**面向实时协同Web前端开发环境的代码重构机制与工具**

摘 要

实时协同编程环境是解决多个程序员协同开发同一项目的方案之一，相比现在主流的版本控制方案，实时协同环境的操作传送与集成都是由底层系统自动完成，对程序员更加友好。但是，实时协作在提供一系列好处的同时，也带来一个严重问题：每当一位程序员对源代码的某个代码域（code region）作了符合语法规则的更改但由于其他部分的源代码对该代码域有依赖而产生语法错误时，其他程序员无法开展正常的编译与调试工作，给编程工作的质量与效率带来负面影响。

为解决上述问题，本项毕业设计考虑在实时协同环境下、以web前端开发环境为运用场景，分析主要包括代码调用重构和文件引用重构两个模块的代码重构机制，以保证每当程序员对源代码的某个代码域作了符合语法规则的更改时，协同工作系统自动地对那些依赖该代码域的源代码执行关联更新。最后在此重构机制基础上以前端开发编辑器atom插件的形式实现这个重构工具。

**关键词：**代码重构，web前端，实时协同，atom

**Code Refactor Mechanism and Tools for**

**Real - time Collaborative Web Front - end Development Environment**

**ABSTRACT**

Real-time collaborative programming environment is to solve a number of programmers to develop the same project together. It is more friendly compared to git because the operation of the transmission and integration are automatically completed by the underlying system. However, there is also a serious problem when real-time collaboration provides a series of benefits: Whenever a programmer made changes to the source code in accordance with grammatical rules, but there are some syntax errors in other parts of the source code because of dependence, so bring a negative impact to the quality and efficiency of programming work.

To solve the above problems, this project is considered in the real-time collaborative environment and using the web front-end development environment as scene, then analyzes the code refactoring mechanism of two modules, including the code call refactoring and the file refactoring, to ensure that the collaborative system automatically performs an association update for source code whenever a programmer makes a change, finally implement this refactoring tool using atom plugin.

**Key words:** code refactor, web front-end, Real-time collaborative, atom

目 录

[1 引 言 1](#_Toc484390868)

[1.1 项目环境 1](#_Toc484390869)

[1.2 项目目标与用途 1](#_Toc484390870)

[1.3 论文组织结构 2](#_Toc484390871)

[2 项目背景及选型 3](#_Toc484390872)

[2.1 实时协同编程环境 3](#_Toc484390873)

[2.2 web前端开发环境 3](#_Toc484390874)

[2.3 代码重构 4](#_Toc484390875)

[2.4 项目选项 4](#_Toc484390876)

[2.5 atom插件开发 5](#_Toc484390877)

[2.5.1 atom开发语言简介 5](#_Toc484390878)

[2.5.2 atom插件安装与使用 5](#_Toc484390879)

[2.5.3 atom插件开发框架 5](#_Toc484390880)

[3 项目需求分析与架构设计 10](#_Toc484390881)

[3.1 功能需求分析 10](#_Toc484390882)

[3.2 用户界面设计 10](#_Toc484390883)

[3.2.1 代码调用重构界面 10](#_Toc484390884)

[3.2.2 文件引用重构界面 11](#_Toc484390885)

[3.3 项目架构设计 11](#_Toc484390886)

[4 项目实现及技术细节 13](#_Toc484390887)

[4.1 代码调用重构 13](#_Toc484390888)

[4.1.1 定义指令触发方式 13](#_Toc484390889)

[4.1.2 构造重构构件 13](#_Toc484390890)

[4.1.3 使用grasp协助重构 14](#_Toc484390891)

[4.1.4 提交重构请求 15](#_Toc484390892)

[4.2 文件引用重构 15](#_Toc484390893)

[4.2.1 定义指令触发方式 15](#_Toc484390894)

[4.2.2 获取当前文件路径 15](#_Toc484390895)

[4.2.3 构造重命名构件 15](#_Toc484390896)

[4.2.4用Redux实现状态记录 16](#_Toc484390897)

[4.2.5 实现重命名函数 17](#_Toc484390898)

[4.3 集成实时协同编程环境 18](#_Toc484390899)

[4.3.1 操作转换(Operation Transformation) 18](#_Toc484390900)

[4.3.2 整合co-editor 18](#_Toc484390901)

[4.4 编写测试代码 19](#_Toc484390902)

[4.4.1 使用测试框架Jasmine 19](#_Toc484390903)

[4.4.2 编写spec代码 20](#_Toc484390904)

[5 实验 22](#_Toc484390905)

[5.1 实验环境 22](#_Toc484390906)

[5.1.1 atom 22](#_Toc484390907)

[5.1.2 服务器 22](#_Toc484390908)

[5.2 功能测试 22](#_Toc484390909)

[5.2.1 代码调用重构功能 22](#_Toc484390910)

[5.2.2 文件引用重构功能 22](#_Toc484390911)

[5.2.3 实时协同编程环境下测试 23](#_Toc484390912)

[6 结论和展望 24](#_Toc484390913)

[6.1 项目总结 24](#_Toc484390914)

[6.1.1 工作量总结 24](#_Toc484390915)

[6.1.2 技术难点总结 24](#_Toc484390916)

[6.2 不足与展望 24](#_Toc484390917)

[6.2.1 文件拖拽 24](#_Toc484390918)

[6.2.2 更友好的重构方式 24](#_Toc484390919)

[参考文献 25](#_Toc484390920)

[谢 辞 26](#_Toc484390921)

# 1 引 言

## 1.1 项目环境

协同编程是软件开发的精髓，随着业界对大规模复杂软件系统的需求持续增长,协作编程在软件开发中也处于越来越重要的地位。统计数据显示，软件工程师70%的时间都是用于软件项目的交流与协作，因此团队开发成员间如何更好地协同工作以确保软件开发的质量显得至关重要，团队间良好的协作能大幅提高软件开发的质量和效率。

目前业界主流的软件开发协作方式主要包括非实时协同编程环境和实时协同编程环境两种。

非实时协同编程环境是一种松耦合的协同，允许一组程序员访问共享的程序设计元素集合(源代码目录及文件)、独立完成程序设计任务,并于特定的时刻合并、同步他们对代码的修改，是一种“复制-修改-合并”的模型。实现这种协同工作的是一系列多样的、复杂的版本控制系统,如 SVN、 Git等，目前已被业界广泛采用。

实时协同编程环境是协同编程的另外一个方案，相比上述以版本控制为代表的非实时方案，实时协同环境更加紧密耦合，它的操作传送与集成都是由底层系统自动完成，无需程序员手动执行相关操作，也不会发生版本控制方案里多个程序员在一次提交中同时编辑了同一文件而导致需要手动解决冲突的额外开销。这种新的方法与技术使程序员只关注开发相关事项不用关注代码合并等繁琐事项，能显著提高软件项目的生产效率与质量。作为一种新兴的协同程序设计范例,实时协同编程环境有着广泛的应用场景,包括:1)某个编程模块工作量太大,以至于单个人无法按时完成,但由于该模块的相互依赖,导致其无法再继续被分解为更小的模块。在这种情况下,需要多名程序员以紧密耦合 的方式工作于同一模块。2)在敏捷软件开发过程中,结对编程是一种可以协助程序员产出更高质量的代码。实时协同环境自然支持结对编程,两位在不同地点的程序员展开虚拟结对编程。3)在教育领域中,实时协同程序可以在一些程序设计课程的实践环节中起到作用,以便学生与讲师一起查看与撰写源代码。

实时协同编程是一种新的方法与技术，具备一系列益处，但是同时也带来一个严重问题：每当一位程序员对源代码的某个代码域（code region）作了符合语法规则的更改但由于其他部分的源代码对该代码域有依赖而产生语法错误时，其他程序员无法开展正常的编译与调试工作，只能持续地编辑源代码，无法尽早通过编译与调试发现问题，背离了程序员应有的工作方式，给编程工作的质量与效率带来负面影响。因此本项毕业设计的主要研究内容是：在实时协同编程环境下、以web前端开发环境为运用场景，分析主要包括代码调用重构和文件引用重构两个模块的代码重构机制，以保证每当程序员对源代码的某个代码域作了符合语法规则的更改时，协同编程环境自动且实时地对那些依赖该代码域的源代码执行关联更新，不影响团队其他程序员正常的编译与调试工作。最后在此重构机制基础上以atom插件的形式实现这个重构工具。

## 1.2 项目目标与用途

Web 2.0时代后，用户对页面的美观与交互易用、开发者对前端项目的架构理解与模式设计都发生了翻天覆地的变化，前端技术与新框架层出不穷。在这种技术快速迭代的趋势下，前端开发受到了越来越多开发者的青睐，然而技术的日新月异也对保持良好的代码规范提出了巨大挑战，为了后续开发和维护的可持续性，很多大型项目不得不选择经常进行重构以适应业界的最佳实践。

同时，业界推出了如atom、subline等很多优秀的前端开发IDE，相比后端开发IDE的“笨重”，前端IDE响应迅速更适应于对网络延时要求高的实时协同编程环境。

鉴于前端IDE对实时协同编程环境更好的支持性以及前端开发对重构的巨大需求，本项毕业设计提供的实时协同编程环境下的这个重构工具可以显著提升前端项目的开发效率与质量。以一个多人小组协同开发以angularJS为基础框架的前端项目为例：

项目组所有成员通过实时协同编程环境，能以紧耦合的方式一起开发该前端项目，在实时协同环境下代码的提交、合并等操作传送与集成都是由底层系统自动完成，因此每个成员可以更专注于自己模块的业务逻辑代码编写，而不用花费大量时间用于项目代码的更新与冲突解决。更重要的是，当angularJS框架发布新版本或者项目结构越来越复杂，项目的架构师a不得不对项目进行重构的时候，使用本项毕业设计开发的重构工具，协同编程环境就会自动地对那些依赖该代码域的源代码执行关联更新，并实时地将改动同步到其它项目组成员的工作空间，使团队其他程序员的编译与调试工作能正常进行，保证了前端项目开发的效率与质量。

## 1.3 论文组织结构

本论文共有六章：

第一章:引言。这一章主要简单介绍了该毕业设计的研究环境,分别分析了协同编程环境的两种实现方案:非实时协同编程环境以及实时协同编程环境的应用场景以及各自的优缺点。然后，在此背景下阐明选择web前端作为运用场景的原因与好处，并以一个具体例子描述了本项目的目标与意义。最后介绍了本论文的组织结构。

第二章:项目背景及选型。本章围绕项目选题,对项目涉及到的运行环境和核心技术作了具体介绍,并从项目需求出发分析了现有的四种主流前端IDE，阐明选择atom作为开发平台的理由。最后,分别从atom插件开发语言、插件的安装与使用、插件开发骨架代码三个方面对atom插件开发作了详细介绍，并着重分析了插件开发的骨架代码各个模块的功能。

第三章:项目需求与功能。在本章节中,从用户的角度出发,首先分析了本项目的功能需求，包括代码调用重构、文件引用重构、集成实时协同编程环境等，并相应地对需求中的两个重构模块做了用户界面设计，同时也根据需求设计了整个系统的架构图，包括基础框架、前端架构、服务器架构、使用的第三份API等。

第四章:项目实现及技术细节。本章是论文的核心章节，在上一章需求分析与系统架构设计的基础上，分别详细介绍了代码调用重构、文件引用重构、集成实时协同编程环境三个模块的具体实现及涉及到的技术细节。最后，以jasmine测试框架为基础，详细阐述了为atom插件编写测试代码的方式。

第五章:实验。这章主要是介绍系统运行的实验情况，首先介绍系统的运行环境，包括atom的版本及依赖环境等，然后分别测试了系统的三个主要功能模块，给出了如何使用系统各个功能模块的详细步骤。

第六章: 结论和展望。在最后一个本章节中,从完成工作量与技术难点两方面对本项目所实现的系统进行了总结,最后基于已实现需求,提出了现有系统的不足和未来的可扩展之处。

# 2 项目背景及选型

## 2.1 实时协同编程环境

实时协同编程环境是一种新兴的协同编程解决方案,这种方案可以帮助程序员实现细粒度的、 即时的交互,以及对并发修改的合并与同步。这种编程环境允许一组程序员以紧密耦合的方式并行编辑同一组源代码文件与目录，且多名程序员对代码的修改可以被即时传送与合并；而且，这些实时的操作传送与集成是由底层系统自动完成的，无需程序员手动执行相关操作。在实时协同编程过程中，多名程序员可以在同一时刻并行访问与编辑共享源代码副本的任意部分，甚至并行编辑同一个源代码文件的内容。如图1所示,实时协同编程环境下,多名程序员可以在同一时刻并行访问与编辑共享源代码副本的任意部分,甚至并行编辑同一个源代码文件的内容。

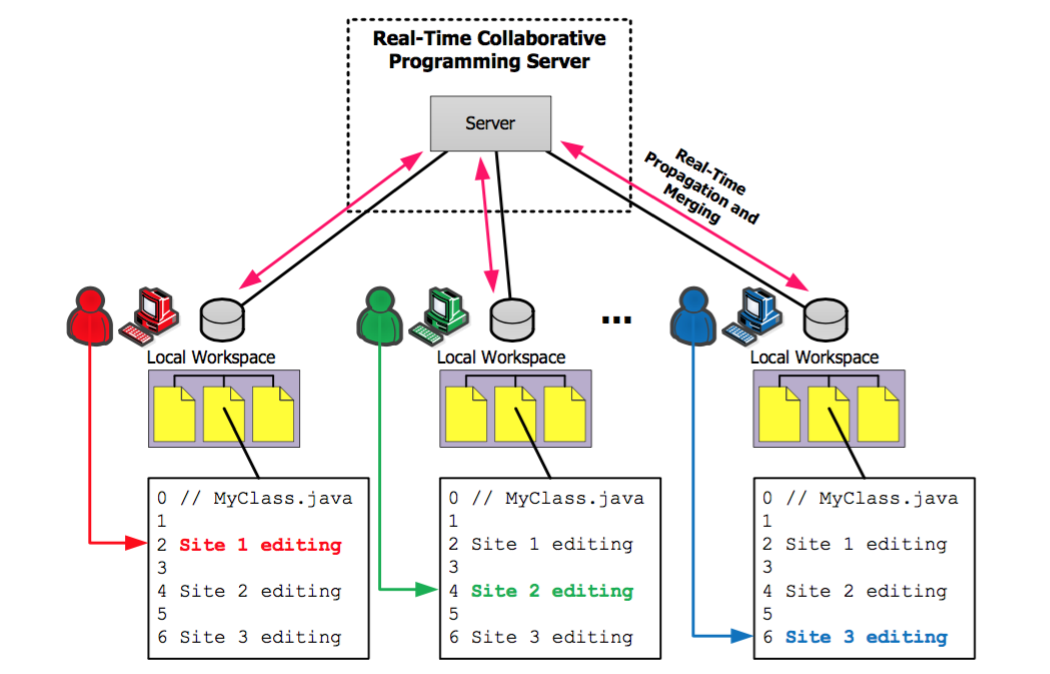


图1 实时协同编程环境下编辑同一个文件

## 2.2 web前端开发环境

web（World Wide Web）即全球广域网也称万维网，它是一种基于超文本和HTTP的、全球性的、动态交互的、跨平台的分布式图形信息系统[1]。随着移动终端的普及与web2.0的强大特性，web应用因其跨平台的特性越来越受到欢迎，不论是windows系统还是Mac系统、不论是dell个人电脑还是iPhone同一个web应用通过浏览器都可以完美运行，这极大减轻了开发者的负担。

Web开发从视图与数据的角度来看也分为前端和后端，前端负责视图的呈现和用户交互，后端负责业务逻辑处理和与数据库交互。随着前端项目的复杂化以及用户终端硬件的变革，开发者对于前端开发设计模式的思考也在经历着快速变革，前端代码的模块化、框架化、按需加载、依赖管理等等都在规范化，前端开发的功能与效率都在不断提升，因此Web前端越来越受到开发者的喜爱和重视。

Web前端项目开发的日益复杂化迫使人们不断提出架构和设计模式的改变，从传统mvc架构到angularJS框架为代表的mvvc、React框架为代表的Virtual DOM等等，而这些框架的出现一方面促进了前端开发的发展提升了前端开发的效率，另一方面也使前端工程像后端工程一样走向了模块化、工程化。相应地，前端项目对于IDE的需求也在逐渐变得更高，这种发展趋势让前端项目也有了代码重构的需求。

## 2.3 代码重构

为了保证软件系统的可维护性以及后续开发的可扩展，代码的可读性与优雅性一直是软件开发中强调的标准和目标。编程不仅仅是编写代码，一次就能写出完美的代码是几乎不太可能的，随着时间的推移与项目复杂度的增加，很多情况下开发者必须重构已编写的代码来修复错误和更改功能。而像代码重构这样大的改动，我们不可能也没必要手动去一一更改涉及到的代码，因此一个好用的代码重构工具对于提升软件开发的效率必不可少。

代码重构机制的核心是自动关联更新（automatic linked update）：每当程序员对源代码的某个代码域作了符合语法规则的更改，项目工程中其它依赖该代码域的源代码自动执行关联更新，避免发生代码依赖错误。

在web前端开发中，代码重构主要有两类运用场景：一是当某位程序员改动了其正在编辑的方法的名称，源代码中（无论是同一源代码文件还是其他源代码文件）所有依赖该方法的代码元素（例如调用该方法的语句）都被自动更新，以保证该项更改不会导致源代码产生语法错误。二是当某位程序员需要重构项目结构而对某个文件路径进行改变时(更改文件名或更改文件位置)，整个项目所有源代码中对该文件的引用都被自动更新，以保证该文件名字或者位置的更改不会导致源代码产生引用错误。

## 2.4 项目选项

完成实时协同Web前端开发环境场景下的源代码智能重构机制的分析与设计后，以此为基础，本项毕业设计将以代码编辑器插件的具体形式实现一个软件工具，以此帮助前端开发者更好地在实时协同环境下进行代码重构。

我们选择了4种现在前端开发主流的IDE：Sublime Text[2]、Visual Studio Code[3]、Adobe Brackets[4]和Atom[5]作为该软件工具候选的代码编辑器，主要从编辑器是否开源、提供的API与文档完善度、开源社区活跃性、编辑器性能等几个方面分析比较如表1：

表1 目前主流前端编辑器比较



Sublime Text优点是性能很好运行速度快，但是不开源且收费，对插件开发者不友好，不利用本项毕业设计。VSCode是微软基于atom改进的编辑器，其设计理念是提供用户一个性能强大、功能完备、质量保证的IDE，而不是鼓励用户自己编写插件扩展所需功能，官方很晚才提供插件开发API，文档与社区支持性都一般，所以VSCode从单纯用户使用角度是很有竞争力的，但是定位与本毕业设计的插件开发也不特别相符。Adobe Brackets 是Adobe公司推出的一款开源编辑器，对Adobe系列软件支持很好很适合偏设计的前端开发人员使用，如可以用Brackets 直接打开一个Psd文件，然后进行颜色和宽度的选择等。Brackets本身提供了丰富的API和详细的文档供插件开发者使用，但是社区活跃性一般不如atom，因此最终本项毕业设计选择Github公司开源的编辑器atom作为插件开发平台。

Atom 原先是Github公司内部使用的编辑器，现在全部开源且鼓励用户按需使用或编写自己所需插件以获得更适合自己的功能。Atom被官方自称为“为 21 世纪创造的可配置的编辑器”，其最显著也是最重要的特点就是插件化架构，对 Atom 而言最重要的是灵活而又完备的插件 API，多次官方新版本发布都在不断改进API与相应文档的支持。atom开发维护团队不仅提供了强大易用的API与使用文档，同时因为与Github深度整合让开源的力量发展迅速，开源社区成长速度极快且十分活跃，包括中文社区都是同类编辑器中最活跃的。社区驱动的方式让atom搭乘上了开源的快车，用户不仅可以共享每年数以万计发布的各类功能的插件，而且还可以按自己喜好改进甚至很方便地开发自己需要的插件。综上，基于atom对Web平台开发者友好对前端更友好，Hackable的显著特色让任何人都能简单贡献自己的插件，最终选择在atom平台下完成实时协同Web前端环境下的源代码重构插件开发。

## 2.5 atom插件开发

### 2.5.1 atom开发语言简介

Atom 文本编辑器是基于 WEB 技术（Chromium+Node.js）开发的，可以理解为编辑器本身就是一个跑在本地的网页。因此我们将完全使用 JavaScript 的 EcmaScript 6 规范[6]来制作插件，至于插件实现功能所需要的样式(如弹出框尺寸、光标选中某代码域后改变该代码域颜色等)可用css或less实现。

### 2.5.2 atom插件安装与使用

在开发插件之前，我们首先得明白atom上插件是如何被安装关联到编辑器。下载好atom后，我们还需要同时安装上apm(atom package manager)才能对atom插件包进行管理。

然后在自己开发或者其他开发者开发的插件代码工程目录下使用apm link命令，即可将此插件link到atom的插件库中(~/.atom/packages)，从而将插件关联到编辑器，实现atom的功能扩展。

安装成功后，在atom上打开manage packages的setting页，即可看到installed packages的数量相应增加，同时在community packages目录下可以看到安装的非atom自带插件的详细信息与插件管理设置功能。

最后启动插件，atom提供4种方式用于启动插件：一是快捷键启动，二是atom控制面板输入相应启动命令启动，三是通过导航栏菜单选择相应插件启动，四是通过右击菜单选择相应插件启动。也可在插件管理页面将插件设置为随编辑器自启动，但是要注意不同插件之间启动命令不能相互冲突

### 2.5.3 atom插件开发框架

atom是如此地鼓励社区进行插件开发，因此对于插件开发者不仅提供完备易用的API与文档，甚至于已经把插件开发框架集成到atom自身，所以我们可以很方便地使用 Atom 提供的工具创建一个新的 package（软件包）来开始插件的开发[7]。

启动atom编辑器后，打开控制面板(command+shift+p)，输入“Package Generator: Generate Package”并点击列表中正确的条目，然后在输入提示框中输入该插件软件包的名称，即可生成插件开发的框架代码。另外，在atom插件开发中需要注意的一点是，每当需要测试我们对软件包的改动的时候，都要重新加载当前窗口以确保 Atom 执行的是我们最新的源代码。atom强大的控制面板允许我们很方便地重载编辑器，在控制面板输入window:reload即可完成。

通过atom自带的package generate生成的插件工程基本结构如图2所示：

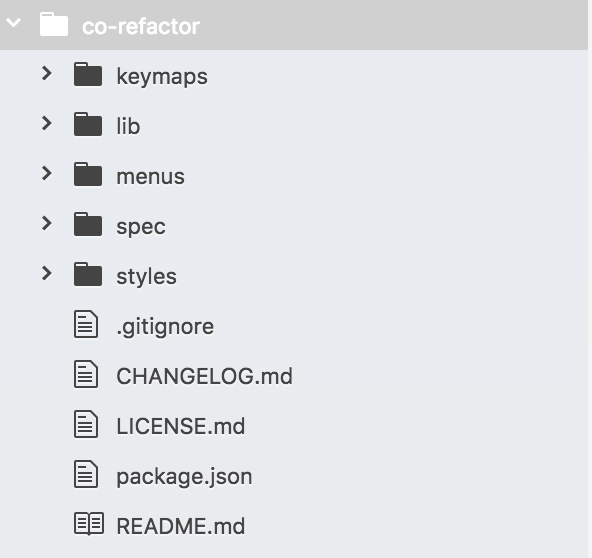


图2 atom插件骨架代码结构

工程主要包括5个目录和package.json文件，用途如下：

1. package.json文件

用于定义主文件位置，主文件是 Atom 软件包的入口文件，Atom 通过设置 package.json 里的main属性来找到主文件的位置，如图3所示：

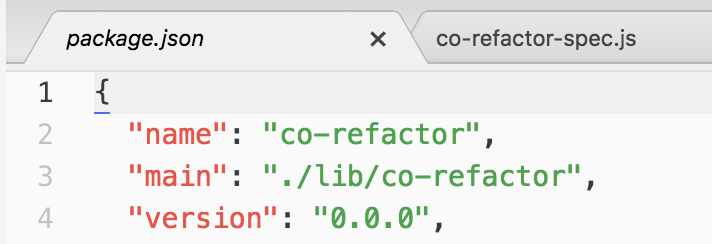


图3 package.json文件

1. keymaps文件夹

用于配置插件快捷键，在keymaps文件夹下自动生成了一个json配置文件co-refactor.json，然后在该json文件指定快捷键与对应指令即可，如图4以下配置可以让用户通过 Ctrl+Alt+O（Windows/Linux） 或 Cmd+Alt+O（MacOS） 来触发指令"co-refactor:toggle"：

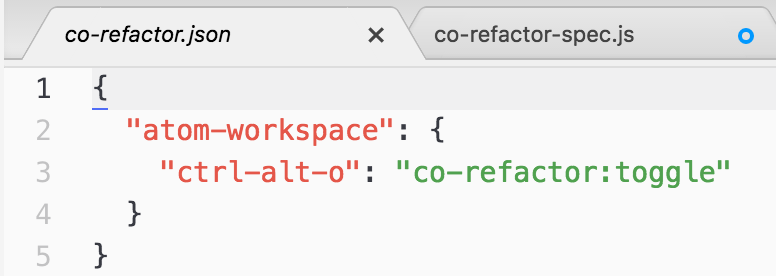


图4 co-refactor.json文件

1. menus文件夹

用于配置右击菜单和导航栏菜单，在menus文件夹下自动生成了一个json配置文件co-refactor.json，其中主要有两项配置内容：一是“menu”  对象用来定义插件的自定义应用菜单，通过 label 属性和submenu属性定义该自定义应用菜单层级，最后通过label 属性和command属性定义点击该菜单后执行的命令，如图5所示：

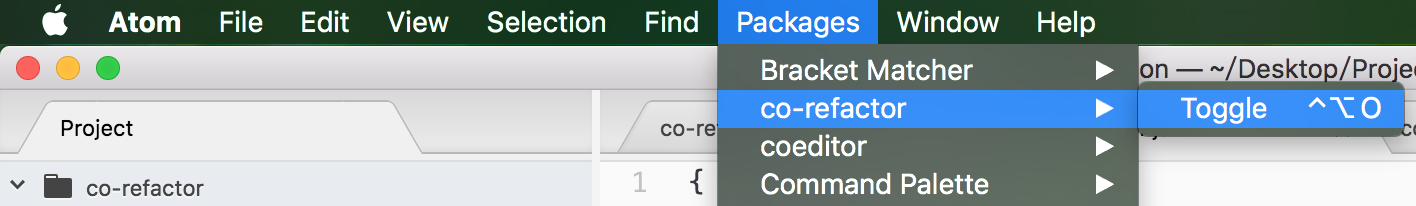


图5 atom导航栏菜单自定义

另外一个是“context-menu” 对象可以让我们定义右击菜单的一些新条目，每一个条目都是通过一个显示在菜单的 label 属性和一个点击后执行的命令的 command 属性来定义的，如图6所示定义右击菜单的“Toggle co-refactor”条目对应指令"co-refactor:toggle"：

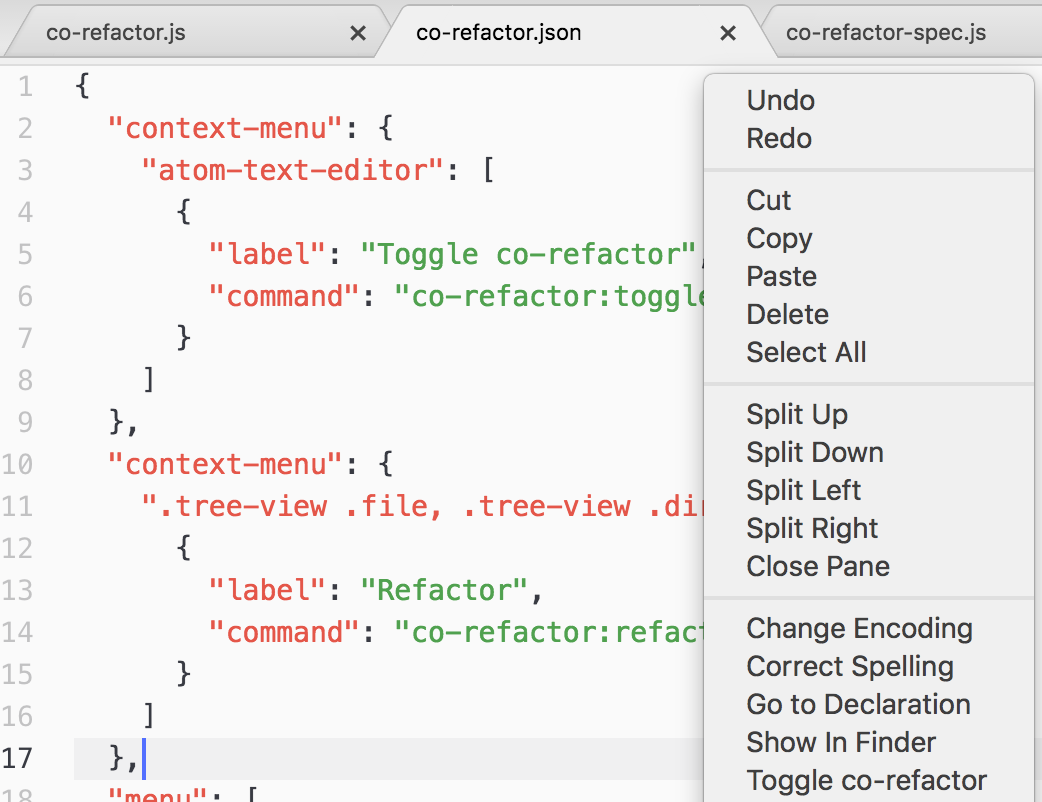


图6 atom右击菜单自定义

1. libs文件夹

主要包括插件主文件和处理业务逻辑的js文件，一般而言在项目package.json文件中通过main属性定义的插件主文件都在该文件夹下，如该generate框架项目就是在package.json文件中通过 "main": "./lib/co-refactor" 定义该插件主文件为lib文件夹下的co-refactor.js。

主文件co-refactor.js导出一个带有生命周期函数（Atom 在特定的事件发生时调用的处理函数）的对象，主要包括4个模块：

1. activate模块

在 Atom 初次加载软件包的时候调用，用来初始化一些诸如软件包所需的用户界面元素的对象，以及订阅软件包命令的处理函数。如图7定义'co-refactor:toggle'命令对应'toggle'函数：



图7 atom定义指令对应函数

1. deactivate模块

在软件包停用的时候调用，例如，当用户关闭或者刷新编辑器的时候。

1. serialize模块

Atom 会在使用软件包的过程中不断调用它以保存软件包的当前状态，然后将返回值在 Atom 下一次加载软件包的时候作为一个参数传递给上述activate模块。

1. toggle模块

为activate模块中定义的'co-refactor:toggle'指令对应函数，也就是插件启动入口函数，后续插件功能扩展由此开始。

# 3 项目需求分析与架构设计

## 3.1 功能需求分析

为了使用户更高效便捷地实现互相协助下的代码重构,我们首先从用户的角度展开系统的功能需求分析。在前端开发场景，最常见的重构主要有两类：代码调用重构和文件引用重构，因此我们的重构系统主要针对这两类场景提供强大的重构功能和友好的使用方法：

1. 代码调用重构:

该功能主要有两方面的需求：函数调用重构和变量重构，即用户选择某个函数或变量进行重构时，该文件中所有对这个函数或变量的调用都会自动关联更新。考虑用户使用的便捷性，用户在选择代码重构时，还应该可视化地告知用户这次重构操作会影响多少处代码，受影响的代码的位置等详细信息。

1. 文件引用重构:

该功能主要也包括两方面的需求：重构单个文件和重构一个文件夹，即用户可以在工程树状导航目录上选择一个文件或一个文件夹，重构这个文件或文件夹的命名、位置，则此工程项目所有引用了该文件的代码都会相应被关联更新。为了对用户使用更加友好，重构完成后也应该可视化地告知用户这次重构操作影响了多少文件。

重构功能完成后，我们再集成实时协同编程环境，将用户对工程的重构操作实时同步到其他参与协同编程的用户终端，实现实时协同编程环境下的代码重构。

## 3.2 用户界面设计

### 3.2.1 代码调用重构界面

考虑用户更改代码中函数名或变量名时的使用场景，该功能设计为用户光标选中重构对象，快捷键启动代码调用重构功能后，生成一个重构构件用于接收和显示重构信息。如图8该构件包括最基本的两个label显示重构对象当前命名和目标命名，为了增强这个工具的可视化，我们还增加一个list控件实时显示当前重构对象及其上下文:



图8 代码调用重构构件

### 3.2.2 文件引用重构界面

考虑用户更改文件名或文件位置时的使用场景，将用户触发“文件引用重构”功能的方式定义为右击文件后的菜单的”refactor”项，如图9所示：

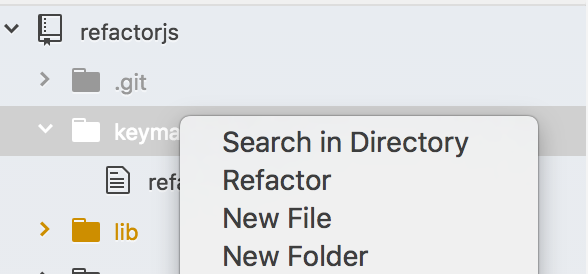


图9 右击菜单启动代码调用重构

用户选择了要重构的文件或文件夹后，右击该目录后选择Refactor选项，然后系统生成一个控件接收目标路径信息，默认显示当前路径并允许用户更改，如图10：



图10 文件引用重构构件

## 3.3 项目架构设计

本系统使用atom官方的generate骨架为项目基本框架，主要包括代码调用重构与文件引用重构两个模块。

其中代码调用重构模块前端使用html原生控件，接收用户的重构对象目标命名输入，然后调用codemod+jscodeshift框架的API完成代码重构；为记录遍历工程所有文件的状态，文件引用重构模块前端使用React-Redux的架构以到达状态与视图的一致性，然后调用grasp框架的API完成代码重构。

实时协同编程环境使用指导老师另外一位学生的项目co-editor[12]，本毕设与该实时协同项目为同一课题下的两个子项目，都是基于atom插件的形式进行开发。该实时协同项目使用node server作为后端服务器，每个用户终端的改动都实时发送到服务器，再由服务器广播给其它所有终端，以到达实时同步。

整体系统架构设计如图11所示：

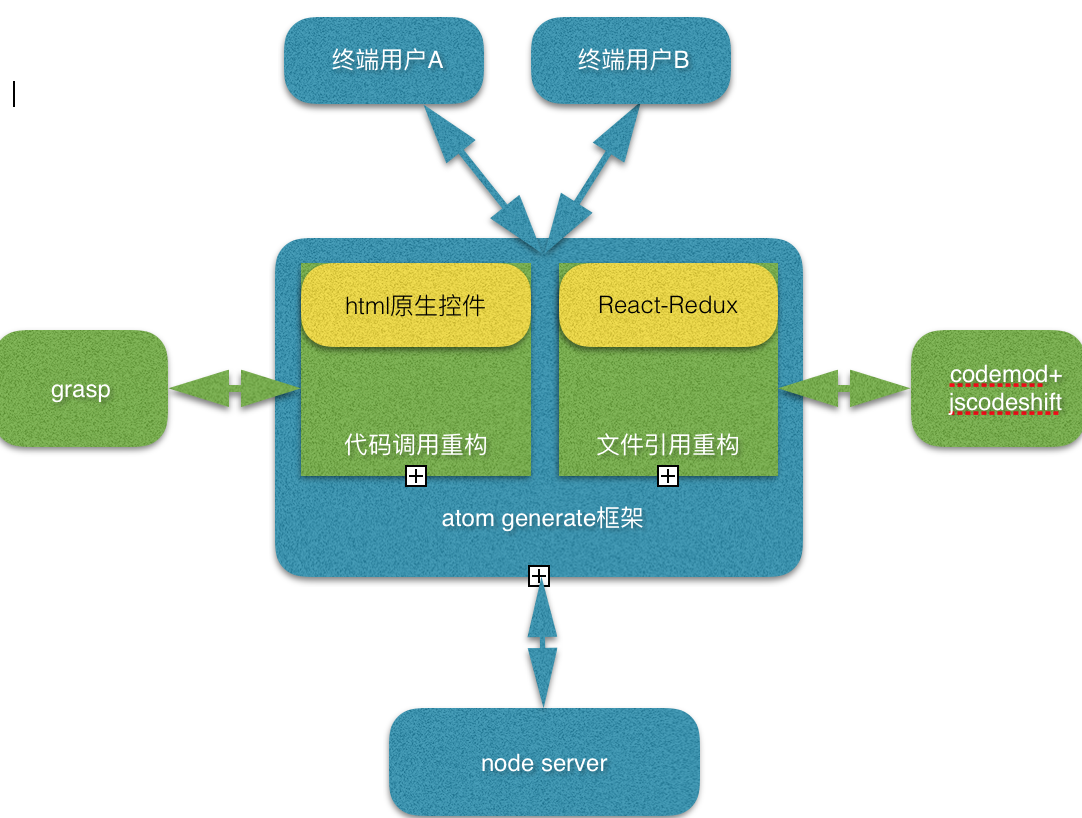


图11 系统架构设计图

# 4 项目实现及技术细节

## 4.1 代码调用重构

代码调用重构功能主要包括两部分：函数调用重构和变量重构，即用户选择某个函数或变量进行重构时，该文件中所有对这个函数或变量的调用都会自动关联更新。代码调用重构主要包括指令触发、获取所选重构对象、构造重构构件、搜索重构对象等模块。

### 4.1.1 定义指令触发方式

考虑用户更改代码中函数名或变量名时的使用场景，使用快捷键的方式是最方便，因此我们选择用快捷键的方式启动重构指令与提交重构指令（注意不同插件之间快捷键不要冲突）。其中用户选中重构对象后使用快捷键Cmd+Alt+R启动重构指令，显示重构构件；用户输入目标重构命名后，使用快捷键Cmd+Enter启动提交指令，执行重构。

对快捷键的配置在keymaps文件夹下的json文件里，如图9指定相应快捷键对应的指令即可：

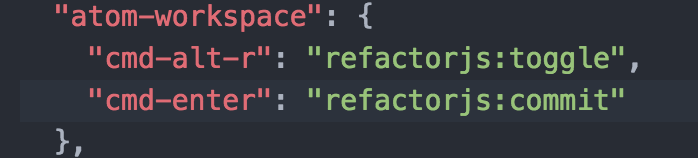


图9 **定义keymaps指令**

然后在入口主文件的activate函数里定义这两个指令的对应执行函数，接下来就可以在对应执行函数里编写所需业务逻辑代码了。

### 4.1.2 构造重构构件

为了方便，定义执行一次快捷键组合Cmd+Alt+R后启动插件，再执行一次退出插件。因此首先通过一个三目运算符this.modalPanel.isVisible() ?exit():start()实现根据插件当前运行状态来启动或退出，若启动则生成重构构件RefactorjsView，显示和输入重构信息。

如图5构件RefactorjsView由两部分组成，一是显示重构对象的当前命名和目标命名的form，另外还有由重构对象在代码中出现位置上下文组成的list。

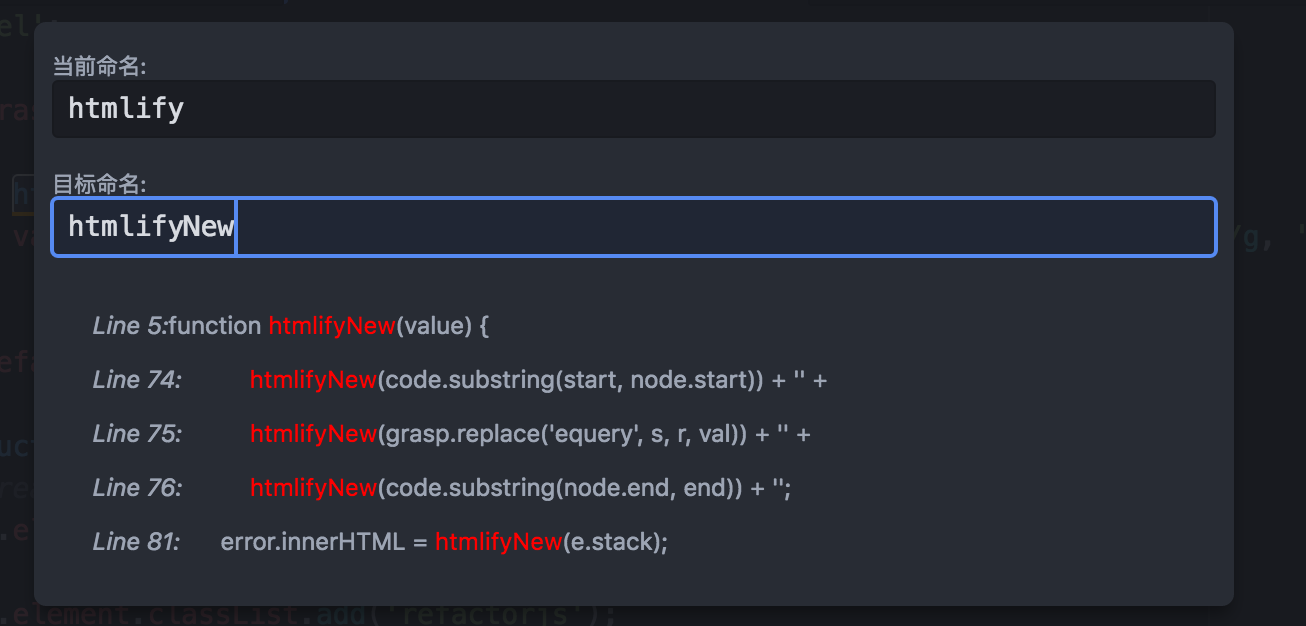


图5 代码调用重构构件

Form控件包括4项：searchText、searchInput、replaceText和replaceInput。其中searchText为一个提示message显示'当前命名:'；searchInput显示通过atom的editor API获取的重构对象当前命名；replaceText为一个提示message显示'目标命名:'；replaceInput为一个可编辑框接收重构对象目标命名，默认显示重构对象当前命名。另外，为了在后面的list组件实时显示重构对象在代码中的位置及上下文，还要将replaceInput的改动事件绑定到list组件：

this.replaceEvent= this.replaceInput.getModel().onDidChange(this.search.bind(this));

最后根据form控件中replaceInput的输入来实时匹配代码，并将重构对象出现的代码上下文实时显示。

### 4.1.3 使用grasp协助重构

使用grasp生成的完全语法树可以很高效地进行js代码的分析，当前最新版本的grasp支持js和css两种语言，因此首先根据API文档选择js的语法树构造模式equery，如图10：

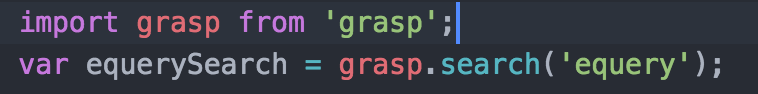


图10

然后获取前面replaceInput输入框的值s和整个文件的代码code，再使用grasp的js模式equerySearch构造出目标代码的完全语法树并得到目标对象s出现的每一个相关上下文node，如图11:

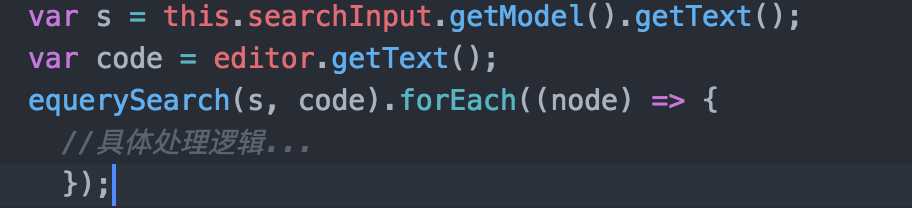


图11

通过grasp得到重构对象在整个代码域中的上下文node后，首先根据接口文档使用node.loc.start.line方法得到该上下文在整个代码中的位置；然后再使用grasp的全局重构方法replace，在equery模式下根据目标对象原始命名s和replaceInput输入的值r实时更改重构对象，为了更直观我们将重构对象颜色设置为亮红色，如图12：

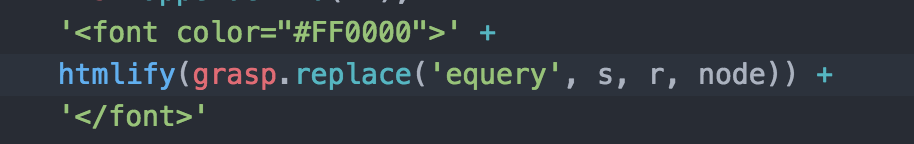


图12

再把重构对象所在代码行相应上下文加入一起显示，方便用户查看该重构对象在代码中的情况，如图13：

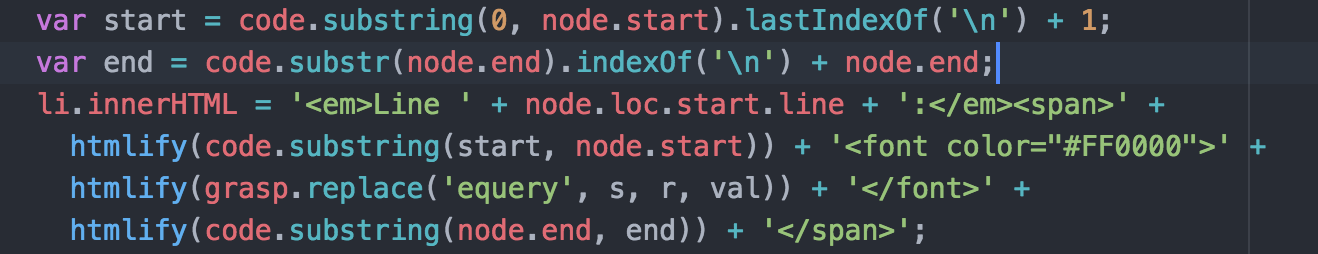


图13

此外，因为atom是基于web技术开发的，在把代码显示到控件时要先把代码html化包括换行符的转换等，我们使用一个htmlify函数协助完成转换代码到html格式，如图14：

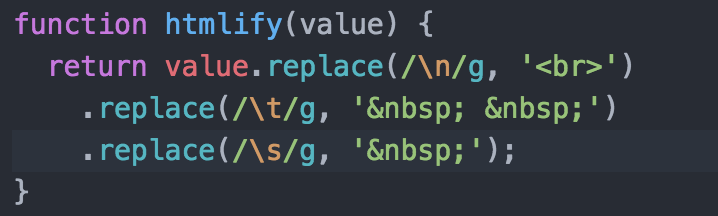


图14

最后把分析每个重构对象出现的代码块node得到的结果都加入到list控件，即可在重构控件里实时显示重构对象所有出现的位置及上下文，方便用户分析、重构代码。

### 4.1.4 提交重构请求

在主文件中，我们已经指定了重构提交指令Cmd+Enter对应commit函数，因此提交请求相关的逻辑处理都在这个函数里实现。

首先获取atom下当前的编辑器editor及editor的整个代码域，然后再根据重构构件中searchInput和replaceInput控件的值得到重构对象原始命名与目标命名，最后在equery模式下调用grasp的全局重构方法replace对代码域完成重构，并将改动设置到editor，如图15：

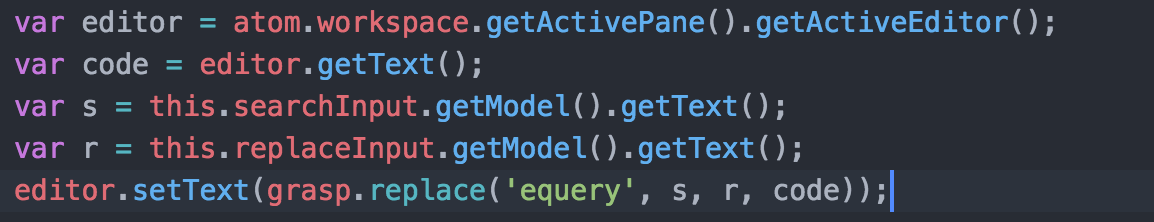


图15

## 4.2 文件引用重构

文件引用重构功能是指，当用户改变一个文件的文件名或位置时，工程项目所有引用了该文件的代码都会相应被重构。文件引用重构功能主要包括指令触发、获取当前文件路径、构造重命名构件、状态记录、文件移动、代码引用重构几个模块。

### 4.2.1 定义指令触发方式

查阅atom的context-menu API可得在文件树状导航栏右击的代码为".tree-view .file, .tree-view .directory"，因此如图相应在menus/co-refactor.json里添加代码即实现将"Refactor"加入到右击树状导航栏中的文件出现的菜单选项中，并将该选项与指令"co-refactor:refactor"关联。

### 4.2.2 获取当前文件路径

atom是基于 WEB 技术（Chromium+Node.js）开发的，因此每个元素本质上都是一个HTMLElement，将点击的文件作为一个 HTMLElement传入相应API即可获取文件路径。

### 4.2.3 构造重命名构件

重命名控件PathRenameForm主要由一个提示输入的label和一个AtomInput组件构成，默认显示传入的路径并允许用户编辑目标路径，然后根据操作选择触发取消或确认函数。

Atom官方鼓励使用React来实现这样的构件，这个构件应包括获取的当前路径、取消操作、确认操作3个属性。Refactor操作触发后在主文件中使用ReactDOM.render将这3个属性传递给该控件，如图16：



图16

PathRenameForm控件接收这3个属性后，首先调用构造函数，生成atom输入框并将属性里面的当前路径设为默认值显示；然后调用onChange函数记录用户当前输入为目标路径，最后根据用户操作选择执行传入属性的确认函数或取消函数：

'core:confirm': () => this.props.onRename({

previousPath: this.props.previousPath,

nextPath: this.state.path,

}),

'core:cancel': () => this.props.onClose()

这部分的主要技术难点是参考atom官方使用了React+Redux的函数式编程（FP）范式[14]。区别于传统的面向对象模式，React+Redux的函数式编程（FP）范式更适用于多交互、多数据源的使用场景，而我们在进行文件名重构的过程正是要记录当前重构的文件路径与欲更改的目标路径，然后与工程里所有文件交互匹配、重构所有引用，所以使用函数式编程更易于实现功能

### 4.2.4用Redux实现状态记录

用户输入目标路径后，若选择确认则触发核心的重命名函数。

重构的过程中，首先要记录当前重构的文件路径与欲更改的目标路径，然后与工程里所有文件交互匹配、重构所有引用，因此使用Redux生成store对象记录每个状态与对应视图。重构开始时传入'refactor-start'状态到store，重构状态为正在进行，然后执行核心函数renamePaths，与工程所有文件匹配结束后，传入'refactor-end'状态到store，重构状态为结束。

首先通过Redux提供的createStore生成Store容器来保存数据：

import { createStore } from 'redux';

this.store = createStore(reducer);

其中reducer是一个函数，它接受 Action 和当前 State 作为参数，返回一个新的 State指明重构是否完成：

export default function reducer(state, action) {

switch (action.type) {

case 'refactor-start':

refactorInProgress: true

case 'refactor-end':

refactorInProgress: false

}

}

在Redux中视图(View)通过store.dispatch() 方法发出 Action到前面的reducer，获取新的状态以更新视图。在本项目中，重构开始时发出'refactor-start'，结束时发送'refactor-end'，视图就会根据reducer返回的状态变化，以匹配所有文件。

### 4.2.5 实现重命名函数

重命名函数renamePaths是本毕业设计的核心函数，按业务处理逻辑可以分为4部分：

a. 判断重命名的对象

用户可选择对单个文件或者整个文件夹进行重构，所以首先要判断重命名的对象并根据所选对象不同执行不同操作。若重命名的对象是文件夹，则遍历该文件夹下所有文件，生成一个json数组保存该文件夹下所有文件的当前路径和目标路径；若重命名的对象是单个文件，则只将该文件当前路径和目标路径存放到json数组。

Atom官方提供的'fs-plus'包[9]囊括了所有文件的基本操作API，查询API文档可得接口**isDirectorySync(directoryPath)**判断所给路径是否存在且是一个文件目录则返回true，否则返回false，应该可用于判断用户的选择是文件目录还是单个文件：

**if (isDirectorySync(previousPath)) {**

**//用户选择文件目录重构**

**}**

**else{**

**//用户选择单个文件重构**

**}**

若用户选择文件目录重构，则首先传入当前目录previousPath，使用'fs-plus'包下的接口**listTreeSync(rootPath)[3]**遍历该目录得到该目录下所有文件路径，然后拼接目标目录nextPath和当前目录下所有文件的相对路径，构成一个由所有原始文件路径prevFilePath和所有目标文件路径nextFilePath组成的数组：

return listTreeSync(previousPath).map(prevFilePath => ({

prevFilePath, nextFilePath:join(nextPath,prevFilePath.slice(previousPath.length)),

}));

若用户选择单个文件重构，则直接将该文件原始路径和目标路径加入数组即可。

b. 判断目标路径是否存在

得到存放所有目标路径的数组后，同样我们使用'fs-plus'包下接口**isFileSync(filePath)**，该接口判断若所给路径存在且是一个文件则返回true。

遍历数组里的所有目标路径，若**isFileSync(filePath)**返回了true则说明该目标路径下已有其它文件，不可进行重构；否则检查通过，进行下一步移动文件。

c. 移动文件

查阅'fs-plus'包文档，atom提供了方便的接口**moveSync(source, target)**用于移动文件。上一步的路径存在检测保障了数组里所有目标路径都是可用的，因此直接遍历路径数组根据prevFilePath和nextFilePath，将所有原始文件移动到目标位置。

1. 重构所有引用语句

被重构文件成功移动到目标位置后，最后一步需要做的就是将项目工程里所有引用了这些文件的引用语句重构。

随着前端界的风起云涌，新功能、新框架、甚至 JavaScript 本身都在快速进化，为了让开发团队保持高效，业界也提出了很多使用重构件（Codemod）加速 JavaScript 开发的想法与框架，其中最优秀的当数Facebook的jscodeshift[11]。jscodeshift 是一个用于在多 JS 文件运行 codemod 的工具包，可用于协助大型代码库的自动化重构。此工具包将 JavaScript 解析为一棵 抽象语法树 ，并在其上进行变换，然后输出符合指定代码风格的新 JavaScript 代码，而且转换过程是用 JavaScript 本身实现的，它提供：

* 一个 runner，它可在每个传递给它的文件之间提供转换，还能输出转换文件的数量。
* recast 的包装，提供不同的 API。Recast 是一个 AST-to-AST 转换工具，并且会尽可能地保护代码的原有风格。

因此我们使用该工具包协助完成引用转换，根据jscodeshift提供的API我们可以很方便地完成代码里的声明引用重构：

首先引入runner：import Runner from 'jscodeshift/dist/Runner';然后调用jscodeshift的声明引用API(ImportDeclaration)，封装为runner函数需要的fileInfo参数，如图17：

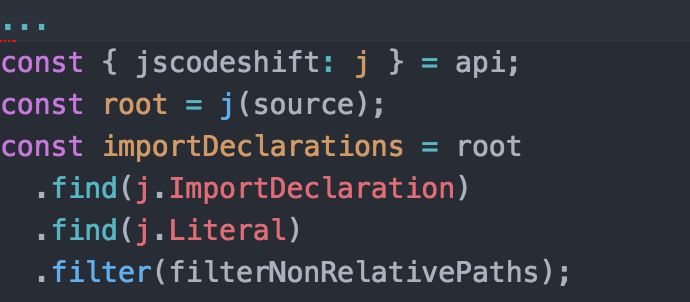


图17

最后将声明引用API、路径参数、用户参数传递给runner函数，jscodeshift即可根据路径参数完成工程里声明引用的重构：

const result = Runner.run(transform, roots, options);

return Promise.resolve(result);

## 4.3 集成实时协同编程环境

### 4.3.1 操作转换(Operation Transformation)

在4.2与4.3中完成web前端开发环境的代码调用重构和文件引用重构模块后，接下来还需要把实时协同环境集成到项目中，以到达实时协同环境下的代码重构。

本毕业设计使用指导老师另外一位学生的项目co-editor[12]作为实时协同环境的基础，本毕设与该实时协同项目为同一课题下的两个子项目，都是基于atom插件的形式进行开发，因此在基本环境集成方面没有大问题，主要难点还是在集成过程中的操作转换(Operation Transformation)上。

操作转换技术是实时协同系统最常用的技术之一，在实时协同中多个终端用户可能同时编辑一个文件，这时候操作执行的顺序就尤为重要，如用户a执行删除第6行的操作与用户b执行在第5行后新添一行的操作同时进行，若执行顺序不对则可能把用户b新添的行误删了。在将重构系统与实时协同系统整合的过程中，也同样存在这样的问题，需要考虑操作的转换顺序。

如用户a重构某函数的命名时，a的终端会向协同服务器同步启动重构的快捷键变化，导致用户b同步这个指令后也在b的终端执行重构；然而，用户a的重构完成后，代码域的改动又会同步到服务区导致用户b的代码域又一次改动。因此在集成整合实时协同系统时，我们要保证操作的转换不影响重构的正确性，即重构的指令和结果不能相互覆盖或重写，为避免这种情况的发生，我们在实时协同系统中就只同步重构的结果(text)，不同步重构指令的事件(event)。

### 4.3.2 整合co-editor

实时协同项目co-editor使用ShareDB[7]作为项目的OT框架，ShareDB框架基于 Node.JS 实现，整合了前后端，可以很方便地自行搭建服务端和客户端。为了保证实时协同过程中重构的指令和结果不要相互覆盖或重写，ShareDB的客户端只监听重构的结果，不监听重构指令的事件，如图18：



图18

启动实时协同环境前，首先需要指定server address、session id和client id，其中server address表示同步服务器的IP地址；session id用于标示文件路径，即第一位编辑某个工程项目的用户对该工程项目设置一个session id，后面的用户输入相同的session id就可以加入到这个工程的实时协同编辑中；client id用于标示用户，当多个用户编辑同一个文件时，光标上不同的client id就表示相应用户。

指定以上参数后，用户就可以开启或加入一个实时协同环境，在这个环境下用户可以进行自己的编辑，同时可以看到有黄色光标表示其他用户也在同时编辑这个文件，如图19：



图19

## 4.4 编写测试代码

### 4.4.1 使用测试框架Jasmine

Atom的测试使用Jasmine[13]作为spec框架，为保证代码质量及便于后续开发和维护，任何新的功能都要拥有specs来防止回归。

Jasmine 是目前最广泛使用的JS单元测试框架之一，该框架不依赖于任何浏览器、DOM，简单易用又功能强大。 一个最基本的jasmine测试用例如图8所示：

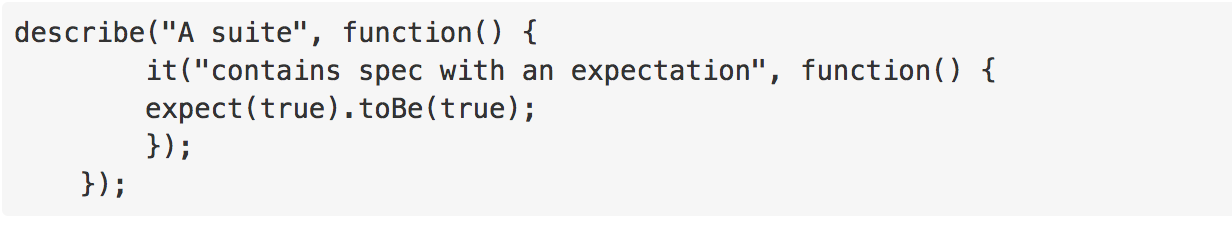


图8 一个最基本的jasmine测试用例

Jasmine框架以函数describe(string, function)封装一个测试集，一个测试集(describe)包含多个Specs(it)，一个Specs(it)包含多个断言（expect）。其中describe方法有两个参数，一个是String一个是function，字符串用来描述这个test suite，function就是测试代码可以包括一个或多个it函数；方法it用于开始specs，it方法和describe方法一样有String和function两个参数，String用来描述测试点（spec），function是具体的测试代码，同样一个it测试点 (spec)可以包含多个expections；方法expect表示测试的断言，返回为true或者false，全部的断言返回true则这个测试点就通过，一个或者多个断言返回false这个测试点就不通过。

另外类似Junit测试框架，为了代码简洁减少重复性的工作，jasmine也提供beforeEach和afterEach方法。beforeEach会在describe函数中每个spec之前执行，after会在describe函数中每个spec之后执行，适用于执行比较耗时或者耗资源的一些共同的初始化和清理工作。

### 4.4.2 编写spec代码

atom插件的测试代码目录为spec，一般业务逻辑文件夹lib下的js文件都需要在spec目录下有相应命名加上-spec的测试文件。

以主文件co-refactor.js的测试文件co-refactor-spec.js为例：

首先以describe函数开始测试框架，每次运行测试代码前需要插件处于active状态等操作写入beforeEach函数，然后启动插件开始测试，如图20：

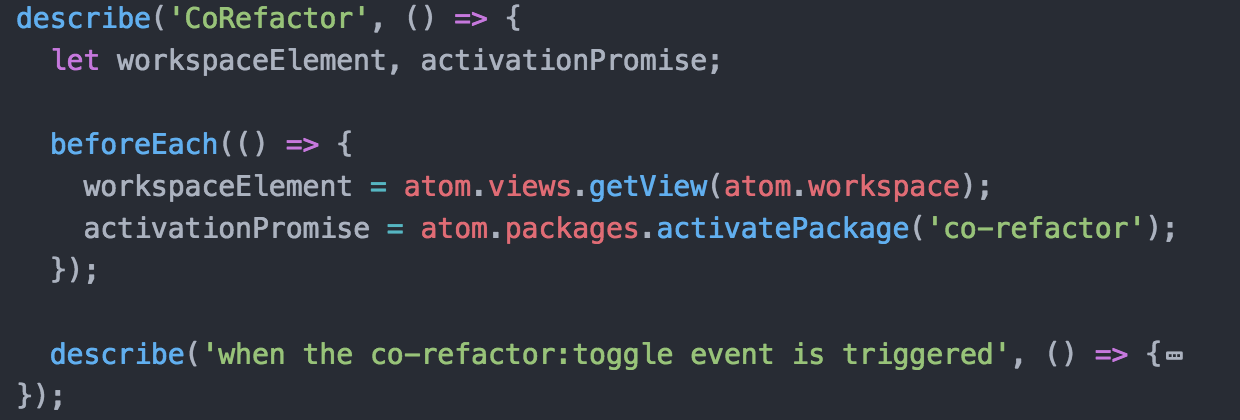


图20

主文件一般需要进行modal panel和view两个测试点的测试，以modal panel为例，主要包括coRefactorElement和coRefactorPanel两个控件的断言测试，如图21：

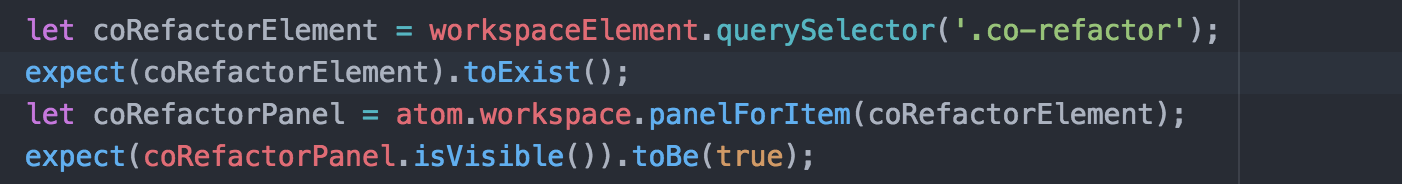


图21

测试代码编写完成后，在atom的控制面板运行window:run-package-specs，atom就会自动寻找spec目录下的测试文件全部运行并输出测试结果和运行时间，如图22：

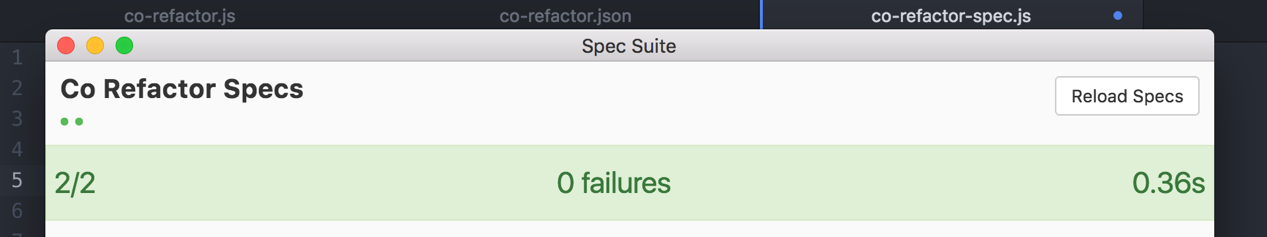


图22

window:run-package-specs命令会运行当前项目spec目录中的所有spec测试文件，若只想运行spec的一个有限的子集可以使用fdescribe和fit方法来聚焦于单个或者几个spec。

# 5 实验

## 5.1 实验环境

### 5.1.1 atom

本毕业设计为基于atom开发的插件，使用atom版本为v1.17.2 for MacOS[9]，环境依赖如图23：



图23

### 5.1.2 服务器

运行实时协同系统时需要有同步服务器，本毕设以一台阿里云Linux服务器作为实时协同系统的同步服务器，系统版本为centOS 6.8。

## 5.2 功能测试

### 5.2.1 代码调用重构功能

代码重构功能主要包括函数名重构和变量名重构，在atom中以光标选中要进行重构的对象，然后快捷键cmd+alt+r启动代码重构，在出现的重构构件中输入重构对象目标命名，快捷键cmd+enter确认提交，即可看到相应调用代码变化，完成代码调用重构。

### 5.2.2 文件引用重构功能

如图24在atom左侧树状文件结构上右击要重构的文件夹或文件，选择Refactor选项，之后在出现的输入框里输入重构对象目标路径，快捷键enter确认提交，即可看到相应引用代码变化，完成文件引用重构



图24

### 5.2.3 实时协同编程环境下测试

首先在插件包管理界面进行服务器地址配置，然后使用快捷键cmd+alt+o启动实时协同环境，输入sessionID以指定协作的文件夹路径，即可加入到该文件夹的实时编辑中。在实时协同环境下，如图25光标闪烁则表示有用户正在编辑，鼠标移动到光标上面可显示该光标对应的用户clientID。



图25

# 6 结论和展望

## 6.1 项目总结

### 6.1.1 工作量总结

如上所述，本毕业设计在实时协同环境下，根据web前端开发环境的特性，主要分析了代码调用重构和文件引用重构两类重构机制，并在此基础上以atom插件的形式完成了重构工具的开发。

开发完成后，通过atom自带的spec测试框架编写了测试代码，确保所有功能测试通过且后续开发不会出现bug回归，基本完成了既定工作目标。

### 6.1.2 技术难点总结

在以atom插件形式开发重构工具过程中，主要遇到了两个技术难点：一是代码调用重构里的js代码结构分析，查阅资料后我发现开源社区有相当完善的工具grasp可以用于构件js的完全语法树，因此最后是在grasp项目的基础上完成了js代码结构分析；另外一个技术难点是在文件引用重构的开发中，因为要去搜索整个工程以更新所有对被重构文件的引用，在这个重构过程中需要记录视图与状态的对应关系，针对这个特性我选择了使用redux+react的架构方式，redux框架提供的store存贮状态的方案能很好地解决我们的问题。

## 6.2 不足与展望

### 6.2.1 文件拖拽

在开始开发文件引用重构功能时，一开始的方案是可以直接拖拽文件到目标位置，然后自动实现重构的，但是后来发现atom没有提供在文件目录导航栏拖拽文件的API，因此只能改成右击重构对象后，再输入目标路径的方案。

### 6.2.2 更友好的重构方式

在开发代码调用重构功能时，为了更友好的使用方式，也是考虑直接光标选择重构对象后，快捷键开启重构功能，直接在编辑器输入目标命名而不用再出现一个输入框专门用于输入目标命名，但是在开发过程中前一种方案遇到了很大的技术难点，限于工作量最终选择了第二种方案，在用户使用友好性上有所不足。

# 

# 参考文献

1. web2.0定义. http://www.chinaw3c.org/about.html.
2. SublimeText3. http://www.sublimetext.com/3.
3. MS VScode. https://code.visualstudio.com/.
4. Adobe Brackets. http://brackets.io/.
5. Atom. https://atom.io/
6. 《ECMAScript 6 入门》阮一峰
7. atom飞行手册. https://www.gitbook.com/book/wizardforcel/atom-flight-manual-zh-cn/details
8. grasp框架. https://github.com/gkz/grasp/
9. fs-plus框架. https://github.com/atom/fs-plus
10. facebook codemod. https://github.com/facebook/codemod/
11. facebook jscodeshift. https://github.com/facebook/jscodeshift
12. 实时协同编程项目. https://github.com/nicktogo/coeditor
13. jasmine测试框架. https://jasmine.github.io/1.3/introduction.html
14. Redux文档. http://www.redux.org.cn/

# 谢 辞

正文内容

出自内心，有感而发。正文：五号，宋体（英文Times New Roman），两端对齐，段落首行左缩进2个汉字符，行距18磅，段前0行，段后0行。