iTeach Final-a : Developer Document

Team Member: b03901028 蔡丞昊、b03901032 郭子生

I. 技術介紹

Overview

Multipeer Connectivity 讓使用者能夠發現周圍裝置所提供的服務，並透過訊息、串流或檔案的方式溝通。在 iOS 的系統中，Multipeer Connectivity 以 infrastructure Wifi、peer-to-peer Wifi 和藍芽作為底層的傳輸方式。

Architecture

在使用 Multipeer Connectivity 時，我們會需要操作以下物件：

- Session：用於已連接裝置之間的通訊。App 會創建一個 Session，接著會邀請其他裝置加入。待他們同意後，就會被加進這個 Session。Session 裡面存了一組 PeerID 來代表連接了 Session 的裝置。
- Advertiser：會將自己的資訊告訴周圍的裝置。
- Browser：會搜尋四周的裝置。
- PeerID：代表裝置上運行 App 的代碼，不同裝置或 App 會擁有不同代碼。

Procedure

裝置的連接分為兩個階段：Discovery 階段與 Session 階段。在 Discovery 階段中，Browser 會搜尋附近的裝置，Advertiser 則會將自己的資訊告知其他人，以便他們能夠邀請自己進入 Session。在這個階段中，裝置彼此間僅能做非常有限的溝通，包括周圍 Advertiser 所提供的資訊，以及當有人邀請你時所提供的資料。當 Browser 發現周遭的 Advertiser 時，可以邀請他們加入到 Session。被邀請的裝置則可以詢問使用者來決定要不要接受邀請。接受邀請後，Broswer 就會和 Advertiser 建立連線，進入 Session 階段。此時，裝置間就可以進行資料的傳輸。當有裝置加入或離開 Session 時，Multipeer Connectivity 會透過 callback 通知 App。

當 App 被移至背景執行時，Multipeer Connectivity 會停止 browse 和 advertise，並將所有 Session 關閉；當 App 回到前台執行，browse 和 advertise 會自動回覆，但 Session 需要被重新連接。

II. 架構介紹

底層

Multipeer Connectivity 的底層由 Objective C 編寫，負責發送 event 到上層，以及實際執行上層傳遞下來的指令。底層程式負責的工作包括指定 PeerID、搜尋附近裝置、在裝置間傳送資訊等等。

以 sendData 為例，當裝置 A 的底層收到上層 sendData 的指令，它會負責將資料傳輸給裝置 B。當裝置 B 收到資料，底層會觸發 RCTMultipeerConnectivityDataReceived 的事件，而上層則會有相應的 Listener 來處理收到的資料。

JS

中間的 JS 程式負責接收底層觸發的 event，並提供介面執行底層的指令。此處的工作包括註冊 Listener、根據接收的 event 改變更上層的 UI、根據使用者對 UI 的操作執行底層的指令。要注意的是，這部分的程式融合了 Redux 的概念，當 Listener 接收到 event，它會 dispatch action，reducer 會接收 action 並將一些資訊如 PeerID 存回 Store，進而影響上層 UI。

以 peerFound 為例，當底層觸發了 RCTMultipeerConnectivityPeerFound 的事件，Listener 會接收並 dispatch multiPeer.backend.onPeerFound。這個 action 經過 middlewares 處理完後（中間會再 dispatch savePeerInfo & updatePeerStatus 的 actions），會由 reducer 接收，將新的 PeerID 加入 Session 中。再以 browse 做為範例，當老師進入課程主頁之後，UI 會 dispatch multiPeer.backend.browse，而執行這個 action 時，則會相應地執行底層的 MultipeerConnection.browse。

UI

頂端的 UI 負責根據 store 來改變使用者介面，或根據使用者的操作 dispatch action 來執行 JS 層提供的指令介面。

以 dataReceived 為例，當學生接收到老師傳過來的課程更新資訊，學生端 App 的 middleware 會將資訊更新到 store 中，並 reload 畫面讓最新的課程資訊顯示在螢幕上。

再以學生 openCourse 為例，當學生點擊在課程選單點擊課程進入課程主頁，UI 會 dispatch openCourse 的 action，並根據它 dispatch updateOwnStatus 和 advertise，將自己設為 ADVERTISE 的狀態並執行 JS 層提供的 advertise 指令。

III. 檔案介紹

Architecture

```
src
├ actions
│    └── multiPeer.action.js
├ submodule
│    └── react-native-multipeer
│          ├── classes
│          │       ├── MultipeerConnection.class.js
│          │       ├── MultipeerConnectionInit.js
│          │       └── Peer.class.js
│          ├── reducers
│          │       └── MultiPeer.reducer.js
│          ├── RCTMultipeerConnectivity.h
│          └── RCTMultipeerConnectivity.m
└ utils
     └── middlewares.js
```

Functionality

- actions/multiPeer.action.js

  定義了與 Multipeer Connectivity 功能相關的 API，分為 student、teacher、backend、以及 common，詳細內容在 **IV. API Reference** 中會完整介紹。

- submodule/react-native-multipeer/classes/MultipeerConnection.class.js

  提供了給 JS 層呼叫的指令，包括 browse、advertise、invite 等等。

- submodule/react-native-multipeer/classes/MultipeerConnectionInit.js

  註冊 Listener 來接收底層觸發的 event，詳細內容見 **V. Listeners**。

- submodule/react-native-multipeer/classes/PeerInfo.class.js

  定義 Peer 的資訊如何被存在 Store 裡。

  - userId：代表不同 user 的 id，同一台裝置只會有一個 userId
  - username：使用者名稱
  - isTeacher：是否為老師

- submodule/react-native-multipeer/classes/PeerStatus.class.js

  定義 Peer 的狀態如何被存在 Store 裡。

  - currPeerId：底層指定的 PeerId，App 每次開啟時都會指定不同 id
  - currCourse：目前參與的課程資訊
  - connected：與自己是否相連
  - invited：是否邀請自己 / 被自己邀請
  - invitationId：老師傳送給學生的 invitationId
  - username：使用者名稱

- submodule/reducers/MultiPeer.reducer.js

  定義 Store 內 MultiPeer 相關 state 的初始狀態，以及 reducer 接收 action 後如何更新 state。

- submodule/RCTMultipeerConnectivity.h/m

  底層的 Objective C 程式碼。

- utils/middlewares.js

  在 action 被傳到 reducer 前，某些 action 會被做一些處理。比如 onDataReceived 會根據 messageType 有不同的操作：CHOSEN_ONE 觸發抽籤功能的視窗、QUESTION_DEBUT 觸發回答問題的視窗、等等。

IV. API Reference

iTeach 是建立在 React + Redux 的架構之下，因此呼叫 Multipeer 的方式是透過 dispatch 在 src/actions/multiPeer.action.js 之中的 action 來實現，之後會由 src/submodule/react-native-multipeer/reducers/MultiPeer.reducer.js 來處理這些 action，因此如果是單純呼叫 API 只需參考 src/actions/multiPeer.action.js 即可。

在 multiPeer 的 actions 之中，可以再細分做四個類別，分別是 student、teacher、backend、以及 common。前兩者顧名思義分別是給學生端以及老師端呼叫使用，後兩者是學生以及老師端的 action 被呼叫之後，後端再 dispatch 的 action，作為與底層的 MultipeerConnectivity 之間的橋樑。因此若單純要呼叫 API，只需要了解與前端有所連結的 student 以及 teacher 的 action 即可。以下簡介 src/actions/multiPeer.action.js 所提供的 actions 呼叫方式、格式、回傳格式，並提供使用範例做為參考。

- multiPeer.student.startSearch()

  呼叫方式：

  Import multiPeerAction from '$PATH/src/actions/multiPeer.action'
  dispatch(multiPeerAction.student.startSearch())

  說明：

  學生在 CourseMenu 頁面點選搜尋 icon 時，系統會跳轉到 CourseSearch 的頁面並 dispatch 這個 action，目的為開始搜尋老師正在發佈中的課程。這個 action 會 dispatch backend 當中負責 advertise 的 action，將學生的資訊透過底層的 MultipeerConnectivity advertise 出去，同時將 student 的狀態透過 common.action 設為 ADVERTISE（isAdvertising = true）。

- multiPeer.student.stopSearch()

  呼叫方式：

  Import multiPeerAction from '$PATH/src/actions/multiPeer.action'
  dispatch(multiPeerAction.student.stopSearch())

說明：

學生在 CourseSearch 的頁面有兩種情況會停止搜尋：1）搜尋到想要的課程並點選加入；2）離開 CourseSearch 頁面。在這兩種情況下這個 action 會被 dispatch，透過 backend 的 hide 告知底層的 MultipeerConnectivity 停止 advertise，並將 student 的狀態透過 common 的 action 設為 STOP_ADVERTISE（isAdvertising = false）。

- multiPeer.student.openCourse()

  呼叫方式：

  Import multiPeerAction from '$PATH/src/actions/multiPeer.action'
  dispatch(multiPeerAction.student.openCourse())

  說明：

  學生在 CourseMenu 的頁面點選課程進去時，系統會跳轉到 CourseHome 的頁面並 dispatch 這個 action。接著這個 action 會 dispatch backend 的 action 指示 MultipeerConnectivity 開始 advertise 學生的 info，同時將學生的狀態設為 ADVERTISE（isAdvertising = true）。

- multiPeer.student.exitCourse()

  呼叫方式：

  Import multiPeerAction from '$PATH/src/actions/multiPeer.action'
  dispatch(multiPeerAction.student.exitCourse())

  說明：

  學生在 CourseHome 的頁面離開時，會 dispatch courseHomeAction 的 exit action。courseHomeAction 的 exit action 除了處理其他的東西之外，還會 dispatch multiPeer.student.exitCourse 這個 action。這個 action 除了要通知 backend hide （停止 advertise）之外，還要和 connected 的其他 Peers 們 disconnect，原因是學生離開課程頁面之後就應被視為已下線。最後透過 common action 將學生的狀態設為 STOP_ADVERTISE（isAdvertising = false）。

- multiPeer.student.requestCourseInfo()

  呼叫方式：

  Import multiPeerAction from '$PATH/src/actions/multiPeer.action'
  dispatch(multiPeerAction.student.requestCourseInfo())

  說明：

  學生上線時會問說是否有課程資訊更新。由於學生上線時老師並不一定在線上，因此學生是利用 backend broadcast 的方式問所有人，如果有人在線上收到 message（老師或學生皆可），且 TimeStamp 比 broadcast 的人更新的話，就會回覆一個 COURSE_INFO_UPDATE 的 message。這個過程可以在 middleware 找到。

- multiPeer.teacher.openCourse()

  呼叫方式：

  Import multiPeerAction from '$PATH/src/actions/multiPeer.action'
  dispatch(multiPeerAction.teacher.openCourse())

  說明：

  老師在 CourseMenu 的頁面中可以點選並進入課程，此時系統會 dispatch 這個 action。系統首先透過 backend 底層的 MultipeerConnectivity 開始進行 browsing，同時透過 common 的 action 將老師的狀態設為 BROWSE（isBrowsing = true）。

- multiPeer.teacher.exitCourse()

  呼叫方式：

  Import multiPeerAction from '$PATH/src/actions/multiPeer.action'
  dispatch(multiPeerAction.teacher.exitCourse())

說明：

當老師離開 CourseHome 的頁面時，會透過 courseHome.action 來 dispatch teacher.exitCourse 這個 action。首先由於老師離開頁面即視為下線，因此要和 Peers disconnect。此外老師也要停止 browsing，並將狀態設為 STOP_BROWSE（isBrowsing = false）和 STOP_RELEASE（isReleasing = false）。

V. Listeners

Listeners 定義在 src/submodule/react-native-multipeer/classes 底下，這底下有兩個主要和 Listerners 相關的檔案：1）MultipeerConnection.class.js；2）MultipeerConnectionInit.js。前者定義了處理 Listeners 的 class，後者則是 Listener 的註冊方式以及事件發生時機。當 Listener 收到不同的 event 時，handler 就會 dispatch src/actions/multiPeer.action.js 中的 backend actions，以下將簡單介紹各事件發生的時機，以及 handler 所 dispatch 的 actions。

- RCTMultipeerConnectivityPeerFound

  當 Listener 聽到這個 event 時，會 dispatch backend.onPeerFound 這個 action，傳入的參數是 peer 的底層 id 以及 info。然而由於即便是同一個 user，每次連線時都會產生不同的底層 id，因此 backend.onPeerFound 會更新這些底層 id 不同，但高層 id 相同的連線，同時也會把更新過的 Peer information 存入 local storage。

- RCTMultipeerConnectivityPeerLost

  當 Listener 聽到這個 event 時，會 dispatch backend.onPeerLost 這個 action，傳入的參數是底層的 id。如果這個 peer 是 state 中的 peer 之一的會，就會再 dispatch action 做處理。

- RCTMultipeerConnectivityPeerConnected

  當 Listener 聽到這個 event 時，會 dispatch backend 的兩個 action。一個 onPeerConnected 告知已經與 Peer 連線，另一個是 requestInfo，傳入 peer 底層 id 為參數並傳送 message。

- RCTMultipeerConnectivityPeerConnecting

  當 Listener 聽到這個 event 時， dispatch onPeerConnecting action，沒有太特別的 handle。

- RCTMultipeerConnectivityPeerDisconnected

  當 Listener 聽到這個 event 時，dispatch onPeerDisconnected action，沒有太特別的 handle。

- RCTMultipeerConnectivityStreamOpened

  當 Listener 聽到這個 event 時，dispatch onStreamOpened action，沒有太特別的 handle。

- RCTMultipeerConnectivityInviteReceived
  當 Listener 聽到這個 event 時，handler 會先創建一個 invitation object，接著將他作為參數 dispatch backend.onInviteReceived 這個 action。若比較後發現與 Peer 的 CourseId 相同，則透過 backend responseInvite。最後會更新 state 中的 Peer 資訊並存入 local storage。

- RCTMultipeerConnectivityDataReceived

  當 Listener 聽到這個 event 時，會把 peer 底層 id 以及 data 作為參數 dispatch backend.onDataReceived 這個 action。系統會在 middleware 當中根據 data 的 messageType 去分類處理。