嗨创软服-研发部开发手册 v1.5.0

版本	修订人	修改信息	修改时间
1.0.0	吉银川	初稿	2020 年 10 月 10 日
1.1.0	吉银川	修改服务器环境版本、修改开发环境版本	2020 年 11 月 09 日
1.2.0	吉银川	新增项目开发配置约束、修改项目开发环境约束 修改 GitLab 代码仓库说明、Jenkins 自动部署说明	2020 年 12 月 01 日
1.3.0	吉银川	新增项目数据库开发约束、修改项目接口约束 修改 Jenkins 服务器环境版本	2020 年 12 月 14 日
1.4.0	吉银川	修改开发环境约束	2020 年 12 月 30 日
1.4.1	吉银川	修改开发约束细节	2021 年 01 月 19 日
1.5.0	吉银川	更新服务器相关信息、更新项目开发约束 更新 GitLab 代码仓库说明、更新 Jenkins 持续集成说明	2021 年 06 月 07 日

目　录

一、测试服务器环境说明

名称	版本	备注
系统版本	CentOS7.9	
CPU	4 核	
内存	16GB	
硬盘	200GB	
Nginx	1.19.8	
MySQL	5.7.33	
Redis	6.2.1	
JDK	1.8.0_291	
PHP	7.3	
Composer	2.0.11	

二、Jenkins 服务器环境说明

名称	版本	备注
JDK	1.8.0_291	
Node	v14.15.1	
Npm	7.16.0	
Yarn	1.22.10	
Maven	3.6.3	

三、项目开发环境约束

（一）Java

名称	版本	备注
JDK	1.8	
Maven	3.6.3	
SpringBoot	2.x	
Mysql	5.7	
Druid	*	
Mybatis	*	
MybatisPlus	*	
Swagger	2+	推荐使用 Knife4j
统一请求前缀	/api	
基础配置文件	application.yml	
本地环境配置文件	application-local.yml	
测试环境配置文件	application-dev.yml	
生产环境配置文件	application-prod.yml	
演示环境配置文件	application-publish.yml	

（二）PHP

名称	版本	备注
PHP	7.3	
Mysql	5.7	
Laravel	6+	
Think-PHP	5.1+	
统一请求前缀	/api	
本地环境配置文件	.env.example.local	
测试环境配置文件	.env.example.dev	
生产环境配置文件	.env.example.prod	
演示环境配置文件	.env.example.publish	

（三）前端

名称	版本	备注
Node	14+	
Npm	7+	
Yarn	1+	
Webpack	*	
Vue	2.x	
Vue CLI	4+	
ElementUI	*	
iView	*	

四、项目开发约束

（一）开发约束

1. 默认项目架构采用单体/多模块，如需使用微服务/分布式请提前与我公司研发经理协商
2. 项目代码须包含必要注释，且注释代码量不低于 30%，不得包含无效注释
3. 项目所有资料文件统一放在【代码根目录/doc】文件夹，包括但不限于基础 sql（包含基础权限、管理员帐号等必要数据，清除所有业务数据）、对接文件、流程图、其他说明文件等，如有多个相同文件使用【yyyyMMdd_文件名】格式进行命名
4. 项目使用架构、依赖版本、主流程介绍、开发配置注意事项等统一放在 REDEME.md 文件中
5. 项目设计应按照“高内聚低耦合”的设计理念，适当封装公共方法或组件
6. 项目开发中应尽可能的去解决开发工具中提示的错误及警告问题

（二）配置约束

1. 第三方帐号、外部接口等信息必须写在配置文件中，以引用的方式调用
2. 服务端代码统一使用空格 4 字符缩进，前端代码使用空格 2 字符缩进
3. 项目所有文件编码格式统一使用 UTF-8
4. 所有配置文件中所有配置项须写明备注
5. 日志输出须配置按天与按类型输出并屏蔽无用日志，减少服务器硬盘占用，至少保留 35 天，日志文件输出在【jar 目录/logs】文件夹下

（三）接口约束

1. 项目接口统一请求前缀：`/api`
2. 项目接口传递参数时在前后端须做必要验证，业务校验放在服务端进行
3. 项目接口调用符合 RESTful 架构风格，通过不同请求方式区别接口类型，对所有非公共接口进行权限安全校验，敏感接口采用对称加密、非对称加密等方式进行加解密，以保证接口安全
4. 项目接口时间类型格式统一：`yyyy-MM-dd HH:mm:ss`，业务需求不满足则手动格式化

（四）数据库约束

1. 数据库表名与字段名必须用全小写下划线分割
2. 数据库表字段须至少包含 id、创建时间、最后修改时间，并动态更新修最后改时间
3. 数据库可不使用外键，使用代码维护关联逻辑
4. 查询较多且数据量较大的情况下可适当使用索引
5. 如多人共同设计统一使用模块缩写作为表名开头

五、GitLab 代码仓库说明

（一）帐号

1. 默认账号：开发者邮箱
2. 默认密码：开发者邮箱
3. 登录 GitLab 后必须修改密码，防止帐号密码泄露
4. 如更换邮箱请及时联系管理员进行锁定帐号

（二）开发前配置

1. 查看当前系统 Git 用户名：`git config user.name`
2. 查看当前系统 Git 邮箱：`git config user.email`
3. 设置 Git 用户名：`git config --global user.name '真实姓名'`
4. 设置 Git 邮箱：`git config --global user.email '开发者邮箱'`

（三）分支

5. master 分支：生产环境（线上环境）代码分支，禁止提交
6. develop 分支：开发环境（测试环境）代码分支，开发阶段均提交至此分支
7. publish 分支：演示环境（内部演示）代码分支，demo 及验收阶段修改配置后提交至此分支，配置修改包括但不限于以下信息：
   (1) 删除客户提供所有第三方帐号及 API 接口
   (2) 模拟所有第三方触发事件及响应数据，例如登陆验证码、推送事件触发等
   (3) 更改/删除所有真实手机号、身份证号及其他私密信息

（四）标签

1. Tag 作为版本定位，通常用于二期及以上项目前期项目备份
2. 开发阶段提交 Tag 须写明 Tag 详细描述信息

（五）提交

1. 每次提交时须写明提交详细信息
2. 所有项目均提交根目录，禁止嵌套目录提交
3. 如遇版本冲突须与相关开发人员确定后进行解决后操作

（六）合并

1. 需要更新生产环境代码时须发起合并请求，develop->master，发起合并请求时须写明合并描述信息，管理人员同意后将合并成功
2. 本地开发时禁止使用合并的方式到 develop 分支，统一使用提交的方式进行管理

（七）其他

1. 每周一、五必须提交代码，其余时间确保代码无误后自由提交，且代码须包含必要注释
2. 提交代码前须提前自测，以保证代码准确无误
3. REDEME.MD 文件中需要对项目开发情况进行说明

六、Jenkins 持续集成说明

（一）基础说明

1. 默认 GitLab 代码仓库 develop 分支配置自动部署至测试环境
2. 默认 GitLab 代码仓库 master 分支配置自动部署至生产环境
3. 自动部署后将覆盖服务器上代码，须提前确保代码准确无误

（二）Java

1. 测试环境默认使用[application-dev.yml]配置文件进行构建
2. 生产环境默认使用[application-prod.yml]配置文件进行构建
3. 演示环境默认使用[application-publish.yml]配置文件进行构建
4. 构建前须配置端口与构建后的 Jar 包名称，具体信息以群公告为准

（三）PHP-Laravel

1. 测试环境默认使用[.env.example.dev]配置文件进行构建
2. 生产环境默认使用[.env.example.prod]配置文件环境进行构建
3. 演示环境默认使用[.env.example.publish]配置文件环境进行构建
4. 构建步骤：
   (1) 压缩代码生成压缩包
   (2) 推送压缩包至服务器
   (3) 解压并删除压缩包
   (4) 根据 [composer.json] 和 [vender/autoload.php] 判断是否执行依赖同步
   (5) 如未同步，执行 [composer install]
   (6) 复制.env [cp .env.example.dev .env] 或 [cp .env.example.prod .env]
   (7) 重新生成 app_key [php artisan key:generate]
   (8) 迁移数据库 [php artisan migrate]
   (9) 执行完毕

（四）前端

前提：项目需要配置多环境构建并修改[package.json]中构建命令

1. 测试环境默认使用[yarn build:dev]环境进行构建
2. 生产环境默认使用[yarn build:prod]环境进行构建
3. 演示环境默认使用[yarn build:publish]环境进行构建

七、Java 开发约束（摘选至《Java 开发手册》嵩山版）

（一）编程规约-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】所有编程相关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，纯拼音命名方式更要避免采用。

正例：ali/alibaba/taobao/cainiao/aliyun/youku/hangzhou 等国际通用的名称，可视同英文。

反例：DaZhePromotion[打折]/getPingfenByName()[评分]/Stringfw[福娃]/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：ForceCode/UserDO/HtmlDTO/XmlService/TcpUdpDeal/TaPromotion

反例：forcecode/UserDo/HTMLDto/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。

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/CACHE_EXPIRED_TIME

反例：MAX_COUNT/EXPIRED_TIME

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 Stringargs[]来定义。

8．【强制】POJO 类中的任何布尔类型的变量，都不要加 is 前缀，否则部分框架解析会引起序列化错误。

说明：在本文 MySQL 规约中的建表约定第一条，表达是与否的变量采用 is_xxx 的命名方式，所以，需要在<resultMap>设置从 is_xxx 到 xxx 的映射关系。

反例：定义为基本数据类型 BooleanisDeleted 的属性，它的方法也是 isDeleted()，框架在反向解析的时候，“误以为”对应的属性名称是 deleted，导致属性获取不到，进而抛出异常。

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.alibaba.ei.kunlun.aap.util、类名为 MessageUtils（此规则参考 spring 的框架结构）

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”成 AbsClass；condition“缩写”成 condi；Function 缩写”成 Fu，此类随意缩写严重降低了代码的可阅读性。

11．【参考】枚举类名带上 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。

正例：枚举名字为 ProcessStatusEnum 的成员名称：SUCCESS/UNKNOWN_REASON。

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，复数结尾，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。

B)领域模型命名规约

1）数据对象：xxxDO，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 DO/DTO/BO/VO 的统称，禁止命名成 xxxPOJO。

（二）编程规约-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。

反例：

```
//本例中，开发者 A 定义了缓存的 key，然后开发者 B 使用缓存时少了下划线，即 key 是
"ld#taobao"+tradeId，导致
出现故障
Stringkey="ld#taobao_"+tradeId;
cache.put(key,value);
```

2．【强制】在 long 或者 Long 赋值时，数值后使用大写字母 L，不能是小写字母 l，小写容易跟数字混淆，造成误解。

说明：Longa=2l;写的是数字的 21，还是 Long 型的 2?

【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，也不利于维护。

正例：缓存相关常量放在类 CacheConsts 下；系统配置相关常量放在类 SystemConfigConsts 下。

（三）编程规约-代码格式

1．【强制】如果是大括号内为空，则简洁地写成{}即可，大括号中间无需换行和空格；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】左小括号和右边相邻字符之间不出现空格；右小括号和左边相邻字符之间也不出现空格；而左大括号前需要加空格。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：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 Usetabcharacter；而在 Eclipse 中，必须勾选 insertspacesfortabs。

正例：（涉及 1-5 点）

```
publicstaticvoidmain(String[]args){
    //缩进 4 个空格
    Stringsay="hello";
    //运算符的左右必须有一个空格
    intflag=0;
    //关键词 if 与括号之间必须有一个空格，括号内的 f 与左括号，0 与右括号不需要空格
    if(flag==0){
    System.out.println(say);
    }
    //左大括号前加空格且不换行；左大括号后换行
    if(flag==1){
    System.out.println("world");
    //右大括号前换行，右大括号后有 else，不用换行
        }else{
    System.out.println("ok");
    //在右大括号后直接结束，则必须换行
    }
}
```

6．【强制】注释的双斜线与注释内容之间有且仅有一个空格。

正例：

```
//这是示例注释，请注意在双斜线之后有一个空格
StringcommentString=newString();
```

7．【强制】方法参数在定义和传入时，多个参数逗号后面必须加空格。

正例：下例中实参的 args1，后边必须要有一个空格。

```
method(args1,args2,args3);
```

8．【强制】IDE 的 textfileencoding 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。

9．【推荐】没有必要增加若干空格来使变量的赋值等号与上一行对应位置的等号对齐。

正例：

```
intone=1;
longtwo=2L;
floatthree=3F;
StringBuildersb=newStringBuilder();
```

说明：增加 sb 这个变量，如果需要对齐，则给 one、two、three 都要增加几个空格，在变量比较多的情况下，是非常累赘的事情

10．【推荐】不同逻辑、不同语义、不同业务的代码之间插入一个空行分隔开来以提升可读性。

说明：任何情形，没有必要插入多个空行进行隔开。

（四）编程规约-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。

3．【强制】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。（建议开发者尽量不用可变参数编程）

正例：publicList<User>listUsers(Stringtype,Long...ids){...}

4．【强制】不能使用过时的类或方法。

说明：java.net.URLDecoder 中的方法 decode(StringencodeStr)这个方法已经过时，应该使用双参数 decode(Stringsource,Stringencode)。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。

5．【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);

反例：object.equals("test");

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

6．【强制】所有整型包装类对象之间值的比较，全部使用 equals 方法比较。

说明：对于 Integervar=?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。

7．【强制】BigDecimal 的等值比较应使用 compareTo()方法，而不是 equals()方法。

说明：equals()方法会比较值和精度（1.0 与 1.00 返回结果为 false），而 compareTo()则会忽略精度。

8．【强制】定义数据对象 DO 类时，属性类型要与数据库字段类型相匹配。

正例：数据库字段的 bigint 必须与类属性的 Long 类型相对应。

反例：某个案例的数据库表 id 字段定义类型 bigintunsigned，实际类对象属性为 Integer，随着 id 越来越大，超过 Integer 的表示范围而溢出成为负数。

9．【强制】禁止使用构造方法 BigDecimal(double)的方式把 double 值转化为 BigDecimal 对象。

说明：BigDecimal(double)存在精度损失风险，在精确计算或值比较的场景中可能会导致业务逻辑异常。

如：BigDecimalg=newBigDecimal(0.1F);实际的存储值为：0.10000000149

正例：优先推荐入参为 String 的构造方法，或使用 BigDecimal 的 valueOf 方法，此方法内部其实执行了

Double 的 toString，而 Double 的 toString 按 double 的实际能表达的精度对尾数进行了截断。

```
BigDecimalrecommend1=newBigDecimal("0.1");
BigDecimalrecommend2=BigDecimal.valueOf(0.1);
```

10．关于基本数据类型与包装数据类型的使用标准如下：

1） 【强制】所有的 POJO 类属性必须使用包装数据类型。

2） 【强制】RPC 方法的返回值和参数必须使用包装数据类型。

3） 【推荐】所有的局部变量使用基本数据类型。

说明：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 NPE 问题，或者入库检查，都由使用者来保证。

正例：数据库的查询结果可能是 null，因为自动拆箱，用基本数据类型接收有 NPE 风险。

反例：某业务的交易报表上显示成交总额涨跌情况，即正负 x%，x 为基本数据类型，调用的 RPC 服务，调用不成功时，返回的是默认值，页面显示为 0%，这是不合理的，应该显示成中划线-。所以包装数据类型的 null 值，能够表示额外的信息，如：远程调用失败，异常退出。

11．【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 createTime 默认值为 newDate()，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。

12．【强制】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。

13．【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。

14．【强制】POJO 类必须写 toString 方法。使用 IDE 中的工具：source>generatetoString 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。

15．【推荐】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，此条规则优先于下一条。

16．【推荐】类内方法定义的顺序依次是：公有方法或保护方法>私有方法>getter/setter方法。

说明：公有方法是类的调用者和维护者最关心的方法，首屏展示最好；保护方法虽然只是子类关心，也可能是“模板设计模式”下的核心方法；而私有方法外部一般不需要特别关心，是一个黑盒实现；因为承载的信息价值较低，所有 Service 和 DAO 的 getter/setter 方法放在类体最后。

17．【推荐】循环体内，字符串的连接方式，使用 StringBuilder 的 append 方法进行扩展。

说明：下例中，反编译出的字节码文件显示每次循环都会 new 出一个 StringBuilder 对象，然后进行 append 操作，最后通过 toString 方法返回 String 对象，造成内存资源浪费。

反例：

```
Stringstr="start";
for(inti=0;i<100;i++){
    str=str+"hello";
}
```

18．【推荐】慎用 Object 的 clone 方法来拷贝对象。

说明：对象 clone 方法默认是浅拷贝，若想实现深拷贝，需覆写 clone 方法实现域对象的深度遍历式拷贝。

（五）编程规约-日期时间

1．【强制】日期格式化时，传入 pattern 中表示年份统一使用小写的 y。

说明：日期格式化时，yyyy 表示当天所在的年，而大写的 YYYY 代表是 weekinwhichyear（JDK7 之后引入的概念），意思是当天所在的周属于的年份，一周从周日开始，周六结束，只要本周跨年，返回的 YYYY 就是下一年。

正例：表示日期和时间的格式如下所示：

newSimpleDateFormat("yyyy-MM-ddHH:mm:ss")

2．【强制】在日期格式中分清楚大写的 M 和小写的 m，大写的 H 和小写的 h 分别指代的意义。

说明：日期格式中的这两对字母表意如下：

1）表示月份是大写的 M；

2）表示分钟则是小写的 m；

3）24 小时制的是大写的 H；

4）12 小时制的则是小写的 h。

3．【强制】获取当前毫秒数：System.currentTimeMillis();而不是 newDate().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。

4．【强制】不允许在程序任何地方中使用：1）java.sql.Date。2）java.sql.Time。

3）java.sql.Timestamp。

说明：第 1 个不记录时间，getHours()抛出异常；第 2 个不记录日期，getYear()抛出异常；第 3 个在构造方法 super((time/1000)*1000)，在 Timestamp 属性 fastTime 和 nanos 分别存储秒和纳秒信息。

反例：java.util.Date.after(Date)进行时间比较时，当入参是 java.sql.Timestamp 时，会触发 JDKBUG(JDK9 已修复)，可能导致比较时的意外结果。

5．【强制】不要在程序中写死一年为 365 天，避免在公历闰年时出现日期转换错误或程序逻辑错误。

正例：

```
//获取今年的天数
intdaysOfThisYear=LocalDate.now().lengthOfYear();
//获取指定某年的天数
LocalDate.of(2011,1,1).lengthOfYear();
```

反例：

```
//第一种情况：在闰年 366 天时，出现数组越界异常
int[]dayArray=newint[365];
//第二种情况：一年有效期的会员制，今年 1 月 26 日注册，硬编码 365 返回的却是 1 月 25 日
Calendarcalendar=Calendar.getInstance();
calendar.set(2020,1,26);
calendar.add(Calendar.DATE,365);
```

6．【推荐】避免公历闰年 2 月问题。闰年的 2 月份有 29 天，一年后的那一天不可能是 2 月 29 日。

（六）编程规约-集合处理

1．【强制】关于 hashCode 和 equals 的处理，遵循如下规则：

1）只要覆写 equals，就必须覆写 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 存储的对象必须覆写这两种方法。

3）如果自定义对象作为 Map 的键，那么必须覆写 hashCode 和 equals。

说明：String 因为覆写了 hashCode 和 equals 方法，所以可以愉快地将 String 对象作为 key 来使用。

2．【强制】判断所有集合内部的元素是否为空，使用 isEmpty()方法，而不是 size()==0 的方式。

说明：在某些集合中，前者的时间复杂度为 O(1)，而且可读性更好。

正例：

```
Map<String,Object>map=newHashMap<>(16);
if(map.isEmpty()){
    System.out.println("noelementinthismap.");
}
```

3．【强制】ArrayList 的 subList 结果不可强转成 ArrayList，否则会抛出 ClassCastException 异常：java.util.RandomAccessSubListcannotbecasttojava.util.ArrayList。

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

4．【强制】使用 Map 的方法 keySet()/values()/entrySet()返回集合对象时，不可以对其进行添加元素操作，否则会抛出 UnsupportedOperationException 异常。

5．【强制】Collections 类返回的对象，如：emptyList()/singletonList()等都是 immutablelist，不可对其进行添加或者删除元素的操作。

反例：如果查询无结果，返回 Collections.emptyList()空集合对象，调用方一旦进行了添加元素的操作，就会触发 UnsupportedOperationException 异常。

6．【强制】在 subList 场景中，高度注意对父集合元素的增加或删除，均会导致子列表的遍历、增加、删除产生 ConcurrentModificationException 异常。

7．【强制】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致、长度为 0 的空数组。

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。

正例：

```
List<String>list=newArrayList<>(2);
list.add("guan");
list.add("bao");
String[]array=list.toArray(newString[0]);
```

说明：使用 toArray 带参方法，数组空间大小的 length：

1）等于 0，动态创建与 size 相同的数组，性能最好。

2）大于 0 但小于 size，重新创建大小等于 size 的数组，增加 GC 负担。

3）等于 size，在高并发情况下，数组创建完成之后，size 正在变大的情况下，负面影响与 2 相同。

4）大于 size，空间浪费，且在 size 处插入 null 值，存在 NPE 隐患。

8．【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=newString[]{"chen","yang","hao"};
Listlist=Arrays.asList(str);
```

第一种情况：list.add("yangguanbao");运行时异常。

第二种情况：str[0]="change";也会随之修改，反之亦然。

9．【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=newArrayList<>();
list.add("1");
list.add("2");
Iterator<String>iterator=list.iterator();
while(iterator.hasNext()){
    Stringitem=iterator.next();
    if(删除元素的条件){
        iterator.remove();
    }
}
```

反例：

```
for(Stringitem:list){
    if("1".equals(item)){
        list.remove(item);
    }
}
```

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？

10．【强制】在 JDK7 版本及以上，Comparator 实现类要满足如下三个条件，不然 Arrays.sort，Collections.sort 会抛 IllegalArgumentException 异常。

说明：三个条件如下

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，交换两个对象判断结果并不互反，不符合第一个条件，在实际使用中可能会出现异常。

```
newComparator<Student>(){
    @Override
    publicintcompare(Studento1,Studento2){
        returno1.getId()>o2.getId()?1:-1;
    }
};
```

11．【推荐】集合初始化时，指定集合初始值大小。

说明：HashMap 使用 HashMap(intinitialCapacity)初始化，如果暂时无法确定集合大小，那么指定默认值（16）即可。

正例：initialCapacity=(需要存储的元素个数/负载因子)+1。注意负载因子（即 loaderfactor）默认为 0.75，如果暂时无法确定初始值大小，请设置为 16（即默认值）。

反例：HashMap 需要放置 1024 个元素，由于没有设置容量初始大小，随着元素增加而被迫不断扩容，resize()方法总共会调用 8 次，反复重建哈希表和数据迁移。当放置的集合元素个数达千万级时会影响程序性能。

12．【推荐】使用 entrySet 遍历 Map 类集合 KV，而不是 keySet 方式进行遍历。

说明：keySet 其实是遍历了 2 次，一次是转为 Iterator 对象，另一次是从 hashMap 中取出 key 所对应的 value。而 entrySet 只是遍历了一次就把 key 和 value 都放到了 entry 中，效率更高。如果是 JDK8，使用 Map.forEach 方法。

正例：values()返回的是 V 值集合，是一个 list 集合对象；keySet()返回的是 K 值集合，是一个 Set 集合对象；entrySet()返回的是 K-V 值组合集合。

13．【推荐】高度注意 Map 类集合 K/V 能不能存储 null 值的情况，如下表格：

集合 类	Key	Value	Super	说 明
Hashtable	不允许为 null	不允许为 null	Dictionary	线程安全
ConcurrentHashMap	不允许为 null	不允许为 null	AbstractMap	锁分段技术 (JDK8:CAS)
TreeMap	不允许为 null	允许为 null	AbstractMap	线程不安全
HashMap	允许为 null	允许为 null	AbstractMap	线程不安全

反例：由于 HashMap 的干扰，很多人认为 ConcurrentHashMap 是可以置入 null 值，而事实上，存储 null 值时会抛出 NPE 异常。

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()进行遍历去重或者判断包含操作。

（七）编程规约-并发处理

1．【强制】获取单例对象需要保证线程安全，其中的方法也要保证线程安全。

说明：资源驱动类、工具类、单例工厂类都需要注意。

2．【强制】创建线程或线程池时请指定有意义的线程名称，方便出错时回溯。

正例：自定义线程工厂，并且根据外部特征进行分组，比如，来自同一机房的调用，把机房编号赋值给 whatFeatureOfGroup

```
publicclassUserThreadFactoryimplementsThreadFactory{
    privatefinalStringnamePrefix;
    privatefinalAtomicIntegernextId=newAtomicInteger(1);
    //定义线程组名称，在利用 jstack 来排查问题时，非常有帮助
    UserThreadFactory(StringwhatFeatureOfGroup){
        namePrefix="FromUserThreadFactory's"+whatFeatureOfGroup+"-Worker-";
    }
    @Override
    publicThreadnewThread(Runnabletask){
        Stringname=namePrefix+nextId.getAndIncrement();
        Threadthread=newThread(null,task,name,0,false);
        System.out.println(thread.getName());
        returnthread;
    }
}
```

3．【强制】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。

说明：线程池的好处是减少在创建和销毁线程上所消耗的时间以及系统资源的开销，解决资源不足的问题。

如果不使用线程池，有可能造成系统创建大量同类线程而导致消耗完内存或者“过度切换”的问题。

4．【强制】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：

1）FixedThreadPool 和 SingleThreadPool：

允许的请求队列长度为 Integer.MAX_VALUE，可能会堆积大量的请求，从而导致 OOM。

2）CachedThreadPool：

允许的创建线程数量为 Integer.MAX_VALUE，可能会创建大量的线程，从而导致 OOM。

5．【强制】SimpleDateFormat 是线程不安全的类，一般不要定义为 static 变量，如果定义为 static，必须加锁，或者使用 DateUtils 工具类。

正例：注意线程安全，使用 DateUtils。亦推荐如下处理：

```
privatestaticfinalThreadLocal<DateFormat>df=newThreadLocal<DateFormat>(){
    @Override
    protectedDateFormatinitialValue(){
        returnnewSimpleDateFormat("yyyy-MM-dd");
    }
};
```

说明：如果是 JDK8 的应用，可以使用 Instant 代替 Date，LocalDateTime 代替 Calendar，DateTimeFormatter 代替 SimpleDateFormat，官方给出的解释：

simplebeautifulstrongimmutablethread-safe。

6．【强制】必须回收自定义的 ThreadLocal 变量，尤其在线程池场景下，线程经常会被复用，如果不清理自定义的 ThreadLocal 变量，可能会影响后续业务逻辑和造成内存泄露等问题。尽量在代理中使用 try-finally 块进行回收。

正例：

```
objectThreadLocal.set(userInfo);
try{
    //...
}finally{
    objectThreadLocal.remove();
}
```

7．【强制】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。

说明：尽可能使加锁的代码块工作量尽可能的小，避免在锁代码块中调用 RPC 方法。

8．【强制】并发修改同一记录时，避免更新丢失，需要加锁。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。乐观锁的重试次数不得小于 3 次。

9．【强制】多线程并行处理定时任务时，Timer 运行多个 TimeTask 时，只要其中之一没有捕获抛出的异常，其它任务便会自动终止运行，使用 ScheduledExecutorService 则没有这个问题。

10．【推荐】资金相关的金融敏感信息，使用悲观锁策略。

说明：乐观锁在获得锁的同时已经完成了更新操作，校验逻辑容易出现漏洞，另外，乐观锁对冲突的解决策略有较复杂的要求，处理不当容易造成系统压力或数据异常，所以资金相关的金融敏感信息不建议使用乐观锁更新。

正例：悲观锁遵循一锁、二判、三更新、四释放的原则。

11．【推荐】避免 Random 实例被多线程使用，虽然共享该实例是线程安全的，但会因竞争同一 seed 导致的性能下降。

说明：Random 实例包括 java.util.Random 的实例或者 Math.random()的方式。

正例：在 JDK7 之后，可以直接使用 APIThreadLocalRandom，而在 JDK7 之前，需要编码保证每个线程持有一个单独的 Random 实例。

12．【参考】ThreadLocal 对象使用 static 修饰，ThreadLocal 无法解决共享对象的更新问题。

说明：这个变量是针对一个线程内所有操作共享的，所以设置为静态变量，所有此类实例共享此静态变量，也就是说在类第一次被使用时装载，只分配一块存储空间，所有此类的对象(只要是这个线程内定义的)都可以操控这个变量。

（八）编程规约-注释规约

1．【强制】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．【强制】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．【强制】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。

4．【强制】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。

5．【推荐】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。

6．【推荐】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。

说明：代码与注释更新不同步，就像路网与导航软件更新不同步一样，如果导航软件严重滞后，就失去了导航的意义。

7．【推荐】在类中删除未使用的任何字段、方法、内部类；在方法中删除未使用的任何参数声明与内部变量。

8．【参考】谨慎注释掉代码。在上方详细说明，而不是简单地注释掉。如果无用，则删除。

说明：代码被注释掉有两种可能性：1）后续会恢复此段代码逻辑。2）永久不用。前者如果没有备注信息，难以知晓注释动机。后者建议直接删掉即可，假如需要查阅历史代码，登录代码仓库即可。

9．【参考】对于注释的要求：第一、能够准确反映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。

10．【参考】好的命名、代码结构是自解释的，注释力求精简准确、表达到位。避免出现注释的一个极端：过多过滥的注释，代码的逻辑一旦修改，修改注释又是相当大的负担。

反例：

```
//putelephantintofridge
put(elephant,fridge);
```

方法名 put，加上两个有意义的变量名 elephant 和 fridge，已经说明了这是在干什么，语义清晰的代码不需要额外的注释

11．【参考】特殊注释标记，请注明标记人与标记时间。注意及时处理这些标记，通过标记扫描，经常清理此类标记。线上故障有时候就是来源于这些标记处的代码。

1）待办事宜（TODO）：（标记人，标记时间，[预计处理时间]）

表示需要实现，但目前还未实现的功能。这实际上是一个 Javadoc 的标签，目前的 Javadoc 还没有实现，但已经被广泛使用。只能应用于类，接口和方法（因为它是一个 Javadoc 标签）。

2）错误，不能工作（FIXME）：（标记人，标记时间，[预计处理时间]）

在注释中用 FIXME 标记某代码是错误的，而且不能工作，需要及时纠正的情况。

（九）编程规约-注释规约

1．【强制】前后端数据列表相关的接口返回，如果为空，则返回空数组[]或空集合{}。

说明：此条约定有利于数据层面上的协作更加高效，减少前端很多琐碎的 null 判断。

2．【强制】服务端发生错误时，返回给前端的响应信息必须包含 HTTP 状态码，errorCode、errorMessage、用户提示信息四个部分。

说明：四个部分的涉众对象分别是浏览器、前端开发、错误排查人员、用户。其中输出给用户的提示信息要求：简短清晰、提示友好，引导用户进行下一步操作或解释错误原因，提示信息可以包括错误原因、上下文环境、推荐操作等。errorCode：参考**附表 3**。errorMessage：简要描述后端出错原因，便于错误排查人员快速定位问题，注意不要包含敏感数据信息。

正例：常见的 HTTP 状态码如下

1）200OK:表明该请求被成功地完成，所请求的资源发送到客户端。

2）401Unauthorized:请求要求身份验证，常见对于需要登录而用户未登录的情况。

3）403Forbidden：服务器拒绝请求，常见于机密信息或复制其它登录用户链接访问服务器的情况。

4）404NotFound:服务器无法取得所请求的网页，请求资源不存在。

5）500InternalServerError:服务器内部错误。

3．【强制】在前后端交互的 JSON 格式数据中，所有的 key 必须为小写字母开始的 lowerCamelCase 风格，符合英文表达习惯，且表意完整。

正例：errorCode/errorMessage/assetStatus/menuList/orderList/configFlag

反例：ERRORCODE/ERROR_CODE/error_message/error-message/errormessage/ErrorMessage/msg

4．【强制】errorMessage 是前后端错误追踪机制的体现，可以在前端输出到 type="hidden" 文字类控件中，或者用户端的日志中，帮助我们快速地定位出问题。

5．【强制】对于需要使用超大整数的场景，服务端一律使用 String 字符串类型返回，禁止使用 Long 类型。

说明：Java 服务端如果直接返回 Long 整型数据给前端，JS 会自动转换为 Number 类型（注：此类型为双精度浮点数，表示原理与取值范围等同于 Java 中的 Double）。Long 类型能表示的最大值是 2 的 63 次方-1，在取值范围之内，超过 2 的 53 次方(9007199254740992)的数值转化为 JS 的 Number 时，有些数值会有精度损失。扩展说明，在 Long 取值范围内，任何 2 的指数次整数都是绝对不会存在精度损失的，所以说精度损失是一个概率问题。若浮点数尾数位与指数位空间不限，则可以精确表示任何整数，但很不幸，双精度浮点数的尾数位只有 52 位。

反例：通常在订单号或交易号大于等于 16 位，大概率会出现前后端单据不一致的情况，比如，"orderId":362909601374617692，前端拿到的值却是:362909601374617660。

6．【强制】在翻页场景中，用户输入参数的小于 1，则前端返回第一页参数给后端；后端发现用户输入的参数大于总页数，直接返回最后一页。

7．【推荐】服务端返回的数据，使用 JSON 格式而非 XML。

说明：尽管 HTTP 支持使用不同的输出格式，例如纯文本，JSON，CSV，XML，RSS 甚至 HTML。如果我们使用的面向用户的服务，应该选择 JSON 作为通信中使用的标准数据交换格式，包括请求和响应。此外，application/JSON 是一种通用的 MIME 类型，具有实用、精简、易读的特点。

8．【推荐】前后端的时间格式统一为"yyyy-MM-ddHH:mm:ss"，统一为 GMT。

9．【参考】在接口路径中不要加入版本号，版本控制在 HTTP 头信息中体现，有利于向前兼容。

说明：当用户在低版本与高版本之间反复切换工作时，会导致迁移复杂度升高，存在数据错乱风险。

（十）异常日志-日志规约

1．【强制】应用中不可直接使用日志系统（Log4j、Logback）中的 API，而应依赖使用日志框架（SLF4J、JCL--JakartaCommonsLogging）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。

说明：日志框架（SLF4J、JCL--JakartaCommonsLogging）的使用方式（推荐使用 SLF4J）

使用 SLF4J：

```
importorg.slf4j.Logger;
importorg.slf4j.LoggerFactory;
privatestaticfinalLoggerlogger=LoggerFactory.getLogger(Test.class);
```

使用 JCL：

```
importorg.apache.commons.logging.Log;
importorg.apache.commons.logging.LogFactory;
privatestaticfinalLoglog=LogFactory.getLog(Test.class);
```

2．【强制】所有日志文件至少保存 15 天，因为有些异常具备以“周”为频次发生的特点。对于当天日志，以“应用名.log”来保存，保存在/home/admin/应用名/logs/目录下，过往日志格式为:{logname}.log.{保存日期}，日期格式：yyyy-MM-dd

正例：以 aap 应用为例，日志保存在/home/admin/aapserver/logs/aap.log，历史日志名称为 aap.log.2016-08-01

3．【强制】根据国家法律，网络运行状态、网络安全事件、个人敏感信息操作等相关记录，留存的日志不少于六个月，并且进行网络多机备份。

4．【强制】在日志输出时，字符串变量之间的拼接使用占位符的方式。

说明：因为 String 字符串的拼接会使用 StringBuilder 的 append()方式，有一定的性能损耗。使用占位符仅是替换动作，可以有效提升性能。

正例：

```
logger.debug("Processingtradewithid:{}andsymbol:{}",id,symbol);
```

5．【强制】对于 trace/debug/info 级别的日志输出，必须进行日志级别的开关判断。

说明：虽然在 debug(参数)的方法体内第一行代码 isDisabled(Level.DEBUG_INT)为真时（Slf4j 的常见实现 Log4j 和 Logback），就直接 return，但是参数可能会进行字符串拼接运算。此外，如果 debug(getName())这种参数内有 getName()方法调用，无谓浪费方法调用的开销。

正例：

```
//如果判断为真，那么可以输出 trace 和 debug 级别的日志
if(logger.isDebugEnabled()){
    logger.debug("CurrentIDis:{}andnameis:{}",id,getName());
}
```

6．【强制】避免重复打印日志，浪费磁盘空间，务必在日志配置文件中设置 additivity=false。

正例：<loggername="com.taobao.dubbo.config"additivity="false">

7．【强制】生产环境禁止直接使用 System.out 或 System.err 输出日志或使用 e.printStackTrace()打印异常堆栈。

说明：标准日志输出与标准错误输出文件每次 Jboss 重启时才滚动，如果大量输出送往这两个文件，容易造成文件大小超过操作系统大小限制。

8．【强制】异常信息应该包括两类信息：案发现场信息和异常堆栈信息。如果不处理，那么通过关键字 throws 往上抛出。

正例：logger.error("inputParams:{}anderrorMessage:{}",各类参数或者对象 toString(),e.getMessage(),e);

9．【强制】日志打印时禁止直接用 JSON 工具将对象转换成 String。

说明：如果对象里某些 get 方法被覆写，存在抛出异常的情况，则可能会因为打印日志而影响正常业务流程的执行。

正例：打印日志时仅打印出业务相关属性值或者调用其对象的 toString()方法。

10．【推荐】谨慎地记录日志。生产环境禁止输出 debug 日志；有选择地输出 info 日志；如果使用 warn 来记录刚上线时的业务行为信息，一定要注意日志输出量的问题，避免把服务器磁盘撑爆，并记得及时删除这些观察日志。

说明：大量地输出无效日志，不利于系统性能提升，也不利于快速定位错误点。记录日志时请思考：这些日志真的有人看吗？看到这条日志你能做什么？能不能给问题排查带来好处？

11．【推荐】可以使用 warn 日志级别来记录用户输入参数错误的情况，避免用户投诉时，无所适从。如非必要，请不要在此场景打出 error 级别，避免频繁报警。

说明：注意日志输出的级别，error 级别只记录系统逻辑出错、异常或者重要的错误信息。

（十一）MySQL 数据库-建表规约

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 unsignedtinyint（1 表示是，0 表示否）。

说明：任何字段如果为非负数，必须是 unsigned。

注意：POJO 类中的任何布尔类型的变量，都不要加 is 前缀，所以，需要在<resultMap>设置从 is_xxx 到 Xxx 的映射关系。数据库表示是与否的值，使用 tinyint 类型，坚持 is_xxx 的命名方式是为了明确其取值含义与取值范围。

正例：表达逻辑删除的字段名 is_deleted，1 表示删除，0 表示未删除。

2．【强制】表名、字段名必须使用小写字母或数字，禁止出现数字开头，禁止两个下划线中间只出现数字。数据库字段名的修改代价很大，因为无法进行预发布，所以字段名称需要慎重考虑。

说明：MySQL 在 Windows 下不区分大小写，但在 Linux 下默认是区分大小写。因此，数据库名、表名、字段名，都不允许出现任何大写字母，避免节外生枝。

正例：aliyun_admin，rdc_config，level3_name

反例：AliyunAdmin，rdcConfig，level_3_name

3．【强制】表名不使用复数名词。

说明：表名应该仅仅表示表里面的实体内容，不应该表示实体数量，对应于 DO 类名也是单数形式，符合表达习惯。

4．【强制】主键索引名为 pk_字段名；唯一索引名为 uk_字段名；普通索引名则为 idx_字段名。

说明：pk_即 primarykey；uk_即 uniquekey；idx_即 index 的简称。

5．【强制】小数类型为 decimal，禁止使用 float 和 double。

说明：在存储的时候，float 和 double 都存在精度损失的问题，很可能在比较值的时候，得到不正确的结果。如果存储的数据范围超过 decimal 的范围，建议将数据拆成整数和小数并分开存储。

6．【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。

7．【强制】表必备三字段：id,create_time,update_time。

说明：其中 id 必为主键，类型为 bigintunsigned、单表时自增、步长为 1。create_time,update_time 的类型均为 datetime 类型，前者现在时表示主动式创建，后者过去分词表示被动式更新。

8．【推荐】表的命名最好是遵循“业务名称_表的作用”。

正例：alipay_task/force_project/trade_config

9．【推荐】如果修改字段含义或对字段表示的状态追加时，需要及时更新字段注释。

10．【推荐】字段允许适当冗余，以提高查询性能，但必须考虑数据一致。冗余字段应遵循：

1）不是频繁修改的字段。

2）不是唯一索引的字段。

3）不是 varchar 超长字段，更不能是 text 字段。

正例：各业务线经常冗余存储商品名称，避免查询时需要调用 IC 服务获取。

11．【推荐】单表行数超过 500 万行或者单表容量超过 2GB，才推荐进行分库分表。

说明：如果预计三年后的数据量根本达不到这个级别，请不要在创建表时就分库分表。

12．【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。

对象	年龄区间	类型	字节	表示范围
人	150 岁之内	tinyintunsigned	1	无符号值：0 到 255
龟	数百岁	smallintunsigned	2	无符号值：0 到 65535
恐龙化石	数千万年	intunsigned	4	无符号值：0 到约 43 亿
太阳	约 50 亿年	bigintunsigned	8	无符号值：0 到约 10 的 19 次方

正例：无符号值可以避免误存负数，且扩大了表示范围。

（十二）MySQL 数据库-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。

2．【强制】count(distinctcol)计算该列除 NULL 之外的不重复行数，注意 count(distinctcol1,col2)如果其中一列全为 NULL，那么即使另一列有不同的值，也返回为 0。

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。

正例：可以使用如下方式来避免 sum 的 NPE 问题：SELECTIFNULL(SUM(column),0)FROMtable;

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。

反例：在 SQL 语句中，如果在 null 前换行，影响可读性。

Select*fromtablewherecolumn1isnullandcolumn3isnotnull;而`ISNULL(column)`是一个整体，简洁易懂。从性能数据上分析，`ISNULL(column)`执行效率更快一些。

5．【强制】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。

6．【强制】不得使用外键与级联，一切外键概念必须在应用层解决。

说明：（概念解释）学生表中的 student_id 是主键，那么成绩表中的 student_id 则为外键。如果更新学生表中的 student_id，同时触发成绩表中的 student_id 更新，即为级联更新。外键与级联更新适用于单机低并发，不适合分布式、高并发集群；级联更新是强阻塞，存在数据库更新风暴的风险；外键影响数据库的插入速度。

7．【强制】禁止使用存储过程，存储过程难以调试和扩展，更没有移植性。

8．【强制】数据订正（特别是删除或修改记录操作）时，要先 select，避免出现误删除，确认无误才能执行更新语句。

9．【强制】对于数据库中表记录的查询和变更，只要涉及多个表，都需要在列名前加表的别名（或表名）进行限定。

说明：对多表进行查询记录、更新记录、删除记录时，如果对操作列没有限定表的别名（或表名），并且操作列在多个表中存在时，就会抛异常。

正例：selectt1.namefromtable_firstast1,table_secondast2wheret1.id=t2.id;

反例：在某业务中，由于多表关联查询语句没有加表的别名（或表名）的限制，正常运行两年后，最近在某个表中增加一个同名字段，在预发布环境做数据库变更后，线上查询语句出现出 1052 异常：Column'name'infieldlistisambiguous。

10．【推荐】SQL 语句中表的别名前加 as，并且以 t1、t2、t3、...的顺序依次命名。

说明：

1）别名可以是表的简称，或者是依照表在 SQL 语句中出现的顺序，以 t1、t2、t3 的方式命名。

2）别名前加 as 使别名更容易识别。

正例：selectt1.namefromtable_firstast1,table_secondast2wheret1.id=t2.id;

11．【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。

12．【参考】因国际化需要，所有的字符存储与表示，均采用 utf8 字符集，那么字符计数方法需要注意。

说明:

SELECTLENGTH("轻松工作")；返回为 12

SELECTCHARACTER_LENGTH("轻松工作")；返回为 4

如果需要存储表情，那么选择 utf8mb4 来进行存储，注意它与 utf8 编码的区别。

13．【参考】TRUNCATETABLE 比 DELETE 速度快，且使用的系统和事务日志资源少，但 TRUNCATE 无事务且不触发 trigger，有可能造成事故，故不建议在开发代码中使用此语句。

说明：TRUNCATETABLE 在功能上与不带 WHERE 子句的 DELETE 语句相同。

（十三）MySQL 数据库-ORM 映射

1．【强制】在表查询中，一律不要使用*作为查询的字段列表，需要哪些字段必须明确写明。

说明：1）增加查询分析器解析成本。2）增减字段容易与 resultMap 配置不一致。3）无用字段增加网络消耗，尤其是 text 类型的字段。

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。

说明：参见定义 POJO 类以及数据库字段定义规定，在 sql.xml 增加映射，是必须的。

3．【强制】sql.xml 配置参数使用：#{}，#param#不要使用${}此种方式容易出现 SQL 注入。

4．【强制】iBATIS 自带的 queryForList(StringstatementName,intstart,intsize)不推荐使用。

说明：其实现方式是在数据库取到 statementName 对应的 SQL 语句的所有记录，再通过 subList 取

start,size 的子集合。

正例：

```
Map<String,Object>map=newHashMap<>(16);
map.put("start",start);
map.put("size",size);
```

5．【强制】不允许直接拿 HashMap 与 Hashtable 作为查询结果集的输出。

反例：某同学为避免写一个<resultMap>xxx</resultMap>，直接使用 HashTable 来接收数据库返回结果，结果出现日常是把 bigint 转成 Long 值，而线上由于数据库版本不一样，解析成 BigInteger，导致线上问题。

6．【强制】更新数据表记录时，必须同时更新记录对应的 update_time 字段值为当前时间。

7．【推荐】不要写一个大而全的数据更新接口。传入为 POJO 类，不管是不是自己的目标更新字段，都进行 updatetablesetc1=value1,c2=value2,c3=value3;这是不对的。执行 SQL 时，不要更新无改动的字段，一是易出错；二是效率低；三是增加 binlog 存储。

8．【参考】@Transactional 事务不要滥用。事务会影响数据库的 QPS，另外使用事务的地方需要考虑各方面的回滚方案，包括缓存回滚、搜索引擎回滚、消息补偿、统计修正等。

附 1：专有名词解释

1．POJO（PlainOrdinaryJavaObject）:在本规约中，POJO 专指只有 setter/getter/toString 的简单类，包括 DO/DTO/BO/VO 等。

2．DO（DataObject）：阿里巴巴专指数据库表一一对应的 POJO 类。此对象与数据库表结构一一对应，通过 DAO 层向上传输数据源对象。

3．DTO（DataTransferObject）：数据传输对象，Service 或 Manager 向外传输的对象。

4．BO（BusinessObject）：业务对象，可以由 Service 层输出的封装业务逻辑的对象。

5．Query：数据查询对象，各层接收上层的查询请求。注意超过 2 个参数的查询封装，禁止使用 Map 类来传输。

6．VO（ViewObject）：显示层对象，通常是 Web 向模板渲染引擎层传输的对象。

7．AO（ApplicationObject）:阿里巴巴专指 ApplicationObject，即在 Service 层上，极为贴近业务的复用代码。

8．CAS（CompareAndSwap）：解决多线程并行情况下使用锁造成性能损耗的一种机制，这是硬件实现的原子操作。CAS 操作包含三个操作数：内存位置、预期原值和新值。如果内存位
置的值与预期原值相匹配，那么处理器会自动将该位置值更新为新值。否则，处理器不做任何操作。

9．GAV（GroupId、ArtifactId、Version）:Maven 坐标，是用来唯一标识 jar 包。

10．OOP（ObjectOrientedProgramming）:本文泛指类、对象的编程处理方式。

11．AQS（AbstractQueuedSynchronizer）:利用先进先出队列实现的底层同步工具类，它是很多上层同步实现类的基础，比如：ReentrantLock、CountDownLatch、Semaphore 等，它们通过继承 AQS 实现其模版方法，然后将 AQS 子类作为同步组件的内部类，通常命名为 Sync。

12．ORM（ObjectRelationMapping）:对象关系映射，对象领域模型与底层数据之间的转换，本文泛指 iBATIS,mybatis 等框架。

13．NPE（java.lang.NullPointerException）:空指针异常。

14．OOM（OutOfMemory）:源于 java.lang.OutOfMemoryError，当 JVM 没有足够的内存来为对象分配空间并且垃圾回收器也无法回收空间时，系统出现的严重状况。

15．一方库:本工程内部子项目模块依赖的库（jar 包）。

16．二方库: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）。

17．三方库:公司之外的开源库（jar 包）。

附 2：错误码列表

错误码	中文描述	说明
00000	一切 ok	正确执行后的返回
A0001	用户端错误	一级宏观错误码
A0100	用户注册错误	二级宏观错误码
A0101	用户未同意隐私协议	
A0102	注册国家或地区受限	
A0110	用户名校验失败	
A0111	用户名已存在	
A0112	用户名包含敏感词	
A0113	用户名包含特殊字符	
A0120	密码校验失败	
A0121	密码长度不够	
A0122	密码强度不够	
A0130	校验码输入错误	
A0131	短信校验码输入错误	
A0132	邮件校验码输入错误	
A0133	语音校验码输入错误	
A0140	用户证件异常	
A0141	用户证件类型未选择	
A0142	大陆身份证编号校验非法	
A0143	护照编号校验非法	
A0144	军官证编号校验非法	
A0150	用户基本信息校验失败	
A0151	手机格式校验失败	
A0152	地址格式校验失败	
A0153	邮箱格式校验失败	
A0200	用户登录异常	二级宏观错误码
A0201	用户账户不存在	

A0202	用户账户被冻结	
A0203	用户账户已作废	
A0210	用户密码错误	
A0211	用户输入密码错误次数超限	
A0220	用户身份校验失败	
A0221	用户指纹识别失败	
A0222	用户面容识别失败	
A0223	用户未获得第三方登录授权	
A0230	用户登录已过期	
A0240	用户验证码错误	
A0241	用户验证码尝试次数超限	
A0300	访问权限异常	二级宏观错误码
A0301	访问未授权	
A0302	正在授权中	
A0303	用户授权申请被拒绝	
A0310	因访问对象隐私设置被拦截	
A0311	授权已过期	
A0312	无权限使用 API	
A0320	用户访问被拦截	
A0321	黑名单用户	
A0322	账号被冻结	
A0323	非法 IP 地址	
A0324	网关访问受限	
A0325	地域黑名单	
A0330	服务已欠费	
A0340	用户签名异常	
A0341	RSA 签名错误	
A0400	用户请求参数错误	二级宏观错误码
A0401	包含非法恶意跳转链接	
A0402	无效的用户输入	

A0410	请求必填参数为空	
A0411	用户订单号为空	
A0412	订购数量为空	
A0413	缺少时间戳参数	
A0414	非法的时间戳参数	
A0420	请求参数值超出允许的范围	
A0421	参数格式不匹配	
A0422	地址不在服务范围	
A0423	时间不在服务范围	
A0424	金额超出限制	
A0425	数量超出限制	
A0426	请求批量处理总个数超出限制	
A0427	请求 JSON 解析失败	
A0430	用户输入内容非法	
A0431	包含违禁敏感词	
A0432	图片包含违禁信息	
A0433	文件侵犯版权	
A0440	用户操作异常	
A0441	用户支付超时	
A0442	确认订单超时	
A0443	订单已关闭	
A0500	用户请求服务异常	二级宏观错误码
A0501	请求次数超出限制	
A0502	请求并发数超出限制	
A0503	用户操作请等待	
A0504	WebSocket 连接异常	
A0505	WebSocket 连接断开	
A0506	用户重复请求	
A0600	用户资源异常	二级宏观错误码
A0601	账户余额不足	

A0602	用户磁盘空间不足	
A0603	用户内存空间不足	
A0604	用户 OSS 容量不足	
A0605	用户配额已用光	蚂蚁森林浇水数或每天抽奖数
A0700	用户上传文件异常	二级宏观错误码
A0701	用户上传文件类型不匹配	
A0702	用户上传文件太大	
A0703	用户上传图片太大	
A0704	用户上传视频太大	
A0705	用户上传压缩文件太大	
A0800	用户当前版本异常	二级宏观错误码
A0801	用户安装版本与系统不匹配	
A0802	用户安装版本过低	
A0803	用户安装版本过高	
A0804	用户安装版本已过期	
A0805	用户 API 请求版本不匹配	
A0806	用户 API 请求版本过高	
A0807	用户 API 请求版本过低	
A0900	用户隐私未授权	二级宏观错误码
A0901	用户隐私未签署	
A0902	用户摄像头未授权	
A0903	用户相机未授权	
A0904	用户图片库未授权	
A0905	用户文件未授权	
A0906	用户位置信息未授权	
A0907	用户通讯录未授权	
A1000	用户设备异常	二级宏观错误码
A1001	用户相机异常	
A1002	用户麦克风异常	
A1003	用户听筒异常	

A1004	用户扬声器异常	
A1005	用户 GPS 定位异常	
B0001	系统执行出错	一级宏观错误码
B0100	系统执行超时	二级宏观错误码
B0101	系统订单处理超时	
B0200	系统容灾功能被触发	二级宏观错误码
B0210	系统限流	
B0220	系统功能降级	
B0300	系统资源异常	二级宏观错误码
B0310	系统资源耗尽	
B0311	系统磁盘空间耗尽	
B0312	系统内存耗尽	
B0313	文件句柄耗尽	
B0314	系统连接池耗尽	
B0315	系统线程池耗尽	
B0320	系统资源访问异常	
B0321	系统读取磁盘文件失败	
C0001	调用第三方服务出错	一级宏观错误码
C0100	中间件服务出错	二级宏观错误码
C0110	RPC 服务出错	
C0111	RPC 服务未找到	
C0112	RPC 服务未注册	
C0113	接口不存在	
C0120	消息服务出错	
C0121	消息投递出错	
C0122	消息消费出错	
C0123	消息订阅出错	
C0124	消息分组未查到	

C0130	缓存服务出错	
C0131	key 长度超过限制	
C0132	value 长度超过限制	
C0133	存储容量已满	
C0134	不支持的数据格式	
C0140	配置服务出错	
C0150	网络资源服务出错	
C0151	VPN 服务出错	
C0152	CDN 服务出错	
C0153	域名解析服务出错	
C0154	网关服务出错	
C0200	第三方系统执行超时	二级宏观错误码
C0210	RPC 执行超时	
C0220	消息投递超时	
C0230	缓存服务超时	
C0240	配置服务超时	
C0250	数据库服务超时	
C0300	数据库服务出错	二级宏观错误码
C0311	表不存在	
C0312	列不存在	
C0321	多表关联中存在多个相同名称的列	
C0331	数据库死锁	
C0341	主键冲突	
C0400	第三方容灾系统被触发	二级宏观错误码
C0401	第三方系统限流	
C0402	第三方功能降级	
C0500	通知服务出错	二级宏观错误码
C0501	短信提醒服务失败	
C0502	语音提醒服务失败	
C0503	邮件提醒服务失败	

吉银川 研发经理

成都嗨创科技有限公司

中国西南地区专业的互联网 IT 解决方案提供商 *24 小时业务电话：17360062006（吴经理，微信同号）*

地址/Add：四川省成都市华阳剑南大道美行中心 2504(邮编:610213)

电话/Tel：(028)-83321870

手机/Mob：15181743604

邮箱/Mail：jiyinchuan@haichuang.pro

官网/Web：www.haichuang.pro