封面

摘要

目录

1、绪论

2、开发约定及思想

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，本文基于传统的web程序架构模式，采用了一种架构方式，充分整合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 demo项目采用的相关框架及技术

|  |  |  |  |
| --- | --- | --- | --- |
| 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demo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demo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”。

前文提到过，上述架构基于经典的四层web应用架构提出：展示层（UI Layer）、web层、业务层（Business Logical Layer）、持久层（Persistence Layer）。限于实践的不足，无法应用于分布式、大吞吐量的环境之中。但是，在遵循面向接口、模块化、组件化、面向对象等良好编程实践和原则的条件下，利用面向服务的架构、分布式等技术，可以拓展至更广泛的适用范围之中。此外，讨论的背景虽然是java平台和工程领域，但是同样的问题存在于各种开发背景之中。接下来，有必要讨论作出上述选择的考量。

3.1版本控制工具——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。

除了版本同步，git还提供了分支（branch）的概念用以从项目的一个版本出发开发适用于不同环境的软件分支，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。

需要强调的是，git不仅能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版本内置集成了对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。



图 通过右键git文件夹打开git命令行

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。

demo项目使用git进行版本控制，其github url为：

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

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库，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

3.2项目构建工具——maven

项目构建工具一般约定了项目的结构，提供了对第三方依赖的管理，并对代码的编译、测试，项目的发布等特定项目生命周期的任务提供支持，大大简化了项目的构建工作。常用的构建工具包括maven和gradle。

典型的maven web项目约定结构如下图所示。通过src/main/java和src/main/test文件夹，可以轻易地将源码和测试代码分置在不同的文件夹中，分开管理。如果其下的包结构相同，那么包下的类的命名空间也相同，这样测试类就能够访问源类的包可见域，而其他包不能访问，从而既保护了封装，又便于测试。此外，webapp文件夹用于存放前端程序，target用于存放程序的生成内容，如编译好的.class文件、jar包、war包、文档等等。



图 典型的maven项目结构

maven通过pom.xml文件进行配置。项目坐标唯一定位了一个项目，通过坐标添加对某项目的依赖。maven将项目开发分为了很多个详细的阶段（如构建、开发、测试、发布等），maven插件通过作用于不同的阶段来完成对maven功能的拓展，同时，插件也通过坐标来进行定位。例如，在pom.xml的dependencies标记下添加对apache math3项目（提供高等数学的算法包）的依赖：

|  |
| --- |
| <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-math3</artifactId>  <version>3.5</version>  </dependency> |

添加了对commons-maths3的依赖。通过使用maven的eclipse插件，在pom.xml发生变更时，maven可以自动检测到依赖的变化，并自动调整程序的依赖项。流程如下图所示：



图 maven依赖管理流程图

demo项目还依赖于两个maven插件：compiler插件和jetty插件。compiler插件指定了java代码的编译器，jetty插件在开发过程中内嵌对jetty容器的支持。详细配置参数表可以在可以在插件相关网站上查询。使用搜索引擎搜索maven xxx或maven xxx plugin 可以查询到第三方依赖项目的坐标以及配置方式。

compiler插件配置：

|  |
| --- |
| <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.6.0</version>  <configuration>  <source>1.8</source>  <target>1.8</target>  </configuration>  </plugin> |

jetty插件配置：

|  |
| --- |
| <plugin>  <groupId>org.eclipse.jetty</groupId>  <artifactId>jetty-maven-plugin</artifactId>  <version>9.2.10.v20150310</version>  <configuration>  <webApp>  <contextPath>/Simulation</contextPath>  <defaultsDescriptor>src/main/resources/webdefault.xml</defaultsDescriptor>  </webApp>  <webAppSourceDirectory>src/main/webapp  </webAppSourceDirectory>  <classesDirectory>target/classes</classesDirectory>  </configuration>  </plugin> |

除maven以外，gradle集成了maven、ant等项目的构建功能，并提供了脚本化的配置方式，使用了dsl（领域特化语言）的风格，简化了配置文件的编写方式。例如，常见的gradle配置文件build.gradle如下所示：

|  |
| --- |
| apply plugin: 'java' // 插件配置  apply plugin: 'jetty'  apply plugin: 'eclipse-wtp'  sourceCompatiability = 1.8 // jdk配置  targetCompatiability = 1.8  repositories{ // 仓库配置  mavenLocal() // 指定maven本地仓库  mavenCentral() // 指定maven中央仓库  }  dependencies{ // 依赖配置  compile 'org.apache.commons: commons-math3: 3.5'  }  jettyRun{ contextPath = 'Simulation'}// 服务器配置  eclipse{ wtp{  component{contextPath = 'Simulation'}  }  } |

可以看出，以上的配置比maven基于xml的配置方式更加的友好。由于gradle是groovy平台下的项目，上述配置文件符合groovy的语法，简洁明了。gradle是maven的可选替代构建工具，并且有相关的gradle eclipse插件可以使用。

demo项目使用maven作为构建工具，由于eclipse 4.6版本内置了对maven的支持，使用maven插件不需要额外的配置。文献《maven详解》对maven的原理及使用作了深入细致的剖析。

3.3 后端架构

（1）新一代平台——java 8

java 8特指java se 8版本。是Oracle公司收购sun以后推出的首个java重要版本。其引入的最重要的java语言特性为lambda表达式，通过函数式接口和lambda表达式语法糖，创造性地为静态类型语言添加了对函数式编程的支持。此外，jdk 8提供了一组新的采用函数式风格的流式api。

如下代码所示，判断列表中所有的元素都大于0。其中，“n -> n>0”实现了内置的Predicate<T>接口（这里是Predicate<Double>），由编译器在内部将这一语法糖转化为该接口的匿名内部类的实现。语法比起之前版本的for循环和迭代器有了进一步的简化。

|  |
| --- |
| List<Integer> numbers=Arrays.asList(1,2,3);  numbers.stream().allMatch(n-> n>0); //true |

demo项目主要完成了水轮机调速系统仿真程序的编制。可能熟悉c/c++开发环境的人会质疑java语言的性能问题，但是，随着jvm（java虚拟机）中JIT（即时编译）等最新编译技术的使用，java的执行速度已经可以和本地代码相媲美。关于这点在demo项目的程序编制中会有说明。

（2）spring mvc和spring jdbc

关于spring框架的原理与使用业界已经有较为深入的了解。文献《spring详解》介绍了spring的原理与使用。spring mvc是基于spring的web框架，采用了命令模式对前端的请求进行拦截、解析、处理及结果渲染。



spring框架内置了对json和java bean互相转换的支持。为了利用这一特性，需要在pom.xml中添加对jackson包的依赖，并在spring web的配置文件dispatcher-servlet.xml中添加如下标记：

|  |
| --- |
| <context:component-scan base-package="zte.hx.action" />  <mvc:annotation-driven /> |

其中，第一句指明了控制器bean的搜索包，第二句表明了通过注解进行配置。有如图所示的Person和PersonAction类及其实现，



图 简单的spring mvc控制器

|  |
| --- |
| @Controller  public class PersonAction {  @ResponseBody // json是默认的返回类型  @GetMapping(path = "/get-person")  public Person getPerson() {  return new Person("hx", 25, "java");  }  } |

当采用url：

<http://localhost:8080/Simulation/get-person.action>

访问该控制器时，spring框架会对结果进行转换得到与Person类对应的json，如下所示：

{"name":"hx",age:25, "hobby":"java"}

demo项目在前端页面和控制器之间反复使用上述特性传输数据，避免使用语法复杂和对浏览器不友好的页面脚本（如jsp），使代码具有简洁性和一致性。

3.4 前端架构——jQuery与lodash

jQuery是目前最为流行的前端javascript框架。个人认为，可以把jQuery看作是处理dom的dsl，它封装并屏蔽了底层浏览器的差异，提供了对于dom的crud（Create Retrieve Update Delete）操作接口、事件处理机制、ajax调用、css操作、以及一些简化语言复杂性的api，结构清晰，使用方便。文献《jQuery权威指南》对jQuery作了详细的阐述。

lodash是纯粹的javascript语言框架，提供了一组易用的api，处理诸如语言类型，数组、集合操作，字符串操作等功能，以及赋予javascript函数式编程的风格。可以在https://lodash.com上查看有关lodash的信息。使用lodash完成“集合中的所有数都大于0”的代码：

|  |
| --- |
| var numbers = [1, 2, 3];  \_.every(numbers, function(n) {return n > 0;}); |

上述代码同样比较简洁，当然，如果javascript引入了lambda表达式，还能进一步得到简化。简洁的代码通常就意味着更高的可读性。

3.5 前后端通讯格式——json

json（JavaScript Object Notation，javascript对象注记）是javascript语言的对象字面量表示法，可以表达具有一定层次结构的数据类型。由于动态弱类型的特性，javascript语言有良好的字面量表示方法，表的对比表明，json在字面量的表示上有更大的灵活性，且受到语言类型系统的影响最小。

表 javascript和java、groovy字面量的对应关系

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| javascript类型 | javascript字面量 | java字面量 | | groovy字面量 | |
| 字面量 | 类型 | 字面量 | 类型 |
| Number | 1.0 | 1 | int | 1 | Integer |
| 1.0 | double | 1.0 | BigDecimal |
| String | "str" | "str" | String | 'str' | String |
| Boolean | true, false | | | | |
| Array | [1,2] | {1,2} | int[] | [1,2] | ArrayList |
| Object | {a:1,b:2} | 无 | | [a:1,b:2] | LinkedHashMap |

后端生成的遵循java bean规范的数据，可以通过spring的支持和json数据进行互相转换。文献《论文》说明了json数据在网络传输时的性能表现，json在大多数情况下可以满足前后端的数据交换需求。

groovy语言从api层面对json提供了支持。通过工具对象JsonSlurper和JsonOutput可以很容易地在json和groovy对象字面量之间进行互相转换。

代码 JsonTest.groovy

|  |
| --- |
| import groovy.json.JsonOutput // 导入相应的工具类  import groovy.json.JsonSlurper  def slurper = new JsonSlurper() // json转换为groovy对象  def obj = slurper.parseText """  {"name":"hx","age":24,"hobby":"java"}  """  assert obj.name == 'hx'  assert obj.age == 24  assert obj.hobby == 'java'  // groovy对象转换为json  def person = [name:'hx',age:24,hobby:'java']  def out = new JsonOutput()  def personJson = out.toJson(person)  assert personJson == '{"name":"hx","age":24,"hobby":"java"}' |

3.6 derby数据库和jetty服务器

derby数据库是由java语言实现的基于文件系统进行组织的数据库，其特点是非常小巧，既可以作为数据库单独运行，也可以内嵌于程序中运行。derby数据库符合sql-92和jdbc 3.0标准。最为重要的是，derby和绝大多数托管于apache基金会的java项目一样，具有十分完善且详细的文档支持，且文档会随同derby的发布版本一同发布，derby的下载地址为：http://db.apache.org/derby/。

demo项目使用derby数据库存储程序相关数据。在项目开发过程中，计算机1属于便携式笔记本，由于配置方面的原因无法在上面部署重量级的商业数据库，然而得益于derby的小巧，可以克服上述不足，满足研究和软件开发中对于数据持久化的需求。

此外，eclipse的Database Development透视图（perspective）提供了一个便捷的数据库连接与交互gui，可以书写标准sql语句对数据库进行crud操作。

选择jetty作为服务器是基于和derby一样的考量。jetty是一个servlet容器，同样是基于纯java语言实现的，小巧易用，可以嵌入到程序中或单独作为服务器运行。与生产环境下使用的服务器（比如tomcat）相比，jetty显得更加轻量，灵活，但是提供的拓展功能更少，不过，对于demo项目的研究性质与大多数非生产环境而言，功能已经足够使用了。

在大型项目开发当中，开发和部署选择不同的配置是常见的策略。像derby和jetty这样小巧的程序具有更加平缓的学习曲线，适合程序员在开发过程中单独使用，更加复杂的需求可以交由专业人士进行，在项目后期再迁移到适用于生产环境的重量级工具进行测试、集成和部署。

maven使用插件jetty-maven-plugin对jetty提供了支持，使用maven命令:mvn jetty-run可以开启jetty内嵌服务器，默认监听8080端口上的请求。另外，可以配置jetty扫描资源文件的时间间隔，以迅速地将文件的变更反映到程序当中，和同样条件下运行的tomcat相比，jetty对改变更加灵敏，tomcat在变化频繁的时候有可能停止继续响应，需要重启服务器才能正确运行。

7.脚本与动态语言支持——groovy

groovy是jvm上的一种强大的、可选类型的、动态语言。groovy旨在像python和c/c++之间的关系那样成为java平台下的“粘合剂”。然而，groovy是一门独立的、完整的、成熟的语言。groovy代码可以作为脚本运行，也可以编译运行。groovy代码和java代码是二进制兼容的，也就是说，二者编译而来的.class文件是兼容的。正因为如此，二者相互调用无需额外操作，只要能在类路径下寻找到彼此即可——groovy代码可以和java代码无缝集成。

从语言层面来讲，groovy提供了很多很强大的特性。早在java 8以前，groovy已经通过闭包（closure）特性来支持函数式风格的编程；groovy特殊的函数调用语法和闭包的代理特性使其天然可以支持DSL（Domain-Specific Language）的编写；元编程（meta-programming）的特性支持groovy动态地对类进行拓展；特质（trait）则提供了“胖接口”和混入（mixin）的特性。gradle（构建工具，https://gradle.org/）和grails（一站式的web应用解决方案http://www.grails.org/）是应用广泛的两个groovy平台下的项目。

eclipse下通过插件对groovy提供了支持。在eclipse ide中选择Help -> install new software-> add，键入url：

http://dist.springsource.org/snapshot/GRECLIPSE/e4.7/

可以安装该插件。通过新建groovy class可以创建groovy类，通过快捷键shift+alt+x, g可以以脚本形式运行groovy。

在实际使用中，可以下载并安装groovy，只需要解压到特定文件夹（即安装目录），然后指定环境变量GROOVY\_HOME并指向groovy安装目录，在path变量中添加%GROOVY\_HOME%\bin，就能够在cmd命令行中通过groovyConsole命令进入groovy自带的控制台环境，实现repl（read-evalue-print loop，输入-执行-打印循环）风格的编程，以得到迅速的反馈、加深对groovy语言的理解。



图 groovy控制台实现repl

对于demo项目的“仿真”主题而言，groovy由于其内部实现机理，执行速度远远不及java（同样的问题也存在于python之中），但是groovy语言的动态特性以及对json的支持十分适合开发接口及上层驱动程序在这种情况下，可以将对性能敏感的业务程序使用java编写，将对变化敏感的接口程序使用groovy编写。从而同时发挥静态类型严谨和动态语言灵活的优势。

4、仿真原理及调速系统模型

前两个章节明确了开发工程应用软件所遵循的思想，并结合工程实际运用确立了一种实用性较高的java web软件开发架构，接下来需要研究计算机控制系统仿真的原理、算法及其程序实现方式。同时，水轮机调速系统的控制系统模型及仿真算法也是题中应有之义。

4.1控制系统仿真的原理

仿真活动的本质在于通过模型去模拟实际系统的表现。从历史发展来看，仿真包括物理仿真和数字仿真。物理仿真借助物理模型对实物进行近似，至今仍应用于水轮机设计和制造之中，使用水轮机原型和相似理论去近似模拟实际水轮机的特性。数字仿真特指借助数学模型和数字计算机所进行的仿真。和物理仿真相比，数学模型的建立更加容易，开发周期较短，适应能力强。今天所说的仿真，大多数是指数字仿真。

仿真的步骤一般如下图所示：



图 仿真的一般步骤

从仿真对象来看，仿真主要分为连续系统的仿真和离散系统的仿真。调速系统的仿真属于连续系统的仿真。

从仿真的时间标尺τ和实际时间标尺t的比例关系来看，仿真分为实时仿真（t/τ=1）和非实时仿真（t/τ≠1）。对调速系统的仿真当中，当应用于分析极值、验证调节规律、调节保证计算等对实时性要求不高的场合，可以采用非实时的仿真。实时仿真一般指有实物参与的仿真，这时候，仿真模型必须具有接收实际数据和发出计算数据的接口，还要等待实物的响应才能进行进一步的仿真，这种仿真适用于输入容易获取、仿真对象操作代价高昂的情况。

水轮机调节系统属于动力系统，这种系统的数学模型一般可以由一组微分方程来表征。根据经典的控制理论，描述系统的微分方程组、控制方框图和状态方程之间可以相互转化，但其物理本质是等价的。另外，由于迟滞、饱和等实际因素的存在，水轮机还存在一定的非线性结构。这些都是编制仿真程序时应该考虑到的因素。

对于系统的三种等效模型，由于转态方程在数学描述上的简单性，适合用来进行理论推导；方框图的形式适合用来进行模块化的开发；微分方程的形式适合用来构造算法。

1.离散相似法

设一线性定常系统的状态方程如下所示：

(4-1)

(4-2)

其方框图如下所示：



图 系统方框图

在连续时间下，式(4-1)可表述为：

(4-3)

式中A、B均为常系数矩阵。对上式进行拉普拉斯变换，得到：

(4-4)

以左乘上式两端，并作拉普拉斯反变换，可得：

(4-5)

其中为转移矩阵，且有：

(4-6)

同时，我们也可以使用左乘(4-3)式两端，整理后得到：

(4-7)

利用微分计算规则，可以将上式改写为：

(4-8)

上式两端同时积分并整理，可得：

(4-9)

比较(4-5)式及(4-9)式，可以得到：

(4-10)

由拉普拉斯反变换可知，上式中矩阵指数可以展开成级数的形式并定义为：

(4-11)

由上述(4-9)式可求得该线性定常系统的准确解，当不易求解时，可以通过式(4-11)求过渡矩阵的近似解。上式是在连续的条件下得到的，计算机无法处理连续的情况，因此要得到适合编制程序进行计算的算法，需要将原来的模型进行离散化处理，从而对实际的连续系统加以近似。

对于上述系统，当t=kT时，带入(4-9)可得：

(4-12)

当t=(k+1)T时，有：

(4-13)

由上述两式得到系统离散解的递推格式为：

(4-14)

当采用零阶保持器并且不进行补偿时，有：

(4-15)

上式带入(4-14)式可得：

(4-16)

令上式中，，可见，实际计算过程就是要根据系统的状态方程求解这两个系数、并进行迭代的过程。

2.数值积分法

离散相似法本质上是对微分进行离散化处理从而近似求解微分方程。从数值积分的思路出发，可以建立起一套新的求解方法。数值积分法本质上也属于离散相似法。

重写离散相似法中的系统如下：

(4-1)

(4-2)

由系统的结构可知：

(4-17)

对上述系统进行离散化处理，分别令t=kT和t=(k+1)T:

(4-18)

(4-19)

用(4-19)减去(4-18)式可得：

(4-20)

于是，对积分的不同近似算法就构成了系统不同的近似求解方法。从数值分析的知识可知，主要的近似方法有欧拉法、梯形法、龙格-库塔法、Adams法等。

当采用欧拉法时，使用零阶保持器，对偏差的离散处理为

(4-21)

上式带入(4-20)可得：

(4-22)

简记为：

(4-23)

由e(k)近似将上式带入(4-1)和(4-2)，可得：

(4-24)

(4-25)

上文从理论上考察了控制系统

4.2控制系统的仿真算法

4.3控制系统仿真程序的计算机实现

4.4水轮机调速系统模型

5、程序原型及快速迭代开发

6、仿真程序的应用