



电子科技大学

University of Electronic Science and Technology of China

学 士 学 位 论 文

BACHELOR DISSERTATION

论文题目 分布式在线程序评测系统——web端实现

学生姓名 李昀

学 号 2010013100008

专 业 通信工程

学 院 通信与信息工程学院

指导教师 饶力

指导单位 电子科技大学

2014年2月24日

摘 要

ACM国际大学生程序设计竞赛（英语：ACM International Collegiate Programming Contest, ICPC）是由美国计算机协会（ACM）主办的，一项旨在展示大学生创新能力、团队精神和在压力下编写程序、分析和解决问题能力的年度竞赛。经过30多年的发展，ACM国际大学生程序设计竞赛已经发展成为最具影响力的大学生计算机竞赛。

程序设计竞赛采用了黑盒测试的思想来评判选手的程序。在黑盒测试中，测试者只知道程序的输入、输出和系统的功能，按照一定的规范设计出一系列测试案例来进行测试。在线程序评测系统（Online Judge）以此为基础，可以对多种语言的源代码进行自动编译、测试、分析及评判。除了被应用于程序设计竞赛，也有一些老师将其引入到日常的程序语言教学之中，并取得了很好的效果。

在本文中，作者首先通过介绍了目前著名的几个在线评测系统来说明目前该类平台的发展趋势，并阐述了MVC架构的特点以及如何应用到web应用中去，随后设计了一个基于Java Spring MVC框架和AngularJS的简单的分布式在线程序评测系统，并描述了它的整体架构和实现。

关键词：程序设计竞赛，黑盒测试，在线程序评测系统，MVC架构，Web应用

ABSTRACT

ACM International Collegiate Programming Contest (abbreviated as ACM-ICPC or just ICPC) is an annual multi-tiered competitive programming competition among the universities of the world. This contest is designed to show students' ability of innovation, teamwork, analysis problem, solve problem and coding under pressure. After 30 years of development, ACM-ICPC has become the most influential collegiate programming contest.

The programming contest use Black-box testing method to judge contestants' program. In Black-box testing, tester only know the functionality of an application (e.g. what the software does and the input/output format), tester design multiply testcases fit in the corresponding format to test. Based on this method, Online Judge System (OJ) can compile, test, analysis and judge multiply kinds of language. Not only in programming contest, many teachers introduce this system into computer language course and achieve significant results.

In this thesis, the writer first introduces the developing trend of Online Judge System through several popular Online Judge Systems, and take a tourial of MVC architecture and how to use MVC in the web application. Then he devises a simple Distributed Online Judge System with Java Spring MVC framework and angularJS, and describes it.

Keywords: Programming Contest, Black-box testing, Online Judge, MVC Architecture, Web Application

目 录

第1章 引言	1
1.1 课题背景	1
1.2 课题意义	2
1.3 国内外研究现状	2
1.4 论文组织架构	3
第2章 相关概念与技术	4
2.1 互联网应用开发历史	4
2.2 系统架构	5
2.2.1 B/S结构	5
2.2.2 MVC架构	5
2.3 开源框架	5
2.3.1 Spring framework	5
2.3.2 Hibernate	6
2.3.3 AngularJS	7
第3章 系统需求分析	8
3.1 部署环境	8
3.2 功能性需求分析	8
3.3 非功能性需求分析	9
3.3.1 界面需求	9
3.3.2 性能需求	9
3.3.3 维护需求	11
第4章 系统概要设计	12
4.1 系统环境	12
4.2 服务器端框架	13
4.2.1 服务器端总体技术架构	13
4.2.2 服务器端模块结构	13
4.2.2.1 配置模块	14
4.2.2.2 数据库模块	14

4.2.2.3 评测器模块	14
4.2.2.4 服务模块	15
4.2.2.5 实用工具	15
4.2.2.6 网站模块	15
4.3 浏览器端框架	15
4.3.1 浏览器端总体技术架构	15
4.3.2 网站结构	16
4.3.3 浏览器端开发流程	16
第5章 系统详细设计	18
5.1 数据库详细设计	18
5.2 服务器端详细设计	18
5.2.1 系统顺序图	21
5.2.2 系统包图	22
5.2.3 Config Package详细设计	22
5.2.3.1 ApplicationContextConfig	23
5.2.3.2 WebMVCResource	23
5.2.3.3 WebMVCConfig	23
5.2.4 Database Package详细设计	23
5.2.4.1 Entity	24
5.2.4.2 DTO	24
5.2.4.3 Condition	25
5.2.4.4 DAO	27
5.2.5 Service Package详细设计	27
5.2.6 Judge Package详细设计	27
5.2.6.1 评测器内核	30
5.2.6.2 JudgeService	30
5.2.7 Web Package详细设计	33
5.2.7.1 AuthenticationAspect	33
5.2.7.2 Controller	35
5.2.7.3 网站地图	38
5.3 浏览器端详细设计	41
5.3.1 客户端结构	41

5.3.2 客户端加载流程	45
5.3.3 顶部导航详细设计	45
5.3.3.1 用户菜单设计	45
5.3.3.2 用户注册流程	47
5.3.3.3 用户登陆流程	48
5.3.3.4 用户登出流程	48
5.3.3.5 用户密码找回流程	49
5.3.3.6 用户账户修改	50
5.3.4 内容渲染	50
5.3.5 列表页面	51
5.3.6 主页设计	53
5.3.7 文章模块详细设计	53
5.3.8 用户模块详细设计	53
5.3.8.1 用户列表页面	53
5.3.8.2 用户中心页面	56
5.3.9 题目模块详细设计	56
5.3.9.1 题目列表页面	56
5.3.9.2 题目页面	57
5.3.9.3 代码提交流程	57
5.3.9.4 题目编辑器	57
5.3.10 评测状态模块详细设计	60
5.3.10.1 评测列表页面	60
5.3.10.2 管理员Rejudge流程	61
5.3.11 比赛模块详细设计	61
5.3.11.1 比赛列表页面	61
5.3.11.2 比赛页面	62
5.3.11.3 比赛编辑器	62
第6章 系统测试	65
6.1 测试方法	65
6.1.1 单元测试	65
6.1.2 Mock	65
6.1.3 Stubbing	66

6.1.4 Mockito	66
6.1.5 Spring MockMVC	67
6.2 测试内容	68
6.2.1 数据库模块测试	68
6.2.2 数据库集成测试	69
6.2.3 服务集成测试	70
6.2.4 实用工具测试	71
6.2.5 控制器测试	72
6.3 测试结果	74
第7章 结束语	76
参考文献	77
致 谢	78
附录 A 部分程序源代码	79
A.1 ApplicationContextConfig.java	79
A.2 resources.properties	84
A.3 WebMVCResource.java	85
A.4 WebMVCConfig.java	86
A.5 spj.cc	88
A.6 AuthenticationAspect.java	91
A.7 navbarTop.jsp	94
A.8 ui.flandre.coffee	95
A.9 controller.listController.coffee	98
外文资料原文	101
外文资料译文	107

第1章 引言

1.1 课题背景

ACM-ICPC (Association of Computing Machinery - ACM International Collegiate Programming Contest, 美国计算机协会——国际大学生程序设计竞赛) 是由国际计算机界历史悠久、颇具权威性的组织ACM于1970年发起组织的年度竞赛活动, 是当今国际计算机界历史悠久并得到全球公认的规模最大、水平最高的国际大学生设计竞赛。大赛旨在展示大学生创新能力、团队精神和在压力下编写程序、分析和解决问题能力, 迄今已经成功举办38届。比赛涌现出的优秀学生往往被各高校和许多知名企业所看重。

ACM-ICPC以团队的形式代表各学校参赛, 每队由3名队员组成^①。比赛期间, 每队使用1台电脑需要在5个小时内使用C、C++或Java中的一种编写程序解决7到10个问题。每个问题都有一组标准的测试数据以及对应的答案, 选手程序完成之后提交裁判运行, 裁判机运行选手提交的程序, 通过其输出与标准答案相比较来得到结果, 运行的结果会判定为“AC(正确)/WA(错误)/TLE(超时)/MLE(超出内存限制)/RE(运行错误)/PE(格式错误)”中的一种并及时通知参赛队。

电子科技大学从2005年起便开始参加这项竞赛, 在最近的第38届ACM-ICPC亚洲区域赛中国大陆赛区共有成都、杭州、南京、长沙、长春5站, 其中成都站的比赛由电子科技大学承办, 本届比赛中, 电子科技大学学子共获4金7银5铜。其中UESTC Aspidochelone代表队在成都站排名第二, 在南京站获得亚军殊荣, 顺利晋级2014年夏季在俄罗斯叶卡捷琳堡举行的世界总决赛^②。

ACM-ICPC与其它竞赛最大的区别在于它采用的是机器评测的方法而不是依靠人的评价, 它采用了黑盒测试[1]的思想来评判选手的程序。在黑盒测试中, 测试者只知道程序的输入、输出和系统的功能, 按照一定的规范设计出一系列测试案例来进行测试。在线程序评测系统(Online Judge)以此为基础, 可以对多种语

① 每位队员必须是在校学生, 有一定的年龄限制, 并且最多可以参加2次全球总决赛和5次区域选拔赛。

② 见: <http://www.new1.uestc.edu.cn/news/index/id/1056>

言的源代码进行自动编译、测试、分析及评判。除了被应用于程序设计竞赛，也有一些老师将其引入到日常的程序语言教学之中，并取得了很好的效果[2][3]。

1.2 课题意义

让我们来看这样一个问题：编写一个计算机程序来求两个任意阶矩阵的乘法。这种题目常常出现在C语言上机考试之中，过去，这些题目的评判大多靠学生将代码写在作业本上然后交给助教批阅。这种评判方式很不方便，而且容易漏掉许多有问题的代码。在线测评系统的出现改变了这一现状，用户将代码提交到在线评测系统上，评测系统用指定的测试数据自动对其进行测试，然后将结果返回给用户。作者希望采用现代的互联网技术来实现一个简单的、易扩展的在线评测系统来给学生和教师带来便捷。同时借此来了解互联网新技术的使用和创新方式。

1.3 国内外研究现状

目前已经存在许多不同种类的Online Judge，如表1-1所示：

表 1-1 几个著名的评测网站

名称	来源	地址
Topcoder	TopCoder, Inc	http://community.topcoder.com/tc
Codeforces	萨拉托夫州立大学	http://codeforces.com/
Project Euler	Colin Hughes	https://projecteuler.net/
HDOJ	杭州电子科技大学	http://acm.hdu.edu.cn/
POJ	北京大学	http://poj.org/
Virtual Judge	华中科技大学	http://acm.hust.edu.cn/vjudge/toIndex.action
ZOJ	浙江大学	http://acm.zju.edu.cn/onlinejudge/

其中HDOJ、POJ、ZOJ都属于传统的Online Judge，有着自己的题库和测评器。HDOJ如今已经成为了国内ACM竞赛界最为著名的Online Judge，每年暑假都会组织多校联合训练，平时还会承办各类程序设计竞赛（如腾讯编程马拉松）。ZOJ则是以每个月举行的浙大月赛而闻名。

相反，Virtual Judge没有自己的题库和评测器，它的题库仅仅是提供了各个OJ题库的一个索引，用户可以在这些题目的基础上组织比赛，然后Virtual Judge将提交的代码送到对应的OJ上去测评，再将结果返回给用户。

Topcoder采用Java applet载入平台，而不是建立于网页之上，它和codeforces都具有challenge环节，在这个环节选手可以互相查看对方代码（在一定条件下），并尝试用自己的测试用例来找出对方代码中的BUG。

1.4 论文组织架构

本文采用了如下的结构：

第一章：引言。主要介绍本论文课题的背景，以及在线评测系统的国内外现状和本论文的主要工作，最后对论文的章节进行了一个合理的逻辑安排。

第二章：相关概念与技术。主要介绍本论文所使用到的相关技术和概念，包括互联网开发的历史和现状、常见的几种架构以、基于Git的维护流程及所使用到的开源框架。

第三章：系统需求分析。主要为本文所研究的在线评测系统进行了需求分析，在分析系统的要求后，对整个系统的功能模块进行划分，并给出了本文需要完成的功能性和非功能性需求。

第四章：系统概要设计。概要设计就是设计软件的结构，包括组成模块，模块的层次结构，模块的调用关系，每个模块的功能等等。除此之外我们还介绍了本系统的开发环境。

第五章：系统详细设计。详细设计阶段就是为每个模块完成的功能进行具体的描述，要把功能描述转变为精确的、结构化的过程描述。本章主要通过类图和包图来描述整个系统的组成和实现，然后详细介绍了网页的各个部分。

第六章：系统测试。系统测试是软件工程中很重要的一部分，这部分介绍了软件测试的基本理论、方法，并给出了本次开发的在线评测系统的测试流程图。

第七章：结束语。本章对本课题进行了归纳和总结，指明了改进和完善的方向。

第2章 相关概念与技术

2.1 互联网应用开发历史

自从互联网诞生以来，网站从最初只能在浏览器中展现静态的文本或图像信息，发展成为功能丰富的各类Web应用，这期间动态技术起着重要的作用。

互联网诞生之初，Web开发还比较简单，开发者经常会去操作web服务器(主要还是他自己的机器)，并且他会写一些HTML页面放到服务器指定的文件夹(/www)下。这些HTML页面，就在浏览器请求页面时使用。但是这样做只能获取到静态内容。由此出现了CGI和Perl脚本，在web服务器端运行一段短小的代码，并能与文件系统或者数据库进行交互。

当时组织CGI/Perl这样的脚本代码太混乱了。CGI伸缩性不是太好(经常是为每个请求分配一个新的进程)，也不太安全(直接使用文件系统或者环境变量)，同时也没提供一种结构化的方式去构造动态应用程序。直到出现了Java Server Pages(JSP)，微软的ASP，以及PHP等技术。

同时，在Google的推广下AJAX（Asynchronous JavaScript and XML，异步的JavaScript与XML技术）开始流行起来，让事情变得很有意思。AJAX允许客户端的JavaScript脚本为局部页面提供请求服务，然后可以在无需回到服务器情况下动态刷新部分页面，也就是更新浏览器中的document对象，通常称作DOM，或者文档对象模型。虽然从服务器端返回的仍然是HTML，但浏览器上的代码能把这HTML片段内嵌到当前页面中。也就是说web应用的响应可以更快，这时我们真正用web应用取代了web页面。谷歌的GMail和谷歌地图都是当时AJAX的杀手级产品。随后用AJAX局部刷新就如雨后春笋般出现。

在随后的几年时间里，AJAX成为了焦点，但在服务器端仍然使用着旧有的技术。大概在2007年，37signals公司公开其成员——Ruby on Rails。那个基于Ruby on Rails 5分钟构建博客的演示完全征服了全世界的开发者。一夜之间，所以谈论的焦点都是关于Rails，Rails的不同之处在于使用规定的方式去设计你的web应用程序，运用一种已经广泛在桌面应用开发，但未被搬到web应用上的开发模式。这种模式就叫做MVC（Model-View-Controller）模式。

直到今天，MVC模式已经被应用于许许多多的框架之中，例如在服务器端运行的Spring MVC框架，在前端运行的AngularJS。这允许我们能够快速构建web服务，以及基于AngularJS的客户端接口，甚至和其它的服务，如PhoneGap或者其它原生移动开发工具一样，进行移动应用的开发。

2.2 系统架构

2.2.1 B/S结构

浏览器-服务器（Browser/Server）结构，简称B/S结构，与C/S结构不同，其客户端不需要安装专门的软件，只需要浏览器即可，浏览器通过Web服务器与数据库进行交互，可以方便的在不同平台下工作；服务器端可采用高性能计算机，并安装Oracle、Sybase、Informix等大型数据库。B/S结构简化了客户端的工作，它是随着Internet技术兴起而产生的，对C/S技术的改进，但该结构下服务器端的工作较重，对服务器的性能要求更高。

2.2.2 MVC架构

MVC模式（Model-View-Controller）是软件工程中的一种软件架构模式，把软件系统分为三个基本部分：模型（Model）、视图（View）和控制器（Controller）。具体细节可以参考本文附带的外文资料[4]及其翻译。

2.3 开源框架

2.3.1 Spring framework

Spring Framework是一个开源的Java / Java EE全功能栈（full-stack）的应用程序框架，以Apache许可证形式发布，也有.NET平台上的移植版本。该框架基于Expert One-on-One Java EE Design and Development[5]一书中的代码，最初由Rod Johnson 和Juergen Hoeller等开发。Spring Framework 提供了一个简易的开发方式，这种开发方式，将避免那些可能致使底层代码变得繁杂混乱的大量的属性文件和帮助类。

Spring 中包含的关键特性：

- 强大的基于JavaBeans的采用控制翻转（Inversion of Control, IoC）原则的配置管理，使得应用程序的组建更加快捷简易。
- 一个可用于从applet到Java EE等不同运行环境的核心Bean 工厂。
- 数据库事务的一般化抽象层，允许声明式（Declarative）事务管理器，简化事务的划分使之与底层无关。内建的针对JTA和单个JDBC数据源的一般化策略，使Spring的事务支持不要求Java EE环境，这与一般的JTA或者EJB CMT相反。
- JDBC抽象层提供了有针对性的异常等级（不再从SQL异常中提取原始代码），简化了错误处理，大大减少了程序员的编码量。再次利用JDBC时，你无需再写出另一个“终止（finally）”模块。并且面向JDBC的异常与Spring通用数据访问对象（Data Access Object）异常等级相一致。
- 以资源容器，DAO实现和事务策略等形式与Hibernate，JDO和iBATIS SQL Maps集成。利用众多的翻转控制方便特性来全面支持，解决了许多典型的Hibernate集成问题。所有这些全部遵从Spring通用事务处理和通用数据访问对象异常等级规范。
- 灵活的基于核心Spring功能的MVC网页应用程序框架。开发者通过策略接口将拥有对该框架的高度控制，因而该框架将适应于多种呈现（View）技术，例如JSP，FreeMarker，Velocity，Tiles，iText以及POI。值得注意的是，Spring 中间层可以轻易地结合于任何基于MVC框架的网页层，例如Struts，WebWork，或Tapestry。
- 提供诸如事务管理等服务的面向方面编程框架。

在设计应用程序Model时，MVC模式（例如Struts）通常难于给出一个简洁明了的框架结构。Spring却具有能够让这部分工作变得简单的能力。程序开发人员可以使用Spring的JDBC抽象层重新设计那些复杂的框架结构。

2.3.2 Hibernate

Hibernate是一种Java语言下的对象关系映射解决方案。它是使用GNU宽通用公共许可证发行的自由、开源的软件。它为面向对象的领域模型到传统的关系型数据库的映射，提供了一个使用方便的框架。

它的设计目标是将软件开发人员从大量相同的数据持久层相关编程工作中解放出来。无论是从设计草案还是从一个遗留数据库开始，开发人员都可以采用Hibernate。

Hibernate不仅负责从Java类到数据库表的映射（还包括从Java数据类型到SQL数据类型的映射），还提供了面向对象的数据查询检索机制，从而极大地缩短的手动处理SQL和JDBC上的开发时间。

2.3.3 AngularJS

AngularJS是一款开源JavaScript函式库，由Google维护，用来协助单一页面应用程序运行的。它的目标是透过MVC模式(MVC) 功能增强基于浏览器的应用，使开发和测试变得更加容易。

函式库读取包含附加自定义（标签属性）的HTML，遵从这些自定义属性中的指令，并将页面中的输入或输出与由JavaScript变量表示的模型绑定起来。这些JavaScript变量的值可以手工设置，或者从静态或动态JSON资源中获取。

AngularJS是建立在这样的信念上的：即声明式编程应该用于构建用户界面以及编写软件构建，而指令式编程非常适合来表示业务逻辑。框架采用并扩展了传统HTML，通过双向的数据绑定来适应动态内容，双向的数据绑定允许模型和视图之间的自动同步。因此，AngularJS使得对DOM的操作不再重要并提升了可测试性。

设计目标：

- 将应用逻辑与对DOM的操作解耦。这会提高代码的可测试性。
- 将应用程序的测试看的跟应用程序的编写一样重要。代码的构成方式对测试的难度有巨大的影响。
- 将应用程序的客户端与服务器端解耦。这允许客户端和服务端端的开发可以齐头并进，并且让双方的复用成为可能。
- 指导开发者完成构建应用程序的整个历程：从用户界面的设计，到编写业务逻辑，再到测试。

Angular遵循软件工程的MVC模式,并鼓励展现，数据，和逻辑组件之间的松耦合.通过依赖注入（dependency injection），Angular为客户端的Web应用带来了传统服务端的服务，例如独立于视图的控制。因此，后端减少了许多负担，产生了更轻的Web应用。

第3章 系统需求分析

3.1 部署环境

本应用部署在电子科技大学ACM-ICPC集训队的服务器中，域名为<http://acm.uestc.edu.cn/>，系统规格如下：

1. 硬件

- Intel(R) Xeon(R) CPU X3430 @ 2.40GHz
- 2GB RAM

2. 软件

- Debian 7.3
- gcc (Debian 4.7.2-5) 4.7.2
- g++ (Debian 4.7.2-5) 4.7.2
- java version "1.7.0_25", OpenJDK Runtime Environment (IcedTea 2.3.10) (7u25-2.3.10-1 deb7u1)

3.2 功能性需求分析

- 对于评测器来说，本系统应该具有如下基本功能：
 1. 对于给定的程序源代码和对应的测试用例，评测器将源代码编译、运行并比较输出结果是否正确，同时将结果返回给用户。
 2. 评测器可以对程序做出限制，基本的限制有时间限制，内存限制和权限限制，并且保证程序运行不会对系统造成破坏。
 3. 不同的题目可以有不同的限制范围。
- 对于用户来说，本系统应该具有如下功能：
 1. 注册并登陆到系统之中，并且每个用户都有自己的权限范围。

2. 在权限范围内浏览题目、提交题目、浏览比赛、参与比赛、查看自己提交的代码状态、修改用户信息、发布帖子、留言。

- 对于管理员来说，本系统应具有如下功能：

题目管理 向题库中加入/管理题目，每个题目都拥有统一的格式，系统应该提供基本的编辑功能。

比赛管理 向比赛列表中新建/管理比赛，并进行适当的统计操作。

提交管理 查看用户提交的代码，在特殊情况下能够对提交进行Rejudge（重新测评）操作。

用户管理 对用户的一些信息做出修改（提升权限，封禁用户）。

公告管理 发布/修改公告来告知用户近期的比赛以及一些新闻。

系统维护 对系统进行维护操作，如备份，回档。

3.3 非功能性需求分析

3.3.1 界面需求

易用性的高低决定着用户对产品的第一印象是好是坏。因为Online Judge需要向用户展示大量的数据，于是用户需要一个简洁、易懂的界面，这样他们就能不需要任何帮助地一眼获得需要的信息。考虑到实际情况，网站最低需要支持IE 7浏览器。

考虑到部分用户会使用移动设备登陆网站，该网站应当对移动设备有一定的优化。

3.3.2 性能需求

网站在各种操作和处理中响应速度应为秒级甚至毫秒级，达到实时要求。尽量减少卡顿情况。同时对于评测器来说要高效的评测代码，正常情况下不能出现待测试代码堆积的情况。同时作为一个成熟的系统应该能保证用户稳定地进行比赛、提交题目，程序在任何情况下都能正常运行，不出现崩溃的情况。

为了说明实际的使用情况，本文统计了老版本电子科技大学Online Judge的部分使用情况，如表3-1所示：

基于上述统计，我们规定系统应该具备以下能力：

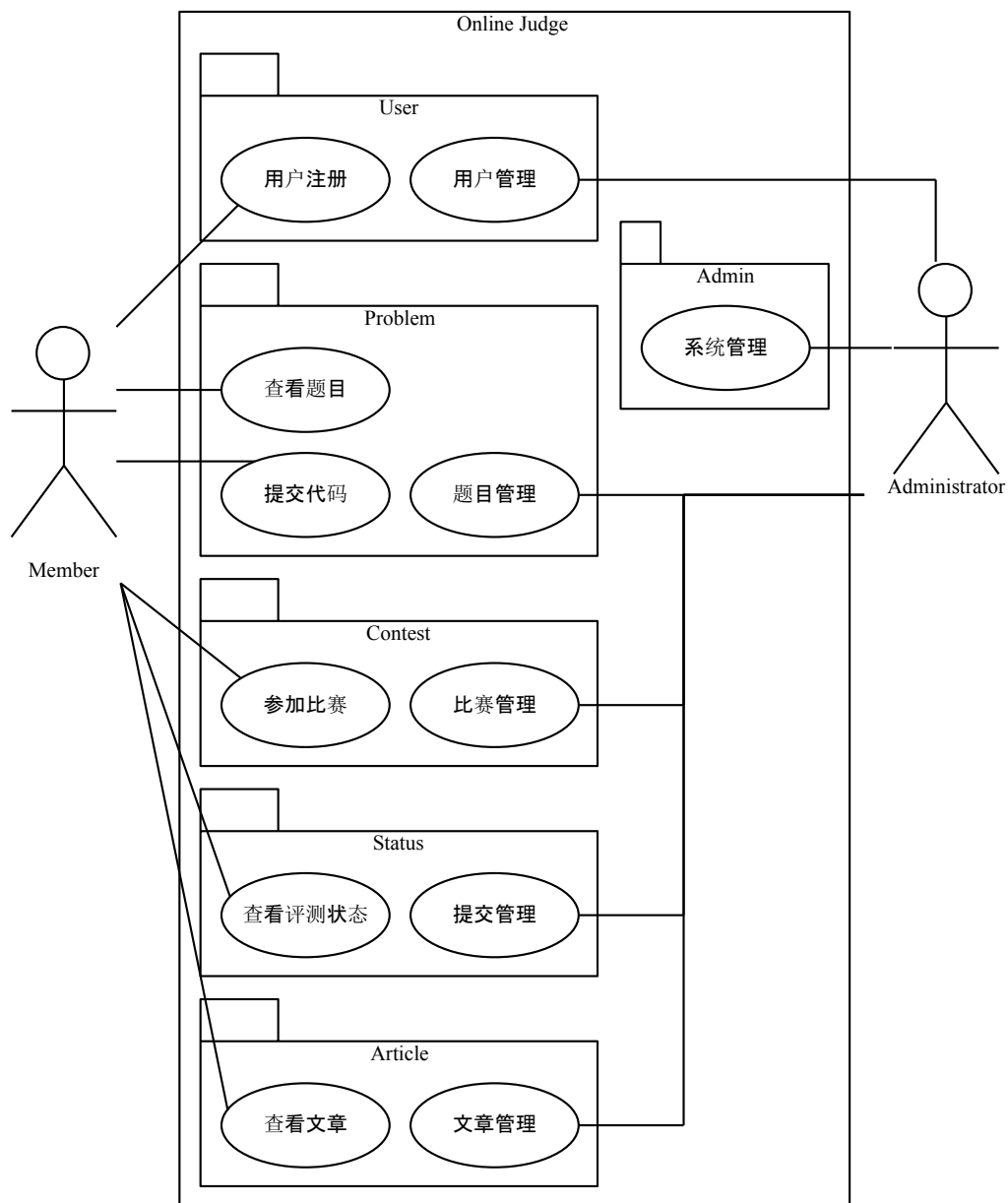


图 3-1 系统用例图

表 3-1 电子科技大学Online Judge使用状况

项目	用户规模	提交规模	历时
总体	32083	475583	4年
第11届电子科技大学校赛	94	1315	5小时
第五届ACM趣味程序设计竞赛第四场（正式赛）	82	676	3小时
数学科学学院2013级C语言第六次上机练习	142	906	3小时15分钟

1. 同时支持至少300个连接。
2. 对大部分响应来说要求在1s内完成（考虑一般的网络连接状况）。
3. 由于比赛中的提交具有突发性的特点，每分钟我们系统应该能够支持评测至少120秒程序。

3.3.3 维护需求

一个软件难免需要人来维护，或者需要对功能进行升级。很多时候，维护或升级的人员因为原先的代码可读性差以致无法顺利地进行进一步操作。因此，系统的程序代码应该具备足够的易读性，具备标准的格式和简洁的代码风格，这样能使维护和升级顺利地进行。同时开发者还需提供详细的说明文档。

第4章 系统概要设计

4.1 系统环境

在开始介绍系统的大体结构之前，首先介绍一下本系统的开发环境：

1. 操作系统：OS X Mountain Lion 10.9.1
2. 部署环境：运行于Virtual Box下的最新版本Arch Linux
3. 建模软件：Visual Paradigm for UML
4. 编辑软件：Eclipse、WebStorm、Vim
5. 版本控制：Git
6. 持续集成：TeamCity^①

本系统涉及到了服务器端、浏览器端、评测器三个模块，它们所使用的编程语言也都不同。服务器端我们采用的是JDK7.0标准下的Java语言，项目通过maven来组织和管理。浏览器端核心框架为AngularJS和jQuery，前端UI采用Bootstrap3，项目通过Grunt来组织和管理，编程语言为Less^②和CoffeeScript^③。评测器采用C++语言，使用32位Linux API实现。

本项目代码托管在Github上^④，评测器内核位于branches文件夹，项目相关文档位于doc文件夹，服务器代码位于trunk文件夹下^⑤。评测器内核不在本文的讨论范围内，我们主要说明下trunk目录的结构。

trunk目录下的pom.xml是maven工程的配置文件，config和script用来保存一些编译脚本和配置文件，所有的代码均在src目录下。

java目录保存java源代码，resources文件夹保存系统配置信息，sql目录下保存数据库脚本，webapp下保存和前端相关的所有文件，包括样式表、图片、JSP文件。

- ① 一款功能强大的持续集成（Continue Integration）工具，可以让团队快速实现持续继承：IDE工具集成、各种消息通知、各种报表、项目的管理、分布式的编译等等。
- ② LESSCSS是一种动态样式语言，属于CSS预处理语言的一种，它使用类似CSS的语法，为CSS的赋予了动态语言的特性，如变量、继承、运算、函数等，更方便CSS的编写和维护。
- ③ CoffeeScript是一套JavaScript的转译语言。受到Ruby、Python与Haskell等语言的启发，CoffeeScript增强了JavaScript的简洁性与可读性。
- ④ 项目主页<http://uestc-acm.github.io/CD0J/>。
- ⑤ 项目最早托管于Google Code上，使用SVN作为版本控制工具，目录结构也是SVN风格的结构。

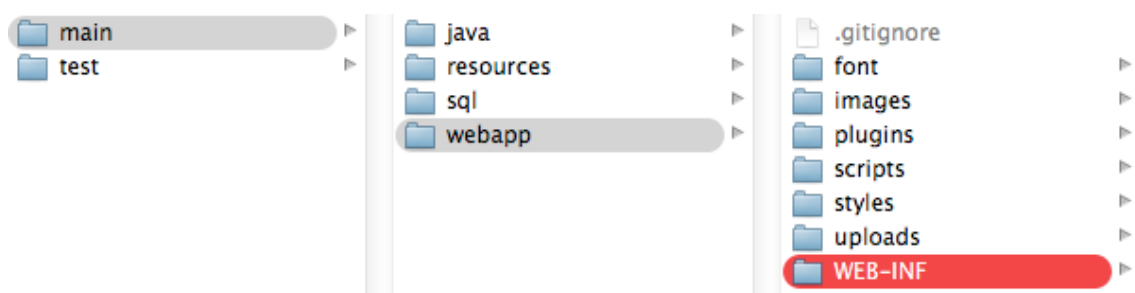


图 4-1 src目录结构

4.2 服务器端框架

4.2.1 服务器端总体技术架构

为确保系统的互操作性、适用性及长期的扩充性，我们应以标准的、开放的要求进行架构。本文所设计的在线评测系统，是在优化、改造原有设计的基础上，借助于分布式的应用模型及先进的MVC体系结构实现的。在服务器端我们集中安置了系统中的数据库、程序和一些其它的组件，而在客户端我们只需要浏览器，同一数据源为用户提供数据查询服务，如此一来也就确保了数据的完整性与及时性。在很多情况下，用户的需要会随时间的推移而改变，因此在业务的处理逻辑出现变化的情况下，我们只需要在服务器端进行程序的修改，随后就可以重新进行发布，这样就方便了程序的研发及发布，也不会对用户产生影响。本系统总体结构如图4-2所示。

客户端发起一个HTTP请求后，经过一系列中间处理最终被分配到该连接对应的控制器上（Controller）。与标准MVC模型不同的是，我们通过一个叫做服务（Service）的中间件来和模型（Model）进行交互，服务调用Hibernate框架的数据访问对象（DAO）来进行数据的持久化操作。在一系列逻辑操作之后，控制器（Controller）根据结果来选择合适的视图（View）返回给客户端。

评测器模块作为一个独立的模块存在于web框架之外，它通过服务（Service）来查找等待评测的任务队列、进行评测和更新任务队列。

4.2.2 服务器端模块结构

根据图4-2的结构图，服务器端的模块可以进一步细化为如下结构：

- 配置模块（config）：负责项目的整体配置。
- 数据库模块（db）：使用Hibernate框架来完成持久化操作。

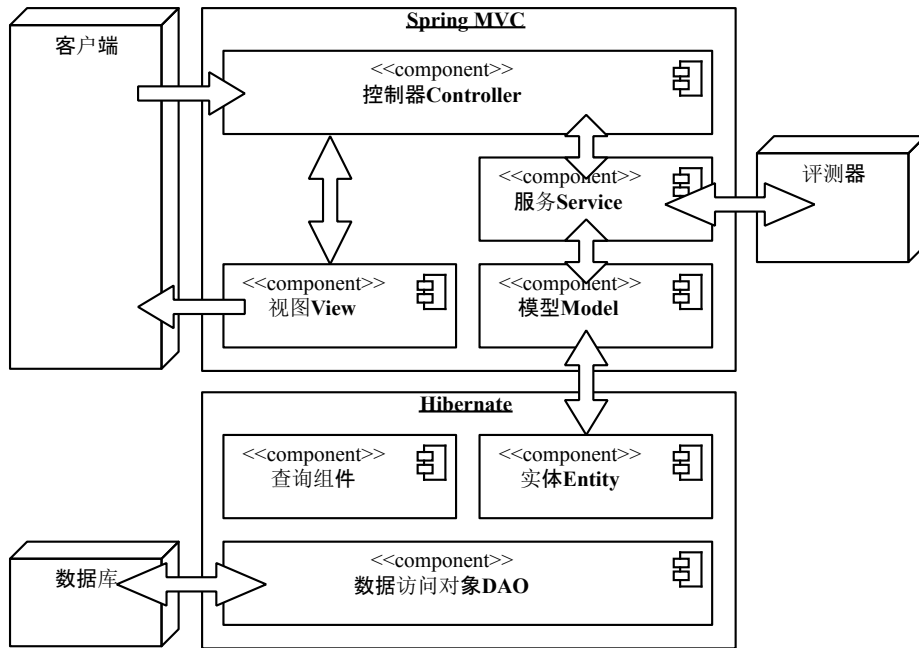


图 4-2 系统总体技术架构

- 评测器模块 (judge)：负责评测任务的调度和执行。
- 服务模块 (service)：充当控制器 (Controller) 和模型 (Model) 的桥梁。
- 实用工具 (util)：提供许多有价值的API。
- 网站模块 (web)：包含了控制器 (Controller) 和许多与服务无关的组件。

4.2.2.1 配置模块

该模块承担的任务是对服务器的基本运行参数进行配置，例如Spring MVC框架的初始化配置、Hibernate框架的属性配置等。

4.2.2.2 数据库模块

该模块中包含了数据访问对象 (DAO) 以及相关的类，例如用于和数据库进行映射操作的实体类 (Entity)，可以转换为HQL语言^①查询条件的条件类 (Condition) 和数据传输对象 (DTO)。

4.2.2.3 评测器模块

该模块包含了一个评测器服务 (Judge Service)，它产生许多个评测线程来进行多线程评测。

① HQL是Hibernate Query Language的简写，即hibernate 查询语言：HQL采用面向对象的查询方式。

4.2.2.4 服务模块

这个模块的主要作用是提供丰富的模型操作API，例如模型实例的新建、修改、查找等操作。

4.2.2.5 实用工具

这个模块包含了所有供内部使用的公共API。

4.2.2.6 网站模块

该模块最主要的部分是控制器（Controller）模块，控制器负责处理来自客户端的HTTP请求，并通过一定的逻辑选择合适的服务（Service）来完成用户的请求，同时根据情况将合适的视图（View）返回给用户。

4.3 浏览器端框架

4.3.1 浏览器端总体技术架构

传统的Web应用允许用户端填写表单（form），当提交表单时就向Web服务器发送一个请求。服务器接收并处理传来的表单，然后送回一个新的网页，但这个做法浪费了许多带宽，因为在前后两个页面中的大部分HTML码往往是相同的。由于每次应用的沟通都需要向服务器发送请求，应用的回应时间依赖于服务器的回应时间。这导致了用户界面的回应比本机应用慢得多。

与此不同，AJAX（Asynchronous JavaScript and XML^①，异步的JavaScript与XML技术）应用可以仅向服务器发送并取回必须的数据，并在客户端采用JavaScript处理来自服务器的回应。因为在服务器和浏览器之间交换的数据大量减少（大约只有原来的5%），服务器回应更快了。同时，很多的处理工作可以在发出请求的客户端机器上完成，因此Web服务器的负荷也减少了。

本系统的服务器端提供以下几种资源：

1. 网页资源

这类资源包含网页HTML代码、样式列表（CSS）、浏览器脚本（JavaScript）、图片和字体，通过GET方式获得。

^① 实际上数据格式可以由JSON代替，进一步减少数据量，形成所谓的AJAJ。本系统使用的便是更加轻便的JSON数据。

2. 数据资源

这类资源均为JSON格式的数据，它有两种不同的获取方式：

(a) GET方式

这种方式和获取网页资源操作相同，区别在于服务器返回的是JSON数据。

(b) POST方式

当客户端需要发送数据给服务器时，需要通过POST方式来传递数据，举个例子来说，当用户在登录窗口登录时，浏览器会将登录窗口的表单打包成一段JSON格式的数据，然后通过POST方式发送给服务器，服务器将登录状态等信息以JSON格式返回给前端，完成一次登录操作。

在2.3.3中我们提到了AngularJS框架，它是本系统最底层的框架。

4.3.2 网站结构

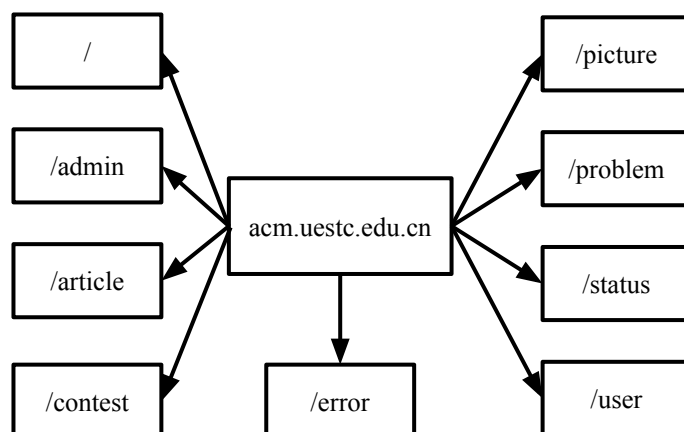


图 4-3 网站结构图

目前本项目域名为`http://acm.uestc.edu.cn/`，根据功能需要我们将其划分若干部分，如图4-3所示。

4.3.3 浏览器端开发流程

本系统所用到的样式表和脚本较多，这些文件统一使用GruntJS来维护，位于`trunk/src/main/webapp/plugins/cdoj`下。我们用LESS作为样式表的编程语言，CoffeeScript作为脚本的编程语言，编译流程如图4-5所示。

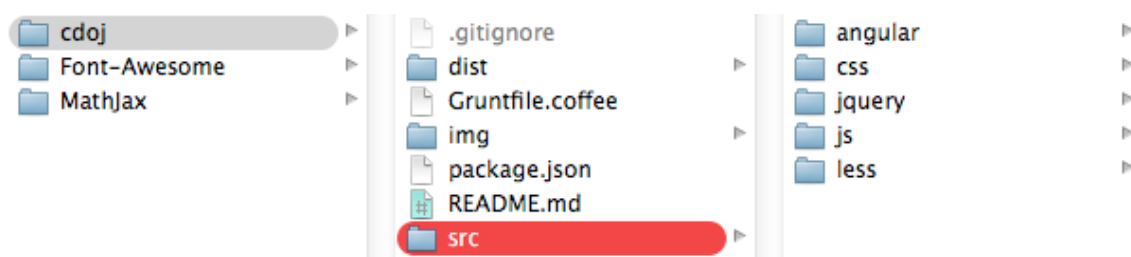


图 4-4 前端文件目录结构

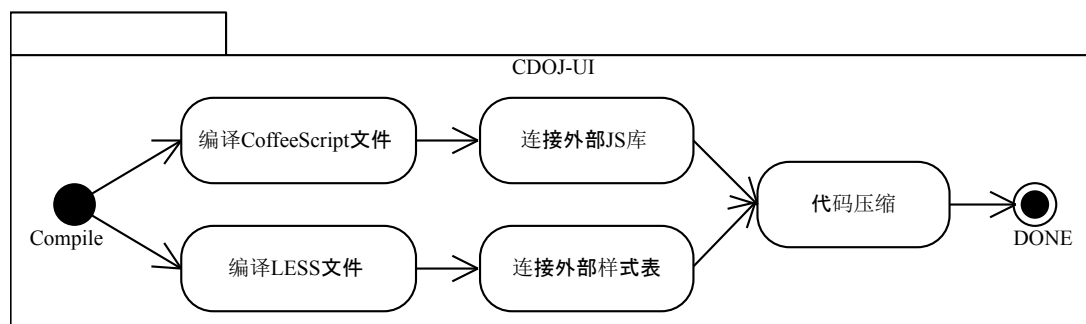


图 4-5 GruntJS编译活动图

项目中与AngularJS框架相关的脚本文件统一放置在src/angular子目录下。LESS文件在src/less下。还有部分功能我们用jQuery实现，位于src/jquery下。src/css和src/js被用来保存外部的CSS和JS文件。编译后的文件在dist目录下。

第5章 系统详细设计

5.1 数据库详细设计

本系统数据库较为复杂，这里我们简述一下各个数据表的作用，如表5-1所示。我们将这些数据表分成了五个部分，数据结构图见图5-1、图5-2、图5-3、图5-4和图5-5。

表 5-1 Entity表

表	作用
Article	文章的内容和基本信息
Code	用户提交的代码
CompileInfo	代码的编译信息
Contest	比赛的基本信息
ContestProblem	比赛和题目的对应关系
ContestTeamInfo	参赛队伍的信息
ContestUser	比赛的注册用户
Department	学校的部门信息
Language	可以使用的语言以及参数
Message	用户短消息
Problem	题目内容和基本信息
ProblemTag	题目和分类标签的对应关系
Status	代码的评测状态
Tag	分类标签
User	用户信息
UserSerialKey	用户激活码

5.2 服务器端详细设计

在服务器端系统设计过程中，最重要的是根据需求分析及用例模型构建系统静态模型和动态模型。顺序图展示对象之间的交互，这些交互是指在场景或用例

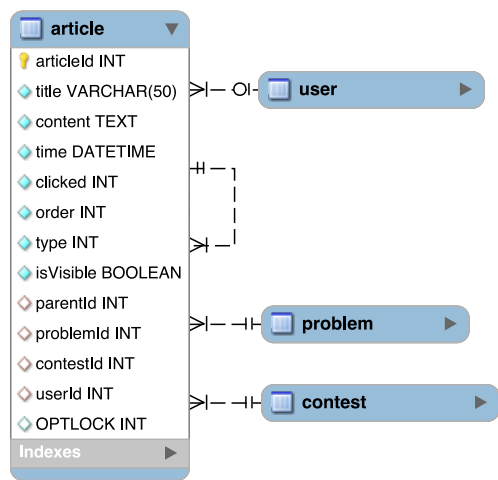


图 5-1 Article 相关数据库结构图

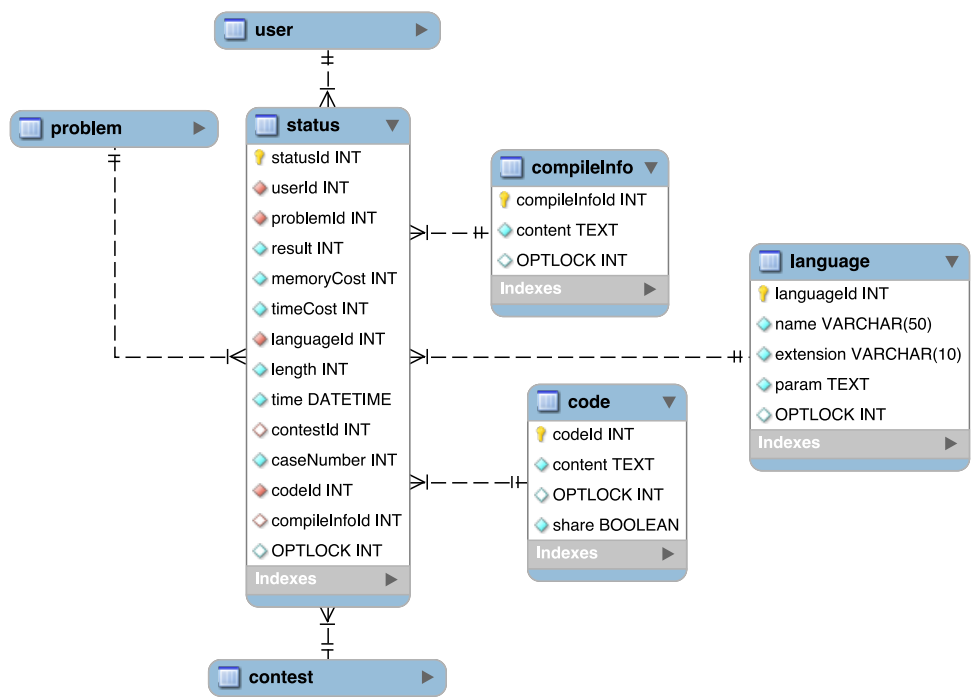


图 5-2 Status 相关数据库结构图

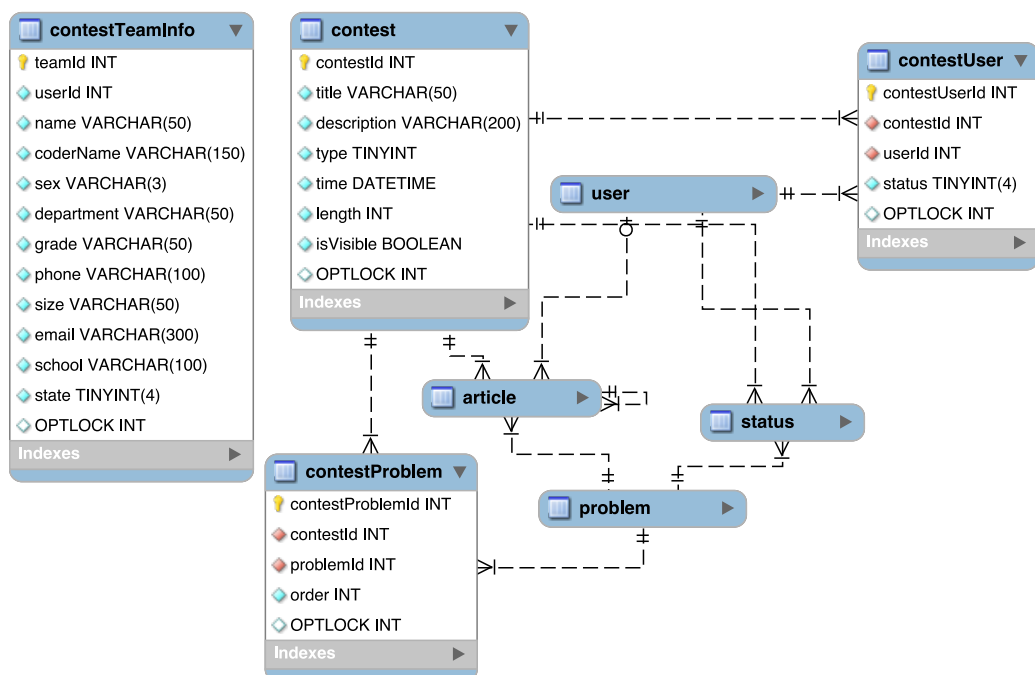


图 5-3 Contest相关数据库结构图

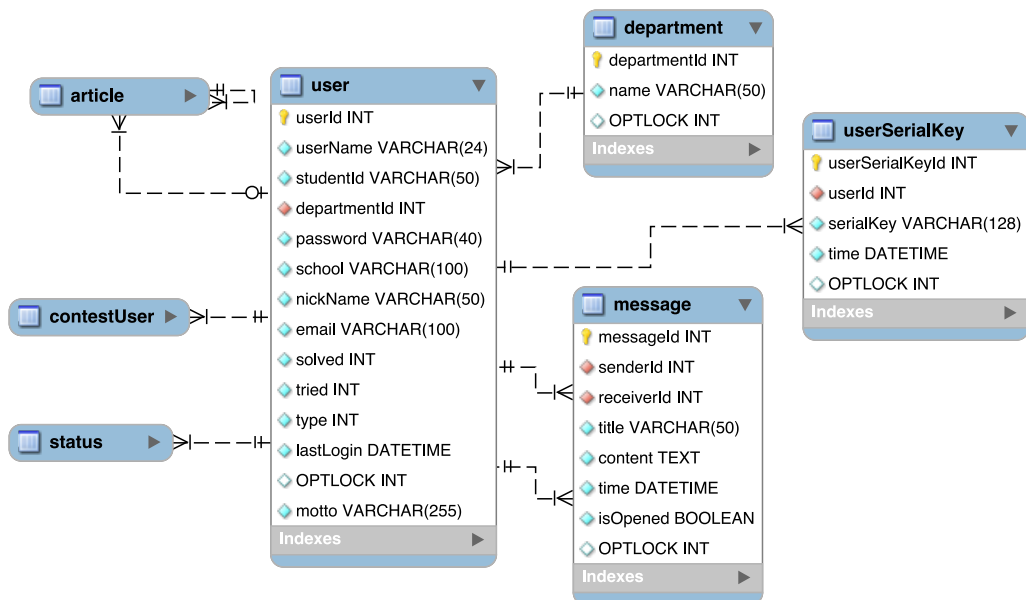


图 5-4 User相关数据库结构图

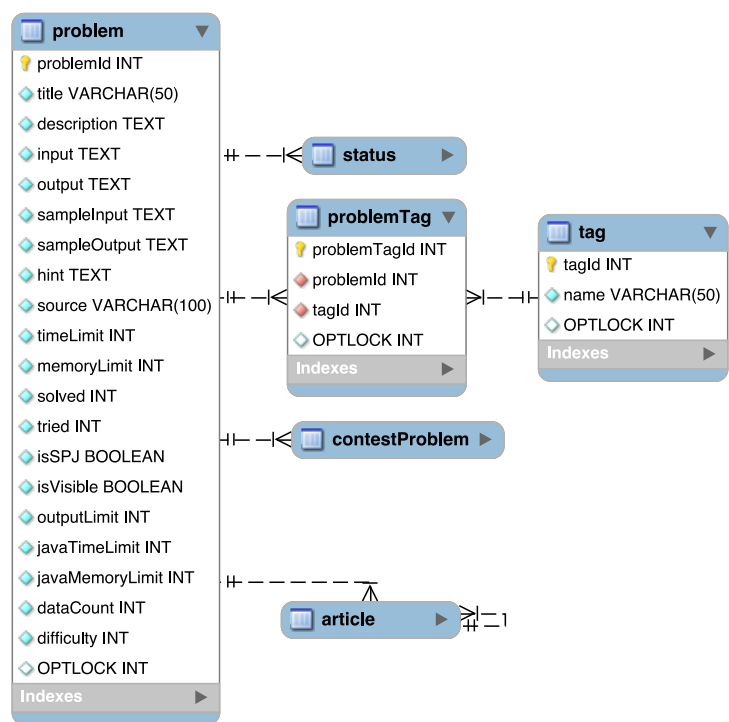


图 5-5 Problem相关数据库结构图

的事件流中发生的。协作图是一种交互图，强调的是发送和接收消息的对象之间的组织结构，使用协作图来说明系统的动态情况。状态图说明对象在它的生命周期中响应事件所经历的状态序列，以及它们对那些事件的响应。活动图是主要用于业务建模时，用于详述业务用例，描述一项业务的执行过程。设计时，描述操作的流程[6]。

5.2.1 系统顺序图

我们用顺序图来说明用户的一次操作在系统之中是如何完成的，通过顺序图来对本系统的工作机制提供一个大概的说明，图5-6说明了用户的一次完整操作。

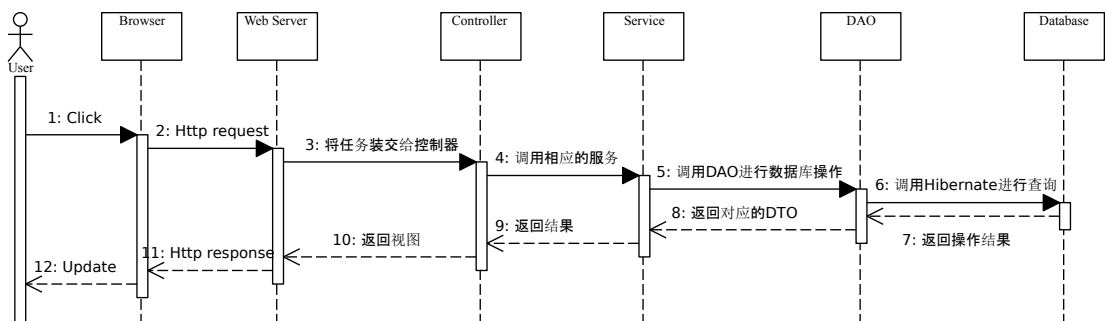


图 5-6 系统顺序图

5.2.2 系统包图

包图说明了系统各个模块之间的依赖关系，在4.2.2中我们已经介绍过了系统的模块结构，根据这个结构，本系统的包结构如图5-7所示。由于本系统内容比较多，我们这里先给出大概的结构，后面再一一详细描述。

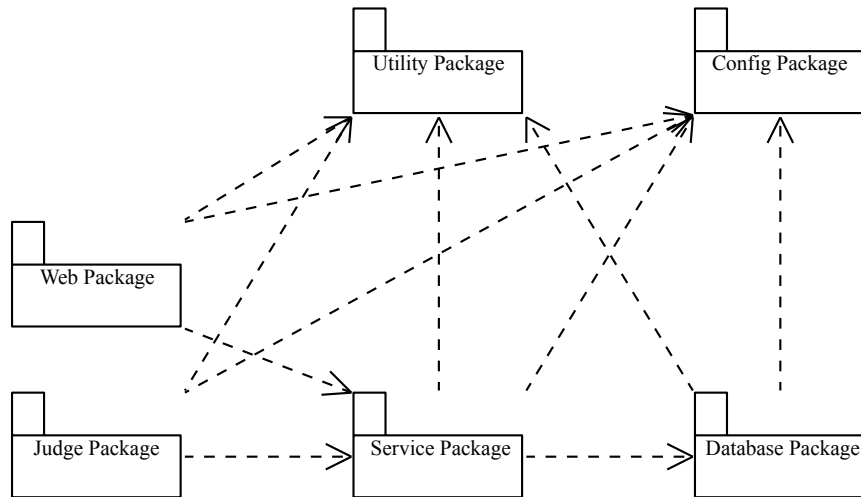


图 5-7 包图

5.2.3 Config Package详细设计

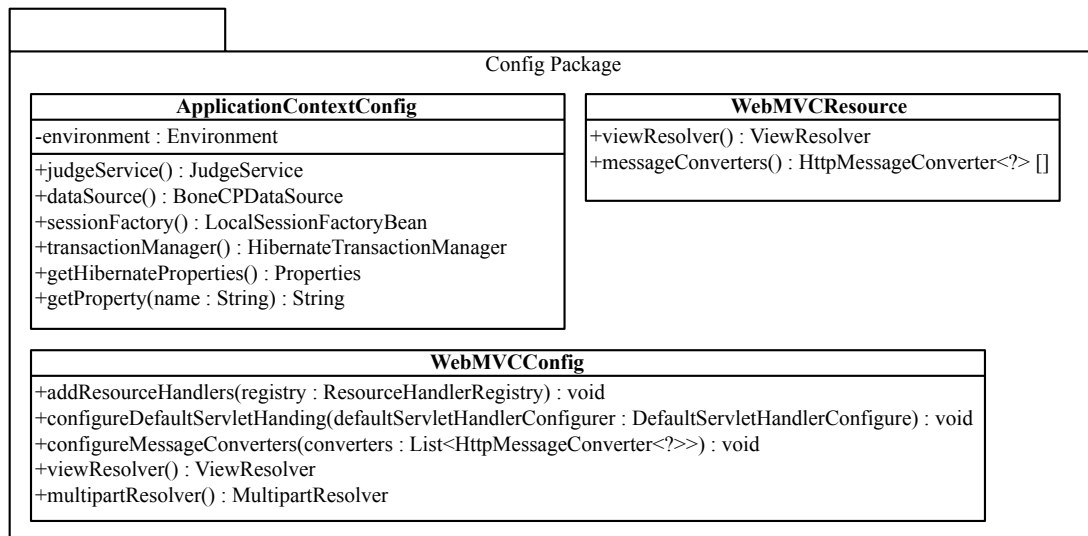


图 5-8 Config Package类图

5.2.3.1 ApplicationContextConfig

ApplicationContextConfig.java见附录A.1。

Spring框架有两种配置方式，一种是通过XML配置文件进行配置，这种方式将所有的配置信息写入一个指定的XML文件之中，这种方式略显麻烦，在本文中我们采用了另外一种方式，这种方式是利用Java的Annotation机制来进行配置。

系统启动时默认将resources.properties文件^①中的键值对初始化成一个Environment实例，我们可以通过getProperty(String): String方法来获得对应的值。

有了**Environment**实例，我们就可以将配置信息从代码中分离开来。

5.2.3.2 WebMVCResource

WebMVCResource.java见附录A.3。

为了方便进行测试，我们将一些比较特殊的资源从WebMVConfig.java中独立开来，放到WebMVCResource.java中。这里主要做了两件事情：一个是配置视图解析器，在这里我们设置视图地址的前缀和后缀，方便Controller调用视图。另外一件事就是配置了JSON数据的转换器，用于解析和构建JSON数据，这里我们使用了fastjson^②。

5.2.3.3 WebMVConfig

WebMVConfig.java见附录A.4 这个文件是SpringMVC框架的配置文件，与之前的ApplicationContextConfig类似，这里配置了与Web相关的参数。

5.2.4 Database Package详细设计

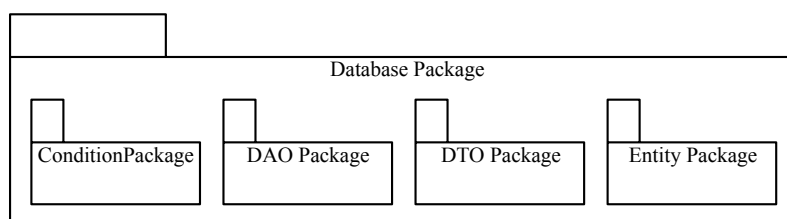


图 5-9 Database Package包图

这个包在MVC模型中处于Model层，所有与数据库有关的API都被包含在里面。

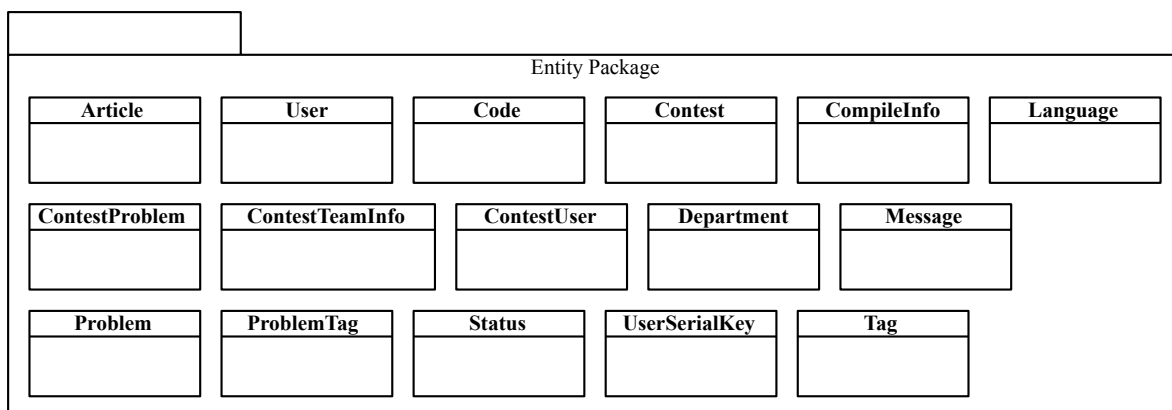


图 5-10 Entity Package类图

5.2.4.1 Entity

Entity即为实体，对应着MVC模型中的Model，它和数据库中的内容有着直接的一对一映射关系。

5.2.4.2 DTO

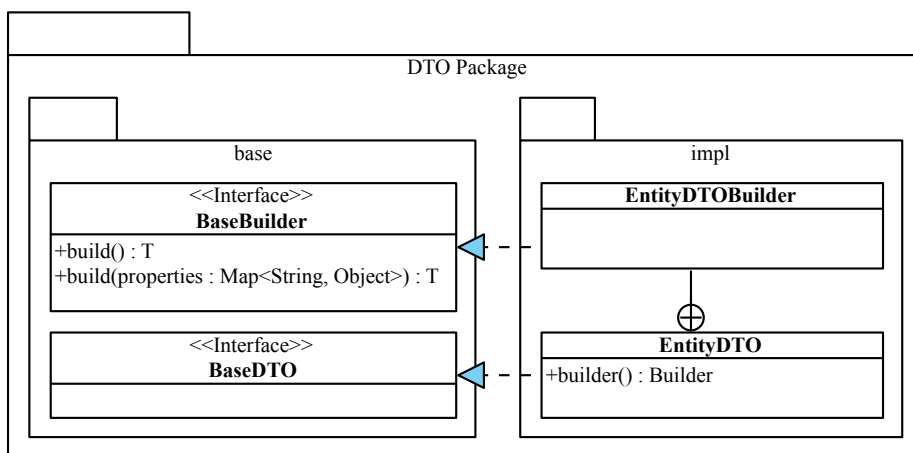


图 5-11 DTO Package类图

数据传输对象DTO有两种，一种是客户端向服务器传输的数据，一种是模型向上层传输的数据。前者我们通过一个简单的类可以实现。

Hibernate自带的数据库API较为复杂，为了提升效率和简化代码维护成本，我们自己构建了一套用来提取数据库数据的工具。在这类DTO中，我们使用了一个@Fields注解来注明这个DTO的信息来自数据库中的哪些域，然后通过这

① resources.properties见附录A.2。

② Fastjson是一个Java语言编写的JSON处理器,由阿里巴巴公司开发。

个field来构建HQL查询语言的**SELECT**命令。如下所示为UserListDTO.java的部分内容：

```
001 @Fields({"userId", "email", "userName", "nickName", "type", 2
002 "school", "motto", "lastLogin", "solved", "tried"})
003 public class UserListDTO implements BaseDTO<User> {
004     // Codes
005 }
```

对应生成的HQL语句为**SELECT userId, email, userName, nickName, type, school, motto, lastLogin, solved, tried FROM User**，配合接下来要介绍到的Condition，我们可以组合出基本的HQL查询语句。

在得到这些域后，我们调用对应的EntityDTOBuilder的build方法来得到这些值。

5.2.4.3 Condition

我们在本系统中使用Hibernate作为持久层框架，它提供了强大的HQL查询语言，Condition包的主要功能就是提供了Condition组件，它可以翻译成HQL查询语言的where条件，来限定检索范围。

根据实际情况，本系统设计的Condition支持三种条件：

1. Order条件：用来限定返回结果的顺序。
2. PageInfo条件：用来实现返回结果的分页功能。
3. 普通条件：既Entry，它既可以是一条普通的条件，如**userId = 5**，也可以是一个Condition。在枚举类型ConditionType中，我们定义了许多常用的条件，如等于、不等于、小于、like、属于等等。

对于每个数据库实体类型Entity，都有一个对应的EntityCondition类，如Problem实体有对应的ProblemCondition。这些EntityCondition类都必须继承自BaseCondition类，并且实现它的**getCondition()**方法。

对于一些比较简单的条件，我们提供了一个@Exp注解，例如在StatusCondition.java中有如下变量：

```
001 /**
002  * Minimal status id.
003  */
```

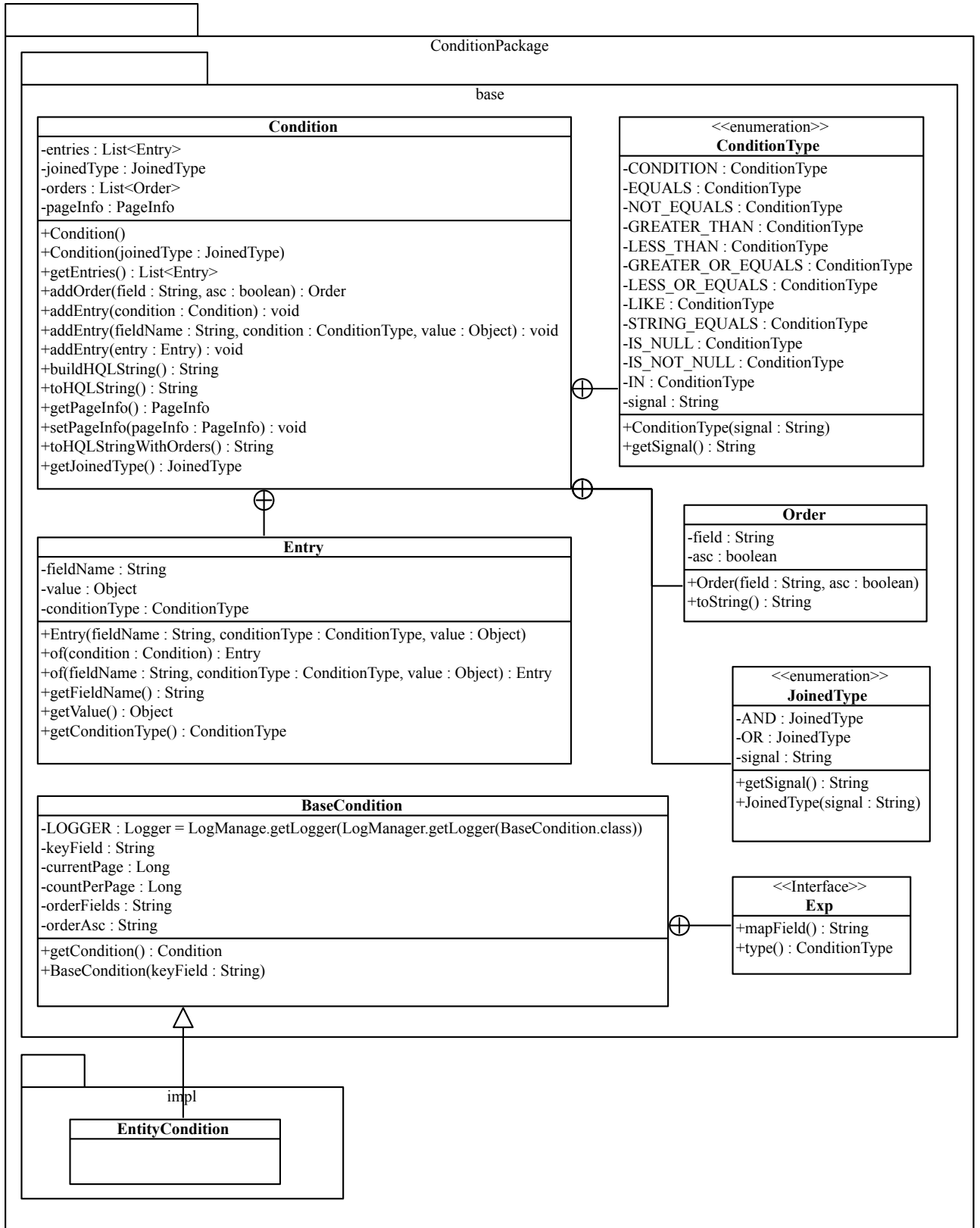


图 5-12 Condition Package类图

```
004 @Exp(mapField = "statusId", type = Condition.ConditionType.2
005 GREATER_OR_EQUALS)
006 public Integer startId;
007
008 /**
009  * Submit user id.
010  */
011 @Exp(mapField = "userByUserId", type = Condition.2
012 ConditionType.EQUALS)
013 public Integer userId;
```

如果这两个成员变量不是空，那么最后我们会得到一个形式如同**WHERE ...
userId >= userId and userByUserId = userId ...**的HQL查询语句。

对于一些比较复杂的条件，开发者可以在**getCondition()**方法中实现复杂的逻辑。

5.2.4.4 DAO

DAO提供了基础的数据库操作API，例如添加数据、修改、删除、查询等等，通过与DTO和Condition的配合使用，我们可以方便的进行数据库操作，而不需要为每种情况都生成一段冗长的HQL语句。

5.2.5 Service Package详细设计

Service为上层应用提供了一系列特定的数据库操作，根据接口隔离原则[7]，我们不希望上层应用直接调用底层的数据库API来进行操作，我们通过Service来隔离它们。在这里，每个EntityService都完成与指定Entity相关的操作，不允许出现跨Entity的调用。

5.2.6 Judge Package详细设计

这个Package包含了与评测器服务相关的内容。

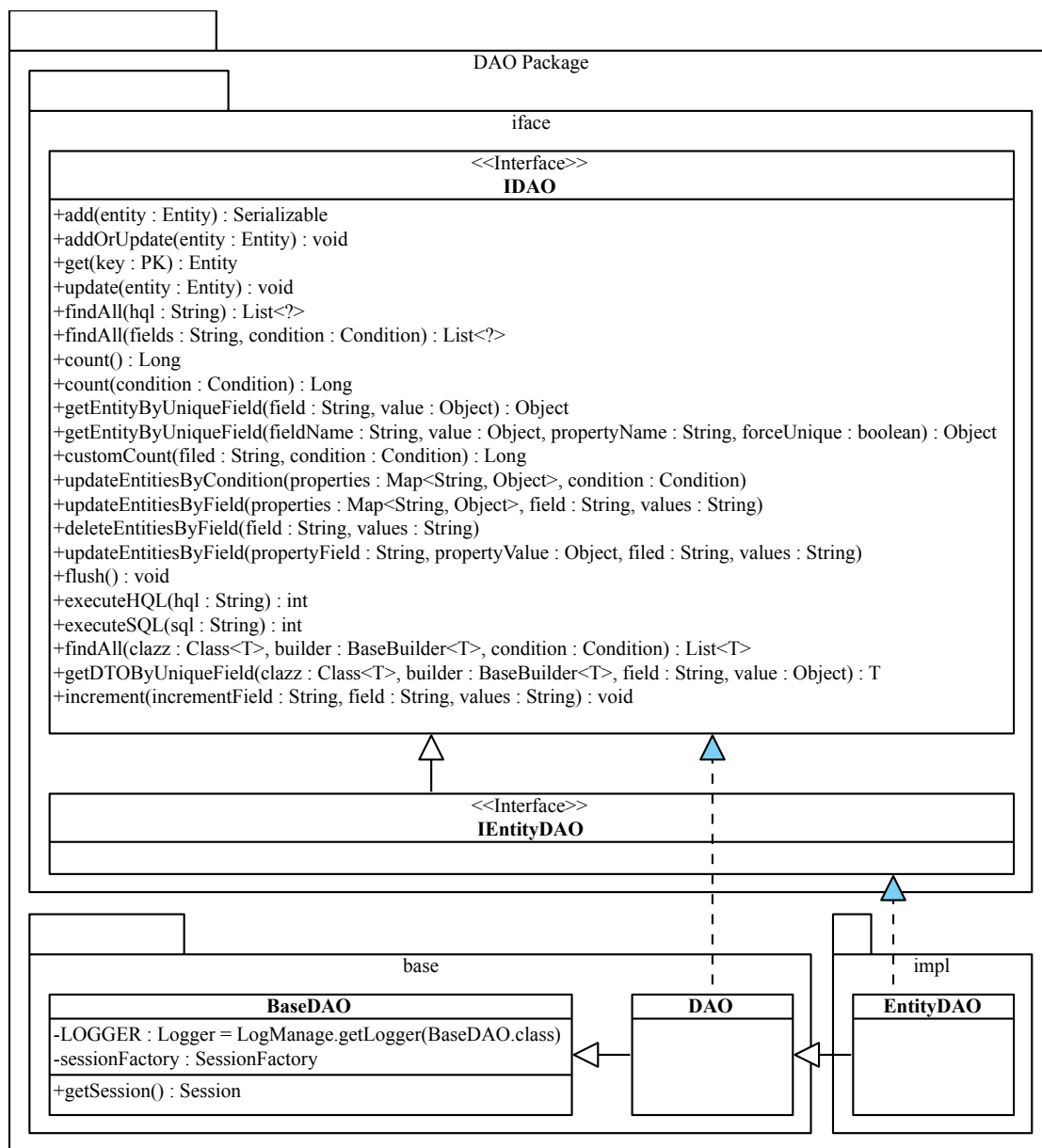


图 5-13 DAO Package类图

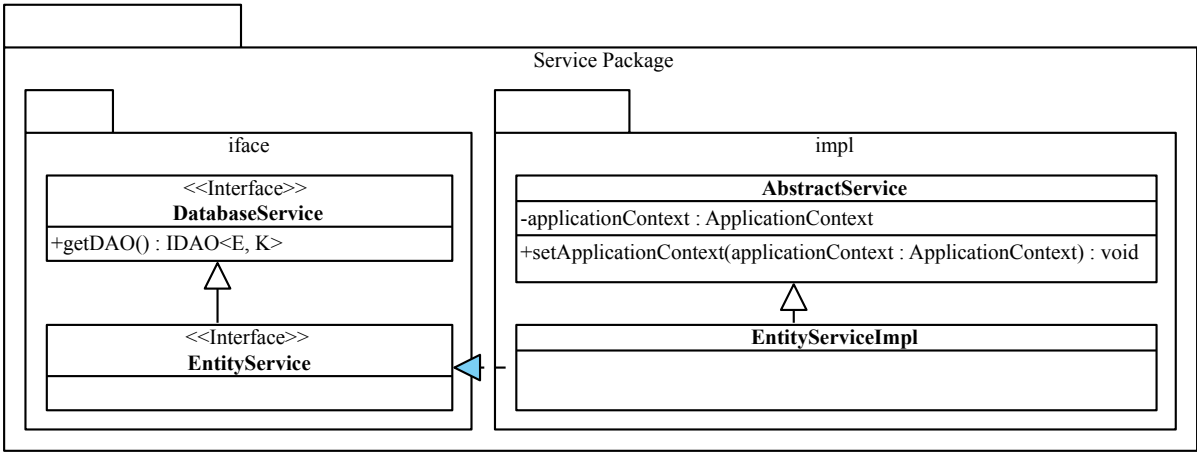


图 5-14 Service Package类图

表 5-2 Judge Core参数

参数	作用
-u	指定任务ID
-s	指定源代码路径
-n	指定题目ID
-D	指定数据文件夹地址
-d	指定运行的工作目录
-t	指定运行时间限制
-m	指定运行内存限制
-o	指定输出大小限制
-S	开启SPJ选项
-l	指定语言类型
-I	指定测试用例的输入文件
-O	指定-I中测试用例的对应标准输出文件
-C	是否需要编译

5.2.6.1 评测器内核

评测器内核负责编译、运行、评测用户代码，是一个控制台程序，通过命令行参数来设定评测任务。评测器内核的主函数参数表见表5-2。

评测结束后，它返回三个整数，分别代表评测结果编号（含义见表5-3）、内存开销、时间开销。

表 5-3 Judge Core结果代码

	编号	结果
OJ_WAIT	0	等待评测开始
OJ_AC	1	结果正确
OJ_PE	2	结果正确但格式错误
OJ_TLE	3	运行时间超过限制
OJ_MLE	4	运行内存超过限制
OJ_WA	5	错误的答案
OJ_OLE	6	输出数据过大
OJ_CE	7	编译错误
OJ_RE_SEGV	8	段错误
OJ_RE_FPE	9	浮点溢出错
OJ_RE_BUS	10	总线错误
OJ_RE_ABRT	11	意外中断
OJ_RE_UNKNOWN	12	未知错误
OJ_RF	13	调用了非法函数
OJ_SE	14	系统错误
OJ_RE_JAVA	15	Java运行时错误

5.2.6.2 JudgeService

JudgeService在系统启动时开始运行^①，在这个类中我们用队列judgeQueue作为评测器的调度队列。它生成schedulerThread线程用来等待评测任务的到来，它每隔一定的时间间隔（在这里我们设置为3秒）调用StatusService查找所有等待测试的任务，将其标记为OJ_JUDGING状态，并加入到judgeQueue中。它还配置了若干个JudgeThread线程用来进行多线程评测操作。每个JudgeThread不停的扫描judgeQueue，直到任务的到来，它首先将代码保存至工作目录下，然后构造

① 见附录A.1的22-26行。

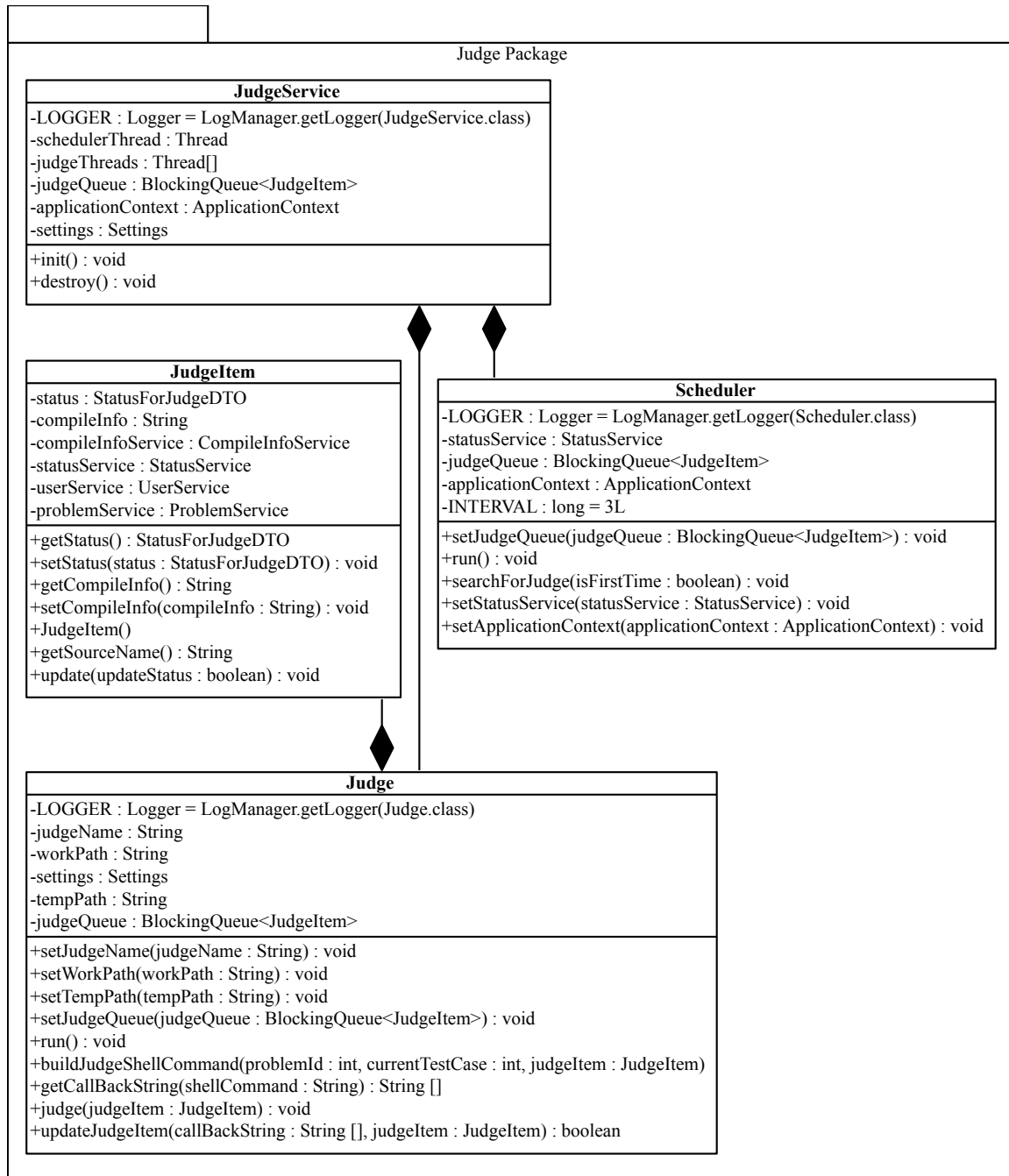


图 5-15 Judge Package类图

控制台命令调用**Runtime.getRuntime().exec(shellCommand)**来和评测器内核交互，并得到结果，然后依据结果来做出相应的更新。如图5-16所示。

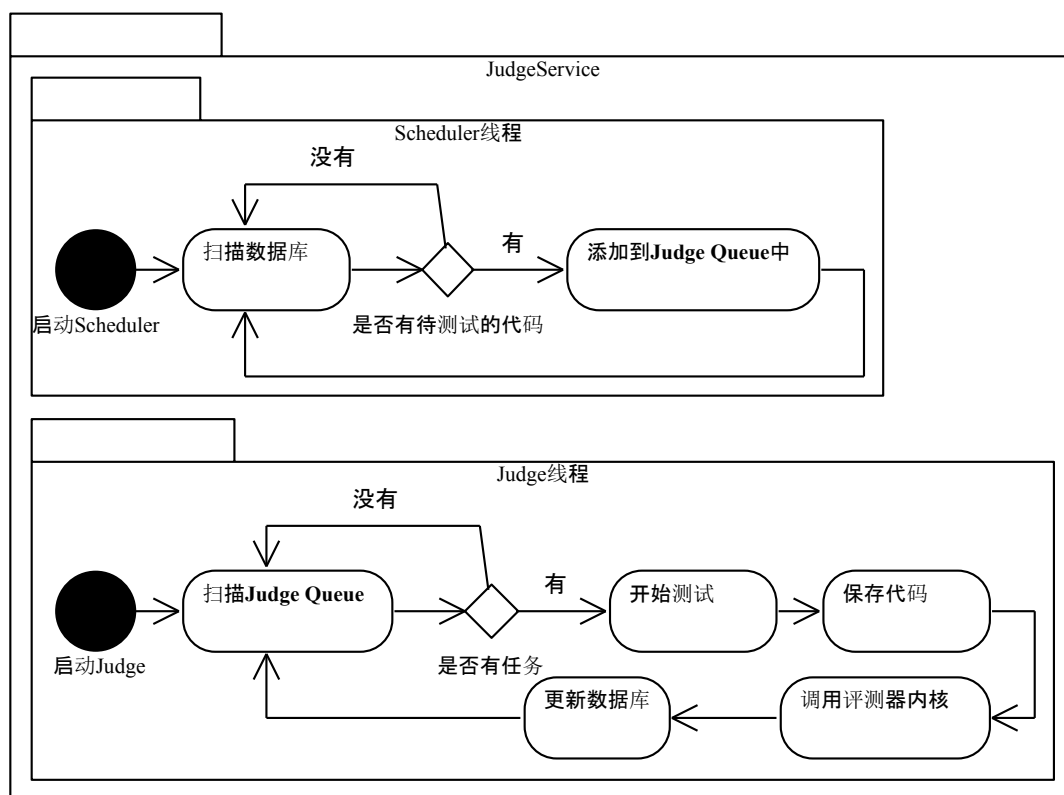


图 5-16 Judge Service活动图

每个题目可能有若干组测试数据，这些数据保存在**WEB-INF**下的**data**文件夹，按题目编号分别保存在不同的子文件夹下，从1开始编号，每个测试用例的输入文件以**.in**为后缀，对应的输出文件以**.out**结尾。

有些开放性的题目存在多个答案，这个时候评测器需要用另外一段程序来判断它是否符合要求。还有一种情况就是结果为浮点数的时候，为了排除浮点误差对结果正确性的干扰，我们需要将用户程序运行结果和标准结果进行比较，一般来说误差小于 10^{-8} 就可以认为两者相同。如果需要进行**Special Judge**，还需要提供一个**spj.cc**文件指定**Special Judge**的代码（见附录A.5）。

出于使用上的考虑，我们在表5-3的基础上添加了三中评测状态：

- OJ_JUDGING，编号：16，作用：测评器启动
- OJ_RUNNING，编号：17，作用：正在运行代码
- OJ_REJUDGING，编号：18，作用：重新评测中

5.2.7 Web Package详细设计

Web Package主要包含的是控制器，以及为控制器服务的一些模块，比如权限验证模块。

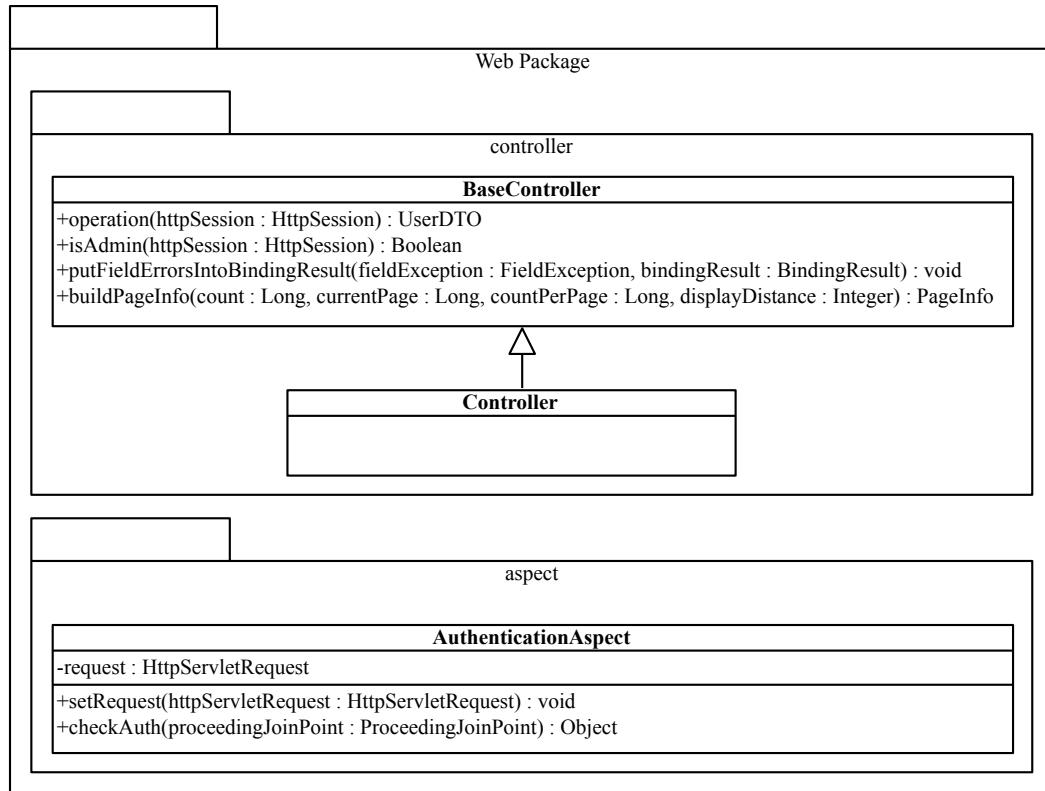


图 5-17 Web Package类图

5.2.7.1 AuthenticationAspect

这是本系统的权限验证模块，我们采用了面向侧面的程序设计^①思想来完成，既在每个Controller之前“切入”一段指定的代码来进行权限验证。这部分我们使用AspectJ框架来完成，它是以代理（Proxy）的形式实现的。代码见附录A.6。

我们在图5-18给出了验证成功和验证失败的两个顺序图，可以看到当验证失败时，我们不会调用原Controller，达到了权限限制的目的。

① aspect-oriented programming, AOP, 又译作面向方面的程序设计、观点导向编程，是计算机科学中的一个术语，指一种程序设计范型。该范型以一种称为侧面（aspect，又译作方面）的语言构造为基础，侧面是一种新的模块化机制，用来描述分散在对象、类或函数中的横切关注点（crosscutting concern）。

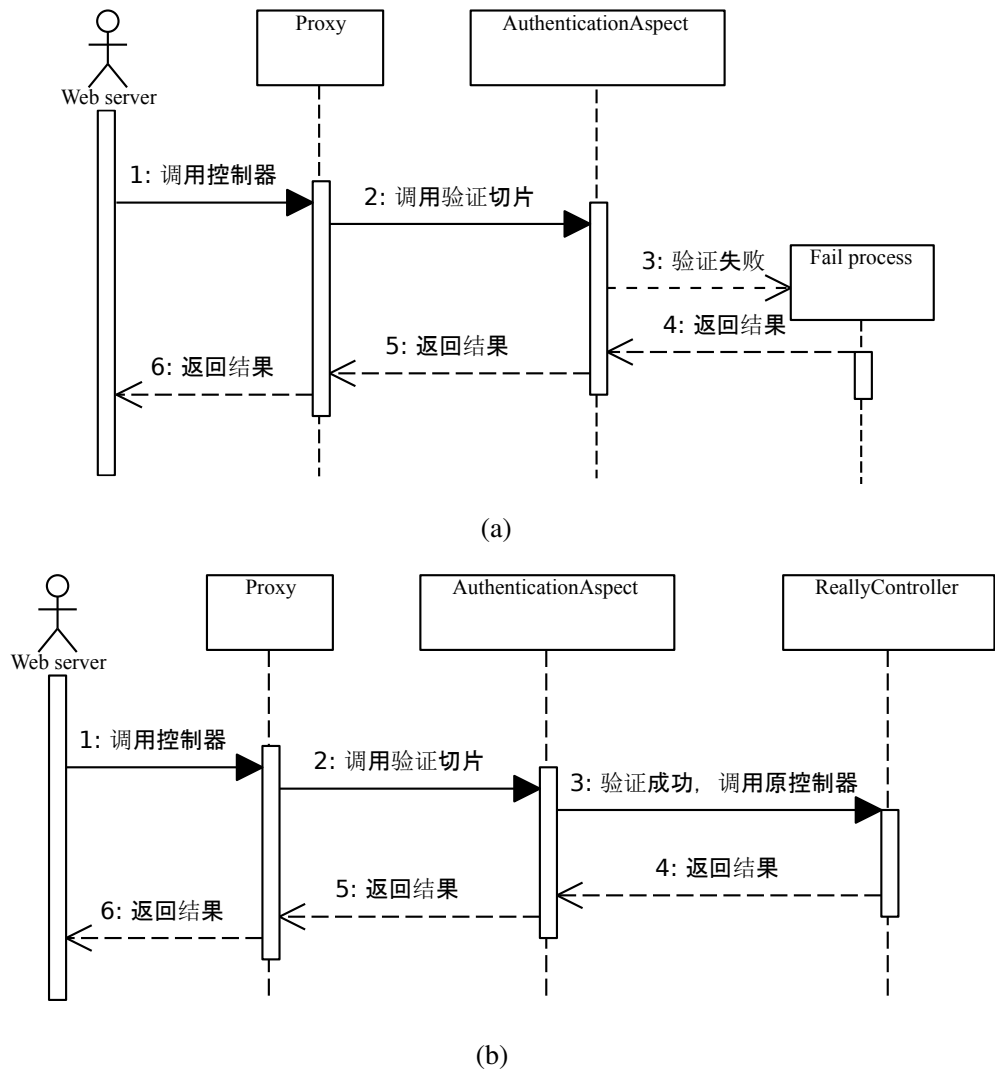


图 5-18 权限验证顺序图

(a)一次失败的权限验证；(b)一次成功的权限验证

5.2.7.2 Controller

Spring MVC框架提供了很强大的Controller，我们只需要在一个类上使用**@Controller**注解就可以将一个类声明为控制器。在本项目中，我们按照域名的分布来划分Controller，在前面的图4-3中我们已经给出网站的结构。

根据返回值的不同，控制器分为两种类型：一种是返回一个代表视图地址的**String**的控制器，一种是返回一个代表JSON数据的**@ResponseBody Map<String, Object>**的控制器。

视图保存在项目的webapp目录下，如**webapp/WEB-INF/views/index/index.jsp**，此视图对应的地址为**/WEB-INF/views/index/index.jsp**，但是我们在WebMVCResource.java（附录A.3）中的**viewResolver()**设置了地址的前缀和后缀，我们可以将其简写为**index/index**。如果控制器返回了一个视图地址，那么服务器会将对应的页面返回给用户。

对于返回**@ResponseBody Map<String, Object>**的控制器，FastJson框架会将这个**@ResponseBody Map<String, Object>**转换成JSON各式的文本返回给用户。

下面我们用户登陆相关的控制器来说明它是如何工作的（代码省略了其余部分）。

```

001 // 声明控制器
002 @Controller
003 // 声明控制器的域名
004 @RequestMapping("/user")
005 public class UserController extends BaseController {
006
007     // /user/login对应的方法
008     @RequestMapping("login")
009     // 登陆权限设置为无
010     @LoginPermit(NeedLogin = false)
011     public
012     // 返回值为JSON数据
013     @ResponseBody
014     Map<String, Object> login(HttpSession session,
015                             @RequestBody @Valid >

```

```

016                UserLoginDTO userLoginDTO,
017                BindingResult validateResult) {
018    Map<String, Object> json = new HashMap<>();
019    // 表单验证失败
020    if (validateResult.hasErrors()) {
021        json.put("result", "field_error");
022        json.put("field", validateResult.getFieldErrors());
023    } else {
024        try {
025            // 获取登陆用户的信息
026            UserDTO userDTO = userService.getUserDTOByUserName(
027                userLoginDTO.getUserName());
028            // 密码验证
029            if (userDTO == null || !userLoginDTO.getPassword().
030                equals(userDTO.getPassword())) {
031                throw new FieldException("password", "User or
032                password is wrong, please try again");
033            }
034            // 更新用户
035            userDTO.setLastLogin(new Timestamp(new Date().
036                getTime() / 1000 * 1000));
037            userService.updateUser(userDTO);
038
039            // 将用户信息放入Session中
040            session.setAttribute("currentUser", userDTO);
041
042            // 构造返回数据
043            json.put("userName", userDTO.getUserName());
044            json.put("type", userDTO.getType());
045            json.put("email", userDTO.getEmail());
046            json.put("result", "success");

```

```
047     } catch (FieldException e) {
048         // 如果存在表单错误
049         putFieldErrorsIntoBindingResult(e, validateResult);
050         json.put("result", "field_error");
051         json.put("field", validateResult.getFieldErrors());
052     } catch (AppException e) {
053         // 如果发生其它错误
054         json.put("result", "error");
055         json.put("error_msg", e.getMessage());
056     }
057 }
058 // 返回结果
059 return json;
060 }
061
062 }
```

我们使用**@RequestMapping**注解来声明方法所对应的网址，**@LoginPermit**定义了该地址的权限。因为Spring通过反射机制来进行自动注入，控制器方法的参数可以以任意顺序排列而不需要指定顺序。在login方法中，除了**HttpSession session**参数以外，另外两个和前端POST来的数据相关，**@RequestBody @Valid UserLoginDTO userLoginDTO**用来保存前端POST来的数据，我们使用了Java验证框架来对前端传递的数据进行一个初步的合法性验证，**@Valid**注解说明了我们需对**userLoginDTO**进行验证，验证的结果保存在**BindingResult validateResult**中，下面是**UserLoginDTO**的部分代码：

```
001 /**
002  * DTO post from user login form.
003  */
004 public class UserLoginDTO {
005
006     /**
007      * Input: user name
```

```

008     */
009     // 非空验证
010     @NotNull(message = "Please enter your user name.")
011     // 正则表达式验证
012     @Pattern(regexp = "\\b^[a-zA-Z0-9_]{4,24}$\\b",
013         message = "Please enter 4-24 characters consist of A-Z, a-z, 0-9 and '_'.")
014     private String userName;
015
016
017     /**
018      * Input: password
019      */
020     // 非空验证
021     @NotNull(message = "Please enter your password.")
022     // 长度验证
023     @Length(min = 40, max = 40, message = "Please enter your 2
024     password.")
025     private String password;
026
027 }

```

顺利登陆之后，前端可以接收到类似如下格式的一段数据：

```

001 {"email":"muziriyun@qq.com","result":"success","type":1,2
002 "userName":"UESTC_Izayoi"}

```

5.2.7.3 网站地图

上面我们举例说明了UserController的login方法的实现，限于篇幅限制其余的部分我们不一一描述，在这里我们给出整个网站的网站地图，一共有47个不同的方法，我们只介绍各个控制器的作用和返回给前端的数据类型。

表 5-4 网站地图

地址	方法	作用	返回类型
AdminController			
/admin/	index	管理员面板	HTML
ArticleController			
/article/data/	data	文章数据	JSON
/article/show/	show	文章页面	HTML
/article/search	search	文章查找	JSON
/article/editor/	editor	文章编辑器	HTML
/article/edit	edit	文章编辑	JSON
/article/operator	operator	文章操作	JSON
ContestController			
/contest/status/	status	比赛评测结果	JSON
/contest/rankList/	rankList	比赛排名	JSON
/contest/data/	data	比赛数据	JSON
/contest/show/	show	比赛页面	HTML
/contest/list/	list	比赛列表	HTML
/contest/search	search	比赛查找	JSON
/contest/operator/	operator	比赛操作	JSON
/contest/editor/	editor	比赛编辑器	HTML
/contest/edit	edit	比赛编辑	JSON
ErrorController			
/error/authenticationError	authenticationError	权限错误页面	HTML
IndexController			
/	index	主页	HTML
/globalData	globalData	全局变量	JSON
PictureController			
/picture/uploadPicture/	uploadPicture	图片上传	JSON
ProblemController			

接下页

接上页

地址	方法	作用	返回类型
/problem/data/	data	题目数据	JSON
/problem/show/	show	题目页面	HTML
/problem/list	list	题目列表	HTML
/problem/search	search	题目查找	JSON
/problem/operator/	operator	题目操作	JSON
/problem/query/	query	题目数据	JSON
/problem/editor/	editor	题目编辑器	HTML
/problem/edit	edit	题目编辑	JSON
/problem/uploadProblemDataFile/	uploadProblemData	题目测试数据上传	JSON
StatusController			
/status/list	list	评测状态列表	HTML
/status/search	search	评测状态查找	JSON
/status/count	count	评测状态统计	JSON
/status/rejudge	rejudge	重新评测	JSON
/status/submit	submit	提交代码	JSON
/status/info/	info	评测状态信息	JSON
UserController			
/user/login	login	用户登录	JSON
/user/logout	logout	用户登出	JSON
/user/register	register	用户注册	JSON
/user/list	list	用户列表	HTML
/user/search	search	用户查找	JSON
/user/center/	center	用户页面	HTML
/user/edit	edit	用户账户修改	JSON
/user/adminEdit	adminEdit	用户账户修改（管理员）	JSON
/user/sendSerialKey/	sendSerialKey	用户激活请求	JSON
/user/profile/	profile	用户信息	JSON
/user/activate/	activate	用户激活页面	HTML
/user/resetPassword	resetPassword	用户密码重置	JSON

5.3 浏览器端详细设计

一个好的Web应用不仅仅要拥有功能完善的后台，还应该拥有一个友好的界面。这部分我们介绍下客户端的基本框架以及使用的编程方法。

本系统用到的样式表和

5.3.1 客户端结构

整个网站可以分为三个部分：侧边栏、顶部导航和页面主体部分。为了统一所有页面的样式，我们使用了SiteMesh框架^①，它通过对用户请求进行过滤，并对服务器向客户端响应也进行过滤，然后给原始页面加入一定的装饰（header,footer等），然后把结果返回给客户端。通过SiteMesh的页面装饰，可以提供更好的代码复用，所有的页面装饰效果耦合在目标页面中，无需再使用include指令来包含装饰效果，目标页与装饰页完全分离，如果所有页面使用相同的装饰器，可以是整个Web应用具有统一的风格。

首先，我们定义了一个全局的装饰器，所有的页面都使用这个装饰器装饰。

```
001 <%--
002   Default decorator
003 --%>
004 <%@ taglib uri="http://www.opensymphony.2
005   com/sitemesh/decorator"
006           prefix="decorator" %>
007 <%@ taglib uri="http://www.opensymphony.com/sitemesh/page" 2
008   prefix="page" %>
009 <%@ page contentType="text/html; charset=UTF-8" 2
010   language="java" %>
011 <!DOCTYPE html>
012 <html lang="en" ng-app="cdoj">
```

① SiteMesh是由一个基于Web页面布局、装饰以及与现存Web应用整合的框架。它能帮助我们在由大量页面构成的项目中创建一致的页面布局和外观，如一致的导航条，一致的banner，一致的版权，等等。它不仅仅能处理动态的内容，如jsp, php, asp等产生的内容，它也能处理静态的内容，如html的内容，使得它的内容也符合你的页面结构的要求。甚至于它能把HTML文件象include那样将该文件作为一个面板的形式嵌入到别的文件中去。所有的这些，都是GOF的Decorator模式的最生动的实现。

```

013 <head>
014   <page:applyDecorator name="head" page="/WEB-INF/views/common/header.jsp"/>
015   INF/views/common/header.jsp"/>
016   <decorator:head/>
017
018   <title><decorator:title/> - UESTC Online Judge</title>
019 </head>
020
021 <body>
022 <page:applyDecorator name="body"
023                       page="/WEB-INF/views/common/navbarTop.jsp"/>
024
025 <div id="cdoj-layout">
026   <div id="cdoj-navbar">
027     <page:applyDecorator name="body"
028                           page="/WEB-INF/views/common/navbarList.jsp"/>
029
030   </div>
031   <div id="cdoj-container">
032     <decorator:body/>
033   </div>
034 </div>
035
036 <page:applyDecorator name="body"
037                       page="/WEB-INF/views/common/modal.jsp"/>
038
039 <page:applyDecorator name="head"
040                       page="/WEB-INF/views/common/footer.jsp"/>
041
042 </body>
043 </html>

```

header.jsp文件包含了网站head标签内的所有信息，head元素是所有头部元素的容器，指引浏览器找到样式表，提供元信息。

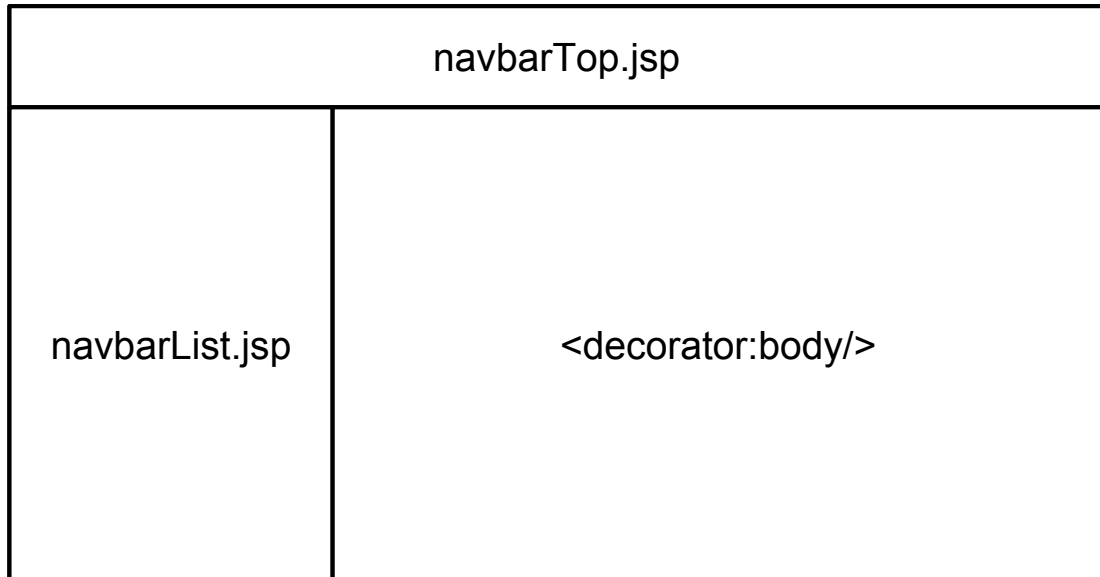


图 5-19 SiteMesh装饰器结构图

body标签首先包含的是**navbarTop.jsp**，这个文件是网站的顶部导航部分。接着是一个双栏格式的div，左侧包含的**navbarList.jsp**是网站的侧边栏部分，右侧的**<decorator:body/>**是网站的主体部分。最后两个部分**modal.jsp**和**footer.jsp**分别保存网站中的对话框元素和网站的脚本文件。如图5-19所示。

当控制器返回一个视图地址时（如index/index），SiteMesh会自动用**index/index.jsp**文件替换掉**<decorator:body/>**标签，得到一个完整的文件。

在样式方面，我们采用了Bootstrap3作为网站前端的UI框架，原因有以下两点：首先，Bootstrap出自twitter，并且开源，久经考验，减少了测试的工作量。同时，Bootstrap的代码有着非常好的代码规范，从中也可以学习到很多，在Bootstrap的基础之上创建项目，日后代码的维护也变得异常简单清晰。

Bootstrap是一个响应式的框架，利用这个特性，我们为移动客户端进行了一些优化，让客户端能够在手机等小屏幕设备上正常浏览。图5-20是最终的整体效果图，我们通过调整浏览器的大小来模拟不同的屏幕尺寸，在实际中效果也很好。

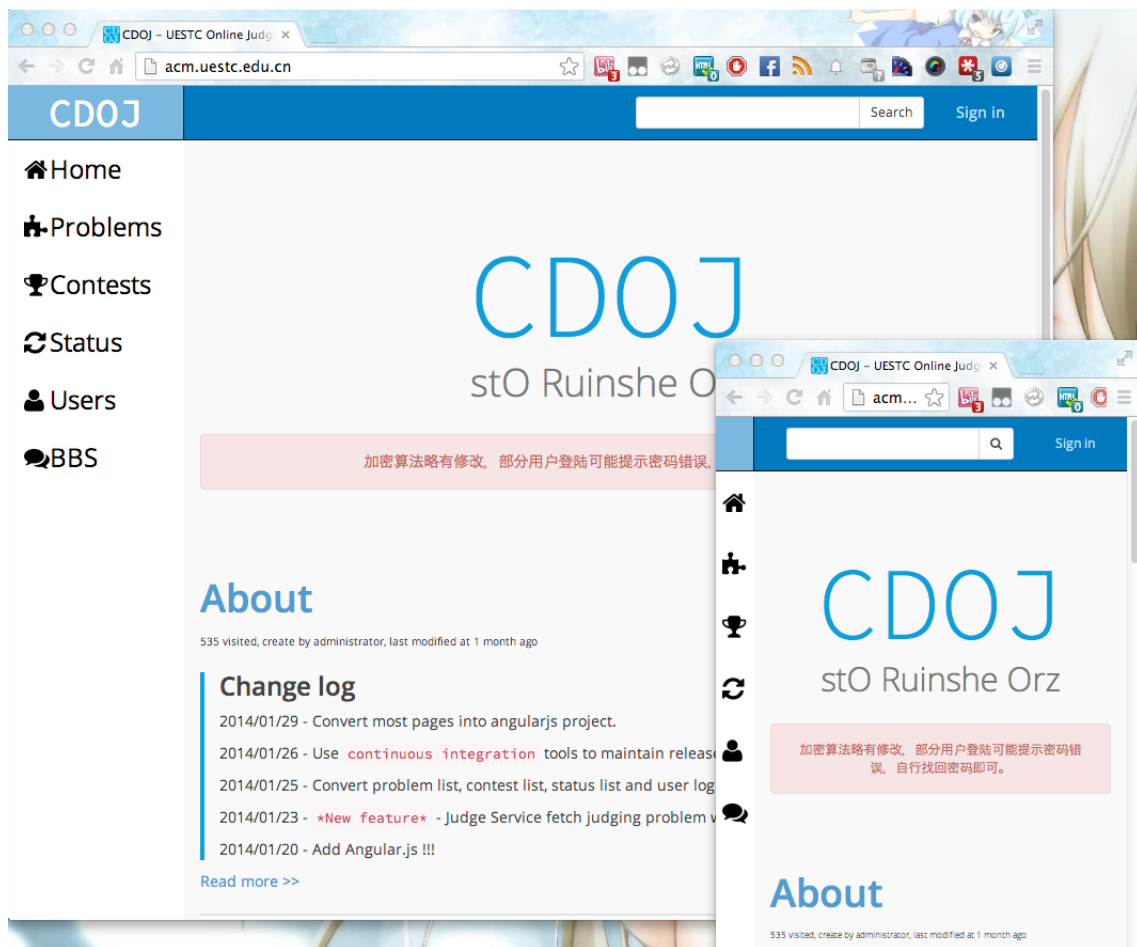


图 5-20 网站前端设计

5.3.2 客户端加载流程

客户端加载完成后，首先运行下面这段代码：

```
001 cdoj = angular.module("cdoj", ["ngSanitize", "monospaced.2  
002 elastic", "ui.bootstrap"])
```

这句话定义了一个叫做cdoj的angular module，后面的数组是本module引用到的外部库。接下来向/globalData这个地址获取用户的基本信息和一些常量。在AngularJS中每个元素都有一个自己的Scope（作用域），整个系统中还有一个rootScope，所有元素都能访问这个Scope，我们将从/globalData获取的全局信息放到rootScope中便于日后使用。

```
001 cdoj.run([  
002   "$rootScope", "$http"  
003   ($rootScope, $http)->  
004     $http.get("/globalData").then (response)->  
005       data = response.data  
006       if data.result == "error"  
007         alert(data.error_msg)  
008       else  
009         ..extend($rootScope, data)  
010 ])
```

完成上述步骤后，AngularJS分析整个网页的文档树，然后将各个Controller和Directive应用到各部分中。

5.3.3 顶部导航详细设计

5.3.3.1 用户菜单设计

顶部导航栏有两部分组成：搜索框和用户菜单。搜索框将在后面介绍，这里先介绍用户菜单。

在用户未登录时用户菜单显示的是一个Sign in按钮，而在登录之后显示的是用户的头像^①。登陆前菜单由一个登录表单组成，登录之后菜单内容变成了与用户相关的内容。见图5-21。

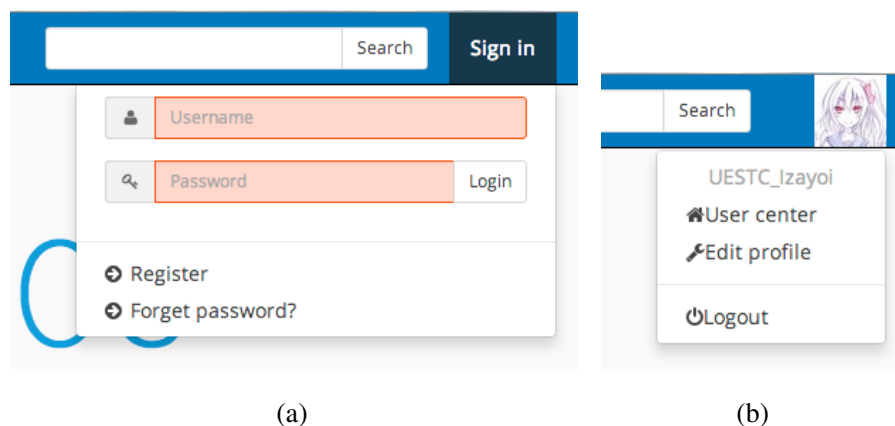


图 5-21 用户菜单

(a)登陆前的用户菜单；(b)登录后的用户菜单

在2.3.3中我们介绍了AngularJS框架，网页的html标签中我们用`ng-app="cdoj"`定义了app的作用域。在HTML中用户菜单只有如下一个标签（详细代码见附录A.7）：

```
001 <!-- User -->
002 <li ng-controller="UserController">
003 </li>
```

`ng-controller="UserController"`声明了这个li标签对应的控制器，我们将一个叫做UserController的控制器绑定到了这个标签上，在这个控制器中，我们实现了如下功能：

1. 用户菜单各个按钮事件的绑定。
2. 根据rootScope中的hasLogin变量来决定用户的登录状况以及根据情况展示相应的内容。

我们在整个用户操作过程中都使用AJAX，这样做的好处是用户可以在原网页上登陆，而不用离开当前页面。

^① 我们使用了Gravatar（英语：Globally Recognized Avatar）头像服务。只要用户在Gravatar的服务器上上传了自己的头像，用户便可以在其他任何支持Gravatar的博客、论坛等地方使用它。

5.3.3.2 用户注册流程

打开Sign in菜单（图5-21(a)），单击Register后弹出注册对话框（图5-22）。对话框中包含一个注册表单，表单的每个输入框都是一个Angular Model，通过在input标签中添加相应的属性将其绑定到指定的变量上去，除此之外还可以进行浏览器端的表单验证。

图 5-22 注册对话框

```

001 <div class="form-group">
002   <label class="control-label col-sm-4 "
003     for="userName">User Name</label>
004   <div class="col-sm-8">
005     <input type="text"
006       ng-model="userRegisterDTO.userName"
007       maxlength="24"

```

```

008         id="userName"
009         ng-required="true"
010         ng-pattern="/^[a-zA-Z0-9_]{4,24}$/"
011         class="form-control input-sm"/>
012     <ui-validate-info value="fieldInfo" for="userName"></ui-?
013     validate-info>
014 </div>
015 </div>

```

ng-model指定了绑定的变量名`scope.userRegisterDTO.userName`，这种绑定是双向的，也就是说当变量改变时，输入框内的数据也会发生变化，同理当输入框变化是变量会自动变化。**ng-required**和**ng-pattern**指定了表单验证的条件。当数据不满足条件时，AngularJS会自动给该输入框添加一个**ng-invalid**的class。

当用户填写完所有项目后，便可以单击Register按钮完成注册。如果表单中存在错误，我们会给出相应的提示（图5-23），否则对话框消失，注册完成，此时新注册的用户自动登陆在系统之中。为了保证用户密码的安全，我们在浏览器端使用Crypto-JS库^①对用户的密码进行加密，然后将加密后的密码和账号传给后台。

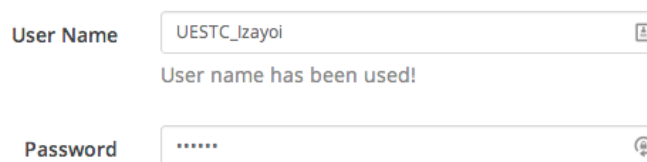


图 5-23 表单错误提示示例

5.3.3.3 用户登陆流程

在Sign in菜单（图5-21(a)）中填写用户名与密码之后单击Login按钮即可登陆。

5.3.3.4 用户登出流程

登陆之后，在用户菜单中（图5-21(b)单击Logout即可安全退出。

① CryptoJS是一个纯javascript写的加密类库。

5.3.3.5 用户密码找回流程

我们给忘记密码的用户提供了一个找回密码的功能，单击Sign in菜单（图5-21(a)）中的Forget password按钮可以弹出一个找回密码的对话框，用户在里面将自己的用户名填入后单击Send email按钮（图5-24），服务器会向用户注册时填写的邮箱中发送一封找回密码用的邮件，并弹出成功提示。

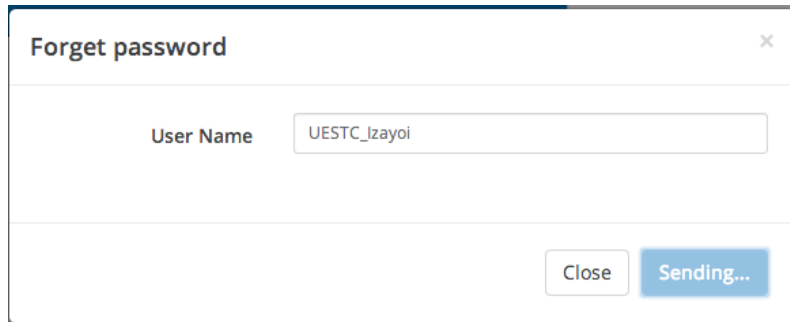


图 5-24 密码找回对话框（单击Send email后）

邮件中包含了一个形如http://acm.uestc.edu.cn/user/activate/UESTC_Izayoi/96ac3b16d84a4bb335bfd8321c7a32f61f70f99f的地址，这个地址由用户名和一个SerialKey组成。为了保证安全，每个SerialKey只有30分钟的有效期，且只能使用一次，这个地址会将用户带到一个密码重置页面（图5-25）。

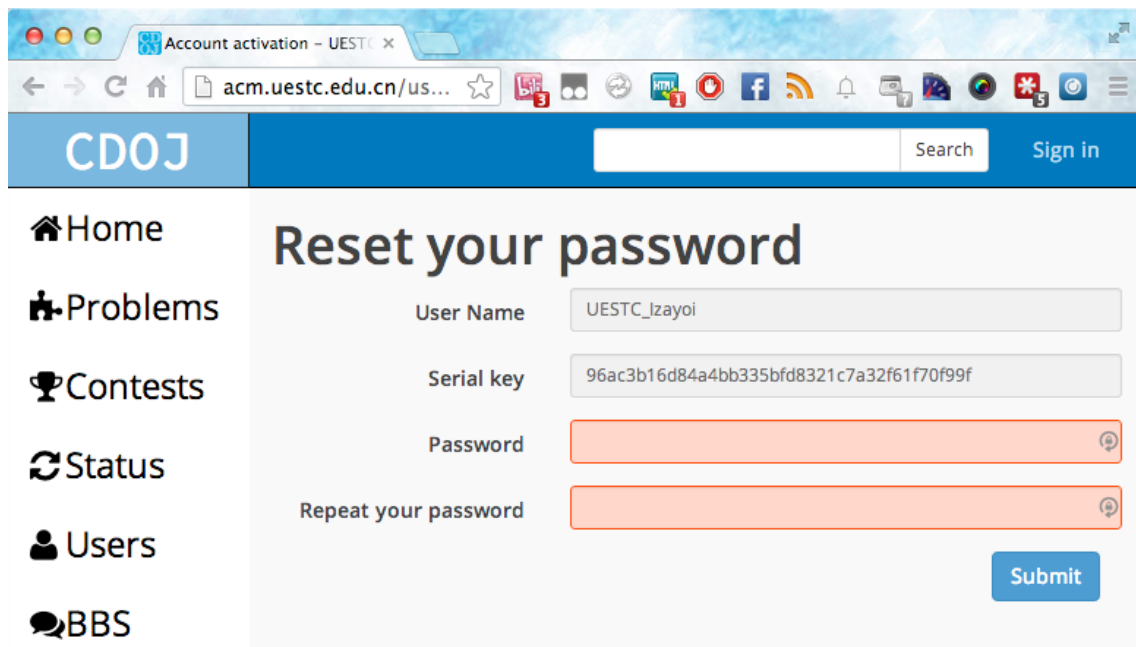


图 5-25 找回密码页面

5.3.3.6 用户账户修改

登陆之后，在用户菜单中（图5-21(b)单击Edit profile后会弹出一个用户信息编辑对话框，在这个对话框中用户能修改一部分基本信息，如密码、学号、学院信息等，但是不允许修改用户名。

5.3.4 内容渲染



图 5-26 内容编辑器

(a)编辑模式；(b)预览模式

为了保证网站内容格式的统一，本系统采用Markdown语言作为页面内容的排版语言^{①②}，同时我们还支持在内容中插入 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 格式的数学公式^③，代码

① Markdown 是一种轻量级标记语言，创始人约翰·格鲁伯（John Gruber）和亚伦·斯沃茨（Aaron Swartz）。它允许人们“使用易读易写的纯文本格式编写文档，然后转换成有效的XHTML（或者HTML）文档”。这种语言吸收了很多在电子邮件中已有的纯文本标记的特性。

② 本系统中使用marked.js来将Markdown内容转换成HTML。

③ 本系统使用MathJax插件来渲染数学公式。

由Prettify插件渲染。在这三个插件的帮助下，我们可以用很简单的方式排版出漂亮的格式。为了满足这些需求，我们实现了一个简单的Markdown编辑器，代码见附录A.8。它具有编辑模式和预览模式两种状态，通过一个Preview开关来切换，工具栏目前只有两个工具，一个是表情插入，这个部分是给用户讨论功能准备的，另一个工具是上传图片，用于向文章中插入一些描述性的图，如图5-26所示。

5.3.5 列表页面

本系统一共有有许多信息通过列表的形式呈现给用户，所以我们设计了一个简单的列表控制器**ListController**来复用一些重复的代码（见附录A.9）。每个列表的HTML代码都是如下形式：

```

001 <div id="xxx-list"
002     ng-controller="ListController"
003     ng-init="condition={
004         currentPage: null,
005         keyword: undefined,
006         orderFields: undefined,
007         orderAsc: undefined
008     }";
009     requestUrl="/xxxx/search">
010 <div class="row">
011     <div class="col-md-12">
012         <div ui-page-info
013             page-info="pageInfo"
014             condition="condition"
015             id="page-info">
016         </div>
017         <div id="advance-search">
018             <a href="#" id="advanced" data-toggle="dropdown"><i
019                 class="fa fa-caret-square-o-down"></i></a>
020             <ul ui-dropdown-menu class="dropdown-menu cdoj-form-2
021                 menu"

```

```

022         role="menu"
023         aria-labelledby="advance-menu">
024     <li role="presentation" id="condition">
025         <form class="form">
026             列表查询条件表单
027             <p class="pull-right">
028                 <button type="button" class="btn btn-danger 2
029                 btn-sm" ng-click="reset()">Reset
030             </button>
031         </p>
032     </form>
033 </li>
034 </ul>
035 </div>
036 </div>
037 </div>
038
039 <div class="row">
040     <div ng-repeat="xxx in list">
041         列表主体
042     </div>
043 </div>
044 </div>

```

列表在**ng-init**属性中定义了初始化查询条件**condition**和查询操作的链接。列表查询表单中的各个输入框与**scope.condition**中的元素绑定，当用户修改查询条件时列表会自动更新。而后通过**ng-repeat**标签来定义列表的主体，AngularJS会自动将查询到的列表中的每一个元素应用到这个模板中去。

在列表页面，顶部导航栏中的搜索框将会开始起作用，在它的右侧会有一个小箭头，通过它可以打开高级搜索下拉菜单，既之前说的列表查询表单。如图5-27所示。

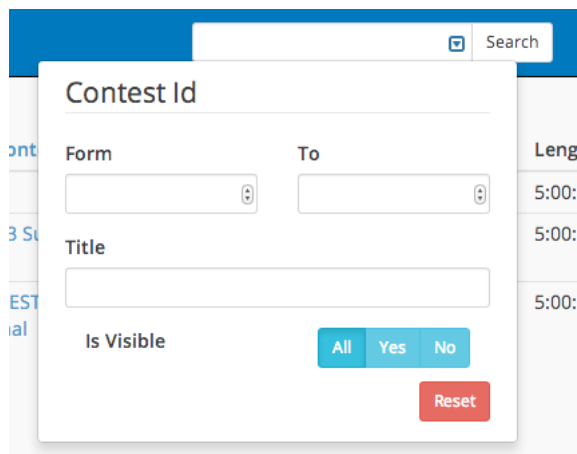


图 5-27 比赛列表中的高级搜索菜单

列表的每一页只显示20个项目，如果查询得到的项目超过这个数量，页面会显示一个分页栏对结果进行划分。

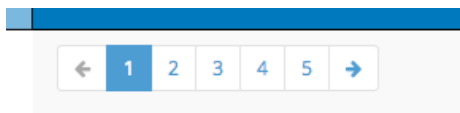


图 5-28 分页栏

5.3.6 主页设计

主页是一个网站的入口，它由三个部分组成：网站标志，重要公告，日志。我们将一些重要的公告放置在醒目的位置。其余部分就是近期的一些日志，例如系统简介、常见问答等，如图5-29所示。

5.3.7 文章模块详细设计

该模块包含两个页面，一个是文章页面，一个是文章编辑器。以管理员身份打开文章页面后在文章标题下方会有一个编辑连接，通过它可以编辑该文章，如图5-30所示。

5.3.8 用户模块详细设计

5.3.8.1 用户列表页面

单击侧边栏中的User选项即可进入用户列表页面，用户列表以名片墙的形式展示用户信息，按题目通过数量排序。用户列表的高级搜索菜单提供了按用户编

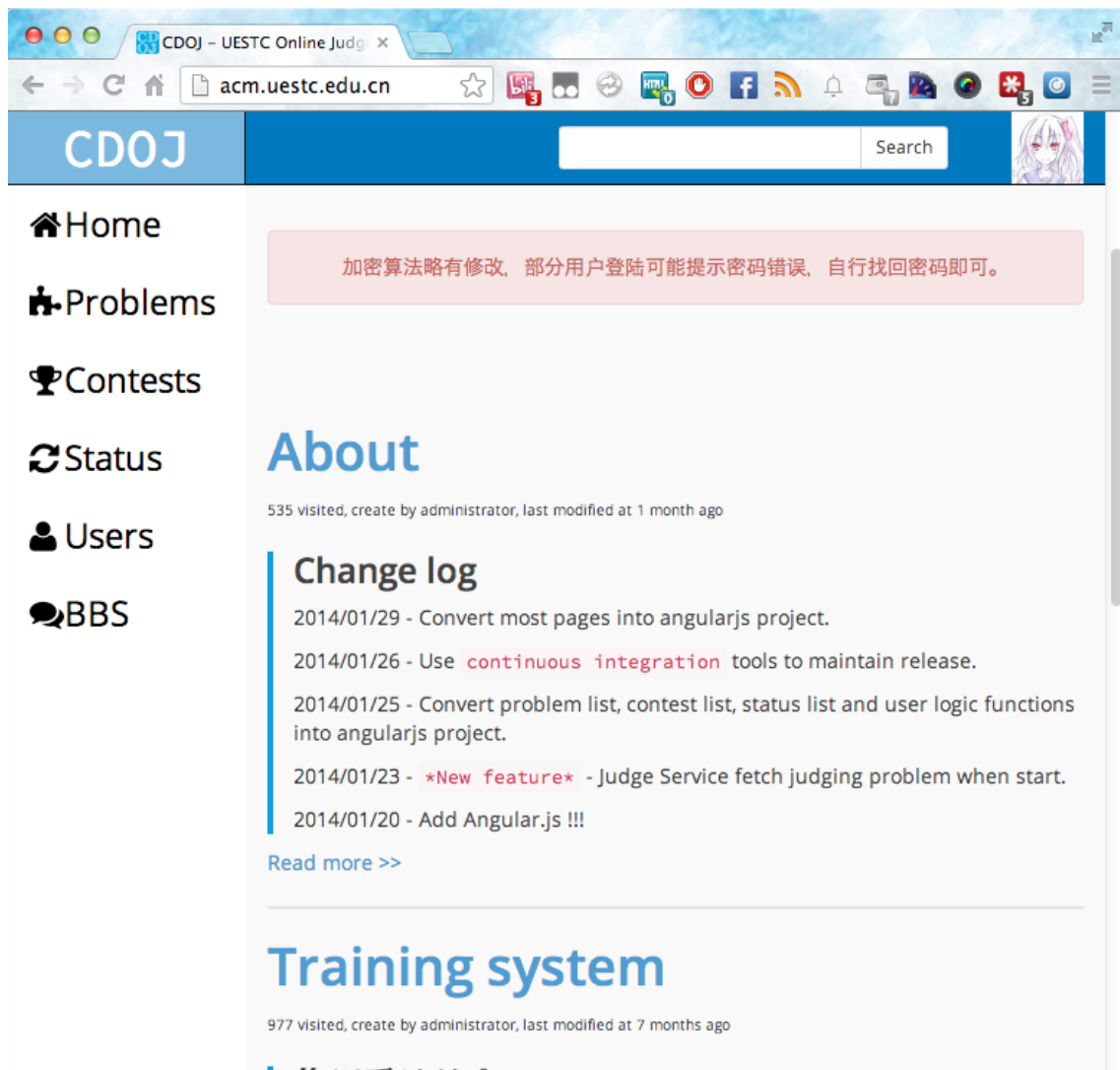


图 5-29 网站主页

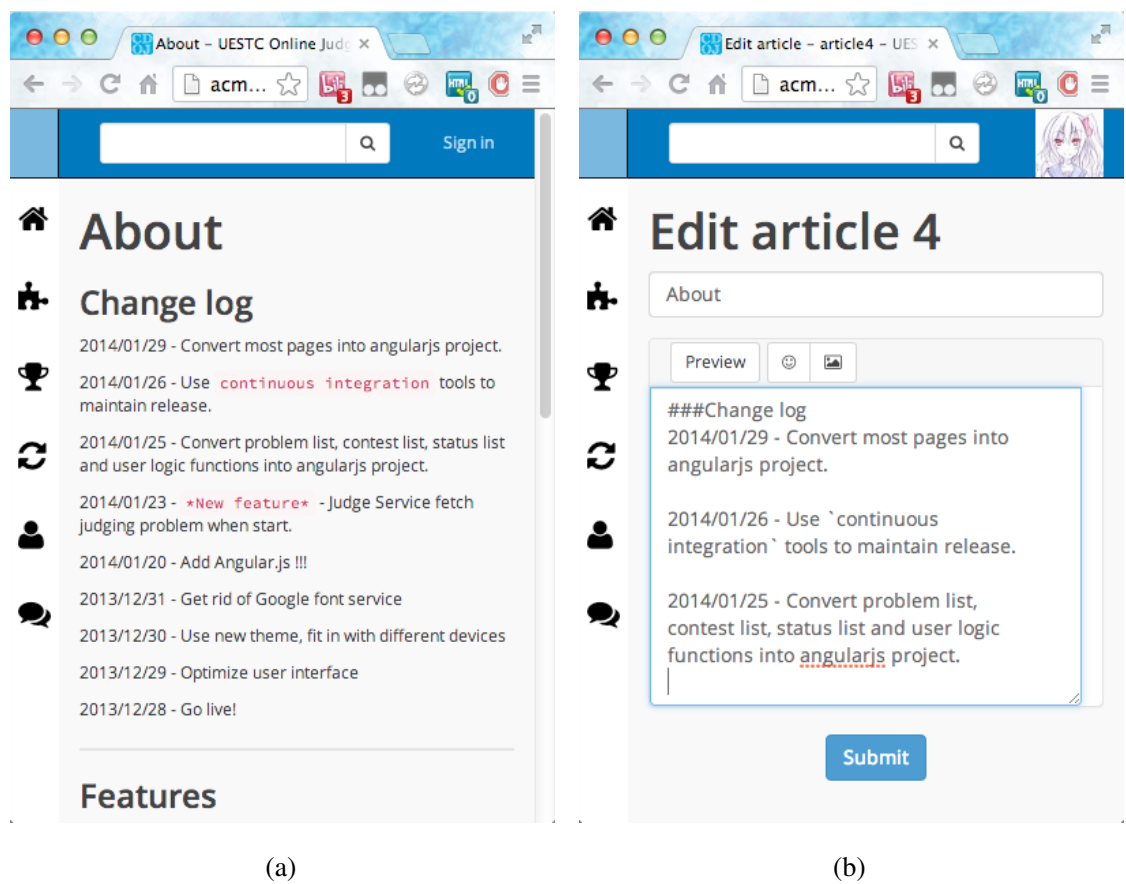


图 5-30 文章相关页面

(a)文章页面；(b)文章编辑器

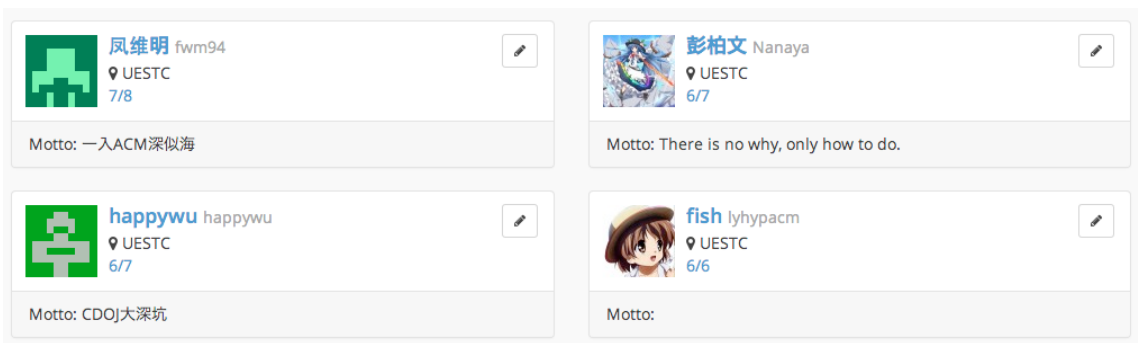



图 5-31 用户列表

号查询、按用户名查询、按用户学校查询等等高级搜索功能。名片左上角还有一个管理员的编辑按钮，方便管理员对用户进行操作，如图5-31所示。

5.3.8.2 用户中心页面

在网页中任何一个地方单击用户名都会进入该用户的用户中心，这个用户中心用来展示该用户的一些信息，如基本信息、题目通过情况等，如图5-32所示。



UESTC_Izayoi

Nick name 木子日勾
School UESTC
Department School of Communication & Information Engineering
Student ID 2010013100008
Email muziriyun@qq.com
Motto 最喜欢何老师了！
Last login 2014-03-03 14:09:04

Problems

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211													

图 5-32 用户中心

5.3.9 题目模块详细设计

5.3.9.1 题目列表页面

单击侧边栏中的Problem选项即可进入题目列表页面，题目列表包含了题目ID、题目名称、题目来源、通过人数，如果以管理员身份登陆还有有题目编辑按钮（一个是隐藏/显示题目，一个是编辑题目）和一个添加新题目的链接。题目列表的高级搜索菜单提供了按题目编号查询、按题目难度查询、按题目标题查询、按题目来源查询等查询功能。

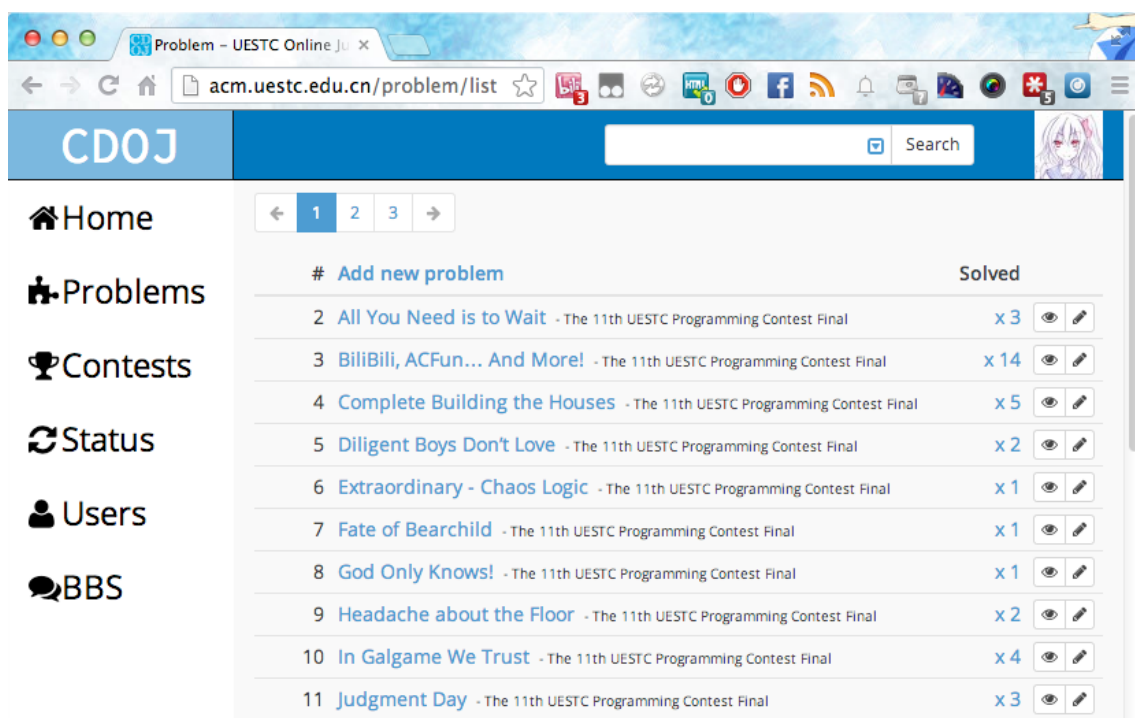


图 5-33 题目列表

5.3.9.2 题目页面

经由题目列表可以进入题目页面。一道题目由题目限制、题目描述、输入格式、输出格式、测试样例、提示（可选）和来源信息（可选）组成。如图5-34所示。

5.3.9.3 代码提交流程

在题目页面之中单击Submit标签按钮，切换到代码提交对话框，如图5-35所示。用户将自己的代码复制到代码框中，然后通过下方的语言选择按钮选择对应的语言，最后单击Submit按钮即可将自己的代码提交至服务器。提交成功后会自动跳转到评测列表页面。

5.3.9.4 题目编辑器

以管理员的身份从题目列表或题目页面的入口可以进入题目编辑器。这个编辑器页面上半部分为基本信息编辑框，下半部分是题面描述编辑框。题目基本信息编辑框见图5-36，题目编辑框和文章编辑器类似，这里就不赘述了。

管理员添加题目时，需要通过文件上传测试用例。我们规定用户使用一个zip压缩包打包，这个zip包种包含若干测试用例，每个测试用例的输入文件

3阶矩阵的乘法

[Edit](#)

Problem

[Submit](#)

[Status](#)

[Discuss](#)

Time limit 3000 / 1000 ms (Java / others)
 Memory limit 65535 / 65535 kb (Java / others)
 Total accepted 8
 Total submissions 10

实现两个3*3矩阵的乘法。

Input

第一行是一个正整数 n 表示测试数据的组数。接下来有 $2n$ 个三阶矩阵。整数的范围为 $[-1000, 1000]$ 。

Output

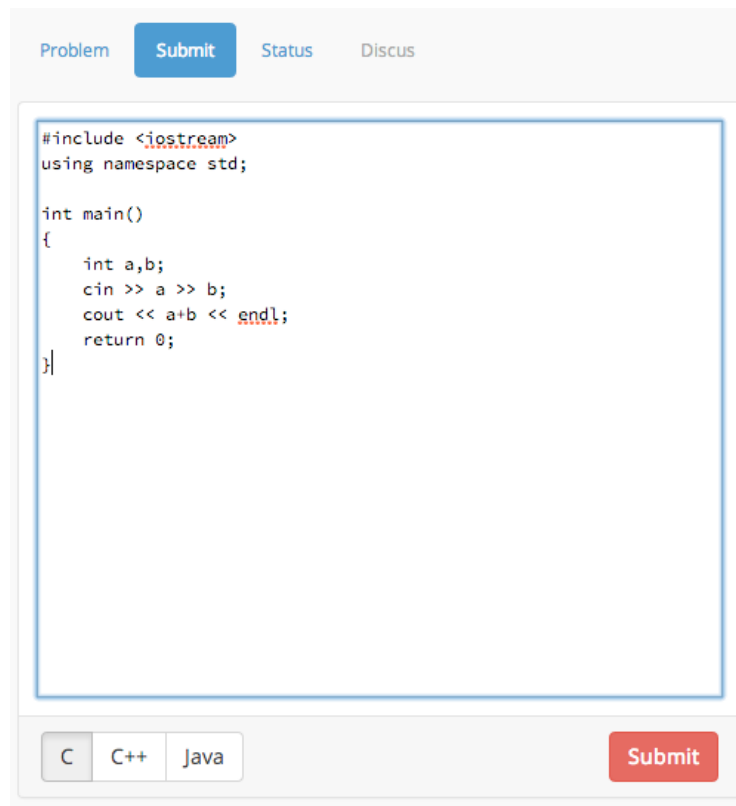
每组数据对应有一个矩阵输出，每个数后输出一个空格。

每组数据之后输出一个空行。

Sample input and output

Sample Input	Sample Output
2	3 3 3
1 1 1	3 3 3
1 1 1	3 3 3
1 1 1	
1 1 1	2 3 4
1 1 1	5 6 7
1 1 1	1 5 9
1 0 0	
0 1 0	
0 0 1	
2 3 4	
5 6 7	
1 5 9	

图 5-34 题目页面

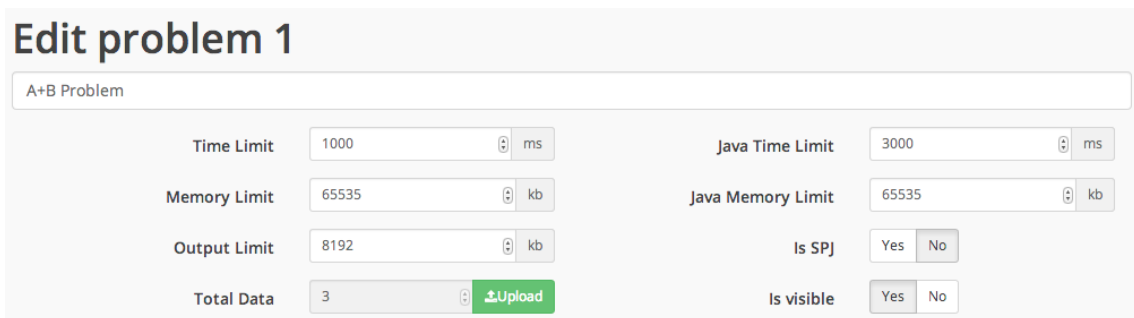


The image shows a code submission interface. At the top, there are four tabs: "Problem", "Submit" (highlighted in blue), "Status", and "Discuss". Below the tabs is a large text area containing C++ code for a simple addition problem. At the bottom, there are three language selection buttons: "C", "C++" (selected), and "Java". To the right of these buttons is a red "Submit" button.

```
#include <iostream>
using namespace std;

int main()
{
    int a,b;
    cin >> a >> b;
    cout << a+b << endl;
    return 0;
}
```

图 5-35 代码提交对话框



The image shows the "Edit problem 1" dialog box. The title is "Edit problem 1". Below the title is a text input field containing "A+B Problem". The dialog is divided into two columns of settings. The left column contains "Time Limit" (1000 ms), "Memory Limit" (65535 kb), "Output Limit" (8192 kb), and "Total Data" (3). The right column contains "Java Time Limit" (3000 ms), "Java Memory Limit" (65535 kb), "Is SPJ" (Yes/No), and "Is visible" (Yes/No). There is an "Upload" button next to the "Total Data" field.

Field	Value	Unit
Time Limit	1000	ms
Memory Limit	65535	kb
Output Limit	8192	kb
Total Data	3	
Java Time Limit	3000	ms
Java Memory Limit	65535	kb
Is SPJ	Yes	No
Is visible	Yes	No

图 5-36 题目基本信息编辑框

以.in为后缀，对应的输出文件以.out结尾。如果需要进行Special Judge，这个zip包种还应该包含spj.cc文件。上传成功后会有如图5-37所示提示。

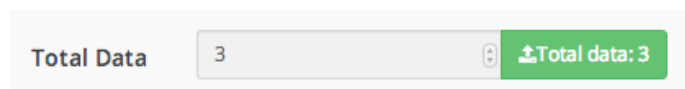


图 5-37 题目数据上传成功提示

编辑完后点击Submit按钮即可提交这次编辑。

5.3.10 评测状态模块详细设计

5.3.10.1 评测列表页面

单击侧边栏中的Status选项或者提交代码后都会进入评测列表页面，评测列表包含了任务ID、提交者用户名、题目编号、评测状态、内存开销、时间开销、编程语言、代码长度、提交时间（见图5-38）。评测列表的高级搜索菜单提供了按评测ID查询、按提交者查询、按提交语言查询、按返回结果查询等功能。

#	User	Prob	Result	Memory	Time	Language	Length	Submit Time
1815	Rebecca	3	Accepted	1060 KB	0 MS	C++	480 B	6 hours ago
1814	Rebecca	3	Wrong Answer on test 1			C++	475 B	6 hours ago
1813	Rebecca	3	Wrong Answer on test 1			C++	473 B	6 hours ago
1812	UESTC_lzayoi	1	Accepted	1176 KB	0 MS	C++	136 B	6 hours ago
1811	zhang201322	25	Compilation Error			C++	960 B	8 hours ago
1810	zhang201322	25	Compilation Error			C++	960 B	8 hours ago

图 5-38 评测列表

用户有权查看自己的代码和编译错误信息，在列表中单击Compile Error链接或代码连接就可以查看相应的信息，图5-39展示了编译错误信息窗口。

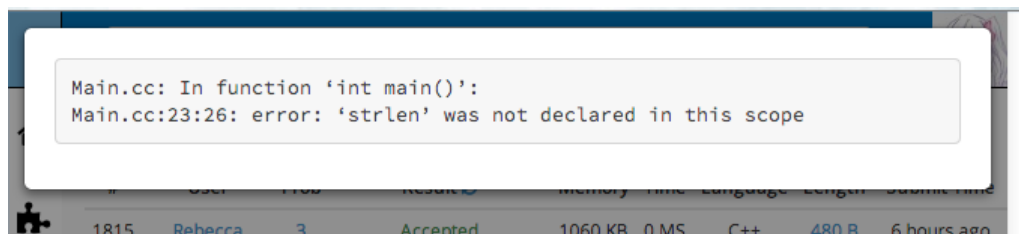


图 5-39 编译错误信息对话框

5.3.10.2 管理员Rejudge流程

测试数据不一直是完美的，有时候管理员可能要对这些数据进行修改和调整，调整之后对之前已经提交的代码来说可能测试结果会发生变化，有些时候这些变化时很重要的，我们需要对这些测试进行重新测试（Rejudge）操作。整个操作流程如下：

1. 管理员通过评测列表的高级搜索功能确定需要Rejudge的评测状态。
2. 管理员按下Rejudge按钮，然后会有一个弹出窗口提示此次操作将要影响到的评测状态数（见图5-40）。
3. 确认无误后管理员在弹出的窗口中按下确认按钮。
4. Rejudge请求被发送，受到影响的评测状态都被更新为OJ_Rejudge。

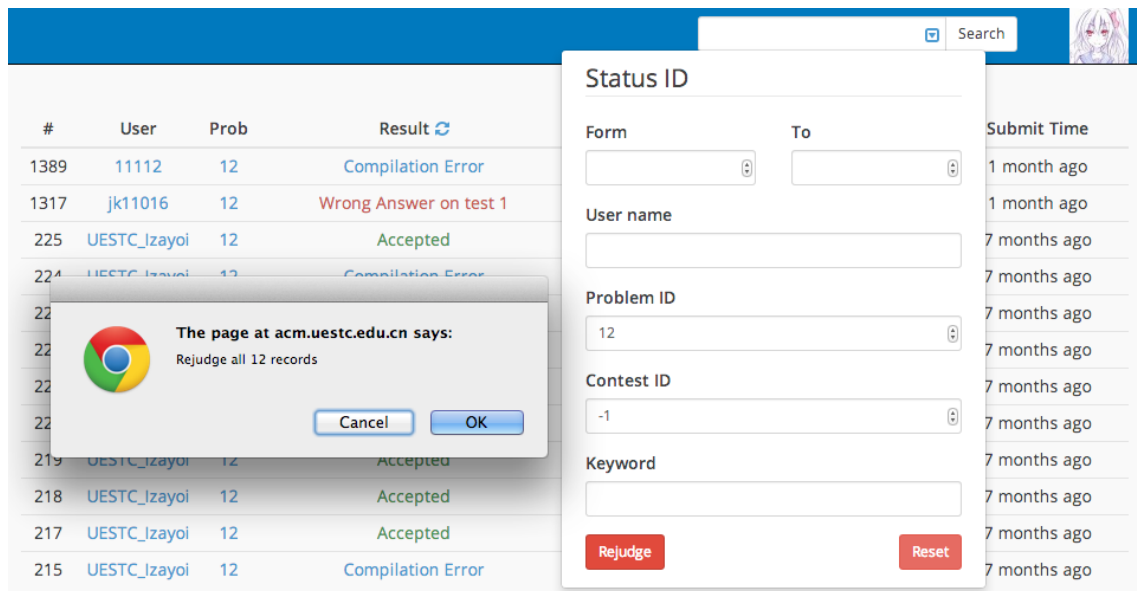


图 5-40 Rejudge操作示例

5.3.11 比赛模块详细设计

5.3.11.1 比赛列表页面

#	Add new contest	Start time	Length
2	UESTC 2013 Summer Training #18 Div.2	7 months ago	5:00:00
1	The 11th UESTC Programming Contest Final	8 months ago	5:00:00

图 5-41 比赛列表

单击侧边栏中的Contest选项即可进入比赛列表页面，比赛列表包含了比赛ID、比赛名称、比赛开始时间、比赛长度，如果以管理员身份登陆还有有比赛编辑按钮（一个是隐藏/显示比赛，一个是编辑比赛）和一个添加新比赛的链接。比赛列表的高级搜索菜单提供了按比赛编号查询、按比赛标题查询的查询功能（见图5-27）。

5.3.11.2 比赛页面

比赛页面集合了题目列表（图5-42）、题目页面（图5-44）、状态页面（图5-45）、在线答疑和比赛排名（图5-43）五个页面，由一个标签页组展示给用户。

UESTC 2013 Summer Training #18 Div.2			
Overview	Problems	Clarification	Status Rank
<p>Current Time: 2014-03-03 14:08:48</p> <p>Start Time: 2013-08-01 12:00:00</p> <p>End Time: 2013-08-01 17:00:00</p> <p>Contest Type: public</p> <p>Contest Status: Ended</p>			
Id		Title	
2 / 51	Problem A	Boggle	
14 / 159	Problem B	Booking	
9 / 110	Problem C	Chess	
35 / 124	Problem D	Kastenlauf	
0 / 10	Problem E	No Trees But Flowers	
13 / 34	Problem F	Peg Solitaire	
0 / 5	Problem G	Ringworld	
0 / 0	Problem H	The King of the North	
0 / 0	Problem I	Ticket Draw	
0 / 3	Problem J	Timing	
0 / 0	Problem K	Triangles	

图 5-42 题目列表

5.3.11.3 比赛编辑器

我们提供了一个方便的题目编辑器供管理员编辑题目，它的界面如图5-46所示，

Overview		Problems	Clarification	Status	Rank									
Rank	ID	Solved	Penalty	A	B	C	D	E	F	G	H	I	J	K
1	gtas5 (王煜)	4	665		3:27:31 (-6)	2:44:28 (-2)	0:20:36		1:52:30					
2	fwm94 (凤维明)	4	12:02:30		3:39:43	2:42:34 (-1)	0:51:24		4:28:49					
3	wilyin (赵轅)	4	757	4:01:47 (-3)	(-3)	2:26:12 (-3)	1:12:21		2:56:35					
4	micsun (孙云鹏)	4	815		1:48:02 (-4)	4:57:37 (-3)	1:31:50		2:57:52					
5	crow (蒋雨珂)	4	826	2:40:00 (-6)	(-2)	4:05:23	1:38:21		3:22:42					
6	Nanaya (彭柏文)	4	1082		3:29:23 (-4)	4:43:46 (-10)	0:40:42		4:07:50 (-1)				(-3)	
7	purewater (吴昊泰)	4	1302	(-3)	4:52:24 (-3)	4:02:34 (-5)	1:47:26 (-1)		4:19:13 (-11)					
8	hjsx1212 (胡剑箫)	4	1574	(-1)	4:27:09 (-33)	3:46:01 (-11)	0:46:50 (-1)		2:13:34					
9	geniuszhaoyi (赵毅)	3	602	(-1)	2:31:11	(-9)	1:33:53 (-1)		4:36:54 (-3)					

图 5-43 比赛排名

Overview

Problems

Clarification

Status

Rank

A

B

C

D

E

F

G

H

I

J

K

A - Boggle

Time Limit: 30000 / 10000 MS (Java/Others) Memory Limit: 262140 / 131070 KB (Java/Others)

Submit

Status

Clarification

I am sure, you are a big fan of the board game **Boggle**. Don't worry if you are not familiar with the rules, I will explain them to you. A Boggle is a 4 × 4 grid of letters where your job is to find as many words as you can. If I play Boggle with (or against) my wife, she always wins, the loser(that's me) then always has to do all these little jobs like to take out the trash. So, please help me to win again.

Words in a Boggle can be constructed from adjacent letters (i.e. horizontally, vertically and diagonally), but the same dice may only be used once in a word. The words have to be listed in our dictionary to be valid.

图 5-44 题目页面


Prob	Result 	Memory	Time	Language	Length	Submit Time
All ▾	All ▾			All ▾		
E	Accepted	1224 KB	596 MS	C++	1214 B	6 months ago
E	Wrong Answer on test 2			C++	1220 B	6 months ago
C	Wrong Answer on test 1			C++	1215 B	7 months ago
C	Wrong Answer on test 1			C++	1257 B	7 months ago
B	Wrong Answer on test 3			C++	1487 B	7 months ago
F	Accepted	1204 KB	4 MS	C++	2309 B	7 months ago
F	Wrong Answer on test 1			C++	2264 B	7 months ago
F	Wrong Answer on test 1			C++	2139 B	7 months ago

图 5-45 状态页面


Edit contest 3

The 12th UESTC Programming Contest Warmup #1

Type

public
 private
 DIY
 invited

Begin time

2014-03-22 12:00 



Length

0 ▾ Days
 5 ▾ Hours
 0 ▾ Minutes

Problem list

+	Id	Title
✕	1	A+B Problem
✕	2	All You Need is to Wait
✕	3	Bilibili, ACFun... And More!

Preview

The 12th UESTC Programming Contest Warmup #1!

Submit

图 5-46 比赛编辑器

第6章 系统测试

软件测试是软件项目中非常重要的一部分，每次代码发生修改，我们都需要检测这次修改是否对原有的功能造成了影响。在本项目中我们主要使用了黑盒测试的方法来对各个模块进行测试。在黑盒测试过程中，我们只需关心三样东西：设置测试数据，设定预期结果，验证结果。

6.1 测试方法

6.1.1 单元测试

在计算机编程中，单元测试（又称为模块测试, **Unit Testing**）是针对程序模块（软件设计的最小单位）来进行正确性检验的测试工作。程序单元是应用的最小可测试部件。在过程化编程中，一个单元就是单个程序、函数、过程等；对于面向对象编程，最小单元就是方法，包括基类（超类）、抽象类、或者派生类（子类）中的方法。通常来说，我们每修改一次程序就会进行最少一次单元测试，在编写程序的过程中前后很可能要进行多次单元测试，以证实程序达到要求。

每个理想的测试案例独立于其它案例。为了测试时隔离模块，我们经常经常使用Stubbing、Mock或Fake等测试马甲程序。单元测试通常由软件开发人员编写，用于确保他们所写的代码符合软件需求和遵循开发目标。它的实施方式可以是手动的（通过纸笔），或者是做成构建自动化的一部分。

6.1.2 Mock

软件中是充满依赖关系的，例如我们会基于Service类写操作类，而Service类又是基于数据访问类（DAO）的，依次下去。单元测试的思路就是我們想在不涉及依赖关系的情况下测试代码。种测试可以在无视代码的依赖关系的情况下去测试代码的有效性。核心思想就是如果代码按设计正常工作，并且依赖关系也正常，那么他们应该会同时工作正常。

在软件开发的世界之外，“mock”一词是指模仿或者效仿。因此可以将“mock”理解为一个替身，替代者。而在软件开发中提及“mock”，通常理解为模拟对象或

者Fake^①。模拟对象的概念就是我们想要创建一个可以替代实际对象的对象。这个模拟对象要可以通过特定参数调用特定的方法，并且能返回预期结果。

6.1.3 Stubbing

Stubbing（桩）是指用来替换一部分功能的程序段。桩程序可以用来模拟已有程序的行为（比如一个远端机器的过程）或是对将要开发的代码的一种临时替代。因此，打桩技术在程序移植、分布式计算、通用软件开发和测试中用处很大。桩程序是一段并不执行任何实际功能的程序，只对接受的参数进行声明并返回一个合法值。这个返回值通常只是一个对于调用者来讲可接受的值即可。桩通常用在对一个已有接口的临时替换上，实际的接口程序在未来再对桩程序进行替换。

6.1.4 Mockito

我们采用了Mockito框架来进行Mock和Stubbing操作，代码保存在src/test目录下，文件夹有图6-1所示的目录结构。

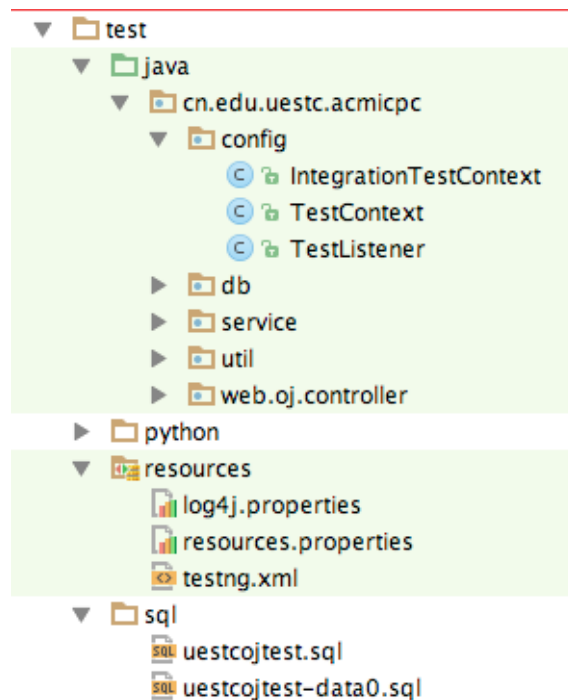


图 6-1 测试目录结构

^① mock等多代表的是对被模拟对象的抽象类，可以把fake理解为mock的实例。

Spring使用@**Bean**注解和@**Autowired**来完成依赖注入操作，为了得到这些依赖的模拟对象，我们使用了Mockito框架中的mock函数。下面这段代码配置了一个UserService的模拟对象：

```
001 @Bean
002 @Primary
003 public UserService mockUserService() {
004     return mock(UserService.class);
005 }
```

通过这种方式得到的模拟对象可以很轻松的实现桩程序，通过Mockito框架中的when函数，我们建立一个简单的桩程序。下面这段代码配置了一个桩程序：

```
001 when(problemService.checkProblemExists(1000)).thenReturn(
002     true);
```

这个桩程序替换了当参数为1000时的problemService的checkProblemExists方法，返回true值。

6.1.5 Spring MockMvc

Spring框架提供了MockMvc对象来模拟浏览器的真实操作。MockMvc使用**standaloneSetup**方法来由控制器生成一个模拟的浏览器对象。在成功得到该对象后，就可以使用它的**perform**方法来进行访问测试了。

```
001 // 建立一个浏览器模拟对象
002 MockMvc mockMvc = standaloneSetup(indexController)
003     .setViewResolvers(WebMvcResource.viewResolver())
004     .setMessageConverters(WebMvcResource.
005         messageConverters())
006     .build();
007
008 // 模拟Get操作，访问/地址
009 // 期望结果为：
010 //     返回状态为OK
```

```

011 //      视图名字是index/index
012 //      且模型中有期望的消息
013 mockMvc.perform(get("/"))
014           .andExpect(status().isOk())
015           .andExpect(view().name("index/index"))
016           .andExpect(model().attribute("message", "home page.2
017           "));

```

6.2 测试内容

6.2.1 数据库模块测试

这里对Condition模块进行了测试。我们通过Condition类来生成HQL查询语句中的条件语句，它在服务器运行过程中是最常使用到的模块之一，所以对它的测试要求比较高，这部分的测试一共有12组，见表6-1。

表 6-1 Condition模块测试用例表

测试描述	期望结果
条件为空	空字符串
简单的样例	where (id='1')
与条件测试	where (id>='1' and id<='5')
或条件测试	where (id>='1' or id<='5')
嵌套测试	where ((id>='1' or id<='5') and (price>'10' or price<='20'))
非空条件测试	where ((userId is not null))
空条件测试	where ((userId is null))
嵌套测试	where ((userId is not null) and (departmentId is null))
排序测试	order by userId desc
条件非空排序测试	where (userId>='1') order by userId desc
多关键字排序测试	where (userId>='1') order by departmentId asc,userId desc
组合测试	where ((userId>='1' or userId<='5') and ((departmentId is not null) or userName like '%user%')) order by departmentId asc,userId desc

6.2.2 数据库集成测试

数据库集成测试的作用是检测数据库各个模块之间是否工作正常。我们构建了一个名叫uestcojtest的测试数据库，这个数据库里面的每个表中都包含了若干数据，用以进行数据库集成测试。这里面包含了许多子测试，见表6-2。

表 6-2 数据库集成测试用例表

测试函数	测试内容	期望结果
基本测试		
testFetchDataSource	当前使用的数据库	uestcojtest
testDataBaseConnection	数据库连接情况	正常
testHQLQuery	HQL查询	正常
Condition测试		
testCondition_emptyEntrySet	查询功能	正常
testCondition_emptyEntrySetWithDescId	查询功能（按ID逆序排列）	正常
testCondition_count_emptyCondition	个数查询功能	正常
testCondition_count_withDepartmentId	个数查询功能（带条件）	正常
查找功能测试（返回实体）		
testDAO_getEntityByUnique	依照主键查找	正常
testDAO_getEntityByUnique_notUniqueField	依照非主键查找	抛出异常
查找功能测试（返回DTO）		
testDAO_findAllByBuilder	查找操作	正常
testDAO_findAllByBuilder_withPageInfo	查找操作（带目录限制）	正常
testDAO_getDTOByUniqueField_null	查找失败	返回null
testDAO_getDTOByUniqueField_intType	查找操作（条件为数字）	正常
testDAO_getDTOByUniqueField_stringType	查找操作（条件为字符串）	正常
testDAO_getDTOByUniqueField_failed	依照非主键查找	抛出异常
更新功能测试		
testSQLUpdate	多行更新	正常
DTO功能测试		
testUserDTO	Builder功能	正常
ContestProblemDAO测试		

接下页

		接上页
测试函数	测试内容	期望结果
testAddContestProblem	添加比赛题目功能	正常
Problem数据库测试		
testStartIdAndEndId	按ID区域查找	正常
testStartIdAndEndId_invalidParameter	按ID区域查找（参数错误）	空列表
testIsSpjQuery_notSpj	按SPJ状态查找（非SPJ）	正常
testIsSpjQuery_spj	按SPJ状态查找（是SPJ）	正常
testProblemCondition_emptyTitle	按标题查找（空标题）	正常
testAddProblem	添加题目	正常
Status数据库测试		
testStatusDAO_withDistinctProblem	查找符合条件的状态的题目ID	正常
Tag数据库测试		
testQuery_fetchAllTags	查找所有的tag	正常
testCount	查找所有的tag的个数	正常
User数据库测试		
testQuery_byName	按姓名查找用户	正常
testQuery_byDepartmentId	按学院查找用户	正常
testUserCondition_byStartIdAndEndId	按ID查找用户	正常
UserSerialKey数据库测试		
testFindUserSerialKeyByUserName	按用户姓名查找激活码	正常

6.2.3 服务集成测试

前面我们已经对数据库进行了集成测试，在这些测试都通过后，我们可以假设数据库模块已经是正常的了，然后在服务中用DAO的模拟对象对服务模块进行测试。这里面包含了许多子测试，由于篇幅限制我们只给出其中两个服务的测试用例，见表6-3。

表 6-3 服务集成测试用例表

测试函数	测试内容	期望结果
ProblemService测试		
		接下页

接上页

测试函数	测试内容	期望结果
testGetProblemDTOByProblemId	按题目ID查询ProblemDTO	正常
testCount	题目数量查询	正常
testUpdateProblem	题目更新	正常
testUpdateProblem_problemNotFound	题目更新（更新不存在的题目）	抛出异常
testUpdateProblem_problemFoundWithNullId	题目更新（题目ID错误）	抛出异常
testCreateNewProblem	新建题目	正常
testGetProblemListDTOList	获取题目列表	正常
testGetAllVisibleProblemIds	获取可见题目ID列表	正常
UserService测试		
testGetUserByUserId	按用户ID查询用户	正常
testGetUserByUserName	按用户名查询用户	正常
testGetUserByEmail	按用户邮箱查询用户	正常
testUpdateUser	更新用户	正常
testCount_emptyCondition	用户数量查询（条件为空）	正常
testCount_byStartId	用户数量查询（ID大于等于2）	正常
testCount_byEndId	用户数量查询（ID小于等于2）	正常
testCount_byStartIdAndEndId	用户数量查询（ID在2到10之间）	正常
testCount_byStartIdAndEndId_empty	用户数量查询（不存在ID区间）	正常
testCount_byDepartmentId	用户数量查询（按学院查询）	正常
testCount_bySchool	用户数量查询（按学校查询）	正常
testCount_bySchool_empty	用户数量查询（不存在的学校）	正常
testCount_byUserName	用户数量查询（按用户名查询）	正常
testCount_byType	用户数量查询（按类型查询）	正常

6.2.4 实用工具测试

系统中我们用到了许多实用工具，例如数据解压工具、数组工具等，这些工具我们也作出了相应的测试。测试用例见表6-4

表 6-4 实用工具测试用例表

测试函数	测试内容	期望结果
题目数据解压工具测试		
testCheck_withoutSpjFile_oneCase	一组数据，无SPJ	解压正常
testCheck_withoutSpjFile_moreCases	多组数据，无SPJ	解压正常
testCheck_withSpjFile_oneCase	一组数据，有SPJ	解压正常
testCheck_withSpjFile_moreCases	多组数据，有SPJ	解压正常
testCheck_withInvalidDataName	无效的数据	解压失败
testCheck_withDirectory	包含无效目录	解压失败
testCheck_withNullDirectory	空压缩包	解压失败
testCheck_invalidDataDirectory	非法数据文件夹	解压失败
testCheck_notSameInputsAndOutputs	无对应的输出文件	解压失败
testCheck_missMatchInputAndOutput	输入输出文件不匹配	解压失败
数组工具测试		
testParseIntArray	解析整数数组	正常
文件比较工具测试		
testSame	完全相等	相等
testSame_deletingWhiteSpace	有多余空格	相等
testSame_endingSpaces	末尾空格	相等
testSame_tabSpaces	多余tab	相等
testDifferent_endingSpaces	不同的字符串	不等
testDifferent_specialCharacter	不同字符串，特殊字符	不等

6.2.5 控制器测试

在服务测试通过的情况下，我们可以在控制器中设置服务的模拟对象来进行控制器测试。这里面包含了许多子测试，由于篇幅限制我们只给出其中三个控制器的测试用例，见表6-5。

表 6-5 控制器测试用例表

测试函数	测试内容	期望结果
IndexController测试		

接下页

接上页

测试函数	测试内容	期望结果
testVisitIndex	主页访问	正常
ProblemController测试		
testShow_successful	题目页面（访问成功）	正常
testShow_problemNotFound	题目页面（不存在的题目）	跳转到404页面
testList	题目列表页面	正常
UserController测试		
testLogin_successful	登陆（成功）	正常
testLogin_invalidUserName_null	登陆（用户名为空）	验证失败
testLogin_invalidUserName_tooShort	登陆（用户名太短）	验证失败
testLogin_invalidUserName_tooLong	登陆（用户名太长）	验证失败
testLogin_invalidUserName_invalid	登陆（用户名非法）	验证失败
testLogin_invalidPassword_null	登陆（密码为空）	验证失败
testLogin_invalidPassword_tooShort	登陆（密码太短）	验证失败
testLogin_invalidPassword_tooLong	登陆（密码太长）	验证失败
testLogin_failed_wrongUserNameOrPassword	登陆（密码错误）	验证失败
testLogin_failed_bothUserNameAndPassword_null	登陆（用户名密码都为空）	验证失败
testLogin_failed_serviceError	登陆（系统错误）	返回错误信息
testLogout_successful	登出（成功）	正常
testRegister_successfully	注册（成功）	正常
testRegister_failed_userName_null	注册（用户名为空）	验证失败
testRegister_failed_userName_whiteSpaces	注册（用户名有空格）	验证失败
testRegister_failed_userName_tooShort	注册（用户名太短）	验证失败
testRegister_failed_userName_tooLong	注册（用户名太长）	验证失败
testRegister_failed_userName_invalid	注册（用户名非法）	验证失败
testRegister_failed_password_null	注册（密码为空）	验证失败
testRegister_failed_password_tooShort	注册（密码太短）	验证失败
testRegister_failed_password_tooLong	注册（密码太长）	验证失败
testRegister_failed_passwordRepeat_null	注册（密码确认为空）	验证失败
testRegister_failed_passwordRepeat_tooShort	注册（密码确认太短）	验证失败
testRegister_failed_passwordRepeat_tooLong	注册（密码确认太长）	验证失败
testRegister_failed_passwordRepeat_different	注册（密码确认与密码不同）	验证失败

接下页

测试函数	测试内容	期望结果
testRegister_failed_nickName_null	注册（昵称为空）	验证失败
testRegister_failed_nickName_whiteSpaces	注册（昵称有空格）	验证失败
testRegister_failed_nickName_tooShort	注册（昵称太短）	验证失败
testRegister_failed_nickName_tooLong	注册（昵称太长）	验证失败
testRegister_failed_nickName_invalid	注册（昵称非法）	验证失败
testRegister_failed_email_invalid	注册（邮箱错误）	验证失败
testRegister_failed_school_tooShort	注册（学校名太短）	验证失败
testRegister_failed_school_tooLong	注册（学校名太长）	验证失败
testRegister_failed_departmentId_null	注册（学院为空）	验证失败
testRegister_failed_departmentNotFound	注册（学院不存在）	验证失败
testRegister_failed_studentId_tooShort	注册（学号太短）	验证失败
testRegister_failed_studentId_tooLong	注册（学号太长）	验证失败
testRegister_failed_usedUserName	注册（用户名已被占用）	验证失败
testRegister_failed_usedEmail	注册（邮箱已被占用）	验证失败
testUser_register_login_logout	用户注册、登陆、登出操作	正常

控制器是直接和用户打交道的函数，我们选择了非常多的数据来检验用户提交表单的各种情况，防止用户的恶意提交对数据库造成破坏。

6.3 测试结果

在mvn test即可进行自动化测试，下面是测试的运行结果：

```
-----  
T E S T S  
-----  
  
Running TestSuite  
PASSED: UserServiceTest  
PASSED: ProblemServiceTest  
PASSED: ConditionTest  
PASSED: ArrayUtilTest  
PASSED: CompareSkipSpacesTest
```

```
PASSED: ZipDataCheckerTest
PASSED: IndexControllerTest
PASSED: UserControllerTest
PASSED: ProblemControllerTest
PASSED: Database integration tests
PASSED: Service integration tests
PASSED: Utility integration tests
Tests run: 183, Failures: 0, Errors: 0, Skipped: 0
Time elapsed: 10.53 sec - in TestSuite
```

Results :

```
Tests run: 183, Failures: 0, Errors: 0, Skipped: 0
```

可以看到所有测试用例全部通过测试。

第7章 结束语

在课题的概要设计和详细设计中，已经给出了软件的结构和详细设计方法。同时也用在文中提到的开发环境实现了绝大多数的功能，并且经过多次测试，本系统能够在多种浏览器平台下顺利地运行，各个功能均能正常运行且没有漏洞，整个系统具有良好的用户体验。这些都达到了预期的期望和课题要求。

本人也通过本课题的研究、学习、设计和程序实现，了解了软件工程的主要的设计方法和流程，也通过学习资料掌握了基本的Web开发知识。在这个过程中，体会到了想开发出一个优秀的Web应用需要在各方面进行精心设计，才能做出一个开发者和用户都满意的应用。

在未来，本人希望该系统能够发展成为一个以编程为中心的交流平台，用户可以将自己出的题目发布到平台上，供大家一起交流、学习，达到共同进步的目的。

参考文献

- [1] B. Beizer. Black-box testing: techniques for functional testing of software and systems[M].[S.l.]: John Wiley & Sons, Inc., 1995.
- [2] 尤枫, 史晟辉. ACM 在线评测在编译原理实践教学中的应用探讨[J]. 计算机教育, 2009, 20:113–115.
- [3] 郭嵩山, 王磊, 张子臻. ACM/ICPC 与创新型 IT 人才的培养[J]. 实验室研究与探索, 2007, 26(12):181–189.
- [4] <http://developer.chrome.com/>. MVC Architecture[EB/OL]. http://developer.chrome.com/apps/app_frameworks.
- [5] R. Johnson. Expert one-on-one J2EE Design and Development[M].[S.l.]: John Wiley & Sons, 2004.
- [6] 张海藩,等. 软件工程导论[M]. 1998, 北京: 清华大学出版社, 卷1.
- [7] C. Szyperski. Component software: beyond object-oriented programming[M].[S.l.]: Pearson Education, 2002.

致 谢

历时将近两个月的时间终于将这篇论文写完，在论文的写作过程中遇到了无数的困难和障碍，都在同学和老师的帮助下度过了。尤其要强烈感谢我的论文指导老师——饶力老师，他对我进行了无私的指导和帮助，不厌其烦的帮助进行论文的修改和改进。另外，在校图书馆查找资料的时候，图书馆的老师也给我提供了很多方面的支持与帮助。在此向帮助和指导过我的各位老师表示最中心的感谢！

感谢这篇论文所涉及到的各位学者。本文引用了数位学者的研究文献，如果没有各位学者的研究成果的帮助和启发，我将很难完成本篇论文的写作。

感谢时富军同学给我们提供了电子科技大学毕业论文的 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 模板，为我们排版论文带来了无比的便捷。

感谢何云鹏、范坚劲和刘施剑学长，为整个系统的体系架构提出了许多丰富的建议。他们在忙碌的工作之余向本系统中提交了许多关键代码，对整个系统进行了深入的审查，确保系统代码的质量。

感谢我的同学和朋友，在我写论文的过程中给予我了很多素材，还在论文的撰写过程中提供热情的帮助。由于我的学术水平有限，所写论文难免有不足之处，恳请各位老师和学友批评和指正！

附录 A 部分程序源代码

A.1 ApplicationContextConfig.java

```
001 /**
002  * Application Context Configuration.
003  */
004 @Configuration
005 // 设置Spring需要扫描的目录
006 @ComponentScan(basePackages = {
007     "cn.edu.uestc.acmicpc.db",
008     "cn.edu.uestc.acmicpc.judge",
009     "cn.edu.uestc.acmicpc.util",
010     "cn.edu.uestc.acmicpc.service",
011     "cn.edu.uestc.acmicpc.web.aspect"
012 })
013 // 设置resources文件的地址
014 @PropertySource("classpath:resources.properties")
015 // 开启事务管理
016 @EnableTransactionManagement
017 public class ApplicationContextConfig {
018
019     @Autowired
020     private Environment environment;
021
022     /**
023      * Bean: Judge service.
024      *
025      * JudgeService is a singleton instance and need start at 2
026      first.
```

```

027     *
028     * @return judgeService bean
029     */
030     // 对Judge Service的配置
031     @Bean(name = "judgeService")
032     @Scope(ConfigurableBeanFactory.SCOPE_SINGLETON)
033     @Lazy(false)
034     public JudgeService judgeService() {
035         return new JudgeService();
036     }
037
038     /**
039     * Bean: Data source
040     *
041     * We use BoneCP to manage connection poll.
042     *
043     * @return dataSource bean
044     */
045     // 对数据源的配置
046     @Bean(name = "dataSource", destroyMethod = "close")
047     public BoneCPDataSource dataSource() {
048         BoneCPDataSource dataSource = new BoneCPDataSource();
049
050         dataSource.setDriverClass(getProperty("db.driver"));
051         dataSource.setJdbcUrl(getProperty("db.url"));
052         dataSource.setUsername(getProperty("db.username"));
053         dataSource.setPassword(getProperty("db.password"));
054         dataSource.setMaxConnectionsPerPartition(Integer
055             .parseInt(getProperty("db.2
056             maxConnectionsPerPartition"))));
057         dataSource.setMinConnectionsPerPartition(Integer

```

```
058         .parseInt(getProperty("db.2
059         minConnectionsPerPartition"))));
060     dataSource.setPartitionCount(Integer.parseInt(2
061     getProperty("db.partitionCount"))));
062     dataSource.setAcquireIncrement(Integer.parseInt(2
063     getProperty("db.acquireIncrement"))));
064     dataSource.setStatementsCacheSize(Integer.parseInt(2
065     getProperty("db.statementsCacheSize"))));
066     return dataSource;
067 }
068
069 /**
070  * Bean: session factory.
071  *
072  * @return sessionFactory bean
073  */
074 // Session Factory
075 @Bean(name = "sessionFactory")
076 @Lazy(false)
077 public LocalSessionFactoryBean sessionFactory() {
078     LocalSessionFactoryBean localSessionFactoryBean = new 2
079     LocalSessionFactoryBean();
080
081     localSessionFactoryBean.setDataSource(this.dataSource())2
082     ;
083     localSessionFactoryBean.setHibernateProperties(this.2
084     getHibernateProperties());
085     localSessionFactoryBean.setAnnotatedClasses(Article.2
086     class,
087     Code.class,
088     CompileInfo.class,
```

```

089         Contest.class,
090         ContestProblem.class,
091         ContestTeamInfo.class,
092         ContestUser.class,
093         Department.class,
094         Language.class,
095         Message.class,
096         Problem.class,
097         ProblemTag.class,
098         Status.class,
099         Tag.class,
100         User.class,
101         UserSerialKey.class);
102
103     return localSessionFactoryBean;
104 }
105
106 /**
107  * Bean: transaction manager.
108  *
109  * @return transactionManagerBean
110  */
111 // 事务管理工具
112 @Bean(name = "transactionManager")
113 public HibernateTransactionManager transactionManager() {
114     HibernateTransactionManager transactionManager = new
115     HibernateTransactionManager();
116     transactionManager.setSessionFactory(this.
117     sessionFactory().getObject());
118     return transactionManager;
119 }

```

```
120
121  /**
122   * Hibernate properties.
123   *
124   * @return properties
125   */
126  // Hibernate配置
127  private Properties getHibernateProperties() {
128      Properties properties = new Properties();
129      properties.setProperty("hibernate.dialect", environment.2
130      getProperty("hibernate.dialect"));
131      properties.setProperty("hibernate.show_sql", 2
132      environment.getProperty("hibernate.show_sql"));
133      properties.setProperty("hibernate.format", environment.2
134      getProperty("hibernate.format_sql"));
135      properties.setProperty("hibernate.2
136      current_session_context_class",
137          environment.getProperty("hibernate.2
138          current_session_context_class"));
139      return properties;
140  }
141
142  /**
143   * Simply get property in PropertySource.
144   *
145   * @param name property name
146   * @return property value
147   */
148  private String getProperty(final String name) {
149      return environment.getProperty(name);
150  }
```

151 }

A.2 resources.properties

```
001 #数据驱动
002 db.driver=com.mysql.jdbc.Driver
003 #数据库连接地址
004 db.url=jdbc:mysql://localhost:2
005 3306/uestcoj?useUnicode=true&characterEncoding=UTF-8
006 #数据库用户名及密码
007 db.username=root
008 db.password=root
009
010 #以下是数据库性能配置
011 db.maxConnectionsPerPartition=30
012 db.minConnectionsPerPartition=10
013 db.partitionCount=3
014 db.acquireIncrement=5
015 db.statementsCacheSize=100
016 db.idleMaxAge=5
017 db.idleConnectionTestPeriod=5
018
019 #Hibernate配置
020 hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
021 hibernate.show_sql=false
022 hibernate.format_sql=false
023 hibernate.current_session_context_class=org.springframework.2
024 orm.hibernate4.SpringSessionContext
025
026 sessionFactory.annotatedPackages=cn.edu.uestc.acmicpc.db.2
```

027 entity

A.3 WebMVCResource.java

```
001 /**
002  * Spring MVC Configuration file resource.
003  */
004 public class WebMVCResource {
005
006     // 视图解析器配置
007     public static ViewResolver viewResolver() {
008         InternalResourceViewResolver viewResolver = new
009             InternalResourceViewResolver();
010
011         viewResolver.setPrefix("/WEB-INF/views/");
012         viewResolver.setSuffix(".jsp");
013
014         return viewResolver;
015     }
016
017     // 设置FastJson为默认的JSON格式转换器
018     public static HttpMessageConverter<?>[] messageConverters(
019 ) {
020         HttpMessageConverter<?>[] converters = new
021             HttpMessageConverter<?>[1];
022         FastJsonHttpMessageConverter
023             fastJsonHttpMessageConverter = new
024             FastJsonHttpMessageConverter();
025         List<MediaType> mediaTypes = new LinkedList<>();
026         mediaTypes.add(MediaType.APPLICATION_JSON);
027         fastJsonHttpMessageConverter.setSupportedMediaTypes(
```

```

028     mediaTypes);
029     converters[0] = fastJsonHttpMessageConverter;
030     return converters;
031 }
032
033 }
```

A.4 WebMVCConfig.java

```

001 /**
002  * Spring MVC configuration file.
003  */
004 @Configuration
005 // 开启Web MVC模式
006 @EnableWebMvc
007 // 指定Controller包位置
008 @ComponentScan(basePackages = {
009     "cn.edu.uestc.acmicpc.web.oj.controller"
010 })
011 // 开启AspectJ代理
012 @EnableAspectJAutoProxy(proxyTargetClass = true)
013 public class WebMVCConfig extends WebMvcConfigurerAdapter {
014
015     // 注册静态资源代理
016     @Override
017     public void addResourceHandlers(ResourceHandlerRegistry registry) {
018
019         registry.addResourceHandler("/images/**").addResourceLocations("/images/**");
020         registry.addResourceHandler("/plugins/**").addResourceLocations("/plugins/**");
021
022     }
```



```
023     registry.addHandler("/scripts/**").>
024     addResourceLocations("/scripts/**");
025     registry.addHandler("/styles/**").>
026     addResourceLocations("/styles/**");
027     registry.addHandler("/font/**").>
028     addResourceLocations("/font/**");
029 }
030
031 // 开启ServletHandler
032 @Override
033 public void configureDefaultServletHandling(
034     DefaultServletHandlerConfigurer >
035     defaultServletHandlerConfigurer) {
036     defaultServletHandlerConfigurer.enable();
037 }
038
039 // 注册消息转换器
040 @Override
041 public void configureMessageConverters(List<>
042     HttpMessageConverter<?>> converters) {
043     converters.addAll(Arrays.asList(WebMVCResource.>
044     messageConverters()));
045 }
046
047 // 视图解析器
048 @Bean
049 public ViewResolver viewResolver() {
050     return WebMVCResource.viewResolver();
051 }
052
053 // 启用多文件上传
```

```

054  @Bean
055  public MultipartResolver multipartResolver() {
056      return new CommonsMultipartResolver();
057  }
058
059 }

```

A.5 spj.cc

```

001 #include <stdio>
002 #include <stdlib>
003
004 #define WA 0
005 #define AC 1
006 #define PE 2
007 #define SE 3
008 #define EPS 1e-8
009
010 /**
011  * @brief
012  *   Basic judge item for judger, store stdard input and 2
013  *   output, user output
014  *   FILE* entity.
015  */
016 struct Node {
017     FILE* data_input;
018     FILE* data_output;
019     FILE* user_output;
020 };
021
022 static Node* node_pointer;

```

```
023
024 /**
025  * @brief
026  *   if statement is not true, print exit_code's string 2
027  and exit SPJ
028  */
029 void dump(bool statement, const int& exit_code) {
030     if (statement)
031         return;
032     if (node_pointer->data_input != NULL)
033         fclose(node_pointer->data_input);
034     if (node_pointer->data_output != NULL)
035         fclose(node_pointer->data_output);
036     if (node_pointer->user_output != NULL)
037         fclose(node_pointer->user_output);
038     if (exit_code == WA)
039         puts("WA");
040     else if (exit_code == AC)
041         puts("AC");
042     else if (exit_code == PE)
043         puts("PE");
044     else
045         puts("SE");
046     delete node_pointer;
047     exit(0);
048 }
049
050 int nextInt(FILE* fp) {
051     int variable;
052     dump(fscanf(fp, "%d", &variable) == 1, fp == stdin ? WA : 2
053     SE);
```

```

054   return variable;
055 }
056
057 long long nextLong(FILE* fp) {
058     long long variable;
059     dump(fscanf(fp, "%lld", &variable) == 1, fp == stdin ? WA :
060         SE);
061     return variable;
062 }
063
064 double nextDouble(FILE* fp) {
065     double variable;
066     dump(fscanf(fp, "%lf", &variable) == 1, fp == stdin ? WA :
067         SE);
068     return variable;
069 }
070
071 void nextString(FILE* fp, char* buf) {
072     dump(fscanf(fp, "%s", buf) == 1, fp == stdin ? WA : SE);
073 }
074
075 int sgn(double variable) {
076     return variable < -EPS ? -1 : variable > EPS;
077 }
078
079 int compare(double lhs, double rhs) {
080     return sgn(lhs - rhs);
081 }
082
083 /**
084  * @brief

```

```
085 *    Judge for specific problem, only function needed to 2
086 modify.
087 * @author RuinsHe<lyhypacm@gmail.com>
088 */
089 void judge() {
090 }
091
092 int main(int argc, char* argv[]) {
093     node_pointer = new Node();
094     dump(argc >= 3, SE);
095     node_pointer->data_input = fopen(argv[1], "r");
096     dump(node_pointer->data_input != NULL, SE);
097     node_pointer->data_output = fopen(argv[2], "r");
098     dump(node_pointer->data_output != NULL, SE);
099     node_pointer->user_output = stdin;
100     dump(node_pointer->user_output != NULL, SE);
101     judge();
102     dump(false, AC);
103     return 0;
104 }
```

A.6 AuthenticationAspect.java

```
001 /**
002  * Authentication aspect
003  */
004 @Component
005 // 声明侧面
006 @Aspect
007 public class AuthenticationAspect {
008
```

```

009  /**
010   * Http request
011   */
012  private HttpServletRequest request;
013
014  // 注入HttpServletRequest
015  @Autowired(required = true)
016  public void setRequest(HttpServletRequest request) {
017      this.request = request;
018  }
019
020  // 2
021  声明该侧面的作用范围为包围在所有有LoginPermit2
022  rmit注解的函数上
023  @Around("@annotation(cn.edu.uestc.acmicpc.util.annotation.2
024  LoginPermit)")
025  public Object checkAuth(ProceedingJoinPoint 2
026  proceedingJoinPoint) throws Throwable {
027      MethodSignature methodSignature = (MethodSignature) 2
028      proceedingJoinPoint.getSignature();
029      Method method = methodSignature.getMethod();
030      // 获取函数上的LoginPermit注解
031      LoginPermit permit = method.getAnnotation(LoginPermit.2
032      class);
033
034      try {
035          // 如果该函数需要登录后才能访问
036          if (permit.NeedLogin()) {
037              // 获得当前用户
038              UserDTO userDTO = (UserDTO) request.getSession().2
039              getAttribute("currentUser");

```

```
040      // 2
041      如果当前未登录，那么就抛出权限异常
042      if (userDTO == null) {
043          throw new AppException("Permission denied");
044      }
045      // 如果权限不足，那么就抛出权限异常
046      if (permit.value() != Global.AuthenticationType.NORMAL) {
047          if (userDTO.getType() != permit.value().ordinal()) {
048              throw new AppException("Permission denied");
049          }
050      }
051      }
052      }
053      }
054      // 2
055      至此，权限验证已经完毕，继续函数的运行
056      return proceedingJoinPoint.proceed();
057  } catch (AppException e) {
058      // 捕获权限异常
059      // 2
060      如果该方法返回值是视图地址，那么我们
061      将其重定向到错误页面上
062      if (method.getReturnType() == String.class) {
063          return "redirect:/error/authenticationError";
064      }
065      // 2
066      否则这个方法返回的是一个Map，我们要做的是设置相应的错误信息
067      else {
068          Map<String, Object> json = new HashMap<>();
069      }
```

```

071         json.put("result", "error");
072         json.put("error_msg", e.getMessage());
073         return json;
074     }
075 }
076 }
077
078 }
```

A.7 navbarTop.jsp

```

001 <%--
002 User menu on navbar
003 --%>
004 <%@ taglib prefix="c" uri="http://java.sun.2
005 com/jsp/jstl/core" %>
006 <%@ taglib uri="http://www.opensymphony.2
007 com/sitemesh/decorator"
008         prefix="decorator" %>
009 <%@ taglib uri="http://www.opensymphony.com/sitemesh/page" 2
010 prefix="page" %>
011 <%@ page contentType="text/html; charset=UTF-8" 2
012 language="java" %>
013 <html>
014 <head>
015     <title></title>
016 </head>
017 <body>
018 <header id="cdoj-navbar-top">
019     <div class="container">
020         <nav>
```



```

021     <ul class="nav navbar-nav navbar-right">
022         <!-- Search -->
023         <li>
024             <div class="input-group input-group-sm" id="cdoj-2
025             search">
026                 <input type="text" class="form-control">
027
028                 <div class="input-group-btn">
029                     <button type="button" class="btn btn-default"
030                         tabindex="-1">
031                         <span id="cdoj-search-icon"><i class="fa fa-2
032                         search"></i></span>
033                         <span id="cdoj-search-text">Search</span>
034                     </button>
035                 </div>
036             </div>
037         </li>
038         <!-- User -->
039         <li ng-controller="UserController">
040             </li>
041     </ul>
042 </nav>
043 </div>
044 </header>
045 </body>
046 </html>

```

A.8 ui.flandre.coffee

```

001 # 一个简单的Markdown编辑器
002 cdoj.directive("uiFlandre",

```

```

003 ->
004   restrict: "A"
005   scope:
006     content: "=ngModel"
007     uploadUrl: "@"
008   controller: [
009     "$scope", "$element",
010     ($scope, $element) ->
011       $scope.mode = "edit"
012       $scope.previewContent = ""
013       $editor = $element.find(".flandre-editor")
014
015       # 预览按钮事件绑定
016       $scope.togglePreview = ->
017         if $scope.mode == "edit"
018           # preview
019           $scope.previewContent = $scope.content
020           $scope.mode = "preview"
021         else
022           # edit
023           $scope.mode = "edit"
024
025       # 图片上传
026       pictureUploader = new qq.FineUploaderBasic(
027         button: $element.find(".flandre-picture-uploader")[2
028         0]
029         request:
030           endpoint: $scope.uploadUrl
031           inputName: "uploadFile"
032         validation:
033           allowedExtensions: ["jpeg", "jpg", "gif", "png"],

```

```
034         sizeLimit: 10 * 1000 * 1000 # 10 MB
035     multiple: false
036     callbacks:
037         onComplete: (id, fileName, data) ->
038             if data.success == "true"
039                 value = "![title]({data.uploadedFileUrl})"
040                 position = $editor.getCursorPosition()
041                 oldText = $editor.val()
042                 # 将代码插入到文章中
043                 oldText = oldText.insert(value, position)
044                 $scope.$apply(->
045                     $scope.content = oldText
046                 )
047                 # ![title](...)
048                 # ~~~~~
049                 # 同时选中title部分
050                 $editor.setSelection(position + 2, position + 2
051                     7)
052             else
053                 # TODO
054                 console.log(data)
055         )
056 ]
057 template: ""
058 <div class="panel panel-default">
059     <div class="panel-heading flandre-heading">
060         <div class="btn-toolbar" role="toolbar">
061             <div class="btn-group">
062                 <button type="button" class="btn btn-default 2
063                     btn-sm"
064                     ng-click="togglePreview()"
```

```

065         ng-class="{active: mode == 'preview'}"
066         ">Preview</button>
067     </div>
068     <div class="btn-group flandre-tools">
069         <span class="btn btn-default btn-sm"><i
070         class="fa fa-smile-o"></i></span>
071         <span class="btn btn-default btn-sm flandre-
072         picture-uploader"><i class="fa fa-picture-o"><
073         /i></span>
074     </div>
075 </div>
076 </div>
077 <textarea class="tex2jax_ignore form-control
078 flandre-editor"
079         msd-elastic
080         ng-class="{ 'flandre-show': mode == 'edit' }"
081         "
082         ng-model="content"></textarea>
083 <div class="flandre-preview" ng-class="{ 'flandre-
084 show': mode == 'preview' }">
085     <div ui-markdown content="previewContent"></div>
086 </div>
087 </div>
088 ""
089 replace: true
090 )

```

A.9 controller.listController.coffee

```

001 cdoj.controller("ListController", [
002     "$scope", "$rootScope", "$http",

```

```
003 ($scope, $rootScope, $http) ->
004   $scope.condition = 0
005   $scope.list = []
006   # 翻页导航
007   $scope.pageInfo =
008     countPerPage: 20
009     currentPage: 1
010     displayDistance: 2
011     totalPages: 1
012   # 列表的请求地址
013   $scope.requestUrl = 0
014   # 查询条件重置
015   $scope.reset = ->
016     ..each($scope.condition, (value, index) ->
017       $scope.condition[index] = undefined
018     )
019   $scope.condition["currentPage"] = null
020   # 列表刷新
021   $scope.refresh = ->
022     if $scope.requestUrl != 0
023       condition = angular.copy($scope.condition)
024       $http.post($scope.requestUrl, condition).then (2
025         response) =>
026           $scope.list = response.data.list
027           $scope.pageInfo = response.data.pageInfo
028   # 在用户登陆/登出时刷新
029   $rootScope.$watch("currentUser", ->
030     $scope.refresh()
031   , true)
032   # 在查询条件发生变化时刷新
033   $scope.$watch("condition", ->
```

```
034         $scope.refresh()  
035     , true  
036     )  
037 ])
```

MVC Architecture[4]

As modern browsers become more powerful with rich features, building full-blown web applications in JavaScript is not only feasible, but increasingly popular. Based on trends on HTTP Archive, deployed JavaScript code size has grown 45% over the course of the year.

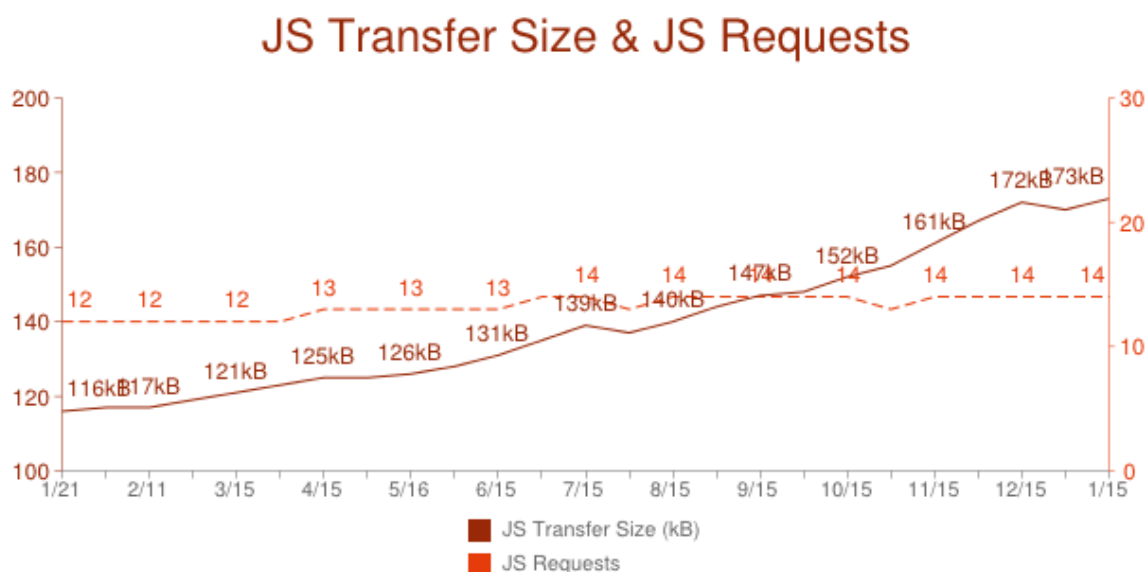


Figure A-1 JS Transfer Size & JS Requests

With JavaScript's popularity climbing, our client-side applications are much more complex than before. Application development requires collaboration from multiple developers. Writing **maintainable** and **reusable** code is crucial in the new web app era. The Chrome App, with its rich client-side features, is no exception.

Design patterns are important to write maintainable and reusable code. A pattern is a reusable solution that can be applied to commonly occurring problems in software design — in our case — writing Chrome Apps. We recommend that developers decouple the app into a series of independent components following the MVC pattern.

In the last few years, a series of JavaScript MVC frameworks have been developed, such as backbone.js, ember.js, AngularJS, Sencha, Kendo UI, and more. While they all have their unique advantages, each one of them follows some form of MVC pattern with the goal of encouraging developers to write more structured JavaScript code.

1.1 MVC pattern overview

MVC offers architectural benefits over standard JavaScript — it helps you write better organized, and therefore more maintainable code. This pattern has been used and extensively tested over multiple languages and generations of programmers.

MVC is composed of three components:

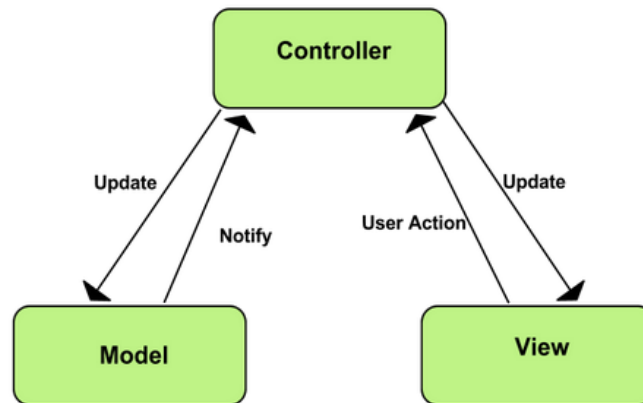


Figure A-2 MVC

1.1.1 Mode

Model is where the application's data objects are stored. The model doesn't know anything about views and controllers. When a model changes, typically it will notify its observers that a change has occurred.

To understand this further, let's use the Todo list app, a simple, one page web app that tracks your task list.

The model here represents attributes associated with each todo item such as description and status. When a new todo item is created, it is stored in an instance of the model.

1.1.2 View

View is what's presented to the users and how users interact with the app. The view is made with HTML, CSS, JavaScript and often templates. This part of your Chrome App has access to the DOM.

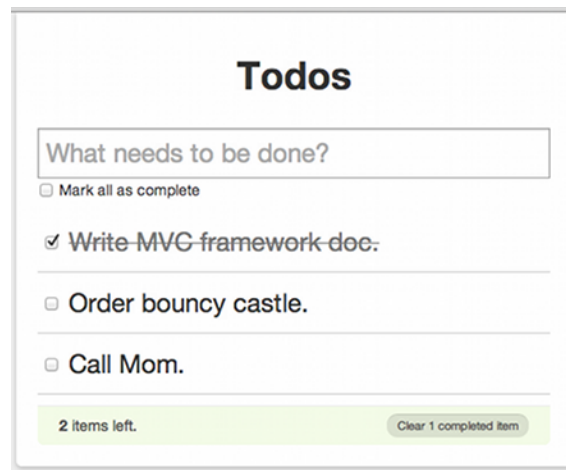


Figure A-3 Todo list app

For example, in the above todo list web app, you can create a view that nicely presents the list of todo items to your users. Users can also enter a new todo item through some input format; however, the view doesn't know how to update the model because that's the controller's job.

1.1.3 Controller

The controller is the decision maker and the glue between the model and view. The controller updates the view when the model changes. It also adds event listeners to the view and updates the model when the user manipulates the view.

In the todo list web app, when the user checks an item as completed, the click is forwarded to the controller. The controller modifies the model to mark item as completed. If the data needs to be persistent, it also makes an async save to the server. In rich client-side web app development such as Chrome Apps, keeping the data persistent in local storage is also crucial. In this case, the controller also handles saving the data to the client-side storage such as FileSystem API.

There are a few variations of the MVC design pattern such as MVP (Model-View-Presenter) and MVVP(Model-View-ViewModel). Even with the so called MVC design pattern itself, there is some variation between the traditional MVC pattern vs the modern interpretation in various programming languages. For example, some MVC-based frameworks will have the view observe the changes in the models while others will let the controller

handle the view update. This article is not focused on the comparison of various implementations but rather on the separation-of-concerns and its importance in writing modern web apps.

If you are interested in learning more, we recommend Addy Osmani's online book: *Learning JavaScript Design Patterns*.

To summarize, the MVC pattern brings modularity to application developers and it enables:

- Reusable and extendable code.
- Separation of view logic from business logic.
- Allow simultaneous work between developers who are responsible for different components (such as UI layer and core logic).
- Easier to maintain.

1.2 MVC persistence patterns

There are many different ways of implementing persistence with an MVC framework, each with different trade-offs. When writing Chrome Apps, choose the frameworks with MVC and persistence patterns that feel natural to you and fit your application needs.

1.2.1 Model does its own persistence - ActiveRecord pattern

Popular in both server-side frameworks like Ruby on Rails, and client-side frameworks like Backbone.js and ember.js, the ActiveRecord pattern places the responsibility for persistence on the model itself and is typically implemented via JSON API.

A slightly different take from having a model handle the persistence is to introduce a separate concept of Store and Adapter API. Store, Model and Adapter (in some frameworks it is called Proxy) work hand by hand. Store is the repository that holds the loaded models, and it also provides functions such as creating, querying and filtering the model instances contained within it.

An adapter, or a proxy, receives the requests from a store and translates them into appropriate actions to take against your persistent data layer (such as JSON API). This is

interesting in the modern web app design because you often interact with more than one persistent data layer such as a remote server and browser's local storage. Chrome Apps provides both Chrome Storage API and HTML 5 FileSystem API for client side storage.

Pros:

- Simple to use and understand.

Cons:

- Hard to test since the persistence layer is 'baked' into the object hierarchy.
- Having different objects use different persistent stores is difficult (for example, FileSystem APIs vs indexedDB vs server-side).
- Reusing Model in other applications may create conflicts, such as sharing a single Customer class between two different views, each view wanting to save to different places.

1.2.2 Controller does persistence

In this pattern, the controller holds a reference to both the model and a datastore and is responsible for keeping the model persisted. The controller responds to lifecycle events like Load, Save, Delete, and issues commands to the datastore to fetch or update the model.

Pros:

- Easier to test, controller can be passed a mock datastore to write tests against.
- The same model can be reused with multiple datastores just by constructing controllers with different datastores.

Cons:

- Code can be more complex to maintain.

1.2.3 ApplicationController does persistence

In some patterns, there is a supervising controller responsible for navigating between one MVC and another. The ApplicationController decides, for example, that a ‘Back’ button moves the client from an editing screen (which contains MVC widgets/formats), to a settings screen.

In the ApplicationController pattern, the ApplicationController responds to events and changes the app’s current screen by issuing a call to the datastore to load any models needed and constructing all of the matching views and controllers for that screen.

Pros:

- Moves persistence layer even higher up the stack where it can be easily changed.
- Doesn’t pollute lower level controllers like a DatePickerController with the need to know about persistence.

Cons:

- Each ‘Page/Screen’ of the app now requires a lot of boilerplate to write or update: Model, View, Controller, ApplicationController.

MVC架构

在现代浏览器拥有越来越丰富的新特性的当下，构建功能成熟的web应用已经不再是一件困难的事，而是更多开发者的选择。通过对网络中的HTTP数据的分析，目前已经部署的JavaScript代码在上一年之中增长了45%。

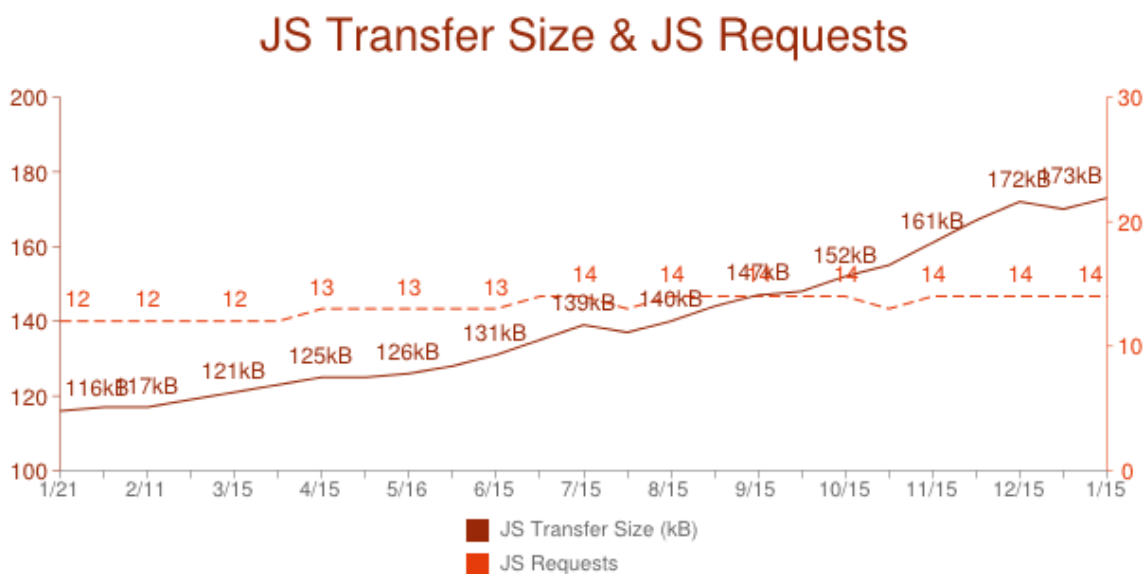


图 A-1 JS Transfer Size & JS Requests

随着JavaScript变得越来越流行，客户端应用也变得越来越复杂。开发一个应用需要许多开发者的合力协作。书写维护性与复用性强的代码成为了当今web应用时代的关键。Chrome应用——以及它丰富的特性，也不例外。

设计模式在如何书写维护性与复用性良好的代码中扮演着重要的角色。在我们的案例中，当我们构建一个Chrome应用时，一个可复用的模式可以应用于在软件设计中遇到的许许多多常见的问题。我们建议开发者们通过MVC模式将应用分解成一系列相互独立的部分。

在最近几年，一系列基于JavaScript的MVC框架先后问世：backbone.js, ember.js, AngularJS, Sencha, Kendo UI……它们各有特点，但是都遵守MVC模式——为了让开发者写出更加结构化的JavaScript代码。

1.1 MVC模式概览

MVC模式在标准JavaScript之上提供了一个有益的架构——它帮助我们写出更有组织性的，维护性强的代码。这种模式还被应用于许多编程语言，并且通过了几代程序员的广泛考验。

MVC模式由三部分组成：

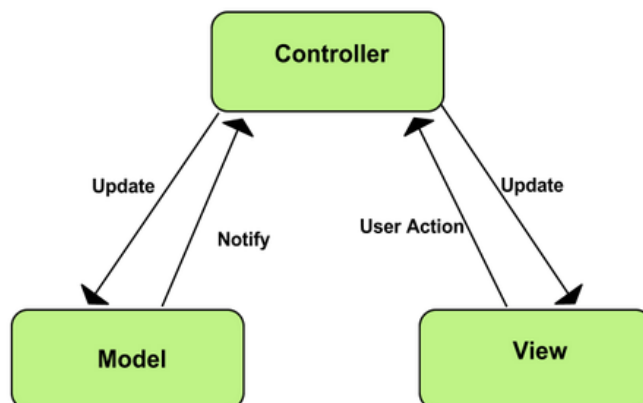


图 A-2 MVC模式

1.1.1 模型（Model）

模型被用于储存应用程序的数据对象。模型不知道视图和控制器的任何细节。当一个模型发生了改变，它会将这个改变通知它的观察者们。

为了理解这种特性，让我们看看这个待办列表应用，它是一个简单的能够管理用户的任务列表的单页应用。

在这里模型被用于表示每个待办项目的属性，例如描述（description）和状态（status）。每当一个新的待办项目被建立时，它被保存在该模型的一个新的实例中。

1.1.2 视图（View）

视图被用于向用户展示数据以及与用户进行交互。视图由HTML，CSS，JavaScript和模板组成。在你的Chrome应用当中，视图可以访问文档对象模型（DOM）。

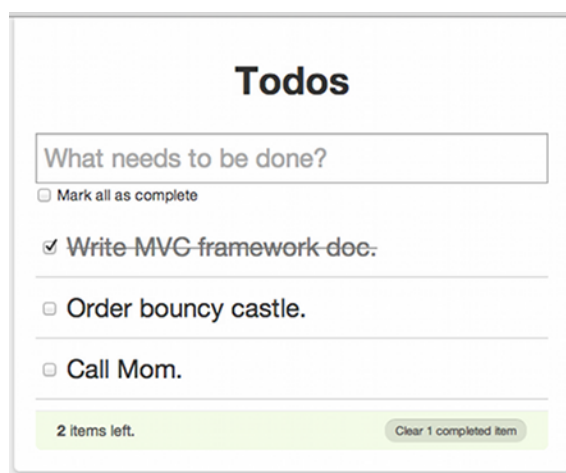


图 A-3 待办列表应用

举例来说，在上面的待办列表应用中，开发者创建了一个很不错的视图用来向用户展示待办列表项。用户可以通过一些固定的输入格式来新建一个新的待办事项，但是要注意一点：视图并不知道如何对模型进行更新操作，这是由控制器来完成任务的。

1.1.3 控制器（Controller）

控制器被用于逻辑控制和充当模型与视图之间的“胶水”。当模型发生改变时，控制器更新对应的视图。除此之外，控制器还可以给视图添加事件监听器，当用户对视图进行操作的时候来通知模型做出对应的更改。

在待办列表应用中，当用户将一条待办事项标记为完成时，这个单击事件被发送给控制器。控制器收到请求后对对应的模型做出修改，将其状态标记为完成。通常它还会与服务器进行一次保存操作来持久化保存这次修改。在一些功能强大的客户端应用（例如Chrome应用）中，有时也需要将数据保存在本地。在这种情况下，控制器也可以用类似于FileSystem API的工具将数据保存在客户端。

MVC模式有一些变种，比如MVP（模型-视图-呈现）和MVVP（模型-视图-视图模型）。甚至对于的MVC设计模式，在传统的MVC模式和不同语言的实现中也存在许许多多的变种。例如有些基于MVC的框架通过监视模型的变化来直接修改视图，而有些则是通过控制器来更新视图。这篇文章的重点不在于比较这些不同的实现方法，而是专注于关注关注点分离这项在现代web应用中被广泛应用的技术。

如果你对此有兴趣的话，我们推荐你Addy Osmani的在线书籍《Learning JavaScript Design Patterns》。

最后，我们来总结一下，MVC模式给应用开发者带来了模块化的能力以及一下特性：

- 可复用易扩展的代码。
- 将视图逻辑和操作逻辑分离。
- 允许许多开发者同时进行不同模块的开发（例如UI和核心逻辑）。
- 易于维护。

1.2 MVC持久化模式

MVC框架有许许多多不同的方法来实现持久化操作，每种都各有优点。在开发Chrome应用时，通常选择适合你和你的应用的框架。

1.2.1 在模型中完成持久化——活动记录模式

活动记录模式（ActiveRecord）流行于许许多多服务器端框架（例如Ruby on Rails）和客户端框架（例如Backbone.js和ember.js）中，在活动记录模式中，模型自己实现持久化的职责，例如通过JSON API来实现。

用模型来处理持久化操作的不同之处在于它引入了数据源和适配器函数的关注分离。数据源，模型和适配器（在某些框架中被称作代理）协同工作。数据源被用来保存所有的模型，同时它还提供了一些函数（例如创建、查询、过滤）来操作它包含的模型实例。

适配器（或者代理）收到来自数据源的请求后将其翻译成合适的操作来对与持久层进行交互（例如JSON API）。这在现代的web应用设计中是很重要的，因为你通常会使用不止一种持久层（例如远程服务器和本地浏览器数据）。在客户端数据中，Chrome应用提供了Chrome Storage API和HTML5 fileSystem API两种不同的选择。

优点：

- 使用简单，便于理解。

缺点：

- 不便于测试，持久层被绑定在模型当中。

- 同时使用使用着不同的数据源的模型是很困难的（例如同时使用文件系统、索引数据库与服务器端中的数据）。
- 复用其它应用中的模型会带来冲突，例如在两个不同的视图中贡献同一个模型，而且这两个视图想要将其储存于不同的地方时。

1.2.2 在控制器中完成持久化

在这种模式中，控制器保存了模型的一个引用以及它在数据源中对应的位置。控制器对模型生命周期中的事件作出相应（例如加载，保存，删除），然后在数据源中用对应的命令来获取和更新模型。

优点：

- 便于测试，控制器可以通过注入一个模拟的数据源对象来完成测试。
- 同一个模型可以在不同的数据源中复用（只需要建立使用对应数据源的控制器即可）。

缺点：

- 代码将会变得复杂且难以维护。

1.2.3 在应用控制器中完成持久化

在一些模式中，有一个应用控制器来监控不同MVC之间的切换操作。例如“后退”按钮将当前页面从编辑窗口（包括了许多MVC控件和格式）切换到设置窗口这个事件是由这个应用控制器来决定的。

在应用控制器模式中，应用控制器响应事件，并且改变应用的当前窗口，向数据源获取任何需要加载的模型，创建窗口中需要的视图和控制器。

优点：

- 将持久层移动到了更高的层次，便于修改。
- 底层控制器不需要关系持久层的实现，保持代码的纯净。

缺点：

- 每个页面都需要一系列重复的文件需要完成：模型、视图、控制器、应用控制器。