封面

摘要

目录

1、绪论

2、开发约定思想

开发约定及思想是一个开发团队经过长期工作总结出来的最佳实践，它不是由特定平台或语言所施加的限制，而是从管理层面上对软件开发工作作出的一种补足。遵循好的约定及思想，可以增加项目各个模块的一致性，减少对各部分进行衔接时的沟通成本，减少团队的学习、培养成本，甚至可以在很大的程度上增加开发效率。

2.1 约定

1.bean命名规范及命名的一致性

java bean并不是一种编程模型，也不是一种语言机制，而是一种简单的命名规范。bean规范对类作出了以下假设：

bean必须具有一个无参构造方法。如果类没有定义构造方法，java语言默认规定会由编译器生成一个无参构造方法。如果类定义了带参数的构造方法，则默认不会生成无参构造方法。所以，建议如果将一个类作为bean，最好显示提供一个无参构造方法。

bean的getter和setter方法被当做是bean类的属性。如getBeanProperty()、setBeanProperty()方法给定义他们的类添加了一个叫做beanProperty的属性。

可选的与属性同名的类字段。bean规范并没有对存储属性的字段提出要求。也就是说，存储属性beanProperty的类字段可以有任意的命名和类型。但是，为了减少命名不一致，增加代码的可读性，建议字段名和属性名一致。

bean给java类带来了属性，并提供了一些附加功能。首先，bean可以作为值对象（value object）用来在不同的类之间传递数据；其次，bean能对内部实现进行屏蔽，对属性的访问提供一个间接层，在间接层中加入额外的限制；最后，遵循bean命名规范的类是很多框架的宠儿，这些框架多使用java反射机制对bean进行操作，包括bean的创建，属性的注入等，这些操作符合约定优于继承的思想。

bean最基本和最重要的功能还是作为值对象。只要符合了上述关于bean的三个约定，就可以作为值对象，但并不要求只作为值对象，而不具备任何的行为。在使用场景中，值对象不仅是数据容器，还可以作为一种接口，相当于对外部用户使用的数据结构进行的一个限定。如方法“void operate(SomeBean someBean);”指明了以类SomeBean作为给方法发送信息的一种数据结构。这样做的好处是，如果bean使用是pojo（Plain Old Java Object，简单java对象）或者属性里包含jdk提供的基础数据结构（如集合框架），那么将不会不损害接口的简单性；除此以外，这样的接口更加容易修改，在bean中增减属性并不会引发接口的变动，只需要对方法的实现加以修改即可。如果软件项目的各个模块在需求分析、软件设计的阶段，或者极限编程活动当中搭建“可以行走的骨架”阶段，就确立好各部分要使用的bean，就可以帮助各个层的开发人员更好地理解业务需求。经过实践，遵循上述约定开发的业务模块更容易修改。

2.类层次分明且接口清晰

类层次的结构设计应该考量好业务和功能。业务始终是依赖于多个功能进行实现的，功能是实现业务的手段，但是功能是不依赖于业务而独立存在的。把功能从业务中独立出来，可以改善程序的结构，增加程序的可读性。接口清晰，是指模块的输入输出要明确，特别的，在java平台中可以使用bean和方法清晰地刻画功能模块和业务模块的接口。

3.实现的简单性

尽可能保持实现的简单性，到再也无法保持为止。简单的实现包括设计的简单性、实现的简单性、模块的简单性、关系的简单性、接口的简单性等。保持设计简单，得益于清晰的业务需求，需求越清晰，划分的模块规模就越小、功能就越明确；保持实现的简单，要求不对业务做多余的假设，采用合乎要求、对已有环境影响较小的技术，消除代码中的重复等；保持模块的简单性，包括功能的单一，接口的清晰等；关系的简单性，要求处理好各部分的关系，保持单向的依赖，拒绝循环依赖等；接口的简单性，要求从外部看去，各模块封装良好，没有模板代码，对调用而言是友好的。

4.不作多余的假设

约定背景：瀑布开发模型适用于需求明确的场景，但是实际开发过程中很少有需求真正明确的时候，大多数情况下客户都不太清楚自己真正想要的是什么。作为项目中基础模块的使用者，我们是自己的客户，当然不希望面对一个复杂的、难以修改的基础框架，我们需要的是可以简单地使用并且能够应付一定变化的结构。

使用场景：当不是真正清楚自己是否需要某个看起来很炫酷的功能时，尽量不作多余的假设。

约定意义：不作多余的假设，就是避免“夸夸其谈的未来可能性”。一旦假设自己在未来的某个时刻需要某种功能，或者假设自己应该给某个功能留下一个可以插入的接口，那么带来的困境将是实现上的复杂性和软件结构的复杂性。时刻保持简单，只在需要的时候才加入相应的功能十分重要。

良好实践：例如，当你预料到某个功能可能在将来有新的实现时，会留下一个接口方便未来的扩展，但实际上当前只有一个实现。这个时候可以果断地抛弃留下接口的念头，因为很有可能这个接口再也不会拥有多于一个的实现了。如果保持了简单性，完全可以通过重构来加入新的接口。

5.可读性很重要

在软件测试领域，有一种开发方法叫做测试驱动的面向对象开发。该方法是测试驱动的开发（TDD，Test-Driven Development）在面向对象领域的延伸，除了要求测试先行以外，该方法还有一系列技术（如junit，mock技术）去探测对象的行为。当采用这种开发方法时，写出来的测试就如同自然语言一般，清晰易懂。例如，对于testIllegalLoginWithWrongPasswordAndFailed方法，说明了测试的内容及结果，该方法名就是最好的注释。

由于c语言根深蒂固的影响以及相关材料随意性，很多c风格的代码都难以读懂。著名的循环变量i，j就是该语言发扬光大。当然，现在依旧被广泛使用，但是python语言的发明者显然对这类命名深恶痛绝，直接从语言层面上消除了使用这种for循环的苗头。

汇编语言是机器的最爱，然而高级语言是面向作为开发者的人的，不好的命名和模块化让代码错失了与人交流的机会。在java这类完全面向对象的语言之中，包名是第一个“表述你自己”的机会，类名是第二个，此外是字段名、方法名，最后才是注释（包括类注释、字段注释、方法注释）。好的命名基本上可以做到零注释。值得注意的是，一些需要行内注释的地方，往往都是隐错存在的地方。

可读性的前提是良好的设计，如果类的设计不好，各个结构之间耦合严重，层次不清晰，就意味着可读性必然会受到影响。提高可读性的最好原则是，不放过前述任何一个可以解释程序做了什么的机会。如果能做到即使是局部变量的命名也不轻易的使用tempN的方式，那么代码的维护者（很可能就是你自己）在两个月后将从中受益。关于注释，一个好的实践是，书写良好的类注释，说明类使用的上下文，依赖，采用的技术等具体事项。

6.不断重构以改善软件的结构

重构，是对软件内部结构的一种调整，目的是在不改变软件可观察行为的前提下，提高其可理解性，降低其修改成本。在《重构-改善既有代码的设计》一书中，Martin Fowler提出了一系列重构的手法，用于在不改变软件可观察行为的前提下，调整其结构。重构虽然是极限编程下的一员“猛将”，但是基本上所有的开发方法都离不开重构。

在重构的世界里，满足设计模式不是最初设计的指导原则，而是一种重构以后软件应该达到的目标。在编程的过程当中，由于需求变化、设计失误、第三方包变化等原因，代码面临着结构失控的风险。新功能的加入，接口的改变都势必影响到程序的可读性和可拓展性。重构的存在可以在关键的时刻挽救逐渐混乱的代码结构，让代码重新满足工程上的最佳实践——也就是设计模式——的要求。

失控的代码往往都存在着“坏味道”。重构就是用来消除代码中的“坏味道”。如重复代码、过长函数、过大的类、发散式变化、霰弹式修改、冗赘类、夸夸其谈的未来性、令人迷惑的暂时字段、过渡耦合的消息链等等，都是代码的“坏味道”。代码“坏味道”的存在是程序存在设计失误、编码失误的信号。

重构不是高端的词汇，而是程序员每天都在做的事情。重构的手法大多由一系列小的操作完成，如：改变方法的签名，移动方法到其他类，抽取出新的类，封装字段等等看似平凡的操作，都是强而有力的重构手法。

2.2 思想

1.模块化的思想

模块是组织代码的良好方式。这里的模块不仅仅是指代码模块，在java中，项目、包、类、方法都可以作为模块。如，spring项目由一个个子项目模块组成。模块化是面向对象的基础。接受模块化思想，意味着要屏蔽自然人根深蒂固的过程化的思想。考虑问题的时候，要更多地考虑“这个功能可以由下面的功能组成”，多过考虑“完成这个功能要先这么做，再那么做”。当然，人在缺乏经验的时候不可能仅凭观察就将一个模块细化，分解为若干子模块，但是我们拥有重构的手法，使我们可以事后再进行“补救”，最终符合模块化的要求。模块化的代码更加可读，更好维护，更易得到重用的机会。

2.面向对象的思想

从形式上，面向对象要求数据和操作数据的方法一起进行组织，这符合客观世界的特点和人类的思考方式。从客观实际中有针对性地抽象出我们关心的部分，就是面向对象。依据面向思想的方法，很容易构建出高内聚、低耦合的程序。

3.函数式编程的思想

函数式编程要求我们把函数当做一个加工厂，而数据经过函数的映射不断地变换形式，最终得到我们需要的格式。例如，在java语言中，有bean类A、B、C及其对象a、b、c，那么链式调用：

a.toB().toC();

最终将a转换成了c，就是一种函数式编程。函数式编程要满足以下两个要求：

函数是无副作用的。也就是说，函数不会改变所处理的数据的内部状态。

数据是不可变的。摒弃对数据的可写性，虽然多了需要四处复制数据的风险，但是好处是不可变的数据是无状态的，数据高度一致。一经创立就不可改变的数据，比起在一处创立被多处改变的数据显然更加友好。

和函数式编程相对应的是命令式编程。和上面进行相同的假设，代码：

a.f1(b);

a.f2(c);

就是一种命令式的风格。该风格符合面向对象的处理方式，a分别依赖于b和c来完成功能。函数式编程相当于对编程施加了更多的限制。但使得调用更加简便、程序更易阅读和维护。在java 8以前java并不支持函数式编程风格，但在java 8中引入了lambda表达式和函数接口的特性完美支持了强类型环境下的函数式编程。此外，jdk 8中还提供了相应的函数式api，大大减少了书写模板代码的工作量。

4.约定优于配置的思想

约定是指按照既定的习惯做法来做事，配置是提供配置的接口，在使用前设值进行配置。约定优于配置（convention-over-configuration）是指，更多地提前采用经受了实践考验的约定配置，从而减少甚至省略每次使用时的冗余配置。

在软件开发中，可以提前约定的地方包括项目架构的，编程接口等。在基于maven构建的项目中，源代码存在于目录src/java/main中，测试代码位于src/test/main中，就是一种约定。两个目录下的目录结构一直（同名的目录属于相同的包），绕开了java数据可见性的限制，将源代码和测试代码分开组织，方便代码的发布。有关于编程接口的约定，可以提前给用于配置的变量赋默认值。

对于javascript等动态、弱类型的语言，约定的使用更加方便。当javascript对象具有相同的方法签名或者同名的变量时，他们就遵循了相同的约定，由于弱类型的原因，可以使用同一个函数统一进行处理。

当然，约定降低了代码的可读性，在程序中引入“约定优于配置”的代码时，要充分地写好文档注释，并提供测试样例，防止维护时摸不着头脑。

5.封装与暴露接口

封装是指隐藏具体的实现细节，向用户暴露一个清晰简洁的外部接口。spring jdbc template技术就是一种对模板代码的封装。封装简化了调用，增强了程序的可读性，改善了代码结构。

6数据视图

同一组数据在不同的层次有不同的表现形式，这一组表现形式就是数据的视图。如，用于存储的视图、业务计算的视图和展示的视图。在工程领域，用于描述客观世界的模型数据是核心的视图，其他诸如存储方式和表现形式的视图都要依据这个视图来确立。数据视图同时是程序代码内部进行通信的协议所在。

7.极限编程

快速确立程序的可行走的骨架，不断地得到程序给予的反馈。极限编程适用于有工程背景的开发环境。当需求不明确或者经常变动时，应该使用极限编程的思想，尽快为客户提供原型，从而验证和调整需求，使软件开发工作符合实际情况。

3、软件环境搭建

从软件项目的开发周期来看，需要处理好诸如项目构建、版本控制、软件开发、软件测试、部署与发布等工作。过去的十五年间，java平台下涌现了大量优秀的开源框架，在项目中合理使用这些框架能大大提高开发效率，使我们可以专注于业务开发，而将其他细节工作托管给框架来完成。

结合工程运用和软件项目实践，本文在传统的web程序架构模式之上，采用了一种架构方式，充分利用当前流行的开源框架和技术，大大简化了软件开发的过程，提升了开发的可控性。架构方式如下表所示：

表

|  |  |  |  |
| --- | --- | --- | --- |
| 适用场景 | 框架/技术 | 简述 | 说明 |
| 版本控制 | git、github | 分布式的代码版本控制软件，适用于同步存在于不同地区的计算机上的软件的各个版本。 | git优于svn等集成管理工具 |
| git shell | git命令行工具 |  |
| 构建及依赖管理 | maven | 遵循约定优于配置的思想，为软件约定了一套实用的结构，便于从不同的“maven仓库”中导出合适的依赖， | 可以使用gradle代替 |
| 前端 | jQuery | dom操作框架 | 简化dom操作及事件处理 |
| lodash | 给javascript提供函数式风格的api | javascript语言框架 |
| jQuery easyui | 前端ui框架 | 易用性较强 |
| 后端 | spring | java语言框架 |  |
| spring jdbc | 基于jdbc技术提供数据库访问的模板api |  |
| derby | 使用java开发的数据库，支持以内嵌或服务器方式运行 | derby足够小巧并且api齐全 |
| 前后端数据通讯方式 | ajax | 异步javascript |  |
| json | javascript对象注记法 | 高效的数据传输格式 |
| groovy json | groovy json处理api |  |
| spring+jackson | spring内置json处理api |  |
| 测试 | junit | 单元测试框架 |  |
| matlab/simulink | 用于验证计算的正确性 |  |
| 开发平台 | jdk8 | java se 8 sdk | 提供了静态语言下的函数式风格编程 |
| groovy 2.7 | 基于jvm的动态类型语言 | 快速原型开发语言及脚本语言 |
| windows 7 32bit操作系统+intel i3处理器 | 计算机1配置 | 用于试验项目开发的计算机1 |
| windows 10 64bit操作系统+intel i5处理器 | 计算机2配置 | 用于试验项目开发的计算机2 |
| eclipse 4.6(Neon.1) | eclipse集成开发环境 | 需要java 8及以上版本 |
| eclipse groovy plugin | groovy的eclipse插件 | 负责groovy代码的编译及配置 |
| eclipse maven plugin | maven的eclipse插件 | 集成maven功能 |
| eclipse git plugin | git的eclipse插件 | 集成git功能 |
| eclipse spring plugin | spring的eclipse插件 | 为spring开发提供gui支持 |
| 服务器 | jetty | jetty servlet服务器 | 使用java开发的小巧服务器 |

本文基于上述架构，对java 8平台下的水轮机仿真程序所涉及的技术进行了较深入的研究；基于java web技术，完成了的“水轮机调速系统仿真”样例（demo）项目的开发，以下简称“demo项目”。

3.1框架及技术选择考量

1.git/github

在软件项目中，如果不进行版本控制，时刻保持项目的各个版本处于一致的状态之中，就会同时存在数个版本，引发“版本灾”。git是一个分布式的版本控制工具，它从文件的二进制编码上对同一文件的两个不同版本进行比对，并对不一致的地方进行合并或者冲突提醒，从而促使不同的版本重新回到统一的状态之中来。本文主要使用github gui、git gui以及git的eclipse插件对位于两个不同机器中的项目版本进行同步。如图所示，在计算机1中开发并提交至本地版本，然后push修改至远端服务器，再到计算机2中pull检出最新的版本并与本地版本进行合并。



图 git进行项目远程同步操作

除了版本同步，git还提供了分支（branch）的概念以从项目的一个版本出发开发适用于不同环境的软件版本，提供了里程碑（tag）的概念以标记项目开发过程中的重要版本。

使用git的原因除了其易用性以外，还包括github（https://github.com）网站提供免费的git服务器，可以将项目提交到远端进行存储。demo项目使用git进行版本控制，其github url为：

https://github.com/yy550956983/graduate.git

需要注意的是，git不仅对文本数据提供同步，二进制文件（如pdf，doc）的同步也能从中获益，比如：软件开发的文档，参考文献。此外，git项目每次提交，要求给出当次提交的描述信息，便于项目开发成员进行交流。eclipse 4.6版本通过插件集成了对git的支持，导入项目时如果检测到git版本库（.git文件夹下的所有内容），可以提供git的gui工具，简化操作。



文献《git权威指南》详细介绍了git的原理以及使用git shell命令行工具及git gui对个人及团队项目进行版本控制。

2.maven

整体结构

版本控制：git、github

项目构建和依赖管理：maven

后端架构：java8、spring jdbc、spring mvc

前端架构：jQuery、easyui、lodash

业务建模：仿真（线性/非线性模型）

领域建模：连续系统微分离散化处理（一组离散化后得到的差分方程）

测试：matlab simulink + junit

前后端通讯：http、jQuery ajax、json

开发平台：windows7 32bit，eclipse 4.6（neon），java se 8

测试服务器：embedded jetty

发布服务器：resin 4.9

脚本工具：git bash、groovy

快速原型开发：groovy

4、软件设计

5、仿真模型

6、仿真程序的应用