# Java安全编码规范

[转至元数据结尾](http://k.2dfire.net/pages/viewpage.action?pageId=12353638" \l "page-metadata-end)[转至元数据起始](http://k.2dfire.net/pages/viewpage.action?pageId=12353638" \l "page-metadata-start)

### SQL Injectioin防范

**ibatis框架**  
**先看一段ibatis的xml配置**

|  |
| --- |
| <**select** id="queryByAccountId" parameterClass="java.util.Map" resultMap="ApplicationInstanceResult">  **SELECT** \* **FROM** product **where** account\_id = $accountId$  </**select**> |

以上的sql中存在一个问题  
$accountId$是变量替换的形式, 容易引入sql注入, 例如$accountId$是前台用户输入的"';select \* from admin--", 那么数据库端就会执行两个sql, 所以需要改成#accountId#, 进行预编译处理  
修改后的配置如下:

|  |
| --- |
| <**select** id="queryByAccountId" parameterClass="java.util.Map" resultMap="ApplicationInstanceResult">  **SELECT** \* **FROM** product **where** account\_id = #accountId#  </**select**> |

**再看一段ibatis的xml配置**

|  |
| --- |
| <**select** id="queryByAccountId" parameterClass="java.util.Map" resultMap="ApplicationInstanceResult">  **SELECT** \* **FROM** product **where** account\_id = #accountId# **ORDER** **BY** $columnName$ $sortType$ LIMIT #start#, #rowNum#  </**select**> |

可能由于某种需要搜索结果的排序很灵活, sql中有ORDER BY后$columnName$和$sortType$两个变量,在这里没法用变量绑定, 只能用$符号去替换,所以也存在隐患  
其实如果这两个变量是程序中指定的, 那么是没有风险的  
但是如果这两个变量是web前端选择填入的,那么就可以被利用构造sql注入的value, 就一定存在安全隐患, 碰到这样的情况, 我们需要映入安全开发的ibaits版本

|  |
| --- |
| <**dependency**>    <**groupId**>com.alibaba.external</**groupId**>    <**artifactId**>sourceforge.ibatis</**artifactId**>    <**version**>2.3.4.726-patch</**version**>  </**dependency**> |

然后是我们可以指定$columnName$和$sortType$的元数据类型, 如:$columnName:METADATA$, $sortType:SQLKEYWORD$, 这样ibatis就会做检查, 杜绝风险.  
修改后的配置如下:

|  |
| --- |
| <**select** id="queryByAccountId" parameterClass="java.util.Map" resultMap="ApplicationInstanceResult">  **SELECT** \* **FROM** product **where** account\_id = #accountId# **ORDER** **BY** $columnName:METADATA$ $sortType:SQLKEYWORD$ LIMIT #start#, #rowNum#  </**select**> |

**其他框架**  
禁止sql拼接生成, 必须使用变量绑定.  
如使用spring jdbcTempalte的一段代码:

|  |
| --- |
| String insert = "insert into spacemapping (oldspaceid,oldspacekey,newspaceid,newspacekey) values (?,?,?,?)";  jdbcTemplate.batchUpdate(insert, **new** BatchPreparedStatementSetter() {  **public** **void** setValues(PreparedStatement ps, **int** i) **throws** SQLException {          Object[] spaceMapping = spaceMappings.get(i);          ps.setLong(1, (Long)spaceMapping[0]);          ps.setString(2, (String)spaceMapping[1]);          ps.setLong(3, (Long)spaceMapping[2]);          ps.setString(4, (String)spaceMapping[3]);      }  **public** **int** getBatchSize() {  **return** spaceMappings.size();      }  }); |

### XSS 防范

攻击者在页面中注入恶意的js或者html代码，从而完全控制用户浏览器  
**假设一个模板中有以下内容**

|  |
| --- |
| <**table**>    <**tr**>      <**td**>用户名:</**td**>      <**td**>$userName</**td**>    </**tr**>  </**table**> |

假设用户输入的$userName为"<script src=xxx></script>", 那么就插入了恶意脚本的标签, 所以我们先对$userName进行一次html字符的转义处理, 就可以防止.  
修改后的代码如下:

|  |
| --- |
| <**table**>    <**tr**>      <**td**>用户名:</**td**>      <**td**>$stringEscapeUtil.escapeHtml($userName)</**td**>    </**tr**>  </**table**> |

**假设模板中有一段构建js的代码,js的功能是同态构建html内容**

|  |
| --- |
| #foreach ( $node in $nodeList )  **var** input = document.createElement("input");     input.name="$node.name";     input.value= "$!node.value";     someForm.appendChild(input);  #end |

假设node中数据是用户输入的, 就可以植入js恶意脚本,如:";alert(/alibaba/);"导致安全漏洞, 所以我们就需要对node的数据输出时进行一次js字符的转义处理, 修改后的代码如下:

|  |
| --- |
| #foreach ( $node in $nodeList )  **var** input = document.createElement("input");     input.name = "$stringEscapeUtil.escapeJavaScript($node.name)";     input.value = "$stringEscapeUtil.escapeJavaScript($!node.value)";     someForm.appendChild(input);  #end |

**webx的template service plugins配置**

|  |
| --- |
| <**plugins**>                  <**vm-plugins:escape-support** defaultEscape="html">                      <**vm-plugins:noescape**>                          <**vm-plugins:if-matches** pattern="^control\." />                          <**vm-plugins:if-matches** pattern="^screen\_placeholder" />                          <**vm-plugins:if-matches** pattern="^stringEscapeUtil\.escape" />                          <**vm-plugins:if-matches** pattern="^csrfToken\.(get)?(\w\*)hiddenField" />                          <**vm-plugins:if-matches** pattern="^tree" />                      </**vm-plugins:noescape**>                  </**vm-plugins:escape-support**>              </**plugins**> |

加了这段配置,velocity进行模板渲染时会把所有的变量中内容进行html转义, noescape定义了不需要进行html转义的变量名表达式, 例如:<vm-plugins:if-matches pattern="^tree" />表明$tree这个变量不需要进行html转义  
**我们要求所有使用webx3框架的应用,都需要加入这段配置,在vm模板中输出js字符串,都需要显示的调用$stringEscapeUtil.escapeJavaScript($someString)**

另外在pipeline.xml中替换webx框架的RenderResultAsJsonValve,如下配置:

|  |
| --- |
| <**when**>                      <!-- 创建JSON，无模板，无layout。 -->                      <**pl-conditions:target-extension-condition** extension="json" />                              <**performAction** />                              <**performScreen** />                              <**valve** class="com.aliyun.console.common.web.valve.AliyunRenderResultAsJsonValve" />                  </**when**> |

用以json请求或其他ajax请求输出时的js标签安全转义

### CSRF 防范

集团防御CSRF方案是采用随机字符串CSRF\_TOKEN, 使用的准则是：所有涉及到数据增删改的操作必须只允许使用POST方式提交，并且在提交的数据中要包括CSRF\_TOKEN, 服务端收到请求后对CSRF\_TOKEN进行校验,决定是否接受请求.

TODO WEBX3默认方案?

### Access Control

业务逻辑中权限控制, 对数据库任何操作都需要做身份认证，特别是UPDATE、 DELETE、INSERT的时候更需要注意.  
actioin里取得当前登录用户的session信息, 从session中取得当前用户loginId，根据loginId及被操作数据的ower对比结果,进行数据更新,删除.  
另外也可以在sql里增加判断条件来控制,如在where条件中增加login\_id='xxx'

### File upload

当应用允许用户上传文件时, 应用必须在服务端对文件类型进行检查, 在安全部门规定的白名单类型下, 才允许上传. 文件上传存放目录必须是不具有执行环境的独立存储，不能将文件放在应用目录下。需要保存到专门的文件存储服务器上, 如盘古.  
应用必须明确允许上传的文件类型，程序必须要根据白名单在服务器端进行检测。其中严禁上传的文件类型有：exe，txt，html，php  
对于上传的图片, 我们需对其进行重格式化, 可以去掉多余的meta信息  
对于上传的其他文件, 进行病毒扫描.  
另外我们必须对上传的文件大小进行限制, webx3的upload service配置如下:<services:upload sizeMax="5M" />

### URL redirect

Web应用程序接收到用户提交的URL参数后，没有对参数做"可信任URL"的验证，就向用户浏览器返回跳转到该URL的指令。  
如果[alibaba.com](http://alibaba.com/)下的某个web应用程序存在这个漏洞，恶意攻击者可以发送给用户一个[alibaba.com](http://alibaba.com/)的链接，但是用户打开后，却来到钓鱼网站页面，将会导致用户被钓鱼攻击，账号被盗，或账号相关财产被盗.

故我们只信任集团相关公司的url, 可以使用fasttexst提供的工具类进行检查, 如:  
Boolean inWhiteList =  CheckSafeUrl.getDefaultInstance().inWhiteList(URL);  
检查在白名单内的,我们执行跳转, 否则定向到错误页面

#### Cookie

防止cookie中用户的敏感信息泄漏, 我们需要对cookie进行高强度的加密, 同时将设置httpOnly的属性.  
webx3中sessionStore的配置如下:

|  |
| --- |
| <**session-stores:single-valued-cookie-store** id="xxxCookieStore" >      <**session-stores:cookie** name="xxx" httpOnly="true" />      <**session-stores:encoders**>          <**session-value-encoders:simple-value-encoder** charset="UTF-8"/>      </**session-stores:encoders**>  </**session-stores:single-valued-cookie-store**> |

这里配置的encoders不能满足我们的安全要求,我们可以实现一个高强度的加解密类,替换之

#### 密码安全

DB中存放的密码必须是加密的，客户端的密码需要加密之后再传给服务端，服务端返回给客户端的数据中不要包含密码。