信息技术部研发规范文档

--java版 v1.4

**修订历史记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **修订** |
| **2018.07.10** | 1.0 | 初稿编写 | 吴超、胡文杰、彭大祥、陈苏强、刘宇、毛震鹏、路龙飞、郝捍东、徐原野、李喜越 |
| **2018.07.15** | 1.0 | 校对 | 许信、胡文杰 |
| **2018.10.8** | 1.1 | 新增禁止使用String的intern方法 | 吴超、胡文杰 |
| **2018.10.25** | 1.2 | 日志规范增加工具类 | 胡文杰、郝捍东 |
| **2018.11.19** | 1.3 | HTTP增加默认超时时间 | 吴超 |
| **2018.11.26** | 1.4 | 去除0.8版本kafka规范，更新0.11版本kafka规范 | 陈苏强 |

[--java版 v1.3 1](#_Toc531026878)

[一、 代码规范 9](#_Toc531026879)

[1. JDK规范 9](#_Toc531026880)

[1) JDK版本 9](#_Toc531026881)

[2. 代码编写规范 9](#_Toc531026882)

[1) 代码格式规范 9](#_Toc531026883)

[2) 面向对象规范 10](#_Toc531026884)

[3) 控制规范 11](#_Toc531026885)

[4) 数字变量使用规范 11](#_Toc531026886)

[5) 命名规范 12](#_Toc531026887)

[6) 性能规范 13](#_Toc531026888)

[7) 异常处理规范 14](#_Toc531026889)

[8) 代码注释规范 15](#_Toc531026890)

[9) 其他规范 16](#_Toc531026891)

[3. 代码上线准则 16](#_Toc531026892)

[1) sonar扫描 16](#_Toc531026893)

[2) 测试 16](#_Toc531026894)

[二、 代码分层规范 16](#_Toc531026895)

[1. 项目结构分层定义 16](#_Toc531026896)

[1) 项目结构分层图 16](#_Toc531026897)

[2) DAL层（数据库持久层） 17](#_Toc531026898)

[3) COMMON层（基础工具层） 17](#_Toc531026899)

[4) MANAGER层（数据处理层） 17](#_Toc531026900)

[5) CORE层（核心逻辑处理层） 18](#_Toc531026901)

[6) INTEGRATION层（服务集成层） 18](#_Toc531026902)

[7) BIZ层（业务逻辑处理层） 18](#_Toc531026903)

[8) SERVICE层（服务封装层） 18](#_Toc531026904)

[9) TASK层（作业实现层） 19](#_Toc531026905)

[10) API层（开放接口层） 19](#_Toc531026906)

[11) WEB层（界面层） 19](#_Toc531026907)

[三、 日志规范 19](#_Toc531026908)

[1. 日志代码规范 19](#_Toc531026909)

[1) 日志输出的规范 19](#_Toc531026910)

[2. 日志监控规范 20](#_Toc531026911)

[1) 日志监控接入说明 20](#_Toc531026912)

[2) 日志监控规范 20](#_Toc531026913)

[3) 接口请求报文 21](#_Toc531026914)

[4) 接口返回报文 21](#_Toc531026915)

[5) 业务处理过程信息打印内容 22](#_Toc531026916)

[3. 日志标准工具配置 22](#_Toc531026917)

[1) 日志标准打印工具 22](#_Toc531026918)

[2) 日志标准配置文件 23](#_Toc531026919)

[四、 API规范 24](#_Toc531026920)

[1. JAR包命名规范 24](#_Toc531026921)

[3) GroupID 格式 24](#_Toc531026922)

[4) ArtifactID 格式 25](#_Toc531026923)

[5) Version格式 25](#_Toc531026924)

[2. 上传JAR包 27](#_Toc531026925)

[1) JAR包定义 27](#_Toc531026926)

[2) JAR包规范 28](#_Toc531026927)

[3. 请求参数 28](#_Toc531026928)

[1) 请求参数规范 28](#_Toc531026929)

[4. 返回解析 28](#_Toc531026930)

[1) 返回参数规范 28](#_Toc531026931)

[5. 错误码 29](#_Toc531026932)

[1) 接口响应result类 29](#_Toc531026933)

[2) 错误码标准规范 29](#_Toc531026934)

[3) 错误码码库 29](#_Toc531026935)

[4) 错误码传递流程 30](#_Toc531026936)

[5) 错误码标准原则 33](#_Toc531026937)

[五、 中间件规范 33](#_Toc531026938)

[1. redis 33](#_Toc531026939)

[1) Key使用规范 33](#_Toc531026940)

[2) Value使用规范 34](#_Toc531026941)

[3) Redis数据类型选择建议 34](#_Toc531026942)

[4) Redis程序中严格禁止使用的命令 35](#_Toc531026943)

[5) redis建议参数配置 35](#_Toc531026944)

[2. kafka 36](#_Toc531026945)

[1) Api 36](#_Toc531026946)

[2) 命名规范 36](#_Toc531026947)

[3) 配置规范 36](#_Toc531026948)

[4) 注意事项 37](#_Toc531026949)

[3. rabbitmq 38](#_Toc531026950)

[1) Api 38](#_Toc531026951)

[2) 资源创建规范 38](#_Toc531026952)

[3) 权限控制 38](#_Toc531026953)

[4) 配置规范 38](#_Toc531026954)

[5) 注意事项 39](#_Toc531026955)

[4. fastdfs 39](#_Toc531026956)

[1) Api 39](#_Toc531026957)

[2) 注意事项 39](#_Toc531026958)

[5. elastic-job 40](#_Toc531026959)

[1) Api 40](#_Toc531026960)

[2) 命名规范 40](#_Toc531026961)

[3) 配置规范 40](#_Toc531026962)

[4) 注意事项 41](#_Toc531026963)

[6. zookeeper 41](#_Toc531026964)

[1) Api 41](#_Toc531026965)

[2) 命名规范 42](#_Toc531026966)

[3) 配置规范 42](#_Toc531026967)

[4) 注意事项 42](#_Toc531026968)

[7. Dubbo 42](#_Toc531026969)

[1) Api 42](#_Toc531026970)

[2) 配置规范 42](#_Toc531026971)

[3) 注意事项 43](#_Toc531026972)

[六、 数据库规范 44](#_Toc531026973)

[1. mybatis sqlmap规范 44](#_Toc531026974)

[1) 变量 44](#_Toc531026975)

[2) 更新时间 44](#_Toc531026976)

[3) 动态条件 44](#_Toc531026977)

[4) 注释 46](#_Toc531026978)

[5) dynamic 47](#_Toc531026979)

[6) delete语句 47](#_Toc531026980)

[2. 数据库表设计规范 47](#_Toc531026981)

[1) 数据库表 47](#_Toc531026982)

[2) 索引 48](#_Toc531026983)

[3) 约束 48](#_Toc531026984)

[4) SEQUENCE 48](#_Toc531026985)

[5) 触发器（禁用） 48](#_Toc531026986)

[6) 过程、函数、包（禁用） 49](#_Toc531026987)

[3. SQL规范 49](#_Toc531026988)

[1) DDL 49](#_Toc531026989)

[2) 绑定变量 49](#_Toc531026990)

[3) 前缀 49](#_Toc531026991)

[4) select \* 49](#_Toc531026992)

[5) 函数 49](#_Toc531026993)

[6) 隐式转换 50](#_Toc531026994)

[7) 模糊查询 50](#_Toc531026995)

[8) 分页写法 50](#_Toc531026996)

[9) HINT 51](#_Toc531026997)

[4. 连接池设置规范 51](#_Toc531026998)

[1) Oracle 和MySQL 51](#_Toc531026999)

[5. 数据库事务 53](#_Toc531027000)

[1) spring @Transactional 标签注意事项 53](#_Toc531027001)

[6. 更多规范 54](#_Toc531027002)

[ 54](#_Toc531027003)

[ 54](#_Toc531027004)

[七、 安全规范 54](#_Toc531027005)

[1. 术语表述 54](#_Toc531027006)

[1) 专有名词缩略语 54](#_Toc531027007)

[2) 威胁分析模式STRIDE 模型 参考标准 55](#_Toc531027008)

[3) PSI DSS 敏感数据规范 56](#_Toc531027009)

[4) PCI DSS 数据存储规范 56](#_Toc531027010)

[5) 甜橙安全定义 56](#_Toc531027011)

[2. 数据传输与存储规范 57](#_Toc531027012)

[3. ftp数据传输规范 58](#_Toc531027013)

[4. 网络通信安全 58](#_Toc531027014)

[5. Web应用安全 58](#_Toc531027015)

[八、 安全开发规范 59](#_Toc531027016)

[1. 数据的校验 59](#_Toc531027017)

[1) SQL注入 59](#_Toc531027018)

[2) 命令注入 62](#_Toc531027019)

[3) 跨站脚本 66](#_Toc531027020)

[4) 任意文件上传 67](#_Toc531027021)

[5) 任意文件下载 70](#_Toc531027022)

[6) 任意文件删除 74](#_Toc531027023)

[7) 任意文件读取与写入 76](#_Toc531027024)

[8) Iframe框架钓鱼 83](#_Toc531027025)

[9) 负值支付漏洞 86](#_Toc531027026)

[2. 认证与授权 90](#_Toc531027027)

[1) 密码修改（无需原密码） 90](#_Toc531027028)

[2) 密码策略不足 95](#_Toc531027029)

[3) 用户名密码明文传输 96](#_Toc531027030)

[4) 缺少验证码功能 98](#_Toc531027031)

[5) 验证码可重复利用 100](#_Toc531027032)

[6) 用户名枚举（用户名和密码单独验证） 107](#_Toc531027033)

[7) 登录尝试次数限制功能失效 110](#_Toc531027034)

[8) 登录后门 114](#_Toc531027035)

[9) 横向越权 117](#_Toc531027036)

[10) 纵向越权 121](#_Toc531027037)

[11) 短信炸弹 123](#_Toc531027038)

[12) 多步骤功能中的Ajax异步控制缺陷 125](#_Toc531027039)

[3. cookie与会话管理 136](#_Toc531027040)

[1) 直接关闭浏览器"退出"系统 136](#_Toc531027041)

[2) 会话标识未更新 139](#_Toc531027042)

[3) Cookie中明文存储用户名、密码 140](#_Toc531027043)

[4. 加密与解密 150](#_Toc531027044)

[1) 密码弱加密（BASE64编码、MD5直接加密） 150](#_Toc531027045)

[2) 密码弱加密（AES ECB模式） 156](#_Toc531027046)

[5. 敏感数据 157](#_Toc531027047)

[1) 用户密码明文存储 157](#_Toc531027048)

[2) 敏感信息泄漏（配置文件） 163](#_Toc531027049)

[6. 错误处理 163](#_Toc531027050)

[1) 数据库交互中的异常处理 163](#_Toc531027051)

[7. 日志安全 167](#_Toc531027052)

[1) 日志记录敏感信息 167](#_Toc531027053)

[8. 未验证的重定向 167](#_Toc531027054)

[1) URL重定向 167](#_Toc531027055)

# 代码规范

## JDK规范

1. JDK版本

* 无特殊情况，如银行侧调用不支持等，JDK版本统一使用version 1.8.0\_45

## 代码编写规范

1. 代码格式规范

* 开发人员必须保证代码格式化的一致性，否则可能会导致代码冲突，轻微的耗费人力合并代码；严重时可能导致代码丢失，引起bug或者故障。
* 每次提交代码之前·，必须对java代码format。
* 每次提交代码之前，必须检查是否有warning，并FIX所有的warning。
* 代码字符每行不能超过120个字符。
* 任何一个方法不能超过50行。
* 代码块不能过于紧密，不方便阅读。
* 失效的代码必须删除，不能以注释形式保留。
* 大括号的使用约定。如果是大括号内为空，则简洁地写成{}即可，不需要换行；如果是非空代码块则：
* 左大括号前不换行。
* 左大括号后换行。
* 右大括号前换行。
* 右大括号后还有 else 等代码则不换行；表示终止右大括号后必须换行。
* if/for/while/switch/do 等保留字与左右括号之间都必须加空格。
* 任何运算符左右必须加一个空格。运算符包括赋值运算符=、逻辑运算符&&、加减乘除符号、三目运行符等。
* 使用 tab 缩进，必须设置 1 个 tab 为 4 个空格。IDEA 设置 tab 为 4 个空格时， 请勿勾选 Use tab character；
* 方法参数在定义和传入时，多个参数逗号后边必须加空格。下例中实参的"a",后边必须要有一个空格。

method("a", "b", "c");

* IDE 的 text file encoding 设置为 UTF-8; IDE 中文件的换行符使用 Unix 格式， 不要使用 windows 格式。

1. 面向对象规范

* 所有的覆写方法，必须加@Override 注解。getObject()与 get0bject()的问题。一个是字母的 O，一个是数字的 0，加@Override可以准确判断是否覆盖成功。另外，如果在抽象类中对方法签名进行修改，其实现类会马上编译报错。
* 相同参数类型，相同业务含义，才可以使用 Java 的可变参数，避免使用 Object。说明：可变参数必须放置在参数列表的最后。（提倡同学们尽量不用可变参数编程）

规范：public User getUsers(String type, Integer... ids)

* Object 的 equals 方法容易抛空指针异常，应使用常量或确定有值的对象来调用equals。

规范： "test".equals(object);

不规范： object.equals("test");

* 所有的相同类型的包装类对象之间值的比较，全部使用 equals 方法比较。说明：对于 Integer var=?在-128 至 127 之间的赋值，Integer 对象是在IntegerCache.cache 产生，会复用已有对象，这个区间内的 Integer 值可以直接使用==进行判断，但是这个区间之外的所有数据，都会在堆上产生，并不会复用已有对象，这是一个大坑， 推荐使用 equals 方法进行判断。
* 序列化类新增属性时，请不要修改 serialVersionUID 字段，避免反序列失败；如果完全不兼容升级，避免反序列化混乱，那么请修改 serialVersionUID 值。注意 serialVersionUID 不一致会抛出序列化运行时异常。
* 构造方法里面禁止加入任何业务逻辑，如果有初始化逻辑，请放在 init 方法中。
* 当一个类有多个构造方法，或者多个同名方法，这些方法应该按顺序放置在一起， 便于阅读。
* 类内方法定义顺序依次是：公有方法 > 保护方法 > 私有方法
* 类成员与方法访问控制从严：
* 如果不允许外部直接通过 new 来创建对象，那么构造方法必须是 private。
* 工具类不允许有 public 或 default 构造方法。
* 类非 static 成员变量并且与子类共享，必须是 protected。
* 类非 static 成员变量并且仅在本类使用，必须是 private。
* 类 static 成员变量如果仅在本类使用，必须是 private。
* 若是 static 成员变量，必须考虑是否为 final。
* 类成员方法只供类内部调用，必须是 private。
* 类成员方法只对继承类公开，那么限制为 protected。

1. 控制规范

* 在一个 switch 块内，每个 case 要么通过 break/return 等来终止，要么注释说明程序将继续执行到哪一个 case 为止；在一个 switch 块内，都必须包含一个 default 语句并且放在最后，即使它什么代码也没有。
* 在 if/else/for/while/do 语句中必须使用大括号，即使只有一行代码，避免使用下面的形式：if (condition) statements;
* 推荐尽量少用 else， if-else 的方式可以改写成：

if(condition){

...

return obj;

}

// 接着写 else 的业务逻辑代码;

如果非得使用 if()...else if()...else...方式表达逻辑，【强制】请勿超过 3 层

* 除常用方法（如 getXxx/isXxx）等外，不要在条件判断中执行其它复杂的语句，将复杂逻辑判断的结果赋值给一个有意义的布尔变量名，以提高可读性。

规范：

//伪代码如下

boolean existed = (file.open(fileName, "w") != null) && (...) || (...);

if (existed) {

...

}

不规范：

if ((file.open(fileName, "w") != null) && (...) || (...)) {

...

}

* 循环体中的语句要考量性能，以下操作尽量移至循环体外处理，如定义对象、变量、获取数据库连接，进行不必要的 try-catch 操作（这个 try-catch 是否可以移至循环体外）。

1. 数字变量使用规范

* 资金计算与处理，禁止使用浮点数（Double，Float等）直接进行处理，否则会有精度问题。
* 任何后台系统，金额必须以分为单位，并且禁止使用INTEGER型，必须使用LONG型。
* 在变量使用中，禁止使用**原始类型int，long等，必须使用包装类INTEGER，LONG等。**
* 处理数据库类型为NUMBER的，禁止使用INTEGER，必须使用LONG。
* Long 初始赋值时，必须使用大写的 L，不能是小写的 l，小写容易跟数字1 混淆，造成误解。Long a = 2l; 写的是数字的 21，还是 Long 型的 2?

1. 命名规范

* 禁止同一个含义的内容，在同一个应用中出现多种不同单词与翻译。
* 枚举类乱用是最常见的，禁止同一个项目中出现2个以上相同的枚举类。
* 类、方法、包的大小写命名依照java基础规范
* 类名：首字母大写，其他单词中首字母大写，其他小写
* 方法名：首字母小写，其他单词中首字母大写，其他小写
* 变量：首字母小写，其他单词中首字母大写，其他小写
* 包名：全部小写
* 禁止无含义的包名、类名、方法名，或是以xxxx1，xxxx2 来区分

不规范：boolean var1=false；

规范：boolean isTimeout= false;

* 代码中的命名均不能以下划线或美元符号开始，也不能以下划线或美元符号结束。

不规范：\_name / $Object / name\_ / name$ / Object$

* 代码中的命名严禁使用拼音与英文混合的方式，更不允许直接使用中文的方式。说明：正确的英文拼写和语法可以让阅读者易于理解，避免歧义。注意，即使纯拼音命名方式也要避免采用。

不规范： DaZhePromotion [打折] / getPingfenByName() [评分] / int 某变量 = 3

规范： bestpay / paytools / youku / hangzhou 等国际通用的名称，可视同英文。

* 类名使用 UpperCamelCase 风格，必须遵从驼峰形式，但以下情形例外：（领域模型的相关命名）DO / BO / DTO / VO 等。

不规范：macroPolo / UserDo / XMLService / TCPUDPDeal / TAPromotion

规范：MarcoPolo / UserDO / XmlService / TcpUdpDeal / TaPromotion

* 方法名、参数名、成员变量、局部变量都统一使用 lowerCamelCase 风格，必须遵从驼峰形式。

规范 ： localValue / getHttpMessage() / inputUserId

* 常量命名全部大写，单词间用下划线隔开，力求语义表达完整清楚，不要嫌名字长。

规范： MAX\_STOCK\_COUNT

不规范： MAX\_COUNT

* 抽象类命名使用 Abstract 或 Base 开头；异常类命名使用 Exception 结尾；测试类命名以它要测试的类的名称开始，以 Test 结尾。
* 中括号是数组类型的一部分，数组定义如下：String[] args；请勿使用 String args[]的方式来定义。
* POJO 类中布尔类型的变量，都不要加 is，否则部分框架解析会引起序列化错误。定义为基本数据类型 boolean isSuccess；的属性，它的方法也是 isSuccess()，RPC 框架在反向解析的时候，“以为”对应的属性名称是 success，导致属性获取不到，进而抛出异常。
* 杜绝完全不规范的缩写，避免望文不知义。随意缩写严重降低了代码的可阅读性。
* 如果使用到了设计模式，建议在类名中体现出具体模式。说明：将设计模式体现在名字中，有利于阅读者快速理解架构设计思想。

规范：public class OrderFactory;

public class LoginProxy;

public class ResourceObserver;

* 对于 Service 和 DAO 类，基于 SOA 的理念，暴露出来的服务一定是接口，内部的实现类用 Impl 的后缀与接口区别。如CacheServiceImpl 实现 CacheService 接口。
* 枚举类名建议带上 Enum 后缀，枚举成员名称需要全大写，单词间用下划线隔开。说明：枚举其实就是特殊的常量类，且构造方法被默认强制是私有。如枚举名字：DealStatusEnum，成员名称：SUCCESS / UNKOWN\_REASON。

1. 性能规范

* 禁用beancopy工具类，与java的converter比较一次慢1ms
* 对系统资源的访问，使用后必须在finally中释放系统资源。这类资源包括：文件流、线程、网络连接、数据库连接等。(JDK 1.7版本不需要手动释放资源)
* 对于线程，线程资源必须通过线程池提供，不允许在应用中自行显式创建线程。
* 对于网络连接与数据库连接，必须由框架通过连接池提供，不允许应用中自行建立网络与数据库连接。
* 为记录加锁时，需要保持一致的加锁顺序，否则可能会造成死锁。
* 总量超过500MB的缓存数据不准放本地服务器内存，使用redis，因为缓存的越多GC刷的很厉害。
* 对象实例要就近原则，不要一上来就new一堆，服务都是很消耗内存的。
* final 可提高程序响应效率，声明成 final 的情况：
* 不需要重新赋值的变量，包括类属性、局部变量。
* 对象参数前加 final，表示不允许修改引用的指向。
* 类方法确定不允许被重写。
* 线程资源必须通过线程池提供，不允许在应用中自行显式创建线程。
* 线程池不允许使用 Executors 去创建，而是通过 ThreadPoolExecutor 的方式，这样的处理方式让写的同学更加明确线程池的运行规则，规避资源耗尽的风险。

说明：Executors 返回的线程池对象的弊端如下：高并发时，同步调用应该去考量锁的性能损耗。能用无锁数据结构，就不要用锁；能锁区块，就不要锁整个方法体；能用对象锁，就不要用类锁。

* 对多个资源、数据库表、对象同时加锁时，需要保持一致的加锁顺序，否则可能会造成死锁。线程一需要对表 A、B、C 依次全部加锁后才可以进行更新操作，那么线程二的加锁顺序也必须是 A、B、C，否则可能出现死锁。
* 并发修改同一记录时，避免更新丢失，要么在应用层加锁，要么在缓存加锁，要么在数据库层使用乐观锁，使用 version 作为更新依据。说明：如果每次访问冲突概率小于 20%，推荐使用乐观锁，否则使用悲观锁。
* 任何数据结构的构造或初始化，都应指定大小，避免数据结构无限增长吃光内存。
* 在使用正则表达式时，利用好其预编译功能，可以有效加快正则匹配速度。不要在方法体内定义：Pattern pattern = Pattern.compile(规则);
* 防止常量池溢出，string的intern()方法禁止使用，在非必要情况下，对string的操作一律使用Google Guava。
* 连接创建时间500ms，读取超时时间依赖第三方实际设置（内部非特殊情况不使用HTTP协议，非第三方内部2s）；

1. 异常处理规范

* 捕捉到的异常，不允许不做任何处理就截断，一定要记入日志，或重新抛出。
* 最外层的业务使用者，必须处理异常，将其转化为上层系统可处理结果返回
* 不要捕获 Java 类库中定义的继承自 RuntimeException 的运行时异常类，如： IndexOutOfBoundsException / NullPointerException，这类异常由程序员预检查来规避，保证程序健壮性。

规范：if(obj != null) {...}

不规范：try { obj.method() } catch(NullPointerException e){...}

* 异常不要用来做流程控制，条件控制，因为异常的处理效率比条件分支低。
* 对大段代码进行 try-catch，这是不负责任的表现。catch 时请分清稳定代码和非稳定代码，稳定代码指的是无论如何不会出错的代码。对于非稳定代码的 catch 尽可能进行区分异常类型，再做对应的异常处理。
* 有 try 块放到了事务代码中，catch 异常后，如果需要回滚事务，一定要注意重新抛出新的自定义异常。
* finally 块必须对资源对象、流对象进行关闭，有异常也要做 try-catch。
* 不能在 finally 块中使用 return，finally 块中的 return 返回后方法结束执行，不会再执行 try 块中的 return 语句。
* 方法的返回值可以为 null，不强制返回空集合，或者空对象等，必须添加注释充分说明什么情况下会返回 null 值。调用方需要进行null判断防止空指针问题。本规约明确防止空指针是调用者的责任。即使被调用方法返回空集合或者空对象，对调用者来说，也必须考虑到远程调用失败，运行时异常等场景返回 null 的情况。

1. 代码注释规范

* 类、类属性、类方法的注释必须使用 Javadoc 规范，使用/\*\*内容\*/格式，不得使用//xxx 方式。
* 任何方法上必须有注释，方法的每个参数也要有注释。
* 任何类上需有整个类的定义注释，类下的非私有方法在类头上有整体注释。
* 类上需添加作者、项目名、日期等基本信息。
* 代码修改的同时，注释也要进行相应的修改，尤其是参数、返回值、异常、核心逻辑等的修改。
* 类注释例子

/\*\*  
 \* 收单退款Biz  
 \* <p>  
 \* 1、收单退款业务流程  
 \* 2、POS退款业务流程  
 \* 3、分账退款业务流程  
 \* 4、强制退款逻辑流程  
 \* <p/>  
 \* User: huwenjie Date:2017/5/2 ProjectName: tradecenter Version: 2.0  
 \*/

* 对于注释的要求：第一、能够准确反应设计思想和代码逻辑；第二、能够描述业务含义，使别的程序员能够迅速了解到代码背后的信息。完全没有注释的大段代码对于阅读者形同天书，注释是给自己看的，即使隔很长时间，也能清晰理解当时的思路；注释也是给继任者看的，使其能够快速接替自己的工作。

1. 其他规范

* 对于“明确停止使用的代码和配置”，如方法、变量、类、配置文件、动态配置属性等要坚决从程序中删除，避免造成过多垃圾。

## 代码上线准则

1. sonar扫描

* 2016年4月前上线的应用，阻断问题数比如为0，才可上线。
* 2016年4月后上线的应用，阻断、严重、主要问题数必须为0，才可上线。

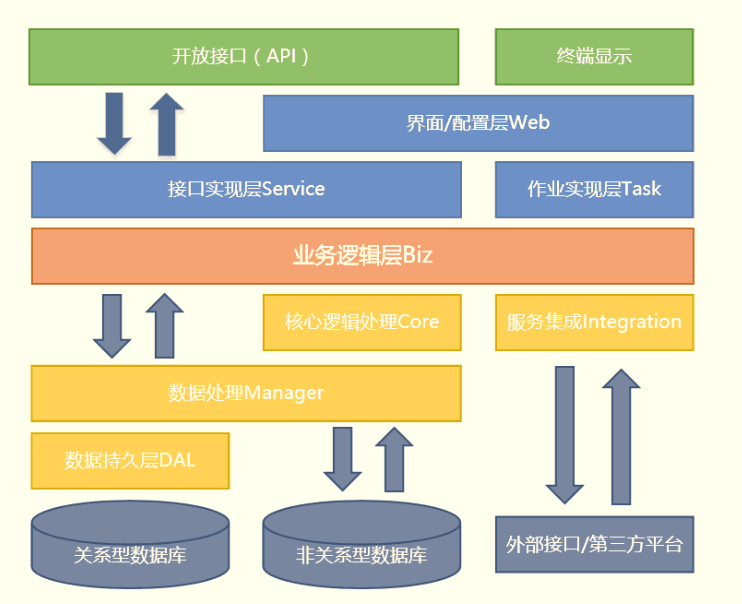
1. 测试

* 任何对外接口必须有对应的单元测试。
* 任何代码改动必须经过测试组功能验证，才可上线。
* 新应用、新流程代码必须经过压力测试与安全测试评估，才可上线。

# 代码分层规范

## 项目结构分层定义

1. 项目结构分层图



如项目业务简单，可适当减少层级，如去除Core层与Integraiton层，用Biz层代替

1. DAL层（数据库持久层）
   * mapper.xml必须按照标准格式书写，标准格式见数据库规范-XML。

* dal层所有的增删改查 提供的接口方法请以 insert、delete、update、select开头。
* mapper.xml 方法尽量不要出现大量动态条件。
* dal数据模型对象以 \*\*\*DO 命名。
* dal只能单表操作，不能出现联表操作。
* mapper.xml 每个方法必须加上注释。
* mapper.xml 任何表操作语句，表名必须起别名。
* 无意义的ID 在insert操作不需要先获取序列直接写在SQL中，减少数据库交互。
* 特长字段且非在线交易需要数据查询时，务必与交易需要的查询方法区分。

1. COMMON层（基础工具层）

* 此层不对外暴露API，如有错误码、枚举需对上层系统暴露，请放在API层。
* 放置自定义错误码、工具类、静态变量、枚举等。

1. MANAGER层（数据处理层）

* 封装基础DAL层能力。
* 业务对象以\*\*\*BO 命名。
* 业务方法对应的增删改查 请以add、remove、modify、query开头。
* **日志输出级别为debug。**
* 访问非关系型数据库（redis）。
* 非特殊情况，不可捕获异常。
* modify空执行情况异常抛出。
* 不可开启事务。
* 业务对象注入统一使用@Component。
* 不对外暴露的枚举应放到common层。

1. CORE层（核心逻辑处理层）

* 基础业务能力封装，对公用核心处理逻辑进行封装。
* 业务对象以\*\*\*BO 命名。
* 非特殊情况，不可捕获异常。
* 此层开启数据库事务。
* 不对外暴露的枚举应放到common层。

1. INTEGRATION层（服务集成层）

* 对外调用接口能力封装。
* 业务对象以\*\*\*BO 命名。
* 所有调用其他系统接口的方法必须由此层完成。
* 异步消息（Kafka、Rabbitmq）发送由此层负责。
* 处理对外接口的异常逻辑、超时逻辑。
* 此层不能直接访问数据库。

1. BIZ层（业务逻辑处理层）

* 此层主要用于整个业务流程的封装。
* 将多个原子性操作封装成一个业务流程，为服务封装提供业务能力。
* 控制业务流程分支，会有部分IF语句控制业务流程。
* 此层注释必须详细到每一步。
* 如项目业务简单，无事务管理，此项目无CORE层，可直接访问MANAGER层。
* 如项目业务复杂，有事务管理，必须通过CORE层访问MANAGER层。
* 业务对象以\*\*\*BO 命名。

1. SERVICE层（服务封装层）

* 实现所有API层接口。
* 业务方法对应的增删改查 请以add、remove、modify、query开头。
* 提供统一的异常信息处理能力。
* 接口请求参数响应参数必须日志输出且为info级别。提供统一的日志输出。
* 必须捕获所有异常，当异常时务必打印异常栈且为error级别。
* 业务对象注入统一使用@Service。
* 异步消息（Kafka、Rabbitmq）接收由此层负责。
* 业务对象以\*\*\*DTO 命名。

1. TASK层（作业实现层）

* 触发应用定时任务。
* 调用Biz层完成业务。
* 必须捕获所有异常，当异常时务必打印异常栈且为error级别。

1. API层（开放接口层）

* 放置入参、出参，服务接口定义。
* 对外提供枚举提供在此层。
* 业务对象以\*\*\*DTO 命名。
* 唯一可对外提供Jar包的层。
* 除基础JAR包，不能引入任何第三方JAR包与其他项目JAR包。

1. WEB层（界面层）

* 应用配置文件（autoconfig）放置在此层。
* 所有层的单元测试放置在此层。
* 提供健康检查页面。
* 提供WEB页面（一般后台应用不需要）。

# 日志规范

## 日志代码规范

1. 日志输出的规范

* 只有在接口层的调用，输出全日志，其他的日志，只输出重要的日志信息，非重要的日志禁止输出。用户的敏感信息包括密码、短信验证码、支付验证码、身份证号、银行卡号、银行密钥，商户密钥等信息不允许直接打印。
* 应用中不可直接使用日志系统（Logback）中的 API，而应依赖使用日志框架SLF4J 中的 API，使用门面模式的日志框架，有利于维护和各个类的日志处理方式统一。
* 生产服务标准为info级别，debug绝对不准上生产。
* 生产代码禁止以System及Throwable.printStackTrace的方式输出日志信息，必须用Logger替代。
* 对于日志的打印，任何情况下都不允许日志错误导致业务失败。
* 对于由于系统原因造成业务处理失败的事件，需要记录错误日志。非系统原因的业务处理失败，不应该记录错误日志（使用warn级别），避免错误日志过大，影响紧急情况下的故障分析与诊断。
* 对于异常堆栈的输出，必须以log.XXX (“msg”,e)的形式输出，禁止log.XXX (“msg”+e)的错误形式。
* 在使用toString() , 一定要使用 “+”链接，或者用StringBuilder进行字符串链接，其效率可以差10倍



## 日志监控规范

1. 日志监控接入说明

* 监控平台通过日志关键字监控的接口成功率和性能。
* 通过日志特定格式返回需要参数，通过agent抽取需要的日志，并对大量日志进行分析，最终给出接口交易量，成功率和性能指标，并根据SPCC算法给出告警对应的阈值，低于告警阈值则告警。

1. 日志监控规范

* []是区分日志各属性的分隔符，在日志中必须打印，就算该属性为空，也要将空[]打印出来，[]里面的内容不要包括特殊字符，已方便监控平台统计。
* [日志级别]：分别有INFO  WARN  ERROR三种，INFO为正常成功，WARN为一般失败，ERROR为严重失败。
* 同一条日志内容各个信息必须唯一，不得在该条日志内有重复。
* 其中[类名] [方法名]在显示上可能和[日志打印代码位置]重复，但为了之后监控的准确性及部分功能实现必须显示。
* [线程ID]根据实际情况显示，为空就留空[]。
* [dstID]与[traceLogid]两者不能同时为空。
* 标红部分为公共属性，内容必填。

1. 接口请求报文

* 日志规范：

[日期时间] [日志级别] [日志打印代码位置][线程ID] traceLogid:[traceLogid]dstTraceId:[dstID] [类名] [方法名]PARAMETER:[接口请求报文]

* 请求接口标准案例：

[2017-06-13 15:59:43,960] [INFO] [c.b.a.service.AccountOpenServiceImpl 165][] traceLogid:[]dstTraceId:[50f13c63ea9c45b1a1214ab0e66f7487][AccountOpenServiceImpl][accountOpenBatch]PARAMETER:[AccountOpenReqDTO(super=com.bestpay.account.service.api.model.AccountOpenReqDTO@4c3eecbc,requestNo=213100102,requestSystem=CIF,accountType=PERS,accountNo=6103131000629156, accountName=我的不可提现账户,contractNo=3177030316687479,ccy=156,settleBranch=null, accAttribute=UNWITHDRAW,accNature=BALANCE,bizType=10,subBizType=10, areaCode=440000,remark=null,operator=accountcenter), AccountOpenReqDTO(super=com.bestpay.account.service.api.model.AccountOpenReqDTO@67eb418c, requestNo=3031682714, requestSystem=CIF, accountType=PERS, accountNo=71030316423244156,accountName=我的余额账户, contractNo=3177030316687479, ccy=156, settleBranch=null, accAttribute=BALANCE, accNature=BALANCE, bizType=10, subBizType=10, areaCode=440000, remark=null, operator=accountcenter)]

1. 接口返回报文

* 日志规范：

 [日期时间] [日志级别] [日志打印代码位置] [线程ID] traceLogid:[traceLogid]dstTraceId:[dstID][类名][方法名][服务调用耗费时间][接口调用状态][响应码]RESPONSE:Result{接口返回报文}

* 返回报文标准案例：

 [2017-06-1315:59:43,969][INFO][c.b.a.service.AccountOpenServiceImpl190][] dstTraceId:[50f13c63ea9c45b1a1214ab0e66f7487][AccountOpenServiceImpl][accountOpenBatch][10ms][Success][000]RESPONSE:Result{success=true,result=[AccountOpenResDTO(super=com.bestpay.account.service.api.model.result.AccountOpenResDTO@36bd8a57, accDate=Tue Jun 13 00:00:00 CST 2017, receiptNo=201706131313112787), AccountOpenResDTO(super=com.bestpay.account.service.api.model.result.AccountOpenResDTO@7ca05cb0,accDate=TueJun1300:00:00CST2017,receiptNo=201706131313112789)]}

* 补充说明：

[接口调用状态]：描述调用接口的状态，只有两种情况Success,Failure，监控信息需要，必填

[响应码]：描述业务是否成功的状态，根据各平台自定义状态填充响应码

[服务调用耗费时间]：接口返回报文里面增加接口/服务调用的时间，单位毫秒(ms)，例如[10ms]

1. 业务处理过程信息打印内容

* 日志规范：

 [日期时间] [日志级别][日志打印代码位置] [线程ID] traceLogid:[traceLogid] dstTraceId:[dstID]自定义打印内容

* 处理过程信息标准案例：

[2017-06-1315:59:44,559][INFO][com.bestpay.account.biz.AbstractBiz 89] [a4dd34aa-34ff-4051-8886-c1cd1b205e35]traceLogid:[]dstTraceId:[5d110df634c542e3a3100b2e7372519f] step checkRepeatFlag

## 日志标准工具配置

1. 日志标准打印工具

* Api

<dependency>

<groupId>com.bestpay</groupId>

<artifactId>commons</artifactId>

<version>1.1.3</version>

</dependency>

* 工具类

com.bestpay.utils. LogUtil

* 代码实例

Result<TradeBaseResDTO> result = new Result<TradeBaseResDTO>();

long startTime = System.currentTimeMillis();

String methodName = "createRealTimeOrder";

try {

LogUtil.printStartLogInfo(log, methodName, createOrderReqDTO);

//调用通用下单方法，交易类型赋值为及时到账

TradeBaseResDTO tradeBaseResDTO = commonCreateOrder(createOrderReqDTO);

result.setResult(tradeBaseResDTO);

LogUtil.printEndLogInfo(log, methodName, startTime, result);

} catch (TradeCenterBaseException e) {

BaseResultHelper.fillWithError(result, e.getErrorCode());

LogUtil.printEndLogInfo(log, methodName, startTime, result);

} catch (Exception e) {

BaseResultHelper.fillWithException(result, e);

LogUtil.printExceptionInfo(log, methodName, e);

}

return result;

1. 日志标准配置文件

<appender name="logfile" class="ch.qos.logback.core.rolling.RollingFileAppender">

<File>${logback.logpath}cif-core.log</File>

<encoder>

<pattern>[%date] [%level] [%logger %line] [%thread] [%X{TRACE\_LOG\_ID}] [%X{dstTraceId}][%traceIp] %msg %n</pattern>

<charset>UTF-8</charset>

<immediateFlush>false</immediateFlush>

</encoder>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>${logback.logpath}cif-core-%d{yyyy-MM-dd}.%i.log</fileNamePattern>

<maxHistory>30</maxHistory>

<TimeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">

<maxFileSize>512MB</maxFileSize>

</TimeBasedFileNamingAndTriggeringPolicy>

</rollingPolicy>

</appender>

<appender name="ASYNC" class="ch.qos.logback.classic.AsyncAppender">

<!-- 不丢失日志.默认的,如果队列的80%已满,则会丢弃TRACT、DEBUG、INFO级别的日志 -->

<discardingThreshold >0</discardingThreshold>

<!-- 更改默认的队列的深度,该值会影响性能.默认值为256 -->

<queueSize>2048</queueSize>

<includeCallerData>true</includeCallerData>

<!-- 添加附加的appender,最多只能添加一个 -->

<appender-ref ref ="logfile"/>

</appender>

# API规范

## JAR包命名规范

1. GroupID 格式

* 格式：com.{公司/BU }.业务线.[子业务线]，最多 4 级。
* 说明：{公司/BU} 例如：alibaba/taobao/tmall/aliexpress 等 BU 一级；子业务线可选。
* 正例：com.bestpay.marketing (营销提供的api)

1. ArtifactID 格式

* 产品线名-模块名。语义不重复不遗漏，先到中央仓库去查证一下。
* 正例：marketing-minus / marketing-minus/ fastjson-api 。

1. Version格式

* 初始版本号为1.0.0 ，新应用首次上生产，版本号必须不能低于1.0.0。但是，若是想在正式上线前，在准生产做一次特殊发布（也就是只发准生产，不发生产），准生产特殊发布的version可以从0.0.1开始。
* 第一位： 主版本号：产品方向改变，或者大规模 API 不兼容，或者架构不兼容升级。

第二位： 次版本号：保持相对兼容性，增加主要功能特性，影响范围极小的 API 不兼容修改。

第三位： 修订号：保持完全兼容性，修复 BUG、新增次要功能特性等

* 申请常规、紧急、变更等版本修改规则说明（设定现版本为version=A.B.C）：

应用重构：A位+1，B位、C位置0

常规版本：B位+1，A位不变、C位置0

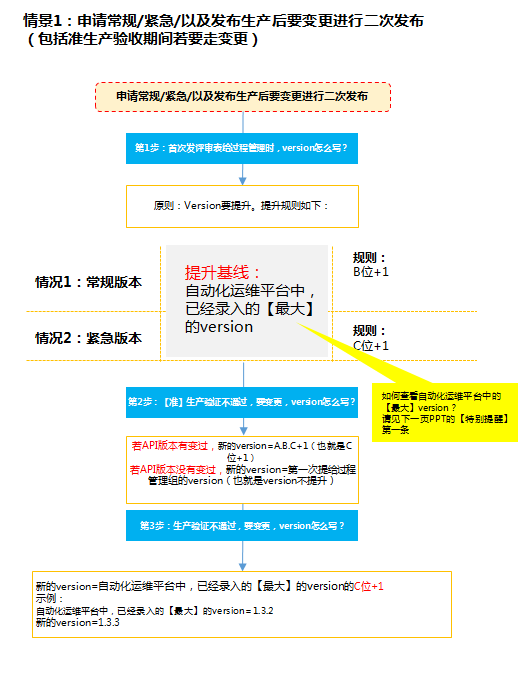
紧急版本：C位+1

生产验证不通过变更：C位+1

准生产验证不通过：C位不变 或者 C位+1（详细规则见下图）

特殊准生产发布：C位不变 或者 C位+1 （详细规则见下图）

* 版本变动图：





## 上传JAR包

1. JAR包定义

* 一方库: 本工程内部子项目模块依赖的库（jar 包）。
* 二方库: 公司内部发布到中央仓库，可供公司内部其它应用依赖的库（jar 包）。
* 三方库: 公司之外的开源库（jar 包）。

1. JAR包规范

* 对外提供的api禁止引用二方库的依赖包，避免依赖传递。
* 对外提供的api引用三方库依赖包最小化。
* 开发测试阶段使用快照包，封版确认版本号后，发布正式包到生产release库。
* 正式版本号不允许覆盖升级，如当前版本1.3.3，那么下一个合理的版本号1.3.4 或1.4.0 或 2.0.0
* 当前maven项目的模块，需要上传二方库的api，由配置好的jenkins统一上传
* 第三方提供的jar包，请确认在私服不存在，规范命名后，交由配置管理员提供对应坐标上传
* nginx限制了上传文件大小5M，如太大无法上传请各自将包解开处理干净后再操作

## 请求参数

1. 请求参数规范

* API的请求参数封装到POJO类中，类名以DTO结尾，无特殊情况，禁止使用MAP的数据形式传参，导致上传参数不清晰。
* 所有的 POJO 类属性必须使用包装数据类型。
* 定义POJO 类时，不要设定任何属性默认值。
* POJO 类属性没有初值是提醒使用者在需要使用时，必须自己显式地进行赋值，任何空指针问题或者入库检查，都由使用者来保证。
* 接口对象必须序列化，并且请不要修改 serialVersionUID 字段，避免反序列失败。

## 返回解析

1. 返回参数规范

* API的返回参数封装到POJO类中。
* 返回对象中明确定义错误码和返回数据。
* POJO类的使用遵循请求参数中对POJO类的要求。
* 返回对象统一使用公共jar包com.bestpay.dubbo.result.Result。
* 对象中 success字段，请明确是业务成功还是调用成功。明确结果集T中是否有字段表明业务成功/失败字段。无特殊情况下，业务类接口表明业务成功（或受理成功），查询类接口表明调用成功。

## 错误码

1. 接口响应result类

* success：调用是否成功
* result：调用结果集
* errorCode：错误码
* errorMsg：错误码描述
* primaryErrorCode：元错误码（只要赋值一定是标准错误码规范）
* primaryErrorMsg：元错误描述（最底层程序员角度希望展示出来的错误描述，primaryErrorCod有值则必填）
* primaryErrorIP：机器IP，标识报该错误指令的机器IP地址（primaryErrorCode有值则必填）

1. 错误码标准规范

* 系统名称（大写）+自定义错误编码

举例：

* errorCode：TRADE\_CENTER+NONSUPPORT\_PAY\_TOOL\_TYPE
* errorMsg：无此支付工具类型
* primaryErrorCode：ACCOUNT\_CORE+ BAL\_CANNOT\_NEGATIVE
* primaryErrorMsg：余额不足
* primaryErrorIP：172.28.133.82

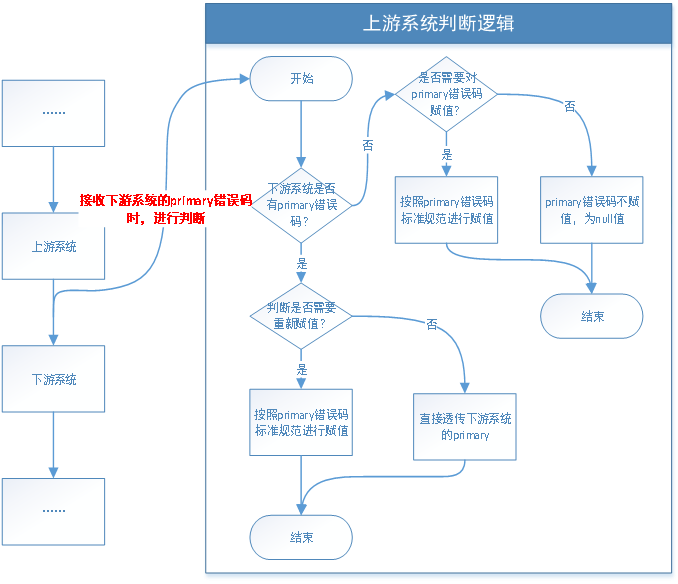
已有的errorCode，errorMsg各系统按照自己需求排期自定义修改成标准错误码体系，需要考虑自身业务逻辑处理和上层调用方的错误码特殊处理逻辑，需要同步变动，各系统审慎处理，建议与第一点分开执行；

1. 错误码码库



* primaryErrorCode错误码符合标准规范，最大长度128位。
* 最上层系统针对业务系统需要根据同一primaryErrorCode要展示不同的primaryErrorMsg错误描述的情况。
* primaryErrorMsg值允许有多个，多key值对应唯一value，primaryErrorCode为必填key值，其他key值可为产品码、事业群、语言版本等。
* 各开发组长加强组内宣贯和新人引导，避免使用出错。

1. 错误码传递流程



**非最上层系统例子：**

* 当前系统调用下游系统接口时对于接口报错的错误处理：

if (!result.isSuccess()) {

//抛出自定义异常(账户不存在),原始错误码，原始错误描述，原始错误IP取上层返回

throw new CustomException(CustomErrorCode.ACCOUNT\_NOT\_EXISTED,

result.getPrimaryErrorCode(),

result.getPrimaryErrorMsg(),

result.getPrimaryErrorIP());

} else {

//成功调用逻辑

}

* 当前系统自身逻辑处理报错的错误处理（以数据库更新报错为例）：

try {

//更新数据库索引异常

} catch (DuplicateKeyException e) {

//更新数据库异常：原始错误码，原始错误描述，原始错误IP自己系统定义

throw new CustomException(CustomErrorCode.DATA\_UPDATE\_ERROR,

CustomConstant.SYSTEM\_NAME\_PREFIX+

CustomErrorCode.DATA\_UPDATE\_ERROR.getCode(),

CustomErrorCode.DATA\_UPDATE\_ERROR.getDesc());

}

* 当前系统返回出去的错误处理（service层）：

catch (Exception e) {

//service 层统一处理异常层

if (e instanceof CustomException) {

//自定义业务异常

CustomException err = (CustomException) e;

log.error("PayParamException :{}", err);

return new Result<String>(err.getErrorCode().getCode(),err.getErrorCode().getDesc(),

err.getPrimaryErrorCode(),err.getPrimaryErrorMsg(),err.getPrimaryErrorIP());

} else {

log.error("SYSTEM\_INNER\_ERROR :{}", e);

//系统未知异常

return new Result<String>(CustomErrorCode.SYSTEM\_INNER\_ERROR.getCode(),

CustomErrorCode.SYSTEM\_INNER\_ERROR.getDesc(),

CustomConstant.SYSTEM\_NAME\_PREFIX+

CustomErrorCode.SYSTEM\_INNER\_ERROR.getCode(),

CustomErrorCode.SYSTEM\_INNER\_ERROR.getDesc(),

IPUtil.getServerIp());

}

}

**最上层系统例子：**

RedisManager redisManager = new RedisManager();

//下层系统返回的结果

Result<String> response = new Result<String>();

//产品码

String productCode ="";

//语言

String language="";

//原始错误码

String primaryErrorCode = response.getPrimaryErrorCode();

//如果原始错误码为空

if(Strings.isNullOrEmpty(primaryErrorCode)){

//按照之前的处理逻辑处理

}else{

//去错误码表里面获取错误描述,key=原始错误码+产品码+语言

String key = primaryErrorCode + productCode + language;

//拼接kye去Redis查询错误描述

String value = redisManager.queryObjectByKey(key);

//没有匹配到错误描述，把返回的原始错误描述返回

if(Strings.isNullOrEmpty(value)){

value = response.getPrimaryErrorMsg();

}

//获取到错误描述之后返回给上层展示

1. 错误码标准原则

* 形成错误码规范，公司内系统保持一致，并且能够定位到错误系统和错误机器。
* 能够返回给上层最底层的错误码和调用层的错误码，由展示层决定使用哪个，建议取最底层错误码。
* 错误码码库，可以按照标准规范、特性化业务制定不同业务类型的错误码描述，优化客户体验。

# 中间件规范

## redis

1. Key使用规范

Key命名规范

* Key命名必须能表达出实际功能意义，并且唯一，长度尽量短小
* 规定键的命名按照如下格式：平台简写\_业务简写\_自定义
* 分片键自定义部分最好为纯数字后缀。
* 长度最大不能超过100字符，尽量短。

设置过期时间

* 键需要设置过期时间，时间尽量短。某些业务要求键长期有效，可以在每次写入时，都设置过期时间，让过期时间顺延。
* 通常情况下，过期时间超过24小时，则认定该键的过期时间超长。
* 超长过期时间的键需联系持久化组进行备案。

1. Value使用规范

Value的长度

* 键的值理论上不应超过128KByte。

大值的处理

* 大值可在前端压缩后存入Redis，尽量保持Value不超过512Byte（压缩会给前端带来一定的压力，但后端服务sharding比前端扩充成本高）
* 若值大于8Kbyte，小于128Kbyte键需单独启用新的实例来存储。
* 若值大于128Kbyte则被认定为特大值，需与持久化组讨论该类型键值缓存的必要性。上线需项目组负责人邮件确认。如因特大值引起的故障，持久化组有权利对该实例进行offline处理。

Value存储数据类型

* 存储数字时应尽量为整型。因为纯整型可以直接存储在指针位上，无需额外分配一个SDS(Redis最小存储结构)存储。

1. Redis数据类型选择建议

String

* 绝大多数值推荐使用此类型

Hash

* 类JSON格式的报文推荐使用此类型。
* 对于同一类对象，且数量不大优先考虑使用hash代替string，因为hash默认在小于512个fields时会使用压缩存储算法
* 单个hash field不超过1000个。
* 单个value size不大于128byte。
* 如有单个hash field 特别大可以做sharding拆成N个小于1000 field的hash key。
* 规避hgetall操作，特别是field很大的情况下。

Sets

* 单个sets 不超过10000个value。
* 单个value size 不大于64byte。

List

* 单个value size 不大于64byte。

1. Redis程序中严格禁止使用的命令

* Bgrewrite
* Bgsave
* Config get
* Config set
* Config resetstat
* Config rewrite
* Flushdb
* Flushall
* Keys
* Client kill
* Client list
* Slowlog
* Monitor
* Slave of
* Info
* Save
* Exits

1. redis建议参数配置

#使用哨兵结构的Redis数据库的名称  
redis.clusterName =   
#使用的Redis数据库编号必须使用0（非0库会增加2倍OPS）  
redis.database = 0  
#客户端超时时间单位是毫秒,设置和read timeout 这个报错有关，当没有收到服务务器返回结果的时间超过这个设置时候，将报read timeout 这个报错，主动断开与服务器的连接的。参数依赖于实际应用开发需求。请根据实际需要配置。  
redis.timeout=2000   
#允许最大活动对象数,最大连接数，请根据实际需要配置  
redis.maxActive=50  
#最大空闲数，空闲链接数大于maxIdle时，将进行回收，请根据实际需要配置  
redis.maxIdle=50  
#允许最小空闲对象数，低于minIdle时，将创建新的链接，请根据实际需要配置  
redis.minIdle=50  
#指明是否在从池中取出连接前进行检验,如果检验失败,则从池中去除连接并尝试取出另一个,生产环境关闭   
redis.testOnBorrow=flase  
#指明是否在归还到池中前进行检验,生产环境关闭   
redis.testOnReturn =flase  
#指明连接是否被空闲连接回收器进行检验.如果检测失败,则连接将被从池中去除  
redis.testWhileIdle=true

## kafka

1. Api

<dependency>

<groupId>org.springframework.kafka</groupId>

<artifactId>spring-kafka</artifactId>

<version>1.3.0.RELEASE</version>

</dependency>

1. 命名规范

* 队列名topicName由小写字母及下划线组成，对于一对多（包括一对一）的场景，格式为：生产者标识\_业务类型标识\_其他；对于多对一的场景，格式为：消费者标识\_业务类型标识\_其他。
* 例如：bpep\_portal\_web\_download\_async
* 消费者组名groupName与应用名保持一致。

1. 配置规范

* 对于可靠性>吞吐量的场景：

生产者配置：

acks=all 分区主从节点均需返回确认

retries=1 失败重试

request.timeout.ms=2000 发送请求超时时间

max.block.ms=2000 发送最大阻塞时间

autoFlush=true 同步发送

消费者配置：

enable.auto.commit=false 关闭自动提交offset

max.poll.records=10 每次最多拉取消息数

ackMode=RECORD 消息逐条确认消费

* 对于吞吐量>可靠性的场景：

生产者配置：

acks=1 分区主节点需返回确认

retries=1 失败重试

autoFlush=false 异步发送

消费者配置：

enable.auto.commit=true 自动提交offset

max.poll.records=100 每次最多拉取消息数

1. 注意事项

* 使用kafka异步返回上层系统处理结果的，为防止kafka丢失情况，上层系统必须含有主动查询的功能。
* jdk版本要求1.7+, spring版本要求4.1.0.RELEASE+。
* 只有消费者需要配置组名group。
* topic属于生产者和消费者双方配置，请使用相同的topic对接。
* 新topic首次上线，消费者请先于生产者上线，否则生产者在消费者上线前发送的消息不会被消费。
* 请将消息对象转为json字符串发送和接收，建议使用Gson。
* key和value的序列化类：org.apache.kafka.common.serialization.StringSerializer
* 反序列化类：org.apache.kafka.common.serialization.StringDeserializer
* 消费者请做好业务幂等性控制。防止广播消费的队列在部分消费者应用需要补单情况下，其他消费者应用出现业务问题。
* kafka发送方的request.timeout.ms和max.block.ms建议设为2000，降低kafka故障时对平台性能的影响。
* 风控和安全吞吐量较大且可靠性要求不高的队列，发送方acks建议设为0，其他应用一般设为1。非可靠性要求极高且不追求吞吐量的情况下不要设为all或者-1。
* 在消费端按照规范具有业务幂等的情况下，发送方retries最大可以设为1。
* 若消费方的业务处理逻辑比较耗时（一般超过500ms），请设置max.poll.records不要超过10，降低消息重复消费的可能。
* 若消费方的业务处理逻辑极其耗时或者包含调用外部接口等不确定行因素的情况下，建议单独起异步线程处理业务，以免阻塞消费线程。

## rabbitmq

1. Api

<dependency>

<groupId>org.springframework.amqp</groupId>

<artifactId>spring-rabbit</artifactId>

<version>1.5.7.RELEASE</version>

</dependency>

1. 资源创建规范

* 用户名：与标准应用名一致
* vhost：应用所属平台简称+host，例如：交费助手pashost
* exchange：e\_应用标识\_业务类型标识，例如：交费助手短信e\_pas\_sms
* 特殊的，如延迟队列的exchange：e\_应用标识\_业务类型标识\_延迟时间标识
* routing-key：r\_应用标识\_业务类型标识，例如：交费助手短信r\_pas\_sms
* queue：q\_应用标识\_业务类型标识，例如：交费助手短信q\_pas\_sms
* 特殊的，如延迟队列的queue：q\_应用标识\_业务类型标识\_延迟时间标识

1. 权限控制

* 若应用仅需访问所属平台vhost内的queue，一般赋予读写权限。
* 若应用需要跨vhost访问，根据访问类型赋予只读或只写权限。
* 应用侧用户原则上不赋予管理权限

1. 配置规范

* 连接工厂

requestedHeartbeat=240 心跳240秒

connectionTimeout=10000 连接超时10秒

automaticRecoveryEnabled=true 自动重连

* 生产者
* 默认情况下<rabbit:template>中仅需配置id，exchange，routing-key和connection-factory。
* 具体的发送方法中无需再指定routing-key参数。
* 消费者
* 默认情况下<rabbit:listener-container>中仅需配置connection-factory，prefetch，acknowledge，concurrency和max-concurrency，其中，

prefetch=1 每次消费一条消息

acknowledge=auto 消费完成后自动ack

concurrency=10 并发消息线程数

max-concurrency=10 最大并发消费线程数

1. 注意事项

* exchange默认为direct类型，exchange和queue均持久到磁盘。
* exchang和queue一旦创建不可变更，如需变更新申请新的exchange和queue。
* 请应用严格使用分配给自己的用户名/密码等信息，不同的用户名拥有不同的权限，不要乱用。
* exchange和routing key属于生产者配置，queue属于消费者配置，vhost属于双方配置。

## fastdfs

1. Api

<dependency>

<groupId>org.csource</groupId>

<artifactId>fastdfs</artifactId>

<version>1.24</version>

</dependency>

1. 注意事项

* 上传时严禁指定组名group，也不需要处理文件后缀。
* 上传后请将fastdfs返回的groupId+fileId入库保存，并做好与原文件名的对应。
* 下载时请使用文件在数据库中保存的groupId+fileId下载对应文件。
* 根据文件的安全性要求决定是否提供http接口直接访问。
* 凡是涉及用户身份信息等敏感文件的存储，需要先调用cif提供的加密接口然后上传，下载时需要对应解密。
* 上传的单个文件不要超过1GB，大文件尽量拆开上传。
* 无需永久保存的文件，请自行通过定时任务按时效要求及时清理。
* 同时访问两套fastdfs服务时，请先复制定义一个新的org.csource.fastdfs.ClientGlobal，然后在new TrackerClient(TrackerGroup tracker\_group)时传入新的ClientGlobal的tracker\_group获取连接。

## elastic-job

1. Api

<dependency>

<groupId>com.bestpay</groupId>

<artifactId>elastic-job-lite-core</artifactId>

<version>2.1.5.1</version>

</dependency>

<dependency>

<groupId>com.bestpay</groupId>

<artifactId>elastic-job-lite-spring</artifactId>

<version>2.1.5.1</version>

</dependency>

1. 命名规范

* namespace：标准应用名-job

1. 配置规范

* 注册中心基础配置<reg:zookeeper>

base-sleep-time-milliseconds="1000" 重试间隔时间

max-sleep-time-milliseconds="3000" 最大重试间隔时间

session-timeout-milliseconds="6000" 会话过期时间

connection-timeout-milliseconds="3000" 连接超时时间

max-retries="3" 连接重试次数

* 任务基础配置<job:simple>

disabled="false" 是否失效

monitor-execution="true" 运行时监控（执行间隔及耗时短的任务不建议开启）

failover="true" 失效转移

misfire="true" 错误重新执行

overwrite="true" 覆盖注册中心配置

reconcile-interval-minutes="10" 自修复间隔时间

description="任务详细描述" 任务描述

event-trace-rdb-data-source="对应数据源id" 任务执行情况记录数据源

1. 注意事项

* jdk版本要求1.7+, zookeeper客户端版本要求3.4.6+。
* 请在业务上做好每个job的幂等性。
* 若该任务不需要在准生产执行，可在准生产将disabled设为true。
* description字段请不要省略，按要求详细描述该任务用途，方便管理任务。
* 失效转移需要同时开启failover和monitor-execution。
* 在严格保证业务幂等性的情况下，可以开启失效转移，否则不要开启失效转移。重点job请做好业务上的告警，如有异常情况可以手动补触发。
* 若是可容忍错过的job（比如几分钟按状态查询一次的job），或者执行间隔及耗时较短的job，不要开启failover、monitor-execution和misfire功能，不要配置event-trace-rdb-data-source数据源参数。
* 如果任务的执行间隔及执行耗时均比较短，建议不要开启monitor-execution功能。
* 数据源请使用bestpay-hikaricp连接池。

## zookeeper

1. Api

<dependency>

<groupId>org.apache.curator</groupId>

<artifactId>curator-recipes</artifactId>

<version>2.12.0</version>

</dependency>

1. 命名规范

* rootPath：标准应用名

1. 配置规范

baseSleepTimeMs=1000 重试间隔时间

maxRetries=3 重试次数

connectionTimeoutMs=3000 连接超时时间

sessionTimeoutMs=6000 会话超时时间

1. 注意事项

* 分布式锁在zk上的根节点请与标准应用名保持一致，子节点请自行根据业务类型命名。
* 默认创建可重入互斥锁（InterProcessMutex），获取（acquire）的锁用完后请及时释放（release），若需要重入获取，请对应相同次数的释放。
* 默认请使用带超时时间的获取锁方法，抢锁等待超时时间建议设为3秒。
* 每次加锁对象不同的情况下，请在释放锁之后，及时一并删除加锁对象对应的zk节点，防止冗余节点过多影响zk服务。

## Dubbo

1. Api

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<version>2.5.8</version>

</dependency>

1. 配置规范

* <dubbo:application>

name dubbo应用名，请与标准应用名保持一致

* <dubbo:registry>

id 注册中心的id，如有多个注册中心请注意区分不同id

protocol=”zookeeper” 注册中心类型为zookeeper

address 注册中心地址

* <dubbo:monitor>

protocol=”registry” dubbo监控配置，异步采集调用信息，不影响性能

* <dubbo:protocol>

name=”dubbo” 协议名为dubbo

port=”20880” 服务端口为20880

threads 提供服务的总线程数

* <dubbo:service>

interface 注册到zk上的服务接口

ref 上述服务对应的实现类bean id

retries=”0” 失败不重试

timeout 服务超时时间

registry 对应注册中心的id

* <dubbo:consumer>

check=”false” 关闭消费者启动时服务检查

* <dubbo:reference>

id 调用用服务的bean id

interface 调用的服务接口

retries=”0” 失败不重试

timeout 调用超时时间

registry 对应注册中心的id

1. 注意事项

* <dubbo:service>和<dubbo:reference>中的重复配置以<dubbo:reference>中的配置为准。
* <dubbo:application>中的name值必须与标准应用名一致。
* <dubbo:monitor>监控项必须配置。
* <dubbo:protocol>中的threads为应用内所有服务的总线程数，如服务数过多请考虑拆分应用。

# 数据库规范

## mybatis sqlmap规范

1. 变量  
   在 sqlmap 中的变量，要用#号，而不要用$符号

* 如#{appid}。因为${name} 是字面意义的替换，这种形式会有 SQL 注入的漏洞，而#{name} 是替换成绑定变量占位符，不存在 SQL 注入的风险。
* 动态排序条件的输入需要使用 ${name} ，但是应用必须做严格的参数检查。

1. 更新时间  
   数据更新必须修改 updated\_at

* 对表的记录进行更新的时候，必须包含对 updatedAt 字段的更新。
* 错误的写法：

update T\_CONTACT\_INFO

<set>

......

<if test="updatedAt != null">

updated\_at= #{updatedAt}

</if>

</set>

where ID = #{id}

* 正确的写法（当然，这里更推荐直接更新为 sysdate）：

update T\_CONTACT\_INFO

<set>

updated\_at= #{updatedAt ,jdbcType=TIMESTAMP}

......

<set>

where ID = #{id}

1. 动态条件  
   全动态条件查询需要特别小心

后台系统对大表进行查询时，需要注意下面 2个问题：

* 并发控制。很多后台查询，因为业务需要，查询条件很多，任由用户选择，而数据库表索引不可能覆盖所有的查询条件，因此，有时这种后台查询会进行全表扫描，因为扫描大表，速度慢，用户不耐烦，因此就拼命点击页面上的“搜索”按钮，结果导致了同样的查询（而且性能较低）大量并发执行，对整个系统性能危害极大，因此，这种后台应用，应当避免并发查询，在点击“搜索”按钮后，禁用该按钮，防止重复点击。
* 对不合理的查询条件进行控制。对于较大的数据表，不给定任何条件的查询，一般来说是无意义查询，不合理，应当制止；另外，对于一些特大表，比如交易表、账务表、用户表等，连查询⽅方式都应当有所控制，确保所有查询⾛走索引，确保整体性能。举例：一种风险较大的编码方式（全部都是动态条件）
* 性能上不保险的写法：

select count from t\_user usr

<where>

<if test=" userId != null" >

usr.iw\_user\_id = #{userId ,jdbcType= varchar}

</if>

<if test=" email != null" >

and usr.email = #{email ,jdbcType= varchar}

</if>

<if test=" certType!= null" >

and usr.cert\_type = #{ certType,jdbcType= varchar}

</if>

<if test=" certNo!= null" >

and usr.cert\_no = #{ certNo,jdbcType= varchar}

</if>

</where>

* 正确写法，注入保护条件：

select count from t\_user usr

<where>

<if test=" userId != null" >

usr.iw\_user\_id = #{userId ,jdbcType= varchar}

</if>

<if test=" email != null" >

and usr.email = #{email ,jdbcType= varchar}

</if>

<if test=" certType!= null" >

and usr.cert\_type = #{ certType,jdbcType= varchar}

</if>

<if test=" certNo!= null" >

and usr.cert\_no = #{ certNo,jdbcType= varchar}

</if>

<if test=" userId == null" >

<if test=" email == null" >

<if test=" certNo== null" >

and 1=2

</if>

</if>

</if>

</where>

1. 注释  
   SQL片段要有注释

* 使用块注释，而不是行注释 /\* 这儿是注释 \*/
* 注释标明SQL是哪个SQLMap的片段生成，方便TOP SQL的快速定位

<select id="selectByPersonId" resultMap="BaseResultMap" parameterType="int”>

/\* personMapper.selectByPersonId \*/

select

PER\_ID,

PER\_FIRST\_NAME,

PER\_LAST\_NAME,

PER\_BIRTH\_DATE,

PER\_WEIGHT\_KG,

PER\_HEIGHT\_M

from T\_PERSON

where PER\_ID = #{value}

</select>

1. dynamic

* 绝不使用 dynamic 标签动态拼接需要 join 的 table
* join 的表是固定的，join 的条件也必须是固定的。

1. delete语句

* 在平台系统中，是不需要，sqlmap里面delete的语句，数据在生产库中没有删除的概念，只有update失效的概念

## 数据库表设计规范

1. 数据库表

字段设计：

* 在设计时必须包含两个日期字段：created\_at(创建日期)，updated\_at (修改日期)且非空, 对表的记录进行更新的时候，必须包含对 updated\_at 字段的更新。
* 尽可能使用简单数据类型，不要使用类似数组或者嵌套表这种复杂类型，可选用类型：VARCHAR , NUMBER , DATE
* 必须要有主键，且列名是ID，类型必须是 number 。
* 需要 join 的字段，数据类型保持绝对一致。
* 当表的字段数非常多时，可以将表分成两张表，一张作为条件查询表，一张作为详细内容表（主要是为了性能考虑）。
* 当字段的类型为枚举型或布尔型时，建议使用 varchar(1)类型，不要使用 char(1)。
* 字段类型尽量做到“所见即是”原则，尽量避免 ‘1’，’2’ 这种情况 。
* Number 字段不要设置精度，如 number(20,1)，统一设置为 Number，精度由应用程序控制。
* 不能使用 blob、clob、long 等长字段，如果一定要使用的话，需要找 DBA 申请。
* 每张表、每个字段必须都有注释。

命名：

* 表名一般用T\_开头，表示为TABLE；
* 系统表，加上SYS\_， 如：T\_SYS\_；
* 基础表，加上INFO\_，如：T\_INFO\_；
* 日志表，加上LOG\_， 如：T\_LOG\_；
* 配置表，加上CFG\_， 如：T\_CFG\_；
* 字段命名应尽可能使用表达实际含义的英文单词或缩写，不要使用类似“VALUE1”这种无意义的字段名。
* 布尔值类型的字段命名为 is+描述。如 member 表上表示是否为 enabled 的会员的字段命名为 is\_enabled，避免使用驼峰命名法，如 isEnabled，ISV8

1. 索引

索引设计：

* Bitmap 索引通常不适合我们的环境。
* 索引根据实际 SQL，由 DBA 创建。
* 不要创建带约束的索引，所有的约束效果都通过显示创建约束然后再 using index一个已经创建好的普通索引来实现。

命名：

* idx\_<table\_name>\_<column\_name>,各部分以下划线（\_）分割。
* 多单词组成的 column name，取前⼏几个单词⾸首字⺟母，加末单词组成 column\_name。如：sample 表 member\_id 上的索引：idx\_sample\_mid。

1. 约束

设计：

* 主键无意义，由 Sequence 产生的 ID 字段，类型为 number，不能使用组合主键。
* 若要达到唯一性限制的效果，不要创建 unique index，必须显式创建普通索引和约束（pk 或 uk），即先创建一个以约束名命名的普通索引，然后创建一个约束，用

using index ...指定索引。

* 当删除约束的时候，为了确保不影响到 index，最好加上 keep index 参数。
* 主键的内容不能被修改。
* 外键约束一般不在数据库上创建，只表达一个逻辑的概念，由程序控制。
* 当万不得已必须使用外健的话，必须在外健列创建 INDEX。

命名：

* 主键约束： pk\_开头，列名结尾，pk\_<table\_name>；
* unique 约束：uk\_开头，列名结尾，uk\_<table\_name>\_<column\_name>；
* check 约束： ck 结尾，ck\_<table\_name>\_<column\_name>；

1. SEQUENCE

命名：

* seq\_<table\_name> ，每个表必须有自己对应的 sequence。

1. 触发器（禁用）

命名：

* <table\_name>\_A(After)B(Before)I(Insert)U(Update)D(Delete)\_trg。
* 若是用于同步的触发器以 sync 作为前缀：sync\_<table\_name>\_trg。

1. 过程、函数、包（禁用）

命名：

* 过程以 proc\_开头，函数以 func\_开头，包以 pkg\_开头。
* 变量命名约定：本地变量以 v\_为前缀，参数以 p\_为前缀，可以带\_I(输入)，\_O(输出)、\_IO(输入输出）表示参数的输入输出类型。

## SQL规范

1. DDL  
   在代码中不允许出现任何 DDL 语句

* DDL 语句一律由 DBA 编写并统一执行。
* 即使有，也是极有限的场景需要 。

1. 绑定变量  
   写 SQL 的时侯一定要使用绑定变量

* 对于极少数情况下不使用绑定变量提高性能，使用之前一定要和 DBA 沟通

1. 前缀  
   写 SQL 的时候一定要给每个字段指定表名做前缀

* 比如 select a.id,a.name from test a; 好处是一来带来性能的提升，二来可以避免一些错误的发生。

1. select \*  
   请不要写 select \* 这样的代码，指定需要的字段名
2. 函数  
   避免在 where 子句中对字段施加函数  
   通常，不允许在字段上添加函数或者表达式，这样将导致索引失效，如：

* 错误的写法：

select \* from iw\_account\_log where to\_char ( trans\_dt, 'yyyy-mm-dd') = '2007-04-04';

select qty from product where p\_id + 12 = 168;

* 正确的写法：

select \* from iw\_account\_log

where trans\_dt >= to\_date ( '2007-04-04', 'yyyy-mm-dd') and trans\_dt < to\_date( '2007-04-05', 'yyyy-mm-dd');

select qty from product where p\_id = 168 - 12;

* 如果是业务要求的除外，但需要在编写时咨询 DBA

特别注意，当表连接时，用于连接的两个表的字段如果数据类型不一致，则必须在一边加上类型转换的函数，如：

* 错误的写法（a.id 是 number 类型，⽽而 b.operator\_number 是 char 类型）：

select count from adm\_user a, adm\_action\_log b where a.id = b.operator\_number and a.username = '小钗';

* 正确的写法：

select count from adm\_user a, adm\_action\_log b where to\_char(a.id) = b.operator\_number and a.username = '小钗';

select count from adm\_user a, adm\_action\_log b where a.id = to\_number(b.operator\_number) and a.username = '小钗';

* 上面两种写法哪个正确？遇到这种情况时必须咨询 DBA！

1. 隐式转换  
   严格要求使用正确类型的变量，杜绝 oracle 做隐式类型转换的情况

* 推荐在 sqlmap 的变量中指定变量的数据类型，如：select \* from iw\_user where iw\_user\_id = #{userid,jdbcType=VARCHAR}
* 对于变量数据类型错误导致 SQL 严重性能问题的，按严重的编码错误 Bug 处理

1. 模糊查询  
   全模糊查询无法使用 INDEX，应当尽可能避免，比如：select \* from table where name like '%jacky%’;

* 除非后台用户在很小的表上进行模糊查询。
* 大表模糊查询，可将数据载入全文搜索引擎进行搜索。

1. 分页写法

包含排序逻辑的分页查询写法，必须是三层 select 嵌套：

* 错误的写法：

SELECT t1.\*

FROM (

SELECT t.\*, ROWNUM rnum

FROM t\_trade\_base t

WHERE seller\_account = #{sellerAccount}

AND created\_at >= TO\_DATE (#{beginDate}, 'yyyy-mm-dd')

AND created\_at < TO\_DATE (#{endDate}, 'yyyy-mm-dd')

ORDER BY created\_at DESC

) t1

WHERE t1.rownum >= #{beginNum} AND t1.rownum < #{endNum}

* 正确的写法：

SELECT t2.\*

FROM (

SELECT t1.\*, ROWNUM rnum

FROM (

SELECT t.\*

FROM t\_trade\_base t

WHERE seller\_account = #{sellerAccount}

AND created\_at >= TO\_DATE (#{beginDate}, 'yyyy-mm-dd')

AND created\_at < TO\_DATE (#{endDate}, 'yyyy-mm-dd')

ORDER BY created\_at DESC) t1

WHERE t1.rownum <= #{beginNum}) t2

WHERE t2.rownum >= #{endNum}

1. HINT

* HINT的使用均有DBA编写和评估

## 连接池设置规范

1. Oracle 和MySQL

* 新应用使用Oracle和MySQL数据库时，统一采用自主开发的[hikaricp](https://wiki.bestpay.com.cn/display/JCJS/bestpay-hikaricp)连接池。
* 这里是一些内置的动态属性，这些属性可以出现在配置文件中，或者项目指定的zookeeper路径中

| **属性名** | **动态属性名** | **默认值** | **描述** | **推荐配置** |
| --- | --- | --- | --- | --- |
| connectionTimeout | config.connection.timeout | 30000 | 从连接池中获取连接的等待时间，最小1000ms，默认 30000 (30 seconds) | 30000 |
| validationTimeout | config.validation.timeout | 5000 | 测试连接是否活着的最大时间，这个值要比connectionTimeout小，最小值1000ms | 5000 |
| idleTimeout | config.idle.timeout | 600000 | 连接池中连接的空闲时间,0表示不从连接池中移除,最小值10s | 600000 |
| maxLifetime | config.max.life.time | 1800000 | 连接最大生命期，这个配置项一定要设置，0表示无限生命期，最少30分钟 | 1800000 |
| leakDetectionThreshold | config.leak.detection.threshold | 0 | 连接泄漏检测阈值，0表示不开启检测，开启检测的最小值是2s | 0 |
| maximumPoolSize | config.max.pool.size | 10 | 连接池最大 | 30 |
| minimumIdle | config.min.idle | 1 | 最小的连接空闲数 | 30 |
| autoCommit | config.is.auto.commit | true | 连接是否自动提交 | true |
| dataSourceConnection | config.data.source.connection |  | 数据源连接信息，如username=root1;password=root1;jdbcUrl=jdbc:mysql://localhost:3306/hikaricp |  |

* 不可配置属性

| **属性名** | **描述** |  |
| --- | --- | --- |
| jdbcUrl | 数据库连接 | 该值从dataSourceConnection中解析，在配置文件中不可配置，不可通过<property name="jdbcUrl" value="xx" />配置 |
| password | 数据库密码 | 该值从dataSourceConnection中解析，在配置文件中不可配置，不可通过<property name="password" value="xx" />配置 |
| username | 数据库用户名 | 该值从dataSourceConnection中解析，在配置文件中不可配置，不可通过<property name="username" value="xx" />配置 |

* 详细信息：
* https://wiki.bestpay.com.cn/display/JCJS/bestpay-hikaricp

1. **MongoDB**

mongo.replicaSet=ip1:port,ip2:port,..

mongo.connectionsPerHost=100

mongo.threadsAllowedToBlockForConnectionMultiplier=10

mongo.connectTimeout=10000 #连接超时时间（毫秒）

mongo.maxWaitTime=10000

mongo.autoConnectRetry=true #自动重连

mongo.socketKeepAlive=true #scoket保持活动

mongo.socketTimeout=50000 #scoket超时时间

mongo.slaveOk=true #读写分离

mongo.writeNumber=1

mongo.riteTimeout=0

mongo.writeFsync=true

## 数据库事务

1. [spring @Transactional 标签注意事项](https://wiki.bestpay.com.cn/pages/viewpage.action?pageId=7013845)

* 在需要事务管理的地方加@Transactional 注解。@Transactional 注解可以被应用于接口定义和接口方法、类定义和类的 public 方法上。
* @Transactional 注解只能应用到 public 可见度的方法上。 如果你在 protected、private 或者 package-visible 的方法上使用 @Transactional 注解，它也不会报错， 但是这个被注解的方法将不会展示已配置的事务设置。
* 注意仅仅 @Transactional 注解的出现不足于开启事务行为，它仅仅 是一种元数据。必须在配置文件中使用配置元素，才真正开启了事务行为。
* @Transactional 的事务开启 ，或者是基于接口的 或者是基于类的代理被创建。所以在同一个类中一个方法调用另一个方法有事务的方法，事务是不会起作用的。

## 更多规范





# 安全规范

## 术语表述

1. 专有名词缩略语

表 专有名词对照表

|  |  |  |
| --- | --- | --- |
| **简称** | **名称** | **描述** |
| MAC | Message Authentication Code | 消息认证码 |
| DDoS | Distributed Denial of Service | 分布式拒绝服务 |
| PCI DSS | Payment Card Industry (PCI) Data Security Standard | 第三方支付行业(支付卡行业PCI DSS)数据安全标准 |
| CVC2/CID/CVN2/CAV2 | Validation Code | 各类信用卡安全码  CVV2:VISA 卡  CVC2:万事达卡  CID：运通卡  CVN2:银联卡  CAV2：JCB卡 |
| CVV | Card Verification Value | CVV代码 |
| PIN | Personal Identification Number | SIM卡的个人识别密码 |

1. 威胁分析模式STRIDE 模型 参考标准

* 此为微软的安全建模的模型，可作为甜橙技术设计产品所需考虑的安全属性与设计原则。STRIDE 包含有：Spoofing（假冒），Tampering（篡改），Repudiation（否认），Information Disclosure（信息泄漏），Denial of Service（拒绝服务），Elevation of Privilege（提升权限）。

表 威胁分析模型STRIDE模型参考标准

|  |  |  |  |
| --- | --- | --- | --- |
| **威胁定义** | **含义** | **定义** | **对应安全属性** |
| Spoofing | 伪装 | 冒充他人身份 | 认证 |
| Tampering | 篡改 | 修改数据或代码 | 完整性 |
| Repudiation | 抵赖 | 否认做过的事情 | 不可否定性 |
| InformationDisclosure | 信息泄露 | 机密信息泄露 | 机密性 |
| Denial of Service | 拒绝服务 | 拒绝服务 | 可用性 |
| Elevation of Privilege | 提升权限 | 未经授权获得许可 | 授权 |

1. PSI DSS 敏感数据规范

PCI DSS 敏感数据规范

|  |  |
| --- | --- |
| **持卡人数据包括** | **敏感验证数据包括** |
| 主账号 | 全磁道数据（磁条数据或芯片上的 等效数据） |
| 失效日 | CAV2/CVC2/CVV2/CID/CVN2/ |
| 业务码 | PIN/PIN 数据块 |

1. PCI DSS 数据存储规范

表PCI DSS数据存储规范

|  |  |  |  |
| --- | --- | --- | --- |
| **账户数据** | **数据元素** | **允许存储** | **存储数据的不可读性** |
| 持卡人数据 | 主账户（PAN） | 是 | 是 |
| 持卡人姓名 | 是 | 否 |
| 业务码 | 是 | 否 |
| 失效日 | 是 | 否 |
| 敏感验证数据 | 全磁道数据 | 否 | 不能存储 |
| CAV2/CVC2/CVV2/CID | 否 | 不能存储 |
| PIN/PIN 数据块 | 否 | 不能存储 |

1. 甜橙安全定义

* 强效加密
* 加解密服务应符合以下表所示要求

表 加解密算法安全要求

|  |  |
| --- | --- |
| 算法 | 传输方式 |
| **摘要算法** | 应使用SHA256及以上强度算法 |
| **对称密码算法** | AES算法得密钥长度应为128bit及以上  3DES算法得密钥长度应为128bit及以上 |
| **非对称密码算法** | RSA算法的密钥算法长度应为2048bit及以上  椭圆曲线算法的密钥长度应为224bit及以上  DSA算法的密钥长度应为2048bit及以上  ECDSA算法的密钥长度应为224bit及以上 |
| **随机数生成器** | 应符合GB/T 32915规范得规定 |

## 数据传输与存储规范

* 敏感数据传输、存储规范标准，按照以下级别定义。

表 传输存储级别定义

|  |  |
| --- | --- |
| **类型** | **级别标准** |
| 传输方式 | 不限制 |
| 明文传输 |
| 脱敏传输 |
| 密文传输(需描述算法) |
| 日志存储类型 | 不限制 |
| 明文存储 |
| 脱敏存储 |
| 密文存储（需描述算法） |

表 传输存储规范

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 传输方式 | 日志存储 | 算法描述 | 备注 |
| **证件号码** | 脱敏/加密 | 脱敏 |  |  |
| **银行卡账号** | 脱敏/加密 | 脱敏 |  |  |
| **持卡人-主账号** | 脱敏/加密 | 脱敏/加密 |  |  |
| **敏感字段-全磁道数据** | 加密 | 不能存储 |  |  |
| **CVV码** | 加密 | 不能存储 |  |  |
| **安全码** | 加密 | 不能存储 |  |  |
| **PIN** | 加密 | 不能存储 |  |  |
| **卡验证代码** | 加密 | 不能存储 |  |  |
| **登录密码** | 加密 | 不能存储 |  |  |
| **支付密码** | 加密 | 不能存储 |  |  |

## ftp数据传输规范

* 【规范】上海广东侧的ftp数据传输，应对传输文件、资料，进行对称密钥加密传输，保证数据安全性。密钥强度请参见上文“强效加密”标准。

## 网络通信安全

* 本部分内容值数据在网络传输过程中，采用的通信协议和安全认证方式，不包括网络基础设施方面的内容。

1. **通讯协议**

* 【规范】应使用强壮的加密算法和安全协议保护客户端与服务器之间所有连接。例如使用SSL/TLS和IPSEC协议
* 【规范】【改造】如果使用SSL协议，应使用3.0及其以上相对高版本的协议，取消对低版本协议的支持。
* 【规范】【改造】客户端到服务器的SSL加密密钥长度应不低于128位；用于签名的RSA密钥长度应不低于2048位，用于签名的ECC密钥长度应不低于224位。
* 【规范】应可防止对交易报文的重放攻击。建议加入时间戳、随机数等动态变化随机因子。

1. **安全认证**

* 【规范】【改造】服务器与客户端（IOS、安卓、WP8等APP）应进行双向身份认证。
* 【规范】整个通讯期间，经过认证的通讯线路应一致保持安全连接状态。
* 【建议】超时断开机制。当连接超过一定时间后，自动关闭当前连接，客户再次操作时必须重新登录。
* 【建议】服务端应判断同义词登录后的所有操作必须使用同一个IP地址和MAC地址，否则服务器端自动终止会话。

## Web应用安全

1. **资源控制**

* 【规范】应能够对系统的最大并发会话连接数进行限制。
* 【规范】应能够对单个用户的多重并发会话进行限制。
* 【规范】应能够对一个时间段内可能的并发会话连接数进行限制。
* 【规范】当应用系统通信双方中的一方在指定时间内未作任何响应，另一方应能够自动结束会话。

1. **会话安全**

* 【规范】会话标识应随机并且唯一。
* 【规范】会话过程中应维持认证状态，防止客户通过直接输入登录后的地址访问登录后的页面。
* 【规范】转账交易后，应确保使用浏览器的“后退”功能无法查看上一交易页面的重要客户信息。

1. **防止敏感信息泄露**

* 【规范】如果在生产服务器上保留部分与web应用程序无关的文件，应该为期创建单独的目录，使其与Web应用程序隔离，并且对此目录进行严格的访问控制。
* 【规范】禁止在Web应用服务器保存客户敏感信息。

1. **防止SQL注入攻击**

* 【规范】网上银行系统 Web 服务器应用程序应对客户提交的所有表单、参数进行有效的合法性判断和非法字符过滤，防止攻击者恶意构造 SQL 语句实施注入攻击。
* 【规范】禁止仅在客户端以脚本形式对客户的输入进行合法性判断和参数字符过滤。

# 安全开发规范

* 为了方便开发人员对安全编码的学习，本章提供了大量常见漏洞和易错的案例。案例中包含正确和错误两方面代码，以便用于参考。案例按照数据的校验、认证与授权、Cookie与会话管理、加密与解密、敏感数据、错误处理、日志安全、未验证的重定向与转发几个方面进行分类阐述。

## 数据的校验

1. SQL注入

* 漏洞描述：

SQL注入攻击指的是通过构建特殊的输入作为参数传入Web应用程序，而这些输入大都是SQL语法里的一些组合，通过执行SQL语句进而执行攻击者所要的操作，其主要原因是程序没有细致地过滤用户输入的数据，致使非法数据侵入系统。

* 检测方法：

检查代码中是否存在将输入的参数值直接拼接成SQL语句（不使用占位符）执行的情况。

检查web.xml文件中是否使用过滤器进行SQL注入防护，注意该过滤器所过滤的路径有哪些，比如过滤的是以“.do”、“.action”结尾的路径，那么以“.jsp”结尾的路径则可能存在SQL注入点。每个系统根据需求的不同，可能会对某些特殊路径不予以过滤，在检查的时候要特别注意。

检查过滤器中过滤的危险字符是否完整（具体可参见3.1.5）。

* 不合规的代码示例：

public User login(String username, String password){

User user = null;

// 获取数据库连接Connection对象

Connection conn = DataBaseUtil.getConnection();

//创建Statement 对象

Statement stm =conn.createStatement();

// 根据用户名及密码查询用户信息

String sql = "select \* from tb\_user where username = '"+ username +"' and password = '"+ password +"'";

try {

// 执行查询获取结果集

ResultSet rs = stm.executeQuery(sql);

// 判断结果集是否有效

if(rs.next()){

// 实例化一个用户对象

user = new User();

// 对用户对象属性赋值

user.setId(rs.getInt("id"));

user.setUsername(rs.getString("username"));

user.setPassword(rs.getString("password"));

user.setSex(rs.getString("sex"));

user.setTel(rs.getString("tel"));

user.setPhoto(rs.getString("photo"));

user.setEmail(rs.getString("email"));

}

// 释放此 ResultSet 对象的数据库和 JDBC 资源

rs.close();

// 释放此 PreparedStatement 对象的数据库和 JDBC 资源

ps.close();

} catch (Exception e) {

e.printStackTrace();

}finally{

// 关闭数据库连接

DataBaseUtil.closeConnection(conn);

}

return user;

}

* 不合规说明：

查询用户信息的SQL语句是通过参数拼接而成的，未使用占位符进行预编译的方式。

* 合规的代码示例：

public User login(String username, String password){

User user = null;

// 获取数据库连接Connection对象

Connection conn = DataBaseUtil.getConnection();

// 根据用户名及密码查询用户信息

String sql = "select \* from tb\_user where username = ? and password = ?";

try {

// 获取PreparedStatement对象

PreparedStatement ps = conn.prepareStatement(sql);

// 对SQL语句的占位符参数进行动态赋值

ps.setString(1, username);

ps.setString(2, password);

// 执行查询获取结果集

ResultSet rs = ps.executeQuery();

// 判断结果集是否有效

if(rs.next()){

// 实例化一个用户对象

user = new User();

// 对用户对象属性赋值

user.setId(rs.getInt("id"));

user.setUsername(rs.getString("username"));

user.setPassword(rs.getString("password"));

user.setSex(rs.getString("sex"));

user.setTel(rs.getString("tel"));

user.setPhoto(rs.getString("photo"));

user.setEmail(rs.getString("email"));

}

// 释放此 ResultSet 对象的数据库和 JDBC 资源

rs.close();

// 释放此 PreparedStatement 对象的数据库和 JDBC 资源

ps.close();

} catch (Exception e) {

e.printStackTrace();

}finally{

// 关闭数据库连接

DataBaseUtil.closeConnection(conn);

}

return user;

}

* 合规说明：

SQL语句的执行是使用占位符进行预编译的方式，参数化查询后得到结果。

1. 命令注入

* 漏洞描述：

命令注入，此处所指的是操作系统命令注入，由于命令的部分或全部指令来自用户输入的数据，一旦被攻击者利用，往往可被彻底拿下服务器，后果不堪设想。

* 检测方法：

检查代码中是否使用Runtime.getRuntime().exec()方法，传入方法的参数是否可控，如果系统命令拼接语句中的变量来自不可信输入，则存在漏洞。

* 不合规的代码示例：

<%

String fileName = (String) request.getParameter("fileName");

String source\_name = (String) request.getParameter("source\_name");

// 上传到wl路径

String webpath = Configuration.getValue("PATH\_UPLOAD2WEB").trim();

String sErrorMessage = " ";

int len1 = 0, len2 = 0;

do {

len1 = fileName.length();

fileName = fileName.replaceAll("\\.\\./", "");

len2 = fileName.length();

} while (len1 != len2);

String sSaveName = webpath + "/" + fileName;

int upLoadFlag = 0;

String ErrorInfo = "";

// 新建一个SmartUpload对象

SmartUpload mySmartUpload = new SmartUpload();

try {

// 上传初始化

mySmartUpload.initialize(pageContext);

mySmartUpload.upload();

} catch (Exception ex) {

upLoadFlag = -1;

sErrorMessage = "上载文件传输中出错！";

}

try {

com.jspsmart.upload.File file1 = mySmartUpload.getFiles().getFile(0);

if (!file1.isMissing()) {

file1.saveAs(sSaveName);

}

String \_shellpath = Configuration.getValue("PATH\_SHELL");

String exePath = "sh " + \_shellpath;

String commandString = "WebToAll.sh " + fileName;

String exeString = exePath + commandString;

// 执行命令

Process p = Runtime.getRuntime().exec(exeString);

p.waitFor();

} catch (Exception ex) {

upLoadFlag = -1;

sErrorMessage = "上载文件存储时出错！";

}

%>

* 不合规说明：

程序在此处执行系统命令时，未对fileName参数过滤“&”、“|”、“;”。

* 合规的代码示例：

<%

String fileName = (String) request.getParameter("fileName");

String source\_name = (String) request.getParameter("source\_name");

// 上传到wl路径

String webpath = Configuration.getValue("PATH\_UPLOAD2WEB").trim();

String sErrorMessage = " ";

int len1 = 0, len2 = 0;

do {

len1 = fileName.length();

fileName = fileName.replaceAll("\\.\\./", "").replaceAll("[&\\|;]", "");

len2 = fileName.length();

} while (len1 != len2);

String sSaveName = webpath + "/" + fileName;

int upLoadFlag = 0;

String ErrorInfo = "";

// 新建一个SmartUpload对象

SmartUpload mySmartUpload = new SmartUpload();

try {

// 上传初始化

mySmartUpload.initialize(pageContext);

mySmartUpload.upload();

} catch (Exception ex) {

upLoadFlag = -1;

sErrorMessage = "上载文件传输中出错！";

}

try {

com.jspsmart.upload.File file1 = mySmartUpload.getFiles().getFile(0);

if (!file1.isMissing()) {

file1.saveAs(sSaveName);

}

String \_shellpath = Configuration.getValue("PATH\_SHELL");

String exePath = "sh " + \_shellpath;

String commandString = "WebToAll.sh " + fileName;

String exeString = exePath + commandString;

// 执行命令

Process p = Runtime.getRuntime().exec(exeString);

p.waitFor();

} catch (Exception ex) {

upLoadFlag = -1;

sErrorMessage = "上载文件存储时出错！";

}

%>

* 合规说明：

对fileName参数过滤危险字符“&”、“|”和“;”。

1. 跨站脚本

* 漏洞描述：

跨站脚本攻击（也称为XSS）指利用网站漏洞从用户那里恶意盗取信息。用户在浏览网站、使用即时通讯软件、甚至在阅读电子邮件时，通常会点击其中的链接。攻击者通过在链接中插入恶意代码，就能够盗取用户信息。攻击者通常会用十六进制（或其他编码方式）将链接编码，以免用户怀疑它的合法性。网站在接收到包含恶意代码的请求之后会产成一个包含恶意代码的页面，而这个页面看起来就像是那个网站应当生成的合法页面一样。

* 检测方法：

检测web.xml中是否配置过滤器进行危险字符的过滤，如果存在全局过滤器，检查该过滤器的过滤规则是否健全，是否可被绕过。

检查页面输出的地方如果使用了<%=>这种写法，则可能存在风险！<bean:write filter="true">标签和<c:out>标签是安全的，如果使用的是<bean:write filter="false">b标签则可能存在风险。

* 不合规的代码示例：

<form method="post" id="appWinForm" name="appWinForm" action="${root }/app/application.do">

<input type="hidden" id="app\_userparam" name="app\_userparam" value="<%=request.getParameter("sso\_username")%>" />

<input type="hidden" id="app\_ticket" name="app\_ticket" value="<%=request.getParameter("sso\_ticket") %>" />

<input type="hidden" id="MAIN\_ID" name="MAIN\_ID" value="<%=request.getParameter("MAIN\_ID")%>" />

<input type="hidden" id="cas\_session\_id" name="cas\_session\_id" value="<%=request.getParameter("cas\_session\_id")%>" />

</form>

不合规说明：

对用户输入、输出的数据未做任何危险字符的过滤。

合规的代码示例：

<form method="post" id="appWinForm" name="appWinForm" action="${root }/app/application.do">

<input type="hidden" id="app\_userparam" name="app\_userparam" value="<c:out value="<%=request.getParameter("sso\_username") %>"></c:out>" />

<input type="hidden" id="app\_ticket" name="app\_ticket" value="<c:out value="<%=request.getParameter("app\_ticket") %>"></c:out>" />

<input type="hidden" id="MAIN\_ID" name="MAIN\_ID" value="<c:out value="<%=request.getParameter("MAIN\_ID") %>"></c:out>" />

<input type="hidden" id="cas\_session\_id" name="cas\_session\_id" value="<c:out value="<%=request.getParameter("cas\_session\_id") %>"></c:out>" />

</form>

* 合规说明：

使用了<c:out>安全标签来输出用户的数据。

1. 任意文件上传

* 漏洞描述：

允许用户上传任意文件可能会让攻击者注入危险内容或恶意代码，并在服务器上运行。由于文件上传功能实现代码没有严格限制用户上传的文件后缀以及文件类型，导致允许攻击者向某个可通过 Web 访问的目录上传任意动态脚本文件（JSP、PHP等），并能够将这些文件传递给 web服务器进行解释，就可以在远程服务器上执行该脚本。

* 检测方法：

检查服务端代码中有没有对可上传的文件类型进行限制，比如采用白名单方式。禁止被上传危险文件类型，比如：jsp、php、html、js。

* 不合规的代码示例：

public ActionForward uploadFile(ActionMapping mapping, ActionForm form,

HttpServletRequest request, HttpServletResponse response)

{

PlanFlowForm planFlowForm = (PlanFlowForm) form;

String city = SessionInfoUtil.getStaffSessBean(request.getSession()).getCityCode();

String filePath = request.getRealPath(UPLOAD\_PATH);

String newFileName = null;

String filename = null;

try{

InputStream in = planFlowForm.getUpFile().getInputStream();

filename = planFlowForm.getUpFile().getFileName();

newFileName = PlanFlowModule.consFileName(city,filename);

try

{

BufferedReader is= new BufferedReader(new InputStreamReader(in));

StringBuffer out = new StringBuffer();

ByteArrayOutputStream baos = new ByteArrayOutputStream();

OutputStream bos = new FileOutputStream(filePath + "/" +newFileName);

int bytesRead = 0;

byte[] buffer = new byte[8192];

while ((bytesRead = in.read(buffer, 0, 8192)) != -1)

{

bos.write(buffer, 0, bytesRead);

}

bos.close();

in.close();

}

catch(Exception e)

{

e.printStackTrace();

}

}

catch(Exception e)

{

log.error(e);

}

request.setAttribute("filename", newFileName);

request.setAttribute("showname", filename);

return mapping.findForward("filewin");

}

* 合规的代码示例：

public static final Pattern p = Pattern.compile("(\\.txt|\\.xls|\\.xlsx|\\.docx|\\.pdf|\\.gif|\\.bmp|\\.jpg|\\.png)");

public static boolean fileNameCheck(String str){

str = str.toLowerCase();

Matcher matcher = p.matcher(str);

if(matcher.find()){

return true;

}else{

return false;

}

}

public ActionForward uploadFile(ActionMapping mapping, ActionForm form,

HttpServletRequest request, HttpServletResponse response)

{

PlanFlowForm planFlowForm = (PlanFlowForm) form;

String city = SessionInfoUtil.getStaffSessBean(request.getSession()).getCityCode();

String filePath = request.getRealPath(UPLOAD\_PATH);

String newFileName = null;

String filename = null;

try{

filename = planFlowForm.getUpFile().getFileName();

//判断文件名的后缀是否在白名单中，如果不存在，则跳转到错误页面

if(!fileNameCheck(filename)){

request.setAttribute("filename", newFileName);

request.setAttribute("showname", filename);

return mapping.findForward("filewin");

}

InputStream in = planFlowForm.getUpFile().getInputStream();

newFileName = PlanFlowModule.consFileName(city,filename);

try

{

BufferedReader is= new BufferedReader(new InputStreamReader(in));

StringBuffer out = new StringBuffer();

ByteArrayOutputStream baos = new ByteArrayOutputStream();

OutputStream bos = new FileOutputStream(filePath + "/" +newFileName);

int bytesRead = 0;

byte[] buffer = new byte[8192];

while ((bytesRead = in.read(buffer, 0, 8192)) != -1)

{

bos.write(buffer, 0, bytesRead);

}

bos.close();

in.close();

}

catch(Exception e)

{

e.printStackTrace();

}

}

catch(Exception e)

{

log.error(e);

}

request.setAttribute("filename", newFileName);

request.setAttribute("showname", filename);

return mapping.findForward("filewin");

}

* 合规说明：

采用白名单的方式检测上传文件的后缀名，防止攻击者上传恶意脚本文件。

1. 任意文件下载

* 漏洞描述：

在文件下载过程中，程序往往会从前端传入文件的名称或者文件的完整地址用于下载。由于程序中未对“../”目录跳转字符进行安全过滤，导致攻击者可以下载任意目录下的文件。

* 检测方法：

检查文件的下载路径是否可以控制（传入的参数值是否为下载路径的一部分）。

检查是否对不可信参数值过滤了“../”这样的特殊字符，防止下载任意目录下的文件。

检查过滤“../”的代码是否存在可被绕过的危险，过滤应该采用循环过滤的方式，直到过滤完全（即过滤之后的值中不存在“../”）。

* 不合规的代码示例：

public String downLoad() throws Exception {

String fileName = Struts2Utils.getRequest().getParameter("fileName");

try {

fileName = URLDecoder.decode(fileName, "UTF-8");

} catch (UnsupportedEncodingException ex) {

logger.error("downLoad exception"

+ ExceptionUtils.getFullStackTrace(ex));

}

String path = SmpConfig.getConfig(CHECK\_ATTACHMENT);

File downFile = new File(path + fileName.trim());

HttpServletResponse response = Struts2Utils.getResponse();

try {

if (!downFile.exists()) {

response.sendError(404, "File not found!");

return Action.NONE;

}

response.setContentType("application/x-msdownload;charset=UTF-8");

response.setHeader("Content-Disposition", "attachment; filename="

+ new String(this.parseFileName(fileName).getBytes("GBK"),

"ISO8859-1"));

OutputStream out = response.getOutputStream();

FileInputStream in = new FileInputStream(downFile);

byte[] buf = new byte[BUFFER\_SIZE];

int len = -1;

while ((len = in.read(buf)) != -1) {

out.write(buf, 0, len);

}

in.close();

out.flush();

} catch (Exception ex) {

logger.error("download file exception !".concat(ExceptionUtils

.getFullStackTrace(ex)));

}

return Action.NONE;

}

* 不合规说明：

对于不可信的fileName参数，程序将该参数的值组合成完整的文件路径，且未对目录跳转字符("../")进行安全过滤。

* 合规的代码示例：

public String downLoad() throws Exception {

String fileName = Struts2Utils.getRequest().getParameter("fileName");

try {

fileName = URLDecoder.decode(fileName, "UTF-8");

} catch (UnsupportedEncodingException ex) {

logger.error("downLoad exception"

+ ExceptionUtils.getFullStackTrace(ex));

}

int len1 = 0, len2 = 0;

do {

len1 = fileName.length();

fileName = fileName.replaceAll("\\.\\./", "");

len2 = fileName.length();

} while (len1 != len2);

String path = SmpConfig.getConfig(CHECK\_ATTACHMENT);

File downFile = new File(path + fileName.trim());

HttpServletResponse response = Struts2Utils.getResponse();

try {

if (!downFile.exists()) {

response.sendError(404, "File not found!");

return Action.NONE;

}

response.setContentType("application/x-msdownload;charset=UTF-8");

response.setHeader("Content-Disposition", "attachment; filename="

+ new String(this.parseFileName(fileName).getBytes("GBK"),

"ISO8859-1"));

OutputStream out = response.getOutputStream();

FileInputStream in = new FileInputStream(downFile);

byte[] buf = new byte[BUFFER\_SIZE];

int len = -1;

while ((len = in.read(buf)) != -1) {

out.write(buf, 0, len);

}

in.close();

out.flush();

} catch (Exception ex) {

logger.error("download file exception !".concat(ExceptionUtils

.getFullStackTrace(ex)));

}

return Action.NONE;

}

* 合规说明：

对于不可信的fileName参数，程序采用循环过滤的方式，对目录跳转字符("../")进行安全过滤。

1. 任意文件删除

* 漏洞描述：

在文件删除过程中，程序往往会从前端传入文件的名称或者文件的完整地址用于删除。由于程序中未对“../”目录跳转字符进行安全过滤，导致攻击者可以删除任意目录下的文件。

* 检测方法：

检查文件路径是否可以被控制（传入的参数值是否为路径的一部分）。

检查是否对不可信参数值过滤了“../”这样的特殊字符，防止删除任意目录下的文件。

检查过滤“../”的代码是否存在可被绕过的危险，过滤应该采用循环过滤的方式，直到过滤完全（即过滤之后的值中不存在“../”）。

* 不合规的代码示例：

public void deleteAttachFile() {

String msg = "";

try {

HttpServletResponse response = ServletActionContext.getResponse();

String fileName = this.request.getParameter("uploadFileName");

File file = new File(ServletActionContext.getServletContext().getRealPath("/upload") + "/" + fileName);

if (file.exists()) {

if (file.delete()) {

response.getOutputStream().print("1");// 删除成功

} else {

response.getOutputStream().print("0"); // 删除失败

}

} else {

response.getOutputStream().print("2"); // 文件不存在

}

} catch (IOException e) {

msg = e.getMessage();

e.printStackTrace();

}

request.setAttribute("msg", msg);

}

* 不合规说明：

对于不可信的uploadFileName参数，程序将该参数的值组合成完整的文件路径，且未对目录跳转字符("../")进行安全过滤。

* 合规的代码示例：

public void deleteAttachFile() {

String msg = "";

try {

HttpServletResponse response = ServletActionContext.getResponse();

String fileName = this.request.getParameter("uploadFileName");

int len1=0,len2=0;

do{

len1 = fileName.length();

fileName = fileName.replaceAll("\\.\\./", "");

len2 = fileName.length();

}while(len1 != len2);

File file = new File(ServletActionContext.getServletContext().getRealPath("/upload") + "/" + fileName);

if (file.exists()) {

if (file.delete()) {

response.getOutputStream().print("1");// 删除成功

} else {

response.getOutputStream().print("0"); // 删除失败

}

} else {

response.getOutputStream().print("2"); // 文件不存在

}

} catch (IOException e) {

msg = "操作异常，附件删除失败！";

e.printStackTrace();

}

request.setAttribute("msg", msg);

}

* 合规说明：

对于不可信的uploadFileName参数，程序采用循环过滤的方式，对目录跳转字符("../")进行安全过滤。

1. 任意文件读取与写入

* 漏洞描述：

在文件读取过程中，根据系统的业务需求，程序往往会从前端传入文件的名称或者文件的完整地址用于文件读取。由于程序中未对“../”目录跳转字符进行安全过滤，导致攻击者可以读取或写入任意目录下的文件。

* 检测方法：

检查文件路径是否可以被控制（传入的参数值是否为路径的一部分）。

检查是否对不可信参数值过滤了“../”这样的特殊字符，防止操作任意目录下的文件。

检查过滤“../”的代码是否存在可被绕过的危险，过滤应该采用循环过滤的方式，直到过滤完全（即过滤之后的值中不存在“../”）。

* 不合规的代码示例：

<%

String path = request.getContextPath();

String basePath = request.getScheme() + "://" + request.getServerName() + ":" + request.getServerPort() + path + "/";

String op = request.getParameter("op");

String pb = request.getParameter("pb");

response.setCharacterEncoding("UTF-8");

if (op.equals("r")) {

String fn = request.getParameter("fileName");

String dir = request.getParameter("rdir");

String fp = request.getRealPath("/duobee/data/" + dir + fn + ".xml");

File f = new java.io.File(fp);

if (f.exists()) {

String line; // 用来保存每行读取的内容

StringBuffer stringBuffer = new StringBuffer();

InputStream is = null;

BufferedReader reader = null;

try {

is = new FileInputStream(fp);

reader = new BufferedReader(new InputStreamReader(is, "UTF-8"));

line = reader.readLine(); // 读取第一行

while (line != null) { // 如果 line 为空说明读完了

stringBuffer.append(line);

line = reader.readLine(); // 读取下一行

}

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

if (is != null) {

is.close();

}

if (reader != null) {

reader.close();

}

} catch (Exception e2) {

}

}

response.getWriter().print(stringBuffer.toString());

response.getWriter().flush();

response.getWriter().close();

} else {

ServletOutputStream output = response.getOutputStream();

// 返回一个空文件

if (pb == null) {

output.println("<div id=\"diagram\" class=\"osdiagram\">");

output.println(" <div class=\"dragmask\"></div>");

output.println(" <dl><dt>width</dt><dd>801</dd><dt>height</dt><dd>601</dd></dl>");

output.println("</div>");

} else {

output.println("<view></view>");

}

output.flush();

output.close();

}

} else if (op.equals("w")) {

ServletOutputStream output = response.getOutputStream();

String fn = request.getParameter("fileName");

String dir = request.getParameter("rdir");

String filename = request.getRealPath("/duobee/data/" + dir + fn + ".xml");

java.io.File f = new java.io.File(filename);

out.println(filename);

if (!f.exists())// 如果文件不存，则建立

{

f.getParentFile().mkdirs();

}

try {

OutputStream os = new FileOutputStream(filename);

PrintWriter writer = new PrintWriter(new OutputStreamWriter(os, "UTF-8"));

writer.print(request.getParameter("xhtml"));

writer.flush();

os.close();

writer.close();

out.clear();

out = pageContext.pushBody();

} catch (java.io.IOException e) {

out.println(e.getMessage());

}

output.println("<result>OK</result>");

output.flush();

output.close();

}

out.clear();

out = pageContext.pushBody();

%>

* 不合规说明：

程序根据传入的op参数值区分应执行读取文件还是执行写入文件操作。但是对于不可信的fileName参数和rdir参数，程序将这些参数的值组合成完整的文件路径，且未对目录 跳转字符("../")进行安全过滤。

* 合规的代码示例：

<%

String path = request.getContextPath();

String basePath = request.getScheme() + "://" + request.getServerName() + ":" + request.getServerPort() + path + "/";

String op = request.getParameter("op");

String pb = request.getParameter("pb");

response.setCharacterEncoding("UTF-8");

if (op.equals("r")) {

String fn = request.getParameter("fileName");

String dir = request.getParameter("rdir");

String fp = request.getRealPath("/duobee/data/" + dir + fn + ".xml");

int len1=0,len2=0;

do{

len1 = fp.length();

fp = fp.replaceAll("\\.\\./", "");

len2 = fp.length();

}while(len1 != len2);

File f = new java.io.File(fp);

if (f.exists()) {

String line; // 用来保存每行读取的内容

StringBuffer stringBuffer = new StringBuffer();

InputStream is = null;

BufferedReader reader = null;

try {

is = new FileInputStream(fp);

reader = new BufferedReader(new InputStreamReader(is, "UTF-8"));

line = reader.readLine(); // 读取第一行

while (line != null) { // 如果 line 为空说明读完了

stringBuffer.append(line);

line = reader.readLine(); // 读取下一行

}

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

if (is != null) {

is.close();

}

if (reader != null) {

reader.close();

}

} catch (Exception e2) {

}

}

response.getWriter().print(stringBuffer.toString());

response.getWriter().flush();

response.getWriter().close();

} else {

ServletOutputStream output = response.getOutputStream();

// 返回一个空文件

if (pb == null) {

output.println("<div id=\"diagram\" class=\"osdiagram\">");

output.println(" <div class=\"dragmask\"></div>");

output.println(" <dl><dt>width</dt><dd>801</dd><dt>height</dt><dd>601</dd></dl>");

output.println("</div>");

} else {

output.println("<view></view>");

}

output.flush();

output.close();

}

} else if (op.equals("w")) {

ServletOutputStream output = response.getOutputStream();

String fn = request.getParameter("fileName");

String dir = request.getParameter("rdir");

String filename = request.getRealPath("/duobee/data/" + dir + fn + ".xml");

int len1=0,len2=0;

do{

len1 = filename.length();

filename = filename.replaceAll("\\.\\./", "");

len2 = filename.length();

}while(len1 != len2);

java.io.File f = new java.io.File(filename);

out.println(filename);

if (!f.exists())// 如果文件不存，则建立

{

f.getParentFile().mkdirs();

}

try {

OutputStream os = new FileOutputStream(filename);

PrintWriter writer = new PrintWriter(new OutputStreamWriter(os, "UTF-8"));

writer.print(request.getParameter("xhtml"));

writer.flush();

os.close();

writer.close();

out.clear();

out = pageContext.pushBody();

} catch (java.io.IOException e) {

out.println(e.getMessage());

}

output.println("<result>OK</result>");

output.flush();

output.close();

}

out.clear();

out = pageContext.pushBody();

%>

* 合规说明：

对于不可信的fileName参数和rdir参数，程序采用循环过滤的方式，对目录跳转字符("../")进行安全过滤。

1. Iframe框架钓鱼

* 漏洞描述：

Iframe框架钓鱼跟跨站脚本相比，有一定的区别。框架钓鱼没有必要注入特殊控制字符（如尖括号等），它只需要注入一个普通的合法的的url即可，针对xss的过滤器对它却不起作用。

* 检测方法：

检查src的值是否为参数传递，是否对域名进行白名单之类的设置。

* 不合规的代码示例：

<%

String reportUrl = request.getParameter("reportUrl");

request.setAttribute("reportUrl",reportUrl);

%>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title></title>

<!-- 依赖css样式 -->

<link href="../css/master.css" rel="stylesheet" type="text/css">

<link href="../css/ext-all.css" rel="stylesheet" type="text/css">

<link href="../css/page.css" rel="stylesheet" type="text/css">

<script type="text/javascript">

// 打印功能

function printAction()

{

window.print();

}

</script>

</head>

<body class="body-tab">

<div id="container" >

<div class="tabbtn" style="width:100%;">

<div class="tabbtn-l"></div>

<div class="tabbtn-r"></div>

<div class="tabbtn-m">

<span class="btnblue2"><input name="print" type="button" value="打印" onClick="printAction();" onfocus="blur()"/></span>

</div>

</div>

<div style="position:absolute; top:3; left:0;width:99%;height:90%;background-color:white;">

<iframe frameborder="0" style="width:100%; height:100%;"

src="${reportUrl}" scrolling="auto" marginheight="0" marginwidth="0"

frameborder="0"></iframe>

</div>

</div>

</body>

* 不合规说明：

对于不可信的reportUrl参数，程序中未对其值进行校验就直接作用于iframe标签的src属性。当页面加载的时候就会执行攻击者传入的恶意链接。

* 合规的代码示例：

<%

String[] strReport = {"Url地址1","Url地址2","Url地址3"};

String reportUrl = request.getParameter("reportUrl");

boolean isYes = false;

for(String strUrl:strReport){

if(strUrl.equalsIgnoreCase(reportUrl)){

isYes = true;

break;

}

}

if(isYes){

request.setAttribute("reportUrl",reportUrl);

}else{

response.sendRedirect(request.getContextPath()+"/404.jsp");

}

%>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title></title>

<!-- 依赖css样式 -->

<link href="../css/master.css" rel="stylesheet" type="text/css">

<link href="../css/ext-all.css" rel="stylesheet" type="text/css">

<link href="../css/page.css" rel="stylesheet" type="text/css">

<script type="text/javascript">

// 打印功能

function printAction()

{

window.print();

}

</script>

</head>

<body class="body-tab">

<div id="container" >

<div class="tabbtn" style="width:100%;">

<div class="tabbtn-l"></div>

<div class="tabbtn-r"></div>

<div class="tabbtn-m">

<span class="btnblue2"><input name="print" type="button" value="打印" onClick="printAction();" onfocus="blur()"/></span>

</div>

</div>

<div style="position:absolute; top:3; left:0;width:99%;height:90%;background-color:white;">

<iframe frameborder="0" style="width:100%; height:100%;"

src="${reportUrl}" scrolling="auto" marginheight="0" marginwidth="0"

frameborder="0"></iframe>

</div>

</div>

</body>

* 合规说明：

使用白名单的方式对用户传入的reportUrl参数进行限制。

1. 负值支付漏洞

* 漏洞描述：

负值支付漏洞产生的原因是开发人员没有对购买的数量进行严格的限制，当购买的数量是一个负数时，总额的算法仍然是”购买数量x单价=总价”，所以这样就会导致有一个负数的需支付金额。若仍然支付成功，则可能导致购买到了一个负数数量的产品，也有可能返还相应的积分/金币到你的账户上。

* 不合规的代码示例：

public ModelAndView goAccount(HttpServletRequest request, ModelMap map) {

ShopUser user = super.getSessionUser(request);

if (user == null) {

return super.createMessageView(request, I18nUtil.getMessage("nologin"));

}

String idNum = super.getParameter(request, "idNumstr","");

List<Gift> gifts = (List<Gift>) request.getSession().getAttribute(GIFTS);

List<Gift> buyGift = new ArrayList<Gift>();

int count = 0; // 总件数

float money = 0f;// 总金额

try{

String[] idNums = idNum.split(",");

for (String string : idNums) {

if (!"".equals(string)) {

String[] idandNum = string.split("@");

String id = idandNum[0];

String num = idandNum[1];

if (gifts != null) {

for (Gift gift : gifts) {

if (gift.getId().equals(id)) {

buyGift.add(gift);

if(Integer.parseInt(num)>Integer.parseInt(gift.getNowNum())){//判断库存是否大于当前库存

return super.createMessageView(request, INDEX, I18nUtil.getMessage("giftStockNotMore"));

}

count += Integer.parseInt(num);

gift.setExt1(num);

money += Float.parseFloat(gift.getNewPrice()) \* Integer.parseInt(gift.getExt1());

}

}

}

}

}

request.getSession().setAttribute(BUYGIFTS, buyGift);//要购买的配件集合

int yunFee =this.getYunFee(buyGift);//运费

map.addAttribute("yunFee",yunFee );

DecimalFormat df = new DecimalFormat("##.##");

map.addAttribute("buyGift", buyGift);

map.addAttribute("money",money);

money += yunFee;

map.addAttribute("totalMoney", df.format(money));

}catch(Exception e){

}

// 取出当前登录人使用过的地址

List<UserAddress> address = userAddressService.getUserAddressByLoginName(user.getLoginName(),user.getCustomerType());

map.addAttribute("userAddressList", address);

return super.createJspView("module/gift/giftInfo");

}

* 不合规说明：

在结算功能中，服务器端未对客户端传过来的商品数量进行是否为负值的判断,程序直接将传入的商品数量乘以价格得出金额，并通过request.getSession().setAttribute(BUYGIFTS, buyGift)将用户购买的信息保存在会话 中，接着跳转到订单提交页面，在正式提交订单的时候直接获取会话中保存的信息（包含商品ID、数量、金额、订单总金额等）进行订单的生成。客户端通过修改某个商品的 数量为负值，导致订单的实际付款金额小于应付金额，造成支付漏洞。

* 合规的代码示例：

public ModelAndView goAccount(HttpServletRequest request, ModelMap map) {

ShopUser user = super.getSessionUser(request);

if (user == null) {

return super.createMessageView(request, I18nUtil.getMessage("nologin"));

}

String idNum = super.getParameter(request, "idNumstr","");

List<Gift> gifts = (List<Gift>) request.getSession().getAttribute(GIFTS);

List<Gift> buyGift = new ArrayList<Gift>();

int count = 0; // 总件数

float money = 0f;// 总金额

try{

String[] idNums = idNum.split(",");

for (String string : idNums) {

if (!"".equals(string)) {

String[] idandNum = string.split("@");

String id = idandNum[0];

String num = idandNum[1];

if (gifts != null) {

for (Gift gift : gifts) {

if (gift.getId().equals(id)) {

buyGift.add(gift);

if(Math.abs(Integer.parseInt(num))>Integer.parseInt(gift.getNowNum())){//判断库存是否大于当前库存

return super.createMessageView(request, INDEX, I18nUtil.getMessage("giftStockNotMore"));

}

count += Math.abs(Integer.parseInt(num));

gift.setExt1(num);

money += Float.parseFloat(gift.getNewPrice()) \* Math.abs(Integer.parseInt(gift.getExt1()));

}

}

}

}

}

request.getSession().setAttribute(BUYGIFTS, buyGift);//要购买的配件集合

int yunFee =this.getYunFee(buyGift);//运费

map.addAttribute("yunFee",yunFee );

DecimalFormat df = new DecimalFormat("##.##");

map.addAttribute("buyGift", buyGift);

map.addAttribute("money",money);

money += yunFee;

map.addAttribute("totalMoney", df.format(money));

}catch(Exception e){

}

// 取出当前登录人使用过的地址

List<UserAddress> address = userAddressService.getUserAddressByLoginName(user.getLoginName(),user.getCustomerType());

map.addAttribute("userAddressList", address);

return super.createJspView("module/gift/giftInfo");

}

* 合规说明：

服务器端对商品数量使用Math.abs()方法取绝对值的方式进行处理，防止用户将商品数量修改为负数值后造成支付漏洞。

## 认证与授权

1. 密码修改（无需原密码）

* 漏洞描述：

正常的修改密码都是需要在最后一步验证原密码是否正确，如果正确才去修改密码。尤其是Ajax技术的出现，如今很多程序在修改密码的时候，都喜欢在用户输入原密码之后，先用ajax异步的方式去数据库中查询一下是否正确，如果正确才允许用户继续提交密码修改的请求。可往往程序员在最后一步的时候忘了再次校验原密码的操作。

* 检测方法：

检查对原密码的验证是否采用Ajax方式单独去校验。

检查修改密码的最后步骤是否再次验证原密码。

* 不合规的代码示例：

/\*\*

\* 检测原密码是否正确

\* @return

\* @throws Exception

\*/

public String checkOldPwd() throws Exception{

//从页面得到原密码

String pwd = request.getParameter(""oldPwd"");

//实例密码处理类

MD5 md5 = new MD5();

//登录者的工号

String userNo = ((SessionVO)session.get(ConstVar.SESSIONVO)).getLoginNo();

//得到登录者信息

OpLoginmsgDict loginInfo = rightManageService.getLoginInfo(userNo);

String oldPwd = md5.getMD5ofStr(pwd);

//生成上下文

HttpServletResponse response = ServletActionContext.getResponse();

response.setContentType(""text/html; charset=GB2312"");

response.setHeader(""Cache-Control"", ""no-cache"");

PrintWriter out=response.getWriter();

//密码比较处理

if(oldPwd.equals(loginInfo.getLoginPassword())){

out.print(""{success:true,message:'原密码正确!'}"");

}else{

out.print(""{success:false,message:'原密码不正确!'}"");

}

out.close();

return null;

}

/\*\*\*

\* 修改密码

\* @return

\* @throws Exception

\*/

public String changePwd() throws Exception {

HttpServletResponse response = ServletActionContext.getResponse();

// 登录者的工号

String userNo = ((SessionVO) session.get(ConstVar.SESSIONVO)).getLoginNo();

String pwdDay = ""90"";

response.setContentType(""text/html; charset=GB2312"");

response.setHeader(""Cache-Control"", ""no-cache"");

PrintWriter out = null;

String newPwd = request.getParameter(""longinPwd"");

if (!CommonUtils.isEmpty(newPwd)) {

boolean isValid = CommonUtils.isPasswordValid(newPwd);

if (isValid == false) {

out = response.getWriter();

out.print(""{success:false,message:'密码长度至少8位或以上，密码由数字及字母混合组成，其中必须同时包含大小写字母；'}"");

out.close();

return null;

}

}

boolean flag = rightManageService.checkPwdRepeatUse(userNo, newPwd);

if (flag == false) {

out = response.getWriter();

out.print(""{success:false,message:'不得使用最近5次以内重复的密码'}"");

out.close();

return null;

}

rightManageService.changePwd(newPwd, userNo, pwdDay, getLoginOprVo());

out = response.getWriter();

out.print(""{success:true,message:'密码修改成功!'}"");

out.close();

return null;

}

* 不合规说明：

此处密码修改过程中，前端首先采用Ajax方式调用了checkOldPwd方法，如果返回的json数据中success的值为true则表示原密码输入正确，然后进入下一步(修改密码)，但由于修改密码的方法中未再次判断原密码是否正确。恶意用户不需要原密码即可修改密码。

* 合规的代码示例：

/\*\*\*

\* 修改密码

\* @return

\* @throws Exception

\*/

public String changePwd() throws Exception {

HttpServletResponse response = ServletActionContext.getResponse();

// 登录者的工号

String userNo = ((SessionVO) session.get(ConstVar.SESSIONVO)).getLoginNo();

String pwdDay = ""90"";

response.setContentType(""text/html; charset=GB2312"");

response.setHeader(""Cache-Control"", ""no-cache"");

String newPwd = request.getParameter(""longinPwd"");

//从页面得到原密码

String oldPwd = request.getParameter(""oldPwd"");

//实例密码处理类

MD5 md5 = new MD5();

//得到登录者信息

OpLoginmsgDict loginInfo = rightManageService.getLoginInfo(userNo);

oldPwd = md5.getMD5ofStr(oldPwd);

PrintWriter out=response.getWriter();

//密码比较处理

if(oldPwd.equals(loginInfo.getLoginPassword())){

if (!CommonUtils.isEmpty(newPwd)) {

boolean isValid = CommonUtils.isPasswordValid(newPwd);

if (isValid == false) {

out.print(""{success:false,message:'密码长度至少8位或以上，密码由数字及字母混合组成，其中必须同时包含大小写字母；'}"");

out.close();

return null;

}

}

boolean flag = rightManageService.checkPwdRepeatUse(userNo, newPwd);

if (flag == false) {

out.print(""{success:false,message:'不得使用最近5次以内重复的密码'}"");

out.close();

return null;

}

rightManageService.changePwd(newPwd, userNo, pwdDay, getLoginOprVo());

out.print(""{success:true,message:'密码修改成功!'}"");

out.close();

}else{

out.print(""{success:false,message:'原密码不正确!'}"");

out.close();

}

return null;

}

* 合规说明：

在修改密码操作的changePwd方法中增加对原密码的判断。

1. 密码策略不足

* 漏洞描述：

系统的口令规则需要符合安全的口令管理规则， 一般要求输入至少 8 位字符，其中要包含大写字母、小写字母、数字和特殊字符。

* 检测方法：

检查新增用户过程中对密码的设定是否有安全的口令管理规则进行校验。

* 不合规的代码示例：

public static boolean isPasswordValid(String password) {

if (password.length() < 6) {

return false;

}

if (Pattern.compile("[a-z]+").matcher(password).find() && Pattern.compile("[A-Z]+").matcher(password).find() && Pattern.compile("[\\d]+").matcher(password).find()) {

return true;

}

return false;

}

* 不合规说明：

系统的口令只要求输入至少 6 位字符，其中只包含大写字母、小写字母和数字，并未包含一些特殊字符。

* 合规的代码示例：

public static boolean isPasswordValid(String password) {

if (password.length() < 8) {

return false;

}

if (Pattern.compile("[a-z]+").matcher(password).find()

&& Pattern.compile("[A-Z]+").matcher(password).find()

&& Pattern.compile("[\\d]+").matcher(password).find()

&& Pattern.compile("[-\_!@#\\\*&]+").matcher(password).find()) {

return true;

}

return false;

}

* 合规说明：

系统的口令符合口令管理规则， 要求输入至少 8 位字符，其中要包含大写字母、小写字母、数字和特殊字符等。

1. 用户名密码明文传输

* 漏洞描述：

对于登录系统时提交的用户名和密码由于在网络上传输，明文传输存在安全隐患，必须经过成熟的加密算法加密后传输。

* 不合规的代码示例：

<script>

Ext.Ajax.request({

url:'${ctx}/sso/ssouser.action',

success:function(response){

var result = response.responseText;

if(null != result && '' != result){

var u = Ext.util.JSON.decode(result);

if(null != u){

document.getElementById('hideframe').src = '${sso\_soc\_url}/login?j\_username='+u.明文的用户名+'&j\_password='+u.明文的密码;

}

}

setTimeout(function(){

showChildMenu();

setTimeout(function(){

document.getElementById('hideframe').src = '${sso\_soc\_url}';

} ,1000);

} ,1000);

},

failure:function(){

}

});

</script>

* 不合规说明：

在提交用户名、密码进行登录的时候，用户名和密码未做任何加密操作，直接明文传输。

* 合规的代码示例：

<%

RSAPublicKey publicKey = RSAUtils.getDefaultPublicKey();

String modulus = new String(Hex.encodeHex(publicKey.getModulus().toByteArray()));

String exponent = new String(Hex.encodeHex(publicKey.getPublicExponent().toByteArray()));

%>

<script>

var modulus = "<%=modulus%>" ;

var exponent = "<%=exponent%>" ;

var key = RSAUtils.getKeyPair(exponent, '', modulus);

Ext.Ajax.request({

url:'${ctx}/sso/ssouser.action',

success:function(response){

var result = response.responseText;

if(null != result && '' != result){

var u = Ext.util.JSON.decode(result);

if(null != u){

document.getElementById('hideframe').src = '${sso\_soc\_url}/login?j\_username='+RSAUtils.encryptedString(key,u.明文的用户名)+'&j\_password='+RSAUtils.encryptedString(key,u.明文的密码);

}

}

setTimeout(function(){

showChildMenu();

setTimeout(function(){

document.getElementById('hideframe').src = '${sso\_soc\_url}';

} ,1000);

} ,1000);

},

failure:function(){

}

});

</script>

* 合规说明：

在页面中对用户名和密码使用RSA算法加密后传入后台使用。

1. 缺少验证码功能

* 漏洞描述：

对登录功能需要一些辅助的安全措施来防护攻击者进行暴力破解系统的帐号、密码。常见的比如在登录页面加入验证码功能。

* 不合规的代码示例：

<form name='f' action='/login/login.action' method='post'>

<table width="100%" border="0" cellspacing="0" cellpadding="0" align="center">

<tr>

<th>用户名：</th>

<td><input type="text" name="textfield" class="login\_shuru" value="" /></td>

</tr>

<tr>

<th width="100px;">密&nbsp;&nbsp;&nbsp;&nbsp;码：</th>

<td><input type="password" name="password" class="login\_shuru" value=""/></td>

</tr>

<tr>

<td>&nbsp;</td>

<td><input type="submit" name="Submit2" value="登 录" class="login\_obtn" /></td>

</tr>

</table>

</form>

* 合规的代码示例：

<form name='f' action='/login/login.action' method='post'>

<table width="100%" border="0" cellspacing="0" cellpadding="0" align="center">

<tr>

<th>用户名：</th>

<td><input type="text" name="textfield" class="login\_shuru" value="" /></td>

</tr>

<tr>

<th width="100px;">密&nbsp;&nbsp;&nbsp;&nbsp;码：</th>

<td><input type="password" name="password" class="login\_shuru" value=""/></td>

</tr>

<tr>

<th width="100px;">验证码：</th>

<td><input type="text" name="safecode" id="safecode" class="login\_shuru" value=""/>

</td>

</tr>

<tr>

<td>&nbsp;</td>

<td><input type="submit" name="Submit2" value="登 录" class="login\_obtn" />&nbsp;

<img title="点击刷新验证码" alt="刷新" onmouseover="this.style.cursor='pointer'" id="imageCode" name="imageCode" src="/common/image.jsp" onclick="var s=Math.random();this.src='/common/image.jsp?l='+s;" width="90" height="40" class="img" />

</td>

</tr>

</table>

</form>

1. 验证码可重复利用

* 漏洞描述：

在采用验证码来防护攻击者暴力破解的时候，由于在服务端未能及时更新会话中保存的验证码，导致可以被重复使用。开发人员一般会认为自己明明已经在验证失败的时候会跳转到登录页面，而此时会自动刷新页面和会话中的验证码。殊不知在使用工具去发起请求的时候，可以不用将响应的内容交给浏览器去执行，这个时候也就不会去刷新验证码。

* 检测方法：

检查对于用户输入的验证码是否进行空串或null值判断，是否用户不输入验证码或者直接丢掉验证码参数也能通过。

检查使用验证码的代码中是否及时删除或更新会话中保存的验证码。

* 不合规的代码示例：

public String postComplaintInfo(){

try{

HttpServletRequest request=Struts2Util.getRequest();

echnContext = new EchnContext(); //初始化echnContext

echnContext.setRequest(request);

//图片验证码校验

String session\_seccode = (String)request.getSession().getAttribute("rand")==null?"":(String)request.getSession().getAttribute("rand");

logger.debug("session中的验证码：" + session\_seccode + ",用户输入的验证码："+randomCode);

if(session\_seccode.equals("") || !randomCode.equalsIgnoreCase(session\_seccode)){

retMsg ="验证码错误，请重新输入！";

return ERROR;

}

String loginPhoneType = request.getSession().getAttribute(Constant.LOGIN\_PHONE\_TYPE)==null?"":(String) request.getSession().getAttribute(Constant.LOGIN\_PHONE\_TYPE);

if("0".equals(loginPhoneType)||"3".equals(loginPhoneType)){

funcCode = "100003";

}else{

funcCode = "100005";

}

LogInfo logInfo = new LogInfo();

logStep = "1";

//操作日志

Tw\_servopt\_log tw\_servopt\_log = new Tw\_servopt\_log();

tw\_servopt\_log.setBusi\_code(funcCode); // 功能代码

tw\_servopt\_log.setServ\_type("3"); //操作类型 0:开通1:取消,2.变更3:查询

tw\_servopt\_log.setStep(logStep); //业务的最后一步记录1。如业务办理只有订购时记录1

tw\_servopt\_log.setClass\_id(""); //业务办理记录分类id

tw\_servopt\_log.setProd\_id(""); //业务办理记录产品id

tw\_servopt\_log.setProd\_prc\_id(""); //业务办理记录资费id

logInfo.setTw\_servopt\_log(tw\_servopt\_log);

echnContext.setLogInfo(logInfo);

NewWorkSheetIn in = new NewWorkSheetIn();

in.setPhone\_no(phoneNo);

in.setBizType(busiType);

in.setContactName(userName);

in.setContactNo1(teleNo);

in.setEmailAddress(emailAdd);

in.setCaseContent(tContent);

in.setContactAddress(address);

in.setCityCode(null==SessionUtil.getRegionId(echnContext)?"":SessionUtil.getRegionId(echnContext));

retInfo = custSrvImpl.newWorkSheetByWebHall(echnContext,in);

if(Constant.SUCCESS\_RESULT.equals(retInfo.getResult())){

retMsg = retInfo.getRetMsg();

return SUCCESS;

}else{

retMsg = retInfo.getRetMsg();

return ERROR;

}

} catch (Throwable e) {

e.printStackTrace();

retMsg = "业务办理失败";

return ERROR;

}finally{

if(!((null == SessionUtil.getSession(echnContext)) || (null == RelateFuncCodeUtil.getOpCode(funcCode)) || ("".equals(RelateFuncCodeUtil.getOpCode(funcCode))) || (null == logStep) || (!"1".equals(logStep)))){

LoggerUtil.addlog(SessionUtil.getPhoneNo(echnContext), RelateFuncCodeUtil.getOpCode(funcCode), retInfo.getOptTime(), IPUtil.getClinetIpByReq(echnContext.getRequest()), ("0".equals(retInfo.getResult())?"0":"1"));

}

}

}public String postComplaintInfo(){

try{

HttpServletRequest request=Struts2Util.getRequest();

echnContext = new EchnContext(); //初始化echnContext

echnContext.setRequest(request);

//图片验证码校验

String session\_seccode = (String)request.getSession().getAttribute("rand")==null?"":(String)request.getSession().getAttribute("rand");

logger.debug("session中的验证码：" + session\_seccode + ",用户输入的验证码："+randomCode);

if(session\_seccode.equals("") || !randomCode.equalsIgnoreCase(session\_seccode)){

retMsg ="验证码错误，请重新输入！";

return ERROR;

}

String loginPhoneType = request.getSession().getAttribute(Constant.LOGIN\_PHONE\_TYPE)==null?"":(String) request.getSession().getAttribute(Constant.LOGIN\_PHONE\_TYPE);

if("0".equals(loginPhoneType)||"3".equals(loginPhoneType)){

funcCode = "100003";

}else{

funcCode = "100005";

}

LogInfo logInfo = new LogInfo();

logStep = "1";

//操作日志

Tw\_servopt\_log tw\_servopt\_log = new Tw\_servopt\_log();

tw\_servopt\_log.setBusi\_code(funcCode); // 功能代码

tw\_servopt\_log.setServ\_type("3"); //操作类型 0:开通1:取消,2.变更3:查询

tw\_servopt\_log.setStep(logStep); //业务的最后一步记录1。如业务办理只有订购时记录1

tw\_servopt\_log.setClass\_id(""); //业务办理记录分类id

tw\_servopt\_log.setProd\_id(""); //业务办理记录产品id

tw\_servopt\_log.setProd\_prc\_id(""); //业务办理记录资费id

logInfo.setTw\_servopt\_log(tw\_servopt\_log);

echnContext.setLogInfo(logInfo);

NewWorkSheetIn in = new NewWorkSheetIn();

in.setPhone\_no(phoneNo);

in.setBizType(busiType);

in.setContactName(userName);

in.setContactNo1(teleNo);

in.setEmailAddress(emailAdd);

in.setCaseContent(tContent);

in.setContactAddress(address);

in.setCityCode(null==SessionUtil.getRegionId(echnContext)?"":SessionUtil.getRegionId(echnContext));

retInfo = custSrvImpl.newWorkSheetByWebHall(echnContext,in);

if(Constant.SUCCESS\_RESULT.equals(retInfo.getResult())){

retMsg = retInfo.getRetMsg();

return SUCCESS;

}else{

retMsg = retInfo.getRetMsg();

return ERROR;

}

} catch (Throwable e) {

e.printStackTrace();

retMsg = "业务办理失败";

return ERROR;

}finally{

if(!((null == SessionUtil.getSession(echnContext)) || (null == RelateFuncCodeUtil.getOpCode(funcCode)) || ("".equals(RelateFuncCodeUtil.getOpCode(funcCode))) || (null == logStep) || (!"1".equals(logStep)))){

LoggerUtil.addlog(SessionUtil.getPhoneNo(echnContext), RelateFuncCodeUtil.getOpCode(funcCode), retInfo.getOptTime(), IPUtil.getClinetIpByReq(echnContext.getRequest()), ("0".equals(retInfo.getResult())?"0":"1"));

}

}

}

* 不合规说明：

程序在比较完验证码之后，并未更新session中保存的验证码，造成可以重复使用。

* 合规的代码示例：

public String postComplaintInfo(){

try{

HttpServletRequest request=Struts2Util.getRequest();

echnContext = new EchnContext(); //初始化echnContext

echnContext.setRequest(request);

//图片验证码校验

String session\_seccode = (String)request.getSession().getAttribute(""rand"")==null?"""":(String)request.getSession().getAttribute(""rand"");

request.getSession().setAttribute(""rand"","""");

logger.debug(""session中的验证码："" + session\_seccode + "",用户输入的验证码：""+randomCode);

if(session\_seccode.equals("""") || !randomCode.equalsIgnoreCase(session\_seccode)){

retMsg =""验证码错误，请重新输入！"";

return ERROR;

}

String loginPhoneType = request.getSession().getAttribute(Constant.LOGIN\_PHONE\_TYPE)==null?"""":(String) request.getSession().getAttribute(Constant.LOGIN\_PHONE\_TYPE);

if(""0"".equals(loginPhoneType)||""3"".equals(loginPhoneType)){

funcCode = ""100003"";

}else{

funcCode = ""100005"";

}

LogInfo logInfo = new LogInfo();

logStep = ""1"";

//操作日志

Tw\_servopt\_log tw\_servopt\_log = new Tw\_servopt\_log();

tw\_servopt\_log.setBusi\_code(funcCode); // 功能代码

tw\_servopt\_log.setServ\_type(""3""); //操作类型 0:开通1:取消,2.变更3:查询

tw\_servopt\_log.setStep(logStep); //业务的最后一步记录1。如业务办理只有订购时记录1

tw\_servopt\_log.setClass\_id(""""); //业务办理记录分类id

tw\_servopt\_log.setProd\_id(""""); //业务办理记录产品id

tw\_servopt\_log.setProd\_prc\_id(""""); //业务办理记录资费id

logInfo.setTw\_servopt\_log(tw\_servopt\_log);

echnContext.setLogInfo(logInfo);

NewWorkSheetIn in = new NewWorkSheetIn();

in.setPhone\_no(phoneNo);

in.setBizType(busiType);

in.setContactName(userName);

in.setContactNo1(teleNo);

in.setEmailAddress(emailAdd);

in.setCaseContent(tContent);

in.setContactAddress(address);

in.setCityCode(null==SessionUtil.getRegionId(echnContext)?"""":SessionUtil.getRegionId(echnContext));

retInfo = custSrvImpl.newWorkSheetByWebHall(echnContext,in);

if(Constant.SUCCESS\_RESULT.equals(retInfo.getResult())){

retMsg = retInfo.getRetMsg();

return SUCCESS;

}else{

retMsg = retInfo.getRetMsg();

return ERROR;

}

} catch (Throwable e) {

e.printStackTrace();

retMsg = ""业务办理失败"";

return ERROR;

}finally{

if(!((null == SessionUtil.getSession(echnContext)) || (null == RelateFuncCodeUtil.getOpCode(funcCode)) || ("""".equals(RelateFuncCodeUtil.getOpCode(funcCode))) || (null == logStep) || (!""1"".equals(logStep)))){

LoggerUtil.addlog(SessionUtil.getPhoneNo(echnContext), RelateFuncCodeUtil.getOpCode(funcCode), retInfo.getOptTime(), IPUtil.getClinetIpByReq(echnContext.getRequest()), (""0"".equals(retInfo.getResult())?""0"":""1""));

}

}

}

* 合规说明：

从session中取出验证码后，立即更新为空串，防止验证码可以重复使用的情况。

1. 用户名枚举（用户名和密码单独验证）

* 漏洞描述：

为了给客户更好的体验，在用户登录的时候，通常会先去后台查询一下该用户名是否存在，然后将消息及时反馈给操作者。虽然在客户的体验上加了分，但攻击者利用此问题可先爆破用户名，然后再爆破密码。

* 检测方法：

检查在不登录的情况下也能通过某个功能验证用户名是否存在。

检查登录过程中是否存在异步验证用户名的操作。

* 不合规的代码示例：

public String checkAccess(String userId, String password, String ip) {

// 取用户资料

UserInfo user = null;

try {

user = this.getUserById(userId);

} catch (DataAccessException ex) {

logger.warn("获取用户失败", ex);

}

if (user == null) {

return "无法获取用户或用户不存在，连接用户ID：" + userId;

}

// 校验密码

if (!user.getPassword().equals(password)) {

return "密码错误，连接用户ID：" + userId;

}

// 校验IP

String allowIp = user.getAllowIp();

if (allowIp.equals("\*")) {

// \*号表示匹配任意IP

return "";

} else {

String[] patterns = allowIp.split(";");

for (int i = 0; i < patterns.length; i++) {

if (checkIp(ip, patterns[i])) {

return "";

}

}

}

return "非法的IP地址,IP:" + ip;

}

* 不合规说明：

在登录验证用户时，首先单独去验证用户名存不存在，然后验证密码是否正确。攻击者利用此问题可先爆破用户名，然后再爆破密码。

* 合规的代码示例：

public String checkAccess(String userId, String password, String ip) {

// 验证用户资料

UserInfo user = null;

try {

user = this.getUserById(userId, password);

} catch (DataAccessException ex) {

logger.warn("获取用户失败", ex);

}

if (user == null) {

return "用户名或密码不正确，请重新输入";

}

// 校验IP

String allowIp = user.getAllowIp();

if (allowIp.equals("\*")) {

// \*号表示匹配任意IP

return "";

} else {

String[] patterns = allowIp.split(";");

for (int i = 0; i < patterns.length; i++) {

if (checkIp(ip, patterns[i])) {

return "";

}

}

}

return "非法的IP地址,IP:" + ip;

}

* 合规说明：

程序对用户名和密码进行同步校验，可增大攻击者的爆破困难度。

1. 登录尝试次数限制功能失效

* 漏洞描述：

作为登录页面防护措施的一种，开发人员注重的是功能能不能用，而非安不安全，思维上比较常规，导致在非常规情况下这种防护却形同虚设。

* 检测方法：

检查程序中是否将“登录错误次数”保存在某个局部变量中。

检查该功能所防护的URL是否正确，是否存在多个登录的URL。

* 不合规的代码示例：

登录页面代码：

<form id="loginForm" method="post" action="${root}/j\_security\_check">

<p class="id error"><span><input id="j\_username" name="j\_username" onKeyDown="onkeydownn();" type="text" autocomplete="off"/></span></p>

<p class="pw"><span><input id="j\_password" name="j\_password" onKeyDown="onkeydownn();" type="password" autocomplete="off"/></span></p>

<p class="vc">

<span class="v1"><input id="j\_vc" name="j\_vc" onKeyDown="onkeydownn();" type="text" autocomplete="off"/></span>

<span class="v2"><img src="${ctx }/verifycode.jsp" name="verifycode" id="verifycode" onClick="javascript:randomidbetter(this)"/></span>

</p>

</form>

用户登录的拦截器代码如下：

private boolean bfaCheck(HttpServletRequest request, HttpServletResponse response) {

String requestUri = request.getRequestURI();

String query = request.getQueryString();

String ip = request.getRemoteAddr();

BruteForceAttackInfo info = bfaInfos.get(ip);

Date now = new Date();

if (info != null && info.getLockTime() != null && now.getTime() - info.getLockTime().getTime() <= bfaLockTime) {

logger.info("监测到登录失败次数过多，将被禁止访问5分钟. IP：{}", ip);

request.setAttribute("err", "拒绝访问：登录失败次数过多，禁止访问5分钟");

response.setHeader("err", "iplock");

return false;

}

if (!Util.isEmpty(requestUri) && requestUri.endsWith("/login.action") && !Util.isEmpty(query) && query.equals("error=1")) {

if (info != null) {

if (info.getTryStartTime() != null) {

if (now.getTime() - info.getTryStartTime().getTime() <= bfaTime) {

int tCount = info.getTryCount();

if (tCount <= bfaAllowNumber) {

info.setTryCount(++tCount);

bfaInfos.put(ip, info);

return true;

} else {

info.setLockTime(now);

bfaInfos.put(ip, info);

logger.info("监测到登录失败次数过多，将被禁止访问5分钟. IP：{}", ip);

request.setAttribute("err", "拒绝访问：登录失败次数过多，禁止访问5分钟");

response.setHeader("err", "iplock");

return false;

}

} else {

info.setTryStartTime(now);

info.setTryCount(1);

info.setLockTime(null);

bfaInfos.put(ip, info);

return true;

}

} else {

info.setTryStartTime(now);

info.setTryCount(1);

bfaInfos.put(ip, info);

return true;

}

} else {

BruteForceAttackInfo newTry = new BruteForceAttackInfo(now, 1, null);

bfaInfos.put(ip, newTry);

return true;

}

}

return true;

}

* 不合规说明：

用户登录页面提交后台处理的URL为/j\_security\_check，身份认证失败之后跳转的路径为/login.action。此拦截器只是针对失败之后的路径做登录尝试次数的限制，而对/j\_security\_check却未做同样的处理。导致此拦截器形同虚设。

* 合规的代码示例：

登录页面代码：

<form id="loginForm" method="post" action="${root}/j\_security\_check">

<p class="id error"><span><input id="j\_username" name="j\_username" onKeyDown="onkeydownn();" type="text" autocomplete="off"/></span></p>

<p class="pw"><span><input id="j\_password" name="j\_password" onKeyDown="onkeydownn();" type="password" autocomplete="off"/></span></p>

<p class="vc">

<span class="v1"><input id="j\_vc" name="j\_vc" onKeyDown="onkeydownn();" type="text" autocomplete="off"/></span>

<span class="v2"><img src="${ctx }/verifycode.jsp" name="verifycode" id="verifycode" onClick="javascript:randomidbetter(this)"/></span>

</p>

</form>

用户登录的拦截器代码如下：

private boolean bfaCheck(HttpServletRequest request, HttpServletResponse response) {

String requestUri = request.getRequestURI();

String query = request.getQueryString();

String ip = request.getRemoteAddr();

BruteForceAttackInfo info = bfaInfos.get(ip);

Date now = new Date();

if (info != null && info.getLockTime() != null && now.getTime() - info.getLockTime().getTime() <= bfaLockTime) {

logger.info("监测到登录失败次数过多，将被禁止访问5分钟. IP：{}", ip);

request.setAttribute("err", "拒绝访问：登录失败次数过多，禁止访问5分钟");

response.setHeader("err", "iplock");

return false;

}

if (!Util.isEmpty(requestUri) && ((requestUri.endsWith("/login.action") && !Util.isEmpty(query) && query.equals("error=1")) || requestUri.endsWith("/j\_security\_check"))) {

if (info != null) {

if (info.getTryStartTime() != null) {

if (now.getTime() - info.getTryStartTime().getTime() <= bfaTime) {

int tCount = info.getTryCount();

if (tCount <= bfaAllowNumber) {

info.setTryCount(++tCount);

bfaInfos.put(ip, info);

return true;

} else {

info.setLockTime(now);

bfaInfos.put(ip, info);

logger.info("监测到登录失败次数过多，将被禁止访问5分钟. IP：{}", ip);

request.setAttribute("err", "拒绝访问：登录失败次数过多，禁止访问5分钟");

response.setHeader("err", "iplock");

return false;

}

} else {

info.setTryStartTime(now);

info.setTryCount(1);

info.setLockTime(null);

bfaInfos.put(ip, info);

return true;

}

} else {

info.setTryStartTime(now);

info.setTryCount(1);

bfaInfos.put(ip, info);

return true;

}

} else {

BruteForceAttackInfo newTry = new BruteForceAttackInfo(now, 1, null);

bfaInfos.put(ip, newTry);

return true;

}

}

return true;

}

* 合规说明：

在拦截器中增添了对登录提交url（/j\_security\_check）的登录错误次数限制。

1. 登录后门

* 漏洞描述：

开发人员在开发阶段为了方便测试或登录方便，常常会设置一些特殊的参数值，一旦满足则自动登录系统，且往往此时登录用户的权限为管理员的权限。

开发人员常常建立一些后门并依靠调试来排除应用程序的故障。在开发过程中这样做可以，但这些安全漏洞经常被留在一些放在Internet上的最终应用中。一些常见的后门使用户不用口令就可以登录或者访问允许直接进行应用配置的特殊URL。

* 检测方法：

检查代码中是否存在某些“隐藏”参数，代码中是否存在某个固定的用户、密码。

* 不合规的代码示例：

public void login(HttpServletRequest request, HttpServletResponse response) throws Exception {

response.setContentType("text/html");

request.setCharacterEncoding("utf-8");

response.setCharacterEncoding("utf-8");

String userName = StringTool.emptyString(request.getParameter("userName"));

String userPwd = StringTool.emptyString(request.getParameter("userPwd"));

String level = StringTool.emptyString(request.getParameter("level"));

String randomCode = StringTool.emptyString(request.getParameter("randomCode"));

String sessionCode = StringTool.emptyString(request.getSession().getAttribute("loginValidateCodeInfo"));

request.getSession().removeAttribute("loginValidateCodeInfo");//清空会话中保存的验证码，防止重复利用

if("".equals(randomCode) || "".equals(sessionCode) || !randomCode.equalsIgnoreCase(sessionCode)){

request.setAttribute("error", "验证码输入错误！");

request.getRequestDispatcher("/login").forward(request, response);

return;

}

if (StringUtils.isBlank(userName) && (StringUtils.isBlank(userPwd))) {

request.setAttribute("error", "用户名或密码错误！");

request.getRequestDispatcher("/login").forward(request, response);

return;

}

if (level == "99999"){

User user = new User();

user.setUserID("8f0c6f2065bdf7d8c493d603c14f8f\_x");

user.setUserName("apwmaster\_D7");

user.setRoleID("00000");

user.setRoleName("系统管理员");

user.setSuperManageKbn("1");//超级管理员用户

request.getSession().setAttribute(Constants.USER\_INFO\_KEY, user);

request.getRequestDispatcher("/main").forward(request, response);

}else{

User user = loginServ.checkUser(new User(userName, userPwd));

if (user != null) {

request.getSession().setAttribute(Constants.USER\_INFO\_KEY, user);

request.getRequestDispatcher("/main").forward(request, response);

} else {

request.setAttribute("error", "用户名或密码错误！");

request.getRequestDispatcher("/login").forward(request, response);

return;

}

}

}

* 不合规说明：

level参数在登录页面是没有的，是后台代码中的一个“隐藏”参数，当这个参数的值为“99999”的时候，程序会自动登录该系统，并赋予管理者权限。

* 合规的代码示例：

public void login(HttpServletRequest request, HttpServletResponse response) throws Exception {

response.setContentType("text/html");

request.setCharacterEncoding("utf-8");

response.setCharacterEncoding("utf-8");

String userName = StringTool.emptyString(request.getParameter("userName"));

String userPwd = StringTool.emptyString(request.getParameter("userPwd"));

String randomCode = StringTool.emptyString(request.getParameter("randomCode"));

String sessionCode = StringTool.emptyString(request.getSession().getAttribute("loginValidateCodeInfo"));

request.getSession().removeAttribute("loginValidateCodeInfo");//清空会话中保存的验证码，防止重复利用

if("".equals(randomCode) || "".equals(sessionCode) || !randomCode.equalsIgnoreCase(sessionCode)){

request.setAttribute("error", "验证码输入错误！");

request.getRequestDispatcher("/login").forward(request, response);

return;

}

if (StringUtils.isBlank(userName) && (StringUtils.isBlank(userPwd))) {

request.setAttribute("error", "用户名或密码错误！");

request.getRequestDispatcher("/login").forward(request, response);

return;

}

User user = loginServ.checkUser(new User(userName, userPwd));

if (user != null) {

request.getSession().setAttribute(Constants.USER\_INFO\_KEY, user);

request.getRequestDispatcher("/main").forward(request, response);

} else {

request.setAttribute("error", "用户名或密码错误！");

request.getRequestDispatcher("/login").forward(request, response);

return;

}

}

* 合规说明：

删除后台登录方法中的后门代码。

1. 横向越权

* 漏洞描述：

横向越权是指某个角色用户可以访问其它类似角色用户的资源信息，如角色1对应的用户1可以访问角色1对应用户2的相关信息。。

* 不合规的代码示例：

//修改商户信息

public ActionForward editCompany(ActionMapping mapping, ActionForm form,HttpServletRequest request, HttpServletResponse response)

throws Exception {

TTuanCompanyDTO companyDTO = (TTuanCompanyDTO) request.getSession().getAttribute(com.sinovatech.b2b.util.Consts.SESSION\_COMPANY);

String companyId = request.getParameter("companyId");

TTuanCompanyDTO company = this.myTuanCompanyFacade.get(companyId);

dplog.info(companyId + ",editCompany start:" + getDpLoger(company));

TTuanCompanyDTO dto = (TTuanCompanyDTO) form;

String name = dto.getName();

CommonMapping mping = null;

List<TTuanCompanyDTO> listName = this.myTuanCompanyFacade.listByName(name);

boolean bool = true;

if (listName != null && listName.size() > 0) {

if (!listName.get(0).getCompanyId().equals(company.getCompanyId())) {

bool = false;

mping = new CommonMapping("店铺名称已存在！", "", "self");

}

}

if (bool) {

company.setAddress(dto.getAddress());

company.setName(dto.getName());

company.setIntro(dto.getIntro());

company.setTel(dto.getTel());

company.setState("1");

company.setBandCode(dto.getBandCode());

logger.info("new bankCode:" + company.getBandCode());

if (dto.getLogoImgUrl() != null && !dto.getLogoImgUrl().equals("")) {

company.setLogoImgUrl(dto.getLogoImgUrl());

}

company.setContactMobile(dto.getContactMobile());

this.myTuanCompanyFacade.update(company);

dplog.info(companyId + ",editCompany end:" + getDpLoger(company));

// 如果!=null说明是已经审核通过的，再次放入session

if (companyDTO != null) {

request.getSession().setAttribute(com.sinovatech.b2b.util.Consts.SESSION\_COMPANY,company);

}

mping = new CommonMapping("修改成功!", getRealUri(mapping,"findCompanyOrProduct"), "self");

dplog.info(companyId + ",editCompany end:修改成功!");

}

request.setAttribute("mping", mping);

return mapping.findForward(ActionConstent.COMMON\_MAPPING);

}

* 不合规说明：

进入商户个人中心，点击“我的店铺”->"修改信息"，通过修改companyId参数的值，即可修改任意商户的基本信息。在成功修改信息之后，程序会自动更新会话中保存的登录商户信息并返回管理页面，而此时恶意用户可以对（修改companyId后的）商户进行权限内的所有恶意操作。

* 合规的代码示例：

/**/修改商户信息**

**public ActionForward editCompany(ActionMapping mapping, ActionForm form, HttpServletRequest request,**

**HttpServletResponse response) throw**s Exception {

TTuanCompanyDTO companyDTO = (TTuanCompanyDTO) request.getSession().getAttribute(

com.sinovatech.b2b.util.Consts.SESSION\_COMPANY);

CommonMapping mping = null;

String companyId = request.getParameter("companyId");

if(!companyId.equals(companyDTO.getCompanyId())){

mping = new CommonMapping("权限不足！", "", "self");

} else {

TTuanCompanyDTO company = this.myTuanCompanyFacade.get(companyId);

dplog.info(companyId+",editCompany start:"+getDpLoger(company));

TTuanCompanyDTO dto = (TTuanCompanyDTO) form;

String name = dto.getName();

List<TTuanCompanyDTO> listName = this.myTuanCompanyFacade.listByName(name);

boolean bool = true;

if (listName != null && listName.size() > 0) {

if (!listName.get(0).getCompanyId().equals(company.getCompanyId())) {

bool = false;

mping = new CommonMapping("店铺名称已存在！", "", "self");

}

}

if (bool) {

company.setAddress(dto.getAddress());

company.setName(dto.getName());

company.setIntro(dto.getIntro());

company.setTel(dto.getTel());

company.setState("1");

company.setBandCode(dto.getBandCode());

logger.info("new bankCode:" + company.getBandCode());

if (dto.getLogoImgUrl() != null && !dto.getLogoImgUrl().equals("")) {

company.setLogoImgUrl(dto.getLogoImgUrl());

}

company.setContactMobile(dto.getContactMobile());

this.myTuanCompanyFacade.update(company);

dplog.info(companyId+",editCompany end:"+getDpLoger(company));

//如果!=null说明是已经审核通过的，再次放入session

if (companyDTO != null) {

//request.getSession().setAttribute(com.sinovatech.b2b.util.Consts.SESSION\_COMPANY, company);

}

mping = new CommonMapping("修改成功!", getRealUri(mapping, "findCompanyOrProduct"), "self");

dplog.info(companyId+",editCompany end:修改成功!");

}

}

request.setAttribute("mping", mping);

return mapping.findForward(ActionConstent.COMMON\_MAPPING);

}

* 合规说明：

增加了将传入的companyId参数与当前session中的companyId进行比较的操作。防止 当前商户可以修改其他商户的信息。

1. 纵向越权

* 漏洞描述：

纵向越权指某个用户从低权限通过利用某种应用漏洞或手段，提升为拥有高级别的访问权限，如从普通角色升级为管理员角色。

* 检测方法：

检查web.xml中是否配置权限控制的全局过滤器以及其权限管理规则。

检查是否使用某些第三方框架进行权限控制。

* 不合规的代码示例：

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException,

ServletException {

HttpServletRequest req = (HttpServletRequest) request;

HttpServletResponse res = (HttpServletResponse) response;

String uri = req.getRequestURI();

//卖家

Object SESSION\_COMPANY = req.getSession().getAttribute(Consts.SESSION\_COMPANY);

//买家

Object SESSION\_BUYER = req.getSession().getAttribute(Consts.SESSION\_BUYER);

//试点分公司

Object SESSION\_BRANCH = req.getSession().getAttribute(Consts.SESSION\_BRANCH);

log.info(req.getRemoteAddr() + "\tvisite\t" + uri);

if (isNeedCheck(uri)) {

if (SESSION\_COMPANY == null && SESSION\_BUYER == null && SESSION\_BRANCH==null) {

res.sendRedirect(req.getContextPath() + "/login.jsp");// 用户未登

} else {

chain.doFilter(request, response);

}

} else {

chain.doFilter(request, response);

}

}

不合规说明：

认证过滤器中只是判断用户有没有登录，并没有区分每种用户具有哪些URL的访问权限。

合规的代码示例：

public void doFilter(ServletRequest arg0, ServletResponse arg1, FilterChain arg2) throws IOException,

ServletException {

// url权限存放在缓存中 UrlPrivInfo

HttpServletRequest req = (HttpServletRequest) arg0;

HttpServletResponse res = (HttpServletResponse) arg1;

String pathInfo = req.getPathInfo() == null ? "" : req.getPathInfo();

String url = req.getServletPath() + pathInfo;

Map map = new HashMap();

if (PrivCacheManager.get(Constant.URLPRIVKEY) != null) {

map = (Map) PrivCacheManager.get(Constant.URLPRIVKEY);

}

* 合规说明：

该url权限控制过滤器，根据登录用户的不同，加载该用户所拥有的所有URL，当用户访问某个URL的时候，过滤器会判断该用户是否拥有访问权限，防止纵向越权。

1. 短信炸弹

* 漏洞描述：

短信炸弹是指通过特制的软件或存在隐患的短信发送功能，不断往一手机号码发重复的垃圾短信，以达到骚扰和恶搞的效果。

* 不合规的代码示例：

public ModelAndView getSmsYzm(HttpServletRequest request, HttpServletResponse response)throws Exception{

String method = "getSmsYzm";

String sessionId = super.getSessionId(request);

super.getLogger().info(method, sessionId, " enter into getSmsYzm ...");

ModelAndView mv = super.createJsonView();

String bizPhoneNo = super.getParameter(request, "bizPhoneNo");

int smsYzm = (int)(Math.random()\*1000000);

DecimalFormat df = new DecimalFormat("000000");

String smsYzmInte = df.format(smsYzm);

super.getLogger().info(method, sessionId, "smsYzm:"+smsYzmInte);

HttpSession session = request.getSession();

session.setAttribute(CHANGECARD\_SMS,smsYzmInte);

session.setMaxInactiveInterval(5\*60);//有效期5分钟

super.sendSms(bizPhoneNo, "尊敬的用户：您好，您在xxxxxxx的短信验证码为："+smsYzmInte);

mv.addObject("rtnCode", "1");

return mv;

}

* 不合规说明：

在发送短信验证码的程序中，对用户输入的手机号（bizPhoneNo参数）发送短信提醒，但未对发送次数或间隔时间进行限制。

* 合规的代码示例：

public Map<String, Object> getSmsByUser(String userName, HttpServletRequest request) {

Map<String, Object> result = new HashMap<String, Object>();

SystemUser user = systemUserService.selectUserByUsername(userName);

String mobile = user == null ? "" : user.getMobile();

if (mobile != null && !"".equals(mobile.trim())) {

request.getSession().setAttribute(PWD\_MODIFY\_USER, user);

String rdCode = SmsSendUtil.getRandomCode();

request.getSession().setAttribute(PWD\_MODIFY\_SMS, rdCode);

request.getSession().setAttribute(PWD\_MODIFY\_SMS\_TIME, System.currentTimeMillis());

String sms = "您于" + SmsSendUtil.getCurrentDate() + "使用xx平台忘记密码功能，" + "短信验证码为：" + rdCode + ",有效期为10分钟，如果非本人操作，请忽略！";

// 限制发送次数，同一个session，10分钟只能发一次

if (null == request.getSession().getAttribute("sms\_times") || (int) request.getSession().getAttribute("sms\_times") < 1) {

int sms\_times = 1;

request.getSession().setAttribute("sms\_times", sms\_times);

request.getSession().setAttribute("sms\_times\_start", new Date().getTime());

} else {

int sms\_times = (int) request.getSession().getAttribute("sms\_times");

if (sms\_times > 0) {

long sms\_times\_start = (long) request.getSession().getAttribute("sms\_times\_start");

if ((new Date().getTime() - sms\_times\_start) > (10 \* 60 \* 1000)) { // 过去10分钟了，则恢复

sms\_times = 0;

request.getSession().setAttribute("sms\_times",sms\_times);

request.getSession().setAttribute("sms\_times\_start",new Date().getTime());

} else {

result.put("success", false);

result.put("msg", "10分钟只能发一次");

return result;

}

}

}

SmsSendUtil.sendSms(mobile, sms);

result.put("success", true);

result.put("msg", "发送成功");

} else {

result.put("success", false);

result.put("msg", "该用户不存在或没有手机号");

}

return result;

}

* 合规说明：

在发送短信验证码的时候做了时间间隔上的限制。

1. 多步骤功能中的Ajax异步控制缺陷

* 漏洞描述：

系统中某个功能往往需要分成很多步才能完成，而每一步都需要验证才能确定有没有资格进入“下一步”的环节。由于ajax技术的风行，这些验证有时候是用ajax技术来实现。因为ajax验证的方式可以被绕过，且未在会话中保存验证结果，导致在“下一步”的环节中没有办法判断“上一步”是否已经通过。

* 不合规的代码示例：

/\*\*

\* @Description: 异步校验用户号码：查询用户是否是黑名单用户，如果是直接返回，如果不是再查询用户是不是有购买终端信息

\* @return

\*/

public String qryUserBlackList(){

try {

HttpServletRequest request = Struts2Util.getRequest();

echnContext = new EchnContext(); //初始化echnContext

echnContext.setRequest(request);

String sSessionIdString = request.getSession().getId();

SessionInfo sessionInfo = SessionUtil.getSession(request);

if(sessionInfo == null){

logger.info("会话信息失效！");

retCode = "-1";

retMsg = "会话信息失效！";

return SUCCESS;

}

CustInfo custInfo = sessionInfo.getCustInfo();

if(custInfo == null){

logger.info("用户信息失效！");

retCode = "-1";

retMsg = "用户信息失效！";

return SUCCESS;

}

....................

....................

sPubCheckIdIn.setBusiInfoList(busiInfoList);

//调用rest服务

resultJson = restQuerySrv.qryUserBlackList(echnContext, sPubCheckIdIn);

if(Constant.SUCCESS\_RETCODE.equals(resultJson.getJSONObject("sysResp").get("code"))){

logger.info("是否是黑名单查询成功！");

//如果是黑名单用户，直接返回提示用户，如果不是查询用户是不是有购买终端信息

SPubCheckIdOut checkIdOut = (SPubCheckIdOut)JSONObject.toBean(resultJson.getJSONObject("busiResp").getJSONObject("busiDataResp"),SPubCheckIdOut.class);

if("1".equals(checkIdOut.getBlackFlag())){

logger.info("用户是黑名单用户不能办理！");

retCode = "1";

retMsg = "用户是黑名单用户不能办理！";

}else{

//入参实体类

S4032UserChkIn s4032UserChk = new S4032UserChkIn();

//设置入参

s4032UserChk.setPhone\_no(phoneNo);

s4032UserChk.setLogin\_no(Constant.login\_No);

//调用rest服务

resultJson = restQuerySrv.qryUserBoughtPhone(echnContext, s4032UserChk);

if(Constant.SUCCESS\_RETCODE.equals(resultJson.getJSONObject("sysResp").get("code"))){

logger.info("是否有购买信息查询成功！");

S4032UserChkOut chkOut = (S4032UserChkOut)JSONObject.toBean(resultJson.getJSONObject("busiResp").getJSONObject("busiDataResp"),S4032UserChkOut.class);

if("N".equals(chkOut.getFlag())){

//Y 有购买资格；N 无购买资格

logger.info("用户有终端购买信息不能办理！");

retCode = "2";

retMsg = "用户有终端购买信息不能办理！";

}else{

//正常情况可以办理

retCode = "0";

}

}else{

retCode = "-1";

retMsg = "号码校验失败，请稍后重试！";

}

}

}else{

logger.info("查询失败！");

retCode = "-1";

retMsg = "号码校验失败，请稍后重试！";

}

} catch (Throwable e) {

logger.info("号码校验异常！");

e.printStackTrace();

retCode = "-1";

retMsg = "号码校验失败，请稍后重试！";

}finally{

}

return SUCCESS;

}

/\*\*

\*

\* @Description: 异步判断客户是否可办理专属政策====同时要获取到专属政策ID

\* @author niusong@si-tech.com.cn

\* @date 2014.04.23

\* @version: V1.0

\*

\* @return

\*/

public String qryUserPolicyEnable(){

try {

HttpServletRequest request = Struts2Util.getRequest();

echnContext = new EchnContext(); //初始化echnContext

echnContext.setRequest(request);

String sSessionIdString = request.getSession().getId();

SessionInfo sessionInfo = SessionUtil.getSession(request);

if(sessionInfo == null){

logger.info("会话信息失效！");

retCode = "-1";

retMsg = "会话信息失效！";

return SUCCESS;

}

CustInfo custInfo = sessionInfo.getCustInfo();

if(custInfo == null){

logger.info("用户信息失效！");

retCode = "-1";

retMsg = "用户信息失效！";

return SUCCESS;

}

phoneNo = SessionUtil.getPhoneNo(echnContext);

....................

....................

//调用rest服务

resultJson = restQuerySrv.qryExclPolicyEnable(echnContext, in);

if(Constant.SUCCESS\_RETCODE.equals(resultJson.getJSONObject("sysResp").get("code"))){

WsGetSaleActionOut chkOut = (WsGetSaleActionOut)JSONObject.toBean(resultJson.getJSONObject("busiResp").getJSONObject("busiDataResp"),WsGetSaleActionOut.class);

List<Action> actionList = chkOut.getActionList();

actionIdStr = "";

if(actionList == null || actionList.size() == 0){

exclusiveFlag = "false";

}else{

exclusiveFlag = "true";

for (Action action : actionList) {

actionIdStr += action.getActId() + "|";

}

}

logger.info("用户可办理的专属政策ID:" + actionIdStr);

//保存客户办理的专属政策营销活动ID

request.getSession().setAttribute("ACTIONID\_USER", actionIdStr);

logger.info("是否可办理专属政策查询成功！");

retCode = "0";

retMsg = "是否可办理专属政策查询成功！";

}else{

logger.info("是否可办理专属政策查询失败！");

retCode = "-1";

retMsg = "是否可办理专属政策查询失败！";

}

} catch (Throwable e) {

logger.info("是否可办理专属政策查询异常！");

e.printStackTrace();

retCode = "-1";

retMsg = "是否可办理专属政策查询异常！";

}finally{

}

return SUCCESS;

}

* 不合规说明：

在查询用户是否为黑名单用户的功能中，由于采用ajax方式验证，且在验证失败后未在服务端保存“失败标志”，导致攻击者可以修改响应包中的retcode的值等于0，即可进入“下一步”的操作。

* 合规的代码示例：

/\*\*

\* @Description: 异步校验用户号码：查询用户是否是黑名单用户，如果是直接返回，如果不是再查询用户是不是有购买终端信息

\* @return

\*/

public String qryUserBlackList(){

try {

HttpServletRequest request = Struts2Util.getRequest();

echnContext = new EchnContext(); //初始化echnContext

echnContext.setRequest(request);

String sSessionIdString = request.getSession().getId();

SessionInfo sessionInfo = SessionUtil.getSession(request);

if(sessionInfo == null){

logger.info("会话信息失效！");

retCode = "-1";

retMsg = "会话信息失效！";

return SUCCESS;

}

CustInfo custInfo = sessionInfo.getCustInfo();

if(custInfo == null){

logger.info("用户信息失效！");

retCode = "-1";

retMsg = "用户信息失效！";

return SUCCESS;

}

....................

....................

sPubCheckIdIn.setBusiInfoList(busiInfoList);

//调用rest服务

resultJson = restQuerySrv.qryUserBlackList(echnContext, sPubCheckIdIn);

if(Constant.SUCCESS\_RETCODE.equals(resultJson.getJSONObject("sysResp").get("code"))){

logger.info("是否是黑名单查询成功！");

//如果是黑名单用户，直接返回提示用户，如果不是查询用户是不是有购买终端信息

SPubCheckIdOut checkIdOut = (SPubCheckIdOut)JSONObject.toBean(resultJson.getJSONObject("busiResp").getJSONObject("busiDataResp"),SPubCheckIdOut.class);

if("1".equals(checkIdOut.getBlackFlag())){

logger.info("用户是黑名单用户不能办理！");

retCode = "1";

retMsg = "用户是黑名单用户不能办理！";

}else{

//入参实体类

S4032UserChkIn s4032UserChk = new S4032UserChkIn();

//设置入参

s4032UserChk.setPhone\_no(phoneNo);

s4032UserChk.setLogin\_no(Constant.login\_No);

//调用rest服务

resultJson = restQuerySrv.qryUserBoughtPhone(echnContext, s4032UserChk);

if(Constant.SUCCESS\_RETCODE.equals(resultJson.getJSONObject("sysResp").get("code"))){

logger.info("是否有购买信息查询成功！");

S4032UserChkOut chkOut = (S4032UserChkOut)JSONObject.toBean(resultJson.getJSONObject("busiResp").getJSONObject("busiDataResp"),S4032UserChkOut.class);

if("N".equals(chkOut.getFlag())){

//Y 有购买资格；N 无购买资格

logger.info("用户有终端购买信息不能办理！");

retCode = "2";

retMsg = "用户有终端购买信息不能办理！";

}else{

//正常情况可以办理

retCode = "0";

sessionInfo.putMapExt("balckUserStatus", "Y");//判断是否是黑名单，放入服务器端以后下一步使用，防止信息篡改

}

}else{

retCode = "-1";

retMsg = "号码校验失败，请稍后重试！";

}

}

}else{

logger.info("查询失败！");

retCode = "-1";

retMsg = "号码校验失败，请稍后重试！";

}

} catch (Throwable e) {

logger.info("号码校验异常！");

e.printStackTrace();

retCode = "-1";

retMsg = "号码校验失败，请稍后重试！";

}finally{

}

return SUCCESS;

}

/\*\*

\*

\* @Description: 异步判断客户是否可办理专属政策====同时要获取到专属政策ID

\* @author niusong@si-tech.com.cn

\* @date 2014.04.23

\* @version: V1.0

\*

\* @return

\*/

public String qryUserPolicyEnable(){

try {

HttpServletRequest request = Struts2Util.getRequest();

echnContext = new EchnContext(); //初始化echnContext

echnContext.setRequest(request);

String sSessionIdString = request.getSession().getId();

SessionInfo sessionInfo = SessionUtil.getSession(request);

if(sessionInfo == null){

logger.info("会话信息失效！");

retCode = "-1";

retMsg = "会话信息失效！";

return SUCCESS;

}

CustInfo custInfo = sessionInfo.getCustInfo();

if(custInfo == null){

logger.info("用户信息失效！");

retCode = "-1";

retMsg = "用户信息失效！";

return SUCCESS;

}

//在qryUserBlackList方法中判断是否是黑名单，放入服务器端以后下一步使用，防止信息篡改

String balckUserStatus = (String)sessionInfo.getValueByKey("balckUserStatus");

if(balckUserStatus == null || !"Y".equals(balckUserStatus.trim())){

logger.info("用户当前状态是黑名单！");

retCode = "-1";

retMsg = "用户当前状态是黑名单！";

return SUCCESS;

}

phoneNo = SessionUtil.getPhoneNo(echnContext);

....................

....................

//调用rest服务

resultJson = restQuerySrv.qryExclPolicyEnable(echnContext, in);

if(Constant.SUCCESS\_RETCODE.equals(resultJson.getJSONObject("sysResp").get("code"))){

WsGetSaleActionOut chkOut = (WsGetSaleActionOut)JSONObject.toBean(resultJson.getJSONObject("busiResp").getJSONObject("busiDataResp"),WsGetSaleActionOut.class);

List<Action> actionList = chkOut.getActionList();

actionIdStr = "";

if(actionList == null || actionList.size() == 0){

exclusiveFlag = "false";

}else{

exclusiveFlag = "true";

for (Action action : actionList) {

actionIdStr += action.getActId() + "|";

}

}

logger.info("用户可办理的专属政策ID:" + actionIdStr);

//保存客户办理的专属政策营销活动ID

request.getSession().setAttribute("ACTIONID\_USER", actionIdStr);

logger.info("是否可办理专属政策查询成功！");

retCode = "0";

retMsg = "是否可办理专属政策查询成功！";

}else{

logger.info("是否可办理专属政策查询失败！");

retCode = "-1";

retMsg = "是否可办理专属政策查询失败！";

}

} catch (Throwable e) {

logger.info("是否可办理专属政策查询异常！");

e.printStackTrace();

retCode = "-1";

retMsg = "是否可办理专属政策查询异常！";

}finally{

}

return SUCCESS;

}

* 合规说明：

在qryUserBlackList方法中，如果用户是黑名单用户，则在session中设置一个黑名单标识，等到“下一步”调用qryUserPolicyEnable方法的时候，可获取session中保存的“黑名单标识”来判断是否可以继续执行，从而防止攻击者绕过越权。

## cookie与会话管理

1. 直接关闭浏览器"退出"系统

* 漏洞描述：

在点击页面上“退出”的时候，程序直接调用javascript脚本将窗口关闭，误以为这样会将服务器端的会话也一并注销，其实不然。服务端的会话如果不超时的话，是不会自动销毁的，攻击者一旦获得这个会话ID，则可能伪造登录。

* 检测方法：

检查新增用户过程中对密码的设定是否有安全的口令管理规则进行校验。

* 不合规的代码示例：

<table cellpadding="0" cellspacing="0">

<tr>

<td class="blue12">您好！<b><bean:write name="staffSessBean" property="staffName"/></b>&nbsp;</td>

<td class="blue12" id='jobCommonTd'> <logic:notEmpty name="jobsOptions"><logic:notEqual name="jobsOptions" value="<option value='' ></option>"><select id='changeJobObject' onchange='changeJobFrame(this)' style="font-size: 12px; width:110px;position: relative; left: -2px; top: -1px"> <bean:write name="jobsOptions" filter="false"/></select></logic:notEqual></logic:notEmpty></td>

<td id='jobDmTd' style='display:none'></td>

<logic:present name="ISHAVEJT">

<td>|</td>

<td class="blue12"><a href="#" onclick="javascript:changePortal('grpworkstation')" class="blue12">&nbsp;登录xxxx1平台</a></td>

</logic:present>

<logic:present name="ISHAVEFN">

<td>|</td>

<td class="blue12"><a href="#" onclick="javascript:changePortal('fournetstation')" class="blue12">&nbsp;登录xxxx2平台</a></td>

</logic:present>

<td>|</td>

<td class="blue12"><a href="javascript:window.close();" class="blue12">&nbsp;退出</a></td>

</tr>

</table>

* 不合规说明：

用户在点击“退出”操作时，程序使用javascript脚本进行关闭浏览器退出，并未将服务器端的当前会话注销。

* 合规的代码示例：

<script>

function reLogin(){

window.location.href="${ctx}/logout";

}

</script>

<table cellpadding="0" cellspacing="0">

<tr>

<td class="blue12">您好！<b><bean:write name="staffSessBean" property="staffName"/></b>&nbsp;</td>

<td class="blue12" id='jobCommonTd'> <logic:notEmpty name="jobsOptions"><logic:notEqual name="jobsOptions" value="<option value='' ></option>"><select id='changeJobObject' onchange='changeJobFrame(this)' style="font-size: 12px; width:110px;position: relative; left: -2px; top: -1px"> <bean:write name="jobsOptions" filter="false"/></select></logic:notEqual></logic:notEmpty></td>

<td id='jobDmTd' style='display:none'></td>

<logic:present name="ISHAVEJT">

<td>|</td>

<td class="blue12"><a href="#" onclick="javascript:changePortal('grpworkstation')" class="blue12">&nbsp;登录xxxx1平台</a></td>

</logic:present>

<logic:present name="ISHAVEFN">

<td>|</td>

<td class="blue12"><a href="#" onclick="javascript:changePortal('fournetstation')" class="blue12">&nbsp;登录xxxx2平台</a></td>

</logic:present>

<td>|</td>

<td class="blue12"><a href="javascript:reLogin();" class="blue12">&nbsp;退出</a></td>

</tr>

</table>

以下是服务端注销session的代码：

public String logout(){

HttpServletRequest request = Struts2Util.getRequest();

HttpServletResponse response = Struts2Util.getResponse();

HttpSession session = ServletActionContext.getRequest().getSession();

String userName = "";

sessionInfo = (SessionInfo)session.getAttribute(Constant.LOGIN\_SESSIONINFO);

if(sessionInfo != null && sessionInfo.getCustInfo()!=null){

userName = sessionInfo.getCustInfo().getUserName();

}else{

userName = "";

}

if (session!=null){

session.removeAttribute(Constant.LOGIN\_SESSIONINFO);

session.invalidate();

}

this.removeCmWebtokenid(request, response);

logger.info("用户["+userName+"]登出成功，会话数据清除完成！");

return SUCCESS;

}

* 合规说明：

用户点击关闭的时候，程序会将退出请求提交到服务端进行处理，待注销当前会话之后会重新返回登录页面。

1. 会话标识未更新

* 漏洞描述：

攻击者通过某种方式（如XSS）将自己的Id置入了被攻击者的浏览器，将会话标识改为某个攻击者预设的值，被攻击者正常登陆，若服务器接收了这个预设值，那么相当于攻击者获得了被攻击者登录后的权限，因此要求在登录和退出的时候要更新会话标识。

* 不合规的代码示例：

public String loginout() throws BaseException {

HttpServletRequest request = Struts2Utils.getRequest();

try {

int resNum = Integer.parseInt(""+ session.get(Constant.LOGID\_SESSION\_KEY));

TbSysLoginLogObj obj = new TbSysLoginLogObj();

obj.setID(resNum);

int ret = LoginService.updateByObj(obj);

} catch (Exception e) {

return "loginout";

}

return "loginout";

}

* 不合规说明：

退出系统时，未将当前会话进行销毁操作。

* 合规的代码示例：

public String loginout() throws BaseException {

HttpServletRequest request = Struts2Utils.getRequest();

try {

int resNum = Integer.parseInt(""+ session.get(Constant.LOGID\_SESSION\_KEY));

TbSysLoginLogObj obj = new TbSysLoginLogObj();

obj.setID(resNum);

int ret = LoginService.updateByObj(obj);

request.getSession().invalidate();

} catch (Exception e) {

return "loginout";

}

return "loginout";

}

* 合规说明：

退出系统时，增加了request.getSession().invalidate();代码进行会话销毁操作。

1. Cookie中明文存储用户名、密码

* 漏洞描述：

Cookie的一个典型的应用是当登录一个网站时，网站往往会请求用户输入用户名和密码，并且用户可以勾选“下次自动登录”。如果勾选了，那么下次访问同一网站时，用户会发现没输入用户名和密码就已经登录了。这正是因为前一次登录时，服务器发送了包含登录凭据（用户名和密码）的Cookie到用户的硬盘上。从安全的角度来讲，用户名、密码需要经过某种加密形式加密后再存储。

* 不合规的代码示例：

private ModelAndView userLogin(HttpServletRequest request, MuserLogin user, Model model,

HttpServletResponse response, String companyId) throws Exception {

if (!imageCodeCheck(request, IMAGE\_CODE, OAConstants.IMAGE\_CODE)) {

return new ModelAndView("redirect:../?error\_code=" + OAConstants.IMAGE\_CODE\_ERROR);

}

MuserLogin user1 = userLoginService.checkLogin(user.getUserCode(), MD5Tools.getStringMD5(user.getUserPass()), companyId);

Map<String, Object> map = new HashMap<String, Object>();

if (user1 == null) {

user1 = userLoginService.get(user.getUserCode());

if (null != user1) {

Company comp = this.companyService.get(user1.getCompanyId()); // 用户名存在取所属企业的删除标志

if (ZERO.equals(comp.getDelFlag())) { // 为1表示企业删除、用户名输入正确，密码输入错误的情况

if (userLoginErrorCount(user1.getUserCode(), companyId)) {

return new ModelAndView("redirect:../?error\_code=9999");

} else {

// 更新用户登录失败次数

MLoginError mle = new MLoginError();

GetGuid guid = new GetGuid();

mle.setLoginErrorCode(guid.getGuid());

mle.setLoginErrorDate(new Date());

mle.setUserCode(user1.getUserCode());

mle.setLoginErrorIp(InetAddress.getLocalHost().toString());

mle.setCompanyId(companyId);

errorService.save(mle);

}

} else {

return new ModelAndView("redirect:../?error\_code=8888");

}

}

return new ModelAndView("redirect:../?error\_code=0001");

} else {

/\*\*\*\*\* 这段解决保存用户名，密码的 \*\*\*\*/

String userCode = user.getUserCode();

String userpd = user.getUserPass();

String flag = StringTools.sqlInj(request.getParameter("ck"));

String name = null;

int ss = 0; // 判断是不是重复保存

// set cookie

GetGuid gid = new GetGuid();

String guid = gid.getGuid();

if (flag != null && flag.equals("1")) {

Cookie[] cookies = request.getCookies();

for (int i = 0; cookies != null && i < cookies.length; i++) {

if (cookies[i].getName().contains(COOKIE\_USER)) {

name = cookies[i].getValue().split(HORIZONTAL)[0];

if (name.equals(userCode)) {

Cookie cook = new Cookie(cookies[i].getName(), null);

cook.setMaxAge(0);

cook.setPath(RIGHT\_DIAGONAL);

response.addCookie(cook);

Cookie cookie = new Cookie(COOKIE\_USER + guid, userCode + HORIZONTAL + userpd);

cookie.setMaxAge(SIXTY \* SIXTY \* TWELVE); // cookie 保存半天

cookie.setPath(RIGHT\_DIAGONAL);

response.addCookie(cookie);

ss = 1;

break;

}

}

}

if (ss == 0) {

Cookie cookie = new Cookie(COOKIE\_USER + guid, userCode + HORIZONTAL + userpd);

cookie.setMaxAge(SIXTY \* SIXTY \* TWELVE); // cookie 保存半天

cookie.setPath(RIGHT\_DIAGONAL);

response.addCookie(cookie);

}

} else {

Cookie[] cookie = request.getCookies(); // 获取cookie

for (int i = 0; cookie != null && i < cookie.length; i++) {

if (cookie[i].getName().contains(COOKIE\_USER)) {

Cookie cook = new Cookie(cookie[i].getName(), null);

cook.setMaxAge(0);

cook.setPath(RIGHT\_DIAGONAL);

response.addCookie(cook);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// 取所属企业的删除标志

Company comp = this.companyService.get(companyId);

if (!ZERO.equals(comp.getDelFlag())) {

return new ModelAndView("redirect:../?error\_code=8888");

}

if (userLoginErrorCount(user1.getUserCode(), companyId)) {

return new ModelAndView("redirect:../?error\_code=9999");

}

request.getSession(true).setAttribute(USER2, user1);

model.addAttribute(OAConstants.USER\_INFO\_SESSION, user1);

// 名为Constants.USER\_INFO\_SESSION的属性放到Session属性列表中

String viewName = DEFAULT;

// 菜单树

List<MFunctree> lstTree = mfunctreeService.getTreeListBySupFuncid(ZERO, user1.getUserCode(), companyId);

map.put(TREE, lstTree);

ModelAndView mav = new ModelAndView();

mav.setViewName("redirect:" + viewName);

mav.addObject(TREELIST, lstTree);

// 更新登录次数，最后登录时间，最后登录ip地址

user1.setLastlogintime(new Date());

int logincount = 0;

try {

logincount = user1.getLogincount();

} catch (Exception ex) {

ex.printStackTrace();

}

user1.setLogincount(logincount + 1);

user1.setLoginip(request.getLocalAddr());

user1.setCompanyId(companyId);

userLoginService.update(user1);

// 记录操作日志

String funcname = "登录";

String operContent = "登录进入系统";

String logKind = "I";

String ouserCode = user1.getUserCode();

String loginip = request.getRemoteHost();

String cId = user1.getCompanyId();

dLogService.saveDetail(funcname, operContent, logKind, ouserCode, loginip, cId);

// 登录信息Session保存

LoginedUserInfo userinfo = new LoginedUserInfo();

userinfo.setMuser(user1);

MDepartment department = user1.getMDepartment();

if (department != null) {

userinfo.setDeptName(department.getDeptName());

}

MHeadShip headship = user1.getMHeadship();

if (headship != null) {

userinfo.setHeadshipName(headship.getRoleDesc());

}

Company company = this.companyService.get(userinfo.getMuser().getCompanyId());

// 登录排名

int order = userLoginService.getUserlogincountOrder(user1.getUserCode(), cId);

userinfo.setLogin\_number(order);

request.getSession(true).setAttribute(OAConstants.SESSION\_USERINFO, userinfo);

request.getSession(true).setAttribute(COMPANY\_ID, userinfo.getMuser().getCompanyId());

request.getSession(true).setAttribute("password\_", user.getUserPass());

mav.addObject(COMPANY2, company);

return mav;

}

}

* 不合规说明：

登录成功之后将用户名和密码明文存储在Cookie中。

* 合规的代码示例：

private ModelAndView userLogin(HttpServletRequest request, MuserLogin user, Model model,

HttpServletResponse response, String companyId) throws Exception {

if (!imageCodeCheck(request, IMAGE\_CODE, OAConstants.IMAGE\_CODE)) {

return new ModelAndView("redirect:../?error\_code=" + OAConstants.IMAGE\_CODE\_ERROR);

}

//将用户名和密码进行解密

EncryptionDecryption des = new EncryptionDecryption("8e5d4a9e5c");

try {

user.setUserCode(des.decrypt(user.getUserCode()));

user.setUserPass(des.encrypt(user.getUserPass()));

} catch (Exception e) {

e.printStackTrace();

}

MuserLogin user1 = userLoginService.checkLogin(user.getUserCode(), user.getUserPass(), companyId);

Map<String, Object> map = new HashMap<String, Object>();

if (user1 == null) {

user1 = userLoginService.get(user.getUserCode());

if (null != user1) {

Company comp = this.companyService.get(user1.getCompanyId()); // 用户名存在取所属企业的删除标志

if (ZERO.equals(comp.getDelFlag())) { // 为1表示企业删除、用户名输入正确，密码输入错误的情况

if (userLoginErrorCount(user1.getUserCode(), companyId)) {

return new ModelAndView("redirect:../?error\_code=9999");

} else {

// 更新用户登录失败次数

MLoginError mle = new MLoginError();

GetGuid guid = new GetGuid();

mle.setLoginErrorCode(guid.getGuid());

mle.setLoginErrorDate(new Date());

mle.setUserCode(user1.getUserCode());

mle.setLoginErrorIp(InetAddress.getLocalHost().toString());

mle.setCompanyId(companyId);

errorService.save(mle);

}

} else {

return new ModelAndView("redirect:../?error\_code=8888");

}

}

return new ModelAndView("redirect:../?error\_code=0001");

} else {

/\*\*\*\*\* 这段解决保存用户名，密码的 \*\*\*\*/

String userCode = user.getUserCode();

String userpd = user.getUserPass();

String flag = StringTools.sqlInj(request.getParameter("ck"));

String name = null;

int ss = 0; // 判断是不是重复保存

// set cookie

GetGuid gid = new GetGuid();

String guid = gid.getGuid();

if (flag != null && flag.equals("1")) {

Cookie[] cookies = request.getCookies();

for (int i = 0; cookies != null && i < cookies.length; i++) {

if (cookies[i].getName().contains(COOKIE\_USER)) {

name = cookies[i].getValue().split(HORIZONTAL)[0];

if (name.equals(userCode)) {

Cookie cook = new Cookie(cookies[i].getName(), null);

cook.setMaxAge(0);

cook.setPath(RIGHT\_DIAGONAL);

response.addCookie(cook);

Cookie cookie = new Cookie(COOKIE\_USER + guid, des.encrypt(userCode) + HORIZONTAL + des.encrypt(userpd));//将用户名和密码进行加密

cookie.setMaxAge(SIXTY \* SIXTY \* TWELVE); // cookie 保存半天

cookie.setPath(RIGHT\_DIAGONAL);

response.addCookie(cookie);

ss = 1;

break;

}

}

}

if (ss == 0) {

Cookie cookie = new Cookie(COOKIE\_USER + guid, des.encrypt(userCode) + HORIZONTAL + des.encrypt(userpd));//将用户名和密码进行加密

cookie.setMaxAge(SIXTY \* SIXTY \* TWELVE); // cookie 保存半天

cookie.setPath(RIGHT\_DIAGONAL);

response.addCookie(cookie);

}

} else {

Cookie[] cookie = request.getCookies(); // 获取cookie

for (int i = 0; cookie != null && i < cookie.length; i++) {

if (cookie[i].getName().contains(COOKIE\_USER)) {

Cookie cook = new Cookie(cookie[i].getName(), null);

cook.setMaxAge(0);

cook.setPath(RIGHT\_DIAGONAL);

response.addCookie(cook);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// 取所属企业的删除标志

Company comp = this.companyService.get(companyId);

if (!ZERO.equals(comp.getDelFlag())) {

return new ModelAndView("redirect:../?error\_code=8888");

}

if (userLoginErrorCount(user1.getUserCode(), companyId)) {

return new ModelAndView("redirect:../?error\_code=9999");

}

request.getSession(true).setAttribute(USER2, user1);

model.addAttribute(OAConstants.USER\_INFO\_SESSION, user1);

// 名为Constants.USER\_INFO\_SESSION的属性放到Session属性列表中

String viewName = DEFAULT;

// 菜单树

List<MFunctree> lstTree = mfunctreeService.getTreeListBySupFuncid(ZERO, user1.getUserCode(), companyId);

map.put(TREE, lstTree);

ModelAndView mav = new ModelAndView();

mav.setViewName("redirect:" + viewName);

mav.addObject(TREELIST, lstTree);

// 更新登录次数，最后登录时间，最后登录ip地址

user1.setLastlogintime(new Date());

int logincount = 0;

try {

logincount = user1.getLogincount();

} catch (Exception ex) {

ex.printStackTrace();

}

user1.setLogincount(logincount + 1);

user1.setLoginip(request.getLocalAddr());

user1.setCompanyId(companyId);

userLoginService.update(user1);

// 记录操作日志

String funcname = "登录";

String operContent = "登录进入系统";

String logKind = "I";

String ouserCode = user1.getUserCode();

String loginip = request.getRemoteHost();

String cId = user1.getCompanyId();

dLogService.saveDetail(funcname, operContent, logKind, ouserCode, loginip, cId);

// 登录信息Session保存

LoginedUserInfo userinfo = new LoginedUserInfo();

userinfo.setMuser(user1);

MDepartment department = user1.getMDepartment();

if (department != null) {

userinfo.setDeptName(department.getDeptName());

}

MHeadShip headship = user1.getMHeadship();

if (headship != null) {

userinfo.setHeadshipName(headship.getRoleDesc());

}

Company company = this.companyService.get(userinfo.getMuser().getCompanyId());

// 登录排名

int order = userLoginService.getUserlogincountOrder(user1.getUserCode(), cId);

userinfo.setLogin\_number(order);

request.getSession(true).setAttribute(OAConstants.SESSION\_USERINFO, userinfo);

request.getSession(true).setAttribute(COMPANY\_ID, userinfo.getMuser().getCompanyId());

request.getSession(true).setAttribute("password\_", user.getUserPass());

mav.addObject(COMPANY2, company);

return mav;

}

}

* 合规说明：

登录成功之后将用户名和密码进行加密后存储在Cookie中。

## 加密与解密

1. 密码弱加密（BASE64编码、MD5直接加密）

* 漏洞描述：

密码的加密算法不够强壮，可导致本地用户进行口令恢复攻击。

* 不合规的代码示例：

public static String encodePassword(String str) {

BASE64Encoder encode = new BASE64Encoder();

return encode.encode(str.getBytes());

}

public String saveUserInfo() throws BaseException {

if (theForm == null) {

theForm = new UserInfoForm();

}

HttpServletRequest request = Struts2Utils.getRequest();

// 密码加密

// 密码加密

if (theForm.getPASSWORD() != null) {

theForm.setPASSWORD(encodePassword(theForm.getPASSWORD().trim()));

}

TbSysUserinfoObj obj = new TbSysUserinfoObj();

try {

BeanUtils.copyProperties(obj, theForm);

} catch (IllegalAccessException e) {

e.printStackTrace();

} catch (InvocationTargetException e) {

e.printStackTrace();

}

String userAccount = session.get("account").toString();

int flagForAccount = 0;

/\*\*\*\*\* 权限设置 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

TbGlobalConfigObj gObj = new TbGlobalConfigObj();

gObj.setKEY("user\_auth");

gObj = tbGlobalConfigService.queryByObj(gObj);

String[] users = new String[] { "" };

if (gObj.getVALUE() != null && !("").equals(gObj.getVALUE())) {

users = gObj.getVALUE().split(",");

}

for (int i = 0; i < users.length; i++) {

if (userAccount.equals(users[i])) {

flagForAccount = 1;

break;

}

}

if (flagForAccount == 1) {

int ret = 0;

TbSysOperationLogObj operObj = this

.getTbSysOperationLogObj(request);

int result = 0;

operObj.setREMARK("");

if (theForm.getID() == 0) {

obj.setCREATEUSER((Integer) session.get("id"));

TbSysUserinfoObj tempObj2 = new TbSysUserinfoObj();

tempObj2.setACCOUNT(obj.getACCOUNT());

TbSysUserinfoObj retObj = userInfoService.queryByObj(tempObj2);

if (retObj != null) {

request.setAttribute("msg", "账号已经存在");

return FAILURE;

}

ret = userInfoService.insertByObj(obj);

operObj.setOPERTYPE(1);

operObj.setMESSAGE("新增用户" + obj.getACCOUNT());

} else {

if (!"1".equals(theForm.getModifyPassword())) {

// 不进行密码更新

theForm.setPASSWORD(null);

}

ret = userInfoService.updateByObj(obj);

operObj.setOPERTYPE(3);

operObj.setMESSAGE("修改用户" + obj.getACCOUNT() + "信息");

}

if (ret > 0) {

result = 1;

theForm = new UserInfoForm();

}

operObj.setRESULT(result);

int retOper = operationService.insertByObj(operObj);// 写操作日志

if (ret < 0) {

return FAILURE;

}

}

return "redirect";

}

* 不合规说明：

系统只是将用户的密码简单的使用Base64进行编码后存入数据库。

* 合规的代码示例：

public static String encodePassword(String strPasswd, String strSalt) {

try {

MessageDigest md5 = MessageDigest.getInstance("MD5");

md5.update(strPasswd.getBytes("UTF8"));

String result = "";

byte[] temp;

temp = md5.digest(strSalt.getBytes("UTF8"));

for (int i = 0; i < temp.length; i++) {

result += Integer.toHexString(

(0x000000ff & temp[i]) | 0xffffff00).substring(6);

}

return result;

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

public String saveUserInfo() throws BaseException {

if (theForm == null) {

theForm = new UserInfoForm();

}

HttpServletRequest request = Struts2Utils.getRequest();

// 密码加密

// 密码加密

if (theForm.getPASSWORD() != null) {

theForm.setPASSWORD(encodePassword(theForm.getPASSWORD().trim(),"xxxx@com.cn"));

}

TbSysUserinfoObj obj = new TbSysUserinfoObj();

try {

BeanUtils.copyProperties(obj, theForm);

} catch (IllegalAccessException e) {

e.printStackTrace();

} catch (InvocationTargetException e) {

e.printStackTrace();

}

String userAccount = session.get("account").toString();

int flagForAccount = 0;

/\*\*\*\*\* 权限设置 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

TbGlobalConfigObj gObj = new TbGlobalConfigObj();

gObj.setKEY("user\_auth");

gObj = tbGlobalConfigService.queryByObj(gObj);

String[] users = new String[] { "" };

if (gObj.getVALUE() != null && !("").equals(gObj.getVALUE())) {

users = gObj.getVALUE().split(",");

}

for (int i = 0; i < users.length; i++) {

if (userAccount.equals(users[i])) {

flagForAccount = 1;

break;

}

}

if (flagForAccount == 1) {

int ret = 0;

TbSysOperationLogObj operObj = this

.getTbSysOperationLogObj(request);

int result = 0;

operObj.setREMARK("");

if (theForm.getID() == 0) {

obj.setCREATEUSER((Integer) session.get("id"));

TbSysUserinfoObj tempObj2 = new TbSysUserinfoObj();

tempObj2.setACCOUNT(obj.getACCOUNT());

TbSysUserinfoObj retObj = userInfoService.queryByObj(tempObj2);

if (retObj != null) {

request.setAttribute("msg", "账号已经存在");

return FAILURE;

}

ret = userInfoService.insertByObj(obj);

operObj.setOPERTYPE(1);

operObj.setMESSAGE("新增用户" + obj.getACCOUNT());

} else {

if (!"1".equals(theForm.getModifyPassword())) {

// 不进行密码更新

theForm.setPASSWORD(null);

}

ret = userInfoService.updateByObj(obj);

operObj.setOPERTYPE(3);

operObj.setMESSAGE("修改用户" + obj.getACCOUNT() + "信息");

}

if (ret > 0) {

result = 1;

theForm = new UserInfoForm();

}

operObj.setRESULT(result);

int retOper = operationService.insertByObj(operObj);// 写操作日志

if (ret < 0) {

return FAILURE;

}

}

return "redirect";

}

* 合规说明：

将原始密码+固定盐值或者固定盐值+用户名+原始密码后再进行MD5加密的方式。

1. 密码弱加密（AES ECB模式）

* 漏洞描述：

密码的加密算法不够强壮，可导致加密数据更容易破解。

* 不合规的代码示例：

byte[] raw = "1qaz@WSX!QAZ2wsx".getBytes("ASCII");

SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");

Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");

cipher.init(Cipher.ENCRYPT\_MODE, skeySpec);

byte[] byteContent = vin.getBytes("utf-8");

byte[] resultContent = cipher.doFinal(byteContent);

* 不合规说明：

使用AES的ECB模式进行加密。

* 合规的代码示例：

byte[] raw = "1qaz@WSX!QAZ2wsx".getBytes("ASCII");

SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");

Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

cipher.init(Cipher.ENCRYPT\_MODE, skeySpec);

byte[] byteContent = vin.getBytes("utf-8");

byte[] resultContent = cipher.doFinal(byteContent);

* 合规说明：

使用CBC模式进行加密。

## 敏感数据

1. 用户密码明文存储

* 漏洞描述：

明文密码是指网站在存储用户密码与资料时采用的是可以看得懂的明文字符。从信息安全的角度出发，任何网络服务都不应该保存或发送明文密码。

* 不合规的代码示例：

public void addUser(String loginno, String pwd, String email,

String loginname, String roleno, String rolesate, String phone,

String memo, String orgUnitId, String pwdDay, String alertpwdDay,

String userNo, LoginOprVO loginOprVO) throws Exception {

// 校验角色，所赋予角色不能有父子关系

StringBuffer sb = new StringBuffer();

sb.append(" SELECT c.role\_code, c.role\_name,")

.append(" (SELECT MAX (LEVEL) ")

.append(" FROM op\_role\_level\_dict a, op\_role\_dict b ")

.append(" WHERE a.role\_code = b.role\_code ")

.append(" START WITH a.role\_code = c.role\_code")

.append(" CONNECT BY PRIOR a.parent\_code = a.role\_code) max\_level ")

.append(" FROM op\_role\_dict c ")

.append(" WHERE c.role\_code IN ( ")

.append(roleno.replaceFirst(",", ""))

.append(")")

.append(" GROUP BY c.role\_code, c.role\_name ");

List<ListOrderedMap> results = this.getGenericDao().findForList(

sb.toString());

String one = null;

for (int i = 0; i < results.size(); i++) {

ListOrderedMap reuslt = results.get(i);

if (i == 0) {

one = String.valueOf(reuslt.get("MAX\_LEVEL"));

} else {

if (!one.equals(String.valueOf(reuslt.get("MAX\_LEVEL")))) {

throw new BusinessException("所授予的角色隶属不同层级，请修改核对！");

}

}

}

// 设置工号信息

OpLoginmsgDict user = new OpLoginmsgDict();

user.setLoginNo(loginno); // 工号

user.setLoginName(loginname); // 姓名

user.setLoginPassword(pwd); // 密码

user.setGroupId(orgUnitId); // 组织单元

user.setValidFlag(rolesate); // 工号状态

user.setContactPhone(phone); // 联系电话

user.setMaxErrnum(new Long(0)); // 最大错误次数

user.setLoginType("0"); // 新增用户所属的系统，0为本系统

user.setModifyPasswdInterval(new Long(0));

user.setOpNotes(memo); // 备注

user.setCreateLogin(userNo); // 操作者id

user.setEmail(email);

// 允许登录时间

Calendar calendar = Calendar.getInstance();

user.setAllowBegin(calendar.getTime());

// 密码修改时间

user.setPassTime(calendar.getTime());

// 生效时间

user.setCreateTime(calendar.getTime());

// 允许登录截止时间

calendar.add(Calendar.DATE, Integer.parseInt(pwdDay));

user.setAllowEnd(calendar.getTime());

// 失效时间

user.setDestroyTime(calendar.getTime());

// 密码修改时间间隔

user.setModifyPasswdInterval(Long.parseLong(pwdDay));

// 密码到期时间

user.setExpireTime(calendar.getTime());

dao.saveUser(user);

// 工号角色关联

String[] arr = roleno.split(",");

List lst = new ArrayList();

for (int i = 0; i < arr.length; i++) {

if ("".equals(arr[i]) != true) {

OpLoginroleRel userRole = new OpLoginroleRel();

OpLoginroleRelId userRoleId = new OpLoginroleRelId(loginno,

Long.parseLong(arr[i].toString()));

userRole.setId(userRoleId);

userRole.setCreateLogin(userNo);

userRole.setBeginDate(user.getAllowBegin());

userRole.setEndDate(user.getAllowEnd());

lst.add(userRole);

}

}

dao.saveUserRole(lst);

// 新建用户时创建默认工作空间配置

dao.setWorkSpaceDefault(loginno);

// 日志信息

loginOprVO.setOpDescribe("添加用户成功");

loginOprVO.setOperateResult("1");

loginOprVO.setOperateType("1");

}

* 不合规说明：

用户的密码直接明文存储到数据库中。

* 合规的代码示例：

public void addUser(String loginno, String pwd, String email,

String loginname, String roleno, String rolesate, String phone,

String memo, String orgUnitId, String pwdDay, String alertpwdDay,

String userNo, LoginOprVO loginOprVO) throws Exception {

// 校验角色，所赋予角色不能有父子关系

StringBuffer sb = new StringBuffer();

sb.append(" SELECT c.role\_code, c.role\_name,")

.append(" (SELECT MAX (LEVEL) ")

.append(" FROM op\_role\_level\_dict a, op\_role\_dict b ")

.append(" WHERE a.role\_code = b.role\_code ")

.append(" START WITH a.role\_code = c.role\_code")

.append(" CONNECT BY PRIOR a.parent\_code = a.role\_code) max\_level ")

.append(" FROM op\_role\_dict c ")

.append(" WHERE c.role\_code IN ( ")

.append(roleno.replaceFirst(",", ""))

.append(")")

.append(" GROUP BY c.role\_code, c.role\_name ");

List<ListOrderedMap> results = this.getGenericDao().findForList(

sb.toString());

String one = null;

for (int i = 0; i < results.size(); i++) {

ListOrderedMap reuslt = results.get(i);

if (i == 0) {

one = String.valueOf(reuslt.get("MAX\_LEVEL"));

} else {

if (!one.equals(String.valueOf(reuslt.get("MAX\_LEVEL")))) {

throw new BusinessException("所授予的角色隶属不同层级，请修改核对！");

}

}

}

// 设置工号信息

OpLoginmsgDict user = new OpLoginmsgDict();

user.setLoginNo(loginno); // 工号

user.setLoginName(loginname); // 姓名

MD5 md5 = new MD5();

user.setLoginPassword(md5.getMD5ofStr(pwd,loginno+Constant.SALT\_KEY)); // 密码 = md5(密码+工号+盐值)

user.setGroupId(orgUnitId); // 组织单元

user.setValidFlag(rolesate); // 工号状态

user.setContactPhone(phone); // 联系电话

user.setMaxErrnum(new Long(0)); // 最大错误次数

user.setLoginType("0"); // 新增用户所属的系统，0为本系统

user.setModifyPasswdInterval(new Long(0));

user.setOpNotes(memo); // 备注

user.setCreateLogin(userNo); // 操作者id

user.setEmail(email);

// 允许登录时间

Calendar calendar = Calendar.getInstance();

user.setAllowBegin(calendar.getTime());

// 密码修改时间

user.setPassTime(calendar.getTime());

// 生效时间

user.setCreateTime(calendar.getTime());

// 允许登录截止时间

calendar.add(Calendar.DATE, Integer.parseInt(pwdDay));

user.setAllowEnd(calendar.getTime());

// 失效时间

user.setDestroyTime(calendar.getTime());

// 密码修改时间间隔

user.setModifyPasswdInterval(Long.parseLong(pwdDay));

// 密码到期时间

user.setExpireTime(calendar.getTime());

dao.saveUser(user);

// 工号角色关联

String[] arr = roleno.split(",");

List lst = new ArrayList();

for (int i = 0; i < arr.length; i++) {

if ("".equals(arr[i]) != true) {

OpLoginroleRel userRole = new OpLoginroleRel();

OpLoginroleRelId userRoleId = new OpLoginroleRelId(loginno,

Long.parseLong(arr[i].toString()));

userRole.setId(userRoleId);

userRole.setCreateLogin(userNo);

userRole.setBeginDate(user.getAllowBegin());

userRole.setEndDate(user.getAllowEnd());

lst.add(userRole);

}

}

dao.saveUserRole(lst);

// 新建用户时创建默认工作空间配置

dao.setWorkSpaceDefault(loginno);

// 日志信息

loginOprVO.setOpDescribe("添加用户成功");

loginOprVO.setOperateResult("1");

loginOprVO.setOperateType("1");

}

* 合规说明：

对于密码进行加密，并采用工号+固定盐值+原始密码进行MD5加密的方式。

1. 敏感信息泄漏（配置文件）

* 漏洞描述：

配置文件中一般都会配置数据库的数据源信息，其中包括数据库地址、端口号、数据库名、用户名和密码都敏感信息。如果数据库的用户名和密码是明文存放的话，则会造成敏感信息的泄漏。

* 不合规的代码示例：

datasource.driverClassName=oracle.jdbc.driver.OracleDriver

datasource.url=jdbc:oracle:thin:@IP地址:端口号:数据库名称

datasource.username=admin

datasource.password=password

* 不合规说明：

在数据库配置文件中，数据库的连接用户名和密码都是明文存储的。

* 合规的代码示例：

datasource.driverClassName=oracle.jdbc.driver.OracleDriver

datasource.url=jdbc:oracle:thin:@IP地址:端口号:数据库名称

datasource.username=U2FsdGVkX1/C3z+Utg962bSu1TT/7o4+

datasource.password=U2FsdGVkX19bBJbey0cGWgjhgUcUPKu4ut1k+a6mIJA=

* 合规说明：

对数据库的用户名和密码进行加密后使用。

## 错误处理

1. 数据库交互中的异常处理

* 漏洞描述：

异常处理（又称为错误处理）功能提供了处理程序运行时出现的任何意外或异常情况的方法。异常处理使用 try、catch 和 finally 关键字来尝试可能未成功的操作，处理失败，以及在事后清理资源。很多时候系统直接将这些报错的异常信息传递给前端页面进行展示，过于详细的错误提示可能被攻击者利用。

* 不合规的代码示例：

public String addRole()throws Exception{

//角色编号

String rolecode = request.getParameter("rolecode");

//角色名称

String rolename = request.getParameter("rolename");

rolename = java.net.URLDecoder.decode(rolename, "UTF-8");

//描述

String roledescribe = request.getParameter("roledescribe");

roledescribe = java.net.URLDecoder.decode(roledescribe, "UTF-8");

//使用标志

String useflag = request.getParameter("useflag");

//上级角色id 因为数据类型的关系，所以为0

String pRoleId = request.getParameter("pRoleId");

//登录者的工号--创建者

String userNo = ((SessionVO)session.get(ConstVar.SESSIONVO)).getLoginNo();

LoginOprVO lvo=this.getLoginOprVo();

String groupId=lvo.getChnOrgUnitId();

HttpServletResponse response = ServletActionContext.getResponse();

response.setContentType("text/html; charset=GB2312");

response.setHeader("Cache-Control", "no-cache");

PrintWriter out = null;

String str = "";

int flag = 0;

try {

rightManageService.addRole(groupId,rolename,roledescribe, useflag, pRoleId, userNo,getLoginOprVo());

out=response.getWriter();

str = "{\"retCode\":\""+flag+"\",\"retMsg\":\"新增角色操作完成！\",\"detailMsg\":\"新增角色操作完成\"}" ;

out.print(str);

} catch (Exception e) {

flag = 1;

str = "{\"retCode\":\""+flag+"\",\"retMsg\":\"新增角色操作失败！\",\"detailMsg\":\"新增角色操作失败"+ StrintTool.replaceSpecial(e.getMessage()) +"\"}" ;

out.print(str);

e.printStackTrace();

}finally{

if(null != out)

out.close();

}

return null;

}

* 不合规说明：

程序将捕获到的向数据库添加角色时的异常详细信息，直接返回前台，显示给用户。

* 合规的代码示例：

public String addRole()throws Exception{

//角色编号

String rolecode = request.getParameter("rolecode");

//角色名称

String rolename = request.getParameter("rolename");

rolename = java.net.URLDecoder.decode(rolename, "UTF-8");

//描述

String roledescribe = request.getParameter("roledescribe");

roledescribe = java.net.URLDecoder.decode(roledescribe, "UTF-8");

//使用标志

String useflag = request.getParameter("useflag");

//上级角色id 因为数据类型的关系，所以为0

String pRoleId = request.getParameter("pRoleId");

//登录者的工号--创建者

String userNo = ((SessionVO)session.get(ConstVar.SESSIONVO)).getLoginNo();

LoginOprVO lvo=this.getLoginOprVo();

String groupId=lvo.getChnOrgUnitId();

HttpServletResponse response = ServletActionContext.getResponse();

response.setContentType("text/html; charset=GB2312");

response.setHeader("Cache-Control", "no-cache");

PrintWriter out = null;

String str = "";

int flag = 0;

try {

rightManageService.addRole(groupId,rolename,roledescribe, useflag, pRoleId, userNo,getLoginOprVo());

out=response.getWriter();

str = "{\"retCode\":\""+flag+"\",\"retMsg\":\"新增角色操作完成！\",\"detailMsg\":\"新增角色操作完成\"}" ;

out.print(str);

} catch (Exception e) {

flag = 1;

log.info("新增角色操作失败： " + e.getMessage());

str = "{\"retCode\":\""+flag+"\",\"retMsg\":\"新增角色操作失败！\",\"detailMsg\":\"新增角色操作失败\"}" ;

out.print(str);

e.printStackTrace();

}finally{

if(null != out)

out.close();

}

return null;

}

* 合规说明：

程序未将这些数据库报错的详细信息直接展示给前端用户，而是将详细错误信息输出到日志文件中。

## 日志安全

1. 日志记录敏感信息

* 漏洞描述：

系统的日志功能会记录详细的用户操作记录以及系统运行时的报错及状态信息等，在日志中可能会记录一些类似用户名、密码之类的用户敏感信息。

* 不合规的代码示例：

public boolean sPassCheck(String loginNo, String phoneNum, String passwd) {

log.info("loginNo = " + loginNo + " phoneNum = " + phoneNum + " passwd = " + passwd);

return this.myBuyerImpl.checkPasschk(loginNo, phoneNum, passwd);

}

* 不合规说明：

在用户校验过程中，程序将用户名、密码、手机号等敏感信息写入到日志文件当中。

* 合规的代码示例：

public boolean sPassCheck(String loginNo, String phoneNum, String passwd) {

return this.myBuyerImpl.checkPasschk(loginNo, phoneNum, passwd);

}

* 合规说明：

已经将这些敏感信息写入日志的代码删除。如果需要记录日志，则按照3.7.1中记录日志的格式进行日志输出。

## 未验证的重定向

1. URL重定向

* 漏洞描述：

在Web应用中重定向是极为普遍的，并且通常重定向所引向的目的是带有用户输入参数的目的URL，而如果这些重定向未被验证，那么攻击者就可以引导用户访问他们所要用户访问的站点。

* 不合规的代码示例：

<%

String z = request.getParameter("z");

if(null != z && z.indexOf("http") != -1) {

CasJSON cj = (CasJSON) session.getAttribute("CAS");

int orgId = 0;

try {

orgId = cj.getMsg().getOrgId().intValue();

if(-1 == z.indexOf('?')) {

z += "?groupid=" + orgId;

} else {

z += "&groupid=" + orgId;

}

response.sendRedirect(z);

} catch (Exception e) {}

}

%>

* 不合规说明：

对用户传入的URL未做任何的校验。

* 合规的代码示例：

<%

String[] strUrl = {"域名地址1","域名地址2","域名地址3"};

String z = request.getParameter("z");

if(null != z && z.indexOf("http") != -1) {

boolean isYes = false;

for(String url:strUrl){

if(z.startsWith(url)){

isYes = true;

break;

}

}

if(!isYes){

response.sendRedirect(request.getContextPath()+"/404.jsp");

}else{

CasJSON cj = (CasJSON) session.getAttribute("CAS");

int orgId = 0;

try {

orgId = cj.getMsg().getOrgId().intValue();

if(-1 == z.indexOf('?')) {

z += "?groupid=" + orgId;

} else {

z += "&groupid=" + orgId;

}

response.sendRedirect(z);

} catch (Exception e) {}

}

}

%>

* 合规说明：

使用白名单的方式对用户传入的z参数进行限制。