**Java代码编写规范**

# ****注释规范****

* 1. **注释的三种形式**

Java语言提供了3种形式的注释：

* //text 单行注释
* /\* text \*/ 注释若干行
* /\*\* text \*/ 文档注释。注释若干行，并可写入javadoc文档
  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。

（1）例如：类的注释：

/\*\*  
 \* description: 该文件说明  
 \* **@author** 张三（张三邮箱）  
 \* **@date yyyy-MM-dd HH:mm:ss**  
 \* **@version** 1.0  
 \*/

（2）方法的注释：

/\*\*

\* description:<方法说明>

\* **@author** 张三（张三邮箱）

\* @date **yyyy-MM-dd HH:mm:ss**

\* @param <参数名称> <参数说明>

\* @return <返回值说明>

\*/

* 1. 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。
  2. 所有的类都必须添加创建者和创建日期。
  3. 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. 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. 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。
  6. 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（代码仓库保存了历史代码）。

* 1. 【参考】对于注释的要求：第一、能够准确反应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。
  2. 【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，修改注释是相当大的负担。
  3. 【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。

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）。

# ****命名规范****

## 命名总体规则

* 名字应该能够标识事物的特性，并且与业务挂钩。
* 名字使用英文单词，尽量不用拼音，禁止直接使用中文的方式。
* 【参考】名字一般由两个或三个单词组成，尽量不要多于四个，控制在3至30个字母以内。
* 在名字中，多个单词用大写第一个字母（其它字母小写）来分隔。
* 名字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。

## 命名概述

以下几点是推荐的命名方法。

Pascal命名法：所有单词的首字母都大写，其他字母小写

Camel命名法：除首字母外，其他单词的首字母大写，其他字母小写

* 在变量名中使用互补对，如 min/max、begin/end 和 open/close。
* 布尔变量名应该包含 Is，这意味着 Yes/No 或 True/False 值，如 fileIsFound。
* 即使对于可能仅出现在几个代码行中的生存期很短的变量，仍然使用有意义的名称。仅对于短循环索引使用单字母变量名，如 i 或 j。

## 缩写

为了避免混淆和保证跨语言交互操作，请遵循有关区缩写的使用的下列规则：

* 杜绝完全不规范的缩写，避免望文不知义。
* 不要使用计算机领域中未被普遍接受的缩写。
* 在适当的时候，使用众所周知的缩写替换冗长的词组名称。例如，用 UI 作为 User Interface 缩写，用 OLAP 作为 On-line Analytical Processing 的缩写。
* 在使用缩写时，对于超过两个字符长度的缩写请使用 Pascal 大小写或 Camel 大小写。但是，应当大写仅有两个字符的缩写，如，System.IO，而不是 System.Io。

## 包

* 使用全小写命名。
* 一般以com，edu，gov，mil，net，org，或1981年ISO 3166标准所指定的标识国家的英文双字符代码开头，后续部分根据不同机构各自内部的命名规范而不尽相同，如可以是公司、项目、机器、注册名等。如com.zuche.price、com.apple.quicktime.v2、edu.cmu.cs.bovik.cheese。
* 不可含有除英文、数字外的其他字符（“.”除外），如任何特殊字符“\_”，“，”等。
* 【推荐】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。

## 类

* 使用 Pascal 大小写。
* 【参考】使用全称避免缩写，除非缩写已是一种公认的约定，如URL、HTML。对于某个命名空间、文件夹下类名中单词都比较长的情况，可以缩写（名字中的关键字不能缩写，其他单词只取首字母），将所有的缩写做一个说明文件，放在同级目录下。
* 不使用下划线“\_”。
* 抽象类命名使用Abstract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。

（1）对于Service和Dao类，实现类用Impl的后缀与接口区别。

例如：UserServiceImpl实现UserService接口；UserDaoImpl实现UserDao。

（2）命名规约

1） Bean：数据对象，xxxBean，xxx即为数据表名。

2) BO（Business Object）：业务对象。由Service层输出的封装业务逻辑的对象。

3） VO（Value Object）：返回值对象，通常为各层的返回对象。

4) Form：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。

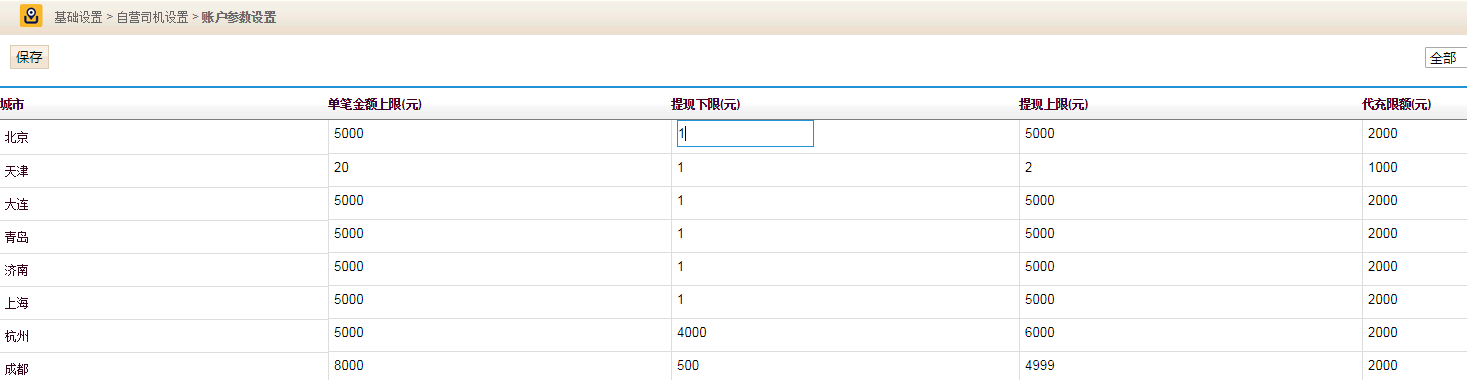
5） DTO (Data Transfer Object)：数据传输对象 xxxDTO，远程接口入参统一以DTO结尾。

6) RE : 远程接口返回值后缀。

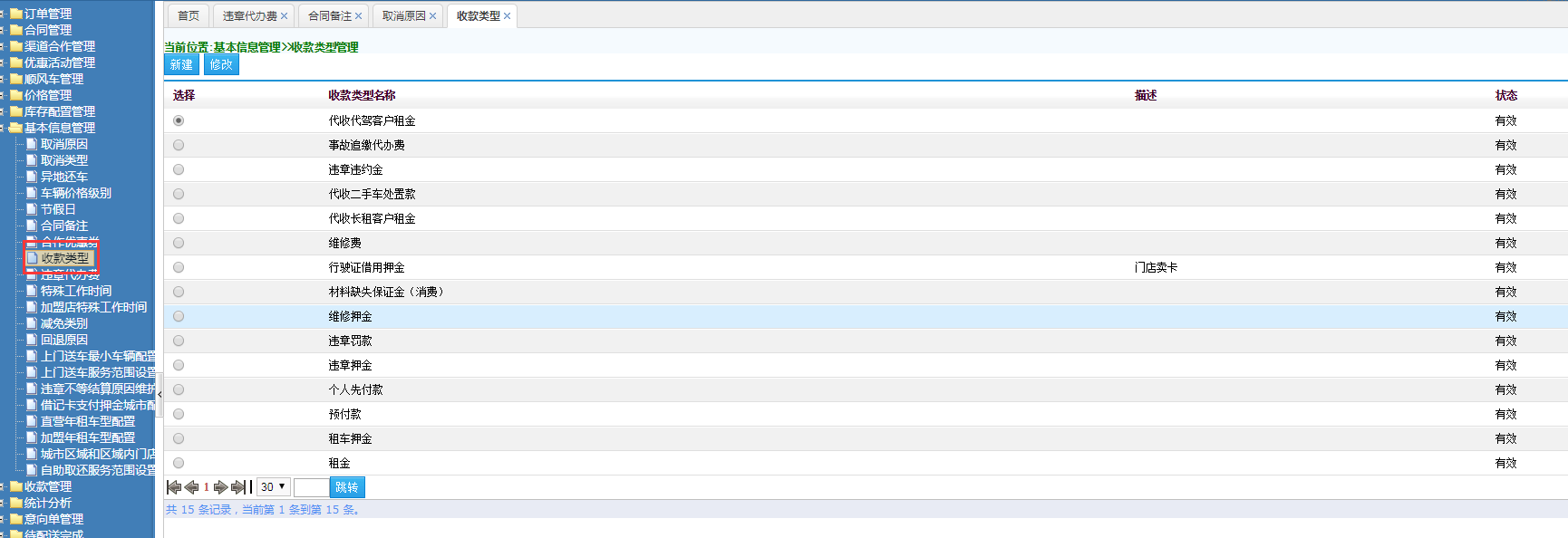
7） POJO是Bean/DTO/BO/VO/Form等的统称，禁止命名成xxxPOJO。

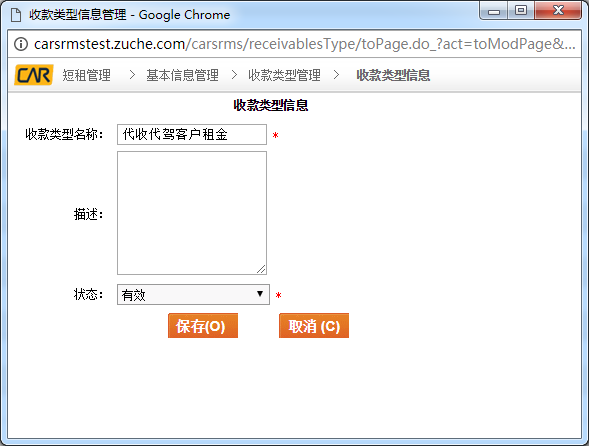
**说明：**

1. 一般只涉及单表的操作出入参使用xxxBean。以下列举几个可能的业务场景
   1. 列表页面可直接修改数据



* 1. 一些简单的数据配置查询、修改、详情等功能





* 1. ……

1. 复杂的页面、功能，通常入参为xxxForm，出参为xxxVO。例如：
   1. 查询条件较多，已超出单表字段范围。
   2. 查询条件与列表差异较大。如条件有开始时间、结束时间，但列表只会有一个时间。
   3. ……
2. 遇到业务逻辑较复杂的场景，可以在service层创建BO对象，BO对象为包含多个VO对象的复杂对象，尽量不要添加单一的属性（id类属性除外）。

## 接口

* 使用 Pascal 大小写。
* 不使用下划线“\_”。
* 【推荐】 如果是形容能力的接口名称，取对应的形容词做接口名（通常是–able的形式）。

正例：AbstractTranslator实现 Translatable。

## 枚举(Enum)

* 命名使用 Pascal 大小写，不使用下划线“\_”。
* 命名建议带上Enum后缀。 统一用包装类
* 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 / UNKOWN\_REASON。

## 参数

* 参数名称使用Camel大小写。
* 参数名称一般以参数类型名称命名，或是类型名称中的某几个单词，或是单词缩写。

## 方法

* 以动词开头。
* 使用Camel大小写。
* 尽量避免缩写。
* 不使用下划线“\_”。

1、Service/Dao层方法命名规约 ：

1） 获取单个对象的方法用get做前缀。 参考：getXXX()

2） 获取多个对象的方法用get做前缀,以List标明结果为多条。

参考：getXXXList getXXXListById

3） 获取统计值的方法用get做前缀，以Count标明统计值。

参考：getXXXCount getXXXCountById

4） 插入的方法用insert做前缀。 批量插入用insertBatch做前缀。

5） 逻辑删除的方法用delete做前缀。物理删除的方法用remove做前缀（理论上禁止物理删除）。

6） 修改的方法用update做前缀。（仅Dao层限制）

**备注**：当且仅当根据单个条件去执行操作的方法，方法名后缀为ByXXX。根据两个或多个条件去执行操作的方法不加ByXXX。

**参考**：getUserById getUserListById deleteUserById updateUserById

2、Service层推荐方法名前缀：

check convert calculate init format exec invoke match compare join

process filter split add copy clone ……

## 属性

* 以名词或形容词命名。
* 使用Camel大小写。
* 不使用下划线“\_”。

## 变量

* 变量名均采用Camel大小写。
* 变量名不应以下划线或美元符号开头。
* long或者Long初始赋值时，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。
* 按作用域定义

## 常量(const)

* 全部大写，单词间以“\_”分隔，力求语义表达完整清楚，不要嫌名字长。

常量的复用层次有五层：跨应用共享常量、应用内共享常量、子工程内共享常量、包内共享常量、类内共享常量。

1） 跨应用共享常量：放置在zuchebase项目中，通常是常量所属的项目对应包中的constant目录下。

2） 应用内共享常量：放置在本应用中，通常是common中的constant目录下。

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（如果该包下没有子包，则常量类直接方法该包下即可）。

5） 类内共享常量：直接在类内部private static final定义。

## 静态变量

* 变量名不应以下划线或美元符号开头。
* 建议全部大写，单词间以“\_”分隔，如果变量类型不是基本数据类型可以采用Camel大小写。如自定义类：public static ToolsUtil toolUtil。

## 集合

* 变量名均采用Camel大小写。
* 变量名使用复数。

## 范型

* 以一个大写字母（建议优先使用T）表示类的类型，以一个小写字母（如：t）表示类名。

# ****编码规则****

## 文件编码规则

* 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。

## 缩进规则

* 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。

## 换行规则

* 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。  
  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） 方法调用时，多个参数，需要换行时，在逗号后进行。  
  5） 在括号前不要换行。

## 空格规则

* 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。
* 注释的双斜线与注释内容之间有且仅有一个空格。
* 方法参数在定义和传入时，多个参数逗号后边必须加空格。

## 空行规则

* 方法体内的执行语句组、变量的定义语句组、不同的业务逻辑之间或者不同的语义  
  之间插入一个空行。相同业务逻辑和语义之间不需要插入空行。
* 注释与它注释的语句间不空行，但与其他的语句间空一行。
* 文件之中不得存在无规则的空行，比如说连续十个空行。空行是为了将逻辑上相关联的代码分块，以便提高代码的可阅读性。

## 大括号规则

* 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。

例如：

if (expression) {

}

## 小括号规则

* 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。
* if/for/while/switch/do 等保留字与括号之间都必须加空格。

## 模块化规则

某一功能，如果重复实现一遍以上，即应考虑模块化，将它写成通用函数。并向小组成员发布。同时要尽可能利用其它人的现成模块。

# ****异常日志****

## 异常处理

1. 【强制】异常不要用来做流程控制，条件控制，因为异常的处理效率比条件分支低。
2. 【强制】对大段代码进行try-catch，这是不负责任的表现。catch时请分清稳定代码和非稳定代码，稳定代码指的是无论如何不会出错的代码。对于非稳定代码的catch尽可能进行区分异常类型，再做对应的异常处理。
3. 【强制】捕获异常是为了处理它，不要捕获了却什么都不处理而抛弃之，如果不想处理它，请将该异常抛给它的调用者。最外层的业务使用者，必须处理异常，将其转化为用户可以理解的内容。
4. 【强制】有try块放到了事务代码中，catch异常后，如果需要回滚事务，一定要注意手动回滚事务。
5. 【强制】finally块必须对资源对象、流对象进行关闭，有异常也要做try-catch。

说明：如果JDK7及以上，可以使用try-with-resources方式。

1. 【强制】不能在finally块中使用return，finally块中的return返回后方法结束执行，不会再执行try块中的return语句。
2. 【强制】捕获异常与抛异常，必须是完全匹配，或者捕获异常是抛异常的父类。

说明：如果预期对方抛的是绣球，实际接到的是铅球，就会产生意外情况。

1. 【推荐】方法的返回值可以为null，不强制返回空集合，或者空对象等，必须添加注释充分说明什么情况下会返回null值。调用方需要进行null判断防止NPE问题。

说明：本手册明确防止NPE是调用者的责任。即使被调用方法返回空集合或者空对象，对调用者来说，也并非高枕无忧，必须考虑到远程调用失败、序列化失败、运行时异常等场景返回null的情况。

## 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。

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

private static final Logger logger = LoggerFactory.getLogger(Abc.class);

* 1. 【强制】对trace/debug/info级别的日志输出，必须使用条件输出形式或者使用占位符的方式。

说明：logger.debug("Processing trade with id: " + id + " and symbol: " + symbol); 如果日志级别是warn，上述日志不会打印，但是会执行字符串拼接操作，如果symbol是对象，会执行toString()方法，浪费了系统资源，执行了上述操作，最终日志却没有打印。

正例：（条件）

if (logger.isDebugEnabled()) {

logger.debug("Processing trade with id: " + id + " and symbol: " + symbol);

}

正例：（占位符）

logger.debug("Processing trade with id: {} and symbol : {} ", id, symbol);

* 1. 【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(各类参数或者对象toString + "\_" + e.getMessage(), e);
  2. 【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？

# ****数据库命名规则****

## 建库规约

数据库名应以【业务线代码\_功能或特性的英文或缩写】为数据库命名实例，字母应为小写，赋以下划线。功能特性的英文或缩写必须是单数名词，不可用复数形式的名词或其他时态、语态的动词。推荐库名与应用名称一致。

例如 ucar\_crm, ucar\_driver。

业务线代码：

神州租车 car

神州专车 ucar

神州卖买车 mmc

神州闪代 fcar

出租车 tcar

神州优车 ucarinc

架构组数据 archit

分时租赁 scar

## 建表规约

1. 【强制】表字符集一定要和现有库上的表保持一致，默认utf8，默认存储引擎为INNODB。
2. 【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（1 表示是， 0 表示否） 。  
   说明： 任何字段如果为非负数，必须是 unsigned。  
   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。
3. 【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。
4. 【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Bean类名也是单数形式，符合表达习惯。
5. 【强制】表命名规约：t\_项目名称缩写\_具体功能，基础表加b区分，关系表加r区分，如需新建前缀，需提前和DBA沟通。

例如：t\_price\_xxx t\_r\_price\_xxx

t\_b\_city，t\_b\_为前缀t 为table的简写，b为 base（拆库共享表）的简写，city为表的功能，为记录城市信息的功能表。

1. 【强制】禁用保留字，如 desc、 range、 match、 delayed 等， 请参考 MySQL 官方保留字。
2. 【强制】 主键索引名为 pk\_字段名； 唯一索引名为 uniq\_字段名； 普通索引名则为 idx\_字段名。  
   说明： pk\_ 即 primary key； uniq\_ 即 unique key； idx\_ 即 index 的简称。
3. 【强制】小数类型为 decimal，禁止使用 float 和 double。  
   说明： float 和 double 在存储的时候，存在精度损失的问题，很可能在值的比较时，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分开存储。
4. 【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。
5. 【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。
6. 【强制】表必备六字段：id, create\_emp, create\_time,modify\_emp,modify\_time,remark（varchar 255）。

说明：id必为主键，类型为 unsigned bigint、单表时自增或者由全局ID生成器生成的bigint递增ID。 create\_time,modify\_time 的类型均为 date\_time 类型。remark 为备注信息。另外如需要数据状态字段，统一使用status。

1. 【强制】每个字段必须有comment 注释，并且为中文注释，如遇status或类似用数字值来代表独特含义的字段，必须将所属数字的特殊含义表达清楚，例如 字段status,值 1 代表上班，2 代表下班必须在comment中注明。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。
2. 【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 varchar 超长字段，更不能是 text 字段。  
   正例： 会员名称使用频率高， 字段长度短，名称基本一成不变， 可在相关联的表中冗余存储，避免关联查询。
3. 【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。  
   正例： 如下表，其中无符号值可以避免误存负数， 且扩大了表示范围。  
   对象 年龄区间 类型 字节 表示范围  
   人 150 岁之内 unsigned tinyint 1 无符号值： 0 到 255  
   龟 数百岁 unsigned smallint 2 无符号值： 0 到 65535  
   恐龙化石 数千万年 unsigned int 4 无符号值： 0 到约 42.9 亿  
   太阳 约 50 亿年 unsigned bigint 8 无符号值： 0 到约 10 的 19 次方

## 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。

说明： 不要以为唯一索引影响了 insert 速度，这个速度损耗可以忽略，但提高查找速度是明显的； 另外，即使在应用层做了非常完善的校验控制，只要没有唯一索引，根据墨菲定律，必然有脏数据产生。

1. 【强制】超过五个表禁止 join。需要 join 的字段，数据类型必须绝对一致； 多表关联查询时，保证被关联的字段需要有索引。  
   说明： 即使双表 join 也要注意表索引、 SQL 性能。
2. 【强制】在 varchar 字段上建立索引时，必须指定索引长度，没必要对全字段建立索引，根据实际文本区分度决定索引长度即可。  
   说明： 索引的长度与区分度是一对矛盾体，一般对字符串类型数据， 长度为 20 的索引，区分度会高达 90%以上，可以使用 count(distinct left(列名, 索引长度))/count(\*)的区分度来确定。
3. 【强制】页面搜索严禁左模糊或者全模糊，如果需要请走搜索引擎来解决。  
   说明： 索引文件具有 B-Tree 的最左前缀匹配特性，如果左边的值未确定，那么无法使用此索引。
4. 【推荐】如果有 order by 的场景，请注意利用索引的有序性。 order by 最后的字段是组合索引的一部分，并且放在索引组合顺序的最后，避免出现 file\_sort 的情况，影响查询性能。  
   正例： where a=? and b=? order by c; 索引： a\_b\_c  
   反例： 索引中有范围查找，那么索引有序性无法利用，如： WHERE a>10 ORDER BY b; 索引a\_b 无法排序。
5. 【推荐】利用覆盖索引来进行查询操作， 避免回表。  
   说明： 如果一本书需要知道第 11 章是什么标题，会翻开第 11 章对应的那一页吗？目录浏览一下就好，这个目录就是起到覆盖索引的作用。  
   正例： 能够建立索引的种类：主键索引、唯一索引、普通索引，而覆盖索引是一种查询的效果，用 explain 的结果， extra 列会出现： using index。
6. 【推荐】利用延迟关联或者子查询优化超多分页场景。  
   说明： MySQL 并不是跳过 offset 行，而是取 offset+N 行，然后返回放弃前 offset 行，返回N 行，那当 offset 特别大的时候，效率就非常的低下，要么控制返回的总页数，要么对超过特定阈值的页数进行 SQL 改写。  
   正例： 先快速定位需要获取的 id 段，然后再关联：  
   SELECT a.\* FROM 表 1 a, (select id from 表 1 where 条件 LIMIT 100000,20 ) b where a.id=b.id
7. 【推荐】 SQL 性能优化的目标：至少要达到 range 级别， 要求是 ref 级别， 如果可以是 consts最好。  
   说明：  
   1） consts 单表中最多只有一个匹配行（ 主键或者唯一索引） ，在优化阶段即可读取到数据。  
   2） ref 指的是使用普通的索引（ normal index） 。  
   3） range 对索引进行范围检索。  
   反例： explain 表的结果， type=index，索引物理文件全扫描，速度非常慢，这个 index 级别比较 range 还低，与全表扫描是小巫见大巫。
8. 【推荐】建组合索引的时候，区分度最高的在最左边。  
   正例： 如果 where a=? and b=? ， a 列的几乎接近于唯一值，那么只需要单建 idx\_a 索引即可。  
   说明： 存在非等号和等号混合判断条件时，在建索引时，请把等号条件的列前置。如： where a>? and b=? 那么即使 a 的区分度更高，也必须把 b 放在索引的最前列。
9. 【推荐】 防止因字段类型不同造成的隐式转换， 导致索引失效。
10. 【参考】创建索引时避免有如下极端误解：  
    1） 宁滥勿缺。 认为一个查询就需要建一个索引。  
    2） 宁缺勿滥。 认为索引会消耗空间、严重拖慢更新和新增速度。  
    3） 抵制惟一索引。 认为业务的惟一性一律需要在应用层通过“先查后插”方式解决。

## 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(distinct col) 计算该列除 NULL 之外的不重复行数， 注意 count(distinct col1, 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： SELECT IF(ISNULL(SUM(g)),0,SUM(g)) FROM table;
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。
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：以学生和成绩的关系为例，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】数据订正时，删除和修改记录时，要先 select，避免出现误删除，确认无误才能执行更新语句。
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。
10. 【参考】 如果有全球化需要，所有的字符存储与表示，均以 utf-8 编码，注意字符统计函数的区别。  
    说明：  
    SELECT LENGTH("轻松工作")； 返回为 12  
    SELECT CHARACTER\_LENGTH("轻松工作")； 返回为 4  
    如果需要存储表情，那么选择 utfmb4 来进行存储，注意它与 utf-8 编码的区别。
11. 【参考】 TRUNCATE TABLE 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： TRUNCATE TABLE 在功能上与不带 WHERE 子句的 DELETE 语句相同。

## 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。
2. 【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。
3. 【强制】 iBATIS 自带的 queryForList(String statementName,int start,int size)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 = new HashMap<String, Object>();  
   map.put("start", start);  
   map.put("size", size);
4. 【强制】不允许直接拿 HashMap 与 Hashtable 作为查询结果集的输出。  
   说明： resultClass=”Hashtable”， 会置入字段名和属性值，但是值的类型不可控。
5. 【强制】更新数据表记录时，必须同时更新记录对应的 modify\_emp,modify\_time 字段值为当前时间（修改时间使用数据库提供的获取日期的函数赋值）。
6. 【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 update table set c1=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。
7. 【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。
8. 【参考】 <isEqual>中的 compareValue 是与属性值对比的常量，一般是数字，表示相等时带上此条件； <isNotEmpty>表示不为空且不为 null 时执行；

# ****远程服务规范****

## 远程服务类

项目提供的远程服务类统一放在本项目的remote包下，远程方法名称必须和服务名中的方法名一致。

比如：服务名为carprice.price.carsrms.queryOrderPrice 类中的方法名为queryOrderPrice

服务名为carprice.price.carsrms.queryOrderPriceV2 类中的方法名为queryOrderPriceV2

## 服务命名

* 服务端项目名(carxxx).模块名(module).客户端项目名(carxxx).操作(update,insert,get,delete...)+资源(orderDetail,orderList...)+版本(V2,V3...) . 一个服务只能给一个系统调用。

为了美观，默认V1版本不加后缀，若一个服务需要提供V2版本，则在最终的资源后增加版本号。

举例：carprice.demo.carsrms.getDemo carprice.demo.carsrms.getDemoV2

* 定时任务的服务项目名称为task，其他不变。比如：carprice.demo.task.getDemo。
* 一个服务只能给一个系统调用。同样的服务内容需要提供给多个系统时，需要包装成多个服务。

## 入参

为保证服务后续扩展和属性描述规范，一律使用DTO对象进行传输。

每个服务使用专用DTO,不得复用。

入参类型名称取服务名称的操作+资源+版本+DTO后缀命名。

DTO内部属性命名规则同数据库命名规则并写全属性注释。

所有DTO放在zuchebase项目中，包名为： 服务端项目名(carxxx).模块名(module).客户端项目名(carxxx)

举例：服务 carprice.demo. carsrms.getDemo 的 入参为 GetDemoDTO，位于carprice.demo. carsrms包下。

服务 carprice.demo. carsrms.getDemoV2 的 入参为 GetDemoV2DTO，位于carprice.demo. carsrms包下。

## 返回结果

为保证后续服务扩展和属性描述规范，一律使用Result对象进行传输，Result对象中包含业务消息主体RE。

每个服务使用专用RE，不得复用。

返回结果名称取服务名称的操作+资源+版本+Re后缀命名。

所有RE和对应的DTO放在同一包下。

Result对象包含协议属性：

re:       业务正常返回的消息体,每个服务使用专用Re对象，不可复用。属性命名规则同数据库命名。

status: 服务状态码，为int类型，0 为成功，-1为业务异常，-2为服务异常。

code:   业务语义状态码，用于做业务语义路由。

msg:    描述信息。用于返回业务异常信息和服务异常信息，不可用于设置日志堆栈信息。

# ****其他****

## ****包结构****

每个项目都必须包含两个包：client、remote 。client 包下为本项目所有调用其他系统远程服务的client类。Remote包下为本项目为其他系统提供的所有接口类。

比如：carprice项目需要调用carsrms的接口，那么客户端类SRMSClient放在carprice项目的client.carsrms包下。

## 功能实现

1. 取消、删除等没有交互页面的功能必须弹出确认框，防止用户误操作。
2. 所有删除功能都为逻辑删除。
3. 单元测试：提供的接口务必写一份对应的单元测试。
4. Sql脚本上线时，必须按照DBA的规范提供备份、恢复脚本。