巨匠線上真人

Java Web OCE JWCD元件系統開發認證

巨匠線上真人

Java Web OCE JWCD元件系統開發認證

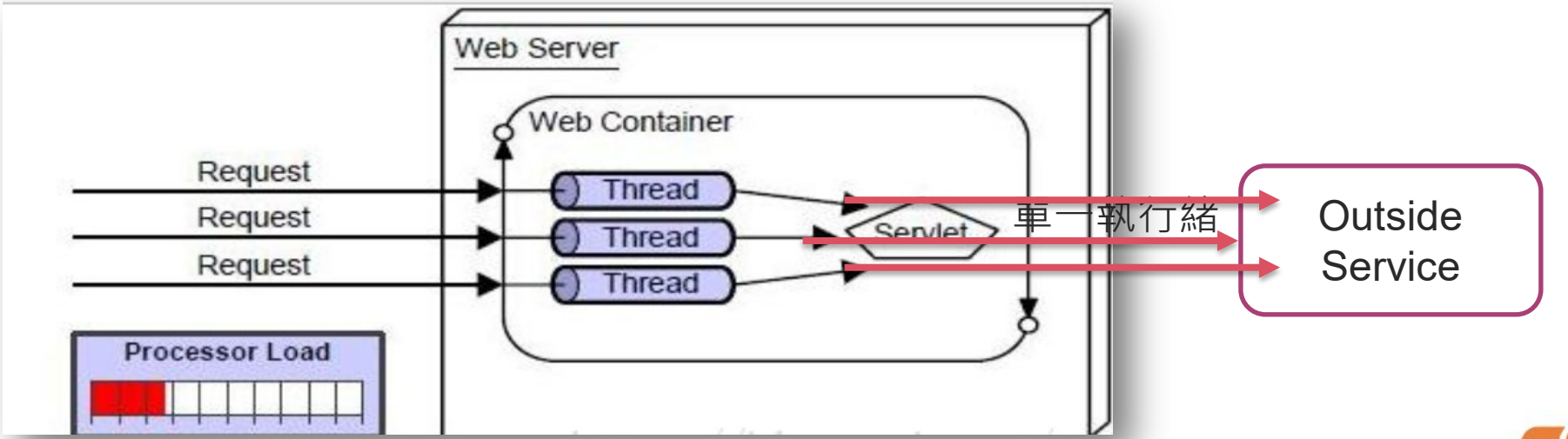
第六堂 Java網站非同步處理機制

本堂教學重點

- Servlet共用下的多工處理架構
- AsyncContext非同步處理應用架構
- 呼叫外部服務整合時，需要非同步處理因應
- Filter非同步處理架構

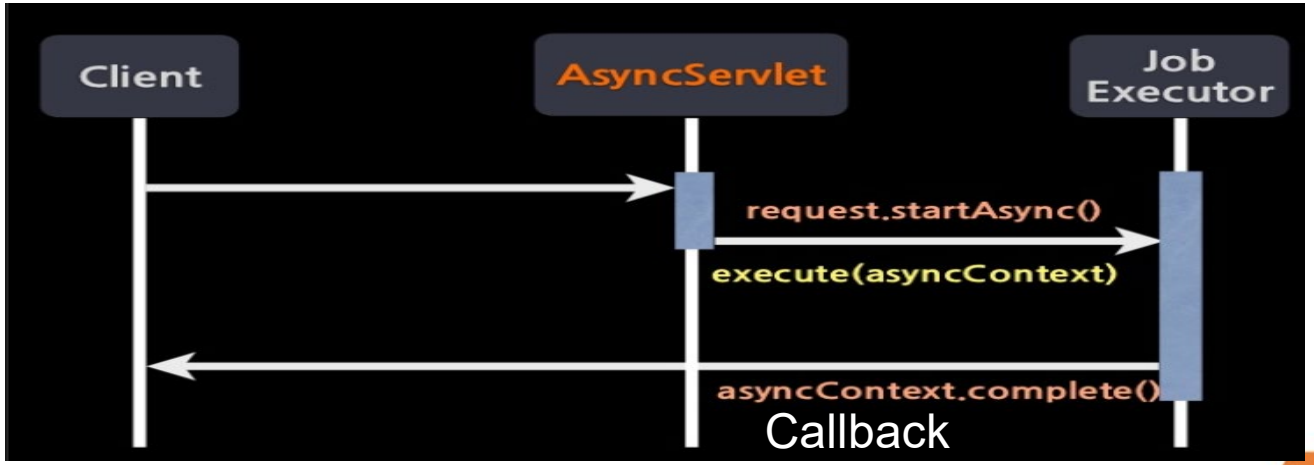
Servlet共用下的多工處理架構

- 在傳統JavaEE容器中，每一個對Servlet的請求，透過service()方法，會被分派一條執行緒(Thread)負責處理，直到處理完畢為止。
- 當處理程序時間過長時，同一個來源後面的請求就會被《卡住》，需要等待前面的執行完成之後，才會釋放出來。



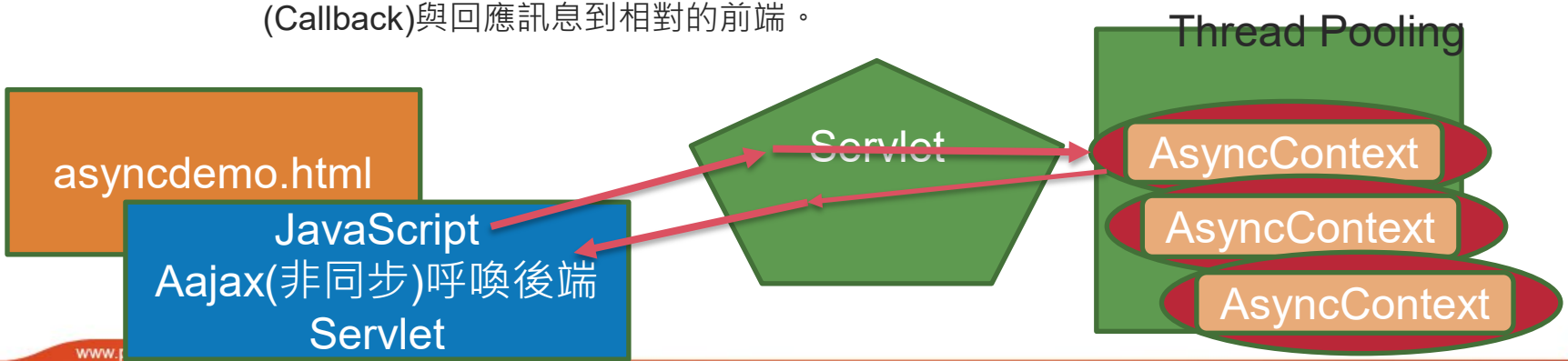
採用非同步處理機制

- 在Servlet 3.0提供AsyncContext介面，調用非同步處理，達到併行多工作業。
- 可以讓Servlet在因應前端請求下，進行併行(非同步處理)，呼喚較長時間或者外部處理的模組程序。
- 並且需要結合回呼程序(Callback)架構，將非同步處理的結果進行回呼與處理，且將處理的結果回應道相對的前端。



AsyncContext調用架構

- 在Servlet 3.0中可以調用AsyncContext中的
  - startAync() (in ServletRequest)
    - 立即將容器所提供的執行緒釋放(進入非同步處理)
    - 以便服務下一個請求
    - 取得AsyncContext物件
      - 原本的請求透過取得AsyncContext物件，保留了當時ServletRequest與ServletResponse物件，所以可以將客戶端的請求回應暫緩(背後處理)，直到調用complete() Method進行回呼(Callback)與回應訊息到相對的前端。



Runnable類別設計

- 用來委派一個執行緒背後的Task
- 透過傳遞進來的AsyncContext執行在處理完成程序之後，使用complete()完成作業，並且進行Callback程序，將相關的訊息回應到前端去。
- 執行AsyncContext.complete()確認非同步處理完成，同時回呼訊息到前端去。

AsyncRequest.java

```
import java.io.PrintWriter;
//實作Runnable
public class AsyncRequest implements Runnable {
      private AsyncContext context;
      private int ms;
      private static int number;
      //自訂建構子 傳遞AsyncContext與佇列區的毫秒
      public AsyncRequest(AsyncContext context, int ms) {
        this.context = context;
        this.ms = ms;
      }
      @Override
      public void run() {
        try {
          // 模擬長時間的處理
          Thread.sleep(ms);
          //callback取出回應訊息到前端相對的ServletResponse物件 寫出訊息到前端
          PrintWriter out = context.getResponse().getWriter();
          out.println("處理完成 讓您久等了.... " + ++number);
          // 正式callback處理 回訊息到前端去
          context.complete(); //正式完成
        } catch (Exception e) {
          e.printStackTrace();
        }
```

Servlet設計
-參照出AsyncContext進行非同步處理

- 必須設定@WebServlet aysncSupport=true，否則執行AsyncContext將會產生錯誤。
- 建構一個Thread Pooling(執行緒集區)
- 透過ServletRequest.getAsyncContext()參照出AsyncContext物件。
- 透過自訂的Runnable物件，注入AsyncContext物件，並且送至Thread Pooling進行非同步處理。

AsyncServlet.java

```
@WebServlet(name = "AsyncServ
asyncSupported = true)

public class AsyncServlet extends HttpServlet {
  // 執行緒池(固定數量的執行緒Pooling)
   private ExecutorService executorService = Executors.newFixedThreadPool(10);
  //Overriding doGet
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    //設定回應編碼
    response.setContentType("text/html; charset=UTF8");
    //取出傳遞進來的Queue參數中的毫秒數
    int ms = Integer.parseInt(request.getParameter("ms"));
    //透過ServletRequest取的非同步處裡的AsyncContext物件
    AsyncContext ctx = request.startAsync();
    //傳遞AsyncContext與停留秒數 注入一個AsyncRequest物件(Runnable)
    executorService.submit(new AsyncRequest(ctx, ms));

  }

  @Override
  public void destroy() {
    executorService.shutdown(); //停止Thread Pooling
  }
}
```

Client Side網頁非同步呼喚服務設計-1

- 網頁UI設計。
- 設計三個按鈕，事件Click執行同一個事件程序，傳遞間隔毫秒。
- 設計當非同步處理回應的結果，所要更新的畫面。

asyncdemo.html

```
<body>
  <input type="button" value="5秒" onclick="asyncProcess(5000, 'data_3')">
  <input type="button" value="1秒" onclick="asyncProcess(1000, 'data_1')">
  <input type="button" value="0.1秒" onclick="asyncProcess(100, 'data_01')">
  <p>
  即時資料(3秒): <span id="data_3">0</span><p>
  即時資料(1秒): <span id="data_1">0</span><p>
  即時資料(0.1秒): <span id="data_01">0</span><p>
  呼叫次數: <span id="count">0</span><p>
</body>
```

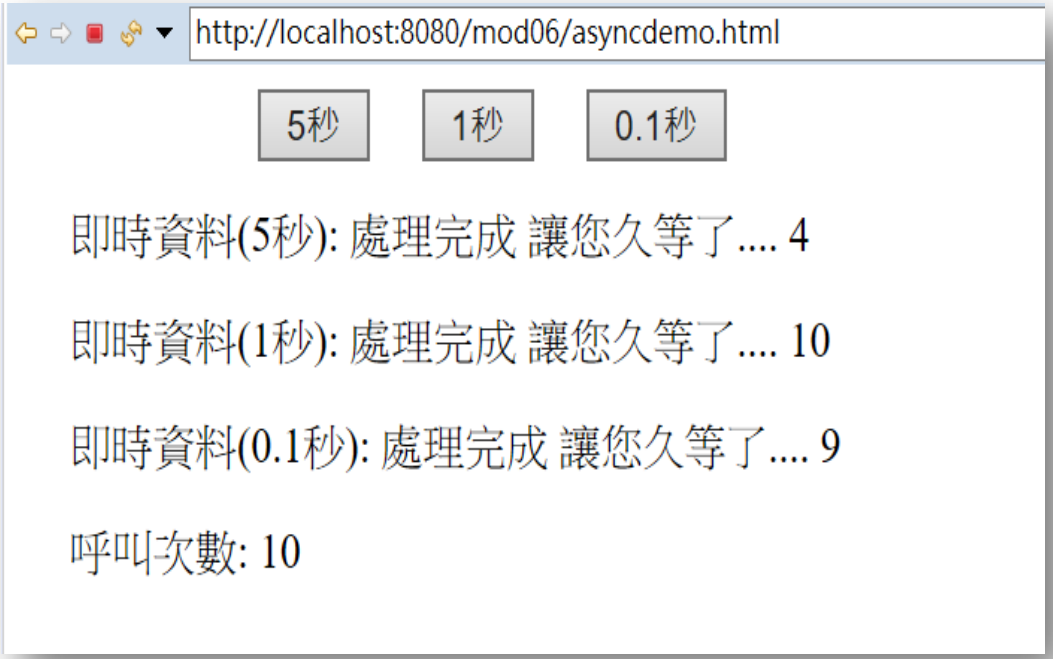
Client Side網頁非同步呼喚服務設計-2

- 採用JavaScript建構Aajx物件XMLHttpRequest。
- 設定XMLHttpRequest onreadyStatechange callback回呼的程序。
- 判斷readyState==4與status code為200，表示已完成callback程序。
- 透過XMLHttpResponse.responseText屬性取回回應的內容

asyncdemo.html

```
<script type="text/javascript">
    var count = 0;
    function asyncProcess(ms, tagName) {
        // 呼叫次數
        document.getElementById('count').innerHTML = ++count;
        var xhr;
        if(window.XMLHttpRequest) {
            xhr = new XMLHttpRequest(); //建構XMLHttpRequest物件 進行Aajx用
        }
        //Ajax callback呼喚後端的async.view
        xhr.onreadystatechange = function() {
            if(xhr.readyState == 4) {
                if(xhr.status == 200) {
                // 即時資料
                    document.getElementById(tagName)
                                .innerHTML = xhr.responseText; //將回應資料送至標籤id tagName
                    asyncUpdate(ms, tagName); //遞迴呼叫
                }
            }
        };
        //採用Http Request Method-GET，指向撰寫的的非同步處理的Servlet
        xhr.open('GET', 'async.view?ms='+ ms + '&timestamp=' + new Date().getTime());
        xhr.send(); //正式執行XMLHttpRequest 進行AJAX非同步呼喚與處理
    }
</script>
```

Demo


http://localhost:8080/mod06/asyncdemo.html
5秒
1秒
0.1秒
即時資料(5秒): 處理完成 讓您久等了.... 4
即時資料(1秒): 處理完成 讓您久等了.... 10
即時資料(0.1秒): 處理完成 讓您久等了.... 9
呼叫次數: 10

Lab

- 請使用AsyncContext設計一個非同步處理程序應用
- 如何使用前端JavaScript AJAX呼喚AsyncContext設計一個非同步處理應用