Java命名规范

- 每个概念一个词

CRUD操作	方法名约定
新增	create
添加	add
删除	remove
修改	update
查询（单个结果）	get
查询（多个结果）	list
分页查询	page
统计	count

- 严格使用对仗词

add -> remove

increment -> decrement

open -> close

begin -> end

insert -> delete

show -> hide

create -> destroy

lock -> unlock

source -> target

fist -> last

min -> max

start -> stop

get -> set

next -> previous

up -> down

old - > new

recovery -> cancel

pass -> reject

- 使用后置限制词

程序中好多表示计算结果的变量，例如：总额、平均值、最大值等。

如果用类似Total、Sum、Average、Max、Min这样的词来修改某个命名，记住要把限定词加到名字的最后，并在项目中贯彻执行，保持命名风格前后一致。

总收入: revenueTotal

总人数：peopleTotal

这种命名的优点，首先，变量名中最重要的部分，就是这一变量赋予主要含义的部分应该放在最前面，可以突出显示，并且被首先阅读到，其次，可以避免同时在程序里出现使用totalRevence和revenueTotal而产生的歧义。

推荐使用限定词后置原则，保持代码的优雅，需要注意Num限定词，放在变量名结束位置表示一个下标作用，customerNum表示是客户的序号，突出的Num这个词意，为了避免Num带来的和上面规则冲突的问题

推荐使用Count和Total来表示总数，用Id表示序号，不用Num，这样customerCount就表示客户总数，customerId表示客户的编号。

- 统一业务语言

  这个是因为如果你和业务方讨论的是一种编程语言，而设计图时使用另一种语言，程序员编写的代码又体现出来的是随意翻译、毫无章法的内容，这无疑会降低代码的可读性，导致业务、文档、架构图、代码之间不一致问题

  统一业务的语言就是为了在团队中交流，建模、代码、文档的都完全一致，这个也是**领域驱动设计**的重要概念。

  例如：业务方交流中，都把推送模块中的一个功能叫 消息转发中心，那么在后面设计 文档 编码也要一致

  这个最好有业务来统一制定。

- 统一技术语言

  有些技术语言是通用的，业内人都清楚理解其意思，我们应该尽量使用这些术语来进行命名，这些通用技术语言包括 DO 、 DAO、 DTO、 Service、 ServiceImpl、 Componet、 Repository等 ，例如在代码中看到OrderDO 和 OrderDAO 马上就能知道OrderDO中的属性就是数据库中的Order字段，对Order表进行操作的都在OrderDAO里面。当然有的公司可能使用Order和Order

- 自明的代码，无需注释

  良好的代码命名可以就是最好的注释，一般要求注释量在20%~30%，也只是对特别复杂的逻辑需要解释。要想达到代码就是文档这个目的，前提必须是其具备很好的可读性和自明性。所谓的自明性就是在不借助其他辅助注释的情况下，代码本身能向读者清楚表达自身的含义才行。

- 中间变量

  我们可以通过添加中间变量让代码变得更自明，就是将计算过程打散成多个步骤。并用有意义的变量名来命名这些中间变量，从而把隐藏的计算过程已显性化的方式表达出来。

  例如：我们通过Regex来获取字符串中的值，并放到map中。

```
Matcher matcher = headerPattern.matcher(line);
if(matcher.find()){
    headers.put(matcher.group(1),matcher.group(2))
}
```

用中间变量，可以写成如下形式：

```
Matcher matcher = headerPattern.matcher(line);
if(matcher.find()){
    String key = matcher.group(1);
    String value = matcher.group(2);
    headers.put(key,value);
}
```

中间变量的这种简单使用，可以显性的表达了第一个匹配组是key，第二个匹配组是value，代码更清晰。

只有把计算过程打散成一系列命名良好的中间值，不透明的语义也会变得透明。

- 设计模式语言

  使用设计模式也是代码自明性的重要手段，在技术人员之间使用设计模式语言可以提高沟通效率。当然，前提是大家都理解熟悉设计模式。

  我们有必要在代码上体现使用到的设计模式，这样阅读代码的人能很快领会设计者的意图。

  例如：Spring里面ApplicationListener就体现了它的设计和用处。通过命名就知道使用观察者模式，每一个注册的ApplicationListener在Application状态发生变化时，都会收到一个notify。这样我们就可以在容器初始化之后进行一些业务操作，比如：加载数据，初始化缓存。

  有比如在进行邮件营销系统中，要根据一些规则来过滤一些客户，像没有邮箱地址的客户，没有订阅关系的客户，3天内不能重复发邮件的客户等。

  在SpringMVC的DispatchServlet这个类中使用到了HandlerExecutionChain这个类，使用责任链模式，Chain表达的是责任链模式，HandlerExecution表示用来进行处理执行的逻辑。