本读书笔记共分为17章。

# 整洁代码

阅读这本书有2种原因，第一，你是个程序员；第二，你想成为更好的程序员。

## 要有代码

我们永远抛不掉代码，因为代码呈现了需求的细节。

## 糟糕的代码

勒布朗法则：稍后等于永不，临时的糟糕代码会毁了项目。

## 混乱的代价

随着混乱的增加，团队的生产力持续下降，趋向于零。当生产力下降时，管理层就会增加更多的人手到项目中，这些新人不懂设计，迫于压力，会制造更多的混乱。此时，管理成立新团队设计新系统；程序员有必要拒绝遵从业务人员与经理的想法且必须告知他们。

赶上期限的唯一办法就是始终保持代码整洁。

整洁的代码：优雅、效率、干净利落，直截了当。

# 第二章 有意义的命名

命名的整洁规则：

1. 名副其实，选择体现本意的名称；2.避免误导，不能起掩藏代码本意的错

误名字；3.做有意义的区分，不能随意添加字母或者数字一区分2个变量（可能类型定义相同）；4.使用读得出来的名字，名字要完整的单词；5.使用可搜索的名称，名称的长短与其作用域大小相对应；6.避免使用编码，名字中尽量不要体现出类型以便增加抽象的级别，不要使用前缀；7.避免思维映射；8.类名与对象名为名词或短语，不要用动词；8.方法名应当是动词或动词短语，属性访问器或者修改器或者谓词加上get、set或者is等；9.不要使用俚语来拜师名字；10.给每个抽象概念选一个词，并且一以贯之，这样就能进行名字联想，不用翻看代码找含义；11.避免名字用于不同目的；12.使用计算机领域名字，不要使用业务名字；13.添加名字的语境，必要时可以加前缀，或者抽象出类来。

# 第三章 函数

函数的书写规则如下：

1. 短小，函数不应该大到能够容纳嵌套，缩进层级不应该多余一层或者2层；
2. 只做一件事，函数只做该函数名级别的抽象操作，并且函数内不能再分出一个函数，差不多就是是做一件事了；
3. 每个函数一个抽象层级，自顶向下按抽象层级写函数；
4. Switch语句，放入抽象工厂中，解耦合；
5. 使用描述性的名称，取长而具有描述性的名称，并且命名方式要保持一致；
6. 函数参数，参数越少越好，方便里面处理同时测试起来也方便，不要超过3个参数；
7. 无副作用，函数的重复性使用造成的时序破坏，尽量避免使用输出参数，如果函数要修改状态，那么久修改所属对象的状态；
8. 分隔指令与询问，要么做什么事，要么回答什么事，不可得兼；
9. 使用异常替代返回错误码，抽离try/catch代码块，形成单独的函数，错误处理形成一个单独的函数；
10. 别重复自己，
11. 结构化编程

# 第四章 注释

注释是弥补用代码表达意图失败时的做法，但是注释会撒谎，随着代码的更改，注释没有得到维护，就会造成错误的后期修改；真相只有代码能告诉我们。

1. 注释不能美化糟糕的代码，要花时间清理糟糕的代码；
2. 用代码来阐述；、
3. 好注释，包括法律信息、提供信息，对意图的解释，阐释含义，警示，TODO注释揭示接下来要做的工作，放大不合理之物的重要性，写javadoc；
4. 坏注释，大多数注释都是不良注释，喃喃自语，不知所云是不好的注释；多余的注释是不好的注释；误导性注释；循规式代码比如必须写javadoc注释等规则；日志式注释；废话注释；能用函数或者变量就不要用注释；位置标记注释；括号后面的注释；归属与签名；注释掉的代码理应删除；HTML注释；非本段代码注释；信息过多；不明显的联系；函数头；非公共代码中的javadoc；范例。

# 第五章 格式

保持良好的代码格式们所有成员遵守。

1. 格式影响代码的可读性，便于维护；代码源文件大小尽量小，便于理解；对代码进行分组，不同的组之间用空行分隔开来，增加可视感；自上至下展示代码依赖，这样读到开头既可以知道函数的大概宗旨；关系密切的概念应该相互靠近；变量应该尽可能靠近使用位置；实体变量在类的顶部声明；
2. 尽力保持代码行短小，适当在表达式中添加空格，严格的对齐没什么作用，适当的缩进；

# 第六章 对象与数据结构

变量设置为私有的一个理由是：不想其他人依赖这些变量。

1. 数据抽象，隐藏底层数据不只是添加一个上层函数如set、get那么简单；就是要高度抽象，让使用者完全猜不到内部实现；
2. 数据与对象的反对称性，过程式代码便于在不改动既有数据结构的前提下添加新函数，面向对象的代码便于在不改动既有函数的前提下添加新类，这个是矛与盾的意思，各有长处短板；
3. 德墨特定理，对象不能通过存取器暴露其内部结构，类C的方法f只能调用以下对象的方法:C、有f创建的对象、作为参数传给f的对象、由C的实体变量持有的对象；方法不应该调用由任何函数返回得到的对象的方法；连续的函数调用称作火车失事，对象与数据结构混杂在一起也是非常不好的（这里还要看书）；
4. 没有函数的类就是数据结构，或者bean也算数据结构，这些都是DTO类，这些数据结构不能塞入业务规则的函数，这导致了对象与数据结构的混杂一体。

# 第七章 错误处理

过多的错误处理会搞乱代码的逻辑，需要清理。

1. 使用异常而不是返回码，这样会更整洁；
2. Try-catch-finally处理事务，try里面要尽可能抽象；
3. 使用不可控异常，因为在函数签名里声明可控异常会在整个函数调用链条上都加加上这样的函数签名；破坏了封装；
4. 给出异常发生的环境说明，尽可能多的错误信息；
5. 依调用者需要定义异常类，对复杂的异常封装打包，以保证简洁性，同时完成了解耦合；
6. 定义常规流程；
7. 别返回NULL值，返回NULL值需要处处检测，自己写的代码不要返回NULL值，可以返回大小为0的对象或者抛出异常，调用第三方API时都可以用着2种方法封装；
8. 别传递NULL值，传递NULL值需要添加大量判断或者还要定义异常，这些都影响代码整洁，直接禁止传递异常。

# 第八章 边界

保持软件边界整洁。

1. 使用第三方代码，使用第三方代码并不总是如愿解决问题，此时就要封装，以解决边界问题；
2. 浏览和学习边界，学习第三方代码的方式可以是直接测试来理解，这叫做学习性测试；
3. 学习log4j，不断进行测试，得到log4j的正确使用方法；
4. 学些性测试能够检查对新版本的兼容性;
5. 使用尚不存在的代码，意思就是可以使用接口，接口的实现可以后面定义，因为可以得到目前想要的接口的样子；
6. 整洁的边界；

# 第九章 单元测试

编写自动化单元测试。

1. TDD三定律，在编写不能通过的单元测试前，不可编写生产代码；只可编写刚好无法通过的单元测试，不能编译也算不通过；只可编写刚好足以通过当前失败测试的生产代码；
2. 保持测试整洁，测试代码是和生产代码一样重要的，否则会导致糟糕的结果；
3. 整洁的测试，
4. 每个测试一个断言（这里看下书）

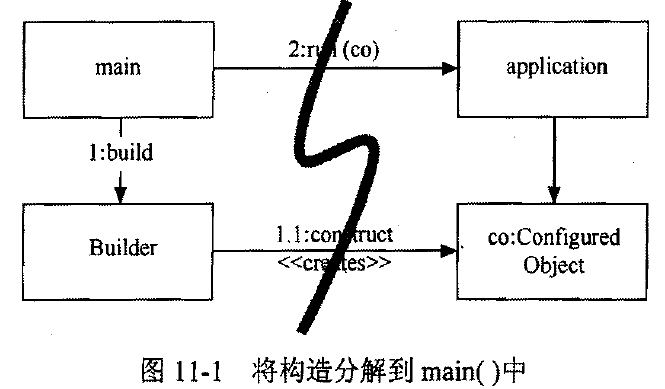
# 第十章 类

类是代码层面的更高的组织方式；

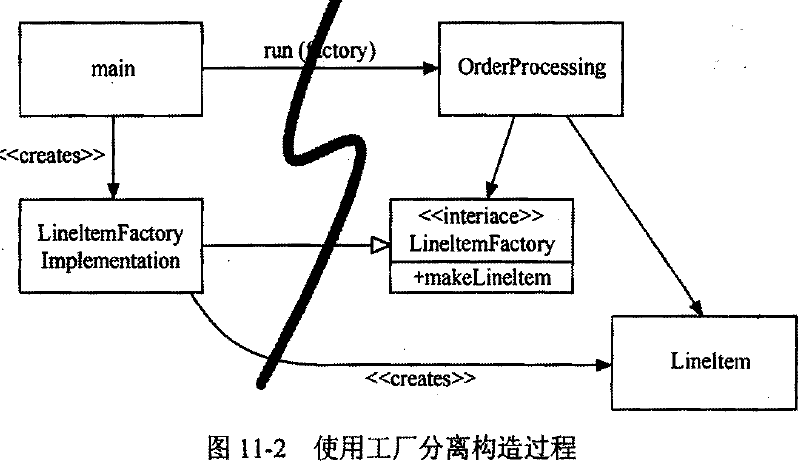
1. 类的组织：静态变量，私有变量，公共方法，私有工具函数；可以适当考虑保护以默认权限；
2. 类应该短小，函数的整洁规则是行数要少，类的整洁规则是权责要少，要明确；这是衡量类是否整洁的准则，如果类名模糊，那么权责就不会清晰，所以，类名要明确单一；单一权责原则：类只包含一个任务；保持高内聚，方法操作的变量越多，越内聚；尽量保持较少的变量；将大函数拆分为小函数，往往也是将类拆分为多个小类的时机。
3. 为了修改而组织，拓展系统而不是修改现有代码，同时代码应该依赖于抽象而不是具体细节。

# 第十一章 系统

1. 将构造与使用分离，软件系统的起始过程和运行时逻辑分离开，在起始过程中创建对象，将构造与使用分开的方法之一是将全部构造过程搬迁到main模块中，



应用程序有时也要确定合适生成对象，此时使用抽象的工厂方法生成，工厂在main中生成，实际来说还是构造与运行分离的；



最好的构造与运行分离的机制就是依赖注入，是控制反转的一种管理手段。

1. 扩容，
2. Java代理，
3. 纯Java AOP框架，

# 第十二章 迭进

1. 简单设计的4条规则：运行所有测试、不可重复、表达了程序员的意图、尽可能减少类与方法的数量；
2. 测试编写的越多，就越容易推断出类应该单一权责与抽象实现分离，且类越小，就能写更多的测试，测试到更多的细节，达到整个系统都是可测试的；
3. 重构时，如果设计退步，则清理代码并重新进行测试，重构可以使用一切软件开发过程中应该遵循的规则；
4. 不可重复；
5. 表达力，要使代码容易理解；
6. 尽量少的类或者方法；

# 第十三章 并发编程

对象是过程的抽象，线程是调度的抽象。

1. 并发是一种解耦策略，把做什么与何时做分开，在单线程应用中，目的与时机紧密耦合，解耦目的与时机能明显改善程序的吞吐量与结构；
2. 并发编程难点在于有多种执行路径，有些是正确的，有些是错误的；
3. 并发防御原则：(1)单一权责原则（SRP），分离并发相关代码与其他代码，(2)限制数据作用域，严格限制对可能被共享的数据的访问，(3)使用数据复本来代替共享对象,(4)推论：线程应尽可能的独立；
4. 了解java库；
5. 了解执行模型，几种需要并发的典型情况；
6. 警惕同步方法之间的依赖；
7. 保持同步区域微小；
8. 今早考虑线程的关闭问题；
9. 测试线程代码：(1)不要讲系统错误归咎于偶发事件，(2)不要同时追踪非线程与线程的缺陷，确保线程之外的代码可工作，(3)编写可插拔的线程代码，能在不同的配置环境下运行，(4)运行时要使线程数量可以自我调整，(5)运行多余处理器数量的线程，(6)在不同平台上都要测试，(7)装置试错代码，尽可能的暴露线程的错误执行路径，可以向代码中写入很多的硬编码代码，这样有很多缺点，建议使用自动化工具。

# 第十四章 逐步改进

# 第十五章 JUnit内幕

# 第十七章 味道与启发

1. 注释，不恰当的信息、废弃的注释、冗余注释、糟糕的注释、注释掉的代码；
2. 环境，需要多步才能实现的构建、需要多步才能做到的测试，统统要压缩到一步；
3. 函数，过多的参数、输出参数、标示参数、死函数；
4. 一般性问题，一个源文件中存在多种语言、明显的行为未被实现、不正确的边界行为、忽视安全、重复、在错误的抽象层级上的代码、基类依赖于派生类、信息过多、死代码、垂直分离、前后不一致、混淆视听、认为耦合、特性依恋、选择算子参数、晦涩的意图、位置错误的权责、不恰当的静态方法、使用解释性变量、把逻辑依赖改为物理依赖、用多态替代if/else swaitch/case、遵循标准约定；
5. 通过使用通配符避免过长的导入清单、不要继承常量使用静态导入、使用枚举代替常量；