摘要

在软件技术飞速发展的今天，软件测试作为对软件质量的保证，也得到了长足的发展。软件测试是对项目给定需求的功能验证，是软件质量管理Quality Assurance（品质保证）的重要手段。它减少了软件的质量问题，并对软件风险进行了调控。

BDD（行为驱动开发）风格的自动化测试在自动化测试的基础上对测试用例进行了行为模式的解释，在测试的基础上拓展了对软件过程管理的思考。而为了管理BDD风格的自动化测试用例，并且为非编码人员提供使用自动化测试的手段，开发基于BDD的自动化测试系统显得尤为迫切。

第一章 绪论

1.1研究背景与意义

随着软件技术的发展、测试难度的需求复杂化，自动化测试的理念逐渐被大多数企业所接受。并且在软件过程管理中，随着迭代式开发与敏捷开发等框架取代瀑布模型，测试的周期缩短，工作增加，需要对系统的多个迭代版本进行验证，并增加了许多重复性的劳动，往往需要自动化测试来节省时间与资源。

软件测试主要由四部分组成：被测应用、环境、方式、测试人员。

完善的自动化流程可以让环境部署的时间大大减少，以自动执行一些重复但是必要的任务的方式减轻测试人员的负担，并且解决许多手工测试完成不了的工作（如并发访问等），所以标准的自动化测试软件服务对项目持续交付和持续测试意义深远，同时对于项目的质量检测、监测都是不可或缺的。

然而，自动化测试在相当一部分的环境下，具有极大的不足。首先，自动化测试难以管理，在对软件进行全面的自动化测试当中，需要将许多的测试代码综合管理。并且，自动化测试的门槛相较于手工测试较高，它要求开发人员具有对环境进行搭建、并编写用例执行代码的编程能力。

所以，对自动化测试的研究并何应用在企业项目开发中的过程管理中的探讨势在必行，而使用SaaS软件提供BDD风格的自动化测试管理平台，将为自动化测试提供一个标准、严格、合乎软件过程管理规范的管理、执行工具。

1.2国内外研究现状

在过去50年中，从结构化系统分析和设计方法到敏捷方法（包括极限编程和测试驱动开发）的软件开发中出现了许多方法，而敏捷方法在软件方法中变得十分流行。理想的软件产品应该是便宜，快速开发，易于扩展和可维护的，而自动化测试为这一特点提供了支持。而由于敏捷方法的流行以及对测试的要求越来越高以及对软件测试方面的方法、自动化的研究深入，软件项目中使用自动化测试的倾向越来越大。

国外对敏捷过程管理中实现自动化有深入的研究。而在国内自动化测试属于较为新鲜的技术，在研究方面稍为落后，并且缺少丰富的应用经验。所以自动化测试在国内许多公司处于比较尴尬的境地：大部分公司依旧重度依赖手工进行缺陷的查找，或想开展自动化测试却并没有收获预想中的效果。

在企业实施自动化过程中，绝大部分国内企业选择自己实现测试自动化，并开展自己的服务，成立自己的自动化测试部门来组建自己的一套流程管理系统工作。而在国外，由于国外SaaS软件的盛行，国外的企业（尤其是中小型企业）往往是通过选择由专门企业提供的测试自动化平台托管服务，并对此服务付费，或是通过外包、派遣、离岸等多种方式相结合来实现对自动化测试的需求。国外的企业关注测试带来的结果、发现的问题，而不是由如TestingWhiz等专业的提供测试服务的公司提供的测试的技术、过程。这相当一部分原因是因为国内外的软件技术发展趋势的不同与SaaS软件并没有在国内得到很好的发展。

1.3 研究内容和论文结构

（1）本文在研究了测试方法与自动化测试的基础上，对基于行为驱动开发的测试设计进行了研究，并依据Cucumber框架对基于BDD的自动化测试系统进行了设计与实现。，并着重介绍了基于BDD风格测试用例的全平台自动化测试web系统的设计。

（2）本文

第二章，先是讨论了测试的必要性以及自动化测试在软件开发中的重要性，再而介绍了BDD（行为驱动开发）的原理与支持BDD的自动化测试风格。

第三章，对前端、后端、测试执行部分开发使用的技术进行了阐述。

第四章，对系统设计理念以及模块进行了阐述分析，包括设计思想与系统结构、数据库设计。

第五章，对系统实现在文件管理层面、模块层面进行了说明。

第二章 软件测试概论

2.1 软件测试的概念与目的

软件测试是对项目给定需求的功能验证，是软件质量管理Quality Assurance（品质保证）的重要手段。测试人员通过特定的手段，针对软件的功能或模块，按照一定的流程来对软件进行执行并找出其中的错误，这是软件测试的基本方式。

IEEE中给出了软件测试的定义：软件测试是使用人工或自动手段来测试某个软件的过程，它的目的在于检查软件是否符合规定并满足需求，或者是对比并找出被测软件的预期行为与实际行为之间的差别。

可以看出，软件测试主要是为了衡量系统是否符合预期。换言之，如果软件系统与用户需求不符，则可判定其中存在缺陷，因为用户需求并不能以系统意志所转移，所以系统将需要得到修正。但是，软件测试的目的并非是追求完美切100%无错的系统，在测试人力、资源上的投入需要满足供需关系，当系统已经趋近无错时，发现漏洞与问题已经愈发困难；而投入过多精力在测试上，则有可能导致不必要的成本增长。

软件测试的通常方式是设计测试用例，并比较实际行为或输出结果与预期行为或预期输出结果之间的差异，且此差异是否在用户可以接受的范围之内。在测试执行的阶段，需要注重测试过程中的缺陷，对用例的执行情况进行评估，以求为发现系统的边界值等极端错误进行合理的测试，来设计优秀的测试用例。

2.2自动化测试

自动化测试是软件测试中的一个有别于手工测试存在的概念，通过相应的测试工具或测试脚本来运行大量测试用例以完成测试计划。它能够完成许多手工测试难以做到的测试方法，并且能够自动地生成测试报告与结果，提示错误信息，从而节省了大量的人力资源与时间。顾名思义，自动化测试是通过机器取代人工，以拟定的方式运行程序，并对程序的结果进行比对以达到测试的目的，从而提高软件质量与可靠性的测试方法。

软件测试自动化有以下几个优点：

1. 减少重复性的测试，节约测试成本。在项目的版本迭代中，或是在项目新功能的添加中，由于代码存在着一定的耦合性，代码之间有可能在一定程度上存在着互相的影响。例如在开发中，由于数据库设计时没有全面的考虑，导致在新功能增填时需要修改数据库的字段，从而导致旧代码可能存在着一些问题。所以，在版本的迭代中，测试人员常常需要对系统进行多次的回归测试（对项目功能进行重构后，对系统进行重新测试以确认此修改没有引入问题或带来其他的影响），而在大型系统中，完整的软件生命周期中回归测试更是不可少的一部分。在整个软件测试的过程中回归测试占了极大的比重，而回归测试在一定程度上可以说是大量的重复性的劳动，它给手工测试带来了许多困难，极大地增加了手工测试的成本与时间的浪费，尤其是在渐进开发与快速迭代开发中，重复地对同一模块进行测试。这使得手工测试的缺陷被放大，而编写代码替代重复性劳动，能很好地适用于此场景。由于自动化测试脚本的可维护性、可重用性，测试开发人员不需要重新开发测试用例对应的测试脚本，只需要让测试脚本再次运行即可。这样节约了测试资源与时间，缩短项目开发周期。
2. 在相当一部分情况下，能够弥补手工测试的不足。例如在对网站、App的负载均衡测试中，人工无法模拟大量的（例如百万级别）访问、使用数据，而在自动化测试中可以通过计算机的执行来模拟出大量的访问，可以对软件进行性能测试或者说压力测试。许多测试工具如LoadRunner即是在程序内部封装了访问的代码，并通过配置决定需要的访问量，来做对软件的性能测试与性能分析。而在多线程并发中，bug往往难以复现，而手工地测试更是基本不可能找出错误，只有自动化测试可以模拟微秒级别的并发问题。所以，自动化测试是对高性能需求项目开发中画龙点睛的一部分。这也是自动化测试开发的主要意义所在。手工测试在人力完成的情况下不足以对软件质量进行十分严谨的保证，而自动化测试在单元测试中，可以提供高覆盖率、高精度的保证。
3. 自动化测试具有很高的复用性。自动化测试用例往往采用脚本技术，而在较多的不同环境下，所需要编写的测试脚本都是类似的，这使得测试脚本具有很好的可复用性，从而使得编写自动化测试脚本所需时间成本降低。而脚本的执行是由计算机来完成，脚本的输入一致的情况下，输出在没有随机的情况下也是一致的，这很好地保证了测试用例的一致性。而脚本代码往往可以保存下来再次使用，可以提供高频率地重复运行来验证软件的可靠性，同时节省了重复执行同一个测试用例的时间，将测试人员的时间集中在新的特征功能中。

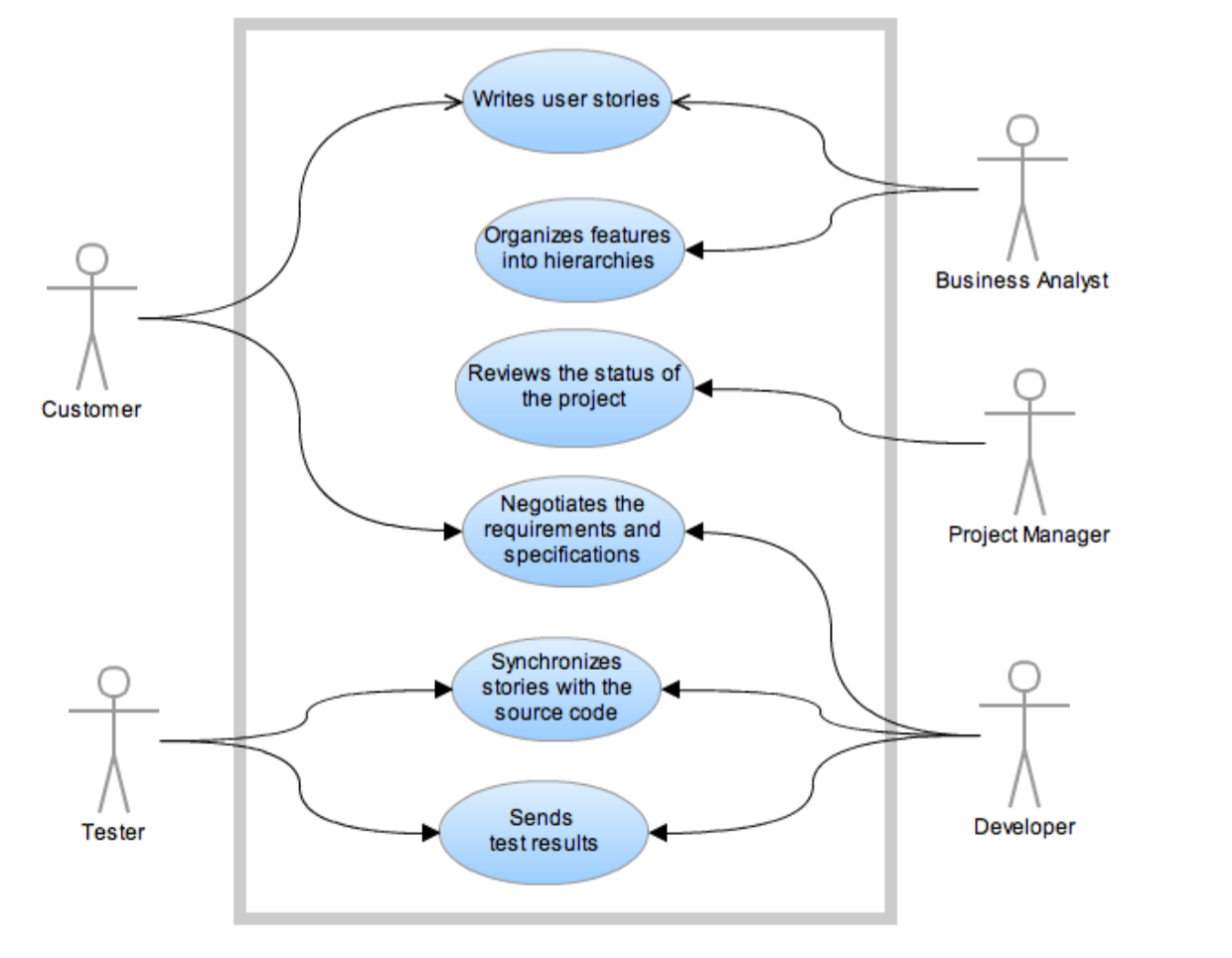
然而，在许多主观性的场景，手工测试还是必不可少的。例如判断web页面中Css样式是否正常、合理，是自动化测试难以完成的。而且手工测试人员在测试过程中可以发现许多测试用例之外的错误或问题，以编写更加完善、有意义的测试用例，这对测试用例为核心的测试过程是十分有意义的。而且，编写测试用例需要测试开发人员具有较好的编程水平，而在某些复杂场景中编写自动化测试脚本往往会花费许多时间，而是否应用自动化，需要妥善的衡量利弊。

综上所述，自动化测试在提高项目质量与可靠性中具有许多优点，但是自动化测试只是对测试的补充。在项目中如何使用、使用哪种测试方法，需要依据具体情况来定义。

2.3BDD与软件测试

基于TDD（测试驱动开发）的思想，本文对软件测试有了新的理解：软件测试是通过执行程序，发现系统漏洞与bug，进行对产品质量的验证，从而对响应的问题发生模块进行研究编码解决，或在执行中发现产品的非错误性不足以驱动改进。BDD（行为驱动开发）是在TDD（测试驱动开发）方法基础上发展的敏捷方法，它鼓励开发人员、测试人员与客户（需求提供者）之间的协作，包括验收测试与客户测试驱动等极限编程的实践，作为对测试驱动开发的回应。2009年，Dan North对行为驱动开发给出了定义：行为驱动开发是由外而内的、需要多方协同合作的、可拓展的、高自动化的第二代敏捷方法。2010年，Chelimsky将BDD具体描述为：行为驱动开发是通过从利益相关者（如客户）的角度描述其行为来实现应用程序的开发方法。

BDD的开发一定程度上借鉴了TDD的思想。TDD是指在编写功能性代码之前，首先选择先编写测试用例，根据测试的需求进行开发。它很好地诠释了“代码即文档”的理念。但是在TDD开发中，由于测试代码往往是程序代码的两倍，并且在开发人员在进行单元测试时，往往是对代码的结构单元，如类或方法进行测试，然而在编写测试时，开发人员往往对代码结构只有一个模糊的想法。此外，测试以来代码结构，使得代码难以重构。

2005年，Astels批评了TDD的不足，并提出了使用行为与规范代替测试，关注软件做什么而不是验证软件是否正确-----即行为驱动开发。行为驱动开发方法提供了一种描述预期行为的方式------使用通用的自然语言描述系统行为，使得项目开发人员与项目的测试人员、需求提供者与产品经理对项目有一致的了解，并允许QA队伍关注系统实现。BDD强烈地关注了客户与客户需求，力求开发出的产品在最大程度上与需求提供者描述的预期相符，从而提高项目质量。 在理想情况下，测试是能够自我描述的。一个好的测试用例应当明确地描述自己要做什么、怎么做，所以，基于BDD开发自动化测试用例，可以让代码本身作为一个可读的自然语言，它定义了用于描述、构造的规范，与测试的理想情况下契合。基于BDD的自动化测试开发不仅在理论上得到了研究，在生产环境中也得到了应用。如2010年开发的RSpec框架，以BDD的风格用Ruby语言编写代码规范，利用了打开现有类并将断言直接并入指定代码（根据Astels给出的建议）的可能性；再如本文中描述的Cucumber框架，在生产中得到了较多的应用。

与其他自动化测试方法相比，基于BDD的自动化测试具有许多优点：

1.基于BDD开发的自动化测试用例有助于生成软件文档。这种方法还可以更好地让开发人员了解测试的目的与目标应该是什么，只需要简单地规划生成文档的规范。它在块（方法或类等）的结构上优化了代码的结构，体现了“代码即文档”的思想。

2.在行为驱动开发风格的自动化测试中，每个期望的行为被定义为具有有关与它的详细信息的描述块。用文本而不是方法名描述此块的选项，为开发人员提供了在将要指定内容的更多自由，使得开发人员能够更好地理解块中的目的，在接口开发中，对输入输出有明确的定义。

3.基于BDD的自动化测试开发中，行为常常被描述为需求并在代码中体现。BDD为测试用例提供了相应的行为的自然语言描述，为开发提供理解性支持，便于重构以及阅读源码。

4.在BDD风格中，自动化测试数据将便于维护。由于在BDD开发风格中，用例与代码将有更紧密的联系，所以在数据的给出中，往往将定义的数据描述在BDD自然语言描述的行为之中，这样分离了数据与代码的耦合，将利于测试数据的维护。

5. BDD方法由于加强团队中各人员的联系，所以在链接自动化测试与手工测试人员的合作中具有极大的帮助。BDD方法在编写行为中，要求行为描述能够被项目中所有人理解，所以自动化测试脚本在提供定义的输入中，可以由手工测试人员负责自动化测试用例的执行。

第三章 项目技术概述

3.1前端技术概述

项目前端使用的是单页应用（single page web application），即在前端中只存在一个html页面，在实现其他与后台交互的过程中完全使用AJAX以实现对前端。前端使用的技术主要有以下几种：

1. React

React的官方网站中声明：React是一个用于构建用户界面的声明性，高效和灵活的JavaScript库。

React由FaceBook开发，起初是因为该公司对市场中的javascript MVC库不甚满意，所以开发了React框架用于架设Instagram网站，并在2013年开源。React让创建交互式UI更加方便，对代码的复用率高，并且性能出众，是一个新兴但是得到了大量应用的流行框架。React的生态圈相当丰富，在npm（前端开源包管理工具/社区）有许多基于React的开源组件，它们在项目中也得到了相当多地应用，在节省编写大量原生dom的时间的同时提升了页面的美观度。

React框架设计的核心前提是认为UI只是将数据通过映射关系转换成其他形式的数据，它用虚拟DOM的思想对数据进行传递，并使用了“状态”（state）来针对具体的渲染目标。用虚拟DOM代替DOM进行DOM操作，大大地改善了在复杂DOM操作时的短板------性能问题。React采用树形式存储虚拟DOM，每当数据变化则重新建立新的虚拟DOM树，并与当前DOM树与之前DOM树进行对比，并只对之间的区别进行真实DOM在浏览器中渲染。React推荐使用Component的方式定义UI的结构，认为UI应该由组件构成。Component树的结构，使得React的Component拥有可组合性、可复用性与可维护性。React还拥有许多开发人员友好的特性，如JSX的语法，能够在组件的模板中将HTML与JavaScript混写，大大地增加了代码的直观性。

React高性能的原因一大部分是解决了DIFF算法低效性的问题。传统计算树形结构转换的最少步骤，复杂度达到O（n^3）。而React采用了对树分层比较、不考虑跨层级移动（在Web UI中DOM节点跨层级操作非常少，且在React中不被支持）、使用key标识相同位置对应关系的方法，与对相同位置节点若虚拟DOM一样则递归向下比较、不同则替换的方法，提高了React渲染的性能，并使以上巧妙的方法结合实际情形，使复杂度降至O（n）。

项目中采用React作为前端JavaScript作为前端JavaScrpit框架，改善了项目前端性能，并使得项目中重复模块得到了很好的代码复用。

1. Redux

Redux是JavaScript的框架，提供状态容器，并提供可预测的状态管理。

Redux的思想是将UI中所有的状态（state）以对象树的形式存储在store中，并使store中的所有数据只能由action改变，通过带有描述性文字的action，匹配reducer对store中的数据进行改变。Redux的优点在于对state进行全局地管理，它避免了数据不同步的问题，并对数据的改变做了规范的管理，使得state能够存放全局数据，减少了与后端的交互。

在项目中，Redux用于做React 中组件变量的管理。在Redux的store中存放了项目中绝大部分数据，用于对渲染结构的规范管理。项目中将对数据的改变映射为发送action的方法，在Reducer中实现对数据的处理，并分发到各个组件中。在React的组件中，由于分发的属性与数据是通过React的Props传过来的，所以React的组件将会得到更新，从而对UI的真实DOM进行改变。

Redux对数据进行了存储、分发与管理，与React精炼的配合对前端逻辑进行了高效、清楚地实现。

1. Swagger-client

Swagger-client是npm社区中的一个开源工具，它生成一个给定指定请求处理程序和Swagger API规范模式的客户端，用于对后端api的访问。

项目中采用了Swagger-client 作为前端中用于访问后端api的工具，它由后端提供api的访问方式，封装了ajax对后台的访问，实现了前后端数据的传输与交互。

Ajax是Asynchronous JavaScript And XML的缩写，它是建立单页应用中不可缺少的技术。单页技术只从后台请求一次HTML页面，而对与后端交互、与后端数据传输，需要用到Ajax作为联通后端的桥梁。Ajax可以访问指定url，并可以携带数据，并得到返回数据，并对返回数据进行处理或者再次渲染前端页面。它避免了每次都从后台中获取整个页面的麻烦，真正实现了异步数据传输请求，可以动态地改变页面的一小部分以减少数据传输的压力。Ajax的核心思想是使用XMLHTTPRequest来实现对后端的访问与数据交换，避免了页面的重载。它在本项目中对比web socket具有很大的优势-----在许多由用户控制是否访问服务器的情况下，对服务器的访问是偶然地、不频繁地，它不需要维持和后端的通信，节省了服务器资源。

3.2后端技术概述。

后端主要使用了Django实现对请求的处理，并使用了git对个人用户的文件的保存与同步管理。

1.Django

Django是一个基于Python开发的MVC模式web开发开源框架。Django被誉为中小型项目开发效率最高的开源web框架，它具有良好的拓展性，并为开发人员提供了许多安全性的支持。

Python语言是一种开发简单、高效的弱类型语言。它支持面向对象编程，具有极为简单的语法与可读性高的关键字，基于缩进的代码结构十分简洁优美，为开发人员减少了编程的工作量，并降低了代码的臃肿性。同时，由于近年来python生态圈的丰富，python被装备了十分庞大的标准库。大量的Python第三方库使得python几乎是全能型语言：无论是桌面游戏的开发，还是web系统的开发，以及科学计算等方面，都得到了完善的支持。

Django项目采用MTV架构。其中，Model层处理数据库的操作，Model类封装了许多查询数据库的方法（如filter方法），基于这些方法，开发人员可以在一定程度上不关注于SQL语句的实现与查询的方法，使用Model的Object类方法操作数据库。而且由于python中\*\*参数操作符的语法，Model方法对数据库的操作更加简洁、可读性更强。由于Django对数据库的强大处理功能，可以在项目中对Model修改，再使用Django自带的makemigrations、migrate或remigrate等工具对数据库参照model进行直接地修改。而Template用于存放页面处理的模板，与JavaEE中的JSP类似，可以接受并渲染后端数据。在本项目中由于是单页应用，所以Template并未得到较多应用。View层对应MVC模式中的Controller层，用于对请求的分发处理，在Controller中根据请求实现相应的业务逻辑，并对请求作出响应，并可以返回Template中存放页面的模板。Django拥有功能强大的URL分发器：基于正则表达式的URL模板，使得Django的URL分发更灵活、拥有更强的可拓展性。

2.Java Play

作为对测试用例中基于Cucumber for Java的基于Java语言测试用例脚本的解析，要求项目中导入Cucumber的第三方jar包对java语言编写的feature文件进行扫描、解析。而Java 的框架，如spring MVC、structs等框架都显得过于庞大臃肿。于是，Guillaume Bort创建了轻量级框架Java play作为对轻巧好用的Ruby on Rail与Django框架的回应 。

Play框架也是基于MVC架构，其中routes文件以极为简单的方式提供了url与controller的映射。

1. Git

Git是由Linus Torvalds开发的一款功能强大、开源免费、设计精巧的分布式版本控制系统。它起初是因为Linus在开发linux内核时由于不满SVN的收费条件，而决定开发的基于命令行的（后陆续推出了图形界面方式）在SVN基础上拥有其他多种功能的git。它起初由于晦涩难懂被人抵制，后随着git的开发深入，对其友好性进行了许多提升，逐渐被开源社区接受。它基于commit本地仓库的思想，可以让开发者提交项目代码更新到本地中。在项目中不仅仅使用git作为项目托管工具，更使用了git对用户执行文件进行了控制。

由于Python集成了git的第三方库，项目中使用python的git模块对用户文件进行操作，这避免了shell脚本难以进行有效控制、维护的问题，并能对个人用户下的文件 git版本状态进行检测。

1. REST

REST（Representational State Transfer，表述性状态转移）是一种web服务设计模式。2000年，REST由Roy Fielding在“Architectural Styles and the Design of Network-based Software Architectures”中提出，定义了一组以URI（Uniform Resource Identifie）为核心，基于HTTP通信接口的规则，利用这些规则为网络资源访问提供了良好的设计模式。而满足这种设计模式的设计风格被成为RESTful架构风格。

论述REST之前，首先要明白URI的含义。URI是URL更高级别的抽象，URI定义了资源，而URL定义了访问资源的方式。REST设计是将互联网资源被标识为URI，作为纯粹的资源地址或者资源名称。在这里，资源作为名词，而URI中不含有动词，作为对资源名词的绝对代表。RESTful架构是无状态的，它具有简单、低耦合的特性与优点。

那么，在项目中该如何对资源进行访问、增删改查？其实，在HTTP协议中早已定义好对资源特定的请求方法。在传统非REST风格项目中，项目大多只使用了GET POST请求方式，并且对GET POST的请求方式作出了非HTTP愿意的区分：GET用于直接访问URL，并在URL中携带特定的参数，而POST可以在提高安全性的同时访问URL，在URL中隐藏提交的数据。RESTful架构中，使用了GET、POST、PUT、PATCH、DELETE等方法对项目中资源进行操作。GET代表获取，对URI的访问只负责获取指定资源功能；而PUT表示添加，对指定资源只进行添加一条记录的操作；而PATCH表示修改，对指定URI指定的资源进行携带数据的修改；DELETE则请求删除URI指定的资源。这样的设计模式简单，并且遵循了HTTP请求方法的原语义。服务器中无需对客户端的状态进行保存，只需要对给定URI请求作出响应，降低了项目代码的耦合性。而这些HTTP方法是可拓展的，如上文中的Patch方法就不是HTTP标准方法，这种特性提高了REST架构的灵活性。

RESTful架构还要求项目中具有统一的接口。在Client与Server之间的交互中，Client调用的Server的接口必须是统一的，这意味着，在调用同一资源时，各个子系统必须提供标准的、一致的、统一的访问方式。这样的设计模式优化了系统接口的风格，在一定程度上减少了各个系统之间通信的压力：由于接口是统一的，在设计、开发中只需要对资源指定URI拥有共识，则在开发中对资源的操作可以变得透明、可猜想、简单。

项目中，还使用了Swagger作为前后端通信的契约。通过swagger，项目可以自动生成REST api接口文档，并可以提供访问页面来查看这个接口文档。Swagger还提供了接口的测试功能，在api文档页面，可以通过页面自定制操作来对api进行测试，同时提供详细的文档说明与示例操作。

3.3 自动化测试部分技术

系统面向的执行流程是使用Cucumber、Lettcue等BDD风格的自动化框架下编写的测试代码，基于Selenium与Appium框架执行。Cucumber框架主要是针对Java语言编写的测试代码，而Lettcue针对Python语言。为了不使本文结构臃肿，只对Cucumber进行描述，而对同样为BDD风格的Lettcue框架不再重复赘述。

1.Cucumber

Cucumber是一个高级别的，基于BDD的自动化测试框架。它通过自然语言描述了独立的、对用户有价值的、可预测的、可测试的用户故事，使用简单、易懂、符合结构化语言的feature文件作为测试文件，而不是让非开发人员难以理解的可执行代码作为测试文件。这样，非开发人员可以读取测试文件，并理解测试文件、甚至在简单的训练过后编写测试文件。这样的feature文件所用的语言称为Gherkin语言。

Gherkin提供可被Cucumber框架解释道的语言。它可以让你在不关心底层如何实现的情况下描述软件的行为，与Python一样使用缩进来定义结构。在以.feature为后缀的文件中，描述了feature的上下文、行动者、动作与目的。基于.feature文件的Gherkin文件特征如下：

login-with-email.feature

1 Feature: GL-2078

2 用Email登录

3

4 @NotAutoAcceptAlert @P1

5 Scenario: 当登录密码错误时弹出警告

6 Given [Welcome Page] Create Account with invalid account Email:"123 @gmail.com" and Password:"12345678"

7 Then [Alert Dialog] Check If Alert Title "" Content "Password must be a t least 8 characters and include at least 1 letter, 1 number and 1 special c haracter or upper case letter." Button Name "OK" pop up

其中第一、二行描述了Feature的名字与描述，而scenario定义步骤顺序、场景描述。Given与Then是Gherkin的关键字，映入了定义给定场景的上下文的步骤。Scenario由Step构成，每个Step由Given或When、Then等关键字开始，Step（6，7行）用自然语言指定了Scenario中执行的操作。

而为了让Cucumber在执行Scenario时知道在运行到Step时该做什么，Step必须有自己的定义文件，以提供Step代码的底层实现，让Cucumber运行自动化测试用例脚本。所以，Step的定义文件------即Step Definition，需要能够匹配step，并给step装配执行的脚本代码。所以，step可以是字符串或正则表达式，当运行Feature文件时，Cucumber将搜索存在的step definition并为Step找到配对的定义，并执行其对应的方法。在Java测试脚本中，它的形式如下：

@When("^\\[Edit Task Page\\] Update Task Title as \"([^\"]\*)\"$")

public void edit\_task\_Page\_update\_task\_title(String arg){

//执行的自动化测试脚本代码

}

Scenario可以用多个标签标记，以@符号开头。这些标记可以为Scenario执行时提供特征标签，可以给出运行的建议、环境、优先级等配置属性。步骤执行是根据Scenario的上下文执行的，它将Scenario作为一个执行模块，步骤们可以共享实例变量与环境变量、系统状态，在Scenario运行结束时重置。

Cucumber为BDD自动化测试提供了简单的规则语法，并对行为进行了自然语言步骤的解释。它使得开发人员与非开发人员使用相同的语言来描述行为，让手工测试人员可以对自动化脚本进行利用。

2.Selenium与Appium

Selenium 是针对web端测试的测试工具。它可以对浏览器进行拟人化操作，达到自动化测试的目的。在项目中，主要针对的是Java自动化测试用例，使用ThoughtWorks提供的开源Selenium Jar包，对浏览器进行操作。简单地来说，Selenium是通过编写Java代码，通过对封装好的Java方法来调用底层js实现，达到操纵浏览器元素的目的，从而让浏览器页面通过脚本规定的顺序运行。

在项目中使用的是Selenium2。与通过使用RC代理劫持请求内容并注入Selenium Core的代码来解决同源问题（跨域脚本不能运行）的Selenium 1不同的是，Selenium2在之前的基础上增加了webdriver的支持。Webdriver使用浏览器提供的原生接口，实现对界面UI的拟人化操作。

Appium是针对移动平台对移动平台应用（包括web应用）进行测试的工具。Appium基于Selenium WebDriver实现，所以在本项目中，Selenium Grid通过将移动平台注册为Node，可以控制多手机分平台测试。这实现了项目的全平台测试功能，基于Selenium Grid的测试开发可以对多种平台进行自动化测试，再基于项目对自动化测试用例的管理实现各自平台脚本的参数配置。

第四章 系统设计概要

4.1系统设计思想概述

本系统是针对BDD风格的自动化测试框架（主要是Cucumber与Lettcue框架）编写的测试用例进行管理、组合执行的一个Web系统，目的是在让手工测试人员通过此系统编辑测试数据、步骤、平台，并能够对自动化测试用例的组合在配置的属性下执行。

BDD风格的自动化测试工具Cucumber与Lettcue（后文皆使用Cucumber作为样例分析）拥有用文本描述测试用例（即.feature文件）并用编写好的自动化测试脚本解析文本的特性。针对此特性，对已经编写好的Step Definition进行扫描，得出一系列的Step模板，再根据这些Step模板，编写已经被解释过的feature文件，达到对Step的复用。

根据Step进行组合、编写的集合被称为Scenario。在Scenario中，Step共享实例变量与环境变量、系统状态，对被测项目进行一组测试。一般地，一个Scenario对应一个测试用例，而往往用户希望同时执行多组测试。所以，系统将Scenario集合起来，作为一个Test Plan，再配置执行参数，对软件进行一些列测试。

以上简短地描述了系统的设计思想。其中，大部分的系统具体设计将在后文系统结构中描述。

4.2系统结构

系统采用了BS架构，利用了web项目不需要预先安装的特性，对随时、随地进行自动化测试的理念进行了贯彻。

与传统BS架构不同，本项目把相当一部分的业务逻辑放在了前端，这也是对目前前端技术发展日新月异的回应；而后端部分利用Rest Api架构，对前端提供了只经过简单处理的数据或未经处理的数据。这种架构的原因在于：1.让前端进行数据处理减少了服务器端的性能损耗，后端无需计算请求对应的页面，一定程度上加快了服务器响应的速度；2.由于计算机的发展，个人电脑拥有更高的性能，在浏览器中计算并不会对浏览器造成负担。

项目主要分为两大模块：测试编辑模块与执行模块

4.2.1测试编辑

测试编辑模块是区分执行模块的的对测试用例、计划进行剪辑的模块。测试编辑又分为三大模块：feature文件的编辑模块（Test Design）、Test Plan模块（Test Plan）、Feature管理模块（Feature List）。

1）Test Design

Test Design模块 提供了对已有feature文件的功能。其中，Test Design将Gerkin语言变得更加简单-----只需在页面中通过对UI的操作，经过人性化的引导，即可通过对Test Design页面进行操作，达到编写feature的需求。

Test Design使用了层级结构。在一个Feature Tree（通一个被测项目中的所有Feature文件）中，包含了许多Feature，而在一个Feature，包含了许多Scenario，而Scenario中包含了许多Step与Tag。其中，Feature层次包括了对Feature名字与描述的修改，对应Cucumber框架中.feature文件中的“Feature:”后跟的feature名字与其通过换行来表述的后一行跟着的描述。而在Scenario层中，Scenario拥有名字与tag。其中，名字对应Scenario名字对应.feature文件中“Scenario：”后跟着的名字，而tag则对应以@为前缀的特征标签，可以给出运行的建议、环境、优先级等配置属性（见3.3.1cucumber框架部分），而其中的Step对应 “When”、“Then”等关键字前缀的可以被执行的步骤自然语言描述。Step则是基于已定义好的可执行的Step模板下在填充数据后的文本描述。

在对Test Design进行编辑修改后，提交到服务器中，即可完成对feature文件的修改。Test Desigin模块的主要功能如下：

1. 新建Scenario
2. 在Scenario与Scenario Outline中互相转化
3. 对编辑后的feature整体进行对比
4. 保存与撤销修改
5. 对Scenario中步骤进行添加、删除、修改

2）Test Plan

故名思议，Test Plan是执行部分将要执行的计划，是测试编辑系统可以用于执行的最终产物。

Test Plan模块主要是对Feature中的Scenario进行选择添加，形成可被执行的集合。Test Plan子系统主要是测试计划列表以及它的详情组成。在每个测试计划中，拥有一或多个Scenario，在测试计划详情中可以看见Scenario列表。

Test Plan是可以执行的一系列Test Case的集合，每一个测试执行与一个Test Plan对应，测试计划是Test Design与执行模块之间的中间件，通过修改Test Case（Scenario）来改变Test Plan，或修改Test Plan中的Scenario顺序、增添Test Plan中的Scenario，都将直接影响到测试的执行模块。

Test Plan子系统包括了对测试计划的管理与修改工作，同时提供了通向单个Scenario的Feature文件的Test Design接口。

3）Feature List

Feature是对Feature进行管理的子系统。在被测项目中，通常有许多个Feature，而手工测试人员在这些Feature中的修改或增加Feature与删除Feature将在映射在Feature List管理系统中。

Feature是Cucumber框架中.feature文件（简单、易懂、不含有自动化测试代码、以Gherkin为结构化语言的执行范本文件）的抽象。在系统中，对.feature文件进行了解释、转换，将Feature的结构以名字、描述等字段分类存储在数据库中，并对个人用户展示不同的、自定义的Feature List作为用户的个人编写的Feature集合。

其中，Feature拥有分类关系。在对待测项目的功能测试之前，需要对测试Feature进行依据功能的分类，以对Feature进行直观性的管理。

4.2.2测试执行

测试执行模块是对提供的.feature文件进行执行的过程。在系统中，对测试编辑的结果在保存后应该对服务器中的个人账号下的文件进行修改，以便在Cucumber（由于Lettcue可以认为是Cucumber的Python实现，故以Cucumber作为样例）中执行。.feature文件提供的步骤是Cucumber执行的基础。由于在以.feature为后缀的文件中，描述了feature的上下文、行动者、动作与目的，Cucumber通过扫描配置的Step Definition库，使用正则匹配来装备Step并执行相应测试用例。

测试执行的模块中，提供了测试执行的状态、时间等属性，对正在执行的测试用例进行监控，并可以终止正在执行的测试计划。

4.3数据库设计

4.3.1数据库概念设计：

数据库中包括Category、Feature、TestCase、Tag等实体表。

4.3.2数据库逻辑设计

1）Feature表

Feature表存储了.feature文件的抽象模型。它存储了.feature文件的文件名信息，以供在需要查看此Feature文件的Web UI时，在服务器中找到此文件并把它解释为界面友好的配置属性。

它的表结构如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 类型 | 是否是主键 | 是否可空 | 简介 |
| id | int(11) | 是 | 否 | 唯一标识符 |
| name | varchar(512) | 否 | 否 | Feature名 |
| Description | varchar(512) | 否 | 否 | Feature描述 |
| type | varchar(64) | 否 | 否 | Feature类型 |
| filename | varchar(255) | 否 | 否 | Feature对应.feature文件名 |
| deactivated | tinyint(1) | 否 | 否 | 是否被删除标识符 |
| category\_id | int(11) | 否 | 否 | 类目id |
| occupied\_by\_id | int(11) | 否 | 是 | 被占用用户id |
| project\_id | int(11) | 否 | 否 | 项目id |

2）TestPlan表

TestPlan表存储了测试计划的基本信息，同时存储了其中包含的TestCases。由于TestPlan与TestCase是Many To Many的关系，故另存了一张TestPlan\_TestCase表来处理他们之间的映射关系）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 类型 | 是否是主键 | 是否可空 | 简介 |
| id | int(11) | 是 | 否 | 唯一标识符 |
| name | varchar(512) | 否 | 否 | 名字 |
| Description | varchar(512) | 否 | 否 | 描述 |
| is\_public | tinyint(1) | 否 | 否 | 是否公开 |
| is\_active | tinyint(1) | 否 | 否 | 是否可用 |
| deactivated | tinyint(1) | 否 | 否 | 是否被删除标识符 |
| release\_name | int(11) | 否 | 否 | 版本名 |
| type | int(11) | 否 | 是 | TestPlan类型 |
| project\_id | int(11) | 否 | 否 | 项目id |

3）execution表

execution表记录了TestPlan的执行状态，它根据Test Plan构建。

表结构如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 类型 | 是否是主键 | 是否可空 | 简介 |
| id | int(11) | 是 | 否 | 唯一标识符 |
| status | smallint(6) | 否 | 否 | 状态 |
| executor | varchar(50) | 否 | 否 | 执行引擎 |
| starttime | datetime(6) | 否 | 否 | 其实时间 |
| endtime | datetime(6) | 否 | 否 | 结束时间 |
| workspace\_id | tinyint(1) | 否 | 否 | 执行的workspace id |
| project\_id | int(11) | 否 | 否 | 项目id |

4）TestCase表

TestCase表存储了Scenario的信息，如 Scenario的Feature名、对应Tag等信息。由于构建TestCase的Step等信息都由Feature的文件名在workspace中扫描而得到，所以TestCase只需要记录Feature名与Scenario名字、描述即可。

由于TestCase与TestPlan是Many To Many的关系，故另存了一张TestCase\_Tag表来处理他们之间的映射关系，这样的设计降低了耦合性，符合数据库的第二范式。

5）Tag表

Tag表存储了Tag的名字、描述、所属project的id这几个属性。在Tag的分类中，可以通过Tag的名字来区分Tag的属性，这需要在前端与后端根据Tag名来确定Tag的属性，所以数据库中不记录Tag类型。

6)Project表

Project表记载了基于待测项目的测试项目的信息。它包括：项目的名字，前缀（即待测项目名字），描述，与驱动的id。其中，项目的名字类似这样：QQ\_IOS端自动化测试，前缀即为待测项目名字。驱动代表的是需要使用何种框架来驱动此项目的脚本的执行，项目中支持Cucumber与Lettcue框架驱动对.feature文件进行扫描与脚本的注入。

以上则是项目所使用的主要的几个实体类数据库设计。

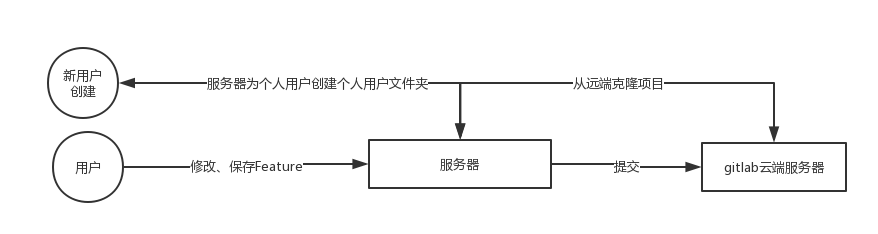
第五章 系统实现

5.1 服务器端用户个人文件管理

系统中，每隔个人用户在远端服务器被分配了个人文件夹。文件夹中包含多个项目，是待测项目的Cucumber框架测试项目仓库，项目仓库中包含：

1. Cucumber框架Step 的定义。Step的定义中含有对基于自然语言描述编写的函数的定义，在函数中，自动化测试脚本将通过以方法体的形式编写。在Cucumber对.feature文件执行时，将会通过正则匹配自动调用这些方法，同时运行脚本，这些脚本是与用户（手工测试人员）无关的，用户不关系步骤怎么实现，用户只关心有哪些已经编辑好的自动化测试脚本，对应的哪些Step模板。用户不能对此文件夹内容进行修改，只能通过此文件夹进行读取Step模板（由Java Play服务器中后台代码实现）。
2. .feature文件集合的文件夹。在web ui中，.feature是以Category分类的形式存在的，而在服务器文件中，.feature文件只拥有一个文件夹，这样通过数据库存储category与feature表的关系，可以大大节省对.feature文件夹的操作以优化性能上的不足。.feature文件夹与用户（手工测试人员）息息相关，用户对feature文件的编辑、增添删除等操作都将直接反应到此。

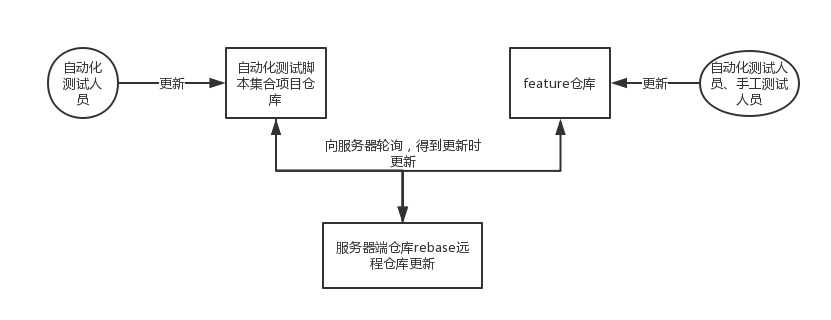
用户的个人文件管理是通过git仓库来实现的。在项目中，使用了git作为用户文件托管工具。在个人用户的账号下，拥有许多的测试执行参数、数据、样本与计划，这些文件是以.feature为后缀的文件，作为测试执行用例参数的编辑结果。在Cucumber框架下，根据已经写好的自动化测试脚本，在此基础上添加.feature文件作为参数配置，可以让测试用例根据用户自定义的数据、顺序被执行。在个人账号创建时，系统服务器端将为个人用户生成以用户名命名的用户文件夹，并push到远端在每一个待测项目的自动化测试项目中创建个人用户分支。如下图所示：



而个人用户下，每个用户都拥有自己的编辑的测试数据、步骤顺序、自己的参数配置等自定义信息，这些不仅保存在数据库中，也需要被保存在.feature文件中被自动化执行。所以，git不仅可以让每个用户拥有自己的分支，通过 git clone的功能将个人初始配置被克隆到项目独立的仓库中，作为自己的个人“文件夹”被编辑修改，还可以让个人账户下的文件被编辑修改时被保存并通过一系列的git命令提交到远端，让项目个人文件存储在云端，有效地防止了服务器可能出现的问题对用户文件带来影响。

使用git还有以下几点好处：

1）对项目的更新时，可以方便地让其更新master，从而基于commit让个人用户对项目文件进行更新。这里项目文件不单单指对.feature文件的更新，在自动化测试人员编写了新的脚本时，也可以对其进行更新作为范本。如下图所示：



2）当对自动化测试文件进行修改时，往往有可能会出现问题或疏漏。Git记载了每次提交的版本信息，对用户行为进行记录。Git lab服务器是公开的，用户可以访问gitlab服务器来查看用户提交的文件，并对每一次的用户行为进行追踪。

其中，向服务器轮询、更新所有的用户文件是通过自己编写的django manage脚本实现的，称为git updater。由于git updater会自主从服务器中轮询是否有更新，在服务器端项目有更改时，git updater会进入每一个个人用户账号，并对远端更新作fetch、rebase操作，对远端服务器的commit与本地合并，再push到远程的个人用户分支。

5.2用户测试编辑模块实现

5.2.1 Test Design模块实现

系统给用户提供了完善的Web界面编辑功能。由于此模块的需求可以总结为对用户的feature文件进行在web ui层面上的修改，所以对此模块进行了如下几种实现：

1. 集成Featre List。在对Feature文件进行编辑时，首先需要一个入口。在Feature List模块中，提供了综合的、强大的对Feature进行分类、增添、删除等管理操作，而在Test Design模块中只需要对Feature的文件进行查看、并提供点击跳转功能。所以，在对此段功能实现时，通过后台中已经实现的跳转接口，将前端的Feature List组件进行了简化、复用，并将其放置在页面的侧边栏中，通过开关开启关闭。React为组件提供了十分友好的复用性支持，所以在Test Design中调用了Feature List的React Component，对Feature的集合进行查看。
2. 自定制路由。对Feature的访问的方式，React-Router提供了一个优秀的方式------通过链接来访问由id标识的Feature。通过Django的正则路由机制，服务器将给定返回Test Design模块，而React-Router组件可以对路由进行参数化定制，通过接受路由中携带参数，对路由进行了访问的同时，向服务器请求参数对应的Feature文件，解析后渲染页面。
3. Step List。Step List是通过Cucumber框架的扫描功能，在Java服务器中扫描个人用户测试项目中Cucumber框架的Step定义文件夹，得出已经给出定义的Step模板集合。这些Step模板是可用的，由开发人员 “装备”了测试脚本的Step。个人用户在对这些模板填上合适的测试数据之后，这些Step便成为了.feature文件的一部分，可供执行。在本项目中，Play框架下的Java服务器提供了一个接口，在项目需要扫描Cucumber框架下的自动化测试用例模板时，通过前端访问Django Server端口，再由Djiango Server访问Play Server，Play Server扫描得到大量的数据，发送到Djiango Server，再由Django Server发送到前端作为相应。此处充分利用了Java的性能高于Python的特点，对项目进行了分布式的开发。
4. Scenario Outline。Scenario Outline是对Scenario的补充。在许多场景下，用户不仅仅需要对一组数据进行执行，有时需要对多组数据进行实行，如边界值测试、等价类测试等。在这些情况下，scenario outline提供了Example关键字，用于对多组数据进行集合测试。在集合测试时，Cucumber会扫描feature中的Example数据集合，并对其进行执行。在UI中，使用了相当多的字符串处理，在其为Scenario Outline时，为Step数据变量提供尖括号<>包含，来标识其为Outline数据变量。
5. Feature与Web UI中的映射。本部分功能采用了以下形式实现：在完整的Feature文件通过服务器传来时，由前端对它进行Parse解析，将它解析为JSON格式的数据。通过此数据方便渲染各个模块的描述。Feature的JSON格式数据被存在Redux的Store中，并保留副本，以实现回退操作。当对Feature文件进行Web UI上的修改时，实则为对JSON格式的Feature数据进行修改，在保存时，将JSON数据传至后端，通过后端的解析格式将其转换为.feature文件的文本形式进行保存。在后端接受数据的同时，通过Model实体修改数据，通过作出相应的修改，以保持数据的同步。

5.2.2 Test Plan模块实现

Test Plan是Scenario的集合。在Test Plan中，主要是对Test Plan的增添、删除、修改工作。

Test Plan拥有自己的Test Case（Scenario）集合。Test Plan的列表中，展示了Test Plan的基本信息，如是否公开、版本信息等。Test Plan提供了查看自身Scenario的接口。

在查看Test Plan的cases接口中，是对单个Test Plan的Scenario进行修改的子系统。在这部分模块中，集成了Feature List的React Component，并提供了点击则将其加入cases列表的功能。Test Plan 的Scenario列表中，每个Scenario都有自己的Feature查看入口，通向Test Design功能，以具体查看或者修改此Scenario。这是借用了Test Design的路由系统工作的。

5.2.3Feature List 模块实现

Feature List是对Feature文件列表进行管理。在Feature List的实现中，与单个Feature文件不同，它并非是扫描服务器端个人用户项目Feature文件夹，而是直接从数据库中读取数据。从数据库中读取数据可以有效地防止当项目feature文件集合过大时导致的性能问题。并且，此部分项目中使用python 的restframework模块，以对Rest架构回应。这是python的第三方类库，可以为项目自动生成标准的对model定义的资源的访问以及各种增删改查操作，这节省了开发人员的时间。并且，这些api是可以被重载的，对api的访问依旧可以如同其他web项目中一样实现相应的自定义业务逻辑。

在对Feature List进行操作时，使用了HTML5的drag drop API，当拖拽文件时，对其放入category的操作进行了ajax方法请求服务端进行前后端功能同步。对category的操作是测试之外的、测试无关的，但是这样的设计大大增加了用户界面的友好程度。

由于Feature是通过id标识的，Feature List 创建了React的组件，在别的模块进行复用，在本身的Feature点击时，使用路由跳转至Test Design模块，而不是重新获取页面。

5.3执行模块实现

执行模块的实现是通过Test Plan的执行按钮向Django后台发送请求，使后台开始执行测试脚本的过程。

项目中使用了Maven对Cucumber自动化测试脚本进行下载与编译待测项目文件。在后台接受前端传来的执行请求时，后台使用Rabbit MQ对请求进行存储，放入消息队列中存储，而Selenium hub接受消息，处理执行请求，并返回响应到Rabbit MQ中，使其得到响应并将请求移除出队列。

同时，项目中还使用了Selenium-Grid作为对分布式测试的支持。在系统需要发布之前，需要对兼容性问题进行严格的测试。在不同环境下的不同情况，如Linux、Windows操作系统之分，还有不同浏览器兼容问题，都需要严格、全面地检测。所以，在服务器中进行处理执行的请求，并分发到远端其他机器进行测试用例的执行，这样节省了服务器的性能，并可以处理多种不同的环境配置，同时在多台服务器上运行可以节省执行时间来配合紧急的项目发布。在服务器中，启动Selenium hub，并注册远程机器为Node，同时在远端机器中安装执行环境，以对测试用例进行执行。

7 总结与展望

7．1总结

本文对软件测试进行了探讨，并深入地关注了BDD与软件测试之间的关系，对基于BDD的自动化测试进行了研究，并依据此阐述了基于BDD的自动化测试平台系统设计。

系统依据BDD风格自动化测试框架的特性，进行自动化系统的搭建，在自然语言描述的步骤中，通过浏览器界面的形式，为用户提供了编辑测试条件、数据与步骤的功能，让不会编程的QA人员拥有运行自动化脚本进行系统测试的能力。

本系统中有以下特点：

1. 采用Rest架构，为系统提高了可维护性，在良好风格的接口中，为系统提供了稳定、易于重构的支持。
2. 使用了云托管服务，将用户的个人测试文件存储在云端，并提供了跟踪、剥离版本的功能。
3. 分布式测试，使用户可以在多种环境下对待测项目进行测试，并减少了时间的浪费。
4. 高安全性。系统解决了潜在的CSRF安全问题，并当系统文件被破坏时，用户文件仍然不会被影响。
5. 实时性。在测试开发人员对测试项目有新的提交或者项目有新的版本时，系统将会自动获取新的提交与通过版本号标识的新的项目版本，对其下载并编译。

7．2展望

由于时间所限，系统仍有许多不足可以完善提高。在许多功能的实现上，当用户对用户文件进行修改时，用户文件修改多了上传到git的步骤，使得响应的速度不尽人意。并且系统的自动化测试脚本数目非常大的时候，巨大的数据量对Cucumber的扫描也带来了沉重的负荷。对此，系统仍需要作如下研究：

1. 对巨大数据量的扫描应当越少越好，将扫描结果存放在数据库中，通过对数据库的访问，并实现延时加载，以对大数据量的情况提供性能上的优化。
2. 对执行的管理应当继承更多的工具，使得执行过程拓展新的功能，如截图等。
3. 许多管理员的功能实现是通过在后台处理实现的，需要在UI界面上拓展上传待测项目等管理员功能。
4. 对浏览器兼容性问题的解决。系统前端在设计之初只基于了Chrome进行开发，对浏览器的兼容性问题没有考虑。

致谢

在本次设计中，首先要感谢我的论文导师沈刚对我在论文上的帮助，尤其是对我在邮件上提出的问题总是提供快速的解答；