Firebase 及時資料庫

使用NoSQL儲存和同步雲端資料，所有不同平台的裝置可以及時同步資料，並且在沒連線狀態時依舊可以使用。

Firebase及時資料庫是雲端主機資料庫。資料儲存的格式JSON，有連線的所有使用者端將會及時同步資料。當使用ios,Android,Javascript開發跨平台的App，所有的不同平台的使用者端將會共同分享一個及時資料庫的實體，並且同時接收到更新的最新資料。

主要的能力

及時能力	取代傳統的HTTP需求，Firebase及時資料庫使用資料改變時及時更新資料的支術，使用連結的裝置在1000分之 1 秒同步更新。提供更好的連結方式和更深入的體驗，將不用再去管理網路連結的程式碼。
離線狀態	當離線模式時，Firebase app依舊可以回應事件。主要的原因是Firebase及時資料庫的SDK保存我們的資料在磁碟內。一但重新連線，裝置同步接受改變的資料讓使用者端保持和Server同樣的資料。
使用者端存取	Firebase及時資料庫可以被裝置或網頁直接存取。不需要再透過Server主機。透過Firebase 及時資料庫的安全規則是可以保護和驗證資料。當資料被讀出或寫入時，描述式的安全規則將會被執行，用以保護資料。

它如何工作？

Firebase及時資料庫讓你建立豐富和協同作業的應用程式，同時允許使用者端安全存取直接存取資料。資料被儲存在使用者端，當離線時，及時資料庫依舊可以繼續使用，讓使用者沒有任何感覺。當裝置重新連線時，及時資料庫會自動同步離線時變更的資料。

及時資料庫提供一個有彈性的描述式語言所建立的安全規則，我們稱為Firebase及時資料庫安全規則，規則定意資料被讀出和寫入時要如何架構資料。當有整合Firebase認證，開發者可以定義誰可以存取那些資料以及如何存取它們。

及時資料庫是一個NoSQL的資料庫，它和關聯式資料庫有不同的效能和功能。及時資料庫的API被設計成為一個可以快速執行的運算。這讓我們可以建立一個同時及時回應上萬使用者的應用程式，而不會讓資料庫回應變慢。就因為如此，思考依據使用者將如何存取資料來架構資料將變的非常重要。

學習路徑

整合Firebase 及時資料庫的SDK進入專案	使用CocoaPod，快速載入SDK進入專案
建立及時資料庫的參考	參考JSON資料，例如”users/user:1234/phone_number”，以便未來可以設定資料和繫結資料改變的事件。

設定資料和資料改變的監聽者	使用這些參考來寫入資料或註冊資料改變
建立離線存取的能力	允許資料可以被寫入至裝置內，達到離線能力
保護資料	使用Firebase 及時資料庫安全規則來保護資料

在ios上安裝和設定

Firebase 及時資料庫是一個雲端資料庫，資料儲存為json的格式，並且可以在及時反應資料內容給所有連線的使用者端。當我們使用Android,IOS和javascript開發的誇平台應用程式，所有的使用者端同時使用一個及時資料庫的實體和自動接收最新的更新資料。

事前準備

1.安裝Firebase SDK。
2.在Firebase console上，將App加入到Firebase 專案。

將Firebase Realtime 資料庫API加入至APP

在專案內的Podfile，確定有加入下面將安裝的內容

```
pod 'Firebase/Database'
```

在終端機執行 pod install和打開建立的.xcworkspace檔案。

調整Firebase 資料庫的規則

及時資料庫提供陳述式的語言來允許我們做設定。定義我們的資料如何被架構，如何被索引和何時資料可以被誰讀取和寫入。基本預設讀和寫資料庫是被限定的，只允許有經過認證的使用者存取。所以一開始不要設定認證的功能，您必需要調整你的資料庫的存取規則。讓你的資料庫可以讓任何人讀取，甚至不是你App的使用者也可以打開。當你設定好認證功能後，確定將限定存取的功能再打開。

設定及時資料庫

在使用Firebase App必需要先初始化Firebase 。
1.在UIApplicationDelegate 子類別，import Firebase Module。

```
import Firebase
```

2.調整FirebaseApp share 實體，一般是在application:didFinishLaunchingWithOptions: 的方法內來執行。

```
// Use Firebase library to configure APIs
FirebaseApp.configure()
```

一但已經完成Firebase realime Database的初始化。定義和建立一個及時資料庫的參考，如下:

```
var ref: DatabaseReference!

ref = Database.database().reference()
```

ref已經可以使用。

架構資料庫

這章節含蓋資料建構的重要觀念，和在一個在Firebase database 架構一個JSON資料的範例。
建構一個適當的資料庫，需要有相當的一個安排和思考。最重要的是您需要計畫那些資料要被儲存和那些資料將被取出。盡可能讓這些動作容易操作。

資料如何被架構:JSON樹

所有Firebase 及時資料庫的資料是儲存成為一個JSON的物件。你可以想像資料庫是一個雲端JSON樹。不像SQL database，沒有表格和資料錄。當加入資料時到JSON樹時，它成為在現有的JSON架構內的一個節點，並且有一個相對應的key。可以提供自訂的key，例如使用者ID或一個適當的名字或者可以使用childByAutoID取得到自動ID。

例如，建立一個聊天應用程式，允許使用者儲存基本的資料和聯絡方式。一個傳統的設定是設立一個路徑，如/users/$uid，使用者alovelace可以有一個資料庫的進入點，就像這樣:

```
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      "contacts": { "ghopper": true },
    },
    "ghopper": { ... },
    "eclarke": { ... }
  }
}
```

雖然資料庫使用JSON樹，資料儲存在資料庫內，可以當作是一個原生的類型，這原生的類型對應到可以使用的JSON類型。有利於我們寫程式控制。

架構資料的重要練習

避免巢狀資料

因為Firebase及時資料庫允許巢狀資料至32層深，你可能被引導認為這是基本的結構。然而當你在資料庫取得一個位置的資料，你同時取得這個位置下的所有節點資料。當你同意某人讀和寫這個資料庫的這個節點。你相對也同意某人存取這個節點下的所有資料。因此，盡可能保持資料平坦化。
下面的範例是不好的巢狀資料:

```
{
  // This is a poorly nested data architecture, because
iterating the children
  // of the "chats" node to get a list of conversation
titles requires
  // potentially downloading hundreds of megabytes of
messages
  "chats": {
    "one": {
      "title": "Historical Tech Pioneers",
      "messages": {
        "m1": { "sender": "ghopper", "message": "Relay
malfunction found. Cause: moth." },
        "m2": { ... },
        // a very long list of messages
      }
    },
    "two": { ... }
  }
}
```

使用這種巢狀設計，重複讀取資料變的困難。例如，要取出所有的聊天的title，需要全部的chars樹，包含所有的members和messages，都會被下載到使用者端。

平坦化資料

如果將資料切割進入到多個路徑，我們也稱為非正規化。當需要時，它可以更有效率的分開下載，這是平坦化資料結構

```
{
  // Chats contains only meta info about each conversation
  // stored under the chats's unique ID
  "chats": {
    "one": {
      "title": "Historical Tech Pioneers",
      "lastMessage": "ghopper: Relay malfunction found. Cause: moth.",
      "timestamp": 1459361875666
    },
    "two": { ... },
    "three": { ... }
  },

  // Conversation members are easily accessible
  // and stored by chat conversation ID
  "members": {
    // we'll talk about indices like this below
    "one": {
      "ghopper": true,
      "alovelace": true,
      "eclarke": true
    },
    "two": { ... },
    "three": { ... }
  },

  // Messages are separate from data we may want to iterate quickly
  // but still easily paginated and queried, and organized by chat
  // conversation ID
  "messages": {
    "one": {
      "m1": {
        "name": "eclarke",
        "message": "The relay seems to be malfunctioning.",
```

現在透過重覆存取，每個聊天室，只要下載部份資料。快速取得聯天室介面需要的資料。訊息也可以分開取的。

建立資料規則

當建立app時，如果資料包含成千上萬筆資料，最好只下載部份的資料。

```
// An index to track Ada's memberships
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      // Index Ada's groups in her profile
      "groups": {
        // the value here doesn't matter, just that the
key exists
        "techpioneers": true,
        "womentechmakers": true
      }
    },
    ...
  },
  "groups": {
    "techpioneers": {
      "name": "Historical Tech Pioneers",
      "members": {
        "alovelace": true,
        "ghopper": true,
        "eclarke": true
      }
    },
    ...
  }
}
```

讀寫資料

取得DatabaseReference

讀和寫入資料庫，您需要一個FIRDatabaseReference實體。

```
var ref: DatabaseReference!

ref = Database.database().reference()
```

讀和寫資料

這個文件包含基本的資料庫讀寫。

Firebase 資料被寫入到FIRDatabase參考和透過參考附加上一個非同步的監聽器來取得資料。
在監聽器被建立時，會觸發一次。後續只要資料改變，就會觸發。

```
重要:存取資料庫是被限制，預設的設定是必需要認證身份才能被讀寫的。在一開始可以設定不需要認證，必需去需改認證設定為公開存取。這將會造成您的資料開放給任何人存取。甚至不是您App的使用者也可以修改。未來要確定有限制資料庫必需要認證身份。
```

基本寫入的的運算

基本的寫入，可以使用setValue儲存資料到特定的路徑參考，將會取代先前路徑的資料。這個方法的參數可以使用的資料類型如下:

1. String
2. Int
3. Double
4. Dictionary
5. Array
6.

setValue使用方法如下範例

```
self.ref.child("users").child(user.uid).setValue(["username": username])
```

使用setValue會覆寫在指定路徑的所有資料，也包含它的子節點。然而你也可以只有更新局部的資料而不是全面的覆寫子節點所有資料。
如果只是更改單獨的子節點使用者名稱，可以使用下面方法:

```
self.ref.child("users/(user.uid)/
username").setValue(username)
```

監聽改變值的事件

讀取路徑節點的資料和監聽內容改變，使用DatabaseReference的observeEventType:with或observeSingleEventOfType:with方法，觀查DataEventTypeValue的改變

事件類型	使用方法
DataEventTypeValue	讀和監聽一個路徑節點的內容改變

你可以使用DataEventTypeValue事件來讀取特定路徑的資料。當將附加這個監聽，這個方法會被觸發一次。還有每次內容改變時，這個方法也會被觸發，這也包含所有子節點的內容被改變時。這個closure被傳遞了一個snapshot的參數,snapshot 包含了節點所有的資料，也包含了子節點的資料。如果沒有任何資料，snapshot的值將是nil。

```
重要:這DataEventTypeValue事件每次被指定路徑資料被改變時就會觸發一次。包含子節點的資料改變，也會觸發一次。為了要限制snapshot的資料大小，我們在指定路徑時，一定要到最適合的路徑節點。千萬不要指定root為監聽的節點。
```

下面是closure的範例:

```
refHandle = postRef.observe(DataEventType.value, with: {
(snapshot) in
  let postDict = snapshot.value as? [String : AnyObject]
?? [:]
  // ...
})
```

這取得的DataSnapshot的資料，並且使用value屬性取得內容值，所對應的為該JSON 指定節點的內容值。一般為NSDictionary，有key和value,需轉換為swift的Dictionary。

僅讀資料一次

在有些地方，我們僅需取得一次資料。例如介面要取得的資料來顯示，而這是不會改變的資料。此時就可以使用observeSingleEventOfType方法。這個方法內的參數closure只會被執行一次。
下面是範例:

```
let userID = Auth.auth().currentUser?.uid
ref.child("users").child(userID!).observeSingleEvent(of:
.value, with: { (snapshot) in
  // Get user value
  let value = snapshot.value as? NSDictionary
  let username = value?["username"] as? String ?? ""
  let user = User.init(username: username)

  // ...
  }) { (error) in
    print(error.localizedDescription)
}
```

更新和刪除資料