```
优推与高效的代码,代码逻辑应该直载了当,叫缺陷增以隐藏、尽量减少依赖关系,使之
使于维护,依据某种分层战略完善错误处理代码;性略调到最优、输的引诱别人做设规
矩的优化、填出一堆混乱来、整洁的代码只做好一件事
                                                                             易于维护
                                                                              做好一件事
                                     整洁代码 ○ 什么是整洁代码 ○
                                    整洁的代码应可由作者之外的开发者阅读和增补. ◎ 便于增补
                                     整洁的代码总是看起来像是某位特别在意它的人写的. 几乎没有改进的余地 ◎ 在意代码
                                    如果每个例程都让你感到深合己意,那就是整洁代码,如果代码让编程语言看起来像是
专为解决那个问题而存在,就可以称之为漂亮的代码
                       变量。函数或类的名称应该已经答复了所有大问题。它该告诉你,它为什么存在,它做什名副其实 ③ 么事,应该怎么用,如果名称需要注释来补充,那就不算是名副其实
                               避免留下掩蔽代码本意的才错误线索
                       避免误导 ○ 避免使用与本意相悖的词
                              不用双关语
                                 如果程序员只是为了满足编译器或解释器的需要写代码,就会制造麻烦
                                       以数字(a1, a2, ...)依次命名变量
                                              假设你有一个Product类,如果还有一个ProductInfo 或ProductData类,那么它们名称虽然不同,意思却无区别
                                      使用读的出来的名称 ③ 记忆单词比记忆多个字母更简单, 更易区分
                       使用可搜索的名称 ③ 单字母名称和数字常量有个问题,就是很难在一大篇文字中找出来
                       避免使用编码 ② 增加命名的复杂性,不便于发音,容易打错 ② 反例 ③ 接口命名前面都加个I,或是加个Interface
                命名
                       不应当让读者在脑中把你的名称翻译为他们熟知的名称, 这种问题经常出现在选择是
避免思维映射 ③ 使用问题领域术语还是解决方案领域术语时
                       类名 ○ 类名和对象名应该是名次或名次短语,不应该是动词
                       方法名 💿 方法名应该是动词或动词短语
                               俚语
                       别扮可爱 ⊙ 梗
                              要使用清晰, 正式的名称来命名
                       每个概念对应一个词 © 每个概念使用统一的名称 ◎ 例如获取数据统一用get. 直询表用query ....
反例 ② get的滥用
                       添加有意义的语境 O 例如 O 对于订单 O orderName, orderStauts, orderPrice
                                           将复杂处理逻辑分解
                                     短小 © if, else, swich 减少控制语句嵌套, 层级不能太深 while, for
                           减小函数的长度
                                     只做一件事 ② 每个函数一个抽象层级 ② 基础概念与细节分开
                                   使用描述性的名称
                           清晰的命名 不用追求命名的简短
                           即通过参数的值决定行为的函数 ◎
遊免标识参数 ◎ 这种函数调用时很难阅读
                函数(方法)
                                  应该根据不同行为拆分为多个函数
                          分离指令和询问 ② 查询与修改分离
                          使用异常代替错误码
                           try/catch代码块丑魉不堪. 把错误处理与正常流程混为一读. 最后吧try/catch代码块
抽离try/catch代码块
的主体部分抽离出来, 另外形成函数
                           別重复自己 ② 重复可能是软件一切邪恶的根源
Clean Code
                             注释并不是越多越好,最好是不用注释就能读懂代码
                            用代码来阐述 ③ 将逻辑代码抽象为函数,并配上描述性的函数名称 ③ 用函数名称传达信息
                              法律信息
                             提供信息的注释
                             对意图的解释
                       好注释 ○ 網释 ○ 把某些晦涩难期的参数或返回值的意义翻译为某种可读形式。也会是有用的 ○ assertTrue(a.compareTo(a) == 0); // a == a assertTrue(a.compareTo(b)!= 0); // a!= b
                             警示 ② 用于警告其他程序员会出现某种后果的注释
                             TODO
功能性注释 ○ FIXME
               注释
                                     注意及时删除
                              喃喃自语 ○ 不知所云
                             多余的注释
                              误导性注释
                              循规蹈矩的注释 ③ JavaDoc里面对每个参数都进行说明 ⑤ 显而易见的东西不用注释
                              日志式注释
                                        /**
* Default constructor.
                                        */
protected AnnualDateRule() (
                              废话注释 ○ 显而易见的东西不要注释
                       坏注释 🤄
                              位置标记 ③ 用来标记位置的注释
                              括号右边的注释 ③ 用来说明右括号对应的块 ④ 如果发现自己想标记右括号,其实应该做的是缩短语句块
                              注释掉的代码 ② 直接把代码注释掉是讨厌的做法。如果代码不在需要,应该直接删掉 ③ 版本控制会帮我们记住代码
                              与程序无关的信息
                              注释要清楚 ② 如果注释本身还需要解释,那就太遗憾了
                              短函数不需要太多描述,为只做一件事的超函数选个好名字,通常比写函数头注释要函数(方法)头 ③ 好
                             非公共代码中的JavaDoc ② JavaDoc注释额外的形式要求几乎等同于八股文章
                               不同概念间空行
                       垂直格式 ○ 相近的概念的代码要靠近
                              被调用的函数应该放在执行调用的函数下面
                               一行代码不要太长 ② 不要超过120字符
                格式
                       使用空格 ◎ 逗号,运算符后
横向格式 ◎ 不用追求变量对齐
                              不用追求变量对齐
                             缩进
                          使用异常而非返回码
                         给出异常发生的环境说明
                        依调用者需要定义异常类
                          定义常规流程 ② 使用一种更简单的处理方式
                        別返回null值 ⊙ 返回空集合而不是null
               第一职贵原则
类应该短小 ○ 第一职贵原则
高内聚 ○ 保持内聚性就会得到许多短小的类
```