数据资源中心
软件设计与代码规范

正式版本 1.2.4

2018 年 08 月 02 日

目　录

文档修订记录

序号	版本号	完成时间	修改人	审核人	备注
1	V1.0	2015-9-23	黄奇	程旭	创建。
2	V1.1	2015-9-24	程旭		增加了技术路线与应用范围等内容。
3	V1.2	2017-4-9	任女尔		调整了技术选型
4	V1.2.3	2018-7-31	程旭		增加了对Java文件注释的要求说明。
5	V1.2.4	2018-08-02	蔡建军		修改了Java文件注释率的要求及对常量定义的要求
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					

序言

制定此次规范的主要目的，是为增强数据资源中心诸多软件项目中代码的易读性，易维护性，从而提升代码的质量。随着部门业务的壮大，我们 IT 系统的软件规模也与日俱增，现在的 J2EE 应用程序，使用 Eclipse 看其源码，打开后往往是黑压压的一片，少则百千，多则上万，可谓“黑云压城城欲摧”，这给后期要做修改维护的人造成极大的心理压力。

其实，任何代码并不怕多，而是怕乱。乱则无序，无序则混沌，毫无规律可循，而或小桥流水，忽又天马行空，渺渺乎不得其所始，飘飘乎无寻其所终，维护代码的人如果要找某一功能段落，往往是大海捞针，只有天知道.

所以，编写代码时有一定的规范参照，一定程度上能使开发层面上的工作标准化，这种标准化自然给后续的易读性带来好处，好比一本汉语字典，即使里面内容再多再杂，我们也可以通过拼音或者是部首轻松定位到某个具体汉字。这种代码的易读性也是软件质量的重要标志。

是故，日月有先后而成昼夜，音律有高低而为乐章，木工取规矩而形圆方，代码效层次样标故得益彰，余自整理七年来代码开发中所遇所见，并参照 CMM 模型理念，初拟一套编码规范，以下就此从命名规范，文件格式，编码风格，编码约定（纠正习惯）等方面进行阐述。

1. 技术路线

Java EE 项目的技术路线与选型，后期项目在此技术选型内开展。具体说明如下。

1.1 开发环境

Java JDK 1.7 及以上。

1.2 整体框架

新项目：基于 SpringBoot 的 SpringMVC+Spring+SpringData+Hibernate。

权限控件： shiro 、SpringSecurity 4.1.2

Druid

1.3 源代码管理

Apache Subversion。

服务器端使用 VisualSVN Server，客户端推荐使用 TortoiseSVN。

新项目推荐 Git，对于分支管理更好一些，代码库共享采用 Gitlab 平台。

1.4 流程引擎

⑴ JBPM 4；

⑵ Activiti 5。新项目统一 Activiti5。

1.5 脚本库

⑴ JQuery 1.12.4

1.6 UI 库

(1) bootstrap 2.3.2

1.7 Web Service

(1) CXF 2.3.2

(2) RESTful

1.8 集成开发环境(IDE)

(1) Eclipse Mars4.5

(2) IDEA

1.9 应用服务器

按推荐顺序：

(1) Apache Tomcat 7.0、6.0；

(2) Oracle WebLogic Server 10.3、12.0.1。

1.10 项目管理工具

(1) maven 3.3.3。

2. 命名规范

1. 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。

反例： _name/__name/$Object/name_/name$/Object$

2. 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。

3. 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）EO / BO / DTO / VO等。

正例：MarcoPolo / UserEO / XmlService / TcpUdpDeal / TaPromotion

反例：macroPolo / UseEo / XMLService / TCPUDPDeal / TAPromotion

4. 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。

正例： localValue / getHttpMessage() / inputUserId

5. 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。

正例： MAX_STOCK_COUNT

反例： MAX_COUNT

6. 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。

7. 中括号是数组类型的一部分，数组定义如下：String[] args;

反例：使用String args[]的方式来定义。

8. 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。

反例：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。

9. 包名统一使用小写，点分隔符之间有且仅有一个自然语义的英语单词。包名统一使用单数形式，但是类名如果有复数含义，类名可以使用复数形式。

正例： 应用工具类包名为com.adc.open.util、类名为MessageUtils（此规则参考spring的框架结构）

10. 杜绝完全不规范的缩写，避免望文不知义。

反例： AbstractClass“缩写”命名成AbsClass；condition“缩写”命名成 condi，此类随意缩写严重降低了代码的可阅读性。

11. 接口和实现类的命名有两套规则：

1）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。

正例：CacheServiceImpl实现CacheService接口。

12.【参考】各层命名规约：

A) Service/DAO层方法命名规约

1） 获取单个对象的方法用get做前缀。

2） 获取多个对象的方法用list做前缀。

3） 获取统计值的方法用count做前缀。

4） 插入的方法用save（推荐）或insert做前缀。

5） 删除的方法用remove（推荐）或delete做前缀。

6） 修改的方法用update做前缀。

B) 领域模型命名规约

1） 数据对象：xxxEO，xxx即为数据表名。

2） 数据传输对象：xxxDTO，xxx为业务领域相关的名称。

3） 展示对象：xxxVO，xxx一般为网页名称。

4） POJO是EO/DTO/BO/VO的统称，禁止命名成xxxPOJO。

3. Java 文件的格式

所有的 Java(*.java)文件都必须遵守如下的样式规则。

1. 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。

说明：在IDE编辑窗口中，Javadoc方式会提示相关注释，生成Javadoc可以正确输出相应注释；在IDE中，工程调用方法时，不进入方法即可悬浮提示方法、参数、返回值的意义，提高阅读效率。

2. 所有的抽象方法（包括接口中的方法）必须要用Javadoc注释、除了返回值、参数、异常说明外，还必须指出该方法做什么事情，实现什么功能。

说明：对子类的实现要求，或者调用注意事项，请一并说明。

3. 所有的类都必须添加创建者和创建日期。

配合 SVN，Eclipse 的使用，所有 Java 文件的头部固定为：

```
/*
   *Lastmodifiedby$Author$on$Date$(UTC)
   *Revision:$Revision$
   */
```

Author，Date 处写明作者和日期。

4. 方法内部单行注释，在被注释语句上方另起一行，使用//注释。方法内部多行注释使用/* */注释，注意与代码对齐。

5. 所有的枚举类型字段必须要有注释，说明每个数据项的用途。

6. Imports

避免import整个package而造成的浪费，而必须import具体的类。如：

```
import java.util.*;//犯规！
import java.util.Observable;//可以用。
```

7. ClassFields

所有的静态变量和成员变量出现在类定义之后。一般按照常量、静态变量、成员变量的顺序定义。

main方法：如果有main(String[])方法,那么它应该写在类的底部。

8. 与其“半吊子”英文来注释，不如用中文注释把问题说清楚。专有名词与关键字保持英文原文即可。

反例：“TCP 连接超时”解释成“传输控制协议连接超时”，理解反而费脑筋。

4. 编码风格

使用 Eclipse 的程序员建议使用 format 功能优化源代码的格式。

1. 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：

1） 左大括号前不换行。

2） 左大括号后换行。

3） 右大括号前换行。

4） 右大括号后还有 else 等代码则不换行；表示终止的右大括号后必须换行。

2. 左小括号和右边相邻字符之间不出现空格；同样，右小括号和左边相邻字符之间也不出现空格。详见第 5 条下方正例提示。

3. if/for/while/switch/do 等保留字与小括号之间都必须加空格。

4. 任何运算符左右必须加一个空格。 说明：运算符包括赋值运算符=、逻辑运算符&&、加减乘除符号、三目运算符等。

5. 缩进采用 4 个空格，禁止使用 tab 字符。

说明：如果使用 tab 缩进，必须设置缩进，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；而在 eclipse 中，必须勾选 insert spaces for tabs。

6. 单行字符数限不超过 120 个，超出需要换行时遵循如下原则：

1） 第二行相对一缩进 4 个空格，从第三行开始不再继续缩进参考示例。

2） 运算符与下文一起换行。

3） 方法调用的点符号与下文一起换行。

4） 在多个参数超长，逗号后进行换行。

5） 在括号前不要换行，见反例。

正例：

```
StringBuffer sb = new StringBuffer();
//超过 120 个字符的情况下，换行缩进 4 个空格，并且方法前的点符号一起换行
sb.append("zi").append("xin")...
```

.append("huang")...

.append("huang")...

.append("huang");

反例：

StringBuffer sb = new StringBuffer();

//超过 120 个字符的情况下，不要在括号前换行

sb.append("zi").append("xin")...append

("huang");

//参数很多的方法调用可能超过 120 个字符，不要在逗号前换行

method(args1, args2, args3, ...

, argsX);

7. 方法参数在定义和传入时，多个参数逗号后边必须加空格。 正例：下例中实参的"a",后边必须要有一个空格。

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

8. 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。

5. 编码约定

5.1 exit()

exit 除了在 main 中可以被调用外，其他的地方不应该调用。因为这样做不给任何代码代码机会来截获退出。一个类似后台服务地程序不应该因为某一个库模块决定了要退出就退出。

5.2 对类成员的访问

一般成员变量应该定义为 private。子类也应该通过方法来访问。

5.3 方法的复杂性

应避免单一方法过于复杂。如果方法实现主要由简单的赋值语句和方法调用构成，长度不应超过 50 行。如果由分支和循环，建议不超过 30 行。

5.4 常量定义

1. 尽量不要出现任何魔法值（即未经定义的常量）直接出现在代码中。若需直接使用常量则必须加上注释说明含义。

反例： String key = "Id#adc_"+tradeId； cache.put(key, value);

2. long或者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。

说明：Long a = 2l; 写的是数字的 21，还是 Long 型的 2?

5.5 OOP 规约

1. 避免通过一个类的对象引用访问此类的静态变量或静态方法，无谓增加编译器解析成本，直接用类名来访问即可。

2. 所有的覆写方法，必须加@Override注解。

反例：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。

3. 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） 正例：public User getUsers(String type, Integer... ids) {...}

4. 外部正在调用或者二方库依赖的接口，不允许修改方法签名，避免对接口调用方产生影响。接口过时必须加@Deprecated注解，并清晰地说明采用的新接口或者新服务是什么。

5. 不能使用过时的类或方法。

说明：java.net.URLDecoder 中的方法decode(String encodeStr) 这个方法已经过时，应该使用双参数decode(String source, String encode)。接口提供方既然明确是过时接口，那么有义务同时提供新的接口；作为调用方来说，有义务去考证过时方法的新实现是什么。

6. 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。

正例： "test".equals(object);

反例： object.equals("test");

说明：推荐使用java.util.Objects#equals （JDK7引入的工具类）

7. 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。

8. 关于基本数据类型与包装数据类型的使用标准如下：

1） 所有的POJO类属性必须使用包装数据类型。

2） RPC方法的返回值和参数必须使用包装数据类型。

9. 定义EO/DTO/VO等POJO类时，不要设定任何属性默认值。

反例：POJO类的gmtCreate默认值为new Date();但是这个属性在数据提取时并没

有置入具体值，在更新其它字段时又附带更新了此字段，导致创建时间被修改成当前时间。

10. 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。

说明：注意serialVersionUID不一致会抛出序列化运行时异常。

11. 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。

12. POJO类必须写toString方法。

使用IDE的中工具：source> generate toString时，如果继承了另一个POJO类，注意在前面加一下super.toString。

说明：在方法执行抛出异常时，可以直接调用POJO的toString()方法打印其属性值，便于排查问题。

5.6 其它

1. 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。

说明：不要在方法体内定义：Pattern pattern = Pattern.compile(规则);

2. velocity调用POJO类的属性时，建议直接使用属性名取值即可，模板引擎会自动按规范调用POJO的getXxx()，如果是boolean基本数据类型变量（boolean命名不需要加is前缀），会自动调用isXxx()方法。 说明：注意如果是Boolean包装类对象，优先调用getXxx()的方法。

3. 后台输送给页面的变量必须加$!{var}——中间的感叹号。 说明：如果var=null或者不存在，那么${var}会直接显示在页面上。

4. 注意 Math.random() 这个方法返回是double类型，注意取值的范围 0≤x<1（能够取到零值，注意除零异常），如果想获取整数类型的随机数，不要将x放大10的若干倍然后取整，直接使用Random对象的nextInt或者nextLong方法。

5. 获取当前毫秒数System.currentTimeMillis(); 而不是new Date().getTime(); 说明：如果想获取更加精确的纳秒级时间值，使用System.nanoTime()的方式。在JDK8中，针对统计时间等场景，推荐使用Instant类。

6. 注释率要求

6.1 名词解释

代码注释率 = 注释行数/(有效代码行数+注释行数)

有效代码行数是指不包含注释行和空白行的行数

6.2 注释率要求

对于提交至 SVN 代码服务器中的项目代码，注释率原则上不少于 23%，即 10 行代码至少得写 3 行注释。

6.3 计算说明

6.3.1 对于 Java 文件

```
1  package com.adc.da;
2  import java.util.concurrent.TimeUnit;
3  import org.mybatis.spring.annotation.MapperScan;
4  import org.springframework.boot.SpringApplication;
5  import org.springframework.boot.autoconfigure.SpringBootApplication;
6  import org.springframework.boot.context.embedded.ConfigurableEmbeddedServletContainer;
7  import org.springframework.boot.context.embedded.EmbeddedServletContainerCustomizer;
8  import org.springframework.boot.web.servlet.ServletComponentScan;
9  import org.springframework.context.ApplicationContext;
10 import org.springframework.context.annotation.Bean;
11 //import com.codahale.metrics.CsvReporter;
12 @SpringBootApplication
13 @ServletComponentScan
14 public class AdcDaApplication {
15     public static void main(String[] args) {
16
17         ApplicationContext applicationContext = SpringApplication.run(AdcDaApplication.class, args);
18         // 启动Metrics 性能监控报表
19 //      CsvReporter reporter = applicationContext.getBean(CsvReporter.class);
20 //      reporter.start(1, TimeUnit.MINUTES);
21     }
22     /**
23      * 设定容器的session失效时间，默认30分
24      *
25      */
26     @Bean
27     public EmbeddedServletContainerCustomizer containerCustomizer(){
28         return new EmbeddedServletContainerCustomizer() {
29             @Override
30             public void customize(ConfigurableEmbeddedServletContainer container) {
31                 container.setSessionTimeout(1800);//单位为s
32             }
33         };
34     }
35 }
```

共6行注释 26行有效代码(不算空行和注释行)

注释率=6/(6+26)=18.8%

此处只算1行注释

6.3.2 对于 CSS 文件

```
/**
*    补充css

*/

.systable{
    min-width: 700px;
}
```

此种情况算 4 行注释。

```
/*  内容项目 */
@media screen and  (max-width: 1366px){
    #mainContents{ min-height: 350px; }  /* non-theme */
}
```

此种情况算 2 行注释，具体示例如下图：

```
/**
*    补充css

*/

.systable{
    min-width: 700px;
}

/*  内容项目 */
@media screen and  (max-width: 1366px){
    #mainContents{ min-height: 350px; }  /* non-theme */
}

@media screen and (min-width: 1366px) and (max-width: 1920px){
    #mainContents{ min-height: 400px; }
}
@media screen and (min-width: 1920px) {
    #mainContents{ min-height: 650px; }
}
```

6行注释 12行有效代码(不含空行和注释)

注释率=6/(6+12)=33.3%

6.3.3 对于 XML 文件

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <!--
        全局参数 -->
    <settings>
        <setting name="callSettersOnNulls" value="true" />
    </settings>
    <!-- 配置别名用  -->
    <typeAliases>

    </typeAliases>
</configuration>
```

此处只算1行注释

注释行数2行，有效行数10行
注释率=2/(2+10)=16.7%

6.3.4 对于 HTML 文件

注释行数只统计 Html 部分的，不统计 JavaScript 和 Css 中的。

```html
<div class="portlet  ">
        <div class="portlet-title ">
            <!--
                列表 头部信息-->
            <div class="caption"><i class="icon-reorder"></i>我的待发事项</div>
        </div>
        <!--<div class="tools tabbletools">-->
            <!--<a id="userAdd" href="javascript:;" class="plus"></a>-->
            <!--<a id="userUpdate" href="javascript:;" class="config"></a>-->
            <!--<a id="userDelete" href="javascript:;" class="delete"></a>-->
        <!--</div>-->
</div>
```

此处算1行注释

共6行注释，有效行数5行
注释率=6/(6+5)=54.5%

6.3.5 对于 JS 文件

```javascript
/*
*  国际化
* */
/*
$('#language li').on('click',function () {
    language =$(this).attr('lan');
    $.cookie('lan',language);
    $("#language-button span").html(language);
});
```

此处注释算1行

此种情况算 2 行注释。

```
var list = {
    url: rurl, //+dictionaryCode,
    data:{
        pageNo : pageNo,
        pageSize : pageSize,
        dictionaryCode : dictionaryCode
    },
    delay: 250,
    allowClear: true,
    placeholder: {
        id: '-1', // the value of the option
        text: '请选择1'
    },
```

7. 集合类的使用

7.1 集合类的选择

方法的参数和返回值如果是集合的话，按以下顺序选择声明的类型：

1.集合中元素类型的数组。比如集合的元素是 Account 类型，则使用：

Account[]

2.根据集合的特性选择：List（顺序、元素可重复），Set(无顺序、元素不可重复)，SortedSet(有顺序、元素不可重复)，Map，Properties（键、值均为字符串），Collection（无任何限制）。

3.具体的 Collection 类，如 ArrayList，HashMap 等。

7.2 ArrayList、HashMap 与 Vector、HashTable

在没有并发访问问题的时候使用 ArrayList 和 HashMap，不使用 Vector 和 HashTable。

1. 关于hashCode和equals的处理，遵循如下规则：

1） 只要重写equals，就必须重写hashCode。

2） 因为Set存储的是不重复的对象，依据hashCode和equals进行判断，所以Set存储的对象必须重写这两个方法。

3） 如果自定义对象做为Map的键，那么必须重写hashCode和equals。

说明：String重写了hashCode和equals方法，所以我们可以非常愉快地使用String对象作为key来使用。

2. ArrayList的subList结果不可强转成ArrayList，否则会抛出classCastException异常：java.util.RandomAccessSubList cannot be cast to java.util.ArrayList ;

说明：subList 返回的是 ArrayList 的内部类 SubList，并不是 ArrayList ，而是 ArrayList 的一个视图，对于SubList子列表的所有操作最终会反映到原列表上。

3. 在subList场景中，高度注意对原集合元素个数的修改，会导致子列表的

遍历、增加、删除均产生ConcurrentModificationException 异常。

4. 使用集合转数组的方法，必须使用集合的toArray(T[] array)，传入的是类型完全一样的数组，大小就是list.size()。

说明：使用toArray带参方法，入参分配的数组空间不够大时，toArray方法内部将重新分配内存空间，并返回新数组地址；如果数组元素大于实际所需，下标为[list.size()]的数组元素将被置为null，其它数组元素保持原值，因此最好将方法入参数组大小定义与集合元素个数一致。

正例：

```
List<String> list = new ArrayList<String>(2);
list.add("guan");
list.add("bao");
String[] array = new String[list.size()];
array = list.toArray(array);
```

反例：直接使用toArray无参方法存在问题，此方法返回值只能是Object[]类，若强转其它类型数组将出现ClassCastException错误。

5. 使用工具类Arrays.asList()把数组转换成集合时，不能使用其修改集合相关的方法，它的add/remove/clear方法会抛出UnsupportedOperationException异常。

说明：asList的返回对象是一个Arrays内部类，并没有实现集合的修改方法。Arrays.asList体现的是适配器模式，只是转换接口，后台的数据仍是数组。String[] str = new String[] { "a", "b" }; List list = Arrays.asList(str);

第一种情况：list.add("c"); 运行时异常。 第二种情况：str[0] = "gujin"; 那么list.get(0)也会随之修改。

6. 泛型通配符<? extends T>来接收返回的数据，此写法的泛型集合不能使用add方法，而<? super T>不能使用get方法，做为接口调用赋值时易出错。 说明：扩展说一下PECS(Producer Extends Consumer Super)原则：

1）频繁往外读取内容的，适合用上界Extends。

2）经常往里插入的，适合用下界Super。

7. 不要在foreach循环里进行元素的remove/add操作。remove元素请使用Iterator方式，如果并发操作，需要对Iterator对象加锁。

反例：

```
List<String> a = new ArrayList<String>();
a.add("1");
a.add("2");
for (String temp : a) {
    if ("1".equals(temp)) {
        a.remove(temp);
    }
}
```

说明：以上代码的执行结果肯定会出乎大家的意料，那么试一下把“1”换成“2”，会是同样的结果吗？ 正例：

```
Iterator<String> it = a.iterator();
while (it.hasNext()) {
    String temp = it.next();
    if (删除元素的条件) {
        it.remove();
    }
}
```

8. 在JDK7版本及以上，Comparator要满足如下三个条件，不然Arrays.sort，Collections.sort会报IllegalArgumentException异常。

说明： 1） x，y的比较结果和y，x的比较结果相反。

2） x>y，y>z，则x>z。

3） x=y，则x，z比较结果和y，z比较结果相同。

反例：下例中没有处理相等的情况，实际使用中可能会出现异常：

```
new Comparator<Student>() {
    @Override
    public int compare(Student o1, Student o2) {
        return o1.getId() > o2.getId() ? 1 : -1;
    }
};
```

8. 控制语句

1. 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。

2. 在if/else/for/while/do语句中必须使用大括号。即使只有一行代码，避免使用单行的形式：if (condition) statements;

3. 【推荐】表达异常的分支时，少用if-else方式，这种方式可以改写成：

```
if (condition) {
    ...
    return obj;
}
// 接着写else的业务逻辑代码;
```

说明：如果非得使用if()...else if()...else...方式表达逻辑，请勿超过3层，超过请使用状态设计模式。

正例：逻辑上超过 3 层的 if-else 代码可以使用卫语句，或者状态模式来实现。

9. 日志

记录日志的目的是记录系统运行的状态、为系统的维护和故障诊断、分析提供信息。

不论是自己用程序写日志还是用 CommongLogging 提供的 API 使用日志功能。都应在日志中记录的内容及其使用级别：

(1) fatal:严重的，导致应用的全部，或一大模块无法正常运行的错误。比如在启动和初始话的过程中的错误，导致应用系统无法正常运行。

(2) error:导致系统的一个功能，用户的一次请求无法执行的错误。

(3) warn:可能的错误或异常情况，但系统功能，或用户请求仍能继续执行。

(4) info:反映系统运行状态、流程的重要信息，供系统管理员判断系统是否正常运行。

(5) debug:主要用于供程序员调试程序而用。

(6) trace:关于程序运行流程的最详细的信息。比如函数的调用参数、返回值，程序分支的执行路径等。前三个级别应该同时输出到控制台和日志文件。后两个级别只记入日志文件。

在源代码中，同时应该有专门的方法来标志日志功能

如下所示：

isFatalEnabled()

isErrorEnabled()

isWarnEnabled()

isInfoEnabled()

isDebugEnabled()

isTraceEnabled()

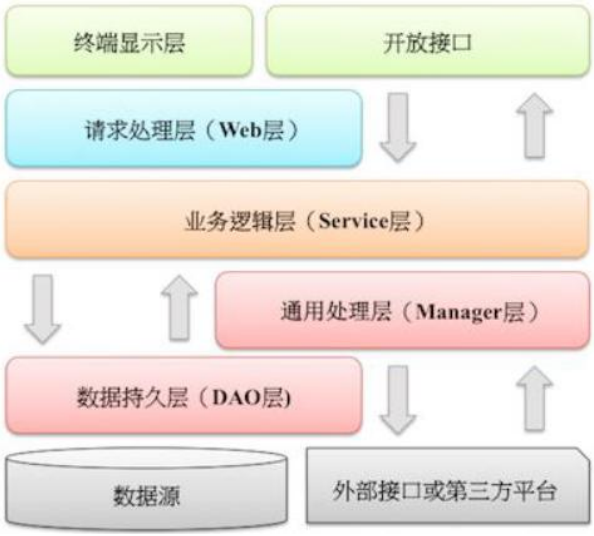
如果需要为某特定目的记录日志信息，比如记录与某外部系统的通信过程中的日志，可以建立专门的 Logcategory，如：

com.adc.sap.communication。然后将此 category 下的日志记入专门的日志文件。

10. 工程级规约

10.1 分层规约

图中默认上层依赖于下层，箭头关系表示可直接依赖，如：开放接口层可以依赖于Web层，也可以直接依赖于Service层，依此类推：



开放接口层：可直接封装Service方法暴露成RPC接口；通过Web封装成http接口；进行网关安全控制、流量控制等。

终端显示层：各个端的模板渲染并执行显示的层。当前主要是velocity渲染，JS渲染，JSP渲染，移动端展示等。

Rest/Controller层：主要是对访问控制进行转发，各类基本参数校验，或者不复用的业务简单处理等。

Service层：相对具体的业务逻辑服务层。

Manager层：通用业务处理层，它有如下特征：

1） 对第三方平台封装的层，预处理返回结果及转化异常信息；

2） 对Service层通用能力的下沉，如缓存方案、中间件通用处理；

3）与DAO层交互，对多个DAO的组合复用。

DAO层：数据访问层，与底层MySQL、Oracle、Hbase进行数据交互。

外部接口或第三方平台：包括其它部门RPC开放接口，基础平台，其它公司的HTTP接口。

10.2 二方库规约

1. 定义GAV遵从以下规则：

1） GroupID格式：com.adc.业务线.[子业务线]，最多4级。

正例：com.adc.jstorm 或 com.adc.dubbo.register

2） ArtifactID格式：产品线名-模块名。语义不重复不遗漏，先到仓库中心去查证一下。

正例：dubbo-client / fastjson-api / jstorm-tool

3） Version：x.x.x三级版本号。

2.版本号命名方式：主版本号.次版本号.修订号

1） 主版本号：当做了不兼容的API 修改，或者增加了能改变产品方向的新功能。

2）次版本号：当做了向下兼容的功能性新增（新增类、接口等）。

3）修订号：修复bug，没有修改方法签名的功能加强，保持 API 兼容性。

说明： 注意：起始版本号必须为：1.0.0，而不是0.0.1正式发布的类库必须先去中央仓库进行查证，使版本号要有延续性，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

3. 线上应用不要依赖SNAPSHOT版本（安全包除外）。

说明：不依赖SNAPSHOT版本是保证应用发布的幂等性。另外，也可以加快编译时的打包构建。

4. 二方库的新增或升级，保持除功能点之外的其它jar包仲裁结果不变。如果有改变，必须明确评估和验证，建议进行dependency:resolve前后信息比对，如果仲裁结果完全不一致，那么通过dependency:tree命令，找出差异点，进行<excludes>排除jar包。

5. 二方库里可以定义枚举类型，参数可以使用枚举类型，但是接口返回值不允许使用枚举类型或者包含枚举类型的POJO对象。

6.依赖于一个二方库群时，必须定义一个统一的版本变量，避免版本号不一致。

说明：依赖springframework-core,-context,-beans，它们都是同一个版本，可以定义一个变量来保存版本：${spring.version}，定义依赖的时候，引用该版本。

7.禁止在子项目的pom依赖中出现相同的GroupId，相同的ArtifactId，但是不同的

Version。

说明：在本地调试时会使用各子项目指定的版本号，但是合并成一个 war，只能有一个版本号出现在最后的 lib 目录中。可能出现线下调试是正确的，发布到线上却出故障的问题。

11. 数据库逻辑设计

11.1 命名规范

11.1.1 语言

命名应该使用英文单词，避免使用拼音，特别不应该使用拼音简写。命名不允许使用中文或者特殊字符。

英文单词使用用对象本身意义相对或相近的单词。选择最简单或最通用的单词。不能使用毫不相干的单词来命名

当一个单词不能表达对象含义时，用词组组合，如果组合太长时，采用用简或缩写，缩写要基本能表达原单词的意义。

当出现对象名重名时，是不同类型对象时，加类型前缀或后缀以示区别。

11.1.2 大小写

名称一律大写，以方便不同数据库移植，以及避免程序调用问题。

11.1.3 单词分隔

命名的各单词之间可以使用下划线进行分隔。

11.1.4 保留字

命名不允许使用 SQL 保留字。

11.1.5 命名长度

表名、字段名、视图名长度应限制在 30 个字符内(含前缀)。

11.1.6 字段名称

同一个字段名在一个数据库中只能代表一个意思。比如 telephone 在一个表中代表“电话号码”的意思，在另外一个表中就不能代表“手机号码”的意思。

不同的表用于相同内容的字段应该采用同样的名称，字段类型定义。

11.1.7 表

✧ 表名

所有表名以 T 开头，以下划线“_”作为单词间分隔符，表名不能用双引号包含，表命名分为如下几大类：

TC_：通用代码表，例如 TC_REC_TYPE 代表所有数据库表记录类型
TG_：与外围系统交互的中间表，例如 TG_SAP_CONTRACT 代表来自 SAP 系统的表
TL_：日志表，各种日志信息记录，例如 TL_IT_LOG 记录用户访问信息记录
TM_：企业内共享的主数据表，例如 TM_PART 代表 零件清单
TR_：关联中间表 TR_USER_ROLE 记录用户和角色之间的关系
TS_：系统表，例如 TS_MENU 菜单信息 TS_USER 用户信息
TT_：业务数据表，TT_RFQ 询报价记录
TE_：临时表，仅在导入数据或生成报表时使用

✧ 表分区名

前缀为 ***P***。分区名必须有特定含义的单词或字串。

例如 ：tbl_pstn_detail 的分区 P04100101 表示该分区存储 2004100101 时段的数据。

✧ 字段名

字段名称必须用字母开头，采用有特征含义的单词或缩写，不能用双引号包含。

✧ 主键名

前缀为 ***PK_***。主键名称应是 前缀+表名+构成的字段名。如果复合主键的构成字段较多，则只包含第一个字段。表名可以去掉前缀。

✧ 外键名

前缀为 ***FK_***。外键名称应是 前缀+ 外键表名 + 主键表名 + 外键表构成的字段名。表名可以去掉前缀。

11.1.8 索引

✧ 普通索引

前缀为 ***IDX_***。索引名称应是 前缀+表名+构成的字段名。如果复合索引的构成字段较多，则只包含第一个字段，并添加序号。表名可以去掉前缀。

✧ 主键索引

前缀为 ***IDX_PK_***。索引名称应是 前缀+表名+构成的主键字段名，在创建表时候用 using index 指定主键索引属性。

✧ 唯一索引

前缀为 ***IDX_UK_***。索引名称应是 前缀+表名+构成的字段名。

✧ 外键索引

前缀为 ***IDX_FK_***。索引名称应是 前缀+表名+构成的外键字段名。

✧ 函数索引

前缀为 IDX***_func_***。索引名称应是 前缀+表名+构成的特征表达字符。

✧ 蔟索引

前缀为 ***IDX_clu_***。索引名称应是 前缀+表名+构成的簇字段。

11.1.9 视图

前缀为 ***V_***。按业务操作命名视图。

11.1.10 实体化视图

前缀为 ***MV_***。按业务操作命名实体化视图。

11.1.11 存储过程

前缀为 ***Proc_*** 。按业务操作命名存储过程

11.1.12 触发器

前缀为 ***Trig_*** 。触发器名应是 前缀 + 表名 + 触发器名。

11.1.13 函数

前缀为 ***Func_*** 。按业务操作命名函数

11.1.14 数据包

前缀为 ***Pkg_*** 。按业务操作集合命名数据包。

11.1.15 序列

前缀为 ***Seq_*** 。按业务属性命名。

11.1.16 表空间

✧ 公用表空间

前缀为 ***Tbs_***。根据应用名字命名，例如：tbs_<缩写的应用名>。

✧ 专用表空间

Tbs_<缩写的应用名>_xxx。xxx 可以是 Index, BLOB/CLOB 属性, Aux 其它辅助信息。

11.1.17 数据文件

<表空间名>nn.dbf 。nn =1，2，3，4，…等。

11.1.18 普通变量

前缀为 ***Var_*** 。 存放字符、数字、日期型变量。

11.1.19 游标变量

前缀为 ***Cur_*** 。存放游标记录集。

11.1.20 记录型变量

前缀为 ***Rec_*** 。 存放记录型数据。

11.1.21 表类型变量

前缀为 ***Tab_*** 。 存放表类型数据。

11.1.22 数据库链接

前缀为 ***dbl_*** 。 表示分布式数据库外部链接关系。从安全和管理角度出发，不建议使用数据库链接。

11.2 数据类型

11.2.1 字符型

固定长度的字串类型采用 char，长度不固定的字串类型采用 varchar。避免在长度不固定的情况下采用 char 类型。如果在数据迁移等出现以上情况，则必须使用 trim()函数截去字串后的空格。

11.2.2 数字型

数字型字段尽量采用 number 类型。

对于数字型唯一键值，尽可能用系列 sequence 产生。

11.2.3 日期和时间

✧ 系统时间

由数据库产生的系统时间首选数据库的日期型，如 DATE 类型。

✧ 外部时间

由数据导入或外部应用程序产生的日期时间类型采用 varchar 类型，数据格式采用：YYYYMMDDHH24MISS。

11.2.4 大字段

如无特别需要，避免使用大字段(blob，clob，long，text，image 等)。

11.3 设计

11.3.1 范式

如果不是性能上的需要，应该使用关系数据库理论，达到较高的范式，避免数据冗余。但是如果在数据量上与性能上无特别要求，考虑到实现的方便性可以有适当的数据冗余，但基本上要达到 3NF。

如非确实必要，避免一个字段中存储多个标志的做法。如 11101 表示 5 个标志的一种取值。这往往是增加复杂度，降低性能的地方。

11.3.2 特殊表设计原则

✧ 分区表

对于数据量比较大的表，根据表数据的属性进行分区，以得到较好的性能。如果表按某些字段进行增长，则采用按字段值范围进行范围分区；如果表按某个字段的几个关键值进行分布，则采用列表分区；对于静态表，则采用 hash 分区或列表分区；在范围分区中，如果数据按某关键字段均衡分布，则采用子分区的复合分区方法。

✧ 聚簇表

如果某几个静态表关系比较密切，则可以采用聚族表的方法。

11.3.3 完整性设计原则

✧ 主键约束

关联表的父表要求有主健，主健字段或组合字段必须满足非空属性和唯一性要求。对于数据量比较大的父表，要求指定索引段。

✧ 外键关联

对于关联两个表的字段，一般应该分别建立主键、外键。实际是否建立外键，根据对数据完整性的要求决定。为了提高性能，对于数据量比较大的标要求对外健建立索引。对于有要求级联删除属性的外键，必须指定 on delete cascade。

✧ NULL 值

对于字段能否 null，应该在 sql 建表脚本中明确指明，不应使用缺省。由于 NULL 值在参加任何运算中，结果均为 NULL。所以在应用程序中必须利用 nvl() 函数把可能为 NULL 值得字段或变量转换为非 NULL 的默认值。例如：NVL（sale,0）。

✧ Check 条件

对于字段有检查性约束，要求指定 check 规则。

✧ 触发器

触发器是一种特殊的存储过程，通过数据表的 DML 操作而触发执行，起作

用是为确保数据的完整性和一致性不被破坏而创建，实现数据的完整约束。

触发器的 before 或 after 事务属性的选择时候，对表操作的事务属性必须与应用程序事务属性保持一致，以避免死锁发生。

在大型导入表中，尽量避免使用触发器。

✧ 注释

表、字段等应该有中文名称注释，以及需要说明的内容。

11.3.4 索引设计

对于查询中需要作为查询条件的字段，可以考虑建立索引。最终根据性能的需要决定是否建立索引。最常用的 SQL 查询应用大量数据来分析执行计划。

对于复合索引，索引字段顺序比较关键，把查询频率比较高的字段排在索引组合的最前面。

在分区表中，尽量采用 local 分区索引以方便分区维护。

应该为索引创建单独的表空间。除非是分区 local 索引，否则在创建索引时候必须指定索引的 tablespace、storage 属性。

11.3.5 视图设计

视图是虚拟的数据库表，在使用时要遵循以下原则：

✧ 从一个或多个库表中查询部分数据项；

✧ 为简化查询，将复杂的检索或字查询通过视图实现；

✧ 提高数据的安全性，只将需要查看的数据信息显示给权限有限的人员；

✧ 视图中如果嵌套使用视图，级数不得超过 3 级；

✧ 由于视图中只能固定条件或没有条件，所以对于数据量较大或随时间的推移逐渐增多的库表，不宜使用视图，以采用实体化视图代替；

✧ 除特殊需要，避免类似 Select * from [TableName] 而没有检索条件的视图；

✧ 视图中尽量避免出现数据排序的 SQL 语句。

11.3.6 包设计

存储过程、函数、外部游标必须在指定的数据包对象 PACKAGE 中实现。存储过程、函数的建立如同其它语言形式的编程过程，适合采用模块化设计方法；当具体算法改变时，只需要修改需要存储过程即可，不需要修改其它语言的源程序。当和数据库频繁交换数据是通过存储过程可以提高运行速度，由于只有被授权的用户才能执行存储过程，所以存储过程有利于提高系统的安全性。

存储过程、函数必须检索数据库表记录或数据库其他对象，甚至修改（执行 Insert、Delete、Update、Drop、Create 等操作）数据库信息。如果某项功能不需要和数据库打交道，则不得通过数据库存储过程或函数的方式实现。

在函数中避免采用 DML 或 DDL 语句。

在数据包采用存储过程、函数重载的方法，简化数据包设计，提高代码效率。

存储过程、函数必须有相应的出错处理功能。

11.3.7 安全性设计

✧ 数据库级用户权限设计

必须按照应用需求，设计不同的用户访问权限。包括应用系统管理用户，普通用户等，按照业务需求建立不同的应用角色。

用户访问另外的用户对象时，应该通过创建同义词对象 synonym 进行访问。

✧ 角色与权限

确定每个角色对数据库表的操作权限，如创建、检索、更新、删除等。每个角色拥有刚好能够完成任务的权限，不多也不少。在应用时再为用户分配角色，则每个用户的权限等于他所兼角色的权限之和。

✧ 应用级用户设计

应用级的用户帐号密码不能与数据库相同，防止用户直接操作数据库。用户只能用帐号登陆到应用软件，通过应用软件访问数据库，而没有其它途径操作数据库。

✧ 用户密码管理

用户帐号的密码必须进行加密处理，确保在任何地方的查询都不会出现密码

的明文。

11.3.8 API 接口

所有的数据库 API 必须定义出错处理，成功/失败的返回结果。

用 Java 访问数据库，应该使用 JDBC。

用 Perl 访问数据库，应该使用 DBI, DBD::Oracle。

用 C++访问数据库，应该使用 OCCI。

可以用 ODBC 来连接 Oracle 与其它数据库。

从客户端或中间层连接数据库服务器时，使用连接池。

11.4 SQL 编写

11.4.1 字符类型数据

SQL 中的字符类型数据应该统一使用单引号。特别对纯数字的字串，必须用单引号，否则会导致内部转换而引起性能问题或索引失效问题。利用 trim(),lower()等函数格式化匹配条件。

11.4.2 复杂 SQL

对于非常复杂的 sql(特别是有多层嵌套，带子句或相关查询的)，应该先考虑是否设计不当引起的。对于一些复杂 SQL 可以考虑使用程序实现。

11.4.3 高效性

✧ 避免 IN 子句

使用 In 或 not In 子句时，特别是当子句中有多个值时，且查询数据表数据较多时，速度会明显下降。可以采用连接查询或外连接查询来提高性能。

✧ 避免嵌套的 SELECT 子句

这个实际上是 IN 子句的特例。

✧ 避免使用 SELECT * 语句

如果不是必要取出所有数据，不要用*来代替，应给出字段列表。

✧ 避免不必要的排序

不必要的数据排序大大的降低系统性能。

11.4.4 健壮性

✧ INSERT 语句

使用 INSERT 语句一定要给出要插入值的字段列表，这样即使更改了表结构加了字段也不会影响现有系统的运行。

11.4.5 安全性

✧ WHERE 条件

无论在使用 SELECT,还是使用破坏力极大的 UPDATE 和 DELETE 语句时，一定要检查 WHERE 条件判断的完整性，不要在运行时出现数据的重大丢失。如果不确定，最好先用 SELECT 语句带上相同条件来果一下结果集，来检验条件是否正确。

11.4.6 完整性

有依赖关系的表，例如主外键关系表，在删除父表时必须级联删除其子表相应数据，或则按照某种业务规则转移该数据。

11.5 编码标准

数据库的代码应该清晰,简明,连贯和有注释。

所有脚本/模块应包含一个头和脚，包括主要功能描述, 模块的名称,最近更新日期,作者和版本。

所有过程和函数应包含一个头和脚，包括功能描述,参数,抛出的异常和返回代码。

11.6 游标

所有游标资源使用后应该被释放。

11.7 连接管理

不使用连接池时,连接应在使用后关闭. 当使用连接池时,连接应该在使用后被释放回连接池。

关闭或释放连接之前，连接用到的所有数据库资源应被释放。错误或异常发生时，必须正确地释放资源。

11.8 数据库接口

连接细节(服务器,用户,密码)应存放在配置文件,而不是编在代码里. 这样方便数据库连接代码的部署,无需修改代码。

应用程序应使用通用(数据库抽象层)连接代码访问数据库,这样应用程序就与数据库连接信息和修改(例如引进连接池,软件负载平衡等)隔离开了。

11.9 数据加载

任何数据库数据加载过程，必须创造一个日志文件和错误文件(记录加载失败的数据项和其它错误)。

应考虑加载过程失败的恢复，例如在 COMMIT 时失败。

11.10 维护

所有 schema 的更改,必须通过脚本实现,而不是通过即兴的操作。

所有 schema 更改脚本必须在 DB_CHANGE_LOG 表插入一行 ,用以确定数据库的版本状态。

11.11 变化日志表

每个数据库的 schema 应包括 DB_CHANGE_LOG 表以记录 schema 的更改。DB_CHANGE_LOG 表的结构如下:

- DCL_ID NUMBER(6)
- DCL_SCRIPT_NAME VARCHAR(50)
- DCL_SCRIPT_RUNDATE DATE (/TIMESTAMP)
- DCL_USER VARCHAR(30)
- DCL_CHANGE_DESCRIPTION VARCHAR(255)
- DCL_VERSION VARCHAR(30)

where :

DCL_ID 更改日志条目唯一的标识，可自动生成或从中央记录文档获得。

DCL_SCRIPT_NAME schema的创建/更改脚本的文件名。

DCL_SCRIPT_RUNDATE 脚本执行的日期时间。

DCL_USER 执行更改的用户名。

DCL_CHANGE_DESCRIPTION 更改的简要描述。

DCL_VERSION 更改后schema/数据库的版本，应与设计文档/ER图一致。

11.12 提交的文档

数据库逻辑设计结束后，相关开发人员需要提交数据模型，PDM 或 EXCEL 格式的文档给数据资源中心项目管理人员。

数据模型的每张表都应有功能说明，表的每列也都应有功能和缺省值等的注释。

11.13 设计工具

图形化设计工作，例如 Power Designer 15 及以上等。

12 SQL REVIEW 及优化

为了保证应用和数据库的性能，应用上线实施前，相关开发人员需要提交文档给数据资源中心项目管理人员，列出应用中那些重要的影响性能的 SQL，还有它们的执行计划。

具体可参考《附件 B-数据资源中心数据库性能测试方法.doc》。

13 上线实施

应用上线实施前，相关开发人员需要提交表归档策略文档给数据资源中心项目管理人员，模板见《附件 A-表归档策略_模板.xls》。数据库接口说明文档在上线实施前也需要提交给数据资源中心项目管理人员。

数据库部署操作需要提前一周通知。数据库模式创建脚本应该是文本格式，相关开发人员需要提交给数据资源中心项目管理人员。

14 后续优化

应用上线实施后如果性能有问题，应先检查数据库性能是否有问题。尤其应该优先检查那些复杂 SQL，确保这些 SQL 是最优的。

15 应用范围

本规范原则上适用于软件本部负责的所有软件类项目，其它部门的项目可进行参考。对于 Java 技术类项目，原则上必须全部符合本规范。对于 Dotnet 技术类项目，可以进行适当取舍。

对于完全新建项目，必须全部符合本规范，对于在旧系统之上进行扩展的项目，可以对本规范进行取舍，对于维护类项目，可以不按本规范进行。

由于项目的特殊原因，可以对设计过程进行取舍，但不得降低所执行设计过程的规范要求。

附录－关于数据中心软件系统中防止内存溢出的若干建议

鉴于 Java 资源对象管理的机制，当一个循环退出时，在它里面的所有变量，对象统统释放，当一个方法，以至于一个类结束后，它其中的变量和对象也统统释放，这个就是 java 语言中的作用域的概念，这样看来，通常情况下的 java 程序，想要他出现内存溢出也难。

可是，事无绝对，物无完物，在有些节点下，可能出现一些非常情况，以下总结几点。

使用 applet

描述和建议：

Applet 通常图形界面的小程序，往往是为监听用户操作，或者服务器动作而编写的，其常驻内存，不断监听，循环不会退出，这样的程序如果没有通过优化处理，往往是系统之一害。

过多的 servlet

描述和建议：

Servlet 在中间件里面是常驻内存的，直到服务器关闭才推出,建立 servlet 要在 was,weblogic 的 web.xml 中配置，通常建议一个系统只有一个 servlet，并且在这个 servlet 中不写业务逻辑代码和循环代码，越简单越好。而业务代码的实现用 java 程序去继承这个 servlet 后得而扩充。

线程中有侦听循环

描述和建议：

在线程中，尽量避免像 while(true){}这样的侦听循环，因为这段代码不会结束，从而不会释放其中的对象资源，造成内存叠加的风险。