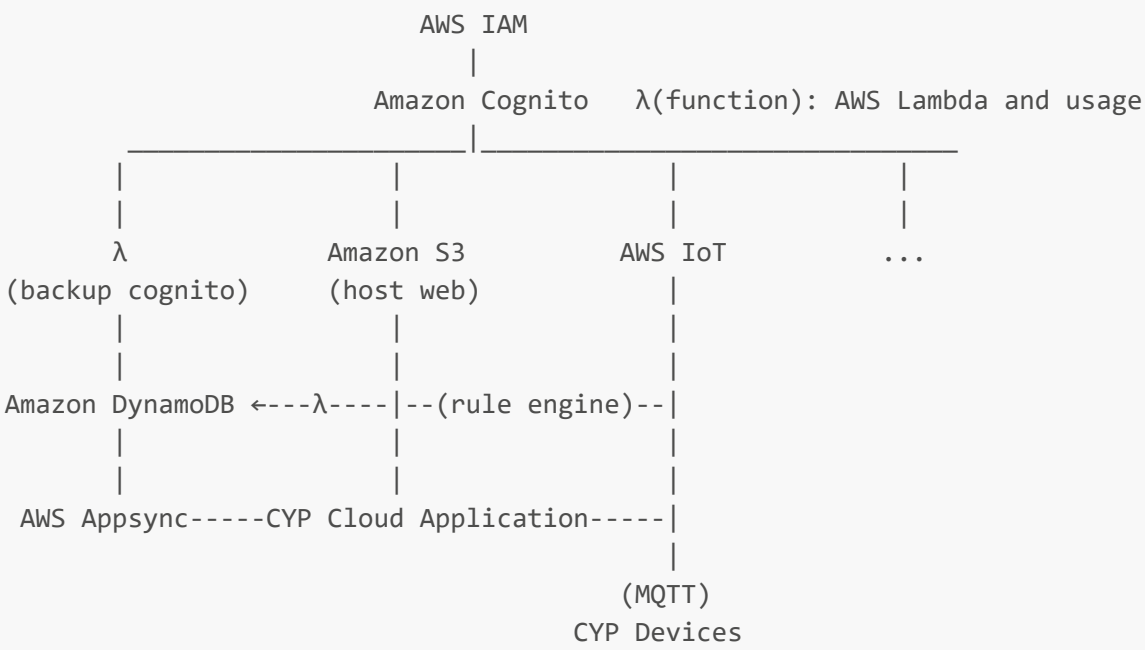
Outline

CYP Cloud Overview



Techniques

1. AWS IAM

IAM (Identify and Access Management)為連接所有服務的核心，只有在IAM授權過的存取才會在應用中生效。Access Permission可以透過Policy去制定。

- **[AWS IAM]**
  https://docs.aws.amazon.com/zh_tw/IAM/latest/UserGuide/introduction.html

2. Amazon Cognito

Cognito透過User Pool Identity Pool管理所有應用程式中的使用者。透過使用者池，管理者及開發人員能夠更好的區分不同等級的使用者權限。

- **[Amazon Cognito]**
  https://docs.aws.amazon.com/zh_tw/cognito/latest/developerguide/what-is-amazon-cognito.html
- **[Serverless-stack create user pool**
  https://serverless-stack.com/chapters/create-a-cognito-user-pool.html
- **[Serverless-stack create federated identities pool]** https://serverless-stack.com/chapters/create-a-cognito-identity-pool.html

3. Amazon S3

S3除了提供付費的使用空間供整個Application運用，還能夠對靜態網頁的內容做管理 *(static web hosting)*，使得 **serverless** 的架構更容易實現。

- **[Amazon S3]**
  https://docs.aws.amazon.com/zh_tw/AmazonS3/latest/dev/Welcome.html
- **[S3 Static Web Hosting]**
  https://docs.aws.amazon.com/zh_tw/AmazonS3/latest/dev/WebsiteHosting.html

4. Amazon DynamoDB

DynamoDB屬於 "*非關聯式資料庫*"*(Non-SQL database)* 的一種。傳統的關聯式資料庫具備可靠性、穩定性；非關聯式資料庫則具有容易擴充、彈性的資料結構等優點。Local Secondary Index, LSI 及 Global Secondary Index, GSI的使用也能補足非關聯式資料庫在條件存取的劣勢。

- **[Amazon DynamoDB]**
  https://docs.aws.amazon.com/zh_tw/amazondynamodb/latest/developerguide/Introduction.html
- **[DynamoDB Secodary Indexes]**
  https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SecondaryIndexes.html

5. AWS IoT

AWS IoT提供了一種透過網路與終端裝置(embedded, mobile, pc, etc.)互動的介面。除了基本的 Message Queuing Telemetry Transport, MQTT訊息傳輸介面外，AWS還在那之上實現了裝置影子(*Device Shadow*)來掌握即時的控制與狀態更新、以及AWS Greengrass(gateway)來對一群裝置做管理。

- **[AWS IoT Core]**
  https://docs.aws.amazon.com/zh_tw/iot/latest/developerguide/what-is-aws-iot.html
- **[AWS IoT Shadow]**
  https://docs.aws.amazon.com/zh_tw/iot/latest/developerguide/iot-device-shadows.html

- **[AWS IoT Greengrass]**
  https://docs.aws.amazon.com/zh_tw/greengrass/latest/developerguide/what-is-gg.html

6. AWS Lambda

7. AWS Appsync

#TODO

8. Web Application

第一，由於Amazon S3只提供靜態網頁託管的服務。第二，使用ReactJS開發的網頁應用可以被build成S3能夠託管的內容。ReactJS基於Node.js，Node.js的原身又是Javascript，因此學習使用這三樣東西是必須的。除此之外，Node.js會使用到的套件會使用npm做為套件管理系統，也需要學習如何使用npm來管理每個project的套件。

- **[npm Tutorial]**
  https://docs.npmjs.com/cli/npm
- **[Javeacript Tutorial]**
  https://www.w3schools.com/js/
- **[Node.js Tutorial]**
  https://www.w3schools.com/nodejs/
- **[ReactJS Tutorial]**
  https://www.tutorialspoint.com/reactjs/

為了要在web上存取AWS的服務，會使用到一些Node.js套件:

- **[AWS SDK for Javascript]**
  https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS.html
- **[AWS Ampllify on Javascript]**
  https://aws-amplify.github.io/docs/js/start?platform=purejs

---

Step by Step

Step: IAM

Basic Rule #1 Protect Your Credential

透過IAM政策提供許可時，僅授予一個對象其最低需要的權限。另外，小心保存您的任何密碼、存取ID、私鑰等等，不要放入任何可以供其他人瀏覽的 **"程式碼"** 中，像是github、bitbucket等等。舉例:

```
server = None
server_init(server, 'my_server')
server_init_auth(server, 'my_account', 'my_password1234') # dangerous
```

Basic Rule #2 Keep an Eye on Resource Usage

"Pay as you go"是AWS的基本原則，釋放任何你不再需要用到的資源，並且到帳單儀表板檢視您的預估帳單。AWS也提供了 Monthly Calculator做為付費計算工具。

- Getting Start

1. 在IAM Console左方導覽列找到「Group」→「Create New Group」
2. 輸入group name→「Next Step」

3. Attach policy (permission) 屬於這個「群」的「使用者」都會擁有policy定義的權限。有許多預設policy可以選擇，但也能自訂一個policy

4. 「Next Step」→「Create Group」
5. 在IAM Console左方導覽列找到「Users」→「Add user」

6. 輸入user name、access type選"*Programmatic access*"→「Next: Permission」 這個選項代表 "此user將會透過程式碼來存取服務"
7. 「Add user to a group」→選擇剛才創建的group (將group的權限套用到新user上)→「Next: Review」

8. 「Final check」→「Create User」
9. 重要: 最後是一個可供下載 *Access key ID* 和 *Secret access key* 的頁面，當離開這個頁面後就再也無法下載，請妥善保存。

Step: Cognito

Amazon Cognito提供了 **Cognito User Pool, 使用者集區** 和 **Cognito Federated Identity Pool, 聯合身分集區**。簡單來說，user pool用來管理使用者的基本資料，例如登入用的使用者名稱、帳號密碼，或是電話號碼等等；而每個使用者的真正身分是在identity pool管理，這決定了某個使用者擁有哪些服務的受限存取權。

更進一步解釋的話，user pool是扮演一個使用者 "*提供者*"*(provider)* 的角色，而 federated identity pool則是將所有的 provider 提供的 user 做聯合的身分管理。因此，我們不僅能在AWS增加新使用者，還能透過第三方提供者，例如 Facebook、Google+ 等，來增加服務的使用者，就是基於這個機制。

- Getting start

1. 進入Amazon Cognito console → 「Manage User Pools」→「Create a user pool」→ 輸入Pool name，選擇「Review defaults」

2. 在左方選單選擇「Attibutes」→ 勾選「Email address or phone number」→開啟「Allow email addresses」讓使用者可以用email來驗證和登入

3. 在左方選單選擇「Review」→「Create pool」

其他設定:

Panel	Function
Attributes	• Username: 決定可以用什麼個人資料登入，例如電子郵件、電話、自定等等 • Standard Attributes: 決定註冊時需要哪些使用者基本資料(標準) • Custom Attributes: 自訂所需要的註冊基本資料
Policies	• Password Policy: 定義密碼強度需求 • Allow Sign-up: 使用者能自行註冊或只允許管理這增加使用者 • Expiration: 定義由管理員創建的帳號多久會過期
MFA and Verification	• Enable/Disable: 開啟/關閉多方驗證功能 • Verification Attributes: 決定驗證手段 • Verification Role: 定義執行驗證動作的執行角色
Message customizations	• Verification message: 自訂郵件驗證訊息 • Invitation message: 自訂郵件邀請訊息
Tags	在使用者集區加入標籤
Devices	決定是否儲存使用者存取服務的裝置
App clients	決定哪種應用客戶端可以存取此使用者集區
Triggers	• Pre Sign-up: 使用Lambda function 自訂註冊工作流程 • Pre Authentication: 使用 Lambda function 自訂認證(登入)工作流程 • Custom Message: 使用 Lambda function 自訂校驗或多方驗證訊息 • Post Authentication: 使用 Lambda function 自訂登入後的工作流程 • Post Confirmation: 使用 Lambda function 自訂校驗使用者後的工作流程 • Define Auth Challenge: 使用 Lambda function 初始化客製的認證流程 • Create Auth Challenge: 使用 Lambda function 創建認證時的 challenge，會在 "Define Auth Challenge" 後被呼叫 • Verify Auth Challenge Response: 使用 Lambda function 校驗使用者對 challenge 的回應 • User Migration: 使用 Lambda function 自訂登入或忘記密碼時的工作流程 • Pre Token Generation: 使用 Lambda function 自訂權杖的產生流程

4. 創建完成後頁面會自動跳轉到該集區的一般設定頁面，也能在Amazon Cognito主控台點選指定集區進入，並記住集區的「Pool id」

5. 在設定介面左方「General settings」中選擇「App client」→「Add an app client」

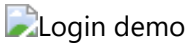
6. 輸入「App client name」和「Refresh token expiration(days)」。
7. 取消勾選「Generate client secret」，因為使用Javascript SDK不支援這種 client secret
8. 勾選「Enable sign-in API for server-based authentication (ADMIN_NO_SRP_AUTH)」，這讓我們可以通過 AWS CLI 管理使用者集區

9. 點擊「Create app client」並記住「App client id」

10. 進入Amazon Cognito console → Manage Federated Identities→「Create new identity pool」→ 輸入 Identity pool name
11. 在下方「Authentication Providers」區塊中選擇「Cognito」分頁，並輸入新創立的 User pool id 和 App client id

12. 點選「Create Pool」後會跳轉到 one-click role 頁面，展開「Hide Details」→「View Policy Document」可以檢視和編輯"驗證通過"和"驗證失敗"時將會套用的角色(政策) → 點選「Allow」產生這些角色

Addiotnal Topic #1: 使用 NodeJS 實現 Cognito 登入



`AWS Amplify`提供了實用的模塊讓開發者更容易實現 *授權存取資源* 這樣的應用模式。在這段主題會說明如何引入`AWS Amplify`模塊來實現讓使用者透過 Amazon Cognito 登入。關於 ReactJS的基本概念，請參考。

- Getting start

1. 使用 terminal 安裝必須的套件

```
npm install --save aws-amplify aws-amplify-react amazon-cognito-identity-js
```

2. 在`.js`檔中導入模塊，並在程式碼中設定好配置。

```
import Amplify, { Auth } from 'aws-amplify';
Amplify.configure({
    Auth: {
        mandatorySignIn: true,
```

```
        region: YOUR_COGNITO_REGION,
        userPoolId: YOUR_COGNITO_USER_POOL_ID,
        identityPoolId: YOUR_COGNITO_IDENTITY_POOL_ID,
        userPoolWebClientId: YOUR_COGNITO_APP_CLIENT_ID
    }
});
```

3. 準備登入用的component。帳號密碼的取得方式通常是靠圖形介面讓使用者輸入，在這裡先忽略。如果有需要帳號密碼以外的登入資訊需求，自行在程式碼和網頁介面中增加就可以了。

```
import React, { Component } from 'react';
class Login extends Component {
    constructor(props) {
        super(props);
        this.state = {
            username: '',
            password: '',
            cognito_user: null,
            cognito_session: null,
        }
    }

    signIn = async () => {
        try {
            let user = await Auth.signIn(this.state.email, this.state.password);
            this.setState( { cognito_user: user } );

            if (user.challengeName === 'SMS_MFA' ||
                user.challengeName === 'SOFTWARE_TOKEN_MFA') {
                // we won't handle this situation here.
            } else if (user.challengeName === 'NEW_PASSWORD_REQUIRED') {
                // we won't handle this situation here.
            } else if (user.challengeName === 'MFA_SETUP') {
                // we won't handle this situation here.
            } else {
                alert("Logged In!");
                let session = await Auth.currentSession();
                this.setState( { cognito_session: session } );
            }
        } catch (e) {
            alert(e.message);
        }
    }
}
```

在這段程式碼中我們做了幾件事:

- ☑ 初始化 component 狀態，在狀態中增加`username`、`password`、`cognito_user`、`cognito_session`四種屬性。`username` 和 `password`用來儲存從使用者介面或外部得到的帳號密碼，`cognito_user` 包含

一些使用者狀態，`cognito_session`包含一些用來存取其他 AWS 服務的憑證，例如 *JSON Web Token, JWT*。

- ☑ 加入一個可以被觸發的"**非同步函式**"，這個函式會呼叫`Auth.signIn()`這個方法。這個方法"承諾"一定會 (可能不是馬上) 回傳一個結果(請參考 Promise)。為了應付這種需要等待的狀況，可以透過`async`運算子去表達一個函式為非同步函式，被表達為"非同步"的函式能夠在內部使用`await`運算子，去等待承諾函式的回傳值。
- ☑ 根據`Auth.signIn()`的回傳值，我們可以知道這個使用者的驗證情形。在範例中我們只處理登入成功的情況，在登入成功後將使用者資訊和 session 更新到 component 的狀態中。

> 若是之後需要處理其他驗證結果，例如需要多方驗證、需要新密碼等等，同樣在這個函式中處理

Addiotnal Topic #2: 解決 "需要新密碼"

當使用者不是自己註冊帳號，而是由管理員在 Amazon Cognito User Pool 建立帳號，此時使用者用預設密碼登入後，仍然不能存取其他資源 (challengeName 回傳`NEW_PASSWORD_REQUIRED`)。以下是處理重設密碼的核心步驟:

1. 準備重設密碼的 component。為此，我們準備了一組新密碼和重複確認密碼，還要包含登入成功時的使用者物件，這在接下來會使用到。

```
import React, { Component } from 'react';
import Amplify, { Auth } from 'aws-amplify';
class ResetPassword extends Component {
    constructor(props) {
        super(props);
        this.state = {
            new_password: '',
            double_check_password: '',
            cognito_user: COGNITO_USER
        }
    }
}
```

2. 建立處理重設密碼的函式。

```
class ResetPassword extends Component {
    constructor(props) {
        // the same...
    }
    resetPassword = async () => {
        if (this.state.new_password === this.state.double_check_password) {
            try {
                Auth.completeNewPassword(
                    this.state.cognito_user,
                    this.state.new_password,
                    // optional parameter
                    // if your cognito user pool needs other information
                    // add them here
                    // example:
```

```
                    // { email: this.state.email, phone_number: this.state.phone }
                ).then(() => {
                    alert('Reset password successfully.');
                });
            } catch (e) {
                alert(e.message);
            }
        }
    }
}
```

這段程式碼宣告了一個非同步函式`resetPassword()`，當新密碼確認無誤後，呼叫一個`Auth.completeNewPassword()`方法來完成新密碼的設定。要取得這個方法的回傳值同樣需要等待一段時間，我們使用了另一種等待手段`then()`。`then()`需要帶入一個回呼函式 (callback) 作為參數，當前面的承諾函式完成時，觸發回呼函式的動作。

Step: ReactJS

在不使用ReactJS的情況下，要建立整個網頁應用程式需要兩個部分，**server(back-end)** 以及 **web page(front-end)**。Programmer編寫網頁內容的同時，也要使用script(javascript)來和伺服器(或任何外部資源)互動。ReactJS則是結合了這兩者，伺服器直接渲染出`DOM(Document Object Model)`，簡單來說就是 **伺服器直接提供html內容** 給使用者閱讀。當然，不使用ReactJS也能使伺服器直接提供內容，像這樣:

```
var AppendStr='';

AppendStr+='<div class="posit switch-fiel"
style="top:10px;left:200px;width:400px;"><fieldset class="setfieldset"
style="height:200px;"><legend>Login</legend>';
AppendStr+='<div id="msg" class="posit fontsize18 fontwidth700"
style="width:380px;top:30px;left:10px;color:#FF1C1C;"></div>';
// ...
$('#body').html(AppendStr);
```

ReactJS則是提供了較容易開發的套件給網頁開發人員。

```
import react, { Component } from 'react';
import ReactDOM from 'react-dom';

class App extends Component {
    render() {
        return (
            <div className="posit switch-fiel">
                <fieldset>
                    {'...'}
                </fieldset>
            </div>
        );
```

```
    }
}
ReactDOM.render(<App />, document.getElementById('root'));
```

- Getting start

> 本篇文章僅記錄環境架設流程與基本概念

1. 安裝 Node.js windows環境在安裝 Node.js時同時也會安裝 npm 其他環境可以參考 Node.js package manager
2. 安裝完成後可以透過 terminal 檢查是否安裝成功

```
cyp@cyp-virtual-machine:~$ node -v
cyp@cyp-virtual-machine:~$ npm -v
cyp@cyp-virtual-machine:~$ npx -v
```

> *npx comes with npm 5.2+ and higher, see instructions for older npm versions*

3. `create-react-app`，這個 sample 中的`package.json`能幫我們快速的建好 ReactJS project 環境。`package.json`中會記錄該安裝哪些套件，透過`npm install`指令可以一次性的全部安裝完畢 (關於npm 的其他指令用法，請參考)。請在terminal 執行這些指令 (關於其他透過 create-react-app 初始化專案的方法，請參考):

```
cyp@cyp-virtual-machine:~$ npx create-react-app my-app
cyp@cyp-virtual-machine:~$ cd my-app
cyp@cyp-virtual-machine:~$ npm install -dev @babel/core
```

4. ReactJS 開發時使用的 Javascript 版本為 **"ECMAScript 2015, ES6"** ，目前幾乎所有瀏覽器都已經支援 ES6 版本，更新的 ES7 目前只有 Chrome 和 Opera 瀏覽器支援。取代原本的`var package = require('package');`模塊引入方法，採用新的`import/export`來導入/導出 module。這邊舉出 ES6-Features 文件上的範例:

```
// ES6
//  lib/math.js
export function sum (x, y) { return x + y }
export var pi = 3.141593

//  someApp.js
import * as math from "lib/math"
console.log("2π = " + math.sum(math.pi, math.pi))

//  otherApp.js
import { sum, pi } from "lib/math"
console.log("2π = " + sum(pi, pi))
```

5. 在我們新建好的專案資料夾中，其目錄架構與用途:

```
my-app
├── README.md                專案文件
├── node_modules             使用 npm install 安裝產生的資料夾
├── package.json             儲存目前專案的基本資料
├── .gitignore               使用 git 做版本控制時忽略的檔案類型
├── public                   網站需要公開的檔案
│   ├── favicon.ico              網站的 icon
│   ├── index.html               網站的唯一頁面
│   └── manifest.json            網站基本資料
└── src                      網站的 source code
    ├── App.css                  開發 App.js 時需要的 css style
    ├── App.js                   主要選染的 App Component 程式碼
    ├── App.test.js              渲染 App 的測試程式
    ├── index.css                開發 index.js 時需要的 css style
    ├── index.js                 執行渲染動作的程式碼
    ├── logo.svg
    └── serviceWorker.js
```

概念一: Component

使用ReactJS開發的網頁會被分為好幾個 "*Component*"，通常以ReactJS開發的網頁應用只有一個 html 頁面，也只會有一個 Component 會被渲染在這個頁面上，如同 `example1.js`程式碼所表達的。

```
// example1.js
import react, { Component } from 'react';
import ReactDOM from 'react-dom';

class Component1 extends Component {
    render() {
        return (<div><h1>Hello</h1></div>);
    }
}

class Component2 extends Component {
    render() {
        return (<div><h2>World!</h2></div>);
    }
}

class App extends Component {
    render() {
        return (
            <div>
                <Component1/>
                <Component2/>
            </div>
        );
```

```
    }
}
ReactDOM.render(<App />, document.getElementById('root'));
```

雖然只有單一頁面，但可以根據狀況不同來顯示不同內容，這也牽涉到另一個基本概念 **state**。

概念二: State and Lifecycle

我們可以將 component 的狀態記錄在 **state** 物件裡面，並透過`setState()`方法去更新 state。透過 `setState` 去做狀態更新時，ReactDOM會計算需要改動的部分並自動重新渲染，可能是一行字、標籤頁的切換、或是刷新整個頁面。為了因應 state 的概念，component 的內部也提供了許多預設方法可供 overwrite，來客製當 component 將要改變、改變之後等等時機點的動作。`example2.js`示範了如何透過 state 改變網頁呈現的內容。

```
// example2.js
import react, { Component } from 'react';
import ReactDOM from 'react-dom';

const OnTitle = () => (<h2>Turn the Light on!</h2>);  // Arrow function
const OffTitle = () => (<h2>Turn the Light off!</h2>);

class App extends Component {
    constructor(props) { // lifecycle method
        // we can't user this here.
        super(props);
        // Don't use this.setState() to initiate state here!
        this.state = {
            on: false
        };
        this.updatePage = this.updatePage.bind(this);
    }
    componentDidMount() { // lifecycle method
        setTimeout(this.updatePage, 3000);
    }
    updatePage(){
        // Don't directly assign this.state after leaving constructor!
        this.setState( { on: !this.state.on } );
        setTimeout(this.updatePage, 3000);
    }
    render() { // lifecycle method
        return (
            <div>
                {(this.state.on) ? <OnTitle /> : <OffTitle />}
            </div>
        );
    }
}

ReactDOM.render(<App />, document.getElementById('root'));
```

這個範例用途是每隔三秒改變網頁一行字的簡單程式碼，但已經包含了一些ReactJS重要的概念，現在一一介紹

- **Arror function -** 另外一種宣告函式的方法，效果大致和一般的函式宣告相同。箭頭後面接大括號代表要函式內執行的程式碼，後面接小括號代表函式直接回傳的內容。

> 如果要渲染某個 function 或 component的回傳值，其名稱必須以大寫字母開頭

- **Lifecycle -** component的生命週期，代表當頁面發生某些事件時，哪些方法(*lifecycle method*)會被以怎樣的順序執行。範例中已經 overwrite 了一系列完整的「**安裝生命週期方法**」，一個安裝生命週期的方法執行順序如下: `constructor()`→`render()`→`componentDidMount()`。我們複寫這三個方法的內容為:
  - ☑ ***constructor()***
    1. 使用`super(props)`初始化`this`與`this.props`。ES6的`subclass`物件可以不需要覆寫constructor，一旦覆寫constructor，必須使用`super()`。props的概念將會在稍後提到。
    2. 初始化 state，為一個擁有的 "on" 屬性為 `true`的物件。**在`constructor()`中，直接 assign 狀態的值；離開`constructor()`後，使用`this.setState()`來更新狀態。**
    3. 將`updatePage()`函式與 component 綁定，否則`updatePage()`無法知道自己與component之間的關係，也就無法讀取到 state (*component 中的* ***Arrow function*** *會自動綁定*)。
  - ☑ ***render()***
    在`render()`方法中必須包含一個回傳值，並避免在`render()`方法中更新任何狀態。若要在html物件中夾帶程式碼(包含條件判斷、變數等等)，需要使用大括號來表達。此範例會在狀態不同的情況下選染不同的標題字串。
  - ☑ ***componentDidMount()***
    複寫此方法，使得在渲染完成後，使用`setTimeout()`來讓`updatePage()`於一段時間後執行。

概念三: Properties

既然 ReactJS 開發的網頁應用是由許多 component 組成，那麼 component 之間的訊息傳遞就是很重要的議題。在 ReactJS 中，我們會通過`Props`將 Parent component 的資訊傳給 child component。`example3.js`會示範如何使用`Props`

```
// example3.js
import react, { Component } from 'react';
import ReactDOM from 'react-dom';

// props will become parameter when child component is a function
const Child = (props) => (<h2>{props.text}</h2>);

// use this.props to access properties from parent component.
class Parent extends Component {
    constructor(props) {
        // we can't use this.props here.
        super(props);
    }
    render() {
        return (<div><Child text={this.props.text}/></div>);
    }
}
```

```
class App extends Component {
    render() {
        return (<div><Parent text="Text from Parent" /></div>);
    }
}
ReactDOM.render(<App />, document.getElementById('root'));
```

在example3.js中，透過props我們將 App component 中的變數傳給 Parent component，Parent component 再將變數傳給 Child function。幾乎所有東西都能使用props傳遞，例如我們也能將函式透過props傳給 child，並在child component/function 執行這個函式。

> 只要不去覆寫 constructor()，預設就能存取this.props
