电子科技大学
University of Electronic Science and Technology of China

学　士　学　位　论　文

BACHELOR DISSERTATION

论文题目分布式在线程序评测系统——web端实现

学生姓名	李昀
学　　号	2010013100008
专　　业	通信工程
学　　院	通信与信息工程学院
指导教师	饶力
指导单位	电子科技大学

2014年2月24日

摘 要

TODO

关键词：Web应用，J2EE，MVC框架，在线评测系统，AngularJS

ABSTRACT

TODO

Keywords: Web application, J2EE, MVC framework, Online judge, AngularJS

目 录

第1章 引言

1.1 课题背景

1.2 研究意义

1.3 研究现状

第2章 系统需求分析

2.1 功能需求

2.2 性能需求

2.3 服务器端与客户端运行要求

第3章 系统概要设计

第4章 结束语

参考文献

[1] http://developer.chrome.com/. MVC Architecture[EB/OL]. `http://developer.chrome.com/apps/app_frameworks`.

致 谢

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

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

感谢时富军同学给我们提供了电子科技大学毕业论文的LaTeX模板，为我们排版论文带来了无比的便捷。

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

MVC Architecture[1]

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

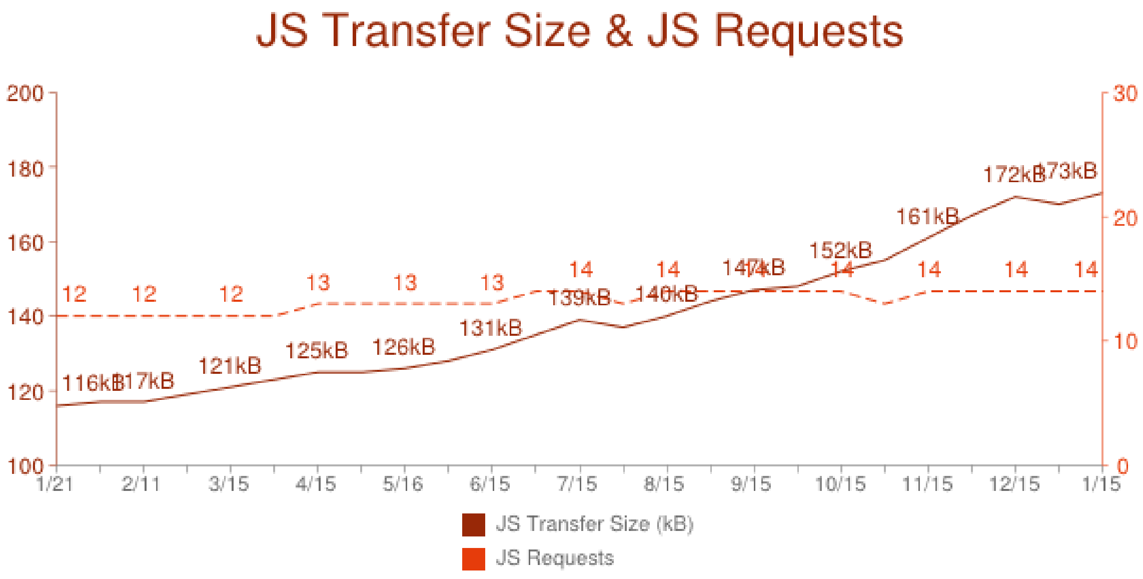

Figure A–1 JS Transfer Size & JS Requests

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

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

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

1.1 MVC pattern overview

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

MVC is composed of three components:

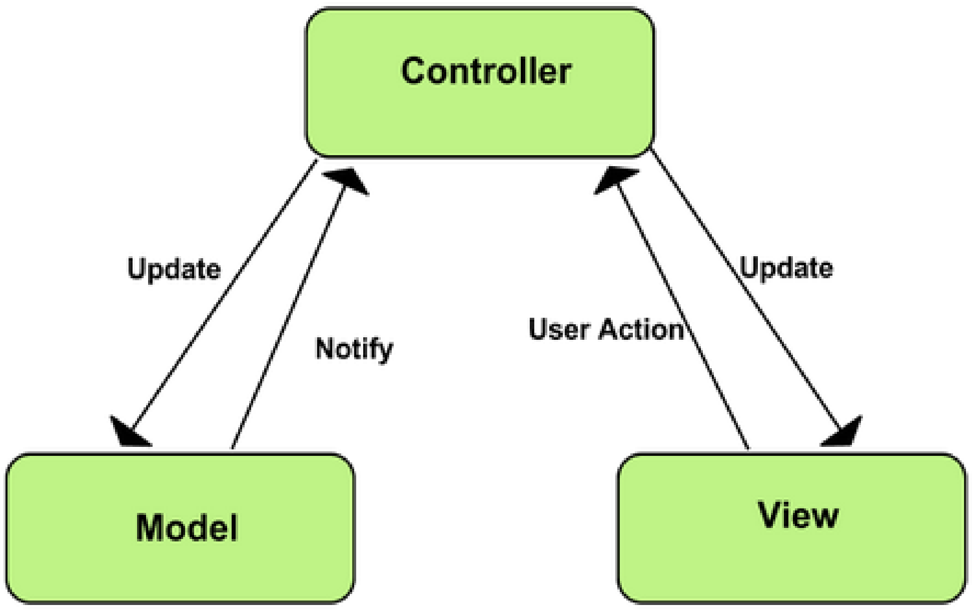

Figure A–2 MVC

1.1.1 Mode

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

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

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

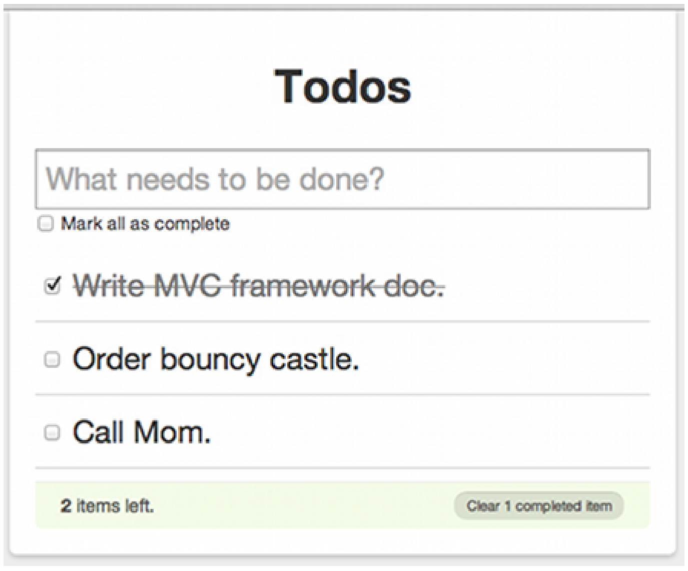

Figure A–3 Todo list app

1.1.2 View

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

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

1.1.3 Controller

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

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

There are a few variations of the MVC design pattern such as MVP (Model–View–Presenter) and MVVP(Model–View–ViewModel). Even with the so called MVC design pattern itself, there is some variation between the traditional MVC pattern vs the modern interpretation in various programming languages. For example, some MVC–based frameworks will have the view observe the changes in the models while others will let the controller handle the view update. This article is not focused on the comparison of various implementations but rather on the separation–of–concerns and it's importance in writing modern web apps.

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

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

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

1.2 MVC persistence patterns

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

1.2.1 Model does its own persistence - ActiveRecord pattern

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

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

An adapter, or a proxy, receives the requests from a store and translates them into appropriate actions to take against your persistent data layer (such as JSON API). This is interesting in the modern web app design because you often interact with more than one persistent data layer such as a remote server and browser's local storage. Chrome Apps provides both Chrome Storage API and HTML 5 fileSystem API for client side storage.

Pros:

- Simple to use and understand.

Cons:

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

1.2.2 Controller does persistence

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

Pros:

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

Cons:

- Code can be more complex to maintain.

1.2.3 AppController does persistence

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

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

Pros:

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

Cons:

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

MVC架构

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

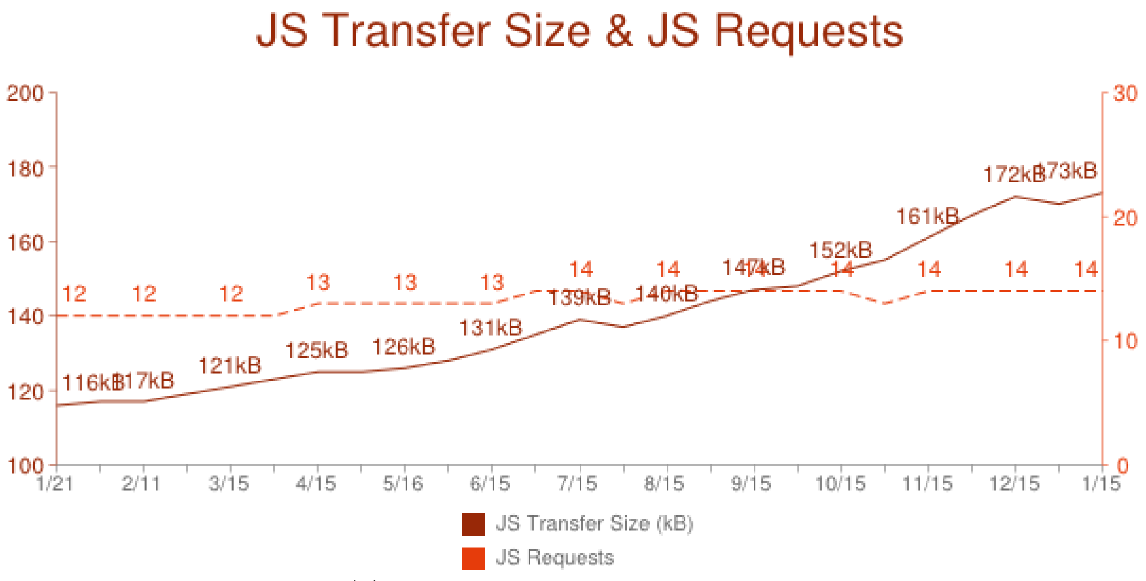

图 A–1 JS Transfer Size & JS Requests

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

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

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

1.1 MVC模式概览

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

MVC模式由三部分组成：

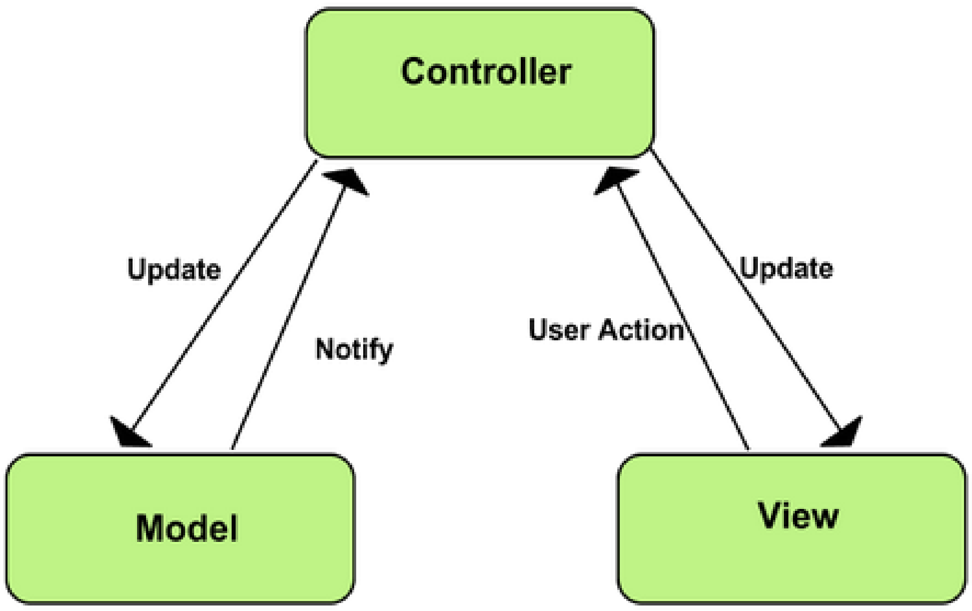

图 A–2 MVC

1.1.1 模型（Model）

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

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

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

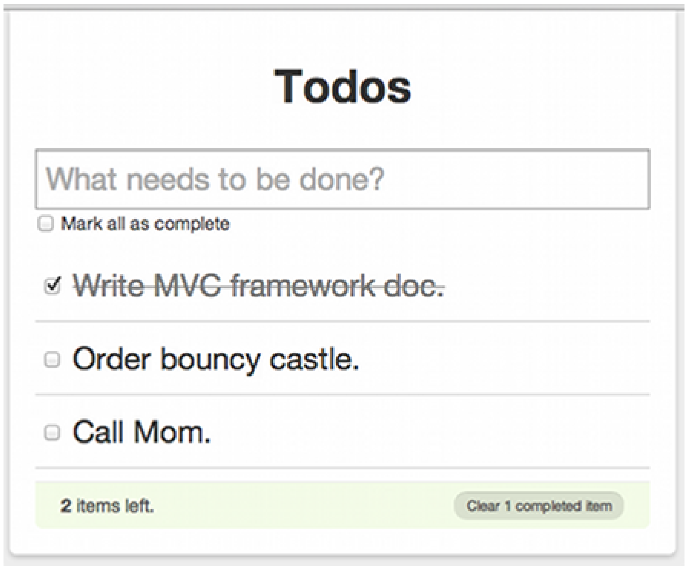

图 A–3 待办列表应用

1.1.2 视图（View）

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

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

1.1.3 控制器（Controller）

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

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

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

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

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

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

1.2 MVC持久化模式

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

1.2.1 在模型中完成持久化——活动记录模式

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

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

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

数据中，Chrome应用提供了Chrome Storage API和HTML5 fileSystem API两种不同的选择。

优点：

- 使用简单，便于理解。

缺点：

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

1.2.2 在控制器中完成持久化

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

优点：

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

缺点：

- 代码将会变得复杂且难以维护。

1.2.3 在应用控制器中完成持久化

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

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

优点：

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

缺点：

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