第二章　相關研究探討 Background

本章將針對與本研究相關的背景知識進行介紹，如本研究的研究平台-分散式運算環境，以網站服務為例說明；又如本研究的核心-服務合約，是結合軟體元件合約的概念，並針對以網站服務為元件進行軟體開發的模式所提出的構想；而本研究的重點為藉著引入服務合約的構想改善分散式運算環境下的例外處理機制，將不同的錯誤情況整合到單一的合約例外處理機制中。

第一節　網站服務

網站服務(Web Services)，就字面上的意義來說，就是透過網路存取的服務。網站服務是一個統稱，包含了許多經過 W3C 核定的標準與協定，如 WSDL、SOAP 等，目的是網站或應用程式之間的資訊交換。簡單的說，網站服務提供者提供了服務要給大家使用，那麼就必須提供一份 XML 格式的文件，稱為 WSDL（Web Services Description Language, 網站服務描述語言），文件裡描述了網站提供的所有服務（或稱 operations），以及服務請求者要如何與網站服務進行溝通，包含協定、參數型態與格式等等的資訊。以文件架構來說，主要分成兩個區塊，簡單敘述如圖 2 及表 1。

圖 2 省略了一些繁瑣且細節的部分，呈現的是 WSDL 的整體架構。很清楚地看見這是一份以樹狀結構呈現的 XML 文件，文件架構的說明請參考表 1。

```
<?xml version="1.0"?>
<definitions ...>
    <types>...</types>
    <message name="echoIn"></message>
    <message name="echoOut"></message>
    <message name="divideIn">
        <part name="a" type="s:int"/>
        <part name="b" type="s:int"/>
    </message>
    <message name="divideOut">
        <part name="return" type="s:int"/>
    </message>
    <portType name="MyWebServiceSoap">
        <operation name="echo">
            <input message="s0:echoIn"/>
            <output message="s0:echoOut"/>
        </operation>
        <operation name="divide">
            <input message="s0:divideIn"/>
            <output message="s0:divideOut"/>
        </operation>
    </portType>
    <binding name="MyWebServiceSoap"...>
        <soap:binding ... style="rpc"/>
        <operation name="echo">
            <soap:operation soapAction="#echo" style="rpc"/>
            <input><soap:body .../></input>
            <output><soap:body .../></output>
        </operation>
        <operation name="divide">...</operation>
    </binding>
    <service name="MyWebServiceSoap">
        <port name="MyWebServiceSoap" ...><soap:address .../></port>
    </service>
</definitions>
```

圖 2：WSDL 文件架構簡示圖

表 1：WSDL 文件架構表[2]

Service Interface Definition （服務介面定義）	
<types>	定義實際對應的資料型態。
<message>	定義各個 operation 的輸入輸出參數型態。
<portType>	定義此服務所提供所有的 operations。
<binding>	定義所使用的通訊協定以及提供的 operations。
Service Implementation Definition （服務實作定義）	
<service>	此 WSDL 文件所描述的服務集合。
<port>	定義服務的進入點，一個 port 會指定一種 binding 方式。

簡單地說明圖 2 就是這一份網站服務的名稱叫 MyWebServiceSoap，提供了兩個服務分別是 echo 和 divide，echo 沒有參數也沒有回傳值而 divide 的回傳值和兩個參數的型態均是整數 int，透過 SOAP 的協定提供服務。

目前市面上的整合開發環境（Integrated Development Environment(IDE)，本研究以 Microsoft Visual C++ 2005 Express Edition 為例)，已經支援直接匯入 WSDL 文件並自動產生對應的程式文件標頭檔(Header Files)。因此只要有了這一份 WSDL 文件，軟體開發者可直接匯入文件並直接引入該標頭檔進行軟體開發事項，和一般的軟體開發模式已無差異。

整合開發環境幫助軟體開發者省略許多的細節，像是真正在使用網站服務的

時候，服務請求者和服務提供者溝通的方式等等。有了 WSDL 文件讓你了解服務的參數型態與格式後，真正要溝通還是要透過 SOAP（Simple Object Access Protocol, 簡單物件存取協定），這也是個以 XML 為文件格式的傳輸協定，內容定義了訊息往來的格式，主要強調簡單性和可擴充性。但是軟體開發者在開發的過程中，幾乎不會直接地接觸 SOAP 的部分，因為整合開發環境已經幫軟體開發者將這些細節都處理好了。

第二節　軟體元件合約

軟體元件合約是軟體元件化之下的產物，為了實現軟體重用的理想，將軟體元件化之後統一元件的介面，並制定使用軟體元件的合約。軟體元件合約大約分成以下四類[1]，其一稱為介面定義語言(Interface definition languages, IDLs)，特別常見於物件導向語言中，敘述元件能操作的行為，以及傳入傳出的參數和可能產生的例外狀況；其二稱為元件行為合約(Behavioral Contracts)，定義元件行為前後需符合的條件，又被稱為 pre- and post-conditions，除此之外還定義了元件行為中的不變量(invariant)，如果行為前後條件不符或是定義的不變量改變了，都屬於違反了元件行為合約。

元件行為合約是定義在獨立運作的元件上，若元件的運作有彼此的關聯性，甚至有同步的問題存在的話，那麼就需要元件同步合約(Synchronization Contracts)了。元件同步合約描述了元件進行同步時的規範，類似 Mutex 之類的同步管理。

第四類則是服務品質合約(Quality-of-Service Contracts)，除了以上敘述如介面、行為、同步等合約之外，服務的品質也是使用者很在乎的部分，而服務的品質概約包含了最大回應時間、平均回應時間、以及回傳資料流的準確性。服務品質合約就是規範如果該服務的回應時間超過了定義的最大回應時間，即違反服務品質合約，使用者便可根據合約定義事項進行處理，如本研究中即會拋出服務合約逾時的例外供軟體開發者進行合約的例外處理。

第三節　例外處理機制

有軟體開發經驗的程式設計人員，對於例外處理一定不陌生。例外是許多程式語言用來表達預期中的錯誤的一種方式，而例外處理則是程式語言所提供用來處理錯誤的一種機制，妥善的處理例外將有效增進系統的強健度並提高系統的可靠度[4][5][6]。然而，例外處理卻是一件相當困難且不易正確完成的工作。有研究顯示，程式中的例外處理程式碼占了所有程式碼的三分之二[7]，而且許多程式的問題來自於例外處理程式碼。因此如何有效地處理例外便成為提昇軟體品與可靠度的重要課題[3]。

例外主要分成兩種類型：可預期例外與不可預期例外(checked / unchecked exception)，又稱為宣告例外及非宣告例外(declared / undeclared exception)。研究指出一個強健的程式應依據規格盡可能接受所有可能的輸入值，並且具有一個正常的程式離開點與零個以上的宣告例外離開點[8]。宣告例外離開點代表開發人員

對此程式發生錯誤有所預期，屆時將以產生例外的方式通知錯誤的發生。一個程式除了可預期的錯誤還可能因為不可預期的錯誤而導致例外的發生，例如記憶體不足或除數為零等錯誤，而此類型的例外則稱為非宣告例外[3]。

以 Java 為例來說，Java 將 exception 區分為三種不同的類別，分別是 Exception、RuntimeException 與 Error[13]，其中 Exception 分類下的均稱做可預期例外，其後兩者則稱做不可預期的例外，由於 Error 是保留給 JVM 發生錯誤時使用，應用程式不應該捕捉並處理，故在此不列入討論。簡單的說，像 RuntimeException 屬於不可預期的例外，如 null pointer 或是 array index out of bound 等，基本上應該要在程式開發階段便由程式開發人員修正，而不是在程式中由例外處理機制進行恢復[14]。而可預期的例外，本質上屬於可恢復的例外，如網路斷線或是系統忙碌等，可透過例外處理進行恢復。

例外處理程序執行完畢後系統控制權將重新回到正常的程式流程中，接下來的程式控制流程將由例外處理機制來決定。例外處理機制主要分成六類[9][10][11]，請參考表 2 的比較說明。其中本研究所使用之程式語言 C++屬於第二種及第五種，例外處理程序返回點是該例外的下一個語法單元，若例外不進行處理，則將例外向外傳播給該方法的呼叫者，若找不到相關的例外處理程序，則繼續回溯尋找前一個呼叫者。

在建立一個有足夠強健度的軟體之前，要先區分例外處理與系統容錯(fault tolerance)兩個緊密相關但卻不同的概念[12]。開發人員可以預期例外的發生，便

可以確切落實例外處理，以使原執行工作可以完成。而不可預期的例外則應該以撰寫容錯程式的方式來處理該例外。事實上，許多程式語言所提供的例外處理應該被稱為容錯處理，因為大部分的程式語言沒有明確定義可預期和不可預期的例外。但是良好的容錯處理是需要穩定的需求，若需求的變動是頻繁且被允許的，那麼例外處理才是較有效率的作法[3]。

表 2：六種例外處理機制比較表[3]

項目	定義	程式語言
non-local transfer	例外處理程序返回點在程式的其他位置	PL/I
termination model	例外處理程序返回點是在該 signal 的下一個語法單元（syntactic unit）。	CLU、Ada、PL/I、Mesa、C++、Java
resumption model	回到錯誤發生之處繼續執行。	PL/I、Mesa
retry model	例外處理程序返回點是在該 signal 的程式區塊起點。	Eiffel 、 Mesa 、 Exceptional C
propagate the exception	例外處理程序傳送例外使得該 signal 的呼叫者（caller）可以偵測到例外。	C++、Java、Ada
replacement model	例外處理程序將以一個值回傳給該 signal 的 caller。	Guide