

**基于众包的隐式信息收集系统**

**的设计与实现**

|  |  |
| --- | --- |
| 作者姓名： | 张帅 |
| 学科专业： | 软件工程 |
| 导师姓名： | 叶勇 教授 Yuki Arase 博士 |
| 完成时间： | 二○一三年九月一日 |

硕士学位论文

中国科学技术大学

内部



**Design and Implementation of implicit information collection system based on crowdsourcing**

A dissertation for master’s degree

University of Science and Technology of China

Limited

|  |  |
| --- | --- |
| Author’s Name: | Shuai Zhang |
| Specialty: | Software Engineering |
| Supervisor: | Prof. Yong Ye P.hd. Yuki Arase |
| Finished Time: | Sep 1st, 2013 |

**基于众包的隐式信息收集系统的设计与实现　软件工程　张帅　中国科学技术大学**

中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文,是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外，论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名：\_\_\_\_\_\_\_\_\_\_\_ 签字日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

中国科学技术大学学位论文授权使用声明

作为申请学位的条件之一，学位论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权，即：学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅，可以将学位论文编入《中国学位论文全文数据库》等有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。本人提交的电子文档的内容和纸质论文的内容相一致。

保密的学位论文在解密后也遵守此规定。

□公开 □保密（\_\_\_\_年）

作者签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 导师签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

签字日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 签字日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**摘要**

Bing词典、机器翻译等跨语言的领域通常会采用机器学习的方法来进行自然语言的处理。机器学习的效果很大程度取决于所提供的训练数据的质量和数量。这些数据只能够由人工产生，因此如何获得更多的高质量的数据，就成为了困扰这些领域发展的一个重要因素。

众包是指一个公司或机构把过去由员工执行的工作任务，以自由自愿的形式，外包给非特定的(通常是大型的)大众网络的做法。

本系统以英语学习平台为载体吸引用户产生人工数据，通过众包和自动评分系统选取出高质量的数据，用于改进Bing词典、机器翻译等产品的数据质量。另一方面，本系统还可以根据需要，针对性的生成一些英语测试供本系统筛选出的高水平用户进行英语水平测试，从而针对性的收集关键数据，进而针对性的改进语料库质量。

**关键词：**众包 自然语言处理

**ABSTRACT**

Bing Dictionary, machine translation and other areas across different languages often use machine learning to approach native language processing. Machine learning highly depends on the quality and quantity of the data for training. These data is traditionally provided by manual. As a conclusion, it would be a great factor which bothering the area of native language processing that how to gather more high quality data.

Simply defined, crowdsourcing represents the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call. This can take the form of peer-production (when the job is performed collaboratively), but is also often undertaken by sole individuals. The crucial prerequisite is the use of the open call format and the large network of potential laborers.

This system is performed as a platform that helps people learning English, during which it collect data produced by users. It then filters high quality data by automatically scoring and crowdsourcing. These data could be used to improve the performance of Bing Dictionary, machine translation, etc. On the other hand, this system could generate English tests by required to focus on specified key data. These tests would be shown to high level users. As a result, we can get some high quality data to improve specified areas.

**Key Words:** Crowdsourcing, Natural Language Processing

**目录**

[第1章 绪论 1](#_Toc364253882)

[1.1 选题的背景、依据和意义 1](#_Toc364253883)

[1.2 国内外技术应用现状和发展趋势 1](#_Toc364253884)

[1.2.1 众包在国内的应用现状和发展趋势 2](#_Toc364253885)

[1.2.2 众包在国外的应用现状和发展趋势 3](#_Toc364253886)

[1.2.3 英语辅助学习软件在国内的应用现状和发展趋势 4](#_Toc364253887)

[1.3 本人的主要工作 5](#_Toc364253888)

[1.4 论文的组织 6](#_Toc364253889)

[第2章 需求分析 7](#_Toc364253890)

[2.1 功能性需求 7](#_Toc364253891)

[2.1.1 客户端子系统 7](#_Toc364253892)

[2.1.2 服务器端子系统 10](#_Toc364253893)

[2.1.3 数据处理子系统 10](#_Toc364253894)

[2.2 非功能性需求 11](#_Toc364253895)

[2.2.1 性能需求 11](#_Toc364253896)

[2.2.2 运行环境需求 12](#_Toc364253897)

[2.2.3 客户端子系统标准规定 12](#_Toc364253898)

[2.2.4 界面风格需求 12](#_Toc364253899)

[2.2.5 法律策略需求 13](#_Toc364253900)

[2.2.6 收集数据 13](#_Toc364253901)

[2.2.7 错误处理 13](#_Toc364253902)

[第3章 概要设计 14](#_Toc364253903)

[3.1 整体结构 14](#_Toc364253904)

[3.2 客户端子系统 14](#_Toc364253905)

[3.3 服务器端子系统 16](#_Toc364253906)

[3.4 数据处理子系统 16](#_Toc364253907)

[3.5 数据库 17](#_Toc364253908)

[第4章 详细设计与实现 18](#_Toc364253909)

[4.1 客户端子系统 18](#_Toc364253910)

[4.1.1 注册表模式 18](#_Toc364253911)

[4.1.2 错误处理模块 19](#_Toc364253912)

[4.1.3 数据反馈模块 21](#_Toc364253913)

[4.1.4 依赖注入模块 21](#_Toc364253914)

[4.1.5 “我的进度”用例 23](#_Toc364253915)

[4.1.6 英语测试用例 29](#_Toc364253916)

[4.1.7 测试结果用例 32](#_Toc364253917)

[4.1.8 查看答案用例 33](#_Toc364253918)

[4.2 服务器端子系统 33](#_Toc364253919)

[4.2.1 统计图表生成 34](#_Toc364253920)

[4.2.2 试题信息查询 34](#_Toc364253921)

[4.2.3 提供者模式 35](#_Toc364253922)

[4.2.4 依赖注入与Web API框架的整合 36](#_Toc364253923)

[第5章 系统测试 37](#_Toc364253924)

[5.1 功能性需求测试 37](#_Toc364253925)

[5.1.1 客户端子系统 37](#_Toc364253926)

[5.1.2 服务器端子系统 42](#_Toc364253927)

[5.2 非功能性需求测试 42](#_Toc364253928)

[5.2.1 性能需求 42](#_Toc364253929)

[5.2.2 运行环境需求 43](#_Toc364253930)

[5.2.3 客户端子系统标准规定 43](#_Toc364253931)

[5.2.4 收集数据 44](#_Toc364253932)

[第6章 关键技术难点分析与解决方案论述 45](#_Toc364253933)

[6.1 众包在本系统中的应用 45](#_Toc364253934)

[6.2 Azure云平台的特殊性 46](#_Toc364253935)

[6.3 数据收集系统的可靠性 46](#_Toc364253936)

[6.4 完形填空答题时文章颜色变化 47](#_Toc364253937)

[6.5 鼠标悬浮时使用滚轮滚动控件内容 48](#_Toc364253938)

[6.6 保证只运行一个实例 49](#_Toc364253939)

[第7章 结论 50](#_Toc364253940)

[参考文献 51](#_Toc364253941)

[致谢 53](#_Toc364253942)

# 绪论

## 选题的背景、依据和意义

微软内部有很多产品都使用机器学习[1–3]的方法来进行自然语言处理[4–8]，比如Bing词典，机器翻译[9]等。这些产品现在都面临一个问题，当收到反馈得知某一结果不准确时，如何自动的纠正这一结果。由机器学习的原理所要求的，这必须要求人工数据的介入，即由聘请的专职编辑专门处理这样的数据。这样就会有两方面的限制。一、数据量大时，可能需要雇佣较多的专职编辑来处理这些内容。二、人工处理数据的速度会受到成本的限制。因此，微软一直在寻找一种相对更加自动化的方法，来产生、收集这些人工数据。

这时，微软亚洲研究院的周明博士提出，我们可以利用“众包”的思想[10]来尝试解决这一问题[11–14]。杰夫·豪（Jeff Howe）于2006年6月在《连线》杂志的一篇文章中首次正式提出Crowdsourcing（众包）的概念，并且这样定义众包，“指一个公司或机构把过去由员工执行的工作任务，以自由自愿的形式，外包给非特定的（通常是大型的）大众网络的做法”[15] 。我们可以构建一个英语学习软件来吸引用户，同时将语料库中一些需要改进的数据拿出来，转换成问题的形式交给高水平的用户来处理，处理的结果就可以作为高质量的人工数据补充进语料库中。

本系统就是诞生于以上背景。本系统以英语学习平台为形式，应用“众包”的思想，在用户不清楚我们的根本目的是改进语料库的情况下，收集高质量的用户答案用于改进语料库质量，因此命题为“基于众包的隐式信息收集系统”。该系统的目标分为两方面。一方面，对于微软公司而言，我们的目的是通过用户的工作来改进语料库质量，这是本系统的根本目的。另一方面，对于用户而言，本系统需要能够吸引高质量的用户来使用，因此需要能够帮助用户提高英语学习的效率和英语水平。这两个目的以前者为主，后者为辅，但是两者需要兼顾，不可偏废。

本人的主要工作是设计并实现这一系统，并且达到上述两个目标。

## 国内外技术应用现状和发展趋势

由于本系统对于微软和用户具有两方面不同的意义，所以下面将从“众包”和英语辅助学习软件两方面进行讨论。对于微软而言，本系统的主要意义是，如何利用“众包”的思想，来解决收集信息，改进语料库质量的问题；对于用户而言，本系统则是一个英语辅助学习软件。

由于国外的主流语言就是英语，因此对于英语辅助学习软件的讨论仅限于国内。

### 众包在国内的应用现状和发展趋势

自20世纪80年代开始，随着互联网不断的成熟，使最初的电子公告牌一路发展到如今的博客和维基百科。当今时代称之为“信息时代”真的当之无愧，信息免费共享的时代已经结束，现如今，知识、智慧、经验、技能也具备商业价值，可以成为商品进行买卖。

众包，一个于2006年新生的词汇，是一种形式，是网络时代的产物。其理念虽然提出的较晚，但是已在国内各行各业广泛应用。[16]

威客网[17,18]、猪八戒网[19]是国内最早应用这种理念的公司。他们主张“让靠谱的需求找到靠谱的高手”，同时致力于服务交易的可靠性和有效性，从而进一步拉近服务购买者和威客之间的距离。这个思想的提出，使很多企业管理者对企业的管理进行了深入的反思。针对自身企业的弊端，引入众包思想，以拓展企业的边界，节约成本，也体现了人文文化[20,21] 。技术永远要走到前沿，才能让一个国家时刻前进。由机器翻译进行语言互译总是存在各种缺陷，针对这些缺陷，结合众包理念，将翻译工作规模协作化，平台开放化，读者译者化[11,13,14]。新闻媒体作为传播信息的载体，当然能更快地接触到这个思想，事实上，他们不仅仅对此思想产生了一定的认识，而是利用其对新闻传媒的形式进行改革。在经济效益的约束下，部分媒体对科技方面的相关传播会稍显轻视；针对科技方面的报道数量较少，同时传播形式有些单一，导致缺乏吸引群众的注意力。针对这些缺陷，引入“众包模式”使接收信息的人同时成为参与者，令科技信息的传播能更好的满足人们的需要[22]。

众包在国内应用最多的无异于图书馆管理了[23–25]，在电子产品丰富的今天，很多人基本上舍弃了很多实体形式的书籍、资料，在此情况下，很多高校图书馆的资源利用效率较低，为帮助高校图书馆早日摆脱尴尬的处境，利用“众包”的优势，制定了图书馆众包策略，例如开展宣传工作，提供在线评价平台，以及进行“众包”参考咨询服务等。

在人们的日常生活中，购物与消费必不可少。当购物时，总会出现各式各样的问题，实体店可以看到实物，但是会有价钱高、地点远的问题；而在网上购物时，虽然价钱合适，但又会有物品不合适等问题。当“众包”出现后，就建立了一种由非专业人士提供专业内容，消费者兼为内容制造者的基于互联网和社交网络，这可使购物更加快捷且享受[26]。销售之前，是产品设计。产品设计当然是由专业人士完成的，然而设计出的产品是要卖给消费者的，销量高与否一定程度上取决于产品设计。当利用网络平台吸引消费者（包括专业人员和业余人员）参与解决技术难题，设计创意时，可使设计师和爱好者们加强交流，也能设计出喜好各异的消费者提供个性产品成为了可能[27]。众包理论在旅游业中也有其用武之地。利用“网络众包”将旅游景点的营业收入增高不少[28]。

事实证明，众包的应用还没有就此而止。金融作为一个国家的主营行业之一，众包也开始在其中起到相关作用。由于新股发行制度中存在的种种缺陷，导致定义的新股发行价不够合理，当借鉴众包思路时，发行价格的制定就可以交给所有具有申购意愿的投资人，这样，在价格制定过程中，就能较充分体现价格的合理性。军事时国家机密行业之一，但是已阻挡不住众包的应用了。国防部计划将众包应用于快适应性强的新一代车辆（FANG）中。这个项目分为三个挑战部分，层次越高，难度越大[29]。

综合以上论文资料，显而易见，“众包”已进入我国的各行各业，其中还不乏命脉行业，但是其适用程度以及改进方式还需要深入挖掘。

### 众包在国外的应用现状和发展趋势

众包的概念首先由外国人发现并提出[15]，在国外发展壮大后，才渐渐传入国内，所以众包在国外的的应用更为迅速，更深入些[30]。

引入众包，收效不错的一个例子就是waze，手机地图是随着智能手机所产生的必备的应用，用户对其细节的准确性要求极高，虽然其采用卫星定位，但是对细节的处理还是不到位。而使用地图应用程序的用户遍布整个城市，整个国家，对于有些细节，他们比对卫星定位描述的更加准确，所以与其产品开发商对整个地图的每个细节做出标示，还不如用户参与进地图修正过程，这是快速、节省成本、以及适应客户需求的好方法，当然这是“众包”所带来的。事实证明，在半年内，waze的用户较之前增长了一倍，达到了两千万人[31]。

当然，众包在信息时代的应用不止waze一个。苹果的产品众人皆知，姣好的外形，加上流畅的操作系统，还有很多独特的好玩游戏应用。苹果耗资聘请应用程序开发工程师开发的很多应用程序，然而财政收入的数据却不能与其投入成正比。在当今经济形势的紧张下，投资也变得更加谨慎。但当引入“众包”，利用大众的生产力，同时给予报酬给那些出色的草根研发工程师，双赢也就是这样吧[32]。这种策略不仅只有苹果会用，百事、IBM、也都在利用“众包”对产品开发[33–38]形式进行改革，以实现更加快捷、风险更低、更加贴近用户的双赢局面。

新闻传媒也因为有了“众包”而变得与众不同。美国因将“公民新闻”转变为“专业余新闻”[39]，其中需要通过一些业余爱好者参与，当然这些爱好者都是来自于群众。韩国的ohmynews新闻是当地人称之为平民新闻的信息媒体、南非的reporter.co.za也是一样的。除了新闻类型的网站外，配音这个行业也开始与“众包”接轨。国外有一家名为VioceBunny的创业公司，创立了方便快速找到配音演员的网络平台，目前活跃于该网站上的配音演员超过10万，覆盖的语言也已经超过了50种[40]。

智能机器人是永久不变的话题。机器人学家以往总需竭尽全力才能让机器人完成高度专业化的任务，然而这个工作其实可以不用这么艰难的进行。引入“众包”，让互联网上对此有兴趣的人去引导机器人做含有复杂动作的实验，集结了群众思维的力量，一切将变得非常顺利。

当然，还有很多“众包”产品，既有像Amazon Mechanical Turk[41]这样的外包任务平台，也有像Doodle4Google这样直接为企业产品线提供内容的外包应用，甚至有Duolingo这样打着外包旗号让人们参与进来提供翻译素材的网站。随着人们不断意识到众包的强大作用，以及发现众包相对于外包的廉价特性，将会有更多的企业愿意在众包上投入一定的资源进行尝试，进而促进众包的发展和流行。

### 英语辅助学习软件在国内的应用现状和发展趋势

英语辅助学习软件在国内已经发展了相当长的一段时间了，但是英语辅助学习软件的形式还非常有限，总体来看有以下几种[42,43]：

1. 教材的配套光盘：主要有①大学英语；②走遍美国；等等。
2. 工具型软件：①金山词霸；②英华金典；等等。
3. 考试辅助类软件：①新东方四级词汇；②新东方考研英语听力；③新东方考研英语阅读；等等。
4. 英语专项技能训练类软件：①洪恩系列；②随心所欲说英语系列；等等。

单词记忆类软件软件在市场中占有极大的比重，主要是人们通常认为记忆单词比较繁琐和困难。单词记忆类软件通常都是以艾宾浩斯（Hermann Ebbinghaus）遗忘曲线[44]的原理为基础，通过重复展现的方式加强记忆。

工具类软件虽然市场占有量非常大，但是用户通常是在使用其他方式学习英语的过程中，临时借助工具型软件的帮助，其使用的时间非常离散，并且每次使用的时间都很短。

考试辅助类软件的市场占有率反而比较低，因为其经常被第四类软件所涵盖。另一方面，一旦考试结束或者通过，这类软件的用户就不再使用这类软件。

后两种类型的英语辅助学习软件通常具有非常低廉的价格和非常广泛的内容。例如词汇类的软件，不仅有从小学一直到高中的所有英语词汇，还有大学英语四级，大学英语六级，GRE英语，甚至还包括了专业相关的英语，比如说计算机专业词汇，对外经济贸易专业词汇等等。资源丰富和相对低廉的价格，是吸引用户使用这类软件的主要因素。

## 本人的主要工作

本系统的筹备、开发、测试、运行和维护等一系列工作，由一个工作组共同完成。本人的主要工作是本系统全部的设计和开发。本人还参与了小部分测试工作，以及全部的运行、维护和部署工作。本人还负责这一系列过程中所必须的工具和脚本的开发。

概要设计环节与需求分析和最终部署都有一定的联系。在进行需求分析的时候，需要概要设计阐述清楚，哪些事情可以做，那些事情不能做或者代价比较大，从而能够为本系统界定明确的需求范围。而预期使用什么样的技术，以什么样的结构去实现，都将影响最终的部署环节。在概要设计环节应该尽可能的不对部署环境做出要求或假设，但是无论如何，概要设计都会对最终部署和运行维护产生影响。

本系统分为三个主要部分：客户端子系统、服务器端子系统和数据处理子系统。其中，数据处理子系统是一个离线系统，只在需要的时候运行。数据处理子系统需要联通很多外部系统，同时，数据处理的流程中，还有一些部分需要人工介入，从而使得整个流程的实现比较复杂。服务器端子系统的最终部署环境是Azure云平台，因此其开发必须使用Azure云平台提供的接口，而不能随心所欲的使用.NET Framework提供的功能。Azure云平台出现的比较晚，还处于高速发展期，变化比较大，文档比较少，配套的工具也不齐全，为设计和开发带来了一定的难度。客户端子系统由于和用户直接交互，更是整个系统的重中之重。一方面，客户端子系统需要分发到客户端，由于中国的互联网环境普遍不好，网络速度普遍比较慢，所以就限制了客户端子系统安装包的大小，进而限制了客户端子系统的运行时环境，使得很多新的功能都不能使用。另一方面，用户的系统环境非常复杂，操作系统从Windows XP、Windows Vista到Windows 7甚至于Windows 8都有，这就给客户端子系统的设计和开发带来了挑战。最后，我们还要顾及用户体验，只有好用的软件用户才会多，为此，我们必须从细节处下手，提供一些WinForm框架没有提供的功能，有时甚至要突破WinForm框架的限制。

服务器端子系统的测试有大量的经验和例子可以遵循，这都是因为无论其依托于什么平台，一旦部署好之后，就只是一个Web服务，我们只需要对其接口进行测试即可，而无需关注其具体是怎么实现的，其平台又具有什么样的特性。但是客户端子系统则复杂得多，用户系统环境的复杂性使得对客户端子系统的测试也需要在多种环境下一一进行测试。另一方面，对界面进行自动化测试本身就是一个困难的事情。而一些隐蔽的情况，则很难通过机器判断，必须手工测试和验证才可以。

Azure云平台给服务器端子系统的部署和维护工作也带来了一些困难。由于Azure云平台是一个分布式系统，使得原本简单的部署操作变得复杂而又危险。另一方面，维护工具的缺乏，又给维护工作带来了一些意想不到的麻烦。

## 论文的组织

本文主要叙述基于众包的隐式信息收集系统的设计与实现。第1章主要介绍选题的背景、依据和意义。第2章主要阐述系统的需求，明确了系统要做的工作以及要达到什么样的目标。对于系统需求的讨论，又分成了功能性需求和非功能性需求两个方面，其中前者是问题的主要矛盾，后者是次要矛盾，但是也不能够轻视非功能性需求。第3章主要叙述系统的概要设计，从较高的层次介绍了整个系统的各个部分，并且阐明了这些部分之间的联系和作用。第4章详细叙述了整个系统是如何实现的，深入的讲解了系统的各个模块是通过什么样的手段实现的。在这个过程中，还带出了一些第3章中没有讲到的部分，因为这些模块只有在具体实现的时候，才发挥了作用，并且正是在实现的过程中，才对其加以考虑和解决的。第5章叙述了对系统测试什么，以及如何对系统进行测试，呼应了第2章中提出的系统需求。系统测试是保证系统质量符合要求的重要手段，一方面，系统测试必须符合系统需求；另一方面，系统测试在一定程度上比系统需求更加细致，要求更高。有时候，有些关键点很难进行测试，需要修改代码进行配合，本章详细的论述了如何对本系统进行测试。第6章回顾了本系统在设计、开发、测试等一系列环节中的关键点和技术问题，这些问题可能分布在系统的各个环节，甚至跨越了模块，跨越了子系统边界。本章将这些无法在前几章中讨论清楚的问题，集中起来加以讨论。

# 需求分析

开发任意一款软件时，我们都应该考虑功能性需求和非功能性需求两个方面，这样才能够全面的认识到对这款软件的需求。大部分软件的非功能性需求都比较宽松，以至于我们很少特别的去考虑非功能性需求。微软对于任何一款需要发布的产品都有着较为严格的规定，这些规定构成了非功能性需求的大部分。除此之外，我们还应该考虑到程序的性能需求，界面风格，数据采集，错误处理等多个方面。

## 功能性需求

因为功能性需求是能够被用户直观感受到的，我们首先来考虑对用户更为重要的功能性需求。根据需要，我们将整个系统划分为三个子系统：客户端子系统、服务器端子系统和数据处理子系统。其中客户端子系统依赖于服务器端子系统的支持，而服务器端子系统的数据又来源于数据处理子系统。

### 客户端子系统

首先我们来看与用户进行直接交互的客户端子系统。

首先我们来明确一下这个系统对于用户的意义，以及目标用户的定位。本系统以英语学习平台为依托吸引用户使用，因此必须能够帮助用户提高英语学习的效率和英语水平。另一方面，我们要达到改进语料库质量的目的，更为倚重的是高英语水平的用户。综上所述，我们需要能够帮助高英语水平的用户提高英语学习的效率和英语水平。为了更好的收集数据，我们需要提供更多的让用户提供信息的机会，因此本系统设计为一个英语测试练习软件。

一个英语测试练习软件，主要可以在两个维度上进行配置：考试项目和考试难度。由于开发成本的限制，我们将只提供大学英语四级、大学英语六级和研究生英语入学考试三个难度的英语测试；对于考试项目，我们将只提供完形填空、阅读理解和翻译三种。对于任意一次英语测试，应该至少包含以下三个场景：测试、测试结果、查看答案。并且每次测试的时候，根据用户的表现，相应的调整下一批试题的难度，使得用户不会因为做到过于简单的试题而觉得无趣，又或是因为做到过于困难的试题而失去信心。在用户完成测试后，应该将测试的结果记录下来，以供用户了解自己的学习进度。从微软的角度而看，在用户进行测试的时候，应该不断的将用户的答案收集起来。因此，客户端子系统的用例图如图 2‑1所示。



图 2‑1 客户端子系统用例图

接下来，我们来考虑每一个用例。

用户学习进度中，我们可以通过标签页的形式来展现不同难度的考试，使用下拉选择框来展现不同的项目。对于每一个考试，根据情况又可以分为三种状态：用户未进行任何项目的测试，用户已经进行过某些项目的测试，用户所有项目都测试过。对于用户未进行过任何测试的情况，我们可以直接把三种项目的测试都展现出来，让用户直接选择，如图 2‑2所示。对于用户已经进行过某些项目的测试的情况，我们又分为两种情况：用户查看已经进行过的项目，用户查看未进行过的项目。对于前者，我们可以展现和用户所有项目都测试过的情况相同的界面，如图 2‑3所示。对于后者，我们还需要特殊处理，如图 2‑4所示。

用户在界面上点击进行测试按钮后，可以跳转到测试页面。根据所进行的项目的不同，测试页面也有不同的调整。进行测试时，如果用户没有进行过测试，则使用默认难度对用户进行测试，否则使用用户最后一次测试的结果来确定试题难度。用户在测试的过程中，我们需要不断地收集用户的答案。每一个测试又分为三页，每一页测试结束后将用户答案上传到服务器进行评分，用该评分来确定下一页试题的难度。测试的过程中，还需要显示时间倒计时，每篇文章的时限为考试难度的所带来的基准时间加上文章长度的修正时间。如果用户在时限内没有完成测试，则自动进入下一篇测试。用户在测试过程中可以随时放弃测试，回到“我的进度”界面。用户再没有答完当前页面的情况下，不能进入下一篇测试，但是超时的情况除外。

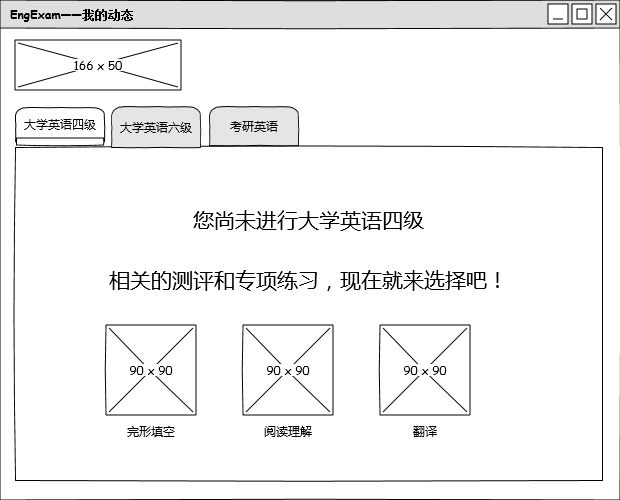


图 2‑2 未进行任何测试项目时的界面示意图

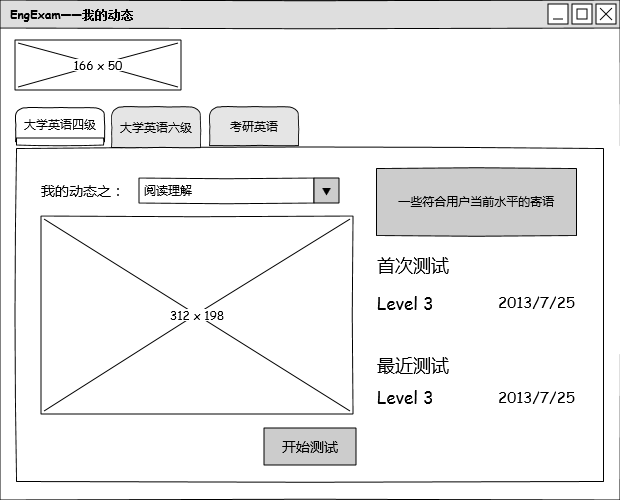


图 2‑3 已经进行过的项目界面示意图



图 2‑4 只完成过部分项目的界面示意图

三页测试完成后，自动展现测试结果页面，同时将用户的表现存入历史记录中。测试结果页面主要由两张统计图表构成：正确率统计图表和弱项分析统计图表。测试结果页还综合当前试题的难度和用户答题的表现，得出一个综合得分，显示为用户在当前测试中的等级。

用户在测试结果页面可以通过点击“查看答案”按钮来到查看答案页面。在这里，不仅会显示当前这次测试考的三页题目，还会显示出用户的答案，以及答案是否正确。对于用户答错的题目，还会给出标准答案。

### 服务器端子系统

客户端子系统的一些功能，是依赖于服务器端子系统提供的服务而运行的。测试功能需要服务器端子系统提供试题数据，查看答案功能也需要服务器端子系统提供试题答案数据，测试结果功能需要服务器端提供统计图表生成功能，信息收集功能也需要服务器端的支持。

服务器端子系统的用例图如图 2‑5所示。

### 数据处理子系统

接下来我们需要考虑数据的来源，即题目从何而来这一问题。从我们的目标来看，我们需要生成一些试题数据。但是在此之前，我们需要一些已有的数据用于区分高水平用户。从吸引用户的角度来考虑，我们需要一些尽可能对用户而言高质量的数据，即大学英语四级、六级以及研究生入学考试的真题，这些题目具有对用户而言最高的质量。除此之外，我们还可以找到一些为这些考试设计的练习题。对于这些题目，我们都能够清楚它们属于哪个考试，难度如何。但是对于我们自己生成的题目，我们却不知道这样的信息。因此，我们还需要一个难度识别的系统。



图 2‑5 服务器端子系统用例图

数据处理子系统的用例图如图 2‑6所示。

由于数据处理子系统的主要功能实现依赖于外部系统，信息提取功能由于涉及微软内部机密，因此数据处理子系统的实现将不予讨论。

## 非功能性需求

非功能性需求相较于功能性需求通常不容易被发现、被重视，但是毫无疑问，非功能性需求也是必须考虑的重要部分。本系统的非功能性需求主要由以下几方面构成：一般性非功能性需求、微软对于产品的强制规定、特殊性非功能性需求。

### 性能需求

要求产品在网络条件良好的情况下，响应时间不超过2秒。要求客户端子系统在接收到服务器端子系统返回的数据后，响应时间不超过0.3秒。

要求服务器端子系统能够至少承载每秒100次请求，并且每次请求的响应时间不超过0.7秒。

要求客户端子系统占用内存不超过40Mb。

要求客户端子系统占用的磁盘空间不超过5Mb，客户端子系统的安装包的大小不超过3Mb（不含.NET Framework）。



图 2‑6 数据处理子系统用例图

### 运行环境需求

要求客户端子系统能够支持以下操作系统的简体中文版本：

* Windows XP SP3
* Windows Vista
* Windows 7

要求客户端子系统运行在.NET Framework 2.0下。

### 客户端子系统标准规定

要求客户端子系统不得要求管理员运行权限，不得进行需要管理员运行权限才能正常运行的操作，比如写当前程序目录，读写系统注册表中的一些特定项。

要求客户端子系统能够重入，但是保证只启动一个实例，即在已经启动本产品后，再次运行本产品，不产生新的实例，并且将当前运行的实例激活。

### 界面风格需求

要求客户端子系统界面风格符合Metro风格。

要求客户端子系统界面英文字体使用Segoe UI，中文字体使用微软雅黑。

### 法律策略需求

要求客户端子系统在显示收集自网络的大段（超过11个字符）文本时，必须同时显示其原始链接，并且用户可以通过点击该原始链接打开该原始页。

要求客户端子系统不得使用除JSON.NET软件包之外的第三方软件包。

### 收集数据

要求产品必须收集以下信息：

* 用户的完整答案和题号，同时必须附上时间戳和客户端子系统实体标识。
* 用户总量和活动用户总量。
* 用户测试完成度。
* 用户在每个页面滞留的时间。
* 所有出现错误的情况时的数据。

以上各项只需能够从收集到的数据中计算得出即可，无需严格按照需求进行数据收集。

### 错误处理

要求服务器端子系统在出现任何错误的时候，终止当前操作，记录错误信息。

要求客户端子系统必须能够捕捉所有的错误，弹出对话框告知用户，然后记录错误信息并将其发送到服务器端子系统。

# 概要设计

有了之前的需求分析，我们就可以初步的确定一些产品的情况，从而进行一些初步的设计。

## 整体结构

由于需求限制了客户端子系统所占用的磁盘空间，并且出于保密的考虑，我们的客户端子系统将不携带试题数据，而是在运行时从服务器端子系统动态取得数据。服务器端子系统的试题数据也必须有一个来源，这就是数据处理子系统。这三个子系统构成了整个系统，三者的关系如图 3‑1所示。这三个子系统均使用C#语言实现。



图 3‑1 子系统整体关系图

图 3‑1中的箭头表示依赖关系。客户端子系统和服务器端子系统通过Internet进行交互。数据生成子系统从外部系统（如机器翻译，必应Bing词典等产品）中得到需要改进的数据，然后针对性的生成试题。数据生成子系统和服务器端子系统通过共享同一数据库来完成信息的交换。数据完整性由数据库保证。同样的，数据分析也依赖于服务器端子系统的数据库。

## 客户端子系统

客户端子系统中，所有的项目都在两个维度上进行了扩展：考试类型和测试项目。考试类型分为大学英语四级、大学英语六级和考研英语。测试项目分为完形填空、阅读理解和翻译。对于不同的考试类型，其界面没有显著变化，只是展示的数据发生变化。对于不同的测试项目，有些界面变化不大，有些界面则需要进行调整，但是无论如何，整体的测试流程并没有发生变化。这种情况特别适合使用MVC架构（Model-View-Controller Architecture）[45]。

本产品的测试环境与最终的部署环境不同，与开发环境也不同，因此在连接服务器端子系统时，就需要准备三套不同的模块，并且能够根据需要在这三者之间切换。这种情况特别适合使用Provider模式，因为Provider模式可以向调用者隐藏这些细节，使得调用者无需处理这种复杂性。而为了能够通过配置文件来控制客户端子系统在这些模块间切换，还需要配合DI模式（Dependency Injection Pattern）[46,47]。调用者通过通用的接口持有Provider实例，这些实例在调用者实例被创建时就已经确定，透过构造函数或公开属性传递给调用者。调用者无需关心该Provider实例到底是什么类型的。这样有利于程序的扩展和复杂性的封装。

客户端子系统的系统架构图如图 3‑2所示。



图 3‑2 客户端子系统系统架构图

按照MVC模式和Provider模式的一般要求，视图不应该越过控制器直接与提供者进行交互，但是有些视图功能十分简单，不需要控制器在模型和视图之间进行协调，因此对于一些页面并没有实现与之配套的控制器。

根据需求，数据收集需要在每一页的进入时刻进行，而其他数据都可以根据此时收集的数据进行计算而得出。因此，只有视图层依赖于数据反馈模块。

日志模块有助于内部测试时发现和跟踪错误，因此日志模块也是横跨视图、控制器、提供者等多个模块的。

错误处理模块要求在程序的任意位置出错时都能接管程序。错误处理模块需要.NET Framework的支持，这样才能够保证能够捕捉到所有的异常。

注册表模块不依赖于系统注册表，而是起到了一个集中配置的作用。注册表模块得名于注册表模式（Register Pattern）。程序可以在任意位置访问注册表模块，查看一些配置信息。这些信息有些是在编译时期根据开发环境、测试环境和最终部署环境特别设置的，有些是在程序运行时动态加载的。使用注册表模式，可以对程序的其他部分隐藏这一细节。

## 服务器端子系统

服务器端子系统通过HTML（HyperText Markup Language，超文本标记语言）和JSON（JavaScript Object Notation）格式与客户端子系统进行交互，通过共享数据库的方式与数据处理子系统进行交互。

由于不需要用户登录，而是采用客户端子系统唯一标识来确认用户，并且传输过程中也不需要认证和加密，服务器端子系统可以被设计为一个无状态的服务。一个无状态的服务有利于性能的提升和扩展。

服务器端子系统采用MVC架构。由于不需要向用户展现数据，因此不需要视图层。但是服务器端子系统仍然是符合MVC架构的，因为一个从模型到JSON格式的转换器起到了视图的作用，而模型层和控制器仍然是存在的，并且按照MVC架构的设计进行协调。服务器端子系统也使用Provider模式来适配不同的环境所带来的影响。

服务器端子系统的架构图如图 3‑3所示。



图 3‑3 服务器端子系统架构图

服务器端子系统的控制器直接接受客户端子系统发出的请求，然后向提供者索要数据。提供者从数据库中取得恰当的数据后，将数据返回给控制器，控制器使用格式化方法将模型转换为JSON格式返回给客户端。

同样的，服务器端子系统也使用了DI模式。并且由于在服务器端子系统发生的异常不会将异常的详细信息传播到客户端子系统，还需要在服务器端子系统添加日志模块，用于记录这些数据。

## 数据处理子系统

由于之前在需求分析中提到的原因，数据处理子系统将不再详细进行叙述。在本产品中，数据生成和数据分析都是离线进行的，并不需要十分复杂的结构。数据生成又分为两个方面。一方面，我们从Internet中搜寻相关的考试真题和练习题，补充进我们的题库中。同时，我们从这些已有的数据中，提取出难度特征，留待以后我们自行生成试题的时候确定难度。另一方面，我们从已有的产品中得到一些需要改进语料库信息的反馈，根据这些反馈信息，通过外部系统——试题生成子系统，生成新的试题，补充进题库中。这些试题，会发放给评分较高的用户，收集到的答案就可以反过来用于改进语料库的质量。

## 数据库

服务器端子系统和数据处理子系统通过共享数据库的方式进行交互，因此我们必须明确数据库这一接口。数据库中要包含试题信息和数据收集信息。由于数据库最终要部署到Azure Storage平台上，而Azure Storage平台属于NOSQL（非关系型数据库），因此要求数据库要尽可能的平坦。

数据库中试题部分的设计如图 3‑4所示。



图 3‑4 数据库试题部分IDEF1X图

数据库的信息采集部分，考虑到扩展性的要求，则采用JSON格式将数据存储到Value域，有选择的破坏了关系型数据库设计的第一范式。

# 详细设计与实现

本章详细介绍了如何设计和实现基于众包的隐式信息收集系统。其中，客户端子系统是整个系统最重要，最复杂的部分。服务器端子系统依托于Azure云平台，而Azure云平台由于发展较快，并且出现的比较晚，资料比较少，从而为服务器端子系统的设计与实现带来了一些额外的困难。数据处理子系统主要依托于外部系统，并且大部分涉密，故不予讨论。

## 客户端子系统

客户端子系统由于直面用户，并且用户的运行环境比较复杂，故受到的限制最大，从而实际上成为了本系统最重要，也是最复杂的部分。

### 注册表模式

注册表是“一个众所周知的对象，其他对象可以通过该对象找到公共的对象和服务”[48]。

客户端子系统对注册表模式进行了一定的扩展，使得可以在编译时刻根据所处的环境是开发环境、测试环境和最终部署环境来配置注册表项。客户端子系统中的注册表的类图如图 4‑1所示。Register是一个静态类，其中的所有方法均为静态方法，并且都是C#中的getter方法，使得其在使用的时候，语法就像是在使用一个field一样。但是其实际上又是一个方法，因此在实际操作中，可以在需要的地方加锁，以使其线程安全，或者是只提供getter方法，使其变得只读。更重要的是，像是图 4‑1所示的这样，在静态类Register中持有一个静态实例，使得一些属性可以从这一静态实例中获得，起到一种可以静态/动态配置注册表的效果。

图 4‑1中的RegisterBase接口扩展了三个类：FakeRegister、InternalRegister和RealRegister。这三个类分别用于测试环境，内部发布环境和最终部署环境。在FakeRegister中，指定了Providers均使用Fake版本的，这些Providers只能提供预先设定好的数量非常有限的内存数据，可以保证Providers的正确性，用于开发和测试时，检验其它组件的正确性。InternalRegister使用Internal版本的Providers，这些Providers将服务桩导向部署于公司内部Lab服务器的服务器端子系统，用于内部测试和发布。RealRegister用于最终部署版本，AppId由安装文件生成并写入注册表，Providers使用导向部署于Azure云平台的服务器端子系统，保证运行稳定性和效率。

在需要的时候，还可以将更多地属性从硬编码于Register静态类中，提取到RegisterBase接口中，应用各种不同的配置。



图 4‑1 注册表类图

### 错误处理模块

错误处理模块的活动图如图 4‑2所示。



图 4‑2 错误处理模块活动图

客户端子系统中的错误处理模块依赖于WinForm框架提供的全局错误处理回调事件。

需要注意的是一旦注册了全局错误处理函数，在调试的时候，由于所有的异常都已经被处理了，这些异常就不能够被调试器所捕获，从而难以调试。因此，在注册全局错误处理函数的时候，应该判断一下当前环境是否存在调试器，如果存在的话，就不进行全局错误处理。

另一方面，WinForm框架提供的全局错误处理回调事件只能处理在UI线程发生的未处理的异常，对于在其他线程或AppDomain（应用程序域）发生的异常则无能为力。因此，除了要注册WinForm框架提供的全局错误处理回调事件，我们还应该注册全局线程错误处理回调事件和全局AppDomain错误处理回调事件。

除此之外，还应当注意到，当服务器端子系统发生错误时，或者网络连接异常时，都会产生WebException异常。我们应该区别对待这两种情况，尤其是在提示用户的时候，后者应该提示用户检查网络连接。我们可以通过WebException的Status属性来区分这两种错误。

### 数据反馈模块

由于数据反馈模块的调用会分布到程序的每一处地方，因此必须要考虑线程安全性。

由概要设计中的图 3‑2知，只有视图层会调用数据反馈模块。而在视图层中，几乎所有的代码都运行于UI线程。只有使用了BackgroundWorker组件和Timer组件的窗体，才有可能引发线程安全问题。因此我们只需要加强BackgroundWorker组件和Timer组件的线程安全原则，把数据反馈模块也当做UI线程中的一部分，就可以保证线程安全性。

另一方面，数据反馈模块的加载需要较长的时间，这有两部分的原因。数据反馈模块在加载的过程中需要处理之前未处理完的数据；数据反馈模块在加载过程中需要访问网络，其中确定系统网络代理的过程需要相对较多的时间。为了提高用户体验，不影响主窗体的加载，我们使用后台线程加载数据反馈模块。这时就产生了新的线程安全性问题：在数据反馈模块加载完成之前，就有可能需要调用数据反馈模块。此时应该使调用线程挂起，等待数据反馈模块加载完毕后，再继续运行。为了保证这一点，我们可以通过改进单例模式（Singleton Pattern），实现一个比较精巧的结构来达成这一目的。

首先使用创建者模式（Builder Pattern）将数据加载等复杂的创建过程从数据反馈模块的构造函数中分离出来。然后将单例模式的创建过程改为调用线程池创建后台线程进行创建，创建并赋值结束后，调用ManualResetEvent的Set方法通知正在等待的线程可以继续执行。使用静态公开属性暴露单例实例，在获取该单例实例的时候，调用ManualResetEvent的WaitOne方法验证该实例是否创建完成。简化后的示例代码如图 4‑3所示。

### 依赖注入模块

依赖注入是一种可以去除硬编码的依赖性，允许在运行时期或者编译时期改变它们的设计模式[46,47,49]。

应用依赖注入能够降低客户端子系统的各个模块之间的耦合度，模块之间将只依赖于接口，而不再依赖于具体的服务。这对于客户端子系统的意义比较大，因为客户端子系统具有好几套模块，需要根据部署环境的不同（开发环境、测试环境、最终部署环境）进行配置。如果不使用依赖注入，则每个模块都需要自行创建其所需要的服务，从而必须清楚当前所有的服务模块，并且需要硬编码其创建过程。而每一服务模块的创建过程都有可能不同，从而导致整个系统十分复杂[49]。

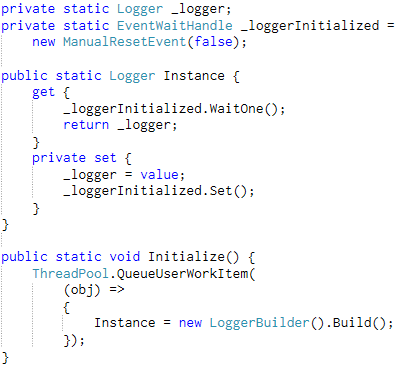


图 4‑3 数据反馈模块的加载示例代码

现在.NET平台上有很多成熟的依赖注入框架，比如Unity，Spring.NET，Castle等。但是这些框架的最新版本都不再支持.NET Framework 2.0，并且由于微软的法律策略需求，我们也不能够使用这些第三方框架。因此也就需要我们手工实现一个依赖注入模块。好在我们不需要这些依赖注入框架带来的强大功能，仅需要实现最简单最朴实的依赖注入功能即可。

人为规定只考虑基于构造函数的依赖注入功能可以极大地简化这一问题。我们创建一个容器对象，这一容器知道所有依赖关系以及对象的创建方法。由这一容器对象创建的实例，将满足其构造函数所需要的一切依赖关系。这一功能的大致实现方法如下。首先建立一个词典，该词典记录着如何创建一个类型的实例，并以其类型为索引。然后提供一个注册方法，可供编译期间通过代码配置创建方法和依赖关系。最后提供一个创建方法，可供运行期间动态创建所需类型的实例。对于不在词典中的类型，则使用反射（Reflector）的方法调用其构造函数进行创建。如果存在多个重载的构造函数，则优先使用有参数的构造函数。对于这些参数，也使用这一创建方法进行创建。特别的，这些类型可以是某一接口，但是其创建方法指定创建某一个实现了该接口的类的实例。

除此之外，我们还可以指定创建的实例，生存周期是什么样的。是每次请求的时候都创建一个新的实例，还是重用已有的实例？

使用依赖注入来控制对象的生命周期，优于使用单例模式创建长生命周期的对象。第一，使用依赖注入来控制对象的生命周期，可以通过代码 、配置文件进行灵活配置，在需要的时候可以转换为其他形式，例如每个线程一个实例。而单例模式必须大量修改代码才能做到这一点，并且如果要将多个类型应用单例模式，则需要对每一个这样的类型进行修改，以实现单例模式。使用依赖注入，则只需要在配置文件中声明就可以，同样的功能可以轻易地在多个类型上应用，而无需较大的代价。第二，使用依赖注入来控制对象的生命周期，不容易遭到程序员的误用。由于单例模式事实上创建了一个全局的对象，从而一定程度上破坏了面向对象的原则。一旦程序员依赖于单例模式创建的全局对象，则该全局对象将与程序的各个角落产生强耦合，不利于代码的进一步演变。而使用依赖注入来控制对象的生命周期，所有的对象都一视同仁，不会有这种后顾之忧。

### “我的进度”用例

客户端子系统启动时，首先进入“我的进度”用例。“我的进度”用例相关的静态图如图 4‑4所示。

我的进度这一界面通过标签页，分为大学英语四级、大学英语六级和考研英语这三门考试；通过下拉列表，分为完形填空、阅读理解和翻译三个考试项目。选择了考试类型和考试项目后，就会显示具体的数据。左侧显示最近一次考试的弱项分析统计图表，右侧显示首次测试和最后一次测试的成绩对比。如图 4‑5所示。

客户端子系统在进入“我的进度”界面时，首先会创建三个空的标签页，然后通知Controller分别根据历史数据Provider提供的数据构建恰当的内容来填充这三个标签页，并且还要注册一些回调事件，以响应鼠标点击等操作。

此时根据历史数据的状态，我们就有就有三种不同的情况需要给用户展示不同的界面。一种情况是，用户没有进行过这门考试的任何一个项目，对于这种情况， 只需在一个标签页中为用户显示所有的项目，任其选择即可，如图 4‑6所示。第二种情况是，用户完成了这门考试的某些项目，但是当前显示的考试项目没有进行过。对于这种情况，不仅需要为其提示当前考试项目没有进行过，随时可以开始进行测试，还需要在下拉列表中显示其他的考试项目，以供用户进行选择，如图 4‑7所示。第三种情况是，用户完成了这门考试，并且当前显示的考试项目也完成过。对于这种情况，我们应该显示完全的数据，包括左边的弱点分析统计图表，可供用户切换考试项目的下拉列表，右侧的鼓励话语，以及首次测试成绩和最后一次测试的成绩，和开始测试按钮。



图 4‑4 我的进度用例相关的静态图



图 4‑5 “我的进度”界面

这三个界面都有一定的共同点和不同点，同样的项目，在点击“开始测评”按钮或者是在无数据的界面点击该项目的按钮时，所做的动作是相同的，均为开始这门考试的这项测试。在有数据的情况下，切换下拉列表中的项目，都应该切换当前所显示的内容到新的项目所需要显示的内容。需要特别注意的是，新的项目所需要显示的内容可能跟当前项目所显示的内容完全不同，例如当前界面是已经测试过的项目，而新的项目界面是没有进行过测试的项目。并且，在用户点击“开始测评”按钮后，从测评界面返回到“我的进度”界面时，也有一些相同的动作。如果用户完成了这门考试的这个项目的测试，则回到“我的进度”界面时，所有的情况均需要跳转到有数据的内容页，并且刷新其中的数据。如果用户没有完成这一测试就返回了“我的进度”界面，则还需要显示与其原来相同的界面。

这些共同点使得我们在编写代码创建这些标签页时，可以将这些共同点抽象出来，编写通用的代码。这样做有很多好处。首先，通用的代码可以适应更多的情况，以后在添加了更为复杂的逻辑后有可能带来更多的状态和公共逻辑，我们可以在现有代码上比较容易的添加简短的代码来适应这些新的变化。其次，当我们修改这些逻辑的时候，只需要修改一处而不需要每门考试每个考试项目每个状态都进行修改，并且能够保证这些逻辑的一致性。第三，短而集中的逻辑容易进行测试和发现错误。最后，如果一个新的程序员需要维护这一项目，较短且集中的逻辑可以比较容易，用比较少的时间理解。但是将这些相同之处抽象出来，使用通用的代码来完成，需要更高的抽象能力和编码能力。



图 4‑6 “我的动态”界面无数据情况

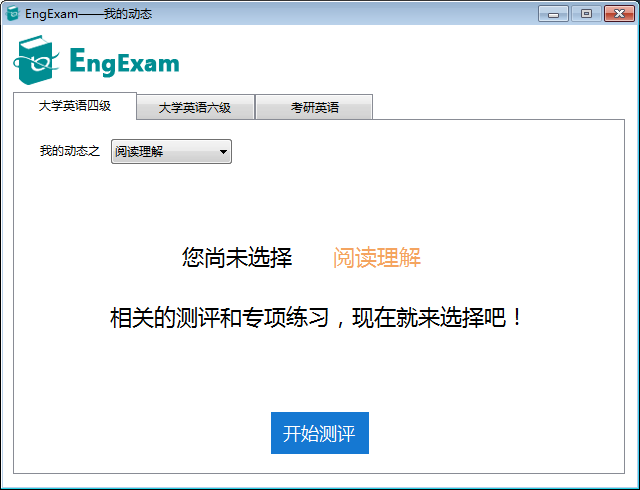


图 4‑7 “我的动态”界面部分无数据情况

“我的进度”界面，载入时期的时序图如图 4‑8所示。



图 4‑8 “我的进度”载入时期时序图

用户在进行测试的期间，需要维持许多状态。所有考试项目通用的有，这次测试的唯一标识，考试类型，考试项目，每页测试的题目、用户答案、等级、所花费的时间、正确率，等等。由于不同考试项目的题目的类型不同，因此保存题目的数据类型也不同。同理，保存不同考试项目的答案所需的数据类型也不同。甚至于最后显示统计图表所需要统计的数据，格式也不相同。这些都使得整个程序的复杂性大大增加。由于这些相同的、不同的数据要在不同的对象之间进行传递、计算、变化，更好的方法是使用一个会话对象来封装这些状态，而将不变的部分硬编码在窗体对象中。

一个通用的会话状态，其类型是无法确定的，因为在不同的地方需要不同的类型来存储所需要的数据。但是对于客户端子系统这一特定的程序，其类型是有限的。一个强类型的对象可以使程序员少犯很多错误，并且编译器也可以检查出来很多错误，而不必等到运行时刻或者是测试时才能发现。但是强类型系统很难应对丰富的变化。.NET Framework 4.0中引入了DLR（Dynamic Language Runtime），用于在强类型系统中引入动态类型，使得程序员可以在需要的时候使用弱类型。但是受限于运行环境需求，我们只能使用.NET Framework 2.0，因此必须在强类型系统解决这一问题，使得这一问题大为复杂。更为麻烦的是，我们在开始测试时，需要根据不同的考试类型和考试项目，创建不同的会话类实例。用户在完成测试后，我们还需要根据这一会话实例中的数据，产生历史数据信息，并使用历史数据Provider保存并更新历史数据。

会话类的类图如图 4‑9所示。



图 4‑9 会话类的类图

### 英语测试用例

用户点击“开始测评”后，将进入英语测试界面。英语测试界面，将是用户停留时间最长的界面，因此也是最重要的界面。用户在进入英语测试界面后，“我的进度”界面将被隐藏，并且在窗体管理器中进行记录，以便在用户在已经打开客户端子系统的情况下，再次运行客户端子系统时，激活当前窗口。

英语测试用例的状态图如图 4‑10所示。

英语测试界面由几部分构成。顶部显示当前考试的名称和难度，以及倒计时显示器。下方显示题目来源，“放弃测试”按钮，“下一篇”按钮，和当前进度显示器。中央部分的左侧显示题目，右侧为答题区域。

由于试题数据都是从服务器端子系统获得到的，因此必须考虑网络传输的延迟。根据性能需求的要求，从服务器端子系统获得数据最大可能会有1.7秒的延迟，在这期间，如果不使用异步消息处理的技术，那么主界面将无法对用户的任何操作做出相应，而是等待网络连接返回的信息读取完毕后，才能够做出处理。这么长的时间不能够响应用户的操作，会让用户产生一种客户端子系统已经停止运行的错觉。因此，一方面客户端子系统必须能够提示用户，其正在加载数据；另一方面，客户端子系统必须通过多线程技术异步加载数据，同时能够正确处理用户操作。

数据加载时，禁用界面上的可点击元素“下一篇”按钮，所有数据绑定项显示内容清空，倒计时显示器显示零值，同时在界面最中央显示加载进度指示器，如图 4‑11所示。加载进度指示器会不停转动，直到数据加载完毕。使用这种“跑马灯”效果的进度指示器，是因为数据加载的过程只需要进行一次网络连接，就可以获得所有的数据，而相对于其他操作的用时，网络连接耗时又占了非常主要的部分，因此事实上无法估计加载操作所需要的时间。此时，使用“跑马灯”效果来指示加载进度，既可以让用户明白当前正在加载数据，而不是系统异常终止响应，又可以避免错误估计进度带来的负面用户体验。

另一方面，界面在显示加载进度指示器的同时，使用BackgroundWorker控件利用后台线程通过Provider加载数据。加载完毕后，通过消息将数据传递给主线程。主线程接收到消息后，进行数据绑定，然后隐藏加载进度指示器，并启用“下一篇”按钮。

第一页的题目难度由最后一次测试的成绩确定，如果之前没有进行过测试，则使用默认难度——难度三。由于同样的题目可能用于不同考试的不同难度，例如一篇难度介于大学英语四级和大学英语六级之间的题目，有可能出现于大学英语四级高难度的情况下，也有可能出现于大学英语六级低难度的情况下，此时，对于不同的考试，所限制的时间也不同，因此考试的倒计时显示器的起始时间是根据考试类型、考试项目和文章长度计算出来的。



图 4‑10 英语测试用例相关状态图

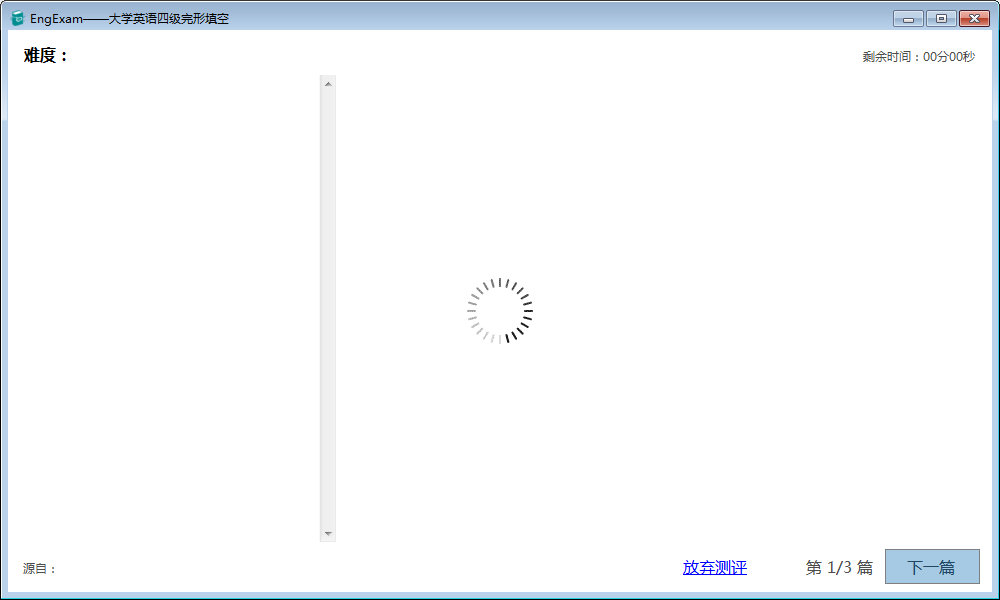


图 4‑11 英语测试界面加载数据时

完形填空和阅读理解都需要显示大段的文章。使用Label控件固然可以显示文本，但是不能够控制文本的间距，段落的间距和对齐方式等文本格式，只能简单地调整文字的字体。对于在用户停留时间最长的界面中用户花费时间最长的最集中注意力观察的部分，不能够用这种简陋的方式进行显示，必须能够给用户更好地阅读体验。使用WebBrowser控件可以在窗体中显示HTML文本，而HTML文本可以通过CSS（Cascading Style Sheets，级联样式列表）灵活控制文本显示的格式，从而带来更好地阅读体验。

用户可以随时通过点击“放弃测评”按钮离开测试界面，此时用户这次测试所做的努力将全被抛弃掉，因此属于一种“危险”操作，应该提示用户进行确认。由于用户可以通过多种途径离开英语测试界面，例如窗口右上角的“X”，快捷键，等等，因此不能仅仅在“放弃测评”按钮处进行处理，而是需要在这个窗体的关闭事件中进行处理。但是又需要另外一种情况，在程序发生未处理的异常时，将会关闭整个程序，此时不应该提示用户是否放弃测评，从而在错误的环境中继续运行，而是应该直接关闭当前窗口，以完成退出应用程序的过程。可以通过判断当前窗体的FormClosing事件参数的CloseReason属性来区分这两者。

用户点击“下一篇”按钮时，首先要判断用户是否完成了所有的题目，如果没有完成的话，则需要提示用户答完所有的题目才可以进入下一页。如果用户完成了所有的题目，则从服务器端子系统取得这篇文章的答案，判断出用户的正确率，以此计算下一篇文章的难度，再从服务器端子系统得到对应的题目。

如果当前题目是最后一页时，则“下一篇”按钮变为“提交”按钮，点击后将进入测试结果界面。

### 测试结果用例

测试结果界面主要由两张统计图表组成：左侧的统计图表显示用户每页题目的正确率，右侧的统计图表显示用户每个考点的错误率。界面的上方以等级的方式显示着这次测试的成绩，还有用户完成这次测试的总用时，以及测试日期。界面的下方有三个功能按钮：查看正确答案、重新测试和回到“我的进度”。

由于两张统计图表都是由服务器端子系统绘制的，因此加载过程中也需要进行异步加载。首先在两张统计图表的位置显示“跑马灯”加载指示器，然后在后台线程中将需要绘制统计图表的数据发送到服务器，并接收服务器绘制的统计图表，下载完成后发送消息到主进程，再由主线程将图片显示出来，并隐藏加载指示器。

测试结果所需要的所有数据，都是由会话对象从测试界面带来的。不仅如此，“查看答案”界面所需要的数据，也是由会话对象提供的，但是需要“测试结果”界面进行传递。

测试结果界面如图 4‑12所示。

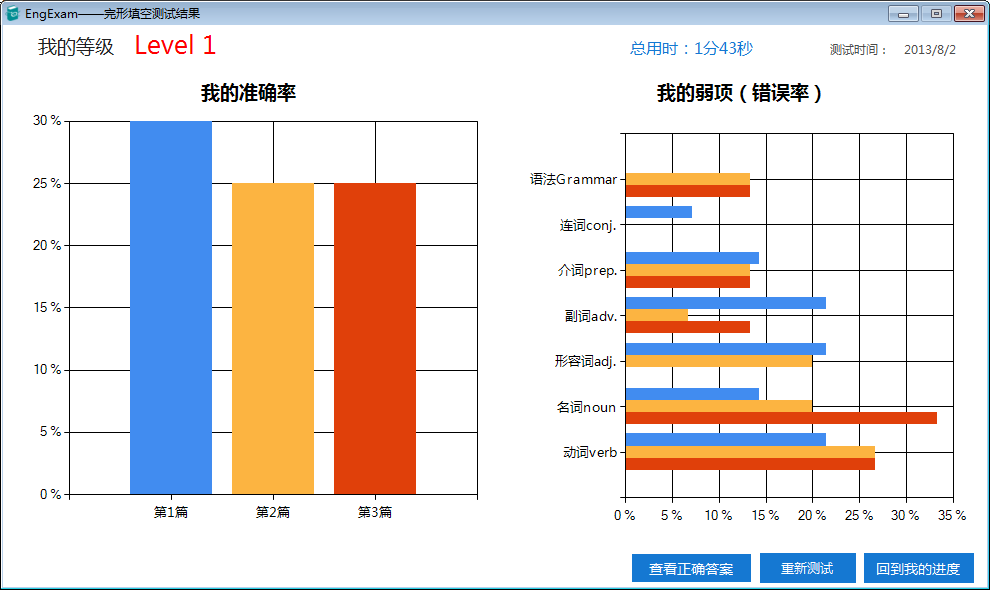


图 4‑12 完形填空测试结果界面

用户点击“查看正确答案”按钮，则会打开“查看答案”界面。用户点击“重新测试”按钮，则首先会回到“我的进度”界面（但是并不显示出来），将完成测试之后的逻辑处理完毕，然后再重新生成会话对象，打开英语测试界面供用户重新进行测试。用户如果点击“回到我的进度”按钮，则会回到“我的进度”界面，运行完成测试之后的逻辑，然后将“我的进度”界面显示出来。

### 查看答案用例

在测试结果界面点击“查看正确答案”按钮，就可以进入查看答案界面，如图 4‑13所示。

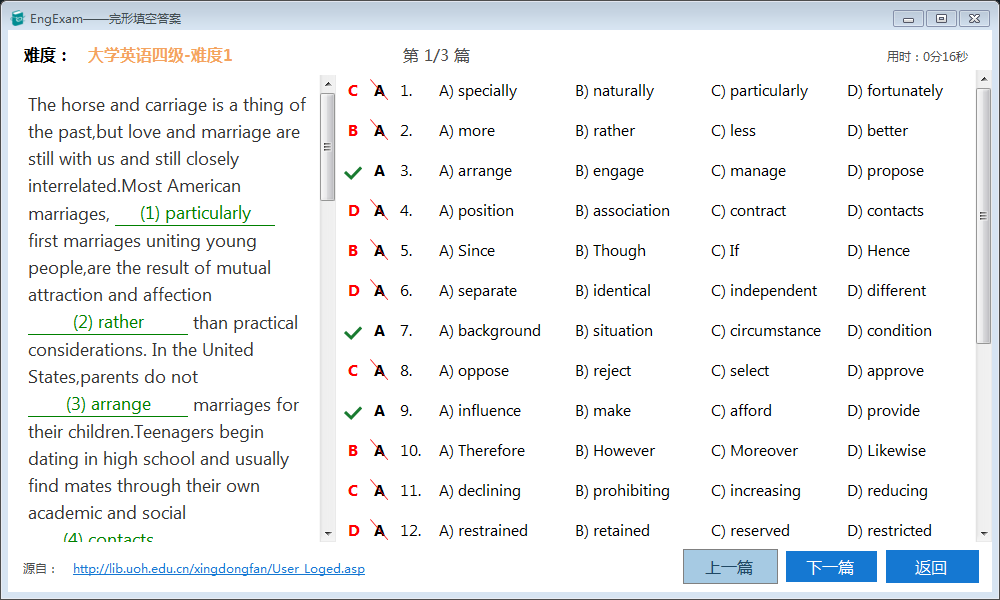


图 4‑13 完形填空答案界面

查看答案界面的数据都是由会话对象携带而来的，因此属于在本地加载的，不需要考虑异步加载，因而在逻辑上比较简单。但是在错误答案上画红线，不是标准控件所支持的，需要手工制作一个控件，用于处理这种情况。此处有两种方法，一种是准备四幅图片，分别是A、B、C、D上面有一条红线，另一种方法是制作一个纯手工控件，可以显示任何内容，但是会带上一条从左上到右下的红线。前者的好处是比较简单，后者的好处是扩展性好且占用的磁盘空间和内存空间都比较小。这里实现采用的是后者，在Label控件的基础上重写（Override）Render函数，调用GDI+接口直接在Label显示的结果上画一条红线，要注意反锯齿。

## 服务器端子系统

与客户端子系统相比较，服务器端子系统的功能比较简单。由于服务器端子系统不需要和用户直接交互，因此这方面的逻辑得到了极大地简化。但是服务器端子系统必须能够同时处理成千上万客户端子系统的请求，因此必须具备高可用性、高性能和高并发性。

### 统计图表生成

由于客户端子系统运行环境的限制，不能够从本地生成统计图表，必须由服务器端子系统提供。所有的统计图表分为三类：柱状图、条形图和饼状图。前两者出现在“测试结果”界面，后者出现在“我的进度”界面。由于不同考试项目绘制的统计图表，其中的统计项名称、数量都不相同，因此绘制统计图表功能需要具有一定的通用性。

服务器端子系统使用MSChart控件生成统计图表，提供生成柱状图、条形图和饼状图三种统计图表的服务。所有的服务都接受以下参数：统计图表的标题、统计图表内容数据和统计图表的大小。为了使生成的统计图表尽量清晰，还需要设置图形和文字的抗锯齿。为了使生成的统计图表具有相同的配色方案，还需要在生成时指定所有的统计图表都使用同一套配色方案。

### 试题信息查询

客户端子系统在向服务器端子系统查询试题时，需要指定的考试类型、考试科目和考试难度下的随机试题。

服务器端子系统接收到请求时，首先根据考试类型和考试科目，将考试难度映射为试题难度；然后再通过提供者，获得指定试题难度的随机试题，以及该试题的配套子问题。

翻译考试类型比较特殊，它不像完形填空和阅读理解，一道题里面包括文章和若干子问题，而是只有原文。为了保持与完形填空和阅读理解的一致性，子客户端子系统中一页将展示三道翻译题，如图 4‑14所示，否则浪费的空间太多，会让用户有一种不协调的感觉。因此，服务器端子系统在收到客户端子系统查询翻译试题的请求时，会将三道同难度的试题打包发送回客户端子系统。



图 4‑14 英语测试翻译界面

### 提供者模式

使用提供者模式，使得服务器端子系统可以适配不同的数据库。在开发初期，可以不使用数据库而是直接将少量数据内置于编码中，在运行期间保存在内存里，用于查看效果，验证代码能够正确运行。在开发中期和测试期间，可以使用局域网内部的SqlServer服务器提供服务，便于在企业内部使用。最终，产品必须部署在Azure云平台，届时必须使用Azure Storage作为数据提供者。

另一方面，即便是同样使用SqlServer，在开发期间和测试期间乃至于部署期间，也可能会使用不同的数据库内容，从而在代码上必须进行一定的配置和变化。

使用提供者模式，可以有效地封装这些变化。以完形填空为例，服务器端子系统使用的提供者模式的类图如图 4‑15所示。



图 4‑15 服务器端子系统提供者模式类图

Controller类分别持有对Provider类的接口的引用，而不依赖于具体的实现，具体的实现由依赖注入模块创建，并通过构造函数传递给Controller类。这使得在实际使用中，可以任意组合这些提供者，例如完形填空和阅读理解使用SqlServer提供者，而翻译使用内存数据。另一方面，这些提供者本身在创建的时候还可以进行一些配置，例如SqlServerClozeProvider，在创建的时候可以指定服务器的地址和数据库名称，或者是直接指定数据库连接字符串，这样甚至可以在开发的时候使用本地开发数据库，而在测试的时候使用局域网的SqlServer服务器，而不需要修改任何Controller类的代码。

### 依赖注入与Web API框架的整合

服务器端子系统使用了Web API框架，这使得服务器端子系统的依赖注入技术需要处理好与Web API框架的关系。除此之外，服务器端子系统与客户端子系统也有一些不同会影响到依赖注入模块的设计。

首先，客户端子系统具有单一的入口——主窗体，而服务器端子系统没有统一的入口，而是由框架自动创建需要的对象。其次，客户端子系统基本上是单线程的系统，大部分操作都在UI线程完成；而服务器端子系统是多线程的，几乎每一次用户请求都会激活一个新的线程，直到线程池耗尽。这两点差别，使得服务器端子系统的依赖注入模块与客户端子系统的大大不同。

首先考虑第二点，因为第二点不同会产生更为本质的问题。必需得区分清楚，服务器端子系统中的类，哪些是可以长期存在的，哪些是必须每次用户请求都创建新的。如果使用内存数据，则所有的类都可以是长期存在的，并且由于所有的操作都是读取操作，因此也不会有并发竞争问题。如果是使用SqlServer，则每一个类也都可以是长期存在的，因为每次执行都会创建线程内的DbCommand对象，而DbCommand对象是向数据库发起请求的单位。如果使用Azure Storage，则所有的对象都必须在每次请求创建，因为Azure SDK内部会使用同一个代理对象发起请求，有可能会产生并发性问题。

接下来，再考虑第一点，第一点问题本质上就是如何让Web API框架支持依赖注入，好在Web API框架已经预留出了接口，让人们可以自行适配不同的依赖注入容器。对于我们自行实现的依赖注入模块，也可以通过编写一个适配器，来完成与Web API框架的融合。编写这一适配器的时候，一定要考虑上面提到的第二点问题。Web API框架预留出的依赖注入容器接口有两个：IDependencyResolver和IDependencyScope。前者有一个方法BeginScope用于创建后者的实例。前者的实例创建的对象将会在整个应用程序域的活动期中生存，而后者创建的实例，将在后者的实例释放（Dispose）的时候一同释放。通过这样两个接口，就实现了不同生命周期的控制。在Global.asax中注册自行编写的依赖注入容器适配器后，Web API框架会调用恰当的方法创建Controllers，并且满足它们的构造函数依赖项。

# 系统测试

## 功能性需求测试

由于用户界面的自动化测试需要编写大量代码，对于本系统来说相对有些不值得，因此大部分测试依靠手工完成。

### 客户端子系统

考试类型分为大学英语四级、大学英语六级和考研英语，考试项目分为完形填空、阅读理解和翻译。进入考试之前的状态则更多，分为之前这项考试的所有考试项目都没有进行过，之前这项考试有其他考试项目完成过但这项考试项目没有进行过，之前这项考试有其他考试项目没有进行过但这项考试项目完成过，以及这项考试的所有考试项目都完成过。还有一种特殊的情况，是从查看答案界面通过点击“重新测试”按钮再次进行测试。其中，当前测试的考试项目完成过的情况和点击“重新测试”按钮再次测试的情况，又分为完成等级为1到5五个等级。由这几个维度组合成的起始状态就相当多了，对于主要的测试用例，我们应该测试完全，需要对每一种起始状态进行测试；对于次要的测试用例，我们只需要在起始状态的集合中选几个不相交的点，这些点能够穷举每一个维度上的所有可能即可。

测试的最主要的测试用例，就是用户进行英语测试的这一整套过程。这一过程必须充分测试，因此需要测试上述提到过的所有可能的起始状态。这一套过程是这样的，用户开始测试，用户完成每一道试题并点击“下一篇”按钮直到最后一页时，点击“提交”按钮，进入测试结果界面。在测试结果界面点击“查看正确答案”按钮，进入查看答案界面。在查看答案界面不断点击“上一页”按钮和“下一页”按钮查看所有三页测试的答案，然后点击“返回”按钮回到测试结果界面。此时有两个选择，一个是点击“重新测试”按钮再次进行测试，另一个是点击“回到我的进度”按钮返回“我的进度”界面。在进行主要测试用例的测试的时候，一切都模拟用户正常使用，查看界面是否有异常。例如，用户答题时填写的答案，是否和查看答案界面显示的用户答案一致，现实的文章的顺序是否一致，等等。

例如，其中一个测试用例是这样的。

起始状态：

1. 用户没有进行过任何测试。
2. 处于“我的进度”界面。

测试步骤：

1. 打开Fiddler2，将目标进程设为“EngExam.Client.Managed.exe”。
2. 点击界面上的“完形填空”按钮。
3. 接换到Fiddler2中，记录下接收到的数据中的ClozeId。
4. 记录下完形填空文章的前三个单词，以及当前题目难度。
5. 随机填写一半题目的答案，并点击“下一篇”按钮。
6. 在弹出的窗口中点击“确定”按钮。
7. 随机填写剩下一半题目的答案，并记录下当前的答案和用时，然后点击“下一篇”按钮。
8. 重复3-7的过程。
9. 重复3-7的过程，但是在需要点击“下一篇”按钮时，改为点击“提交”按钮。
10. 记录“测试结果”界面中的总用时，最终等级，两张统计图表。
11. 点击“查看正确答案”按钮。
12. 进行验证，并点击“下一篇”按钮。
13. 重复12。
14. 进行验证，并点击“上一篇”按钮。
15. 进行验证，并点击“返回”按钮。
16. 点击“返回我的进度”按钮，并验证。

期望结果：

1. 在上述操作流程中没有异常提示窗口弹出。
2. 第一篇测试的难度是3。
3. 在数据库中查找记录下来的ClozeId，验证其题目、题目难度、子问题是否符合记录下来的信息。
4. 在只填写了一半答案时点击“下一篇”按钮或“提交”按钮时，弹出如图 5‑1所示的提示窗口。
5. 在“测试结果”界面中显示的“总用时”与三次测试中花费的时间的总和相差在两秒以内。（可能有毫秒未在界面中显示出来，导致总和有所偏差）
6. 在“测试结果”界面中显示的两张统计图表和“最终等级”正确。
7. 在“查看正确答案”界面“上一篇”按钮在当前显示的是第一篇题目时不可用，在其他时候可用；“下一篇”按钮在当前显示的是第三篇题目时不可用，在其他时候可用。
8. 在“查看正确答案”界面中显示的题目、难度、子问题、标准答案、用户答案、用时正确。
9. 这三篇测试的答案正确率和难度符合难度变化的规则。
10. 返回“我的进度”界面后，该界面更新为如图 4‑5所示的界面。
11. 返回“我的进度”界面后，该界面显示的统计图表正确，第一次测试的成绩和最后一次测试的成绩相同，并且和在“测试结果”中显示的内容一致。



图 5‑1 答完问题提示界面

除了主要测试用例，还要进行一些边缘条件的测试，例如用户的正确率是零或者是百分之百的时候，统计图表是否正确，再比如，用户答题的时候，时间如果耗尽的话，程序的处理是否正确。更难测试的是，程序出现异常情况的时候，是否能够正确处理，例如突然切断网络连接，或者人为地制造一些异常。为了便于测试这些情况，在用于测试的版本中，专门注册了一个全局快捷键，用于引发异常。

一个测试边缘条件的测试用例如下所示。

起始状态：用户在没有进行过任何测试的情况下，进行完形填空测试，测试完成后，在“测试结果”界面，通过点击“重新测试”按钮，重新开始完形填空测试。

测试步骤：

1. 记录下当前的最终等级。
2. 打开Fiddler2，将目标进程设为“EngExam.Client.Managed.exe”。
3. 点击“测试结果”界面上的“重新测试”按钮。
4. 接换到Fiddler2中，记录下接收到的数据中的ClozeId。
5. 记录下完形填空文章的前三个单词，以及当前题目难度。
6. 随机填写一半题目的答案，并点击“下一篇”按钮。
7. 在弹出的窗口中点击“确定”按钮。
8. 在数据库中查询ClozeId对应的答案，并将所有答案改为正确答案，并使用正确答案填写剩下的一半题目的答案，记录下用时，然后点击“下一篇”按钮。
9. 重复4-7的过程。
10. 在数据库中查询ClozeId对应的答案，并将所有答案改为错误答案，并使用错误答案填写剩下的一半题目的答案，记录下用时，然后点击“下一篇”按钮。
11. 重复4-7的过程，但是在需要点击“下一篇”按钮时，改为点击“提交”按钮。
12. 在数据库中查询ClozeId对应的答案，并将所有答案改为正确答案，剩下的一半题目的答案留空，然后等待计时器归零。
13. 记录“测试结果”界面中的总用时，最终等级，两张统计图表。
14. 点击“查看正确答案”按钮。
15. 进行验证，并点击“下一篇”按钮。
16. 重复15。
17. 进行验证，并点击“上一篇”按钮。
18. 进行验证，并点击“返回”按钮。
19. 点击“返回我的进度”按钮，并验证。

期望结果：

1. 在上述操作流程中没有异常提示窗口弹出。
2. 第一篇测试的难度是点击“重新测试”按钮之前记录下的最终等级。
3. 在数据库中查找记录下来的ClozeId，验证其题目、题目难度、子问题是否符合记录下来的信息。
4. 在只填写了一半答案时点击“下一篇”按钮或“提交”按钮时，弹出如图 5‑1所示的提示窗口。
5. 第三篇测试中，时间用尽后不提示用户应该完成所有问题，而是直接提交答案。
6. 在“测试结果”界面中显示的“总用时”与三次测试中花费的时间的总和相差在两秒以内。（可能有毫秒未在界面中显示出来，导致总和有所偏差）
7. 在“测试结果”界面中显示的两张统计图表和“最终等级”正确，尤其是正确率统计图表，应该大致如图 5‑2所示（第三篇文章的准确率在50%左右）。

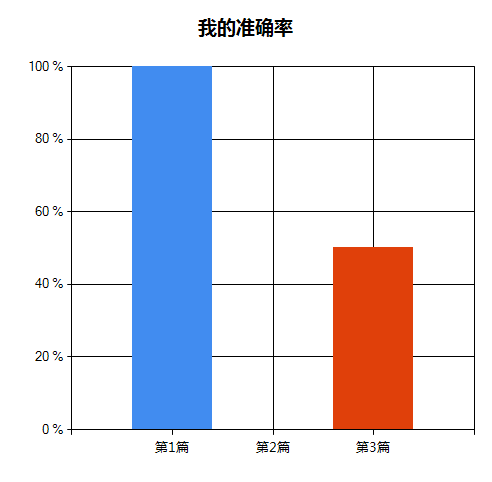


图 5‑2 准确率统计图表边缘情况

1. 在“查看正确答案”界面“上一篇”按钮在当前显示的是第一篇题目时不可用，在其他时候可用；“下一篇”按钮在当前显示的是第三篇题目时不可用，在其他时候可用。
2. 在“查看正确答案”界面中显示的题目、难度、子问题、标准答案、用户答案、用时正确。
3. 第三篇文章的答案，用户未填写的部分仅显示正确答案，而不显示用户答案，并且判断这道题用户做错了，如图 5‑3所示。
4. 这三篇测试的答案正确率和难度符合难度变化的规则。
5. 返回“我的进度”界面后，该界面更新为如图 4‑5所示的界面。

返回“我的进度”界面后，该界面显示的统计图表正确，第一次测试的成绩和最后一次测试的成绩相同，并且和在“测试结果”中显示的内容一致。

除此之外，还要检测程序是否能够处理被破坏的外部文件，比如用户如果删除或者篡改了用户文件夹中存储的历史测试成绩，程序是否能够妥善处理。

除此之外，还有一些次要的测试用例，比如中途退出系统，或者使用任务管理器结束进程等等。

### 服务器端子系统

服务器端子系统由于没有界面，因此相对容易进行测试，但是生成统计图表的部分，仍然需要人工辨别。

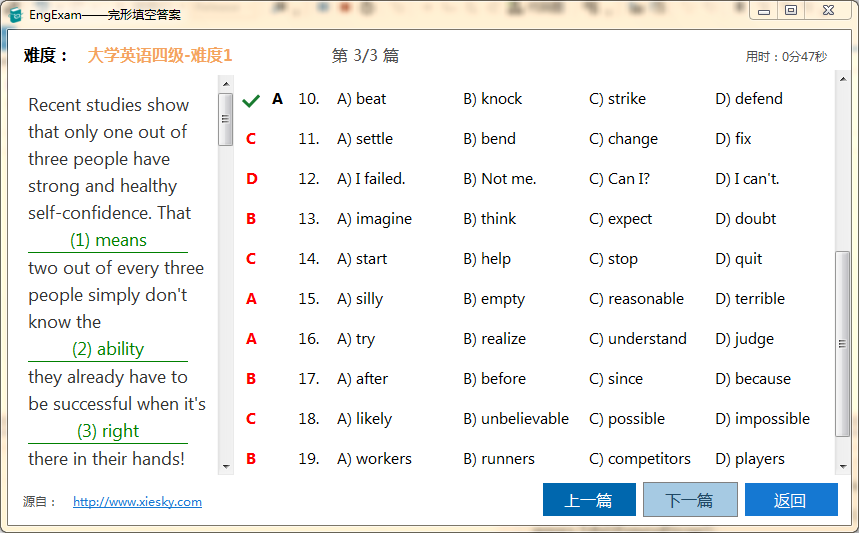


图 5‑3 查看答案边缘情况界面

服务器端子系统的自动测试程序使用和服务器端子系统相同的数据库，测试程序自动向服务器端子系统发起查询，然后再去数据库中取回相同的数据进行验证，难度等级的规则需要额外进行验证。

## 非功能性需求测试

除了要对明显的功能性需求进行测试，也需要对非功能性需求进行测试，以保证产品能够满足这些需求。对非功能性需求的测试没有规律可循，有些测试项目比较容易进行测试，有些则十分复杂，甚至需要编码时的配合。

### 性能需求

性能需求对于客户端子系统的要求有三点。第一点是，要求客户端子系统在接收到服务器端子系统返回的数据后，响应时间不超过0.3秒。第二点是要求客户端子系统占用内存不超过40Mb。第三点是要求客户端子系统占用的磁盘空间不超过5Mb，客户端子系统的安装包的大小不超过3Mb（不含.NET Framework）。

后两点比较容易验证，而第一点比较困难。但是我们可以通过配置依赖注入模块，使用内存提供者提供数据，这样就没有网络连接的开销，然后就可以测定显示数据是否不超过0.3秒。

如图 5‑4所示，客户端子系统在运行一段时间后，内存占用约21Mb。



图 5‑4 客户端子系统内存占用

性能需求中，对于服务器端子系统的要求是能够至少承载每秒100次请求，并且每次请求的响应时间不超过0.7秒。

使用Visual Studio中的“Web性能和负载测试”项目，对服务器端子系统进行测试。调节参数，预热时间1分钟，每秒钟发送100个请求，预期每次请求持续时间1.7秒，测试时间30分钟。这些请求随机均匀分布到服务器端子系统的不同服务接口上。测试结果显示，总共有不到100个请求发生了错误，有不到5个请求响应时间达到2秒。排除网络负载波动的影响，说明服务器端子系统基本满足需求。

### 运行环境需求

比较容易验证，客户端子系统在不同版本的操作系统下的表现。需要特别注意的是，客户端子系统使用了WebBrowser控件，而WebBrowser控件最终会使用IE内核对内容进行渲染，因此浏览器兼容性也必须考虑在内，特别是在IE6下的表现。经过验证，客户端子系统在各个版本的IE下，均运行良好且显示内容正常。

### 客户端子系统标准规定

在Windows中创建一个受限账户，即可测试对客户端子系统权限限制的需求。

对于重入的需求的测试，则复杂一些。首先测试的是，无论在什么样的情况下，都不会启动客户端子系统的两个实例。一般的情况比较容易测试，除此之外，还需测试在已启动的实例失去响应的情况下，是否能够运行第二个实例，测试的方法是使用任务管理器将已启动的实例挂起，然后再启动第二个实例，此时虽然无法激活已启动实例的当前窗口，但是也不会同时启动多个实例。另一方面，在测试激活已启动实例的窗口时，还需要考虑窗口最小化的情况，窗口被其他窗口遮挡的情况，窗口位于其他显示器的情况，等等。

### 收集数据

最直接的测试收集数据的方法就是运行一遍测试，然后去数据库中查询是否有需要手机的数据。但是实际情况中，由于数据收集数据库是高度分布式的，每天处理的数据量非常大，因此数据生效需要比较长的时间，甚至不能够保证所有的数据都有效。因此，不得不考虑用其他的方法进行测试。

考虑到所有的请求都是通过TCP发送的，因此可以在本地使用工具将发送的收集数据的数据包拦截下来，进行记录。这样做的好处是，第一，见效快；第二，可以针对性的测试每一个发送数据的点；第三，可以看到记录的数据是否正确。但是第三点既是优点又是缺点，可以人工观察发送的数据是否正确固然是一个优点，但是反过来数据的正确性验证全都依赖于人工，费时费力，不得不说也是一个缺点。

# 关键技术难点分析与解决方案论述

本系统有很多的特色散布在系统的不同模块之间，从而在以模块划分进行详细设计和实现的阐述中不能够讲述清楚。下面将以本系统的特色为中心进行阐述，有些特色属于关键的理念或技术实现，有些特色则是在实践中遇到的困难以一种比较特殊的方式解决。

## 众包在本系统中的应用

众包在本系统中有两种形式的体现。一种体现在完形填空和阅读理解这两种考试项目的测试中，另一种体现在翻译中。

首先来看完形填空和阅读理解，这两种考试项目除了需要题目以外，还需要提供四个候选答案。因此，只有已经确定了答案范围的情况下，才能够利用这种形式，通过众包的力量选出真正正确的答案。举例说明，比如最近互联网出现了一个新的流行词——Crowdsourcing，必应Bing搜索中经常有人搜索这一单词，从而被系统所注意到，但是由于这个词太新了，微软的词库中并没有关于这个词的解释，这个时候这个词被提取出来连同用户在必应Bing搜索中经常点到的搜索推荐作为候选答案一起进入数据处理子系统，生成题目。这个题目由于含有难词——Crowdsourcing，从而被标记为难度较高的题目，因此只有高水平的用户会抽到这一题目。很多高水平的用户做了这道题后，就会产生很多关于Crowdsourcing这一单词意思的答案，我们认为这些答案如果收敛程度超过某一阈值的话，则Crowdsourcing的意思就能够确定了，从而可以将这一新词补充进语料库中。

再看翻译，翻译不同于完形填空和阅读理解的地方是，翻译的答案不确定，因此也就使得我们可以有更大的自由度来生成题目，而不需要把答案限制到某一范围内这一限制。但是这又带来了另一些问题，由于答案的开放程度太高，使得真正答案完全有可能收敛于几个互不相同的点上，从而很难将正确答案和错误答案区分开。这个问题一方面可以靠更加深入的基于规则的自然语言处理进行一定程度上的解决，另一方面也可以更加深入的引入众包进行解决。例如通过社区的形式，让更多的用户参与进来，把收敛后的答案作为候选答案，在显示答案界面显示这些候选答案让用户进行评价、修改。这一版本一方面由于成本的原因没有加入这些功能，另一方面中低端用户的反馈有多大作用还有待评估。

## Azure云平台的特殊性

本系统的服务器端子系统使用Azure云平台托管，数据库使用Azure Storage托管，这使得Azure云平台一些微妙之处，都对开发和部署造成了影响。

首先，Azure云平台不能托管Native文件，这给使用外部系统带来了一定的不便。其次Azure云平台的配置文件需要使用云平台提供的API进行读取，而不是使用ASP.NET的配置文件接口。最后，Azure云平台的部署方式和使用IIS部署ASP.NET程序有很大的区别。Azure云平台使用SDK提供的方式将编译后的程序集打包成一个自动部署包上传到Azure云平台上，然后由Azure云平台自己部署程序。Azure云平台根据配置会使用若干个实例（类似于虚拟机）托管这一程序集，并自动进行负载均衡，因此在部署的时候，就要保证程序集版本的一致性，这使得部署的工作复杂且时间较长。

Azure Storage完全不同于SQL Server。SQL Server是关系型数据库，而Azure Storage是非关系型数据库。这意味着，需要手工维护表之间的关系，并且还需在必要的时候使用事务。Azure Storage使用Partition Key和Row Key来唯一标识数据，同一个表中的数据，其结构可以是不同的。同样的，Azure Storage也会将数据分布到不同的实例中，并且只能保证数据的最终一致性。这意味着修改操作需要花费一定的时间，才能够扩散到所有的实例中去。因此，如果需要修改数据，则需要等待一定的时间，以使这一操作生效。Azure Storage限制批量操作（含读取）一次最多只能对1000个结果生效，这使得操作大量数据十分困难，尤其是验证数据的正确性。

Azure云平台的管理工具还不全面，尤其是针对Azure Storage的工具，目前还没有一个工具可以像SQL Server Management一样可视化的操作Azure Storage中的表和数据。这意味着我们必须手工操作，或者是自己写一些工具出来。

## 数据收集系统的可靠性

客户端子系统在运行期间会收集很多数据，这些数据如果存放在内存中，很容易丢失。用户在使用过程中，随时有可能退出客户端子系统。更糟的是，客户端子系统所处的网络环境并不能够得到保证，收集到的数据很有可能暂时无法发送出去。在这样的情况下，如何保证客户端子系统收集到的数据最终能够不重不漏的发送到服务器端子系统，就是一个非常需要重视的问题了。

客户端子系统在收集一条信息时，首先将其写入数据文件和内存数据库中，然后尝试后台发送这条数据。如果发送成功了，则将这条数据从数据文件中标记为删除，并且在内存数据库中将这条信息真正删除，否则尝试重新发送，如果三次尝试均失败了，则什么也不做。客户端子系统的数据收集模块在初始化时，还会在后台注册一个定期触发的事件，定期将内存中保存的信息覆盖到数据文件中，这会保证数据文件可以定期的真正删除那些已经发送出去的数据。数据收集模块在加载时，首先查找磁盘上的数据文件，如果存在数据文件，则从数据文件中加载所有的数据，然后一一尝试将其发送出去。这样一来，利用类似于检查点、日志的原理，保证了数据收集模块的可靠性。

## 完形填空答题时文章颜色变化

完形填空这一界面，实际上经过了颇多的考虑。其中的一种是，只显示文章，而不显示右侧的答题界面，用户通过点击填空的方式，将候选答案悬浮显示在填空的上方供用户进行选择。但是经过试验，发现这种方式和传统的完形填空考试相差太大，并且在用户做完这套题后，复查自己的答案的时候，由于不能够看到其他候选项，反而会造成一些困扰。最终，还是决定向传统试卷上的完形填空靠拢，一侧显示文章，另一侧显示候选答案。但是这里面仍然有一些特色：当用户选择答案的时候，左侧的文章会通过颜色的变化来提示用户当前做到了什么位置，如图 6‑1所示。

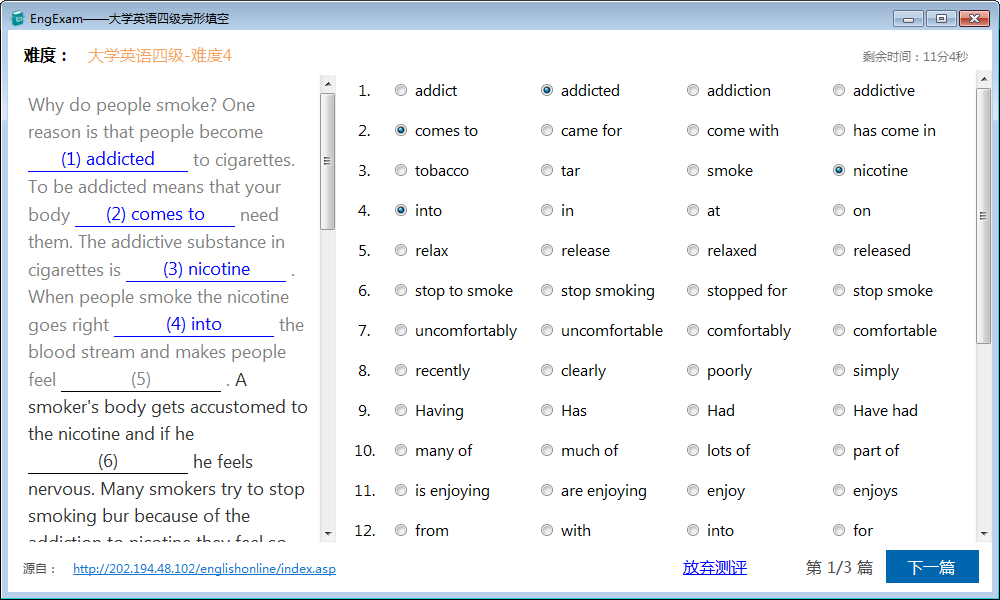


图 6‑1 完形填空界面

其实现的原理是，一开始在文章的开头处插入一个空的span标记，在答题的时候，移动span的结束标记到当前空之后的第一个标点符号处。这一span标记之内的文字，通过css控制显示为浅灰色，用于表示用户已经完成了这部分的内容，方便用户在右侧选择了答案后，迅速的将注意力集中回之前被打断的地方。

我们还考虑过另一种方案，即当用户选择完答案后，立刻将左侧的文章滚动到这个空处，这样用户每次选择完答案后只需从文章的最上侧继续阅读即可。通过用户调查发现，这种方案有一种弊端：用户通常会将视线转回原来被打断的位置。除此之外，有时我们还需要这个空之前的部分单词，才能够连贯的读完当前这一句子。因此，我们最终摒弃了这一方案。

## 鼠标悬浮时使用滚轮滚动控件内容

WinForm提供的控件支持使用鼠标滚轮滚动当前激活（Active）的控件，但是有的时候这样多少还有点不便，如果能够在控件处于非激活（Inactive）状态的时候，也可以使用鼠标滚轮滚动鼠标所在的控件，则会比较方便。特别是在客户端子系统中，完形填空和阅读理解考试项目的英语测试界面，左边的文章和右面的试题都需要使用滚动条才能够显示完全，用户需要不断地滚动左右两个控件。这时，如果能够在不激活（不使用鼠标点击，键盘Tab键等方式将将焦点移动到）控件的情况下，使用鼠标滚轮滚动鼠标位置所在的可滚动控件，则会让用户觉得非常方便。

由于WinForm框架的设计本身，没有考虑到这种情况，因此我们必须突破WinForm的限制，直接使用Windows API进行操作。C#并不直接提供Windows API调用，我们必须使用P/Invoke（Platform Invocation Services，平台调用服务）调用非托管（Native）的Windows API。WinForm的底层仍然遵循Windows API创建窗口的事件机制，在客户端子系统内使用鼠标滚轮滚动，也会向当前UI线程的消息泵发送消息。我们需要做的就是，在WinForm的消息泵（Message Loop）处理并分发（Dispatch）这一消息之前，将其拦截下来，判断鼠标位置是否处于我们需要的可滚动控件（左侧显示试题文章的WebBrowser控件，或者右侧显示题目的Panel控件）中，如果是的话，我们自行将这一消息发送到对应的控件上，让对应的控件处理好这一消息。

WinForm提供了一个机制，可以注册一个应用程序范围的消息过滤器，用于在消息被分发之前，对其进行处理，并指示消息泵是否在处理后继续对其进行分发。由于UI线程只有一个，所以消息泵也只有一个，所以这一消息过滤器一定会是整个应用程序范围内的，而不是以窗体为单位的。但是我们所需要处理的控件却是在窗体内的，其生命周期与窗体相同，因此，我们需要注意，一定要在窗体关闭的时候，取消注册这一消息过滤器，否则将会产生错误。另一方面，我们不希望在当前窗体处于非活动状态的时候，也响应鼠标滚动事件，因此需要分别在窗体激活和失去活动状态时，注册或取消注册这一消息过滤器。

消息过滤器会拦截所有的消息，首先判断其是不是一个鼠标滚动的消息，其次判断发出这个消息时，鼠标所在的位置是否处于目标控件所在区域内，并且目标控件和目标控件内的所有控件都不处于激活状态，此时，我们就处理这一消息，否则就留给消息泵继续分发这一消息。

对于WinForm控件，如果其实现了ScrollableControl抽象类，则其可以正常处理鼠标滚动事件，我们只需将拦截下来的消息转发给目标控件即可。而WebBrowser控件则比较特殊，它没有实现ScrollableControl抽象类因此我们不能把鼠标滚动事件直接转发给它，但是其内部使用了IE内核，并且IE内核部分支持鼠标滚动。因此，我们需要找到WebBrowser控件内部使用的IE内核的句柄（Handler），然后直接将拦截下来的消息转换成IE内核能处理的消息，然后转发给IE内核。

## 保证只运行一个实例

大多数应用程序，用户都不希望其能够启动多个实例，而是希望在启动第二个实例的时候能够将第一个实例激活。微软的标准规定里要求，不需要启动多个实例的客户端应用程序必须保证实现只能运行一个实例的功能。

那么我们该如何实现这一功能呢？

同一个应用程序的两个实例，必然是运行于不同的进程的。因此要保证只能运行一个实例，就必须使用进程间通信。使用进程间通信，如果一直开启端口监听是否有从第二个实例传来的消息，代价太大，必须想办法使用其他的方法。

仔细思索这一问题，发现其可以分解为两个步骤。第一，要保证只能启动一个实例；第二，要在启动第二个实例的时候，通知第一个实例激活当前窗口。前者可以通过一个命名的互斥量（Named Mutex）来实现，后者的实现要想效率比较高则稍微复杂一些。首先，第一个实例建立一个TcpServerChannel用于服务，然后将回调函数置于一个MarshalByRefObject类中用于跨越应用程序域边界来引用调用这一对象。封送（Marshal）这一对象时，将对象的地址写入一处共享内存中。这样，第二个实例在启动时，获得互斥量失败时，就可以从共享内存中得到封送对象的地址，并取得封送对象的代理，调用其回调函数通知程序的第一个实例，激活主窗口。

此时又产生了新的问题：当前应用程序的主窗口是谁？为了解决这一问题，还需要引入一个窗体管理器，用于追踪当前的活动窗口以及活动窗口的状态。这样做的副作用就是，每当打开、关闭或者是改变一个窗口的状态时，都必须手工向窗体管理器注册。

通过这样两方面处理，我们就解决了这一问题。

# 结论

本系统以英语辅助学习软件为载体，应用“众包”的思想解决收集高质量的信息以改进自然语言处理语料库的问题。一方面，为解决传统问题提供了新的思路，并且在这一思想的道路上迈出了探索性的一步；另一方面，本系统将“众包”这一新思想与传统问题相结合，为“众包”这一思想的应用做出了一定贡献。同时，本系统依托于Azure云平台这一新兴技术，为使用Azure云平台提供了一个参考实例。另外，本文还探讨了使用Azure云平台作为依托时，遇到的一些问题，可以给其他准备使用Azure云平台的读者带来一些警示。

本系统的所有部分均已经通过了测试，服务器端子系统已经托管在Azure云平台上提供服务。本系统符合预先提出的需求，完成了设计目标，为产品发布并最终投入使用做好了准备。

但是，由本系统所探讨的内容可以看出，本系统只是对将“众包”思想应用于自然语言处理问题的一个初步设想的探索，功能还比较简单，有一些问题在取得更详尽的数据之间，还得不到解答。例如，是否应该加入社区功能，让更多的用户参与答案的评选，而不是仅取决于高水平用户的答案。这样的问题，只有将本系统作为产品发布出去，收集到一定量的数据后加以分析，才能够得出答案。同时，这些问题得到解决之后，又为这一系统的下一期目标指出了方向。这样循环递进下去，才能够产生一个优秀的产品。

另一方面，还可以看到，“众包”能够在一定程度上，用比较小的代价解决一些使用传统方法不容易解决的问题。我们对于“众包”这一新鲜概念，既需要小心探索，又需要大胆发挥。这样，才能够充分利用“众包”这一新事物，获得最大的利润。

# 参考文献

[1] MITCHELL T M. Machine Learning[M]. 第1版. McGraw-Hill Science/Engineering/Math, 1997.

[2] MURPHY K P. Machine Learning: A Probabilistic Perspective[M]. The MIT Press, 2012.

[3] 李航. 统计学习方法[M]. 清华大学出版社, 2012.

[4] JOSHI A K. Natural Language Processing[J]. Science, 1991, 253(5025): 1242–1249.

[5] MANNING C D, SCHÜTZE H. Foundations of statistical natural language processing[M]. Cambridge (Mass.): MIT press, 2001.

[6] KOEHN P. Statistical Machine Translation[M]. 第1版. Cambridge University Press, 2010.

[7] 陈郑汉;周经野. 自然语言与机器学习[J]. Natural Language and Machine Learning, 2000, 19(3): 6.

[8] 徐睿;王惠临;XU RUI;WANG HUILIN. 自然语言处理中的机器学习方法研究[J]. Machine Learning Techniques Applied to Natural Language Processing, 2008(z1): 5.

[9] 刘群. 统计机器翻译综述[J]. JOURNAL OF CHINESE INFORMATION PROCESSING, 2003, 17(4): 12.

[10] 张利斌, 钟复平, 涂慧. 众包问题研究综述[J]. Science &amp; Technology Progress and Policy, 2012, 29(6).

[11] AMBATI V, VOGEL S, CARBONELL J. Active learning and crowd-sourcing for machine translation[J]. Language Resources and Evaluation (LREC), 2010, 7: 2169–2174.

[12] NEGRI M, BENTIVOGLI L, MEHDAD Y, et al. Divide and conquer: crowdsourcing the creation of cross-lingual textual entailment corpora[C]//Proceedings of the Conference on Empirical Methods in Natural Language Processing. Stroudsburg, PA, USA: Association for Computational Linguistics, 2011: 670–679.

[13] ZAIDAN O F. Crowdsourcing Annotation for Machine Learning in Natural Language Processing Tasks[J]. 2012.

[14] 陆艳. 众包翻译模式研究[J]. Shanghai Journal of Translators, 2012(3).

[15] HOWE J. The rise of crowdsourcing[J]. Wired magazine, 2006, 14(6): 1–4.

[16] 霍绪艳. 中国众包发展现状研究[J]. 商情, 2011(47).

[17] 史新, 邹一秀. 威客模式研究述评[J]. 图书与情报, 2009(1): 69–72.

[18] 史新. “威客”模式在国内的发展现状及优化研究[J]. JOURNAL OF INTELLIGENCE, 2009, 28(1): 156–160.

[19] 洪丽萍. 猪八戒网: 第三方服务交易平台[J]. 新领军, 2011(12): 106–106.

[20] 吕部, 郑晓奋, 盛旗锋. 众包——企业管理创新新模式[J]. Journal of Suzhou College, 2011, 26(9): 24–26.

[21] 魏拴成. 众包的理念以及我国企业众包商业模式设计[J]. TECHNOECONOMICS &amp; MANAGEMENT RESEARCH, 2010(1): 36–39.

[22] 张玉凤. “众包模式”对科技传播的革新[J]. PUBLIC COMMUNICATION OF SCIENCE & TECHNOLOGY, 2010(4): 42–44.

[23] 赵景明, 时永梅. 图书馆众包模式的理论与实践研究[J]. Library Theory and Practice, 2011(8): 12–13,22.

[24] 盛芳, 耿艾莉. 网络环境下高校图书馆的四项工作的众包策略[J]. Library Tribune, 2012, 32(1): 15–19.

[25] 盛芳, 李正龙, 焦坤, et al. 众包与众包馆员制度：助推图书馆服务转型[J]. Document,Information &amp; Knowledge, 2012(4): 95–102.

[26] 颜艳春. 下一个消费革命:众包购物[J]. China Computer &amp; Communication, 2012(6): 48–51.

[27] 肖岚, 陈晨. 服装设计“众包”案例分析及其启示[J]. SHANDONG TEXTILE ECONOMY, 2010(3): 69–71.

[28] 宋鹏, 肖晓, 崔佳春. “众包”理论在旅游企业营销中的应用——以成都圆和圆佛禅客栈为例[J]. 管理学家：学术版, 2010(11): 66–69.

[29] 无. 众包将用于战车设计[J]. 工业设计, 2012(7): 10–10.

[30] 谭婷婷, 蔡淑琴, 胡慕海. 众包国外研究现状[J]. JOURNAL OF WUHAN UNIVERSITY OF TECHNOLOGY(INFORMATION &amp; MANAGEMENT ENGINEERING), 2011, 33(2).

[31] 豆瑞星. Waze：社交元素让地图变“活”[J]. 互联网周刊, 2012(22): 28–29.

[32] BERGVALL-K\AAREBORN B, HOWCROFT D. The Apple business model: Crowdsourcing mobile applications[C]//Accounting Forum. 2013.

[33] ROSSEN B, LOK B. A crowdsourcing method to develop virtual human conversational agents[J]. International Journal of Human-Computer Studies, 2012, 70(4): 301–319.

[34] DJELASSI S, DECOOPMAN I. Customers’ participation in product development through crowdsourcing: Issues and implications[J]. Industrial Marketing Management, 2013.

[35] STOLEE K T, ELBAUM S. Exploring the use of crowdsourcing to support empirical studies in software engineering[C]//Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. 2010: 35.

[36] ALBORS J, RAMOS J C, HERVAS J L. New learning network paradigms: Communities of objectives, crowdsourcing, wikis and open source[J]. International Journal of Information Management, 2008, 28(3): 194–202.

[37] BAZILIAN M, RICE A, ROTICH J, et al. Open source software and crowdsourcing for energy analysis[J]. Energy Policy, 2012.

[38] BÜCHELER T, SIEG J H. Understanding science 2.0: crowdsourcing and open innovation in the scientific method[J]. Procedia Computer Science, 2011, 7: 327–329.

[39] 吴乐珺. “众包” 模式推进美国公民新闻再发展[J]. 国际新闻界, 2007(8): 40–43.

[40] 豆瑞星. VoiceBunny: 配音众包化[J]. 互联网周刊, 2013(5): 24–24.

[41] An Innovative Crowdsourcing Approach for Amazon Mechanical Turk[J]. International Journal of Computer Applications, 2012, 52(4).

[42] 成竟. 当前英语学习软件的分类、分析与思考[J]. CHINA EDUCATION INFO, 2004(6): 31–33.

[43] 英语学习软件大阅兵[J]. DIANZI YU DIANNAO, 2003(1): 126–129.

[44] EBBINGHAUS H. Memory: A contribution to experimental psychology[M]. Teachers college, Columbia university, 1913.

[45] BURBECK S. Applications programming in smalltalk-80 (tm): How to use model-view-controller (mvc)[J]. Smalltalk-80 v2. 5. ParcPlace, 1992.

[46] FOWLER M. Inversion of Control Containers and the Dependency Injection pattern[EB/OL]. (2004)[2013-07-31]. http://martinfowler.com/articles/injection.html#ServiceLocatorVsDependencyInjection.

[47] SEEMANN M. Dependency Injection in .NET[M]. 第1版. Manning Publications, 2011.

[48] FOWLER M, RICE D. Patterns of enterprise application architecture[M]. Boston, MA: Addison-Wesley, 2003.

[49] SCHWARZ N, LUNGU M, NIERSTRASZ O. Seuss: Decoupling responsibilities from static methods for fine-grained configurability.[J]. Journal of Object Technology, 2012, 11(1): 1–23.

# 致谢

在研究学习期间，我有幸得到了三位老师的教导，他们分别是：我的导师叶勇副教授，中国科学技术大学的石竹讲师，微软亚洲研究院的Yuki Arase教授。三位老师深厚的学术功底，严谨的工作态度和敏锐的科学洞察力使我受益良多。衷心感谢他们给予我的悉心教导和热情帮助。

感谢微软亚洲互联网工程院给我提供实习的机会，并允许我以实习期间所做的项目完成这篇论文。感谢微软亚洲互联网工程院的资深开发组长石瑨，在工作上和生活上帮我解决了很多难题。感谢和我一起完成这个项目的小组其他成员：赵帅，严煜华，赵立，郭玮，Dason Song。项目的成功离不开我们的共同努力。

感谢我的父亲母亲，是你们一直支持我走到了现在。

2013年8月