JAVA 项目开发规范

版本：V2.0

文 档 编 号	JY-SOFT_J-001	保 密 等 级	保密
作　　者	张鹏宏	最后修改日期	
审 核 人		最后审批日期	
批 准 人		最后批准日期	

修订历史记录

日期	文档版本号	修订类型	说明	作者
2014-08-04	1.0	A	根据信息技术部要求制定 JAVA 项目开发规范	张鹏宏
2014-08-11	1.1	M	增加第 6 章事务使用规范；增加 4.4 流水号的定义规范；增加表 3 代码示例；	张鹏宏
2014-09-02	1.2	M	增加 4.1.2 章节增加静态变量定义规则；	张鹏宏
2014-09-03	1.3	M	增加 6.2 事务配置方式	邵广玉
2014-09-03	1.4	M	增加第 8 章集群部署规范	别双平
2014-09-17	1.5	A	增加 4.5 节关于 Controller 的定义规范；修改了 4.3 错误与异常的定义规范；修改了 2.3 接口名称的定义规则，采用 I+功能名称；	张鹏宏
2014-09-18	1.5	A	增加4.6章国际化详细说明了国际化信息的定义以及存取方式	陈钢
2016-07-18	2.0	A	增加子系统接口服务规范	陈钢

说明：修改类型 A 标示新增；D 标示删除；M 标示修改。

目　录

表目录

第1章 引言

本文档预期读者为信息技术部所有参与项目开发的人员，包括项目经理及开发人员，为了执行本规范，并提高部门软件项目开发的效率和质量，要求所有开发人员必须一致遵守。

编码规范在项目开发过程中具有非常重要的指导意义：

- 提高项目开发的整体质量；
- 提高代码的可读性；

制定编码规范的目标和要求：

- 统一编码风格；
- 统一命名规范；
- 统一项目结构；

第2章 命名规范

2.1 包的命名规范

J2EE 项目开发包名基本命名规则：com+jy+工程简写+具有实际意义的功能类别+……；

- jy 为公司名称捷越拼音首字母；
- 包名由小写字母组成，尽量用具有实际意义的单一单词定义；
- 功能类别可以根据包下实体类具体所要实现的功能来定义，
  - ✓ Controller：业务逻辑控制处理 Aciton；
  - ✓ Service：为 Controller 提供基础数据处理服务；
  - ✓ Mapper：数据存取操作接口；
  - ✓ Module：数据模型或数据映射实体；
  - ✓ Utils：可复用的工具类；
  - ✓ Filter：过滤器，主要包括编码转义过滤器、事物拦截过滤器；
  - ✓ Servlet：servlet 服务包；

举例：

UserCenter 系统英文简称为 uc，那么基础服务完成包命名，com.jy.uc.service ;

2.2 类名的命名规范

类的命名需要遵循以下规范

- 类名要用名词或名词短语定义，首字母大写，不同单词首字母大写；
- 不同功能类别下类的命名规则如下：
  - ✓ Controller：业务名称+ Controller；
  - ✓ Service：功能+Service；
  - ✓ Model：表名+Table；
  - ✓ Mapper：表名简称+Mapper；
  - ✓ Filter：业务名称+Filter；

2.3 接口命名规范

接口命名需要遵循以下规范

- 类名要用名词或名词短语定义，首字母大写，不同单词首字母大写；
- I+业务名称/功能说明；

2.4 配置文件命名规范

Mybatis 配置文件的命名规则需要遵循以下规则，

- Mapper 命名
  - ✓ 单表查询的 Mapper 文件名称：表名+Table+-+Mapper.xml；
  - ✓ 视图查询的 Mapper 文件名称：视图名称+View+-+Mapper.xml；
- Configuration 命名
  - ✓ Mybatis 的全局参数配置，文件名称：Mybatis-Configuration.xml；
- Spring 配置文件命名
  - ✓ 持久层 Context 配置文件名称：mybatis-context.xml；
  - ✓ 缓存数据库配置文件名称：redis-context.xml；
  - ✓ Spring 容器配置文件名称：servlet-context.xml；
- 属性文件命名
  - ✓ 数据库地址、用户名称、密码等属性配置文件名称：jdbc.properties；
  - ✓ 缓存数据库相关参数配置文件：redis.properties；

2.5 字段

2.5.1 变量

静态长量采用完整的英文大写单词，在词与词之间用下划线连接，如：DEFAULT_VALUE；

变量命名尽量采用有意义的单词或简写，不允许包含特殊字符和数字。

2.5.2 变量和参数

变量及参数命名需要遵循以下规范

- 尽量采用具有实际意义的单词作为变量名称；
- 除第一各个单词外其余单词首字母大写；
- 对静态常亮的定义要采用全部大写，不同单词间采用“_”下划线分隔，例如

public String static final USER_TYPE = “1”。

2.5.3 组件/部件

应采用完整的英文描述符命名组件（接口部件），

如：btnOK，lblName。

2.5.4 集合

一个集合，例如数组和矢量，应采用复数命名来表示队列中存放的对象类型。命名应采用完整的英文描述符，名字中所有非开头的单词的第一个字母应大写，适当使用集合缩写前缀。如：

Vector vProducts = new Vector(); //产品向量

Array aryUsers = new Array(); //用户列表

2.6 方法

方法的命名应遵循以下规范：

- 方法的命名应采用完整的英文描述符，大小写混合使用：所有中间单词的第一个字母大写。方法名称的第一个单词常常采用一个有强烈动作色彩的动词；
- 取值类使用 get 前缀，设值类使用 set 前缀，判断类使用 is(has)前缀；

例： getName()

setSarry()

isLogon()

- 方法参数建议顺序：(被操作者，操作内容，操作标志，其他…) ；

例：public void replace(String sourceStr, //源字串

String oldStr, //被替换字串

String newStr){ //替换为字串

2.7 异常

异常类名由表示该异常类型的单词和Exception组成，如ActionException。

异常实例一般使用e、ex等，在多个异常时使用该异常名或简写加E，Ex等组成，如：

sqlEx

actionEx

表1　命名约定表

操作项	命名约定	示例
参数	使用传递值/对象的完整的英文描述符。	userName
字段/属性	字段采用完整的英文描述，第一个字母小写，任何中间单词的首字母大写。	firstName
布尔型的获取成员函数	所有的布尔型获取函数必须用单词 is（has）做前缀。	isString() hasMoney() stringOrEmpty()
类	采用完整的英文描述符，所有单词的第一个字母大写。	CustomerController.java

编译单元文件	使用类或接口的名字，或者如果文件中除了主类之外还有多个类时，加上后缀 java 来说明它是一个源码文件。	Customer.java
部件/组件	使用完整的英文描述来说明组件的用途，将组件类型使用匈牙利命名法则作其前缀。	clickOK,checkType
构造函数	使用类名	Customer()
析构函数	Java 没有析构函数，但一个对象在垃圾收集时，调用成员函数 finalize() 。	finalize()
异常类名	由表示该异常类型等的单词和Exception组成	SQLException ActionException
异常实例名	通常采用字母 e 、ex 表示异常。 多个异常时使用异常名或其简写加E、Ex等构成	e ex
静态常量字段（常量）	全部采用大写字母，单词之间用下划线分隔。采用静态常量获取成员函数。	DEFAULT_NAME
获取成员函数	被访问字段名的前面加上前缀 get。	getUserName()

接口	采用完整的英文描述符说明接口封装，所有单词的第一个字母大写。使用I前缀，其后使用able，或者 er等后缀，但这不是必需的	RunnableInterface PrompterInterface
局部变量	采用完整的英文描述符，第一个字母小写，但不要隐藏已有字段。例如，如果有一个字段叫 firstName，不要让一个局部变量叫 firstName。	strName,totalMoney
循环计数器	通常采用字母 i，j，k 或者 counter，index	i,j,k,count,index
包	采用完整的英文描述符，所有单词都小写，多个单词以下划线相连。所有包都属于 org.jy	com.jy.demo com.jy.demo.dao
成员函数	采用完整的英文描述说明成员函数功能，第一个单词尽可能采用一个生动的动词，除第一个单词外，每个单词第一个字母小写。	openFile() addUser()
设置成员函数	被访问字段名的前面加上前缀 set。	setName() setPower()

第3章 注释规范

3.1 基本原则

- ✧ 增加程序的可读性；
- ✧ 注释信息要结构清晰，内容简洁；
- ✧ 注释信息要重点描述代码要实现的功能，涉及代码修改变更的地方需要着重说明；
- ✧ 变量及短语的行尾注释不换行；

3.2 JavaDoc

对类、方法、变量等的注释需要符合 JavaDoc 规范，对每个类、方法都应详细说明其功能、条件、参数等，并使用良好的 HTML 标记格式化注释，以使生成的 JavaDoc 易阅读和理解。

3.3 文件注释

在每个文件的头部都应该包含该文件的功能、作用、作者、版权以及创建、修改记录等。并在其中使用 SVN 标记自动跟踪版本变化及修改记录等信息。注意是/**/注释而不是/***/JavaDoc 注释。

文件(File)注释模板示例如下：

```
/*
 * Copyright (C) 2006-${year} 北京捷越联合信息咨询有限公司.
 * 本系统是商用软件,未经授权擅自复制或传播本程序的部分或全部将是非法的.
```

```
    ================================================
=======
     * FileName: ${file_name}
     * Created: [${date} ${time}] by ${user}
     * $$Id$$
     * $$Revision$$
     * $$Author$$
     * $$Date$$
    ================================================
=======
     * ProjectName: ${project_name}
     * Description:
    ================================================
=====*/
```

3.4 类、接口注释

在类、接口定义之前当对其进行注释，包括类、接口的目的、作用、功能、继承于何种父类，实现的接口、实现的算法、使用方法、示例程序等，在作者和版本域中使用 SVN 标记自动跟踪版本变化等，具体参看注释模板。

类(Types)注释模板示例如下：

```
    /**
     * Description:
     * @author ${user}
     * @version 1.0
     * <pre>
```

```
     * Modification History:
     * Date         Author      Version     Description
    ------------------------------------------------------------------
---------
     * ${date}     ${user}        1.0         1.0 Version
     * </pre>
     */
```

3.5 方法注释

依据标准 JavaDoc 规范对方法进行注释，以明确该方法功能、作用、各参数含义以及返回值等。复杂的算法用/**/在方法内注解出。

参数注释时当注明其取值范围等

返回值当注释出失败、错误、异常时的返回情况。

异常当注释出什么情况、什么时候、什么条件下会引发什么样的异常。

方法(Methods)注释模板示例如下：

```
    /**
     * Description:
     * @param
     * @return ${return_type}
     * @throws
     * @Author ${user}
     * Create Date: ${date} ${time}
     */
```

3.6 其他注释

应对重要的变量加以注释，以说明其含义等。

应对不易理解的分支条件表达式加注释。不易理解的循环，应说明出口条件。过长的方法实现，应将其语句按实现的功能分段加以概括性说明。

对于异常处理当注明正常情况及异常情况或者条件，并说明当异常发生时程序当如何处理。

表2　注释参考表

注释参考表

项目	注释内容
参数	参数类型 参数用来做什么 约束或前提条件 示例
字段/属性	字段描述 注释所有使用的不变量 示例 并行事件 可见性决策
类	类的目的 已知的问题 类的开发/维护历史、版本 注释出采用的不变量 并行策略

编译单元 （文件）	每一个类/类内定义的接口，含简单的说明 文件名和/或标识信息 修改/维护记录 版权信息
获取成员函数	若可能，说明为什么使用滞后初始化
接口	目的 它应如何被使用以及如何不被使用
局部变量	用处/目的
成员函数注释	成员函数做什么以及它为什么做这个 哪些参数必须传递给一个成员函数 成员函数返回什么 已知的问题 任何由某个成员函数抛出的异常 可见性决策 成员函数是如何改变对象的 包含任何修改代码的历史 如何在适当情况下调用成员函数的例子 适用的前提条件和后置条件
成员函数内部注释	控制结构 代码做了些什么以及为什么这样做 局部变量

难或复杂的代码处理顺序	
包	包的功能和用途

第4章 编码规范

4.1 方法

4.1.1 基本原则

一个方法只完成一项功能，在定义系统的公用接口方法外的方法应尽可能的缩小其可见性。

避免用一个类是实例去访问其静态变量和方法。

避免在一个较长的方法里提供多个出口：

```
//不要使用这钟方式，当处理程序段很长时将很难找到出口点
if(condition){
return A;
}else{
return B;
}
//建议使用如下方式
String result = null;
if(condition){
result = A;
}else{
result = B;
}
return result;
```

4.1.2 参数和返回值

不允许定义静态具有业务意义的变量，可能会造成集群部署服务器中数据不一致；

避免过多的参数列表，尽量控制在5个以内，若需要传递多个参数时，当使用一个容纳这些参数的对象进行传递，以提高程序的可读性和可扩展性。

参数类型和返回值尽量接口化，以屏蔽具体的实现细节，提高系统的可扩展性，例如：

```
public void joinGroup(List userList){}

public List listAllUsers(){}
```

4.2 表达式与语句

4.2.1 基本原则

- 表达式和语句当清晰、简洁，易于阅读和理解，避免使用晦涩难懂的语句；
- 每行至多包含一条执行语句，过长当换行；
- 避免在构造方法中执行大量耗时的初始化工作，应当将这中工作延迟到被使用时再创建相应资源，如果不可避免，则当使用对象池和Cache等技术提高系统性能；
- 避免在一个语句中给多个变量赋相同的值。它很难读懂；
- 不要使用内嵌(embedded)赋值运算符试图提高运行时的效率，这是编译器的工作；
- 尽量在声明局部变量的同时初始化。唯一不这么做的理由是变量的初始值依赖于某些先前发生的计算；
- 一般而言，在含有多种运算符的表达式中使用圆括号来避免运算符优先级问题，是个好方法。即使运算符的优先级对你而言可能很清楚，但对其他人未必如此。你不能假设别的程序员和你一样清楚运算符的优先级；
- 不要为了表现编程技巧而过分使用技巧，简单就好。

4.2.2 控制语句

判断中如有常量，则应将常量置与判断式的左侧。如：

```
if ( true == isAdmin())...
```

```
if ( null == user)...
```

尽量不使用三目条件判断。

所有if语句必须用{}包括起来,即便是只有一句：

```
if (true){
//do something......
}
if (true)
i = 0; //不要使用这种
```

当有多个else分句时当分别注明其条件，注意缩进并对齐，如：

```
//先判断i 是否等于1
if (i == 1){//if_1
//.....
}//然后判断i == 2
else if (i == 2){
//i == 2说明。。。。。。
j = i;
}//如果都不是(i > 2 || i < 1)
else{
//说明出错了
```

```
//....
}//end if_1
```

过多的else分句请将其转成switch语句或使用子函数。

每当一个case顺着往下执行时(因为没有break语句)，通常应在break语句的位置添加注释。如：

```
switch (condition) {
case ABC:
//statements;
//继续下一个CASE
case DEF:
//statements;
break;
case XYZ:
//statements;
break;
default:
//statements;
break;
}//end switch
```

4.2.3 循环语句

循环中必须有终止循环的条件或语句，避免死循环。

当在for语句的初始化或更新子句中使用逗号时，避免因使用三个以上变量，而导致复杂度提高。若需要，可以在for循环之前(为初始化子句)或for循环末尾(为更新子句)使用单独的语句。

因为循环条件在每次循环中多会执行一次，故尽量避免在其中调用耗时或费资源的操作，比较一下两种循环的差异：

```
//不推荐方式_____________________________________________

while(index < products.getCount()){

//每此都会执行一次getCount()方法，

//若此方法耗时则会影响执行效率

//而且可能带来同步问题，若有同步需求，请使用同步块或同步方法

}

//推荐方式_____________________________________________

//将操作结构保存在临时变量里，减少方法调用次数

final int count = products.getCount();

while(index < count){ }
```

4.3 错误与异常

通常情况下我们在编码规范中需要注意的是运行时异常、IO 异常的处理以及具有业务意义的异常定义，在平台中定义了俩大类异常，ValidationException 和 ErrorException，其中 ValidationException 专用于处理数据以及业务逻辑校验异常的处理，ErrorException 则专用于运行时异常或逻辑错误的处理。
异常信息的定义必须遵循国际化标准，异常信息要定义在 messages_en_US.properties、messages_zh_CN.properties 属性文件中。

错误码的定义：业务系统简称+.+validation/error+.+唯一数字代码；
提示信息的定义：若有动态替换的参数可采用{n}的方式定义；
例如，

```
uc.validation.1001=test error
uc.validation.1002=test {0} error {1}
```

4.3.1 基本原则

- 逐级上抛，外层定义

后台代码捕获异常必须逐级向上抛出，最终由展现层根据业务场景的不同，返回有意义友好的提示信息；

- 严禁捕而不抛，吞噬异常

任何情况下都不允许业务系统中代码仅捕获，而不抛出的方式隐藏异常；

- 严格鉴定，粗放处理

理论上在捕获异常时一定要严格鉴定是什么异常，不能泛泛采用 Exception 处理，而当异常抛出到展示层时，就可以采用 AbaboonException 这类具有业务意义的异常类进行定义；

4.3.2 异常的捕捉与处理

为保证代码的健壮性，必须按照要求进行异常捕捉处理，原则上一般的异常都需要抛出到控制层进行统一异常转义，决不允许捕获但不处理。

服务层代码处理多个异常应分别捕捉并处理，避免使用一个单一的catch来处理。如：

```
try {
//do something
}catch(IllegalStateException e){
Log.error(e.printStackTrace(),e);
}catch(SQLException ex){
Log.error(ex.getStackTrace(),ex);
throw ex; //抛给调用者处理
```

```
    }finally{
    //释放资源
    }
```

展示层处理异常时数据校验异常必须抛出 ValidationException，其他运行时异常以及逻辑错误异常必须抛出 ErrorException；

```
try {
       }catch (Exception e) {
          throw new ErrorException(
                getMessage("uc.validation.1002", new String[] { "1",
"2" }, new Locale("zh", "CN")), e);
       }
```

4.4 流水号的定义规范

- 应用流水号必须按照标准格式设计，定长，当 sequence+时间长度不能满足规定时采用左侧 0 补位，不同用途的流水号 Sequence 不能重用；
  - 示例：

  Sequence+时间+随机数；

4.5 展示层代码的编写

- Controller 的定义必须唯一，并需要继承公共抽象类 BaseController，同时实现 execute()或者 multiExecute（T formDatas）、multiExecute（T firstData, T secendData）、multiExecute（T firstData, T secendData , T thirdData）方法；
- 其实现的功能必须单一，在同一个 Controller 中建议定义一个@RequestMapping，同时建议使用@RequestMapping("/execute")来命名请求入口；
- 一个 Controller 代码行数不超过 500 行；
- Controller 中不允许包含 Service 层代码；
- 一个 Controller 只能实现单一功能，一个方法定义参数不能超过 3 个，4 个参数以上的情况必须定义实体 Bean；
- Controller 可使用父类中的 get 方法获取上下文数据或国际化信息；
- 每个 Jsp 页面必须增加<ababoon:viewRecall />、<ababoon:validationMessage />俩个标签；

表3　　Controller 示例

```
@Controller
@Scope("prototype")
@RequestMapping("/consign")
public class ConsignController extends BaseController {
@RequestMapping("/execute")
    public ModelAndView execute() throws AbaboonException {
```

```
        ModelAndView model = new ModelAndView();

        String opData = (String) this.getParameter("opData");
        try {
            System.out.println("@@@2:"
                    + getMessage("uc.validation.1002", new String[]
{ "1", "2" }, new Locale("zh", "CN")));
            System.out.println("-----------------opData:" + opData);
            if ("opData".equals(opData)) {
                model.setViewName("consign/consignList");
            }
        }
        catch (Exception e) {
            throw new ErrorException(
                    getMessage("uc.validation.1002", new String[] { "1",
"2" }, new Locale("zh", "CN")), e);
        }

        return model;
    }
}
```

4.6 国际化

4.6.1 基本原则

- 代码中具有业务意义的提示信息以及异常信息，不允许直接写中文或英文，必须在属性文件中定义提示信息，程序通过提示信息代码获取提示信息；
- 页面上获取后台提示信息、标题需要通过 Message 标签从上下文中获取，不允许直接将文字写入页面或 JS；

4.6.2 国际化信息的定义

项目中需要创建 messages_en_US.properties、messages_zh_CN.properties 属性文件；

文件中提示信息的定义需要遵循 key = value 格式，key 由系统简称+validation/error/message+消息代码组成，消息代码为固定长度 6 位数字。

表4 提示信息定义示例

```
uc.validation.1001=test error
uc.validation.1002=test {0} error {1}
```

4.6.3 国际化信息的获取

BaseController提供标准方法获取国际化信息：

```
String getMessage(String messageCode)
String getMessage(String messageCode, Locale locale)
String getMessage(String messageCode, Object[] args)
String getMessage(String messageCode, Object[] args, Locale locale)
String getMessage(String messageCode, Object[] args, Locale locale,
String defaultMessage)
```

表5　代码示例

```
/**
 * Copyright (C) 2006-2014
 * 版权所有者为北京捷越联合信息咨询有限公司。本系统是商用软件,未经授权擅自复制或传播本程序的部分或全部将是非法的。
 *
 * @title: DemoTest.java
 * @package com.jy.demo.web
 * @author zph
 * @date 2014-8-11 上午9:25:01
 * @version v1.00
 * @description: 该类的创建是为了给大家做一个规范的示例
 */

package com.jy.demo.web;

import org.apache.log4j.Logger;

/**
 * @classname: DemoTest
 * @description: 该类提供一些方法注释、定义的规范
 */

public class DemoTest {

    public static final Logger log = Logger.getLogger(DemoTest.class);

    private String userName;

    public static final String IDENTIFY_TYPE = "0";// 用户识别标识 0：普
```

通用户；1：超级用户；

```
    /**
     * @return the userName
     */

    public String getUserName() {
        return userName;
    }

    /**
     * @param userName the userName to set
     */

    public void setUserName(String userName) {
        this.userName = userName;
    }

    /**
     * @title: queryUser
     * @author zph
     * @description:
     * @date 2014-8-11 上午9:39:35
     * @param userName
     * @return
     * @throws Exception
     */
    public String queryUser(String userName) throws Exception {
        try {

        }
        catch (Exception e) {
            log.error("query user error:", e);
            throw e;
        }
        return userName;
    }
}
```

第5章 自测与跟踪

5.1 基本原则

测试不通过的代码不得提交到SVN库或者发布。

不得隐瞒、忽略、绕过任何Bug，有Bug不一定是你的错，但有了Bug不作为就是你的不对了。

多做测试，测试要完全，尽量将各种可能情况都测试通过，尽量将可能的Bug在开发中捕捉并处理掉。

测试要保证可再测试性。

测试应当对数据库等资源不留或少留痕迹，例如，当测试添加一个用户时，在其成功后当及时从数据库中删除该记录，以避免脏数据的产生（由此衍生的一个经验是将添加、获取、删除一起测试）。

对关键功能必须测试并通过，对辅助功能及非常简单之功能可不做测试。

5.2 Junit 单元测试

在Java应用中，单元测试使用Junit及其衍生工具。

在 TestCase 的 setUp()中初始化应用，在 tearDown()中释放资源。可由一个基础 TestCase 完成这些任务，其他 TestCase 继承之。

第6章 事务处理规范

6.1 事务的使用规范

事务的使用必须遵循以下规范：

- 凡是多表写操作，逻辑上要求数据一致性的，必须使用事务进行控制，以保证数据的完整性；
- 凡是多表删除操作，逻辑上要求数据一致性的，必须使用事物进行控制，以保证数据的完整性；
- 查询操作不需要定义事务；
- 在任何出口之前，只要存在事务未结束，必须提交或者回滚，除非有特殊设计考虑；

6.2 事务配置方式

6.2.1 公共事务配置

基于 Spring AOP，针对数据操作（增，删，改）的所有业务接口命名保持一致。

如果配置如下：

```
<tx:method name="delete*" propagation="REQUIRED" read-only="false"
   rollback-for="java.lang.Exception" no-rollback-for="java.lang.RuntimeException" />
<tx:method name="insert*" propagation="REQUIRED" read-only="false"
   rollback-for="java.lang.RuntimeException" />
<tx:method name="update*" propagation="REQUIRED" read-only="false"
   rollback-for="java.lang.Exception" />
<tx:method name="find*" propagation="SUPPORTS" />
<tx:method name="get*" propagation="SUPPORTS" />
<tx:method name="select*" propagation="SUPPORTS" />
```

删除类接口必须以 delete 开头。

新增类接口必须以 insert 开头。

更新类接口必须以 update 开头。

如果业务接口命名符合公共配置规范，在业务类中可以不再考虑事务的配置。

6.2.2 定制事务配置

基于事务标签@Transactional，适用于以下场景：

（1） 业务场景比较复杂，业务接口名称为了满足见名知义的命名规范，接口名称个性化比较强。

（2） 特定业务场景下，对于特定异常类型的处理与公共处理逻辑不同。

（3） 特定业务场景下，对于事务的传播属性或隔离级别有特殊的要求。

注意：如果要启用 @Transactional 标签，必须开启 <tx:annotation-driven transaction-manager="transactionManager"/>配置，否则事务将不能生效。

DemoService.java

```java
package com.jy.demo.dao.service;

import org.springframework.transaction.annotation.Isolation;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;

@Transactional(readOnly=true) //配置事务 查询使用 只读
public class DemoService {

    /**
     * Propagation.REQUIRED ：有事务就处于当前事务中，没事务就创建一个事务
     * isolation=Isolation.DEFAULT：事务数据库的默认隔离级别
     * readOnly=false：可写 针对 增删改操作
     * Spring默认情况下会对运行期例外(RunTimeException)进行事务回滚。这个例外是unchecked
     * 如果遇到checked意外就不回滚。 如果要改变默认的规则，可以采用rollbackFor和noRollbackFor属性

     * 注意：@Transactional放在类级别上等同于该类的每个公有方法都放上了@Transactional
     *       方法上的@Transactional会覆盖类上面声明的事务
     * */
    @Transactional (propagation=Propagation.REQUIRED,isolation=Isolation.DEFAULT,readOnly=false,
            rollbackFor=Exception.class,noRollbackFor=RuntimeException.class)
    public void demoMethod() throws Exception{

    }

}

```

第7章 集群部署规范

7.1 HttpSession

Session 中的数据必须可以被序列化，如果 session 中包含对象，需要保证对象中的每个域都可以被序列化；

Session 中不能放太多的东西，因为 session 复制时，序列化和反序列化都会造成信息损耗，session 太大，也会影响并发数量；

交易数据不允许放入 Session。

7.2 缓存

需要关闭二级缓存。

7.3 静态变量

系统中一般存在着静态常量和静态变量，静态常量一旦申明将不会被改变，对于集群中的服务器节点在使用上是一致的，但是对于静态变量则不同，集群中的节点都可以改变该变量的值，造成不一致现象，故有这类静态变量的需求是，都放入数据库表中，在程序中需要改变和使用该变量时，直接操作数据库表即可。

7.4 可变的配置文件处理

对于系统中需要操作可变的配置文件（内容经常改变），可采用数据库配置信息表进行存储，这样方便参数的修改。。

7.5 上传和下载文件处理

类似可变的配置文件处理，可以放到 blob 字段中，也可以在数据库所在服务器上，开共享磁盘空间，把上传或下载的文件放在共享磁盘空间内。

7.6 单线程和定时器

对于集群中服务器的节点，单线程的程序会变成多线程。比如一个上传任务，多个节点的单线程同时上传数据，会造成重复上传而导致出现错误，并且对于负载均衡和故障转

移都会有所影响。定时器也雷同。对于这种情况，建议采用数据库来管理单线程和定时器的任务管理。

特别说明：

（1)单例模式，有许多网上的文章都说要在集群开发时避免单例模式，其实是没有必要的，当系统切换时，如果系统本身有这个类的实例，则会运行初始化方法进行实例化。

(2)配置文件，配置文件作为系统参数，在系统运行的过程中也是不变的，所以当在集群部署时，该配置文件的读取并初始化系统参数，集群中所有机器都会初始化该参数，所以机器切换过程中也不会有什么问题

（3) 静态常量，在系统机器切换中，每个机器的静态常量都完全相同，所以都不会有什么问题。

第8章 模板的使用规范

代码的创建和格式化必须使用统一的模板：
codetemplates.xml、jy-java-cleanup-1.1.xml、jy-java-formatter-1.1.xml；
使用方法请参见《java 代码注释模板和格式化模板使用手册.docx》

第9章 ESB 接口服务规范

9.1 部署环境

ESB 部署现状	部署 IP	部署应用	服务目标
ESB 内网服务	172.19.100.11 172.19.100.41	Mule	仅提供内部业务系统间调用使用
ESB 外网服务	172.19.100.31 172.19.100.61	Mule/tomcat	仅提供内部系统调用外网接口服务
ESB 公网服务	172.20.102.11 172.20.102.21	Mule/tomcat	仅提供外部系统（第三方合作公司）调用公司内部系统接口服务
ESB 移动服务	172.19.100.48 172.19.100.49	Mule	仅提供理财 APP、还款易 APP 调用业务系统接口的服务
ESB 办公服务	待定	待定	仅提供办公类系统（OA/KK/HR/财务）间接口调用的服务

9.2 服务规范

1、 ESB 内网服务仅提供内部业务系统间的接口调用的服务。

2、 ESB 外网服务仅提供内部系统调用外网(外部第三方)接口的服务。

3、 ESB 公网服务仅提供外部系统（第三方合作公司）调用公司内部系统接口的服务。

4、 ESB 移动服务仅提供理财 APP/还款易 APP 调用业务系统接口的服务。

5、 ESB 办公服务仅提供办公类系统（OA/KK/HR/财务）间接口调用的服务。

9.3 使用规范

1、 明确要求系统间的接口调用必须通过 ESB 服务。
2、 明确要求业务系统统一使用“参数配置”管理 ESB 接口调用地址。
3、 明确要求 ESB 接口服务的开发须发送邮件(描述接口服务内容/调用方/服务方)至 ESB 系统负责人。
4、 ESB 接口研发人员须先进行接口登记，再进行研发/升级等后续处理。
5、 ESB 接口升级时，须邮件明确告知接口调用地址。

第10章　子系统接口服务规范

10.1 服务协议

1、定义规范系统间接口服务均通过 ESB-mule 方式进行通信。
2、定义规范子系统间接口服务均通过 dubbo 方式进行通信。

10.2 服务方规范

1、接口服务方提供子系统间调用接口服务文档及 demo 实例。
2、针对有业务数据变更的接口服务，接口服务方必须实现接口幂等或去重逻辑防止调用方重复调用。
3、接口服务方须考虑接口容错能力设计。
4、接口服务方事务控制由 spring-base.xml 统一提供支持。
5、接口服务方服务通过增加 dubbo-provider-spring.xml 方式注册接口服务。
具体参考：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://code.alibabatech.com/schema/dubbo
        http://code.alibabatech.com/schema/dubbo/dubbo.xsd ">
    <!-- 提供方应用信息，用于计算依赖关系 -->
    <dubbo:application name="cus_provider"  />
    <!-- 使用multicast广播注册中心暴露服务地址
    <dubbo:registry address="multicast://localhost:1234" />-->
    <!-- 使用zookeeper注册中心暴露服务地址 -->
    <dubbo:registry address="zookeeper://192.168.64.7:2181" />
    <!--  <dubbo:registry  address="zookeeper://zk1:2181?backup=zk2:2181,zk3:2181"
/> -->
    <!-- 用dubbo协议在20880端口暴露服务 -->
    <dubbo:protocol name="dubbo" port="20880" />
    <!-- 声明需要暴露的服务接口 version="1.0.0"-->
    <dubbo:service          interface="com.jy.modules.dubbo.api.ICustomerService"
ref="com.jy.modules.befLoan.CustomerService"
     cluster="failfast" loadbalance="random" actives="100" executes="200">

    </dubbo:service>

</beans>
```

6、接口服务方须定义 API-maven 子项目，且 API-maven 子项目中不应包含其他第三方包的引用，此 API-jar 包须提供给接口调用方使用。
具体参考 com-jy-platform-api 工程 POM.xml 定义:

```
<project                                    xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.jy.platform</groupId>
    <artifactId>com-jy-platform</artifactId>
    <version>2.2.0</version>
  </parent>
  <groupId>com.jy.platform</groupId>
  <artifactId>com-jy-platform-api</artifactId>
  <version>2.2.0</version>
  <name>com-jy-platform-api</name>
  <description>com-jy-platform-api</description>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java-version>1.6</java-version>
    </properties>
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.6</source>
          <target>1.6</target>
        </configuration>
      </plugin>
      <plugin>
        <artifactId>maven-javadoc-plugin</artifactId>
        <executions>
            <execution>
                <id>attach-javadoc</id>
                <goals>
                    <goal>jar</goal>
                </goals>
            </execution>
        </executions>
        <configuration>
```

```
            <show>public</show>
            <charset>UTF-8</charset>
            <encoding>UTF-8</encoding>
            <docencoding>UTF-8</docencoding>
            <links>
                <link>http://docs.oracle.com/javase/6/docs/api</link>
            </links>
        </configuration>
    </plugin>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-source-plugin</artifactId>
        <version>2.1.1</version>
        <executions>
            <execution>
                <id>attach-sources</id>
                <goals>
                    <goal>jar-no-fork</goal>
                </goals>
            </execution>
        </executions>
        <configuration>
            <attach>true</attach>
        </configuration>
    </plugin>

    </plugins>

    <resources>
            <resource>
            <directory>src/main/java</directory>
                <excludes>
                    <exclude>**/*.java</exclude>
                    <exclude>**/.svn/*</exclude>
                </excludes>
                </resource>
            <resource>
            <directory>src/main/resources</directory>
                <includes>
                <include>**/*.properties</include>
                <include>**/*.xml</include>
                </includes>
            </resource>
        </resources>
```

```
  </build>
  <dependencies>
    <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <scope>provided</scope>
        </dependency>
  </dependencies>
</project>
```

10.3 调用方规范

1、接口调用方通过增加 dubbo-customer-spring.xml 配置文件引入接口服务。

2、异步接口调用设计规范参考贷款系统异步接口调用实现。后续由平台封装统一异步接口调用。设计思想：将待调用接口系统及业务主键 ID（通过业务主键 ID 可以查询组装接口调用参数）新增保存至异步接口任务轮训表，由定时任务依次唤醒接口服务任务放入异步任务线程池中执行，同时记录异步接口执行状态。如失败则尝试继续执行，3 次连续失败则不再唤醒执行此失败接口服务。

3、接口服务须明确接口调用超时时间、尝试次数（具体需在 dubbo-customer-spring.xml 配置即可）。

4、业务场景须调用实时接口时（调用方有事务控制且服务亦有事务控制），为防止出现接口调用方异常回滚，接口服务方无法事务回滚的场景。须将接口调用逻辑放置业务实现最后方。

5、dubbo-customer-spring.xml 参考：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://code.alibabatech.com/schema/dubbo
        http://code.alibabatech.com/schema/dubbo/dubbo.xsd ">
    <!-- 消费方应用名，用于计算依赖关系，不是匹配条件，不要与提供方一样 -->
    <dubbo:application name="consumer-of-app" />
      <!-- 使用multicast广播注册中心暴露服务地址
    <dubbo:registry address="multicast://localhost:1234" />-->
    <!-- 使用zookeeper注册中心暴露服务地址 -->
    <dubbo:registry                                          protocol="zookeeper"
address="zookeeper://192.168.64.7:2181" />
    <!-- 生成远程服务代理，可以和本地bean一样使用demoService 尝试次数0 超时时间-->
    <dubbo:reference                                              id="demoService"
interface="com.jy.modules.dubbo.api.ICustomerService"                  retries="0"
```

```
timeout="500000"/>
</beans>
```