Atitit 提升可读性的艺术

目录

[1. 几大原则 2](#_Toc21254)

[1.1. 直接原则，无脑原则。。。 3](#_Toc19445)

[1.2. 避免过度设计 3](#_Toc27715)

[2. 提升可读性命名，见名知意防止翻译式注释 3](#_Toc4984)

[2.1. 适当注释 3](#_Toc18736)

[2.2. 本地化命名法 3](#_Toc6896)

[2.3. 可以使用管理命名法 多个api 比如old api，new api 3](#_Toc11820)

[2.4. 相关方法使用前缀发命名空间 3](#_Toc30577)

[2.5. 提高抽象级别，what与how相分离，优先指明what 3](#_Toc32098)

[2.6. 命名规范 参考知名api 3](#_Toc17984)

[2.7. 使用命名空间，不支持命名空间的事业类似前缀 3](#_Toc7104)

[2.8. 多个执行方法适当的排序前缀 ，按照执行顺序排列 4](#_Toc2660)

[2.9. Sql api命名法 4](#_Toc12987)

[3. 减少语法噪音 4](#_Toc22497)

[3.1. Lambda 4](#_Toc23840)

[3.2. 多使用static import导入机制。。注意函数名不要重复了。。 4](#_Toc16830)

[3.3. 不要使用try catch包裹，直接方法上throw 出去，提升可读性，减少语法噪音 4](#_Toc32643)

[3.4. 方法链模式 4](#_Toc26946)

[4. 逻辑容易理解 5](#_Toc20748)

[4.1. 大力使用dsl ，what how相互分离 5](#_Toc16395)

[4.2. 控制流和逻辑的改进 5](#_Toc32681)

[4.3. 符合人类自然语言的表达习惯 5](#_Toc23532)

[4.3.1. 条件语句中的正负逻辑： 5](#_Toc29435)

[4.4. f/else语句块的顺序 5](#_Toc29369)

[4.5. 使用return提前返回 5](#_Toc2928)

[4.6. 一个流程代码不要跨文件，跳转过多 6](#_Toc1381)

[4.7. 减少语法噪音嵌套使用lambda 与脚本 6](#_Toc26474)

[4.8. 减少跳转 内部类也不错 6](#_Toc15647)

[4.9. 减少嵌套与跳转 6](#_Toc18021)

[4.10. 减少简化层次结构，单层其实可以满足大部分架构，其次双层，多层 6](#_Toc21076)

[4.11. 匿名方法，匿名构造函数 初始化块，动态map模拟匿名类匿名方法 输出 6](#_Toc8820)

[4.12. 同步 法 优先于异步法 6](#_Toc21694)

[4.13. TPL实现多线程 任务式 6](#_Toc6804)

[5. 四. 代码组织的改进 6](#_Toc9445)

[5.1. 抽取出与程序主要目的“不相关的子逻辑” 7](#_Toc23715)

[5.2. 重新组织代码使它一次只做一件事情 7](#_Toc12998)

[5.3. 借助自然语言描述来将想法变成代码 7](#_Toc29691)

[5.3.1. 把代码分成"段落" 7](#_Toc29295)

[6. 适当分类 命名空间模式 7](#_Toc18950)

[6.1. 数据库使用分库提升可读性 7](#_Toc16092)

[6.2. 命名空间前缀发 7](#_Toc20543)

[7. 大力简化减少样板代码 7](#_Toc25685)

[7.1. 简化表达式 7](#_Toc18307)

[7.1.1. 德摩根定理 8](#_Toc5914)

[7.2. 管理代码存储区域 优先文件，函数 oo模式 8](#_Toc30366)

[7.3. 静态模块最好了 8](#_Toc42)

[7.4. 其次静态方法 ，最后oo动态方法 8](#_Toc6487)

[7.5. 大力减少实体类 8](#_Toc27685)

[8. 参数 使用通用接口 大力减少代码 8](#_Toc6936)

[9. Dsl 8](#_Toc24156)

[9.1. 表格映射表代替选择 决策树模式等 8](#_Toc4521)

[10. Other 8](#_Toc30755)

[10.1. 适当的事件驱动法 8](#_Toc7298)

[10.2. 异常模式代替返回 8](#_Toc23858)

[10.3. 方法链 8](#_Toc12841)

[10.4. 字符串模板技术 8](#_Toc19677)

[10.5. 数据结构参照标准规范，方便文档对照 各种微格式等rss 9](#_Toc21480)

[10.6. 递归代替循环 9](#_Toc3458)

[10.7. 中缀表达式 优先于 前后缀表达式 9](#_Toc14326)

[11. 各种dsl 9](#_Toc10371)

[11.1. Ognl nodejs python 9](#_Toc26123)

[11.2. Guava等lib 9](#_Toc8982)

[12. 如何提高代码的可读性? - 读《编写可读代码的艺术》 10](#_Toc24957)

# 几大原则

## 直接原则，无脑原则。。。

## 避免过度设计

代码可读性 程序语言可读性 sql可读性 json等可读性 txt可读性

库表可读性

# 提升可读性命名，见名知意防止翻译式注释

## 适当注释

## 本地化命名法

## 可以使用管理命名法 多个api 比如old api，new api

## 相关方法使用前缀发命名空间

## 提高抽象级别，what与how相分离，优先指明what

## 命名规范 参考知名api

参考知名api 参考知名sdk 游戏cocos2d、等..

Sql style api

这样可以大大减少资料文档的编撰。。互联网上已经有了

## 使用命名空间，不支持命名空间的事业类似前缀

缺点就是命名长度变长了，单是可读性优先，会提升可读性，名字长度有ide自动补全缓解。。

## 多个执行方法适当的排序前缀 ，按照执行顺序排列

## Sql api命名法

# 减少语法噪音

## Lambda

## 多使用static import导入机制。。注意函数名不要重复了。。

## 不要使用try catch包裹，直接方法上throw 出去，提升可读性，减少语法噪音

结构可读性

分层

## 方法链模式

# 逻辑容易理解

## 大力使用dsl ，what how相互分离

## 控制流和逻辑的改进

## 符合人类自然语言的表达习惯

### 条件语句中的正负逻辑：

在判断一些正负逻辑的时候，建议使用if(result)而不是if(!result)。

因为大脑比较容易处理正逻辑，比如我们可能比较习惯说“某某某是个男人”，而不习惯说“某某某不是个女人”。如果我们使用了负逻辑，大脑还要对它进行取反，相当于多做了一次处理。

## f/else语句块的顺序

在写if/else语句的时候，可能会有很多不同的互斥情况（好多个elseif）。那么这些互斥的情况可以遵循哪些顺序呢？

* **先处理掉简单的情况，后处理复杂的情况**：这样有助于阅读代码的人循序渐进地地理解你的逻辑，而不是一开始就吃掉一个胖子，耗费不少精力。
* **先处理特殊或者可疑的情况，后处理正常的情况**：这样有助于阅读代码的人会马上看到当前逻辑的边界条件以及需要注意的地方。

## 使用return提前返回

在一个函数或是方法里，可能有一些情况是比较特殊或者极端的，对结果的产生影响很大（甚至是终止继续进行）。如果存在这些情况，我们应该把他们写在前面，用return来提前返回（或者返回需要返回的返回值）。

这样做的好处是可以减少if/else语句的嵌套，也可以明确体现出：“哪些情况是引起异常的”。

## 一个流程代码不要跨文件，跳转过多

特别是脚本会很麻烦。。带来ide和大脑跳转过多。。

## 减少语法噪音嵌套使用lambda 与脚本

## 减少跳转 内部类也不错

## 减少嵌套与跳转

## 减少简化层次结构，单层其实可以满足大部分架构，其次双层，多层

## 匿名方法，匿名构造函数 初始化块，动态map模拟匿名类匿名方法 输出

匿名函数虽然没有名字，但也是可以有构造函数的，它用构造函数块来代替，那上面的3个输出就很清楚了：虽然父类相同，但是类还是不同的。

## 同步 法 优先于异步法

## TPL实现多线程 任务式

# 四. 代码组织的改进

关于代码组织的改进，作者介绍了以下三种方法:

## 抽取出与程序主要目的“不相关的子逻辑”

## 重新组织代码使它一次只做一件事情

## 借助自然语言描述来将想法变成代码

### 把代码分成"段落"

在写文章的时候，为了能让整个文章看起来结构清晰，我们通常会把大段文字分成一个个小的段落，让表达相同主旨的语言凑到一起，与其他主旨的内容分隔开来。

而且除了让读者明确哪些内容是表达同一主旨之外，把文章分为一个个段落的好处还有便于找到你的阅读“脚印”，便于段落之间的导航；也可以让你的阅读具有一定的节奏感。

其实这些道理同样适用于写代码：如果你可以把一个拥有好几个步骤的大段函数，以空行+注释的方法将每一个步骤区分开来，那么则会对读者理解该函数的功能有极大的帮助。这样一来，代码既能有一定的美感，也具备了可读性。其实可读性又何尝不是来自于规则，富有美感的代码呢？

# 适当分类 命名空间模式

## 数据库使用分库提升可读性

## 命名空间前缀发

# 大力简化减少样板代码

## 简化表达式

### 德摩根定理

## 管理代码存储区域 优先文件，函数 oo模式

## 静态模块最好了

## 其次静态方法 ，最后oo动态方法

## 大力减少实体类

# 参数 使用通用接口 大力减少代码

# Dsl

## 表格映射表代替选择 决策树模式等

# Other

## 适当的事件驱动法

## 异常模式代替返回

## 方法链

## 字符串模板技术

## 数据结构参照标准规范，方便文档对照 各种微格式等rss

数据库的schedu\_info

日程的ics

邮件eml

文章 meatawebblog

通讯录 vcf

## 递归代替循环

## 中缀表达式 优先于 前后缀表达式

# 各种dsl

## Ognl nodejs python

## Guava等lib

Atitit readablity enhance art 可读性的艺术v4 t99.docx

Atitit. 提升可读性推荐标准规范解决方案 关于编程语言的v5 docx

Atitit. 提升可读性推荐标准规范解决方案 关于编程语言的v5 docx

1. 提升可读性的意义 1

2. 提升可读性大原则： 2

2.1. 分解 分类 层次结构 2

2.2. 命名规范推荐标准 2

2.3. 关注点分离原则 2

2.4. 面向人类编程，优先于面向机器，可读性优先于性能原则 2

3. 具体措施 2

3.1. （推荐）Dsl \*\*重要 2

3.2. \*\*（推荐）使用汉字命名，获取更大的可读性，适合于绝大多数项目利大于弊（推荐） 2

3.3. （推荐）使用命名空间，不支持命名空间的事业类似前缀 3

3.4. （推荐）有时候异常处理也会提升可读性 3

3.5. （推荐）限制使用spring等框架范围，防止滥用 3

3.6. （推荐）提高抽象级别，what与how相分离，优先指明what 3

3.7. （推荐）减少架构层次，双层比三层四层架构更加简单易读 3

3.8. （推荐）注意学院派与工程派完全不同 3

3.9. 命名规范 参考知名api 3

3.10. （推荐）Sql style api 4

3.11. 适当分层、DI和AOP是继OO之后的分解方法 4

3.12. 函数式样 流程控制全部函数化 4

3.13. 递归代替循环 4

3.14. 中缀表达式 优先于 前后缀表达式 5

3.15. 防止出现大量接口，，接口过多过滥 5

3.16. 减少 嵌套级别 5

3.17. （推荐）使用模板法，关注点分离。。字符串拼接太难读怎么办？？ 5

3.18. 减少“语法噪音” 5

3.19. 参考 5

4. Refactor 5

4.1. 方法链 5

5. 参考资料 5

Atitit 可读性技术与实践范例 艾提拉著

# **如何提高代码的可读性? - 读《编写可读代码的艺术》**