JAVA 安全编码规范参考

目录

1 安全编码基本原则

1.1 所有输入数据都是有害的

直接输入数据：

对于用户通过 GET, POST, COOKIE, REQUEST 等输入的数据以及框架提供的数据来源，即通信协议中从客户端传过来的一切变量，无论是用户手动填写的数据或是客户端浏览器或操作系统自动填写的数据，都可能产生安全问题，需要进行严格的安全性检查。

间接的输入数据：

从数据库、文件、网络、内部 API 获取的数据等，即一些不直接来源于用户，但是又不是程序中定义好的常量数据。比如用户的输入经过层层转化输出到数据库或文件，后面又再次利用的时候，这时获得的数据依然是不可信的，同样需要进行严格的安全性检查。

1.2 不依赖运行环境的安全配置

不能寄希望于配置文件的安全选项，必须将程序置身于最不安全的配置下进行考虑。

1.3 安全控制措施落实在最后执行阶段

每个安全问题都有其产生的原因,例如 SQL 注入的原因是 SQL 语句参数拼接。因此对 SQL 注入问题的防范，需要在 SQL 语句执行前对参数进行安全处理，因为此时才能确定预期的参数数据类型、数据范围等。

1.4 最小化

最小化原则适用于所有安全相关的领域，在代码安全方面主要表现为：

1、用户输入最小化。尽可能少地使用用户的输入。

2、用户输入范围最小化。过滤参数时应使用白名单策略，对于可明确定义范围的参数检查参数的有效性，譬如 Email，卡号，身份证号等。

3、返回信息最小化。程序错误信息等应对用户屏蔽，不要将原始错误信息直接返回到用户侧。

1.5 失败终止

对用户提交的数据进行安全性检查的时候，如果发现数据不符合要求应终止业务的执行，不要试图修正和转换用户提交的参数继续向下执行。

2 常见漏洞对应的安全编码方法

命令注入

正解编码方法：

1. 精确匹配用户提交数据

```
String ip = request.getParameter("ip");
if(null==ip){
        //handle error
}
Boolean ret = Pattern.matches("((?:(?:25[0-5]|2[0-4]\\d|[01]?\\d?\\d)\\.){3}(?:25[0-5]|2[0-4]\\d|[01]?\\d?\\d))", ip);
if(!ret){
      //handle error
}
String[] cmd = new String[]{"ping", "-c", "2", ip};
Runtime rt = Runtime.getRuntime();
Process proc = rt.exec(cmd);
```

2. 使用白名单

```
String dir=request.getParameter("dir");
if(null==dir){
        //handle error
}
switch (dir){
        case "test1":dir="test1";
                break;
        case "test2":order_by="test2";
                break;
        default:order_by="test";
}
Runtime runtime=Runtime.getRuntime();
Process process=runtime.exec(new String[]{"ls ", dir});
int result=process.waitFor();
//do something
```

代码注入

正解编码方法：

应使用白名单：

```
public Object fix(HttpServletRequest request,Map<String, Class<?>>
whiteList ,org.apache.log4j.Logger logger) {
        Object obj = null;
        try {
                String className = request.getParameter("className");
                if(null==className){
                        //handle error
                }
                if (whiteList.containsKey(className)) {                                //
白名单
                        obj = whiteList.get(className).newInstance();
                }
        } catch (InstantiationException e) {
                //do something
        }
        return obj;
}
```

SQL 注入

正解编码方法：

应使用参数化查询

a、参数化查询方法：

jdbc

应使用 PreparedStatement：

```
HttpServletRequest request = ...;
String userName = request.getParameter("name");
if(null==userName){
        //handle error
}
```

```
Connection con = ...
String query = "SELECT * FROM Users where user=?";
PreparedStatement pre=conn.prepareStatement(query);
pre.setString(1, userName);
pre.execute();
```

mybatis

应使用”#”的写法:

```
<select id="getByPage" resultType="com.domain.Users" parameterType="com.Param">
        SELECT
                username,id
        FROM tb_users
        WHERE isdeleted=1
        <if test="name!=null and name!=''">
                AND nickname LIKE CONCAT('%', #{name}, '%')
        </if>
        ORDER BY
                createtime DESC
        limit #{fromIndex},#{count}
</select>
```

注意:

不论项目是否使用了框架，用户参数需要用到表名、字段名或涉及到 order by、group by、limit 操作时，参数化查询会使表名、字段名失去原有的意义

1. 使用 java 安全 SDK 中的方法

```
        String columnName = request.getParameter("columnName");
        if(null==columnName){
                //handle error
        }
        String columnNameEncode = sqlTool.mysqlSanitise(columnName, true);
        query = "SELECT NAME FROM users order by " + columnNameEncode ;
```

2. 使用白名单处理：

```
switch (columnName){
        case "name":columnName="name";
                break;
        case "num":columnName="num";
                break;
        default:columnName="id";
}
```

Mongo 注入

正解编码方法：

不可以直接拼接参数，使用 BasicDBObject

```
String name = request.getParameter(" name");
if(null==name){
        //handle error
}
BasicDBObject databaseQuery = new BasicDBObject("name", name);
DBCursor cursor = characters.find(databaseQuery);
try {
        while(cursor.hasNext()) {
                System.out.println(cursor.next());
        }
} finally {
        cursor.close();
}
```

XXE

正解编码方法：

1. 在解析 XML 数据时应限制 DTDs（doctypes）参数的解析：

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
try {
            String FEATURE = "http://apache.org/xml/features/disallow-doctype-decl";
            dbf.setFeature(FEATURE, true);
```

```
}catch (Exception e) {
            // This should catch a failed setFeature feature
}
```

Xpath 注入

正解编码方法：

应使用 Xpath 参数化的查询：

```
DocumentBuilderFactory builderFactory=DocumentBuilderFactory.newInstance();
builderFactory.setNamespaceAware(true);
DocumentBuilder builder=builderFactory.newDocumentBuilder();
Document document =builder.parse(new File(filepath));
XPathFactory factory=XPathFactory.newInstance();
XPath path=factory.newXPath();
String  statement="/user[loginID/text()=$username  and  password/text()=$password]/text()";
//$username 和$password 占位
SimpleVariableResolver variableResolver = new SimpleVariableResolver();
variableResolver.addVariable(new QName("password"), password);                    //参数绑定
variableResolver.addVariable(new QName("username"), username);                    //参数绑定
path.setXPathVariableResolver(variableResolver);
XPathExpression xPathExpression = path.compile(statement);
```

XSS

正解编码方法：

应编码后输出，输出到前端不同的 html 标签或属性中时，就采用不同的编码方法，

1. 使用 ESAPI 时：

```
        //参数输出到 html 实体， <div>..xssinput..</div>
        String safe = ESAPI.encoder().encodeForHTML(xssInput);
        //参数输出到 html 标签的属性， <div attr=.. xssinput..>content</div>
        String safe = ESAPI.encoder().encodeForHTMLAttribute(xssInput);
//参数输出到 JavaScript 中， <script>x='...xssInput...'</script>
        String safe = ESAPI.encoder().encodeForJavaScript(xssInput);
        //富文本
        ESAPI.validator(). getValidSafeHTML()
```

2. 使用 Spring 框架时，可用框架自带的 HtmlUtils.htmlEscape 编码输出到 html 实体：

```
@RequestMapping("/xsstest")
public String xssTest(@RequestParam("id") String id, Model model){

        id=HtmlUtils.htmlEscape(id);
        model.addAttribute("id",id);
        return "index";
}
```

3. 未使用框架时，可使用 commons-lang 库的 StringEscapeUtils.escapeHtml 编码输出到 html 实体：

```
<%
      String id=request.getParameter("id");
      out.println(StringEscapeUtils.escapeHtml(id));
%>
```

CSRF

正解编码方法：

1. web passport 认证通过后会在 cookie 植入 csrf_token。此类安全问题前端应从 cookie 中获取 csrf_token，以 POST 方式提交包含 csrf_token 值的请求，代码如下：

```
function getCookie() {
        var value = "; " + document.cookie;
        var parts = value.split("; csrf_token=");
        if (parts.length == 2)
                return parts.pop().split(";").shift();
}
$.ajax({
        type: "post",
        url: "/xxxx",
        data: {csrf_token:getCookie()},
        dataType: "json",
        success: function (data) {
                if (data.ec == 200) {
```

```
                //do something
            }
        }
});
```

后端应从 POST 请求体中提取 csrf_token 参数值，进行校验，代码如下：

```
public boolean isCSRFProtectPassed(String session,String csrf_token){
        if (null==session || null==csrf_token){
                return false;
        }
        if (session.length()!=32 || csrf_token.length()!=32){
                return false;
        }
        if (csrf_token.equals(getCSRFTokenBySession(session))){
                return true;
        }
        return false;
}
```

其中 getCSRFTokenBySession 方法实现如下：

```
public String getCSRFTokenBySession(String session){
                return md5(session);
        }
```

URL 跳转漏洞

正解编码方法：

服务端应根据具体的业务需求防止不安全的重定向和跳转：

1. 如果只希望在当前域跳转，或者跳转后的链接比较少且比较固定，那么应在服务端对参数进行白名单限制，非白名单里面的 URL 禁止跳转；

```
String index=request.getParameter("index");
if(null==index){
        //handle error
}
switch (index){
        case "1": url="https://www.trust1.com";
```

```
                break;
        case "2": url="https://rule.trust1.com";
                break;
        default:url="https://www.trust1.com";
}
response.sendRedirect(url);
```

2. 如果因为业务需要，跳转后的链接会经常变化而且比较多，应做个中间跳转页面，提示用户将跳转到其他网站，请用户注意防范钓鱼攻击。

SSRF

正解编码方法：

此类安全问题应在服务器端对请求的 URL 进行限制：服务器端维护一个资源请求列表的映射关系，服务器端根据客户端提交的请求参数从映射关系中获取实际请求的资源。同时应禁止请求私有地址段及内网域名。

任意文件遍历

正解编码方法：

应使用白名单控制路径：

```
String directory=request.getParameter("directory");
if(null==directory){
        //handle error
}
switch (directory){
        case "./image": directory="./image";
                break;
        case "./page": directory="./page";
                break;
        ...
        default:directory="./image";
}
while(line = readFile(directory))
{
        //do something
}
```

文件上传

正解编码方法：

- 校验上传文件大小
- 文件类型是否符合要求
- 不可直接使用参数中的原文件名，要随机生成文件名，并限定后缀
- 保存到文件服务器中

```
private Long FILE_MAX_SIZE = 100L*1024*1024;//100M
@RequestMapping(value = "/upload", method = POST)
@ResponseBody
public String upload(@RequestParam("file") MultipartFile file) {
        if(null == file){
                //handle error
        }
        Long filesize = file.getSize();
        if(FILE_MAX_SIZE<filesize){
                //handle error
                return "error";
        }
        String file_name = file.getOriginalFilename();
        String[] parts = file_name.split("\\.");
        String suffix = parts[parts.length - 1];
        switch (suffix){
                case "jpeg":
                        suffix = ".jpeg";
                        break;
                case "jpg":
                        suffix = ".jpg";
                        break;
                case "bmp":
                        suffix = ".bmp";
                        break;
                case "png":
                        suffix = ".png";
                        break;
                default:
                        //handle error
                        return "error";
        }
        if(!file.isEmpty()) {
                long now = System.currentTimeMillis();
```

```
                        File tempFile = new File(now + suffix);
                        FileUtils.copyInputStreamToFile(file.getInputStream(), tempFile);
                        //将 tempFile 保存到文件服务器中，然后删除 tempFile
                }
                return "OK";
        }
```

反序列化漏洞

正解编码方法：

1. 此类问题应重载 ObjectInputStream 类的 resolveClass 方法，校验待反序列化对象的类名：

```
public final class SecureObjectInputStream extends ObjectInputStream{
        public SecureObjectInputStream() throws IOException{
                super();
        }
        public SecureObjectInputStream(InputStream in) throws IOException{
                super(in);
        }
        protected Class<?> resolveClass(ObjectStreamClass desc) throws ClassNotFoundException,
IOException{
                if (!desc.getName().equals("java_security.Person")) {
                        throw new ClassNotFoundException(desc.getName()+" not found");
                }
                return super.resolveClass(desc);
        }
}
```

2. fastjson 和 jackson 也存在反序列化的问题，应使用如下版本：

- fastjson，1.2.46 及以上版本
- jackson，2.9.8 及以上版本

WebSocket 劫持

正解编码方法：

服务端应校验 Origin 头。

1. 继承 ServerEndpointConfig.Configurator 并重写 checkOrigin 方法：

```
public class CustomConfigurator extends ServerEndpointConfig.Configurator {

        private static final String ORIGIN = "https://www.trust1.com";
        @Override
        public boolean checkOrigin(String originHeaderValue) {

                if(null==originHeaderValue || originHeaderValue.trim().length()==0)
                        return false;
                return ORIGIN.equals(originHeaderValue);
        }
}
```

或者白名单为列表时：

```
private static final List<String> ORIGIN_LIST = Arrays.asList(“http://m.trust1.com”,“http://test-s.trust1.com”,“https://s.trust1.com”);
if(!ORIGIN_LIST.contains(req.headers().get(“Origin”))) {
        return false
}
```

2. 使用自定义的配置：

```
@ServerEndpoint(value="/echo",configurator = CustomConfigurator.class)
public class EchoEndpoint {
        //do something
}
```

逻辑漏洞

入参判断

正解编码方法：

1. 入参判正负，金额、数量等相关

```
if(request.getCoupons() <= 0){
```

```
    throw new Exception();
    }
```

2. 入参取值范围

- 礼物 id、抽奖类型、年龄、手机号等
- 及时下线过期活动类型、礼物 id

```
long expireTime = getExpireTime(productId);
Boolean isExpire = checkExpire(expireTime);
```

3. 入参组合判断，例如 a 类型活动应该得到礼物 1，而不能获取 b 类型活动的礼物 2

```
String type = request.getType();
String productId = request.getProductId();
if(null==type || null==productId){
        //handle error
}
if('a'.equals(type){
    if(!'1'.equals(productId)){
                throw new Exception();
        }
} else if('b'.equals(type){
        if(!'2'.equals(productId)){
                throw new Exception();
        }
} else
{
        //handle error
}
```

4. 签名验签，入参中加入 sign 标志验证请求来源，同时防止请求参数被篡改

```
public static String checkSign(String appId, Object... args) {
        //线下约定 appId
        String appSecret = getAppsecret(appId);
        if(null==appSecret){
                //handle error
        }
```

```
        return DigestUtils.sha256Hex(appSecret + “|” + Joiner.on("|").join(args));
}
```

5. 三方支付漏洞，例如：限量的优惠购买的，保证只生成一次定单

```
//1、加锁
Lock(id+productId);
try {
        lock.acquire();
        //2、判断是否已有定单
        if(Exist(id+productId))){
                //3、如果定单成功，返回已购买过；如果定单失败，返回请支付
                …
        }
        return response;
} finally
        lock.release();
}
```

或者在三方支付的回调中判断（建议采用前一种方法）

```
//1、加锁
Lock(id+productId);
try {
        lock.acquire();
        //2、判断是否已支付
        if(Payed(id+productId)){
                //3、如果购买过且支付成功，退款
                …
        }
        return response;
} finally
        lock.release();
}
```

整数溢出

正解编码方法：

1. 类型转换应校验数据类型及数据范围：

```
public static boolean isValid(String str) {
        if(null==str){
                return false;
        }
        if (str.length() > 8 || str.length() <= 0) {
                return false;
        }
        char[] chars = str.toCharArray();
        for (int i = 0; i < chars.length; i++) {
                if (!Character.isDigit(chars[i])) {
                        return false;
                }
        }
        return true;
}
```

2. 若涉及到交易数据，还应考虑数据的实际意义，如订单数、支付金额等为正数，涉及到的数值计算，应使用减、除取代加、乘:

```
public static boolean checkValue(int number,int increase) {
        final int total=100000;
        if(number<0 || increase<0 || number>total-increase){
                return false;
        }
        return true;
}
```

3. 加法和乘法，也可使用 jdk 中的 java.lang.Math 方法，Math.addExact 和 Math.multiplyExact，这两个函数在溢出时会抛出异常

```
        try {
                int ret = Math.addExact(number, increase);
                if(ret > total){
                        return false;
                }
                return true;
        }catch (Exception e){
                return false;
        }
```

4. 在使用常数时，注意 L 的位置

```
aa = 2147483647*1000*100L;//有溢出
aa = 2147483647L*1000*100;//无溢出
```

资源未释放

正解编码方法：

此类安全问题应保证任一执行路径释放资源：

```
try {
          Statement stmt = conn.createStatement();
          ResultSet rs=stmt.executeQuery(sqlBase);
          //do something
} catch (Exception e) {
          //do something
}
finally{
          stmt.close();
}
```

越权

正解编码方法：

应判断数据归属：

```
@RequestMapping(value="/delete/{addrId}")
public Object remove(@PathVariable Long addrId){
        Map<String, Object> respMap = new HashMap<String, Object>();
        if (WebUtils.isLogged()) {
              this.addressService.removeUserAddress(addrId,WebUtils.getLoggedUserId());
                //关联用户身份
              respMap.put(Constants.RESP_STATUS_CODE_KEY,
Constants.RESP_STATUS_CODE_SUCCESS);
              respMap.put(Constants.MESSAGE,"地址删除成功!");
        }
```

并发问题

正解编码方法：

1. 使用 mysql 事务，使用事务的前提下，应使用悲观锁或乐观锁解决：

悲观锁： 在事务中使用 select for update 加悲观锁，保证总是获取最新的数据

```
        String sql_select="select num from oversold where id=1 for update";
        String sql_update="update oversold set num=? where id =1";
        conn.setAutoCommit(false);
        try {
                PreparedStatement pre_select=conn.prepareStatement(sql_select);
                PreparedStatement pre_update=conn.prepareStatement(sql_update);
                ResultSet res=pre_select.executeQuery();
                if (res.next()) {
                        int num=Integer.parseInt(res.getString(1));
                        num--;
                        if(num>0){
                                //do something
                                pre_update.setInt(1, num);
                                pre_update.executeUpdate();
                        }
                }
                conn.commit();
        } catch (Exception e) {
                conn.rollback();
        }
```

乐观锁： 需要使用一个新的字段 version 保存版本号：

```
        String sql_select="select num,version from oversold where id=1";
        String sql_update="update oversold set num=num-1,version=version+1 where id =1 and
version=?";
        conn.setAutoCommit(false);
        try {
                PreparedStatement pre_select=conn.prepareStatement(sql_select);
                PreparedStatement pre_update=conn.prepareStatement(sql_update);
                ResultSet res=pre_select.executeQuery();
                if (res.next()) {
                        int num=Integer.parseInt(res.getString("num"));
                        int version=Integer.parseInt(res.getString("version"));
                        TimeUnit.SECONDS.sleep(10);
```

```
                        if(num>0){
                                //do something
                                pre_update.setInt(1, version);
                                int ret = pre_update.executeUpdate();
                                if(ret <= 0){
                                        //update 失败，此时 version 可能已过期
                                }
                        }
                }
                conn.commit();
        } catch (Exception e) {
                conn.rollback();
        }
```

悲观锁会带来比较大的性能开销，而乐观锁会读取到脏数据，具体采用哪种加锁方式可根据具体业务场景确定。

2. 通过 redis 加锁，目前公司 redis 版本均升级到 2.6.12 以上，所以使用 set 代替 inc(setnx)/expire，同样也是保证原子性

```
jedis.set(String key, String value, String nxxx, String expx, int time)
```

3. 使用分布式锁(例如：使用 InterProcessMutex)

```
DistributedLock lock = new DistributedLock(“****”, id;
try {
                lock.acquire();
                //比较：判断是否已经支付、领取等
                //修改：支付、修改领取数量
                return response;
        } finally {
                lock.release();
        }
```

敏感信息

正解编码方法：

应配置 web.xml 文件对异常全局处理：

```
<error-page>
    <error-code>404</error-code>
    <location>/error.jsp</location>
    <error-code>500</error-code>
    <location>/error.jsp</location>
</error-page>
```

前端敏感信息:

1）应在代码上线之前删除注释信息（特别是 js 脚本、html 页面中的敏感信息，账号密码、手机号、特殊链接等等）

2）web 接口返回给前端的参数，如有敏感信息，应打码处理或加密，如下所示：

```
    1、身份证
        1****************4
    2、手机号
        13*******34
    3、姓名（注意所有接口保持一致，不要有的接口返回是姓*，有的接口返回是*名，这样还是会导致信息泄漏）
        姓*
    4、地理位置
        小数点后三位，(39.910, 116.397)
    5、IP
        不要返回 ip
```

如需加密：

- 业务需要展现时：参数内容用 aes-256 加密
- 不需要展现：参数内容用 hash 算法 sha-256