

本指南說明 Bot Framework 中的識別碼欄位特性。

通道識別碼

每個 Bot Framework 通道都會有可供識別的唯一識別碼。

範例： `"channelId": "skype"`

通道識別碼可作為其他識別碼的命名空間。Bot Framework 通訊協定中的執行階段呼叫必須在通道的內容中進行；通道會為通訊時所使用的對話和帳戶識別碼提供意義。

依照慣例，所有通道識別碼都是小寫字母。通道會保證它們所發出的通道識別碼有一致的大小寫，因此，Bot 可以使用序數比較來確定相等性。

通道識別碼的規則

- 通道識別碼會區分大小寫。

Bot 控制代碼

每個已向 Azure Bot Service 註冊的 Bot 都有一個 Bot 控制代碼。

範例： `FooBot`

Bot 控制代碼代表 Bot 已向線上 Azure Bot Service 進行註冊。這項註冊會與 HTTP Webhook 端點以及通道註冊相關聯。

Azure Bot Service 可確保 Bot 控制代碼的唯一性。Azure 入口網站會執行不區分大小寫的唯一性檢查（亦即會將各種大小寫有所變化的 Bot 控制代碼視為單一控制代碼），不過，這是 Azure 入口網站的特性，而不一定是 Bot 控制代碼本身的特性。

Bot 控制代碼的規則

- Bot Framework 內的 Bot 控制代碼都是唯一的（不區分大小寫）。

應用程式識別碼

每個已向 Azure Bot Service 註冊的 Bot 都有一個應用程式識別碼。

NOTE

過去，應用程式通常稱為「MSA 應用程式」或「MSA/AAD 應用程式」。現在，應用程式更常簡稱為「應用程式」，但某些通訊協定元素可能會一直將應用程式稱為「MSA 應用程式」。

範例： `"msaAppId": "353826a6-4557-45f8-8d88-6aa0526b8f77"`

一個應用程式代表在身分識別小組應用程式入口網站中的一個註冊，並可作為 Bot Framework 執行階段通訊協定內的服務對服務身分識別機制。應用程式可能有其他非 Bot 關聯，例如網站和行動/傳統型應用程式。

每個已註冊的 Bot 只會有一個應用程式。雖然 Bot 擁有者無法獨立變更與其 Bot 相關聯的應用程式，但 Bot Framework 小組可以在少數例外狀況中這麼做。

Bot 和通道可能會使用應用程式識別碼來唯一識別已註冊的 Bot。

應用程式識別碼保證會是 GUID。在比較應用程式識別碼時，不應該區分大小寫。

應用程式識別碼的規則

- 在 Microsoft 應用程式平台內，應用程式識別碼是唯一的 (GUID 比較)。
- 每個 Bot 都只有一個對應的應用程式。
- 變更 Bot 所關聯的應用程式需要 Bot Framework 小組的協助。

通道帳戶

每個 Bot 和使用者在每個通道內都有一個帳戶。帳戶中會包含識別碼 (`id`) 和其他參考性 Bot 非結構化資料，例如選擇性的名稱。

範例： `"from": { "id": "john.doe@contoso.com", "name": "John Doe" }`

此帳戶會描述通道內的地址，訊息可在其中傳送和接收。在某些情況下，這些註冊存在於單一服務內（例如 Skype、Facebook）。在其他情況中，這些註冊會註冊於許多系統（電子郵件地址、電話號碼）。在更匿名的通道中（例如，網路聊天），註冊可能是暫時性的。

通道帳戶會以巢狀方式存在於通道內。例如，Facebook 帳戶只是數字。這個數字在其他通道中可能會有不同的意義，而在所有通道外則沒有意義。

通道帳戶和使用者（實際人員）之間的關聯性，取決於與每個通道相關聯的慣例。例如，簡訊號碼通常會指涉某個人員一段時間，之後該號碼可能會轉移給其他人。相反地，Facebook 帳戶通常會永久指涉某個人員，不過，兩個人共用 Facebook 帳戶的情況也不罕見。

在大部分通道中，您可以將通道帳戶視為一種可遞送訊息的信箱。大部分通道一般都允許將多個地址對應至單一信箱；例如，`jdoe@contoso.com` 和 `john.doe@service.contoso.com` 可能會解析成相同收件匣。某些通道會更進一步，根據對它進行存取的 Bot 來變更帳戶的地址；例如，Skype 及 Facebook 會改變使用者識別碼，讓每個 Bot 有不同的訊息收送地址。

雖然您可以在某些情況下，於地址之間確定相等性，但若要在信箱之間確定相等性以及在不同人之間確定相等性，則必須了解通道內的慣例，在許多情況下，您無法做到這一點。

透過傳送給 Bot 的活動上所存在的 `recipient` 欄位，Bot 會得知其通道帳戶地址。

通道帳戶的規則

- 通道帳戶在相關聯的通道內才會有意義。
- 多個識別碼可能會解析成相同帳戶。
- 可能會使用序數比較來確定兩個識別碼是否相同。
- 通常沒有比較可用來識別兩個不同的識別碼是否會解析成相同的帳戶、Bot 或人員。
- 識別碼、帳戶、信箱和人員之間的關聯性是否穩定，會取決於通道。

對話識別碼

訊息會在對話的內容中傳送及接收，而對話可由識別碼來識別。

範例： `"conversation": { "id": "1234" }`

對話會交換訊息和其他活動。每個對話都有零個以上的活動，而且每個活動只會出現在一個對話中。對話可能是永久的，也可能有不同的開始和結束。建立、修改或結束對話的程序發生於通道內（亦即，對話會在通道察覺時存在），而且這些程序的特性會由通道確定。

對話內的活動是由使用者和 Bot 傳送的。使用者「參與」對話的定義會隨通道而異，而且理論上，可能會包括目前的使用者、不斷收到訊息的使用者，以及傳送訊息的使用者。

數個通道 (例如, 簡訊、Skype 以及其他可能項目) 習慣將指派給 1 對 1 對話的對話識別碼當作是遠端通道帳戶識別碼。這個習慣有兩個副作用：

1. 對話識別碼是由檢視識別碼的人主觀認定的。如果參與者 A 和 B 正在談話, 參與者 A 所看到的對話識別碼是 "B", 參與者 B 所看到的對話識別碼是 "A"。
2. 如果 Bot 在此通道內有多個通道帳戶 (例如, 如果 Bot 有兩個簡訊號碼), 則對話識別碼並不足以唯一識別 Bot 觀點內的對話。

因此, 對話識別碼不一定能唯一識別通道內的單一對話, 即便只有單一 Bot 也是如此。

對話識別碼的規則

- 對話在相關聯的通道內才會有意義。
- 多個識別碼可能會解析成相同對話。
- 序數相等不一定能確定兩個對話識別碼是相同的對話, 不過大多數時候會是相同的。

活動識別碼

活動會在 Bot Framework 通訊協定內傳送及接收, 有時您可以識別這些活動。

範例： `"id": "5678"`

活動識別碼是選擇性的, 通道會採用活動識別碼讓 Bot 有辦法在後續 API 呼叫中參照識別碼 (如果可供使用)：

- 回覆特定活動
- 查詢活動層級的參與者清單

由於尚未再建立任何使用案例, 所以活動識別碼的處理方式沒有任何其他規則。

Application Insights 金鑰

2019/2/28 • [Edit Online](#)

Azure **Application Insights** 會在 Microsoft Azure「資源」中顯示您應用程式的相關資料。若要新增遙測到您的 Bot，您需要 Azure 訂用帳戶與為您的 Bot 建立的 Application Insights 資源。針對此資源，您可以取得三個金鑰來設定您的 Bot：

1. 檢測金鑰
2. 應用程式識別碼
3. API 金鑰

此主題將說明如何建立這些 Application Insights 金鑰。

NOTE

在建立 Bot 或在註冊程序期間，我們提供選項讓您開啟或關閉 **Application Insights**。若您已將它開啟，則您的 Bot 已經有所需的所有 Application Insights 金鑰。不過，若您已將它關閉，則您可以依照此主題中的指示協助您手動建立這些金鑰。

檢測金鑰

若要取得檢測金鑰，請執行下列動作：

1. 從 [Azure 入口網站](#) 的 [監視器] 區段下，建立新 **Application Insights** 資源（或使用現有資源）。

The screenshot shows the Azure portal's 'Monitor - Applications' section. On the left, the 'Monitor' option in the sidebar is highlighted with a red box. The main content area displays a list of applications under the 'Subscriptions: All 5 selected' heading. One application, 'ContosoReservationBot', is highlighted with a red box and has a small profile icon next to its name.

2. 從 Application Insights 資源清單中，按一下您剛建立的 Application Insight 資源。
3. 按一下 [概觀]。
4. 展開 [基本] 區塊並尋找 [檢測金鑰]。

The screenshot displays two views of the Azure Application Insights Overview page for the ContosoReservationBot. The left pane shows navigation links: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, and Investigate. The right pane has a search bar and a 'Show data for last' dropdown with options: 30 minutes, 1 hour, 6 hours, 12 hours, 1 day, 3 days, 7 days, and 30 days. Below the search bar are two cards: 'Failed requests' (100) and 'Server response time' (100ms). The second screenshot provides more detailed resource information: Resource group (ContosoReservationBot), Location (West US 2), Subscription ID (Bot Connector Internal), and Instrumentation Key (redacted). It also shows Tags (Click here to add tags).

- 複製檢測金鑰並將它貼到您 Bot 設定的 [Application Insights 檢測金鑰] 欄位。

應用程式識別碼

若要取得應用程式識別碼，請執行下列動作：

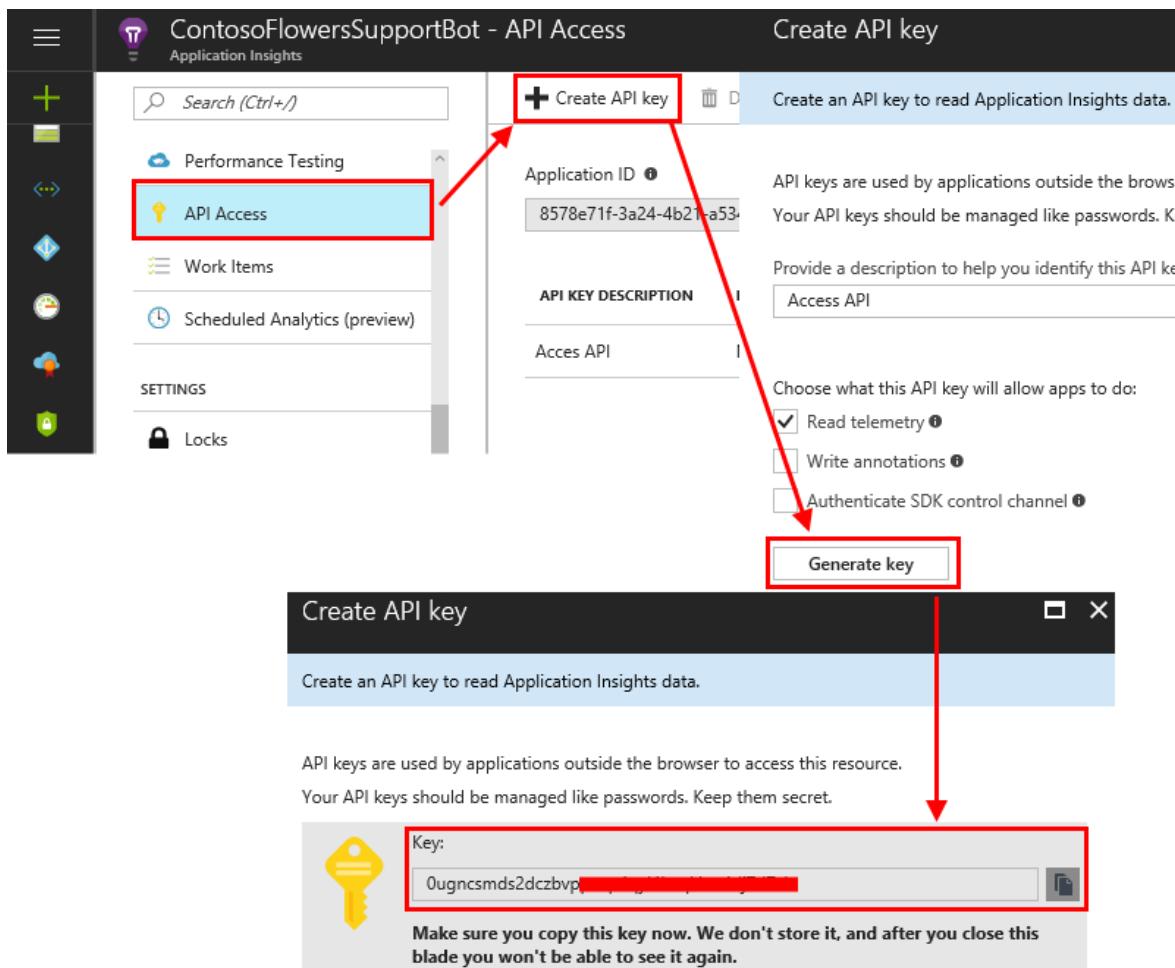
- 從您的 Application Insights 資源，按一下 [API 存取權]。
- 複製應用程式識別碼 並將它貼到您 Bot 設定的 [Application Insights 應用程式識別碼] 欄位。

The screenshot shows the Azure Application Insights API Access page for the ContosoFlowersSupportBot. On the left, there's a sidebar with icons for Performance Testing, API Access (which is highlighted with a red box), Work Items, Scheduled Analytics (preview), and SETTINGS. The main area has a search bar and buttons for 'Create API key', 'Delete API key', and 'Help'. A table lists an API key: Application ID (8578e71f-3a24-4b21-a534-dae2c3b4abb), API KEY DESCRIPTION (Access API), LAST USED (Never), CREATED ON (5/15/2017), and PERMIS (Read t...).

API 金鑰

若要取得 API 金鑰，請執行下列動作：

- 從 Application Insights 資源，按一下 [API 存取權]。
- 按一下 [建立 API 金鑰]。
- 輸入簡短說明，選取 [讀取遙測] 選項，並按一下 [產生金鑰] 按鈕。



WARNING

複製此 **API 金鑰** 並儲存它，因為我們再也不會向您顯示此金鑰。若遺失此金鑰，您必須重新建立。

4. 將 API 金鑰複製到您 Bot 設定的 [Application Insights API 金鑰] 欄位。

其他資源

如需有關如何將這些欄位連結到您 Bot 設定的詳細資訊，請參閱[啟用分析](#)。

Bot Framework User-Agent 要求

2019/2/28 • [Edit Online](#)

如果您正在閱讀這則訊息，您可能已從 Microsoft Bot Framework 服務收到要求。本指南會協助您了解這些要求的本質，並提供一些停止要求的步驟（如果您想要的話）。

如果您從我們的服務收到要求，其中可能會有類似下列字串的已格式化 User-Agent 標頭：

```
User-Agent: BF-DirectLine/3.0 (Microsoft-BotFramework/3.0 +https://botframework.com/ua)
```

這個字串最重要的部分是 **Microsoft-BotFramework** 識別碼，其可供 Microsoft Bot Framework 這個工具和服務集合用來讓獨立軟體開發人員建立及操作他們自己的 Bot。

如果您是 Bot 開發人員，您可能已經知道系統為什麼會將這些要求導向您的服務。如果您不是，請繼續閱讀以深入了解。

為什麼 Microsoft 會連絡我的服務？

Bot Framework 會將 Skype 及 Facebook Messenger 等聊天服務上的使用者連線到 Bot，此 Bot 是 Web 伺服器，且其中會有 REST API 在可由網際網路存取的端點上執行。Bot 的 HTTP 呼叫（也稱為 Webhook 呼叫）只會傳送到 Bot 開發人員（已向 Bot Framework 開發人員入口網站註冊）所指定的 URL。

如果您收到從 Bot Framework 服務到 Web 服務的不請自來要求，可能是因為開發人員不小心或故意輸入您的 URL 作為其 Bot 的 Webhook 回呼。

停止這些要求

如需協助阻止不要求的服務，使其無法到達您的 Web 服務，請連絡 bf-reports@microsoft.com。

Bot 檢閱指導方針

2019/3/4 • [Edit Online](#)

我們歡迎您，並感謝您投入您的才華和時間為 Microsoft 通道建置 Bot、Botlet、Web 應用程式、增益集或技能（「應用程式整合」）。下列是您的應用程式整合在發佈到 Microsoft 通道（例如 Skype 或 Microsoft Teams）之前必須滿足的最低需求。除了下列詳細說明的需求之外，每個通道可能還有特定需求。如果適用的話，您可以在每個通道的設定頁面上找到通道的特定條款，而且您可能需要先註冊通道的服務，才能將 Bot 發佈至該通道。

應用程式整合原則

1. 值、表示法、安全性和可用性。

您的應用程式整合和其相關聯的中繼資料必須：

- 擁有多樣且更具資訊性的中繼資料，必須提供有價值和高品質的使用者體驗；
- 精確且清楚地反映應用程式整合的來源、功能和特性，並描述任何重要的限制；
- 不使用與其他應用程式類似的名稱或圖示，如果您沒有權限建立該表示，則不得聲稱來自公司、政府機構或其他實體；
- 不會危害或損害使用者安全性或 Microsoft 通道的安全性或功能；
- 不會試圖變更或擴充違反這些原則或適用的 Microsoft 通道條款中所述的功能；
- 不得包含或啟用惡意程式碼；
- 可測試；
- 繼續執行並保持對使用者輸入的回應；
- 包含您的服務條款的有效連結；
- 依照其描述、設定檔、使用和隱私權原則中的描述進行操作；
- 根據適用於 Microsoft Bot Framework 的條款（或適用於其開發的其他條款）和 Microsoft 通道條款（或發佈您的應用程式整合之通道的其他適用條款）執行；
- 如果您的應用程式整合包含人為互動（例如客戶服務或現場人員支援），請通知使用者；
- 為其支援的所有語言進行當地語系化。應用程式整合的描述文字，必須以您聲明的每種語言進行當地語系化。

2. 隱私權

- 如果您的應用程式整合處理使用者的個人資訊（個人資訊包括識別或可用來識別某人的所有資訊或資料，或與這類資訊或資料相關聯的所有資訊或資料。個人資訊的範例包括：姓名和地址、電話號碼、生物特徵辨識的識別項、位置、聯絡人、相片、音訊與視訊錄製、文件、簡訊、電子郵件或其他文字通訊、螢幕擷取畫面，以及在某些情況下的合併瀏覽歷程記錄），您必須提供從您的應用程式整合到適用之隱私權原則的重要連結，此類隱私權原則必須遵守所有適用的法律、法規與原則需求。此原則應涵蓋您正在收集或傳送的資料、您將對該資料進行的作業以及（如果適用）您將與之共享的人員。如果您沒有隱私權聲明，以下是一些可能有所幫助的第三方資源*：

Future of Privacy 論壇 - [應用程式隱私權原則產生器](#)

Iubenda - [隱私權原則產生器](#)

* 貴用戶同意承擔因使用這些第三方資源所產生之所有風險和責任，並且對您因使用這些資源而引起的任何問題，Microsoft 概不負責。

- 未經使用者明確同意，您的應用程式整合不得收集、儲存或傳輸與其主要用途不相關的個人資訊。您必須取得使用者的所有同意，以便在法律要求的情況下處理個人資訊。
- 未經 Microsoft 明確許可，您不得發佈針對 13 歲以下兒童的應用程式整合（如「兒童網路隱私保護法」中所定義）。

3. 財務交易

- 針對已啟用付款的應用程式整合：
 - 您的應用程式整合可能無法透過使用者介面傳送財務檢測詳細資料；
 - 根據任何通道或第三方平台限制，您的應用程式整合可能：(a) 支援透過 [Microsoft 賣方中心](#) 付款，但須遵守 Microsoft 賣方中心合約條款；或 (b) 傳輸連結至其他安全的付款服務；
 - 如果您的應用程式整合支援前述的付款機制，則必須在使用者同意使用您的應用程式整合之前，在您的應用程式整合條款和隱私權原則（以及應用程式整合的任何設定檔頁面或網站）中揭露此資訊；
 - 您必須在其設定檔中明確指出應用程式整合是否已啟用付款，並向使用者提供商家的客戶服務詳細資料；以及
 - 未經 Microsoft 明確許可，您不得在任何包括連結或以其他方式將使用者導向至購買數位貨品之付款服務的 Microsoft 通道上發佈應用程式整合。

4.內容

- 您的應用程式整合和相關中繼資料中的所有內容，必須是原先發行者建立，由協力廠商權限持有者適當授權，在權限持有者允許的情況下使用，或者在法律所允許的情況下使用。
- 您不會在應用程式整合中發佈下列應用程式整合或內容：
 - 不合法；
 - 利用、傷害或威脅傷害兒童；
 - 包含廣告、垃圾郵件，垃圾或來路不明或大量通信、文章或訊息；
 - 可能會被視為不當或具冒犯意味的內容（例如，裸露、人獸交、褻瀆、色情圖片或成人內容、圖形或無端暴力、菸草產品、犯罪、危險或不負責任的活動、毒品、侵犯人權、製造武器或非法對某人或動物使用武器、濫用酒精產品）；
 - 虛假或具誤導性（例如，詐騙索取金錢，冒充他人身分）；
 - 對您、Microsoft 通道或其他人有害或造成安全風險（例如，傳播病毒、跟蹤、張貼恐怖主義內容、傳播仇恨言論，或針對種族、族裔、國籍、語言、性別、年齡、殘疾、宗教、性取向、退伍軍人身份或任何其他社會團體的成員資格考量，提倡歧視、仇恨或暴力）；
 - 侵犯他人的權利（例如，未經授權分享受著作權保護的音樂或其他受著作權保護的資料）；
 - 謗謗、中傷、詆毀或威脅；
 - 侵犯他人隱私；
 - 處理下列資訊：(a) 與病患的病情、治療或治療費用有關；(b) 將個人識別為患者、健康計劃成員或受益人或與之溝通；或者 (c) 根據「健康保險可攜性及責任法案」（經修訂的 "HIPAA"）為「受保護的健康資訊」，或者執行受 HIPAA 管轄的任何活動（如果您是 HIPAA 下定義的「受保實體」或「商業夥伴」）；
 - 在您的應用程式所針對的任何國家/地區都具有攻擊性。根據當地法律或文化規範，特定國家/地區的內容可能被視為具有攻擊性；
 - 協助他人打破這些規則。
- 如果您對應用程式整合進行任何重大變更，則必須透過向已發佈的特定通道傳送電子郵件來事先通知 Microsoft。對 Bot 註冊進行變更可能使得我們需要重新檢閱 Bot，以確保其繼續符合此處所述的需求。Microsoft 有權自行斟酌，間歇性地檢閱任何通道上的應用程式整合，並在不另行通知的情況下移除。

Bot 服務合規性

2019/2/28 • [Edit Online](#)

NOTE

本主題適用於最新版的 SDK (v4)。您可以在[這裡](#)找到舊版 SDK (v3) 的內容。

Azure Bot 服務符合 ISO 27001:2013、ISO 27019:2014、SOC 1 和 2、支付卡產業資料安全標準 (PCI DSS) 及健康保險流通與責任法案商業夥伴合約 (HIPAA BAA) 規範

Azure Bot 服務符合 ISO 27001:2013 和 ISO 27018:2014 規範

Azure Bot 服務已成功完成 ISO 27001:2013 和 ISO 27018:2014 稽核，稽核報告中沒有任何不符合規範的項目（結果）。此外，我們也已取得「CSA STAR 認證」，獲得成熟度能力評鑑最高可能金級獎。Azure 是第一個獲得此認證的主要公用雲端服務提供者。如需詳細資料，您可以在[信任中心 ISO](#) 頁面上所參照的 Azure 主要[合規性概觀文件](#)中，找到包含在已更新範圍聲明中的 Azure Bot 服務。

Azure Bot 服務符合 PCI DSS 規範

Azure Bot 服務已成功符合支付卡產業資料安全標準 (PCI DSS) 規範！QSA (稽核員) 所核發之合規性證明 (AoC) 和合規性報告 (RoC) 中的評估結果。合規性的有效期限自通過稽核並收到評估機構的 AoC 開始，至 AoC 簽署日期 (2017 年 12 月 22 日) 後一年結束。如需詳細資料，您可以在[信任中心 PCI](#) 頁面上所參照的 Azure 主要[合規性概觀文件](#)中，找到包含在已更新範圍聲明中的 Azure Bot 服務。您可在[服務信任入口網站](#)中找到憑證。

Azure Bot 服務現已納入 Microsoft HIPAA BAA 範圍

Azure Bot 服務現已納入 Microsoft 的健康保險流通與責任法案商業夥伴合約 (HIPAA BAA) 範圍！如需詳細資料，您可以在[信任中心 HIPAA](#) 頁面上所參照的 Azure 主要[合規性概觀文件](#)中，找到包含在已更新範圍聲明中的 Azure Bot 服務。

Azure Bot 服務符合 SOC 1 和 SOC 2 規範

Azure Bot 服務已順利完成 SOC 1 和 2 的稽核。Microsoft 雲端服務的稽核涵蓋資料安全性控制、可用性、處理完整性和機密性，適用於每項服務的範圍內信任原則。如需詳細資料，您可以在[信任中心 SOC](#) 頁面上所參照的 Azure 主要[合規性概觀文件](#)中，找到包含在已更新範圍聲明中的 Azure Bot 服務。

對一般問題進行疑難排解

2019/5/10 • [Edit Online](#)

這些常見問答集可協助您對常見的 Bot 開發或作業問題進行疑難排解。

如何對 Bot 的問題進行疑難排解？

1. 使用 [Visual Studio Code](#) 或 [Visual Studio](#) 針對 Bot 的原始程式碼進行偵錯。
2. 在將 Bot 部署至雲端之前，使用[模擬器](#)來測試它。
3. 將 Bot 部署至如 Azure 的雲端主機平台，然後在 [Azure 入口網站](#) 中使用 Bot 儀表板上的內建網路聊天控制項來測試 Bot 的連線能力。如果您在將 Bot 部署至 Azure 後遇到問題，則可以考慮使用這篇部落格文章：[了解 Azure 疑難排解和支援](#)。
4. 排除問題是由[驗證](#)所導致的可能性。
5. 在 Skype 上測試您的 Bot。這將能協助您驗證端對端的使用者體驗。
6. 請考慮在具有額外驗證需求的通道（例如直接線路或網路聊天）上測試 Bot。
7. 請檢閱[偵錯 Bot](#)操作說明，以及該區段中的其他偵錯文章。

如何對驗證問題進行疑難排解？

如需有關對 Bot 驗證問題進行疑難排解的詳細資料，請參閱[針對 Bot Framework 驗證進行疑難排解](#)。

我使用適用於 .NET 的 Bot Framework SDK。如何對 Bot 的問題進行疑難排解？

尋找例外狀況。

在 Visual Studio 2017 中，移至 [偵錯] > [Windows] > [例外狀況設定]。在 [例外狀況設定] 視窗中，選取位於 [Common Language Runtime 例外狀況] 旁邊的 [當擲回時中斷] 核取方塊。當有擲回或未處理的例外狀況時，您也可以查看 [輸出] 視窗中的診斷輸出。

查看呼叫堆疊。

在 Visual Studio 中，您可以選擇是否要對 [Just My Code](#) 進行偵錯。檢查完整的呼叫堆疊可能可以針對問題提供額外的見解。

確保所有對話方法都會以能處理下一個訊息的方案作為結尾。

所有對話方塊步驟都必須饋送到瀑布的下一個步驟，或結束目前的對話方塊才能脫離堆疊。如果未正確處理步驟，如您所預期，將不會繼續對話。查看[對話方塊](#)概念文章，更進一步了解對話方塊。

為何輸入活動不會執行任何動作？

某些通道在其用戶端中並不支援暫時性輸入更新。

SDK 中的連接器程式庫和建立器程式庫有何差異？

連接器程式庫為 REST API 的說明。建立器程式庫能加入交談式對話程式設計模型及其他功能，例如通知、瀑布圖、鏈結，以及引導式表單完成功能。建立器程式庫也能提供如 LUIS 等認知服務的存取。

造成具有 HTTP 狀態碼 429「太多要求」之錯誤的原因為何？

具有 HTTP 狀態碼 429 的錯誤回應，表示在特定期間內發出太多要求。回應的主體應該會包含問題的解釋，也可能

會指定要求之間所需的最低間隔。此錯誤其中一個可能的來源為 [ngrok](#) (英文)。如果您是使用免費方案並達到 ngrok 的限制，請移至其網站的定價與限制頁面以取得更多[選項](#) (英文)。

我的 Bot 訊息為何不是由使用者接收？

回應中產生的訊息活動必須正確地定址，否則無法送達其目的地。在大部分情況下，您不需要明確處理此事，SDK 會負責為您將訊息活動定址。

將活動正確地定址，表示包含適當的「對話參考」詳細資料，以及傳送者和接收者的相關詳細資料。在大部分情況下，訊息活動會以回應形式傳送至已送達的其中一個活動。因此，可以從輸入活動取得定址詳細資料。

如果您檢查追蹤或稽核記錄，您可以檢查以確保您的訊息正確地定址。若非如此，請在 Bot 中設定中斷點，並查看在何處針對您的訊息設定識別碼。

如何在 ASP.NET 中執行背景工作？

在某些情況下，您可能會想要起始非同步工作以在幾秒後執行某些程式碼，來清除使用者設定檔或重設交談/對話狀態。如需達成此目的的詳細資料，請參閱[如何在 ASP.NET 中執行背景工作](#) (英文)。請特別考慮使用 [HostingEnvironment.QueueBackgroundWorkItem](#) (機器翻譯)。

使用者訊息和 HTTPS 方法呼叫有何關聯？

當使用者透過通道傳送訊息時，Bot Framework Web 服務將會對 Bot 的 Web 服務端點發出 HTTPS POST。該 Bot 會透過針對其所傳送的每個訊息向 Bot Framework 發出個別的 HTTPS POST，於該通道上回傳零個、一個或多個訊息給使用者。

Bot 對其所接收到的第一則訊息的回應速度很慢。要如何讓它快一點？

Bot 為 Web 服務，而某些主機平台 (包括 Azure) 在該服務於一段時間內沒有接收到流量時，便會自動使該服務進入睡眠狀態。如果您的 Bot 發生此情況，它於下次接收到訊息時便必須從頭重新啟動，這會讓它的回應速度比起已在執行情況下的速度還要慢上許多。

某些主機平台會允許您設定服務，使它不會進入睡眠狀態。如果要在 Azure 中執行此步驟，請在 [Azure 入口網站](#) 中瀏覽至 Bot 的服務，選取 [應用程式設定]，然後選取 [一律開啟]。此選項可供大部分 (但非全部) 的服務方案使用。

如何保證訊息傳遞順序？

Bot Framework 將會盡可能地保留訊息排序。例如，如果您先傳送訊息 A 並等候該 HTTP 作業完成，然後才起始另一個 HTTP 作業以傳送訊息 B，則 Bot Framework 將會自動了解訊息 A 應該排在訊息 B 之前。不過，一般而言 Bot Framework 並無法保證訊息傳遞順序，因為通道是最終負責傳遞訊息的角色，並可能會對訊息進行重新排序。如果要降低訊息以錯誤順序傳遞的風險，您可以考慮在訊息之間實作時間延遲。

如何攔截使用者與我的 Bot 之間的所有訊息？

使用適用於 .NET 的 Bot Framework SDK 時，您可以對 `Autofac` 相依性插入容器提供 `IPostToBot` 和 `IBotToUser` 介面的實作。使用適用於任何語言的 Bot Framework SDK 時，您可以針對大致相同的目的使用中介軟體。[BotBuilder-Azure](#) (英文) 存放庫包含 C# 和 Node.js 程式庫，並能將此資料記錄至 Azure 資料表。

我的訊息文字為何有一部分被卸除？

Bot Framework 和許多通道都會把文字當成是以 [Markdown](#) 進行格式化。請檢查您的文字是否包含可能會被解譯為 Markdown 語法的字元。

如何在同一個 Bot 服務端點支援多個 Bot？

此範例 (英文) 示範如何以正確的 `MicrosoftAppCredentials` 設定 `Conversation.Container`，並使用簡單的 `MultiCredentialProvider` 來驗證多個應用程式識別碼與密碼。

識別項

識別碼在 Bot Framework 中的運作方式為何？

如需有關 Bot Framework 中識別碼的詳細資料，請參閱[針對識別碼的 Bot Framework 指南](#)。

如何存取使用者識別碼？

SMS 和電子郵件訊息將會在 `from.Id` 屬性中提供未經處理的使用者識別碼。在 Skype 訊息中，`from.Id` 屬性將會包含適用於該使用者的唯一識別碼，其有別於該使用者的 Skype 識別碼。如果您需要連線至現有帳戶，您可以使用登入卡並實作自己的 OAuth 流程，以將使用者識別碼連線至您自己服務的使用者識別碼。

為何已不再顯示我的 Facebook 使用者名稱？

您是否有變更您的 Facebook 密碼？這麼做將會使存取權杖無效，且您必須在 [Azure 入口網站](#) 中針對 Facebook Messenger 通道更新您 Bot 的組態設定。

為何我的 Kik Bot 會回覆 "I'm sorry, I can't talk right now" (抱歉，我現在無法交談)？

在 Kik 上開發的 Bot 僅允許有 50 個訂閱者。在有 50 個唯一使用者與您的 Bot 互動之後，任何嘗試與 Bot 聊天的新使用者，都會接收到訊息 "I'm sorry, I can't talk right now" (抱歉，我現在無法交談)。如需詳細資訊，請參閱[Kik 文件 \(英文\)](#)。

如何從 Bot 使用已驗證的服務？

如需 Azure Active Directory 驗證，請參閱新增驗證 [V3](#) | [V4](#)。

NOTE

如果您有將驗證和安全性功能加入您的 Bot，您應該確保自己在程式碼中所實作的模式會符合適用於您應用程式的安全性標準。

如何限制只有預先決定清單中的使用者才能存取我的 Bot？

某些通道 (例如 SMS 和電子郵件) 會提供不限範圍的位址。在這些情況下，來自使用者的訊息將會在 `from.Id` 屬性中包含未經處理的使用者識別碼。

其他通道 (例如 Skype、Facebook 和 Slack) 會提供有範圍或承租的位址，這會使 Bot 無法預估使用者的識別碼。在這些情況下，您將會需要透過登入連結或共用祕密來驗證使用者，以判斷該使用者是否有使用 Bot 的授權。

為何我的直接線路 1.1 交談會在每則訊息之後重新開始？

如果您的直接線路交談會在每則訊息之後重新開始，便代表直接線路傳送至 Bot 之訊息中的 `from` 屬性很可能已經遺失或為 `null`。當直接線路用戶端所傳送之訊息中的 `from` 屬性遺失或為 `null` 時，直接線路服務會自動配置識別碼，這會使用戶端所傳送的每則訊息都會像是來自全新且不同的使用者。

如果要修正此問題，請將直接線路用戶端所傳送之每則訊息中的 `from` 屬性，設定為能唯一代表傳送該訊息之使用者的穩定值。例如，如果使用者已經登入網頁或應用程式，您便可以將該使用者的現有識別碼，作為其所傳送之訊息中 `from` 屬性的值。或者，您可以選擇在頁面載入或應用程式載入時產生隨機的使用者識別碼，將該識別碼儲存

於 Cookie 或裝置狀態中，然後將該識別碼作為使用者所傳送之訊息中 `from` 屬性的值。

造成直接線路 3.0 服務回應 HTTP 狀態碼 502「不正確的閘道」的原因為何？

直接線路 3.0 會在嘗試連絡 Bot 但無法成功完成要求的情況下，傳回 HTTP 狀態碼 502。此錯誤表示 Bot 傳回錯誤或要求逾時。如需 Bot 會產生的錯誤詳細資訊，請在 Azure Portal 內移至 Bot 的儀表板，然後按一下 [問題] 連結以了解受影響的通道。如果您已經針對 Bot 設定 Application Insights，便也可以在那裡找到詳細的錯誤資訊。

適用於 .NET 的 Bot Framework SDK 中的 `IDialogStack.Forward` 方法是什麼？

`IDialogStack.Forward` 的主要目的是重複使用經常為「被動」的現有子對話；在此情況下，(`IDialog.StartAsync` 中的)子對話會針對某些 `ResumeAfter` 處理常式等候物件 `T`。特別是在具有會等候 `IMessageActivity` `T` 之子對話的情況下，您可以使用 `IDialogStack.Forward` 方法來轉送傳入的 `IMessageActivity` (已由某些父對話接收)。例如，若要將傳入的 `IMessageActivity` 轉送至 `LuisDialog`，請呼叫 `IDialogStack.Forward` 以將 `LuisDialog` 推送至對話堆疊上，在 `LuisDialog.StartAsync` 中執行程式碼直到其針對下則訊息做出等候的排程為止，然後立即透過轉送的 `IMessageActivity` 來滿足該等候。

通常是 `IMessageActivity`，因為 `IDialog.StartAsync` 通常會建構為等候該類型的活動。您可以針對 `LuisDialog` 使用 `IDialogStack.Forward` 作為攔截來自使用者之訊息的機制，以在將該訊息轉送至現有 `LuisDialog` 之前進行某些處理。或者，您也可以針對該目的搭配 `ContinueToNextGroup` 使用 `DispatchDialog`。

您可以預期在由 `StartAsync` 所排程的第一個 `ResumeAfter` 處理常式 (例如 `LuisDialog.MessageReceived`) 中找到已轉送的項目。

「主動」和「被動」之間的差異為何？

從您 Bot 的觀點來看，「被動」表示交談是由使用者傳送訊息給 Bot 來起始，而 Bot 則是透過回應該訊息來反應。相反地，「主動」表示交談是由 Bot 傳送第一則訊息給使用者來起始。例如，Bot 可能會傳送主動式訊息以通知使用者計時器已到期，或是已發生某個事件。

如何傳送主動式訊息給使用者？

如需如何傳送主動式訊息的範例，請參閱 GitHub 上 BotBuilder-Samples 存放庫內的 [C# V4 範例](#) 和 [Node.js V4 範例](#)。

如何參考 C# 對話中不可序列化的服務？

有數個選項：

- 透過 `Autofac` 和 `FiberModule.Key_DoNotSerialize` 解析相依性。這是最乾淨的解決方案。
- 使用 `NonSerialized` (機器翻譯) 和 `OnDeserialized` (機器翻譯) 屬性來還原針對還原序列化的相依性。這是最簡單的解決方案。
- 不儲存相依性，以使它不會被序列化。此解決方案就技術上而言雖然可行，但並不建議。
- 使用反映序列化代理。此解決方案在某些情況下可能不可行，並會有過度序列化的風險。

交談狀態的儲存位置為何？

使用者、交談及私人交談屬性包中的資料，會使用連接器的 `IBotState` 介面進行儲存。每個屬性包都會依 Bot 的識別碼設定範圍。使用者屬性包是依使用者識別碼進行索引、交談屬性包是由交談識別碼進行索引，而私人交談屬性包則是由使用者識別碼和交談識別碼進行索引。

如果您是使用適用於 .NET 的 Bot Framework SDK 或適用於 Node.js 的 Bot Framework SDK 來建置 Bot, 對話堆疊和對話資料都將會自動儲存為私人交談屬性包中的項目。C# 實作會使用二進位序列化, 而 Node.js 實作則會使用 JSON 序列化。

`IBotState` REST 介面是由兩個服務所實作。

- Bot Framework 連接器會提供能實作此介面並將資料儲存於 Azure 中的雲端服務。此資料會進行待用加密, 且不會刻意到期。
- Bot Framework 模擬器會提供此介面的記憶體內部實作, 以對 Bot 進行偵錯。此資料會在模擬器處理序結束時到期。

如果您想要將此資料儲存在資料中心內, 則可以提供狀態服務的自訂實作。這可以透過至少兩種方式來完成:

- 使用 REST 層來提供自訂 `IBotState` 服務。
- 在語言 (Node.js 或 C#) 層中使用建立器介面。

IMPORTANT

建議您不要將 Bot Framework 狀態服務 API 用於生產環境, 未來版本可能會將它淘汰。建議更新您的 Bot 程式碼以使用記憶體內部儲存體進行測試, 或將其中一個 Azure 擴充功能用於生產環境 Bot。如需詳細資訊, 請參閱適用於 .NET 或 Node 實作的管理狀態資料主題。

什麼是 ETag？它與 Bot 資料包儲存體之間的關聯為何？

ETag 為適用於 [開放式並行存取控制](#)的機制。Bot 資料包儲存體會使用 ETag 來防止資料更新衝突。具有 HTTP 狀態碼 412「前置條件失敗」的 ETag 錯誤, 表示針對該 Bot 資料包有多個並行執行的「讀取-修改-寫入」序列。

對話堆疊和狀態會儲存到 Bot 資料包中。例如, 在 Bot 仍在處理前一則訊息時便接收到新交談訊息的情況下, 您可能就會看見「前置條件失敗」ETag 錯誤。

造成具有 HTTP 狀態碼 412「前置條件失敗」或 HTTP 狀態碼 409「衝突」之錯誤的原因為何？

連接器的 `IBotState` 服務是用來儲存 Bot 資料包 (也就是使用者、交談及私人 Bot 資料包, 其中私人 Bot 資料包會包含「控制流程」狀態)。`IBotState` 服務中的並行控制是透過 ETag 由開放式並行存取進行管理。如果在「讀取-修改-寫入」序列期間發生更新衝突 (基於針對單一 Bot 資料包的並行更新), 則:

- 如果有保留 ETag, `IBotState` 服務會擲回具有 HTTP 狀態碼 412「前置條件失敗」的錯誤。這是適用於 .NET 的 Bot Framework SDK 中的預設行為。
- 如果未保留 ETag (也就是已將 ETag 設定為 `*`), 則「最後寫入者優先」的原則將會生效, 並會在具有資料遺失風險的情況下防止發生「前置條件失敗」錯誤。這是適用於 Node.js 的 Bot Framework SDK 中的預設行為。

如何修正「前置條件失敗」(412) 或「衝突」(409) 錯誤？

這些錯誤表示您的 Bot 針對相同的交談同時處理多則訊息。如果您的 Bot 已連線至要求精準排序之訊息的服務, 您應該考慮鎖定交談狀態, 以確保訊息不會以平行方式處理。

適用於 .NET 的 Bot Framework SDK 會提供機制 (會實作 `IScope` 的類別 `LocalMutualExclusion`) 以搭配記憶體內部旗號, 來以保守模式序列化單一交談的處理。您可以延伸此實作, 以使用由交談位址設定範圍的 Redis 租用。

如果您的 Bot 未連線至外部服務, 或服務可接受以平行方式處理來自相同交談的訊息, 您便可以加入此程式碼以忽略發生在 Bot 狀態 API 中的任何衝突。這將會允許最後的回覆設定交談狀態。

```
var builder = new ContainerBuilder();
builder
    .Register(c => new CachingBotDataStore(c.Resolve<ConnectorStore>(),
CachingBotDataStoreConsistencyPolicy.LastWriteWins))
    .As<IBotDataStore<BotData>>()
    .AsSelf()
    .InstancePerLifetimeScope();
builder.Update(Conversation.Container);
```

如何設定透過狀態 API 所儲存之 Bot 資料的版本？

IMPORTANT

建議您不要將 Bot Framework 狀態服務 API 用於生產環境或 v4 Bot，因為未來版本可能會完全淘汰。建議更新您的 Bot 程式碼以使用記憶體內部儲存體進行測試，或將其中一個 Azure 擴充功能用於生產環境 Bot。如需詳細資訊，請參閱[管理狀態資料](#)主題。

狀態服務可讓您保存交談中的對話進度，讓使用者於稍後繼續與 Bot 交談時不會失去原先的位置。若要保留此進度，透過狀態 API 所儲存的 Bot 資料屬性包將不會於您修改 Bot 的程式碼時自動清除。您應該根據已修改的程式碼是否能與舊版資料相容，來決定是否要清除 Bot 資料。

- 如果您想要在開發 Bot 的期間手動重設交談的對話堆疊及狀態，則可以使用 `/deleteprofile` 命令來刪除狀態資料。請務必包含此命令的前置空格，以防止通道解譯它。
- 在您的 Bot 已部署至生產環境之後，您便可以對 Bot 資料進行版本設定，使系統會在您提升版本時清除相關聯的資料。透過適用於 Node.js 的 Bot Framework SDK，可以使用中介軟體來達成此目的；透過適用於 .NET 的 Bot Framework SDK，則可以使用 `IPostToBot` 實作來達成此目的。

NOTE

如果對話堆疊因序列化格式變更或程式碼大幅變更而無法正確還原序列化，交談狀態將會重設。

LUIS 內建日期、時間、持續時間及設定實體有哪些可能的電腦可讀取解決方法？

如需範例清單，請參閱 LUIS 文件的[預先建置實體](#)小節。

如何使用超過最大值的 LUIS 意圖？

您可以考慮分割您的模型，並以序列或平行方式呼叫 LUIS 服務。

如何使用多個 LUIS 模型？

適用於 Node.js 的 Bot Framework SDK 和適用於 .NET 的 Bot Framework SDK 皆支援從單一 LUIS 意圖對話呼叫多個 LUIS 模型。請記住下列注意事項：

- 使用多個 LUIS 模型時，系統會假設 LUIS 模型具有未重疊的意圖集合。
- 使用多個 LUIS 模型時，系統會假設不同模型的分數是可以互相比較的，以從多個模型中選出「最相符的意圖」。
- 使用多個 LUIS 模型代表當某個意圖符合其中一個模型時，它也會強烈符合其他模型的「無」意圖。在此情況下，您可以避免選取「無」意圖；適用於 Node.js 的 Bot Framework SDK 將會自動相應減少「無」意圖的分數以避免此問題。

我可以在哪裡取得更多有關 LUIS 的協助？

- [Language Understanding \(LUIS\) 簡介 - Microsoft 認知服務 \(英文\) \(影片\)](#)
- [適用於 Language Understanding \(LUIS\) 的進階學習課程 \(英文\) \(影片\)](#)
- [LUIS 文件](#)
- [Language Understanding 論壇](#)

有哪些由社群撰寫的對話？

- [BotAuth \(英文\) - Azure Active Directory 驗證](#)
- [BestMatchDialog \(英文\) - 以規則運算式為基礎的使用者文字至對話分派方法](#)

有哪些由社群撰寫的範本？

- [ES6 BotBuilder \(英文\) - ES6 Bot 建立器範本](#)

為何在建立 Bot 時會收到 Authorization_RequestDenied 例外狀況？

建立 Azure Bot 服務 Bot 的權限是透過 Azure Active Directory (AAD) 入口網站來管理。如果未在 [AAD 入口網站](#) 中正確設定權限，使用者在嘗試建立 Bot 服務時將會收到 **Authorization_RequestDenied** 例外狀況。

請先檢查您是否為該目錄的「來賓」：

1. 登入 [Azure 入口網站](#)。
2. 選取 [所有服務]，並搜尋 *active*。
3. 選取 **Azure Active Directory**。
4. 按一下 [使用者]。
5. 從清單上找到該使用者，並確定 [使用者類型] 不是 [來賓]。

NAME	USER NAME	USER TYPE
C	[REDACTED]	Member
R	[REDACTED]	Owner
Globe	[REDACTED]	Guest
Q	[REDACTED]	Member
Globe	[REDACTED]	Guest

在您確定自己不是來賓之後，若要確保有效目錄內的使用者可以建立 Bot 服務，目錄管理員必須設定下列設定：

1. 登入 [AAD 入口網站](#)。移至 [使用者和群組] 並選取 [使用者設定]。
2. 在 [應用程式註冊] 區段底下，將 [使用者可以註冊應用程式] 設定為 [是]。這可讓您目錄中的使用者建立 Bot 服務。
3. 在 [外部使用者] 區段底下，將 [來賓使用者權限受限] 設定為 [否]。這可讓您目錄中的來賓使用者建立 Bot 服務。

The screenshot shows the Azure Active Directory admin center with the 'User settings' page selected. The left sidebar lists various management options like 'All users', 'All groups', and 'User settings'. The main content area is titled 'User settings' and contains several configuration sections with 'Yes' or 'No' toggle buttons. Some sections are highlighted with red boxes.

Section	Setting	Status
Enterprise applications	Users can consent to apps accessing company data on their behalf	Yes
	Users can add gallery apps to their Access Panel	Yes
	Users can only see Office 365 apps in the Office 365 portal	No
App registrations	Users can register applications	Yes (highlighted)
	Guest users permissions are limited	Yes (highlighted)
	Admins and users in the guest inviter role can invite	Yes
External users	Members can invite	Yes
	Guests can invite	No
	Restrict access to Azure AD administration portal	No
Administration portal	Save	Discard

為何我無法移轉 Bot？

如果您無法移轉 Bot，這可能是因為該 Bot 所屬的目錄不是您的預設目錄。請嘗試這些步驟：

- 在目標目錄中，新增不是預設目錄成員的新使用者（透過電子郵件地址），將要移轉之目標訂用帳戶上的參與者角色授與該使用者。
- 在 [Dev Portal](#) (英文) 中，將該使用者的電子郵件地址新增為要移轉之 Bot 的共同擁有者。然後登出。
- 以新使用者的身分登入 [Dev Portal](#) (英文)，然後繼續移轉該 Bot。

我可以在哪裡取得更多協助？

- 在 [Stack Overflow](#) (英文) 上利用先前已解答之問題中的資訊，或使用 `botframework` 標記來張貼您自己的問題。請注意，Stack Overflow 有一些使用上的指導方針，例如要求必須使用描述性的標題、完整且簡潔的問題陳述，以及能夠重現問題的足夠詳細資料。功能要求或過於廣泛的問題都是偏離主題的；如需詳細資訊，新的使用者應瀏覽 [Stack Overflow 說明中心](#)。
- 請參閱 GitHub 中的 [BotBuilder 問題](#) (英文) 以取得 Bot Framework SDK 的相關已知問題，或回報新的問題。
- 利用 BotBuilder 社群在 [Gitter](#) (英文) 上的討論內容中的資訊。

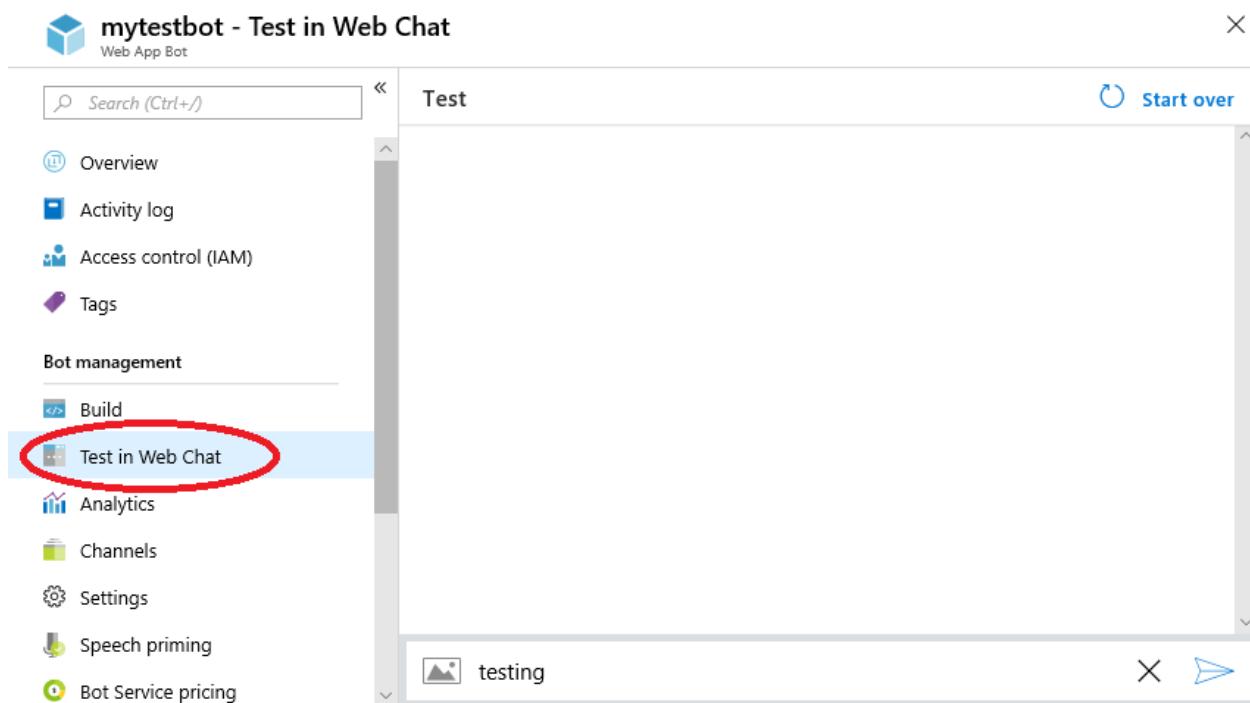
針對 Bot 設定問題進行疑難排解

2019/5/10 • [Edit Online](#)

針對 Bot 進行疑難排解的第一個步驟，是在網路聊天中進行測試。這可讓您判斷問題是否為 Bot (Bot 不適用於任何通道) 或特定通道 (Bot 適用於有些通道，但不適用於其他通道) 所特有。

在網路聊天中測試

1. 在 [Azure 入口網站](#) 中開啟 Bot 資源。
2. 開啟 [在網路聊天中測試] 窗格。
3. 將訊息傳送給您的 Bot。



如果 Bot 並未以預期的輸出回應，請移至 [Bot 不適用於網路聊天](#)。否則，移至 [Bot 適用於網路聊天，但不適用於其他通道](#)。

Bot 不適用於網路聊天

Bot 無法運作的可能原因很多。最可能是 Bot 應用程式已關閉且無法接收訊息，或 Bot 收到訊息但無法回應。

若要查看 Bot 是否正在執行：

1. 開啟 [概觀] 窗格。
2. 複製 [訊息端點] 並將其貼到您的瀏覽器。

如果此端點傳回 HTTP 錯誤 405，則表示 Bot 可觸達且 Bot 能夠回應訊息。您應該調查 Bot 是否逾時或因為 HTTP 5xx 錯誤而失敗。

如果此端點傳回錯誤「此站台無法連線」或「無法觸達此頁面」，這表示 Bot 已關閉而且需要重新部署。

Bot 適用於網路聊天，但不適用於其他通道

如果 Bot 如預期般在網路聊天中運作，但在某些其他通道中失敗，可能原因如下：

- [通道設定問題](#)
- [通道特有的行為](#)
- [通道中斷](#)

通道設定問題

通道設定參數的設定可能不正確，或已從外部進行變更。例如，Bot 已針對特定頁面設定 Facebook 通道，而此頁面後來遭到刪除。最簡單的解決方式是移除通道並重新進行通道設定。

以下連結提供設定 Bot Framework 所支援通道的指示：

- [Cortana](#)
- [DirectLine](#)
- [電子郵件](#)
- [Facebook](#)
- [GroupMe](#)
- [Kik](#)
- [Microsoft Teams](#)
- [Skype](#)
- [商務用 Skype](#)
- [Slack](#)
- [Telegram](#)
- [Twilio](#)

通道特有的行為

有些功能的實作可能因通道而有所不同。例如，並非所有通道都支援調適型卡片。大部分通道都支援按鈕，但是會以通道特有方式轉譯。如果您看到某些訊息類型在不同通道中的運作方式有所差異，請參閱 [Channel Inspector](#)。

以下是一些額外的連結，可協助使用個別的通道：

- [將 Bot 新增至 Microsoft Teams 應用程式](#)
- [Facebook: Messenger 平台簡介](#)
- [Cortana 技能設計原則](#)
- [適用於開發人員的 Skype](#)
- [Slack: 能夠與 Bot 互動](#)

通道中斷

偶爾，某些通道可能會中斷服務。通常，這類中斷不會持續很久。不過，如果您懷疑發生中斷，請洽詢通道網站或社交媒體。

判斷通道是否中斷的另一種方法是建立測試 Bot (例如簡單的 Echo Bot) 並新增通道。如果測試 Bot 適用於某些通道，但不適用於其他通道，則表示問題不在您的生產 Bot。

其他資源

請參閱[偵錯 Bot](#)操作說明，以及該區段中的其他偵錯文章。

針對 HTTP 500 錯誤進行疑難排解

2019/5/14 • [Edit Online](#)

針對 500 錯誤進行疑難排解的第一個步驟是啟用 Application Insights。

如需有關如何將 Application Insights 新增至現有 Bot 的資訊，請參閱[對話分析遙測](#)。

在 ASP.NET 上啟用 Application Insights

如需基本 Application Insights 支援，請參閱如何[設定 ASP.NET 網站的 Application Insights](#)。Bot Framework (從 4.2 版開始) 會提供一層額外的 Application Insights 遙測，但並非診斷 HTTP 500 錯誤的必要項目。

在 Node.js 上啟用 Application Insights

如需基本 Application Insights 支援，請參閱如何[使用 Application Insights 監視 Node.js 服務和應用程式](#)。Bot Framework (從 4.2 版開始) 會提供一層額外的 Application Insights 遙測，但並非診斷 HTTP 500 錯誤的必要項目。

查詢例外狀況

分析 HTTP 狀態碼 500 錯誤的最簡單方法是從例外狀況著手。

下列查詢會告訴您最近的例外狀況：

```
exceptions  
| order by timestamp desc  
| project timestamp, operation_Id, appName
```

從第一個查詢，選取一些作業識別碼並尋找更多資訊：

```
let my_operation_id = "d298f1385197fd438b520e617d58f4fb";  
let union_all = () {  
    union  
    (traces | where operation_Id == my_operation_id),  
    (customEvents | where operation_Id == my_operation_id),  
    (requests | where operation_Id == my_operation_id),  
    (dependencies | where operation_Id == my_operation_id),  
    (exceptions | where operation_Id == my_operation_id)  
};  
  
union_all  
| order by timestamp desc
```

如果您只有 `exceptions`，請分析詳細資料並查看是否對應於您程式碼中的各行。如果您只看到來自通道連接器的例外狀況 (`Microsoft.Bot.ChannelConnector`)，請參閱[沒有 Application Insights 事件](#)以確保 Application Insights 已正確設定且您的程式碼正在記錄事件。

沒有 Application Insights 事件

如果您收到 500 錯誤，而且 Application Insights 中沒有來自您 Bot 的進一步事件，請檢查下列各項：

確保 Bot 在本機執行

先使用模擬器確保 Bot 在本機執行。

確保正在複製組態檔案 (僅限 .NET)

確保在部署過程中正確封裝 `appsettings.json` 和其他設定檔。

應用程式組件

確保在部署過程中正確封裝 Application Insights 組件。

- Microsoft.ApplicationInsights
- Microsoft.ApplicationInsights.TraceListener
- Microsoft.AI.Web
- Microsoft.AI.WebServer
- Microsoft.AI.ServeTelemetryChannel
- Microsoft.AI.PerfCounterCollector
- Microsoft.AI.DependencyCollector
- Microsoft.AI.Agent.Intercept

確保在部署過程中正確封裝 `appsettings.json` 和其他設定檔。

appsettings.json

在 `appsettings.json` 檔案內，確保已設定檢測金鑰。

- [ASP.NET Web API](#)
- [ASP.NET Core](#)

```
{  
  "Logging": {  
    "IncludeScopes": false,  
    "LogLevel": {  
      "Default": "Debug",  
      "System": "Information",  
      "Microsoft": "Information"  
    },  
    "Console": {  
      "IncludeScopes": "true"  
    }  
  }  
}
```

確認設定檔

確保設定檔包含 Application Insights 金鑰。

```
{  
  "ApplicationInsights": {  
    "type": "appInsights",  
    "tenantId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",  
    "subscriptionId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",  
    "resourceGroup": "my resource group",  
    "name": "my appinsights name",  
    "serviceName": "my service name",  
    "instrumentationKey": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",  
    "applicationId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",  
    "apiKeys": {},  
    "id": ""  
  }  
},
```

查看記錄

Bot ASP.Net 和 Node 會在伺服器層級發出可以檢查的記錄。

設定要監看您記錄的瀏覽器

1. 在 [Azure 入口網站](#) 中開啟 Bot。
2. 開啟 [App Service 設定 / 所有 App Service 設定] 頁面，以查看所有服務設定。
3. 開啟應用程式服務的 [監視 / 診斷記錄] 頁面。
 - 確保已啟用 [應用程式記錄 (檔案系統)]。如果變更此設定，請務必按一下 [儲存]。
4. 切換至 [監視 / 記錄資料流] 頁面。
 - 選取 [Web 伺服器記錄]，並確保您看到已連線的訊息。您應該會看到如下的內容：

```
Connecting...
2018-11-14T17:24:51 Welcome, you are now connected to log-streaming service.
```

讓此視窗保持開啟。

設定瀏覽器以重新啟動您的 Bot 服務

1. 使用不同的瀏覽器，在 Azure 入口網站中開啟 Bot。
2. 開啟 [App Service 設定 / 所有 App Service 設定] 頁面，以查看所有服務設定。
3. 切換至應用程式服務的 [概觀] 頁面，按一下 [重新啟動]。
 - 系統會提示您是否確定選取 [是]。
4. 返回第一個瀏覽器視窗並監看記錄。
5. 確認您正在接收新的記錄。
 - 如果沒有任何活動，請重新部署您的 Bot。
 - 然後切換到 [應用程式記錄] 頁面並尋找是否有任何錯誤。

針對 Bot Framework 驗證進行疑難排解

2019/5/10 • [Edit Online](#)

此指南可協助您透過評估一系列的案例，來判斷問題所在，從而解決 Bot 的驗證問題。

NOTE

若要完成此指南中的所有步驟，您必須下載並使用 [Bot Framework Emulator](#)，而且必須能夠存取 [Azure 入口網站](#) 中的 Bot 註冊設定。

應用程式識別碼和密碼

Bot 安全性由您在使用 Bot Framework 註冊 Bot 時取得的 **Microsoft 應用程式識別碼** 和 **Microsoft 應用程式密碼** 進行設定。這些值通常在 Bot 的設定檔內指定，且用來擷取來自 Microsoft 帳戶服務的存取權杖。

如果您還沒有這樣做，請[將您的 Bot 部署到 Azure](#)，以取得可用於進行驗證的 **Microsoft 應用程式識別碼** 和 **Microsoft 應用程式密碼**。

NOTE

若要尋找已部署 Bot 的 **AppID** 和 **AppPassword**，請參閱 [MicrosoftAppId](#) 和 [MicrosoftAppPassword](#)。

步驟 1：停用安全性，並在 localhost 上進行測試

在此步驟中，您將在停用安全性時驗證您的 Bot 是否可在 localhost 上存取與執行。

WARNING

停用 Bot 的安全性可能會允許未知的攻擊者冒充使用者。只有當您在受保護的偵錯環境中操作時，才實作下列程序。

停用安全性

若要停用 Bot 的安全性，請編輯其組態設定，以移除應用程式識別碼和密碼的值。

如果您使用適用於 .NET 的 Bot Framework SDK，請在 Web.config 檔案中編輯這些設定：

```
<appSettings>
  <add key="MicrosoftAppId" value="" />
  <add key="MicrosoftAppPassword" value="" />
</appSettings>
```

如果您使用適用於 Node.js 的 Bot Framework SDK，請編輯這些值（或更新對應的環境變數）：

```
var connector = new builder.ChatConnector({
  appId: null,
  appPassword: null
});
```

如果您使用適用於 .NET 的 Bot Framework SDK，請在 `appsettings.json` 檔案中編輯這些設定：

```
"MicrosoftAppId": "<your app ID>",
"MicrosoftAppPassword": "<your app password>"
```

如果您使用適用於 Node.js 的 Bot Framework SDK, 請編輯這些值 (或更新對應的環境變數):

```
const adapter = new BotFrameworkAdapter({
  appId: null,
  appPassword: null
});
```

在 localhost 上測試您的 Bot

接下來, 使用 Bot Framework 模擬器在 localhost 上測試您的 Bot。

1. 在 localhost 上啟動您的 Bot。
2. 啟動 Bot Framework 模擬器。
3. 使用模擬器連線至您的 Bot。
 - 將 `http://localhost:port-number/api/messages` 輸入模擬器的網址列, 其中連接埠號碼必須符合應用程式執行所在瀏覽器中顯示的連接埠號碼。
 - 請確定 Microsoft 應用程式識別碼和 Microsoft 應用程式密碼欄位皆留白。
 - 按一下 [連接]。
4. 若要測試與 Bot 的連線, 請在模擬器中輸入一些文字, 然後按 Enter。

如果 Bot 回應輸入且聊天視窗中沒有任何錯誤, 則您已驗證在停用安全性時, 您的 Bot 可在 localhost 上存取和執行。請繼續進行 [步驟 2](#)。

如果聊天視窗中顯示一或多個錯誤, 請按一下錯誤以取得詳細資料。常見問題包括:

- 模擬器設定為 Bot 指定了不正確的端點。請確定您在 URL 中包含適當的連接埠號碼, 以及在 URL 結尾包含適當的路徑 (例如, `/api/messages`)。
- 模擬器設定指定以 `https` 開頭的 Bot 端點。在 localhost 上, 端點應以 `http` 開頭。
- 模擬器設定指定 Microsoft 應用程式識別碼欄位和/或 Microsoft 應用程式密碼欄位的值。這兩個欄位應為空白。
- Bot 的安全性尚未停用。[驗證](#) Bot 未指定應用程式識別碼或密碼的值。

步驟 2: 驗證 Bot 的應用程式識別碼和密碼

在此步驟中, 您將驗證 Bot 將用於驗證的應用程式識別碼和密碼是否有效 (如果您不知道這些值, [現在就取得](#))。

WARNING

下列指示會停用 `login.microsoftonline.com` 的 SSL 驗證。僅在安全的網路上執行此程序, 並考慮之後變更您的應用程式密碼。

向 Microsoft 登入服務發出 HTTP 要求

這些指示會說明如何使用 [cURL](#) 向 Microsoft 登入服務發出 HTTP 要求。您可以使用替代工具 (例如 Postman), 只需確定要求符合 Bot Framework [驗證通訊協定](#)。

若要驗證 Bot 的應用程式識別碼和密碼是否有效, 請使用 [cURL](#) 發出下列要求, 將 `APP_ID` 和 `APP_PASSWORD` 取代為 Bot 的應用程式識別碼和密碼。

TIP

您的密碼包含特殊字元，以致下列呼叫無效。若是如此，請嘗試將您的密碼轉換成 URL 編碼。

```
curl -k -X POST https://login.microsoftonline.com/botframework.com/oauth2/v2.0/token -d  
"grant_type=client_credentials&client_id=APP_ID&client_secret=APP_PASSWORD&scope=https%3A%2F%2Fapi.botframewo  
rk.com%2F.default"
```

此要求會嘗試交換 Bot 的應用程式識別碼和密碼，以取得存取權杖。如果要求成功，您將收到包含 `access_token` 屬性的 JSON 承載。

```
{"token_type": "Bearer", "expires_in": 3599, "ext_expires_in": 0, "access_token": "eyJ0eXAjKV1Q..."}  
access_token
```

如果要求成功，則您已驗證在要求中指定的應用程式識別碼和密碼有效。請繼續進行 [步驟 3](#)。

如果在回應要求時收到錯誤，請檢查回應以找出錯誤的原因。如果回應指出應用程式識別碼或密碼不正確，請從 Bot Framework 入口網站 [取得正確的值](#)，並使用新值重新發出要求，以確認其是否有效。

步驟 3：啟用安全性，並在 localhost 上進行測試

此時，您已驗證在停用安全性時，您的 Bot 可在 localhost 上存取與執行，並確認 Bot 將用於驗證的應用程式識別碼和密碼有效。在此步驟中，您將在啟用安全性時驗證您的 Bot 是否可在 localhost 上存取與執行。

啟用安全性

您的 Bot 的安全性依賴 Microsoft 服務，即使您的 Bot 只在 localhost 上執行。若要啟用 Bot 的安全性，請編輯其組態設定，以使用您在 [步驟 2](#) 中驗證的值填入應用程式識別碼和密碼。此外，確定您的套件是最新狀態，特別是 `System.IdentityModel.Tokens.Jwt` 和 `Microsoft.IdentityModel.Tokens`。

如果您使用適用於 .NET 的 Bot Framework SDK，請在 `appsettings.config` 中填入這些設定，或在 `appsettings.json` 檔案中填入對應的值：

```
<appSettings>  
  <add key="MicrosoftAppId" value="APP_ID" />  
  <add key="MicrosoftAppPassword" value="PASSWORD" />  
</appSettings>
```

如果您使用適用於 Node.js 的 Bot Framework SDK，請填入這些設定（或更新對應的環境變數）：

```
var connector = new builder.ChatConnector({  
  appId: 'APP_ID',  
  appPassword: 'PASSWORD'  
});
```

NOTE

若要尋找 Bot 的 `AppID` 和 `AppPassword`，請參閱 [MicrosoftAppID](#) 和 [MicrosoftAppPassword](#)。

在 localhost 上測試您的 Bot

接下來，使用 Bot Framework 模擬器在 localhost 上測試您的 Bot。

1. 在 localhost 上啟動您的 Bot。
2. 啟動 Bot Framework 模擬器。

3. 使用模擬器連線至您的 Bot。

- 將 `http://localhost:port-number/api/messages` 輸入模擬器的網址列，其中連接埠號碼必須符合應用程式執行所在瀏覽器中顯示的連接埠號碼。
- 在 **Microsoft 應用程式識別碼** 欄位中輸入您 Bot 的應用程式識別碼。
- 在 **Microsoft 應用程式密碼** 欄位中輸入您的 Bot 密碼。
- 按一下 [連接]。

4. 若要測試與 Bot 的連線，請在模擬器中輸入一些文字，然後按 Enter。

如果 Bot 回應輸入且聊天視窗中沒有任何錯誤，則您已驗證在啟用安全性時，您的 Bot 可在 localhost 上存取和執行。請繼續進行 [步驟 4](#)。

如果聊天視窗中顯示一或多個錯誤，請按一下錯誤以取得詳細資料。常見問題包括：

- 模擬器設定為 Bot 指定了不正確的端點。請確定您在 URL 中包含適當的連接埠號碼，以及在 URL 結尾包含適當的路徑（例如，`/api/messages`）。
- 模擬器設定指定以 `https` 開頭的 Bot 端點。在 localhost 上，端點應以 `http` 開頭。
- 在模擬器設定中，**Microsoft 應用程式識別碼** 欄位和/或 **Microsoft 應用程式密碼** 欄位未包含有效的值。應填入這兩個欄位，而且每個欄位應包含您在 [步驟 2](#) 中驗證的對應值。
- Bot 的安全性尚未啟用。[驗證](#) Bot 組態設定指定應用程式識別碼和密碼的值。

步驟 4：在雲端測試您的 Bot

此時，您已驗證在停用安全性時，您的 Bot 在 localhost 上可存取並正常執行、已確認您的 Bot 應用程式識別碼和密碼有效，且已驗證在啟用安全性時您的 Bot 在 localhost 上可存取並正常執行。在此步驟中，您會將您的 Bot 部署到雲端，並在啟用安全性的情況下驗證它是否可存取並正常執行。

將您的 Bot 部署至雲端

Bot Framework 要求可從網際網路存取 Bot，因此您必須將您的 Bot 部署至裝載平台，例如 Azure 雲端。確保在部署之前為您的 Bot 啟用安全性，如 [步驟 3](#) 中所述。

NOTE

若您還沒有雲端裝載提供者，則可以註冊[免費帳戶](#)。

如果您將 Bot 部署至 Azure，系統會自動為您的應用程式設定 SSL，以啟用 Bot Framework 所需的 **HTTPS** 端點。如果您部署至其他雲端裝載提供者，請務必確認您的應用程式已設定 SSL，讓 Bot 能夠有 **HTTPS** 端點。

測試 Bot

若要在啟用安全性的情況下在雲端中測試 Bot，請完成下列步驟。

1. 請確定您的 Bot 已成功部署和執行。
2. 登入 [Azure 入口網站](#)。
3. 按一下 [我的 Bot]。
4. 選取您想要測試的 Bot。
5. 按一下 [測試] 以在內嵌的 Web 聊天控制項中開啟 Bot。
6. 若要測試與 Bot 的連線，請在網絡聊天控制項中輸入一些文字，然後按 Enter。

如果聊天視窗中指出錯誤，請使用錯誤訊息來判斷錯誤的原因。常見問題包括：

- 在 Bot Framework 入口網站中針對 Bot 的 [設定] 頁面上指定的傳訊端點不正確。請確定您在 URL 結尾包含適當的路徑（例如，`/api/messages`）。
- 在 Bot Framework 入口網站中針對 Bot 的 [設定] 頁面上指定的傳訊端點未以 `https` 開頭，也不受 Bot Framework 信任。您的 Bot 必須具有有效的鏈結信任憑證。

- 為 Bot 設定之應用程式識別碼或密碼的值遺失或不正確。[驗證](#) Bot 組態設定指定應用程式識別碼和密碼的有效值。

如果 Bot 對輸入做出了適當的回應，那麼您已驗證您的 Bot 在啟用安全性的情況下可以在雲端中存取與執行。此時，您的 Bot 已準備好安全地[連線到通道](#)，例如 Skype、Facebook Messenger，直接線路等。

其他資源

如果在完成上述步驟之後仍然遇到問題，您可以：

- 請檢閱[偵錯 Bot](#)操作說明，以及該區段中的其他偵錯文章。
- 使用 Bot Framework Emulator 和 [ngrok](#) 通道軟體[在雲端對您的 Bot 進行偵錯](#)。*ngrok 並非 Microsoft 產品。*
- 使用像 [Fiddler](#) 這樣的 Proxy 處理工具，來檢查進出 Bot 的 HTTPS 流量。*Fiddler 並非 Microsoft 產品。*
- 若要了解 Bot Framework 所使用的驗證技術，請檢閱[Bot 連接器驗證指南](#)。
- 使用 Bot Framework [支援](#)資源向其他人請求協助。