**系统异常设计规范与原则**

* 程序异常理论概述
* JAVA 业务系统异常设计实践
* 常见的错误的异常处理方式
* 程序异常理论概述

当我们系统出现异常时怎么办?

### **系统异常设计的出发点：**

1. 良好的异常信息展示，开发运维人员能快速定位问题。
2. 响应外部调用异常时，应能明确指明是内部异常还是调用条件不满足导至。
3. 响应用户操作异常时，能友好的提示用户。

如何做到以上3点？

**首先**我们需要对异常进行分类

一．内部异常

1. 资源环境导致（系统环境异常、数据库连接超时、第三方服务响应超时）
2. 第三方服务错误响应
3. 第三方响应结果错误
4. 外部传入参数非法
5. 错误的编码逻辑
6. 错误的配置
7. 异常的业务数据（业务数据缺失）

二．业务异常

1. 用户操作错误
2. 业务条件不满足

**其次**需要在系统中正确的捕获这类异常，并抛出。

一.方法入参进行合法性验证。

1. 对系统外部提供的接口，是必须要进行参数验证（必须）
2. 系统内部对外外层提供接口，进行验证
3. 工具类进行参数验证
4. public 方法要进行验证
5. private 方法(不建议参数验证)

二．第三方响应结果合法性验证。

1. 获取第三方法结果后，根据你们的约定进行验证。

三．业务处理前，对业务业务前置条件进行验证。

1. 业务处理前，验证业务条件（验证佘额、验证这个帐户有没有被公安门锁定）
2. 要考虑性能成本（验证身份证号码是不是存在的）

四．业务处理后，对处理结果进行验证。

1. 验证对方帐户是不是到帐了，转出帐户是不是成功扣款

五．对于可能会出现异常的代码进行 try catch 捕获。

1. 尝试恢复处理
2. 直接抛出
3. 转换后抛出

**最后** 在系统出口统一拦截处理。

统一拦截的目的是确保出去的异常是可控的， 调用方能够明白的异常信息。

这里出口是指系统对外统一响应逻辑，一般我们可分三类场景

一．WEB Response

1. 内部异常：引导至异常提示页。
2. 业务异常：返回对应提示消息至前端。
3. 未知异常：尝试进行认别，如果认识不了，转换成 编码异常(BUG)。

二．Http API 接口响应

1. 内部异常：返回接口不可用消息。
2. 参数错误：基于API文档中的异常列表进行进行响应返回。表明参数非法，需要调用方加强参数合法性验证
3. 业务错误：基于掊口约定反回对应code与消息。

三．RPC Service 响应

1. 内部异常：返回服务不可用消息
2. 参数错误：基于接口文档进行响应，直接返回异常堆栈。
3. 业务错误 ：直接返回异常堆栈。

### **checkedException 与uncheckedException 声明原则**

1. 如果是参数非法抛出,返回结果非法（即软件BUG） uncheckedException
2. 如果你认为调用方程序员需要有意识地采取措施，那么抛出检查型异常。
3. 程序产品有明确的条件约束的要求，可声明检测型业务异常

### **总结异常设计与处理原则**

## JAVA 业务系统异常设计实践

监控系统无法区分异常code和msg是什么东西,必须去代码里面去查看，无法快速定位，易重复

### **异常的定义技巧**

基于分包表示异常的分类，不建议使用继承

创建异常来类定义业务异常，不建议使用Code来定义

使用枚举来表示业务异常的几种结果，不建议使用code

### **统一对异常进行分类处理**

异常转换

异常信息处理

逻辑断言

参数合法性验证

返回结果合法性验证

### **异常捕获**

### **统一对异常进行拦截处理**

目的：防止不明确的异常流出系统

RPC Service 响应拦截

Web Control 响应拦截

Http API 响应拦截

## 常见的错误的异常处理方式

**直接勿略异常:**

**try** {

**new** String(source.getBytes("UTF-8"), "GBK");

} **catch** (UnsupportedEncodingException e) {

e.printStackTrace();

}

**正确的处理**

try {

new String(source.getBytes("UTF-8"), "GBK");

} catch (UnsupportedEncodingException e) {

throw new RuntimeException("环境不支持UTF-8",e)

}

业务异不提供任何信息

**public** **class** DuplicateUsernameException **extends** Exception {

}

给每个异常处理都定义一个Code

用一个统一异常替代所有业务异常

public class ServiceException extends RuntimeException {

public ServiceException(Exception e) {

super(e);

}

public ServiceException(String message) {

super(message);

}

}

错误：1 、必须明确定义业务异常 2、 尽可能申明成checkedException 3要带上具体的业务数

正确方式：定义明确的业务异常