代码规范

代码规范

(一) 命名风格

1. 【强制】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 / $name / name_ / name$ / name__
2. 【强制】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。
正例：alibaba / taobao / youku / hangzhou 等国际通用的名称，可视同英文。
反例：DaZhePromotion [打折] / getPingfenByName() [评分] / int 某变量= 3
3. 【强制】类名使用UpperCamelCase风格，但以下情形例外：DO / BO / DTO / VO / AO /
PO / UID等。
正例：MarcoPolo / UserDO / XmlService / TcpUdpDeal / TaPromotion
反例：macroPolo / UserDo / XMLService / TCPUDPDeal / TAPromotion
4. 【强制】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. 【强制】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. 【强制】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.【强制】类型与中括号紧挨相连来表示数组。
正例：定义整形数组int[] arrayDemo;
反例：在main 参数中，使用String args[]来定义。
8. 【强制】POJO类(java Bean)中布尔类型的变量，都不要加is前缀，否则部分框架解析会引起序列化错误。
9. 【强制】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.alibaba.ai.util、类名为MessageUtils（此规则参考spring的框架结构）
公司名称.项目名
10. 【强制】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. 【推荐】为了达到代码自解释的目标，任何自定义编程元素在命名时，使用尽量完整的单词
组合来表达其意。
正例：在JDK 中，表达原子更新的类名为：AtomicReferenceFieldUpdater。
反例：变量int a 的随意命名方式。
13. 【推荐】接口类中的方法和属性不要加任何修饰符号（public 也不要加），保持代码的简洁
性，**并加上有效的Javadoc注释**。尽量不要在接口里定义变量，如果一定要定义变量，肯定是与接口方法相关，并且是整个应用的基础**常量**。
正例：接口方法签名void commit();
接口基础常量String COMPANY = ”alibaba”;
反例：接口方法定义public abstract void f();
说明：JDK8中接口允许有默认实现，那么这个default方法，是对所有实现类都有价值的默认实现。
14. 接口和实现类的命名有两套规则：
1）【强制】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。（就是接口的实现啦）
正例：CacheServiceImpl实现CacheService接口。
2）【推荐】如果是形容能力的接口名称，取对应的形容词为接口名（通常是–able的形式）。
正例：AbstractTranslator实现Translatable接口。
16. 【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，复数形式结尾如：listObjects。
3）获取统计值的方法用count做前缀。
4）插入的方法用save/insert做前缀。
5）删除的方法用remove/delete做前缀。
6）修改的方法用update做前缀。

(二) 常量定义

1. 【强制】不允许任何魔法值（即未经预先定义的常量）直接出现在代码中。
2. 【强制】在long或者Long赋值时，数值后使用大写的L，不能是小写的l，小写容易跟数字1混淆，造成误解。
float后面加F
3.【推荐】不要使用一个常量类维护所有常量，要按常量功能进行归类，分开维护。
说明：大而全的常量类，杂乱无章，使用查找功能才能定位到修改的常量，不利于理解和维护。
正例：缓存相关常量放在类CacheConsts下；系统配置相关常量放在类ConfigConsts下。

3）子工程内部共享常量：即在当前子工程的constant目录下。
4）包内共享常量：即在当前包下单独的constant目录下。
5）类内共享常量：直接在类内部private static final定义。

(三) 代码格式

1. 【强制】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. 【强制】左小括号和字符之间不出现空格；同样，右小括号和字符之间也不出现空格；而左大
括号前需要空格。详见第5条下方正例提示。
反例：if (空格a == b空格)
3. 【强制】if/for/while/switch/do等保留字与括号之间都必须加空格。
4. 【强制】任何二目、三目运算符的左右两边都需要加一个空格。
说明：运算符包括赋值运算符=、逻辑运算符&&、加减乘除符号等。
5. 【强制】采用4个空格缩进，禁止使用tab字符。
说明：如果使用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；而在eclipse中，必须勾选insert spaces for tabs。
6. 【强制】注释的双斜线与注释内容之间有且仅有一个空格。
7. 【强制】单行字符数限制不超过120 个，超出需要换行，换行时遵循如下原则：
1）第二行相对第一行缩进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）方法调用中的多个参数需要换行时，在逗号后进行。
5）在括号前不要换行，见反例。
8. 【强制】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args1，后边必须要有一个空格。
method(args1, args2, args3);

(四) OOP规约

1. 【强制】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. 【强制】所有的覆写方法，必须加@Override注解。
说明：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。
5. 【强制】不能使用过时的类或方法。
6. 【强制】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);
7. 【强制】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
包括integer这样的！！！
9. 【强制】定义DO/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。

11. 【强制】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。

五

5. 【强制】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[] { ”you”, ”wu” };
List list = Arrays.asList(str);
第一种情况：list.add(”yangguanbao”); 运行时异常。
第二种情况：str[0] = ”gujin”; 那么list.get(0)也会随之修改。
7. 【强制】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。
正例：
List<String> list = new ArrayList<>();
list.add(”1”);
list.add(”2”);
Iterator<String> iterator = list.iterator();
while (iterator.hasNext()) {
String item = iterator.next();
if (删除元素的条件) {
iterator.remove();
}
}
10. 【推荐】集合初始化时，指定集合初始值大小。
说明：HashMap使用HashMap(int initialCapacity) 初始化。
14. 【参考】利用Set元素唯一的特性，可以快速对一个集合进行去重操作，避免使用List的contains方法进行遍历、对比、去重操作。

七

2. 【强制】在if/else/for/while/do语句中必须使用大括号。即使只有一行代码，避免采用单行的编码方式：if (condition) statements;
4. 【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;
}
7. 【推荐】避免采用取反逻辑运算符。

五、MySQL数据库

1. 【强制】表达是与否概念的字段，必须使用is_xxx的方式命名，数据类型是unsigned tinyint（表示是，表示否）。
3. 【强制】表名不使用复数名词。
5. 【强制】主键索引名为pk_字段名；唯一索引名为uk_字段名；
6. 【强制】小数类型为decimal
7. 【强制】如果存储的字符串 长度几乎相等，使用char定长字符串类型。
8. 【强制】varchar是可变长字符串，不预先分配存储空间，长度不要超过5000，如果存储长度大于此值，定义字段类型为text，独立出来一张表，用主键来对应，避免影响其它字段索引效率。
10. 【推荐】表的命名wt_表的作用
15. 【参考】合适的字符存储长度，不但节约数据库表空间、节约索引存储，更重要的是提升检
索速度。

(二) 索引规约

1. 【强制】业务上具有唯一特性的字段，即使是多个字段的组合，也必须建成唯一索引

5. 【强制】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。

(三) SQL语句
1. 【强制】不要使用count(列名)或count(常量)来替代count(*)，count(*)是SQL92定义的标准统计行数的语法，跟数据库无关，跟NULL和非NULL无关。
说明：count(*)会统计值为NULL的行，而count(列名)不会统计此列为NULL值的行。
3. 【强制】当某一列的值全是NULL时，count(col)的返回结果为0，但sum(col)的返回结果为NULL，因此使用sum()时需注意NPE问题。
4. 【强制】使用ISNULL()来判断是否为NULL值。
说明：NULL与任何值的直接比较都为NULL。
1）NULL<>NULL的返回结果是NULL，而不是false。
2）NULL=NULL的返回结果是NULL，而不是true。
3）NULL<>1的返回结果是NULL，而不是true。

5. 【强制】在代码中写分页查询逻辑时，若count为0应直接返回，避免执行后面的分页语句。
9. 【推荐】in操作能避免则避免，若实在避免不了，需要仔细评估in后边的集合元素数量，控制在1000个之内。

2. 【强制】POJO类的布尔属性不能加is，而数据库字段必须加is_，要求在resultMap中进行字段与属性之间的映射。

JavaDoc编写

写在类上面的Javadoc
写在类上的文档标注一般分为三段：
第一段：概要描述，通常用一句或者一段话简要描述该类的作用，以句号作为结束
第二段：详细描述，通常用一段或者多段话来详细描述该类的作用，一般每段话都以句号作为结束
第三段：文档标注，用于标注作者、创建时间、参阅类等信息

第一段：概要描述

```
package java.lang;

/**
 * Class {@code Object} is the root of the class hierarchy.
 * Every class has {@code Object} as a superclass. All objects,
 * including arrays, implement the methods of this class.
 */
```

@code：{@code text}

将文本标记为代码样式的文本，在code内部可以使用 < 、> 等不会被解释成html标签, code标签有自己的样式

在Javadoc中只要涉及到类名或者方法名，都需要使用@code进行标记。

格式：{@code 包名.类名#方法名(参数类型)}

–ctrl+鼠标左键可以跳转

第二段：详细描述

详细描述一般用一段或者几个锻炼来详细描述类的作用，详细描述中可以使用html标签，如<p>、<pre>、<a>、<ul>、<i>等标签， 通常详细描述都以段落p标签开始。

详细描述和概要描述中间有一个空行来分割

```
package org.springframework.util;

/**
 * Miscellaneous {@link String} utility methods.
 *
 * <p>Mainly for internal use within the framework; consider
 * <a href="http://commons.apache.org/proper/commons-lang/">Apache's
Commons Lang</a>
 * for a more comprehensive suite of {@code String} utilities.
 *
 * <p>This class delivers some simple functionality that should really be
 * provided by the core Java {@link String} and {@link StringBuilder}
 * classes. It also provides easy-to-use methods to convert between
 * delimited strings, such as CSV strings, and collections and arrays.
 *
 */
public abstract class StringUtils {
```

段落都用p标签来标记

凡涉及到类名和方法名都用@code标记

第三段：文档标注

@author

详细描述后面一般使用@author来标记作者，如果一个文件有多个作者来维护就标记多个@author

@see 另请参阅

@see 一般用于标记该类相关联的类,@see即可以用在类上，也可以用在方法上。

@since 从以下版本开始

```
package org.springframework.util;

/**
* @since 16 April 2001
*/
public abstract class StringUtils {}
```

@since 一般用于标记文件创建时项目当时对应的版本，一般后面跟版本号，也可以跟是一个时间，表示文件当前创建的时间

@version 版本

@version 用于标记当前版本，默认为1.0

小乌龟先commit到本地的库，再push（解决冲突）（推到了远端的库）
应该先创建一个develop分支，最后全部没问题了，才会合并到master
eclipse可以直接操作emmm

三：写在方法上的Javadoc

- 第一段：概要描述，通常用一句或者一段话简要描述该方法的作用，以英文句号作为结束
- 第二段：详细描述，通常用一段或者多段话来详细描述该方法的作用，一般每段话都以英文句号作为结束
- 第三段：文档标注，用于标注参数、返回值、异常、参阅等

一、二

方法详细描述上经常使用html标签来，通常都以p标签开始，而且p标签通常都是单标签，不使用结束标签

三

@param 后面跟参数名，再跟参数描述

```
/**
* @param str the {@code CharSequence} to check (may be {@code null})
*/
```

@return 跟返回值的描述

```
/**
* @return {@code true} if the {@code String} is not {@code null}, its
*/
```

@value 用于标注在常量上，{@value} 用于表示常量的值

```
/** 默认数量 {@value} */
private static final Integer QUANTITY = 1;
```

@see 既可以用来类上也可以用在方法上，表示可以参考的类或者方法

```
/**
* @see java.net.URLDecoder#decode(String, String)
*/
```

@throws 跟异常类型 异常描述 , 用于描述方法内部可能抛出的异常

```
/**
* @throws IllegalArgumentException when the given source contains invalid
encoded sequences
*/
```

@exception 用于描述方法签名throws对应的异常

```
/**
* @exception IllegalArgumentException if <code>key</code> is null.
*/
```

@inheritDoc

@inheritDoc用于注解在重写方法或者子类上，用于继承父类中的Javadoc

- 基类的文档注释被继承到了子类
- 子类可以再加入自己的注释（特殊化扩展）
- @return @param @throws 也会被继承