非同步程式設計

kevin

非同步程式的使用情境

CPU密集的複雜運算

將阻塞運算放到背景處理(e.g:IO處理)

日常生活中的非同步事件

你在打英雄聯盟

媽媽叫你買醬油

這個時候你會回答什麼

好

一分鐘後....

媽媽:快點去買醬油

二分鐘後....

媽媽:快點去買醬油

(ノ °Д°)ノ ┻━┻

```
using System;
using System.Threading;

public class Program{
    private static bool 去買醬油了 = false;
    private static int 等待分鐘數 = 1;

    public static void Main(){
        印出旁白("媽媽：幫我買醬油");
        印出旁白("我：好");
        準備在打完英雄聯盟後買醬油();
        媽媽催我去買醬油();
        印出旁白("……");
        印出旁白("(╯°Д°)╯ ┻┻");
    }


    private static void 印出旁白(string 旁白){
        Console.WriteLine(旁白);
    }

    private static void 準備在打完英雄聯盟後買醬油(){
        new Thread(去買醬油).Start();
    }

    private static void 媽媽催我去買醬油(){
        while(!去買醬油了){
            Console.WriteLine(string.Format("{0}分鐘後....",等待分鐘數));
            Console.WriteLine("媽媽:快點去買醬油");
            Thread.Sleep(等待分鐘數* 1000);//以秒數取代分鐘數
            等待分鐘數 += 1;
        }
    }

    private static void 去買醬油(){
        Thread.Sleep(2500);
        去買醬油了 = true;
    }
}
```

```
媽媽：幫我買醬油
我：好
1分鐘後....
媽媽:快點去買醬油
2分鐘後....
媽媽:快點去買醬油
……
(╯°Д°)╯ ┻┻
```

為什麼程式碼是中文的

為了服務台下非RD的聽眾，小弟把程式碼寫成中文的了

程式碼是真的可以跑的

請不要在實際專案中使用中文來撰寫程式碼

Polling

非同步程式設計中的一種技巧

-反覆檢查是否有結果以後再取得資料

能不能讓媽媽不要一直催

跟媽媽說忙完手邊的事情後就去買

```
using System;
using System.Threading;

public class Program{
    private static bool 去買醬油了 = false;
    private static int 等待分鐘數 = 1;

    public static void Main(){
        印出旁白("媽媽：幫我買醬油");
        準備在打完英雄聯盟後買醬油(new 通知媽媽要買去買醬油的動作());
        過場旁白();
    }

    private static void 印出旁白(string 旁白){
        Console.WriteLine(旁白);
    }

    private static void 準備在打完英雄聯盟後買醬油(通知媽媽要買去買醬油的動作 動作){
        印出旁白("我：好，我忙完去買");
        ParameterizedThreadStart 通知買醬油的參數 = new ParameterizedThreadStart(去買醬油);
        new Thread(通知買醬油的參數).Start(動作);
    }

    private static void 過場旁白(){
        while(!去買醬油了){
            Console.WriteLine(string.Format("{0}分鐘後....",等待分鐘數));
            Thread.Sleep(等待分鐘數* 1000);//以秒數取代分鐘數
            等待分鐘數 += 1;
        }
    }

    private static void 去買醬油(object 動作){
        Thread.Sleep(2500);//誰知道你要打多久啊！
        ((通知媽媽要買去買醬油的動作)動作).跟媽媽說我要去買醬油();

    }

    private class 通知媽媽要買去買醬油的動作{
        public void 跟媽媽說我要去買醬油(){
            印出旁白("我：我要去買醬油了");
            去買醬油了 = true;
        }
    }
}
```

```
媽媽：幫我買醬油
我：好，我忙完去買
1分鐘後....
2分鐘後....
我：我要去買醬油了
```

Callback

提供呼叫者處理結果的接口

在完成非同步呼叫後

直接呼叫實作出來的接口

Take away

~~媽媽叫你去買醬油但是你在忙的時候記得跟媽媽說忙完會去買~~

處理非同步作業常見的其中兩種方法

-Polling

-Callback

Callback Hell(俗稱波動拳)

```
function hell(win) {
 // for listener purpose
 return function() {
   loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
     loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
       loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
         loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
           loadLink(win, REMOTE_SRC+'/lib/underscode.min.js', function() {
             loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
                     async.eachSeries(SCRIPTS, function(src, callback) {
                       loadScript(win, BASE_URL+src, callback);
                     });
                   });
                 });
               });
             });
           });
         });
       });
     });
   });
 };
}
```

還有一種方法-等待直到完成

主執行緒先發出非同步請求，在需要非同步請求的結果時再呼叫取得方法，

若結果還沒回來則阻塞執行緒直到完成。

-媽媽把菜炒完了才開始跟你要醬油

JAVA上的非同步處理API-Future API

Future Interface

一個等待結果的容器

提供阻塞取得結果的方法

Future API

java.util.concurrent

Interface Future<V>

Type Parameters:

V - The result type returned by this Future's `get` method

All Known Subinterfaces:

Response<T>, RunnableFuture<V>, RunnableScheduledFuture<V>, ScheduledFuture<V>

All Known Implementing Classes:

ForkJoinTask, FutureTask, RecursiveAction, RecursiveTask, SwingWorker

Future API X 等待直到完成

```
Future<Result> asyncResult = doSomeAsyncTask();
try {
    doSomething();
    Result result = asyncResult.get();//阻塞直到完成
} catch (InterruptedException e) {
    Thread.currentThread().interrupt();
    e.printStackTrace();
} catch (ExecutionException e) {
    e.printStackTrace();
}
```

Future API X Callback

```
public class MyFutureTask extends FutureTask<Main.Result> {

    public MyFutureTask(Callable<Main.Result> callable) { super(callable); }

    @Override
    public Main.Result get() throws InterruptedException, ExecutionException {
        return super.get();
    }

    @Override
    protected void done() {
        super.done();
        //callback after async work finish
    }
}
```

組合相依的非同步作業

修改已經寫好的非同步作業並開Callback

-寫不好會有Callback hell

多開一條執行緒做polling解決同步問題

-浪費資源

CompletableFuture

Java 8 開始支援

實作Future介面

更強大的非同步處理接口

```java
final CompletableFuture<Integer> f = compute();
f.thenCompose(new Function<Integer, CompletionStage<Integer>>() {
    @Override
    public CompletionStage<Integer> apply(Integer integer) {
        return CompletableFuture.supplyAsync(new Supplier<Integer>() {
            @Override
            public Integer get() {
                try {
                    Thread.sleep( 1: 3000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("task 1 integer:"+integer);
                return integer + 1;
            }
        });
    }
}).thenCompose(integer -> CompletableFuture.supplyAsync(() -> {
    try {
        Thread.sleep( 1: 1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("task 2 integer:"+integer);
    return integer + 1;
}));
f.complete( t: 0);
```

```
/usr/local/android-studio/jre/bin/java ...
task 1 integer:0
task 2 integer:1
```

Promise pattern

在不同程式語言中有不同的實作

（e.g:Java 中的future api、JavaScript中的promise、C++中的std::future….）

Promise物件代表

-非同步完成的計算結果

-承諾會在未來提供結果，無論成功或是失敗

來聊聊JavaScript

在瀏覽器環境下(2017/10/16)

除非特別開Worker, 不然JavaScript都是單執行緒在跑的

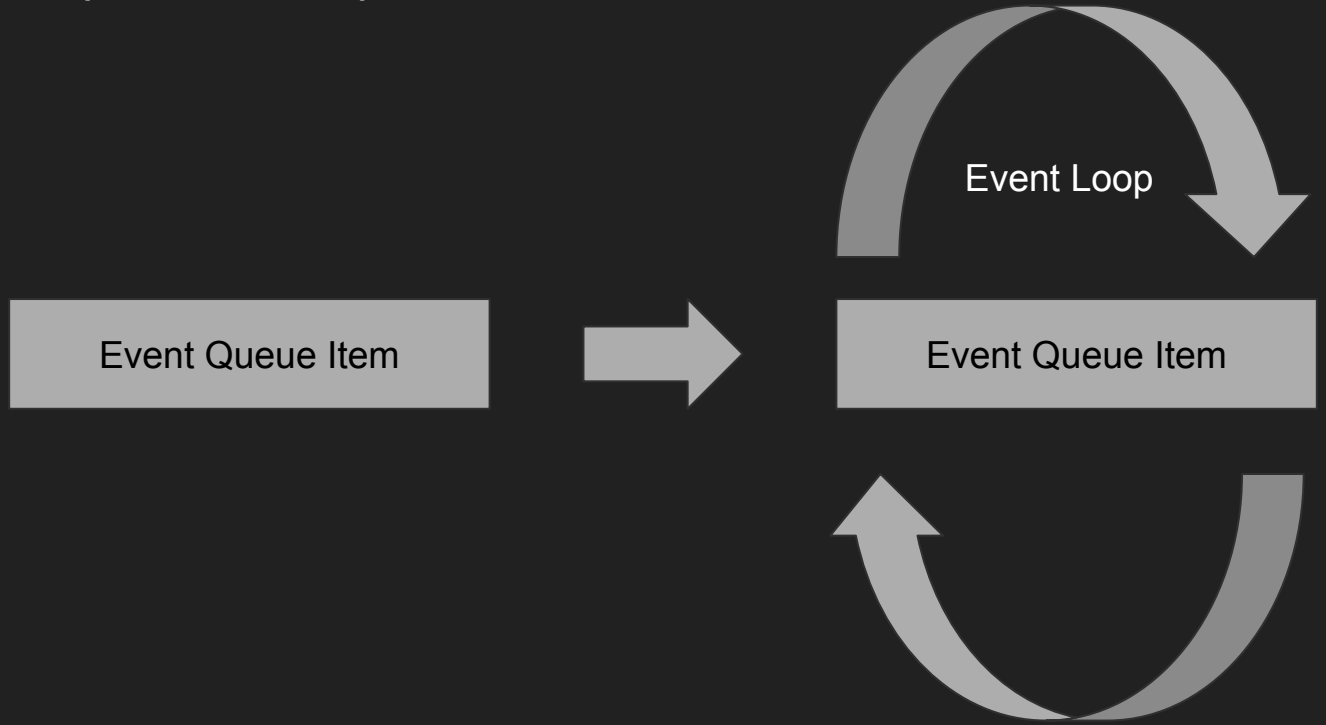
非同步的處理(e.g:IO存取)用的是同一條執行緒

JavaScript ▾

```
setTimeout(function () {
  console.log("1");
    isEnd = false;
}, 1000);
 while (isEnd);
 console.log("2");
```

Event Driven Model

透過Event loop以及Event queue來處理非同步的運算



JavaScript Library中經典的Promise實作

jQuery.Deferred()

jQuery 1.5中實作

提供一個Deferred物件來處理非同步請求

提供三種狀態

-pending（未解決）

-resolved（已解決）

-rejected（已拒絕）

jQuery.Deferred()

常用的幾個接口

deferred.done(callback) // 成功時執行

deferred.fail(callback) // 失敗時執行

deferred.always(callback) // 無論成功或失敗都會執行

Promise vs Deferred in jQuery

Deferred 提供了更改狀態的接口，如

-deferred.resolve()//標記非同步請求成功

-deferred.reject()//標記非同步請求失敗

Promise則只能接受Callback不能更改狀態

Promise要從Deferred中取得

-$.Deferred().promise()

jQuery下的多非同步請求組合

```
$.when($.ajax(url1),$.ajax(url2))
.done(function(result){
  //success callback
})
.done(function(result){
  //callback can be chainable
})
.fail(function(result){
  //fail callback
})
.always(function(result){
  //callback if success or not
});
```

JavaScript中原生的Promise實作

ES6特性

Promise/A+ 標準

=>狀態只會發生一次變化

=>從pending(等待中) 變成 fulfilled(已實現)或rejected(已拒絕)

JavaScript Promise 支援度

```
var getAsyncTask = function(){
  return new Promise(function (onFulfilled,onRejected){
     setTimeout(function(){
       onFulfilled('success');
     },2000);
  });
};
getAsyncTask().then(function(value){
  //on fulfillment(已實現時)
  console.log(value);
},function(){
  //on rejection(已拒絕時)
}).catch(function(err){
  //on catch error(已拒絕時)
});
console.log('main thread!');
```

```
"main thread!"
"success"
```

Async / Await

新的非同步處理方法

部份程式語言有實作（C#、JavaScript…..）

Async / Await in JS

```
var getAsyncTask = function(){
  return new Promise(function (onFulfilled,onRejected){
     setTimeout(function(){
       onFulfilled('success');
     },2000);
  });
};

async function main()
{
    const x = await getAsyncTask();
    console.log(x);}
main();
console.log('main thread!');
```

```
var getAsyncTask = function(){
  return new Promise(function (onFulfilled,onRejected){
     setTimeout(function(){
       onFulfilled('success');
     },2000);
  });
};
getAsyncTask().then(function(value){
  //on fulfillment(已實現時)
  console.log(value);
},function(){
  //on rejection(已拒絕時)
}).catch(function(err){
  //on catch error(已拒絕時)
});
console.log('main thread!');
```

```
"main thread!"

"success"
```

Thank you