小程序

前置步驟

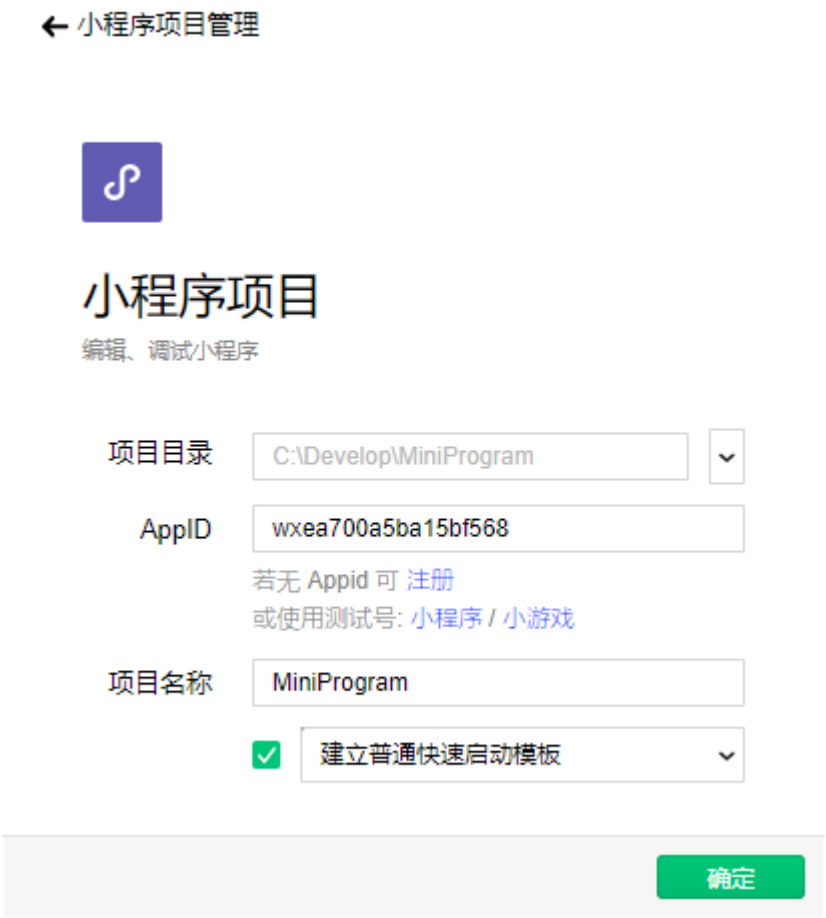
在實際開發前，首先需要擁有一個小程序帳號，小程序帳號與微信公眾平台上的其他服務（服務號、訂閱號、企業微信）視為不同帳號個體，因此就算已經擁有服務號，還是需要註冊一個新的帳號供小程序使用

完成註冊後，為了在後續開發流程方便，尚需要進行：

- 依據情況設置不同成員間的權限，只有**管理者**擁有該設置的權限（管理者預設為註冊時綁定的帳號），若需要提供體驗版給多人使用，也需要添加到成員權限中
- 生成小程序的密鑰 AppSecret，供後續身分驗證流程使用，只有**管理者**擁有生成密鑰的權限

開始開發小程序

開發小程序需要使用「微信Web開發者工具」來進行開發，至微信官網下載此開發工具。登入開發者身分，輸入小程序 AppId 後開始進行開發，登入者微信帳號需具備該小程序的開發權限才能開始開發，或是在 **AppID** 輸入項下方使用小程序的測試號



Hello world!

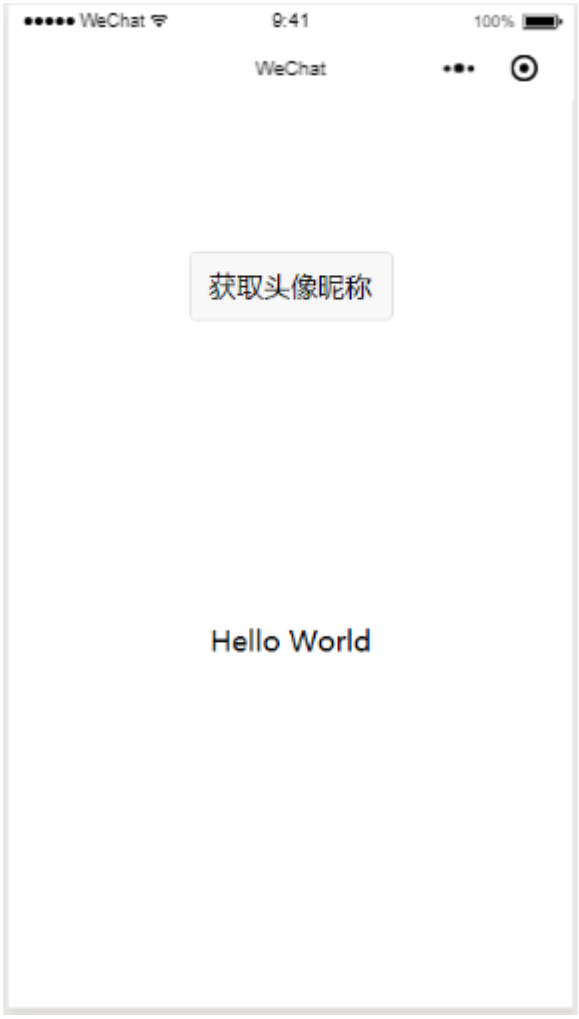
https://developers.weixin.qq.com/miniprogram/dev/framework/structure.html

建立一個新的小程序專案後，在中間的目錄瀏覽視窗可以看到如下的架構

目錄檔案的說明如下：

- pages/：存放各分頁內容的目錄，依據分頁可以再區分為多個資料夾目錄，一個分頁基本包含下列四個檔案
  - /*/*.js：頁面邏輯層，基底語言為 javascript，由 Page() 包覆註冊一個新的分頁
  - /*/*.wxml：頁面架構，WeiXin Markup Language，與 html 擁有類似的標籤結構
  - /*/*.wxss：頁面樣式，WeiXin Style Sheet，與 css 類似，可使用選擇器等功能
  - /*/*.json：頁面定義，與頁面配置相關的定義檔，此定義檔與 app.json 中的 window 結構相同，為頁面個別的定義內容
- utils/：工具模組等 .js 放置在此目錄下
- app.js：包含小程序初始化邏輯及全域變數等放置在此檔案中，由 App() 包覆註冊內容
- app.json：小程序的頁面全域配置，包含分頁、導覽按鈕等
- project.config.json：定義小程序 sdk 版本及相關編譯設置

新增的專案已經可以運行，在每次編輯儲存後，左側模擬器即可看到執行結果



示範專案 - 電商小程序

藉由建立一個簡單的電商頁面來了解小程序架構及機制，該小程序擁有幾個主要功能：

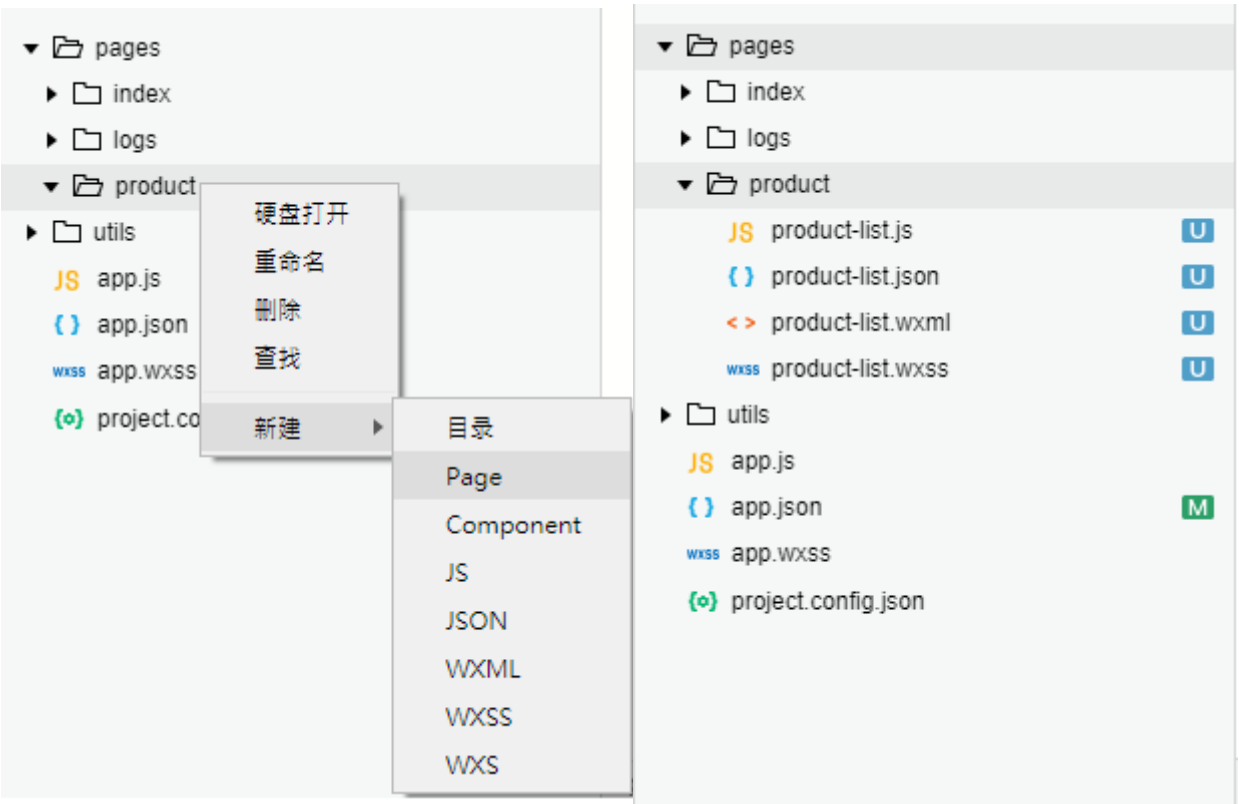
1. 瀏覽商品列表

2. 瀏覽商品詳細資料
3. 從商品資料導至官網介紹
4. 轉發商品消息給其他微信聯絡人

另外，由於連接到後台需要經過小程序域名驗證，故此專案僅專注在前端實做的部分，在一邊實做的同時，若是對於其他配置有興趣，也可以自行至小程序的文件查找並加到自己的專案中

新建頁面

延續新建的小程序，我們需要新增一個頁面來瀏覽商品，因此我們需要在 pages 目錄下新增一個 product 的資料夾來存放商品頁面，之後在 product 依序建立 .js .json .wxml .wxss 等頁面所需的檔案，也可以在目錄下右鍵新建 -> Page 來自動建立，這裡我們新建一個名稱為 product-list 的新頁面



當我們點開 app.json 可以發現，product-list 已經自動被加入到 pages 中

```
"pages": [
    "pages/index/index",
    "pages/logs/logs",
    "pages/product/product-list"
]
```

導覽列

接著，我們要將商品頁面加入到下方的導覽列中以供使用者切換

在 app.json 中，我們新增一個名為 tabBar 的欄位，在該欄位下包含一個 list 清單，藉由這個清單我們可以定義導覽按鈕的配置。需要注意的是，一旦定義了 tabBar，就必須要定義 list 清單，該清單至少需含 2 個項目，且最多不得超過 5 個

```
"tabBar": {
    "list": [
        {
            "pagePath": "pages/index/index",
            "text": "首頁"
        },
        {
            "pagePath": "pages/product/product-list",
            "text": "商品"
        }
    ]
}
```

完成以上設置，我們已經可以透過下方的導覽按鈕到我們剛剛新增的頁面了

為了要讓使用者開啟小程序之後可以直接進到我們商品頁面，在 pages 中，將 product-list 移至清單的第一個，之後開啟小程序都會直接導到商品頁

```
"pages": [
    "pages/product/product-list",
    "pages/index/index",
    "pages/logs/logs"
]
```

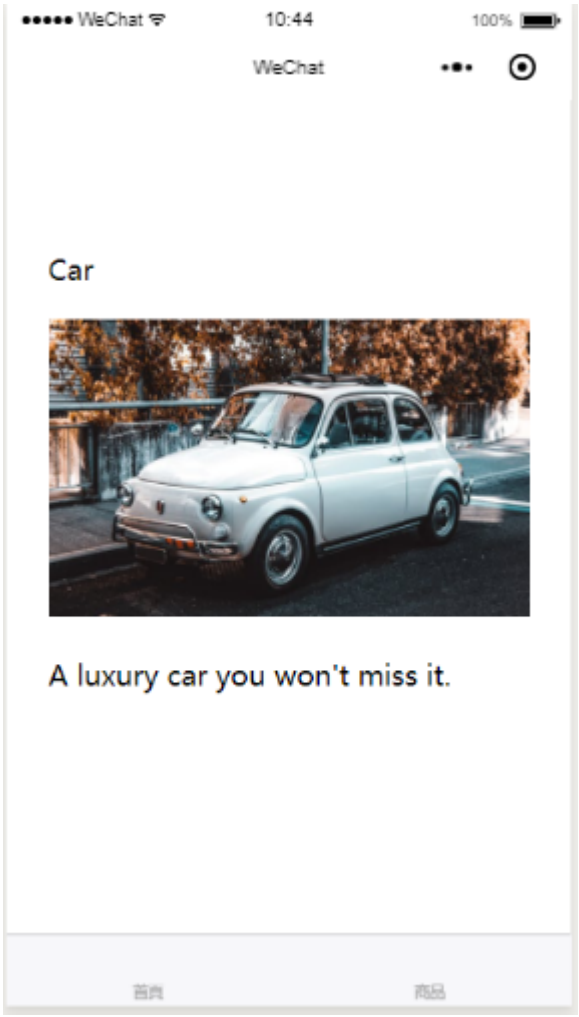
頁面架構 (WXML)

接著，讓我們來完善商品頁的內容。首先，定義頁面架構，開啟 /pages/product/product-list.wxml，將原有的內容刪除，改將下面的內容加入

> 關於 wxml tag 的詳細屬性，請參考小程序文件中的「組件」頁籤

```
<view class="container">
  <view class="product-container">
    <view class="product-title">
      <text>Car</text>
    </view>
    <view class="product-image">
       <image mode="aspectFit" src="https://images.pexels.com/photos/1200458/pexels-photo-
1200458.jpeg?auto=compress&cs=tinysrgb&h=350" />
    </view>
    <view class="product-info">
      <text>A luxury car you won't miss it.</text>
    </view>
  </view>
</view>
```

在模擬器中，你已經可以看到新的商品 - Car，出現在頁面上



讓我們再來新增一個商品，將類似的架構放在 <view class="container"> 之中，<view class="product-container"> 之下

```
<view class="product-container">
    <view class="product-title">
      <text>Bike</text>
    </view>
    <view class="product-image">
       <image mode="aspectFit" src="https://images.pexels.com/photos/1239460/pexels-photo-
1239460.jpeg?auto=compress&cs=tinysrgb&h=350" />
    </view>
    <view class="product-info">
      <text>A durable bike you've never ride.</text>
    </view>
</view>
```

在模擬器中，你已經可以看到兩個商品 Car, Bike 出現在頁面上

頁面樣式 (WXSS)

再來，我們稍微調整一下頁面的樣式，讓商品清單看起來更吸引注目，開啟 product-list.wxss 來加入以下內容

```
.product-container {
  width: 700rpx;
  margin: 5px;
  border: 1px solid #dddfe2;
  background: #e9ebee;
  -webkit-border-radius: 4px;
  -moz-border-radius: 4px;
  border-radius: 4px;
}

.product-title {
  font-size: 1.5em;
  text-indent: 0.5em;
  padding: 5px;
  border-bottom: 1px solid #dddfe2;
  -webkit-border-top-right-radius: inherit;
  -moz-border-top-right-radius: inherit;
  border-top-right-radius: inherit;
  -webkit-border-top-left-radius: inherit;
  -moz-border-top-left-radius: inherit;
  border-top-left-radius: inherit;
  background-color: #f6f7f9;
}

.product-image {
  text-align: center;
}

.product-info {
  padding: 10px;
  font-size: 1.1em;
  background-color: #f6f7f9;
  -webkit-border-radius: inherit;
  -moz-border-radius: inherit;
  border-radius: inherit;
}
```

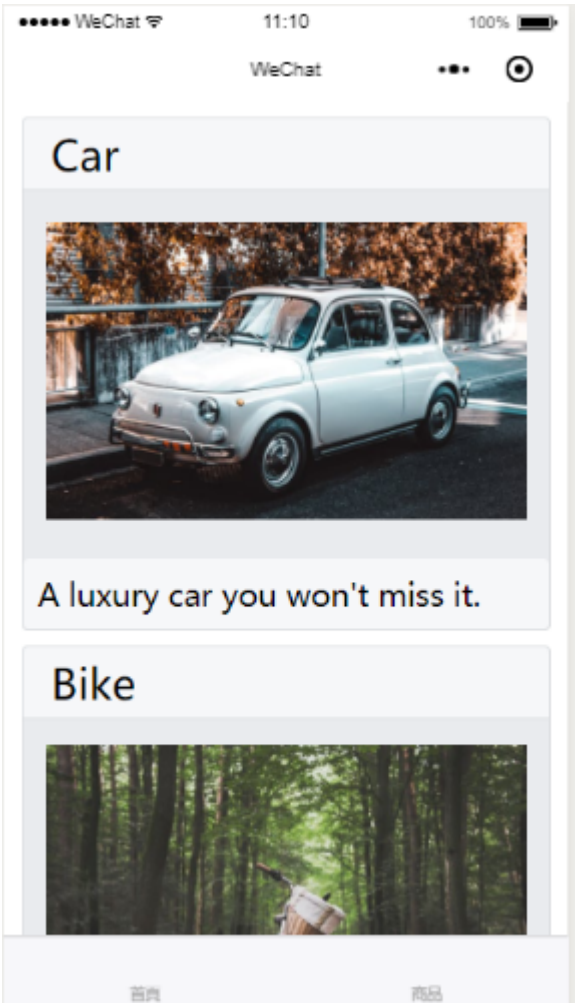
現在頁面看起來好多了，但是在頁面上方仍有一塊空白，這是因為我們在最外層的 view 套用了 container 的 class，而這個 class 被定義在 app.wxss 之中，讓我們來修改 app.wxss 的 .container 樣式，將 padding 改為

```
padding: 10rpx 0;
```

> rpx (responsive pixel) 為 wxss 用來因應響應式頁面新增的屬性，並定義螢幕寬為 750 rpx，詳細內容請參考小程序中「框架」頁籤的 視圖層 -> WXSS

之後在所有頁面中，只要套用了 conatiner class 都會套用這裡設定的屬性，由此來定義全域的樣式

完成的結果畫面如下



看起來我們已經完成了商品清單瀏覽的功能，但若是之後要再加入其他商品，我們就要再在 <view class="container"> 中加入新的內容，若是商品有 100 個，那麼這樣的動作就要重複 100 次，因此我們需要透過一些邏輯處理來簡化這樣的程序

我們可以觀察剛剛已經完成的 product-list.wxml，其中大部分的標籤內容是重複的，僅有少部分的資料因應不同商品而有所不同，如下

```
<view class="product-container">
    <view class="product-title">
      <text>商品名稱</text>
    </view>
    <view class="product-image">
       <image mode="aspectFit" src="商品圖片位址" />
    </view>
    <view class="product-info">
      <text>商品描述</text>
    </view>
  </view>
```

若是我們可以將商品資料獨立出來，再使用類似 for 的方法來一一讀取商品資料並放入相應的位置上，是不是就可以簡化 WXML 內容？

頁面資料處理

小程序的架構較貼近於 MVVM，將頁面所需資料獨立處理，另外再透過類似 two-way binding 的方式，將資料與頁面顯示同步，也因此，小程序無法也不需要在邏輯層操作 DOM

開啟 product-list.js 可以看見頁面相關配置都被包覆在 Page() 之中，在這裡我們先關注在 data 屬性上，data 定義的是頁面的初始資料，可以說需要與頁面顯示 binding 的資料都可以在此處定義

回到我們的需求上，我們把商品資料獨立出一個清單，然後將此清單放進 data 的 products 屬性之中，如下

```
Page({
    data: {
        products: [
          { productId: 1, name: 'Car', imgUrl: "https://images.pexels.com/photos/1200458/pexels-photo-1200458.jpeg?auto=compress&cs=tinysrgb&h=350", description: "A luxury car you won't miss it." },
          { productId: 2, name: 'Bike', imgUrl: "https://images.pexels.com/photos/1239460/pexels-photo-1239460.jpeg?auto=compress&cs=tinysrgb&h=350", description: "A durable bike you've never ride." }
        ]
    }
})
```

透過將 products 放進 data 中，我們已經將資料獨立在頁面顯示之外

接著，該如何將資料顯示在頁面上？首先，我們先修改一下 WXML 內容，如下

```
<view class="container">
    <view class="product-container">
        <view class="product-title">
            <text>{{ products[0].name }}</text>
        </view>
        <view class="product-image">
            <image mode="aspectFit" src="{{ products[0].imgUrl }}" />
        </view>
        <view class="product-info">
            <text>{{ products[0].description }}</text>
        </view>
    </view>
    <view class="product-container">
        <view class="product-title">
            <text>{{ products[1].name }}</text>
        </view>
        <view class="product-image">
            <image mode="aspectFit" src="{{ products[1].imgUrl }}" />
        </view>
        <view class="product-info">
            <text>{{ products[1].description }}</text>
        </view>
    </view>
</view>
```

原本 hard code 在上面的資料都被 double brace 包覆，且內容改為像是 javascript 物件實字的資料，利用 Mustache 語法，可以在頁面取用剛剛在 Page() data 定義的資料，於是我們已經可以將資料抽離，但這樣還是沒有解決原先的問題：當商品越多，重複的結構還是需要複寫多次。接著，我們可以用列表渲染 (wx:for) 來解決這個問題

列表渲染 (wx:for)

小程序的列表渲染會在編譯後，依據後方所接陣列的數量來重複產生列表渲染所包覆的結構，列表渲染的基本寫法如下

```
<view wx:for="{{array}}">
    <text>{{index}}: {{item.message}}</text>
</view>
```

- 陣列 index 預設透過 {{index}} 取得
- 陣列中每個項預設透過 {{item}} 取得

進階的寫法如下

```
<view wx:for="{{array}}" wx:key="key" wx:for-index="idx" wx:for-item="doc">
    <text>{{idx}}: {{doc.message}}</text>
</view>
```

- 自訂義 index 可透過 `wx:for-index` 指定
- 自訂義陣列項可透過 `wx:for-item` 指定
- `wx:key` 用來指定每一個項的唯一識別內容，通常為項目的 id，或是使用保留字 *this。當陣列會動態加減項目時，藉由指定 `wx:key` 來避免同樣的項目重複渲染，以保持原本的狀態及提高渲染效率

> 如果沒有指定 wx:key，console 會跳出警告，若確定該陣列為靜態，或者不需要保留其狀態時，可以忽略該警告
>
> 更詳細的用法參考小程序文件「框架」中的 視圖層 -> WXML 列表渲染，另外還有條件渲染 wx:if ，在此不加以敘述

我們再次修改原本商品清單的寫法如下

```
<view class="container">
  <view wx:for="{{products}}" wx:key="id">
    <view class="product-container">
      <view class="product-title">
        <text>{{item.name}}</text>
      </view>
      <view class="product-image">
        <image mode="aspectFit" src="{{item.imgUrl}}" />
      </view>
      <view class="product-info">
        <text>{{item.description}}</text>
      </view>
    </view>
```

```
  </view>
</view>
```

到這裡，我們已經完成了簡單的商品列表瀏覽頁面，目錄結構

```
pages
-- index
---- index.js
---- index.json
---- index.wxml
---- index.wxss
-- logs
---- logs.js
---- logs.json
---- logs.wxml
---- logs.wxss
-- product
---- product-list.js
---- product-list.json
---- product-list.wxml
---- product-list.wxss
-- utils
---- util.js
app.js
app.json
app.wxss
project.config.json
```

新建頁面

接下來我們要實作商品詳細資料瀏覽頁面，透過在商品列表瀏覽頁面點擊個別商品可以到該商品的詳細介紹，但我們不需要將此頁放在下方的導覽按鈕，因此我們再做一次新增頁面的步驟，這一次直接將頁面路徑加在 app.json 的 pages 下，開發者工具會自動將我們需要的檔案新建至對應的位置

```
"pages": [
    "pages/product/product-list",
    "pages/product/product-info",
    ...
]
```

檢視頁面跳轉的步驟

1. 在商品列表頁，偵測使用者點擊商品
2. 將頁面導致商品詳細頁
3. 呈現商品詳細資訊

按照步驟，要偵測使用者點擊商品，我們需要先了解「事件」

事件

事件是頁面與邏輯層溝通的管道，當使用者觸發事件時，將資料傳送至邏輯層進行處理

我們再次修改 WXML 如下

```
<view class="container">
  <view wx:for="{{products}}" wx:key="id">
    <view class="product-container" bindtap="onProductTap">
      <view class="product-title">
        <text>{{item.name}}</text>
      </view>
      <view class="product-image">
        <image mode="aspectFit" src="{{item.imgUrl}}" />
      </view>
      <view class="product-info">
        <text>{{item.description}}</text>
      </view>
    </view>
  </view>
</view>
```

在 product-container 加入 bindtap 屬性，並將此屬性值指定為 onProductTap，如此是將頁面點擊事件綁定到 product-container 之中，當使用者點擊 product-container 元素時，即會觸發點擊事件，該事件會呼叫 onProductTap 方法

因此需要再將 onProductTap 加入至頁面方法中，在 Page() 所傳入的物件中，新增 onProductTap 屬性，並定義其方法內容

```
Page({
    data: {
        ...
    },
    ...,
    onProductTap: function (e) {
        console.log(e)
    }
})
```

當我們點擊商品時，console 就會顯示該事件物件

```
▶ {type: "tap", timeStamp: 1252, target: {…}, currentTarget: {…}, detail: {…}, …}    product-list.js? [sm]:71
```

雖然已經偵測到按鈕事件，但我們還無法區分該事件是由哪個元素所觸發，因此我們需要將參數傳入到事件中，透過 data-* 可以自訂參數名稱，這裡我們定義參數名稱為 product，並將商品資訊傳入

```
<view class="product-container" bindtap="onProductTap" data-product="{{item}}">
     ...
</view>
```

再透過 event.currentTarget.dataset 取用傳入的參數，我們在上面定義名稱為 product，因此取用 dataset 的 product 來取得傳入的商品資訊

```
onProductTap: function (e) {
    console.log(e.currentTarget.dataset.product)
}
```

如此就可以順利偵測使用者點擊事件並且得知使用者點擊的是哪個商品

> 事件詳細內容請參考小程序文件「框架」中的 視圖層 -> WXML 的事件

頁面導向

我們沒辦法將所有頁面都可以定義在下方的導覽列，因此大部分的頁面導向都要透過 navigator 組件或是邏輯層來處理

接續步驟 2：將使用者導向商品詳細的頁面，我們需要透過小程序 api： wx.navigateTo() 來處理頁面導向，修改 onProductTap 方法

```
onProductTap: function (e) {
    wx.navigateTo({
      url: 'product-info'
    })
}
```

到此，我們已經可以在使用者點擊商品後導向到商品詳細頁，但我們還是不知道導了哪一個商品資訊過去，因此需要將該商品的資訊附加在 url 的 query string 上

```
onProductTap: function (e) {
    let product = e.currentTarget.dataset.product

    wx.navigateTo({
        url: `product-info?
id=${product.id}&name=${product.name}&description=${product.description}&imgUrl=${product.imgUrl
}`
    })
}
```

注意這裡有兩處使用 ES6 規範： `let` 與 template literals (``)

生命週期函數－監聽頁面加載

接著進行步驟 3：顯示頁面資訊

剛剛我們已經將產品資料藉由 url query string 傳送到商品詳細頁 product-info，接著要做的事情就是將 query string 的內容取出並呈現在頁面上，在 Page() 傳入的物件中有一個 onLoad 方法，當頁面加載時觸發該方法，藉由這個方法中的 options 參數，可以取得 query string

```
Page({
    ...
    onLoad: function (options) {
        console.log(options)
    },
    ...
})
```

在 console 中我們可以看到剛剛傳入的 query string

```
product-info.js? [sm]:18
▶ {id: "1", name: "Car", description: "A luxury car you won't miss it.", imgUrl:
"https://images.pexels.com/photos/1200458/pexels-photo-1200458.jpeg"}
```

在將這些資料綁定到 data 上，我們就可以在頁面上取得這些資料

```
Page({
    data: {
        id: 0,
        name: '',
        imgUrl: '',
        description: ''
    },
    onLoad: function (options) {
        this.setData({
            id: +options.id,
            name: options.id,
            imgUrl: options.imgUrl,
            description: options.description
        })
    },
})
```

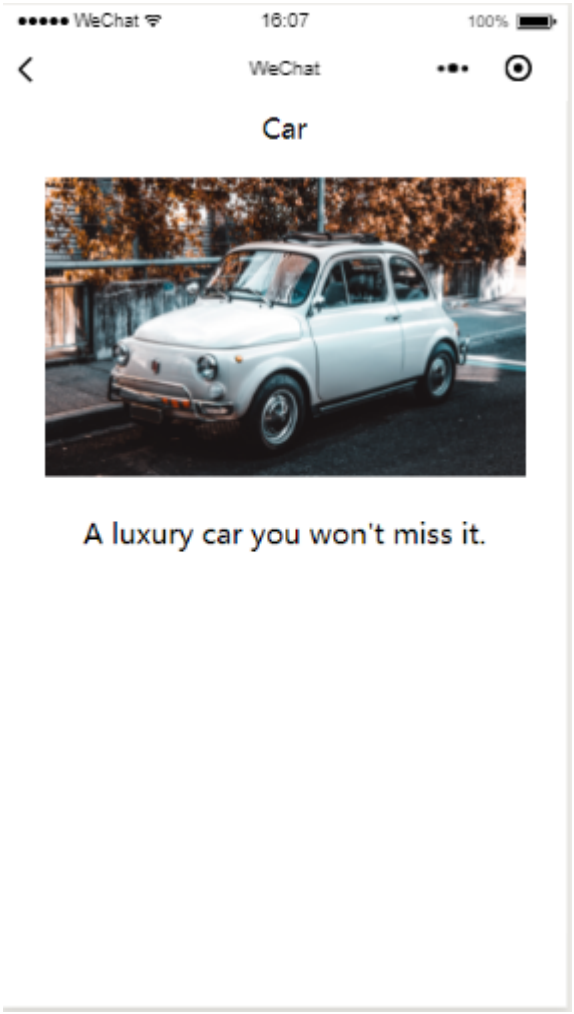
記得 data 內的屬性皆需要賦予初始值，且後續賦值不能為 undefined

> 注意在設定 data 時，id 屬性的值有一個 + 號，可以觀察有沒有加的差別

我們簡單地刻一下商品詳細頁面內容

```
<view class="container">
  <view>
    {{name}}
  </view>
  <view>
    <image mode="aspectFit" src="{{imgUrl}}"></image>
  </view>
  <view>
    <text>{{description}}</text>
  </view>
</view>
```

呈現結果如下



> 這裡要注意，因為我們沒有將 product-info 定義在 tabBar 中，在這個頁面看不到導覽列出現在下方，需要透過左上角的 < 來返回上一頁面

到目前為止，我們已經完成了這個示範專案的 1. 瀏覽商品列表 及 2. 瀏覽商品詳細資料，接著我們將繼續實作 3. 從商品資料導至官網介紹 與 4. 轉發商品消息給其他微信聯絡人

目錄結構

```
pages
-- index
---- index.js
---- index.json
---- index.wxml
---- index.wxss
-- logs
---- logs.js
---- logs.json
---- logs.wxml
---- logs.wxss
-- product
---- product-info.js
```

```
---- product-info.json
---- product-info.wxml
---- product-info.wxss
---- product-list.js
---- product-list.json
---- product-list.wxml
---- product-list.wxss
-- utils
---- util.js
app.js
app.json
app.wxss
project.config.json
```

Web View

透過 web view 來顯示網頁內容

小程序提供 <web-view> 來顯示網頁內容，唯此網域需要在小程序後台先通過業務域名驗證

新增一個頁面用來承載 web view 內容

```
"pages": [
    "pages/product/product-list",
    "pages/product/product-info",
    "pages/web-view/web-view",
    ...
]
```

先在商品詳細 product-info 新增一個按鈕來將導到剛剛新增的頁面

```
<view class="container">
    ...
    <view style="padding-top: 10px">
        <button type="primary" bindtap="onViewDetail">Detail</button>
    </view>
</view>
```

```
Page({
    ...,
    onViewDetail: function () {
        let url = encodeURIComponent(`https://www.google.com/search?q=${this.data.name}`)

        wx.navigateTo({
            url: `../web-view/web-view?navigateTo=${url}`
        })
    }
})
```

再來，簡單修改 web-view 內容如下

```
<web-view src="{{navigateTo}}"></web-view>
```

```
Page({
    data: {
        navigateTo: ''
    },
    onLoad: function (options) {
        this.setData(options)
    },
    ...
})
```

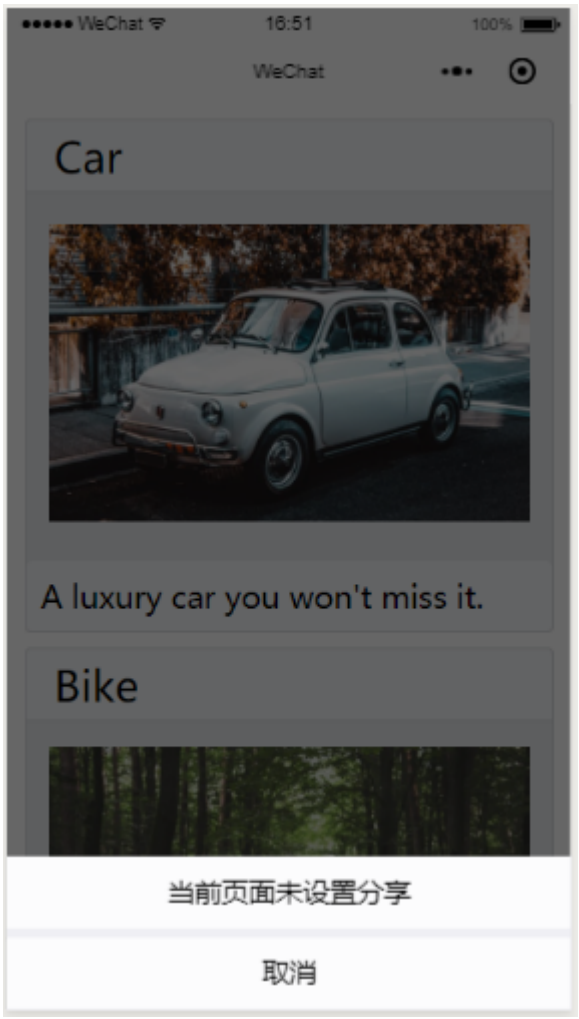
當使用者點擊 Detail 按鈕時，即會將頁面導至 web-view 頁面，並透過 query string 將欲查詢的網址填入到 <web-view> 並開啟網頁

轉發事件

最後一個功能，使用者可以透過轉發，將頁面分享給對話對象

首先，在欲提供轉發功能的頁面 .js 上，加入 onShareAppMessage 方法，如此即可在使用者點擊右上角的功能按鈕時，找到轉發的功能

```
Page({
    ...,
    onShareAppMessage: function () {}
})
```



onShareAppMessage 需要 return 一個物件，物件的屬性包含

- title：轉發的標題
- path：轉發到聊天視窗中，當使用者點擊轉發內容時會導向的位址
- imageUrl：轉發內容的圖片

我們在商品詳細中定義 onShareAppMessage 如下

```
onShareAppMessage: function () {
    let product = this.data

    return {
        title: `商品資訊 - ${product.name}`,
        path: `/pages/product/product-info?
id=${product.id}&name=${product.name}&description=${product.description}&imgUrl=${product.imgUrl
}`,
        imageUrl: product.imgUrl
    }
}
```

轉發後，在聊天對話視窗即會看到下圖的內容

目錄結構

```
pages
-- index
---- index.js
---- index.json
---- index.wxml
---- index.wxss
-- logs
---- logs.js
---- logs.json
---- logs.wxml
```

```
---- logs.wxss
-- product
---- product-info.js
---- product-info.json
---- product-info.wxml
---- product-info.wxss
---- product-list.js
---- product-list.json
---- product-list.wxml
---- product-list.wxss
-- web-view
---- web-view.js
---- web-view.json
---- web-view.wxml
---- web-view.wxss
-- utils
---- util.js
app.js
app.json
app.wxss
project.config.json
```

到此，我們已經完成此專案的"基本"功能開發，若要在手機上運行測試，點擊開發者工具上方的「預覽」按鈕即會產生一組 QR Code，透過 微信 / 企業微信 掃描此 QR Code 即可在手機上運行

小程序中還有許多運用，包括 頁面模組化、wxml 樣板、以及多樣的 api ... 等，另外，雖然目前 微信 / 企業微信 都可以使用小程序，但開發時須注意兩方調用的 api 有所不同，需要依據使用者的系統資訊來決定調用哪個 api，更多資訊請參考小程序官方文件

審核與發佈

在完成了小程序的開發及測試調校後，若要讓使用者可以使用小程序，還需要到小程序後台進行審核與發佈

首先，在開發工具的功能按鈕點擊「上傳」將開發版本上傳到小程序程式庫（若使用測試 AppId 則沒有此按鈕）

上傳成功後，在小程序後台 -> 開發管理 最下方可以看到新上傳的開發版本資訊

在確認沒有問題之後即可提交審核，審核後便可發佈上線