

DISCOVER METEOR

Building Real-Time JavaScript Web Apps

Version 1.9 (updated 2015年4月15日)

Tom Coleman & Sacha Greif

Cover photo credit: **Perseid Hunting** by Darren Blackburn, licensed under a Creative Commons Attribution 2.0 Generic license.

www.discovermeteor.com

简介

先来活动一下大脑。假设你坐在电脑面前，在两个窗口中打开同一个文件夹。

在其中一个窗口中删除一个文件，另一个窗口中的这个文件会消失吗？

不用实际操作你也知道肯定会消失的。在本地文件系统中的操作，不用刷新或者回调，变动就能应用到所有地方。

我们再来看一下相同的事情在网页中会有什么结果。例如，你在两个浏览器窗口中打开同一个 WordPress 后台页面，在其中一个窗口中新建了一篇文章。和桌面系统不同的是，不管等待多长时间，另一个窗口都不会发生变化，除非你手动刷新网页。

过去这些年，我们已经习惯了，人和网站之间的通信是离散的。

但是，作为新一代框架和技术之一的 Meteor，尝试挑战这一现状，让网页能够实时和响应。

Meteor 是什么？

Meteor 是一个构建在 Node.js 之上的平台，用来开发实时网页程序。Meteor 位于程序数据库和用户界面之间，保持二者之间的数据同步更新。

因为 Meteor 是基于 Node.js 开发的，所以在客户端和服务器端都使用 JavaScript 作为开发语言。而且，Meteor 程序的代码还能在前后两端共用。

Meteor 这个平台很强大，网页程序开发过程中的很多复杂、容易出错的功能都能抽象出来，实现起来很简单。

为什么使用 Meteor？

那么，你为什么要花时间学习 Meteor，而不去学其他框架呢？拨开 Meteor 的各种功能，我们

认为原因只有一个：因为 Meteor 易于学习。

而且，和其他框架不同，使用 Meteor，几小时之内就能开发出一个正常运行的实时网页程序。如果之前做过前端开发，对 JavaScript 已经有所了解，甚至都不用再学习一门新的编程语言。

Meteor 可能就是你要找的理想框架，当然，也可能不是。既然只要几晚或一个周末就能上手，为什么不试试呢？

为什么选择这本书？

在过去的几年中，我们一直在开发很多个 Meteor 项目，范围从网站到移动应用，从商业项目到开源项目。

我们学到了很多，但总是不容易找到问题的答案。我们不得不从不同来源讲东西拼凑在一起，并且在许多情况下，我们甚至创造了我们自己的解决方案。所以通过这本书，我们想分享所有这些经验教训，并创建了一个简单的一步一步的指导，来引导你从零开始构建一个完整的 Meteor 应用。

我们即将构建的应用是一个简化版的社交新闻网站，类似 **Hacker News** 或 **Reddit**，我们称之为 Microscope（借鉴 Meteor 开源应用 **Telescope**），在开发的过程中，我们会解决构建 Meteor 应用所会遇到的各种要素，例如用户账户、Meteor Collection、路由等等。

这本书为谁编写？

我们在写这本书时，目标之一就是要让内容通俗易懂。所以，即使你没有任何 Meteor、Node.js、MVC 框架或服务器端编程经验，都能够读完这本书。

但另一方面，我们也假设你熟悉基本的 JavaScript 语法和概念。但是如果你曾经玩过一些 jQuery 代码或接触过浏览器开发者控制台，你应该没有问题的。

如果你还不太熟悉 JavaScript，我们建议你在开始阅读本书之前，先阅读一下我们的 **JavaScript primer for Meteor**（英文）。

关于作者

如果你想知道我们是谁，为什么要相信我们，这里是两位作者的一些背景介绍。

Tom Coleman 是 **Percolate** 工作室的一员。Percolate 工作室是一个程序开发商，致力于高品质的产品和用户体验。他也是 **Atmosphere** 包仓库的维护人之一，同时也参与开发了多个 Meteor 开源项目（例如 **Iron Router**）。

Sacha Greif 是一名产品设计师和网页设计师，为创业项目工作，例如 **Hipmunk** 和 **RubyMotion**。他开发了 **Telescope** 和 **Sidebar**，还是 **Folio** 的创始人。

章节和附录

我们希望这本书对 Meteor 初学者和有经验的程序员都有所帮助，因此把内容分成了两类：常规的章节（1-14 章）和附录（带 .5 的序号）。

常规的章节会讲解如何开发程序，尽量保证你能跟着我们的步伐实际操作，只关注开发过程中最重要的步骤，不会太深入细节。

而附录则会深入 Meteor 错综复杂的细节，帮助你更好的理解背后到底发生了什么。

如果你是初学者，第一次阅读完全可以跳过附录，熟悉 Meteor 之后再回过头来阅读。

代码提交和线上演示

阅读编程相关的书籍时最怕遇到这种事情，虽然一直跟着书中的步骤，但突然发现代码和示例不一样了，而且程序也不能正常运行。

为了避免这种情况发生，我们特意在 **GitHub** 上为 **Microscope** 建了仓库。改动一些代码后，会给出指向 `git` 提交的链接，而且还会链接到该提交对应的线上演示，方便和你自己本地的版本对比。下面一个示例：



我们提供的代码提交链接，并不是为了让你使用 `git checkout` 从一个提交跳到另一个提交。自己动手输入程序的代码，学习效果才能更好。

一些其他资源

如果想更深入地学习 Meteor 的各项功能，最好的资料就是官方文档。

遇到问题我们建议到 **Stack Overflow** 网站上寻找帮助。如果想获得实时帮助，可以加入 **IRC** 的 `#meteor` 频道。

需要了解 Git 吗？

阅读本书虽然不完全要求你熟悉 Git 版本控制，但我们强烈推荐你去学。

如果想快速上手，我们推荐阅读 Nick Farina 撰写的文章《**Git Is Simpler Than You Think**》。

对 Git 初学者，我们推荐使用 **GitHub** (Mac OS)，或 **SourceTree** (Mac OS & Windows)，这两个软件都是免费的。

联系方式

- 如果想和我们联系，可以发送邮件到 hello@discovermeteor.com。
- 如果发现本书中有错别字或不当之处，请到 [GitHub](#) 上的仓库中提交。
- 如果发现 Microscope 的代码有问题，请到 [Microscope](#) 的仓库中提交。
- 如果还有其他问题，可以直接在网页的侧栏中留言。

开始

第一印象十分重要，安装 Meteor 并不会遇到什么困难。大多数情况下，在五分钟内便可以完成。

首先，如果在 Mac OS 或 Linux 系统下，你可以打开终端窗口，输入以下命令来安装 Meteor：

```
$ curl https://install.meteor.com | sh
```

如果你使用 Windows 系统，请参考 Meteor 网站的 安装指导。

以上命令会在系统中安装 `meteor` 可执行文件，然后就可以使用 Meteor 了。

选择不安装 Meteor

如果你无法或者不想在本地安装 Meteor，我们推荐你使用 **Nitrous.io**。

使用 Nitrous.io 可以让你在浏览器中直接编辑代码并运行程序。我们撰写了一篇简短的指南，介绍如何使用 Nitrous.io。

你可以一直阅读那篇指南直到“Installing Meteor”部分，然后再回到本章，从“创建简单应用”一节开始阅读。

创建简单的应用

安装好 Meteor 之后，我们来创建一个应用。创建应用要使用 Meteor 的命令行工具 `meteor`：

```
$ meteor create microscope
```

上述命令会下载 Meteor，然后新建一个基本可用的 Meteor 项目。命令执行完成后，会看到新建了一个文件夹，名为 `microscope/`，包含以下文件：

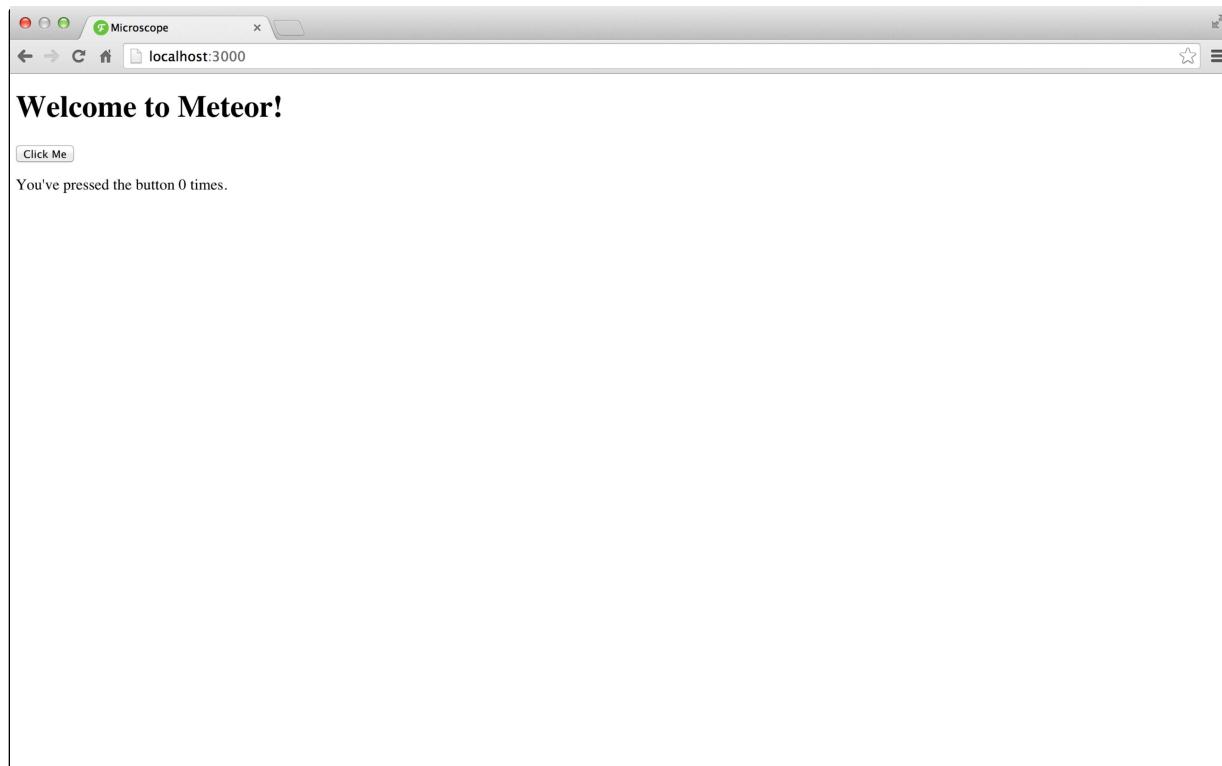
```
.meteor  
microscope.css  
microscope.html  
microscope.js
```

Meteor 生成的应用只是一个简单的骨架，演示一些简单的模式。

虽然这个应用没什么功能，但也能运行。要运行应用，请切换到终端，输入下面的命令：

```
$ cd microscope  
$ meteor
```

现在打开浏览器，访问 `http://localhost:3000`（或者等效的 `http://0.0.0.0:3000`），应该能看到下面的网页：



Meteor 的 Hello World 网页

提交 2-1

创建 Microscope 项目的基础文件

在 [GitHub](#) 中查看

[查看演示](#)

恭喜！你的第一个 Meteor 应用顺利运行了。顺便说一下，如果想停止运行程序，只要切换到对应的终端窗口按 `ctrl+c` 键即可。

如果你使用 Git，正是时候用 `git init` 来初始化你的项目仓库。

再见 Meteorite

曾经有段时间，Meteor 依赖于外部代码包管理器 Meteorite。自从 Meteor 0.9.0 版本以后，就不再需要 Meteorite 了，因为它的功能已经融入 Meteor 之中。

所以，如果你在这本书或在浏览 Meteor 相关的资料时，遇到 Meteorite 的 `mrt` 命令行工具，你可以放心地用 `meteor` 来替换它。

添加代码包

下面我们使用 Meteor 的 package 系统在项目中引入 **Bootstrap** 框架。

这与通常手动添加 Bootstrap 的 CSS 和 Javascript 文件的方法是没有区别的，只不过我们依赖代码包维护者来为我们更新这些文件。

既然我们说到此，我们也来添加 **Underscore** 代码包。Underscore 是一个 JavaScript 工具库，对于操纵 JavaScript 数据结构非常有用。

bootstrap 代码包由 twbs 用户维护，所以该代码包的全名为 twbs:bootstrap。

另一方面，underscore 代码包依然算作 Meteor “官方”的代码包，所以这个包没有作者：

```
meteor add twbs:bootstrap  
meteor add underscore
```

注意的是现在我们添加了 Bootstrap 3。而这本书中的一些截图是老版本的 Microscope 使用 Bootstrap 2 时截取的，所有它们看起来会有稍微不同。

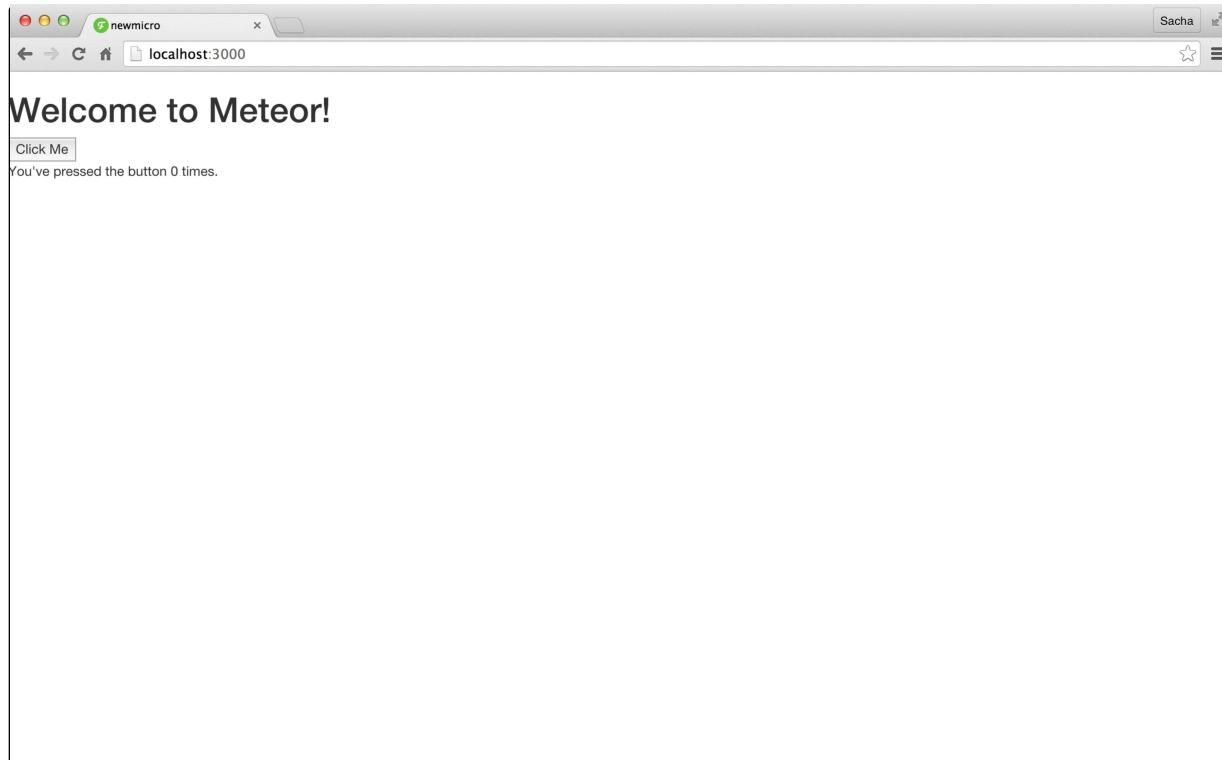
提交 2-2

添加 bootstrap 和 underscore 代码包

在 [GitHub](#) 中查看

[查看演示](#)

一旦你添加了 Bootstrap 代码包，你应会注意到我们应用的变化：



添加了 *Bootstrap*.

与“传统”方式添加外部资源不同，我们还没有链接任何 CSS 或 JavaScript 文件，因为 Meteor 已经帮我们搞定了！这就是 Meteor 代码包的众多优势之一。

关于代码包

Meteor 中的代码包有点特殊，分为五种：

- Meteor 核心代码本身分成多个核心代码包（core package），每个 Meteor 应用中都包含，你基本上不需要花费精力来维护它们
- 常规 Meteor 代码包称为“**isopack**”，或同构代码包（isomorphic package，意味着它们既能在客户端也能在服务器端工作）。第一类代码包例如 `accounts-ui` 或 `appcache` 由 Meteor 核心团队维护，与 Meteor 捆绑在一起。
- 第三方代码包就是其他用户开发的 isopack 上传到 Meteor 的代码包服务器上。你可以访问 **Atmosphere** 或 `meteor search` 命令来浏览这些代码包。
- 本地代码包（local package）是自己开发的代码包，保存在 `/packages` 文件夹中。
- **NPM** 代码包（NPM package）是 Node.js 的代码包，虽不能直接用于 Meteor，但可以在上述几种代码包中使用

Meteor 应用的文件结构

开始编写代码之前，我们必须要正确的设置项目。为了保证项目整洁，请打开 `microscope` 文件夹，删除 `microscope.html`、`microscope.js` 和 `microscope.css`。

请在 `microscope` 文件夹中新建四个子文件夹：`/client`，`/server`，`/public` 和 `/lib`。然后在 `/client` 文件夹中新建两个空文件：`main.html` 和 `main.js`。如果程序无法运行了先别担心，从下一章开始我们会编写代码。

值得一提的是，上述文件夹中有一些拥有特别的作用。关于文件，Meteor 有以下几条规则：

- 在 `/server` 文件夹中的代码只会在服务器端运行。
- 在 `/client` 文件夹中的代码只会在客户端运行。
- 其它代码则将同时运行于服务器端和客户端上。
- 请将所有的静态文件（字体，图片等）放置在 `/public` 文件夹中。

知道 Meteor 以什么顺序加载文件也很有用：

- 在 `/lib` 文件夹中的文件将被优先载入。
- 所有以 `main.*` 命名的文件将在其他文件载入后载入。
- 其他文件以文件名的字母顺序载入。

需要注意的是，即便 Meteor 包含上述规则，这并不意味着它强制你为你的 Meteor 应用采用任何预设的文件结构。上述结构只是我们的建议，并不是一成不变的。

对此如果你想了解更多，我们强烈建议你参阅 **Meteor** 官方文档。

Meteor 采用 MVC 架构吗？

如果你之前有过在其它诸如 Ruby on Rails 框架下开发的经历，此时你心中可能会有这样的疑问， Meteor 采用 MVC（Model View Controller）架构吗？

简短的回答是，不。与 Rails 不同， Meteor 并不为你的应用强加任何预设的架构。因此本书将直接给出我们认为最合理的代码，而不对任何现有架构作过多考虑。

不需要 `public` 文件夹？

好吧，我们承认在之前小小的忽悠了大家一下。其实我们并不需要为我们的应用建立一个 `public/` 文件夹，因为 Microscope 并不需要使用任何的静态文件。但是值得注意的是，大多数 Meteor 应用都会或多或少使用一些图片，因此我们认为 `public/` 文件夹还是值得一谈的。

另外，你可能注意到了一个隐藏的 `.meteor` 文件夹。这是 Meteor 存储它内部代码的地方，尝试更改里面的内容并不是什么好主意。事实上，你根本不需要关心其中的内容。有两个例外是 `.meteor/packages` 文件和 `.meteor/release` 文件。它们分别列出了你安装的所有智能代码包和你使用的 Meteor 版本。当你为你的应用添加代码包或更改 Meteor 版本时，查看这两个文件的变更可能会为你带来一些帮助。

下划线命名法 vs 驼峰命名法

对于历史悠久的下划线命名法（`my_variable`）和驼峰命名法（`myVariable`）我们认为选择哪种并不重要，只要你坚持在项目中贯彻它。

在本书中，我们将采用驼峰命名法，因为它是 JavaScript 中的惯例（毕竟它叫 `JavaScript` 而不是 `java_script` 呀！）。

对此唯一的例外是，对文件的命名，我们将采用下划线命名法（`my_file.js`）。对于 CSS 类，我们将使用连字号（`.my-class`）。这样做的原因是在文件系统中，下划线命名法最常见，而 CSS 语法本身就使用连字号作为连接（比如 `font-family`，`text-align` 等）。

搞定 CSS

本书并不侧重于 CSS。所以为了避免在 CSS 细节上花费过多时间，我们决定在本书一开始就给大家提供完整的 CSS 文件。因此你不必再担心这个问题。

CSS 文件将被 Meteor 自动加载并简化。因此，不同于其它的静态文件都被放置于 `/public` 文件夹，请将 CSS 文件放入 `/client` 文件夹。请创建一个 `client/stylesheets/` 文件夹并将以下 `style.css` 文件放置入内。

```
.grid-block, .main, .post, .comments li, .comment-form {
  background: #fff;
  border-radius: 3px;
  padding: 10px;
  margin-bottom: 10px;
  -webkit-box-shadow: 0 1px 1px rgba(0, 0, 0, 0.15);
  -moz-box-shadow: 0 1px 1px rgba(0, 0, 0, 0.15);
  box-shadow: 0 1px 1px rgba(0, 0, 0, 0.15); }

body {
  background: #eee;
  color: #666666; }

#main {
  position: relative;
}

.page {
  position: absolute;
  top: 0px;
  width: 100%; }

.navbar {
  margin-bottom: 10px; }
/* line 32, ../sass/style.scss */
.navbar .navbar-inner {
  border-radius: 0px 0px 3px 3px; }

#spinner {
  height: 300px; }

.post {
  /* For modern browsers */
  /* For IE 6/7 (trigger hasLayout) */
  *zoom: 1;
  position: relative;
  opacity: 1; }
.post:before, .post:after {
  content: "";
  display: table; }
.post:after {
  clear: both; }
.post.invisible {
  opacity: 0; }
.post.instant {
```

```
-webkit-transition: none;  
-moz-transition: none;  
-o-transition: none;  
transition: none; }  
.post.animate{  
-webkit-transition: all 300ms 0ms;  
-moz-transition: all 300ms 0ms ease-in;  
-o-transition: all 300ms 0ms ease-in;  
transition: all 300ms 0ms ease-in; }  
.post .upvote {  
display: block;  
margin: 7px 12px 0 0;  
float: left; }  
.post .post-content {  
float: left; }  
.post .post-content h3 {  
margin: 0;  
line-height: 1.4;  
font-size: 18px; }  
.post .post-content h3 a {  
display: inline-block;  
margin-right: 5px; }  
.post .post-content h3 span {  
font-weight: normal;  
font-size: 14px;  
display: inline-block;  
color: #aaaaaa; }  
.post .post-content p {  
margin: 0; }  
.post .discuss {  
display: block;  
float: right;  
margin-top: 7px; }  
  
.comments {  
list-style-type: none;  
margin: 0; }  
.comments li h4 {  
font-size: 16px;  
margin: 0; }  
.comments li h4 .date {  
font-size: 12px;  
font-weight: normal; }  
.comments li h4 a {  
font-size: 12px; }
```

```
.comments li p:last-child {  
  margin-bottom: 0; }  
  
.dropdown-menu span {  
  display: block;  
  padding: 3px 20px;  
  clear: both;  
  line-height: 20px;  
  color: #bbb;  
  white-space: nowrap; }  
  
.load-more {  
  display: block;  
  border-radius: 3px;  
  background: rgba(0, 0, 0, 0.05);  
  text-align: center;  
  height: 60px;  
  line-height: 60px;  
  margin-bottom: 10px; }  
.load-more:hover {  
  text-decoration: none;  
  background: rgba(0, 0, 0, 0.1); }  
  
.posts .spinner-container{  
  position: relative;  
  height: 100px;  
}  
  
.jumbotron{  
  text-align: center;  
}  
.jumbotron h2{  
  font-size: 60px;  
  font-weight: 100;  
}  
  
@-webkit-keyframes fadeIn {  
  0% { opacity: 0; }  
  10% { opacity: 1; }  
  90% { opacity: 1; }  
  100% { opacity: 0; }  
}  
  
@keyframes fadeIn {  
  0% { opacity: 0; }  
  10% { opacity: 1; }  
}
```

```
90% {opacity: 1;}  
100% {opacity: 0;}  
}  
  
.errors{  
  position: fixed;  
  z-index: 10000;  
  padding: 10px;  
  top: 0px;  
  left: 0px;  
  right: 0px;  
  bottom: 0px;  
  pointer-events: none;  
}  
  
.alert {  
  animation: fadeOut 2700ms ease-in 0s 1 forwards;  
  -webkit-animation: fadeOut 2700ms ease-in 0s 1 forwards;  
  -moz-animation: fadeOut 2700ms ease-in 0s 1 forwards;  
  width: 250px;  
  float: right;  
  clear: both;  
  margin-bottom: 5px;  
  pointer-events: auto;  
}
```

client/stylesheets/style.css

提交 2-3 重新整理文件结构

[在 GitHub 中查看](#) [查看演示](#)

CoffeeScript 说明

在本书中我们将使用纯 JavaScript。但是如果你更倾向于使用 CoffeeScript，Meteor 可以帮助你做到这点。你只需添加 CoffeeScript 代码包，之后便可以在项目中使用 CoffeeScript 了！

```
meteor add coffeescript
```

有些人喜欢不被打扰地工作，直到项目足够完美才去发布，而有些人则迫不及待的要向大家展示自己的项目。

如果你是第一种人，现在宁愿在本地开发，那么可以果断跳过这一章。相反，如果你更愿意花时间去学习如何把 Meteor 应用部署到线上，我们下面为你提供一些方法。

我们将学习几种不同的方法去部署一个 Meteor 应用。无论你是在开发 Microscope 或任何其他的 Meteor 应用，在你开发过程的任何阶段，可以随意地从它们当中挑选一个。让我们马上开始吧！

引入附录

这是一个附录章节。不同于其他书的是，本书的附录会让我们深入去了解更多关于 Meteor 的知识。

现在如果你更愿意去继续构建 Microscope，你现在可以先忽略这一章，等有空再回来看也没问题。

部署在 Meteor

首先最简单的是部署到 Meteor 的子域名上（例如：<http://myapp.meteor.com>），这是我们首先要去学习的。在项目早期，这对于展示你的应用和快速设置一个测试服务器都很有用途。

而部署在 Meteor 是非常简单的。打开终端，定位到你 Meteor 应用的目录，并输入：

```
meteor deploy myapp.meteor.com
```

当然，你要把“myapp”替换成你想要的名称，最好是命名一个没有被使用的。如果你的名称已经被使用，Meteor 会提示你去输入密码。如果发生这样的情况，只需通过 `ctrl+c` 来取消当前操作，然后用另一个不同的名称再试一次。

如果顺利地部署成功了，几秒钟后你就能够在 `http://myapp.meteor.com` 上访问到你的应用了。

你可以参考官方文档去了解更多关于如何直接访问你域名下的数据库，或者为你的应用设置一个自定义域名等等的相关信息。

部署在 Modulus

Modulus 是一个部署 Node.js 应用很好的选择。这是为数不多的 PaaS（platform-as-a-service 平台即服务）的提供商，并且已经正式支持 Meteor，已经有不少人在它上面去搭建 Meteor 应用了。

你可以通过阅读他们的部署 **Meteor** 应用指南去了解更多关于 Modulus 的信息。

Meteor Up

虽然每天都有新的云端解决方案出来，但是它们通常都有自己的一些问题和限制。目前，把 Meteor 应用部署在自己的服务器才是一个最好的方式。然而麻烦的是，部署到自己的服务器并不是那么简单，尤其如果你注重产品部署上去的质量的话。

Meteor Up（简称 `mup`）是另一个通过命令行的操作去帮助你解决安装和部署问题。所以让我们看看如何通过 Meteor Up 来部署 Microscope。

在此之前，我们需要一个服务器来发布。我们建议使用 **Digital Ocean**（每月最低5美元），或者 **AWS**（它为小型实例提供免费，如果你只是想试玩玩 Meteor Up 就已经足够了）。

无论选择哪种服务，你应该要解决这三样东西：你服务器的 IP 地址，登录账号（通常是

root 或者 ubuntu) 和登录密码。将它们安全地保存起来，我们很快就会用到。

Meteor Up 的初始化

首先，我们需要通过 `npm` 去安装 Meteor Up:

```
npm install -g mup
```

然后我们将创建一个单独的目录，为我们的 Meteor Up 提供一个特定的部署环境。我们使用单独的目录出于两个原因：第一，这可以很好的避免里面包含任何你 Git 存储库的隐藏文件，尤其如果你是在公共代码库去操作。

第二，通过使用多个单独的目录，我们能够并行地进行多个 Meteor Up 管理和配置。这将会用在实际产品的部署以及分段实例的部署。

所以我们来创建这个新目录，并使用它来初始化一个新的 Meteor Up 项目：

```
mkdir ~/microscope-deploy  
cd ~/microscope-deploy  
mup init
```

通过 Dropbox 分享

为了确保你和你的团队都使用相同的部署设置，一个很好的方法就是把你的 Meteor Up 配置文件夹放在你的 Dropbox 上，或者任何类似的服务上。

Meteor Up 的配置

当初始化一个新项目的时候， Meteor Up 会为了创建两个文件： `mup.json` 和 `settings.json`。

`mup.json` 会保存所有我们部署的相关设置，而 `settings.json` 会保存所有应用的相关设置（OAuth token、Analytics token，等等）。

下一步就是去配置你的 `mup.json` 文件。`mup.json` 会默认在执行 `mup init` 的时候生成，而你要做的就是把空白的填上：

```
{  
  //server authentication info  
  "servers": [{  
    "host": "hostname",  
    "username": "root",  
    "password": "password"  
    //or pem file (ssh based authentication)  
    //"pem": "~/.ssh/id_rsa"  
  }],  
  
  //install MongoDB in the server  
  "setupMongo": true,  
  
  //location of app (local directory)  
  "app": "/path/to/the/app",  
  
  //configure environmental  
  "env": {  
    "ROOT_URL": "http://supersite.com"  
  }  
}
```

`mup.json`

让我们了解一下这些设置。

服务器身份验证

你会注意到 Meteor Up 提供了基于密码和基于私钥（PEM）的身份验证，所以它几乎可以用于

任何的云提供商。

重要提示：如果你选择使用基于密码的身份验证，确保你在这之前已经安装了 `sshpass`（使用指南）。

MongoDB 配置

下一步是为你的应用配置 MongoDB 数据库。我们建议使用 **Compose** 或者其他提供云端 MongoDB 的提供商，因为它们提供专业支持和更好的管理工具。

如果你决定使用 **Compose**，把 `setupMongo` 设置为 `false`，并添加 `MONGO_URL` 环境变量到 `mup.json` 中的 `env` 模块。如果你决定通过 Meteor Up 去访问 MongoDB，只需要设置 `setupMongo` 为 `true`，然后 Meteor Up 会完成剩下的工作。

Meteor 应用路径

因为 Meteor Up 的配置作用在不同的目录，我们需要通过 `app` 属性去把 Meteor Up 指回到应用。只需要设置你完整的本地路径，当你位于你的应用目录里面的时候，你可以使用 `pwd` 命令去获取它。

环境变量

你可以在 `env` 模块中指定应用的所有环境变量（比如：`ROOT_URL`，`MAIL_URL`，`MONGO_URL` 等等）

设置和部署

在我们可以部署之前，我们还需要设置服务器去为 Meteor 应用托管。Meteor Up 把这个复杂的过程封装在一个简单的命令上！

```
mup setup
```

可能需要几分钟，这取决于服务器的性能和网络连接速度。设置成功后，终于可以去部署我们的应用：

```
mup deploy
```

这将会打包我们的 Meteor 应用并部署到我们刚刚设置好的服务器上。

显示日志信息

日志也是非常重要的， Meteor Up 提供非常简单的方法去处理它，通过模仿 `tail -f` 命令，输入：

```
mup logs -f
```

这一小节概述了 Meteor Up 的用法。了解更多关于它的信息，我们建议看看 **Meteor Up** 在 [GitHub](#) 上详细介绍

这三种部署 Meteor 应用的方式应该足够满足大多数的案例了。当然，我们知道一些人会喜欢更进一步地控制和设置他们的 Meteor 服务器。然而这将会是另一个主题，或者另一本书！

模板

为了更容易地进入 Meteor 的开发，我们将采用从外向内的方法来搭建项目。换句话说，我们将首先建立一个 HTML/JavaScript 的外壳，然后把它放到我们的项目里，内部细节处理稍后再说。

这意味着在本章中，我们只关注 `/client` 目录里面的事情。

让我们先在 `/client` 目录创建一个 `main.html` 文件，并写入以下代码：

```
<head>
  <title>Microscope</title>
</head>
<body>
  <div class="container">
    <header class="navbar navbar-default" role="navigation">
      <div class="navbar-header">
        <a class="navbar-brand" href="/">Microscope</a>
      </div>
    </header>
    <div id="main">
      {{> postsList}}
    </div>
  </div>
</body>
```

`client/main.html`

这是我们主要的 App 模板。在上面看到很多熟悉的 HTML 标签，除了这个 `{{> postsList}}` 标签，它是 `postsList` 模板的插入点，等一下我们就会说到。现在，先让我们创建更多的模板吧。

Meteor 模板

我们项目的核心是社会新闻网站，它是由一系列的帖子所组成的，而这正是我们要调用模板

的原因。

我们先在 `/client` 里面创建一个 `/templates` 目录。这里用来放我们所有的模板，这样可以保持项目结构的清晰整洁，接着在 `/templates` 里面再创建 `/posts` 目录来存放与帖子相关的模板。

查找文件

Meteor 的强大之处在于文件的查找。无论你把代码文件放在 `/client` 目录下的任何地方，Meteor 都可以找到它并且正确地进行编译。这意味着你永远都不需要手动编写 JavaScript 或 CSS 文件的调用路径。

这也意味着你可能会把所有的文件放在同一目录，甚至所有的代码放在同一个文件。但由于 Meteor 会把一切的代码都编译到一个压缩的文件里面，因此我们更偏向于把项目弄得井井有条，使用更整洁的文件结构，提高项目的可读性。

接下来我们开始创建第二个模板。在 `client/templates/posts` 目录中，创建 `posts_list.html`：

```
<template name="postsList">
  <div class="posts">
    {{#each posts}}
      {{> postItem}}
    {{/each}}
  </div>
</template>
```

`client/templates/posts/posts_list.html`

和 `post_item.html`：

```
<template name="postItem">
  <div class="post">
    <div class="post-content">
      <h3><a href="{{url}}>{{title}}</a><span>{{domain}}</span></h3>
    </div>
  </div>
</template>
```

client/templates/posts/post_item.html

注意模板的 `name="postsList"` 属性，它的作用是告诉 Meteor 去根据这个名称来跟踪这个模板的位置。（注意的是实际文件的文件名不相关。）

是时候来介绍 Meteor 的模板系统 **Spacebars** 了。Spacebar 就是简单的 HTML 加上三件事：Inclusion（有时也称作“partial”）、Expression 和 Block Helper。

Inclusion：通过 `{{> templateName}}` 标记，简单直接地告诉 Meteor 这部分需要用相同名称的模板来取代（在我们的例子中就是 `postItem`）。

Expression：比如 `{{title}}` 标记，它要么是调用当前对象的属性，要么就是对应到当前模板管理器中定义的 `helper` 方法，并返回其方法值（后面会详细讨论）。

Block Helper：在模板中控制流程的特殊标签，如 `{{#each}}...{{/each}}` 或 `{{#if}}...{{/if}}`。

进一步学习

你如果想了解更多关于 Spacebars，可以参考 **Spacebars** 文档。

有了这些知识，我们就可以很容易去理解了。

首先，在 `postsList` 模板里，我们通过 `{#each}...{/each}` Block Helper 去遍历一个 `posts` 对象。然后，每次迭代我们去包含 `postItem` 模板。

这个 `posts` 对象来自哪里？好问题。它实际上是一个模板 **helper**，你可以想象它是动态值的占位符（placeholder）。

`postItem` 这个模板本身相当简单。它只使用三个标签：`{{url}}` 和 `{{title}}` 都返回其集合的属性，而 `{{domain}}` 则调用模板对应的 helper 方法。

模板 Helper

到目前为止我们已经学会了使用 Spacebars，这只是在 HTML 的基础上多几个标签而已。不像其他语言如 PHP（甚至常规 HTML 页面，还包含了 JavaScript），Meteor 只是让模板和逻辑进行分离，而这些模板本身并不需要做很多复杂的事情。

为了让连接变得更流畅，一个模板需要 **helper**。你可以想象这些 helper 就是厨师用食材（你的数据）烹饪好佳肴（模板），再由服务员端到你面前。

换句话说，模板的作用局限于显示或循环变量，而 helper 则扮演着一个相当重要的角色：把值分配给每个变量。

Controller 控制器？

我们也许会情不自禁地认为包含所有模板 helper 的文件是个 controller（控制器）。但这是很模糊的，因为 controller（至少在 MVC 情况下）通常有些不同的作用。

所以我们决定远离那个术语，在谈论关于模板涉及的 JavaScript 代码时，就简单地称为“模板的 helper”或是“模板的逻辑”。

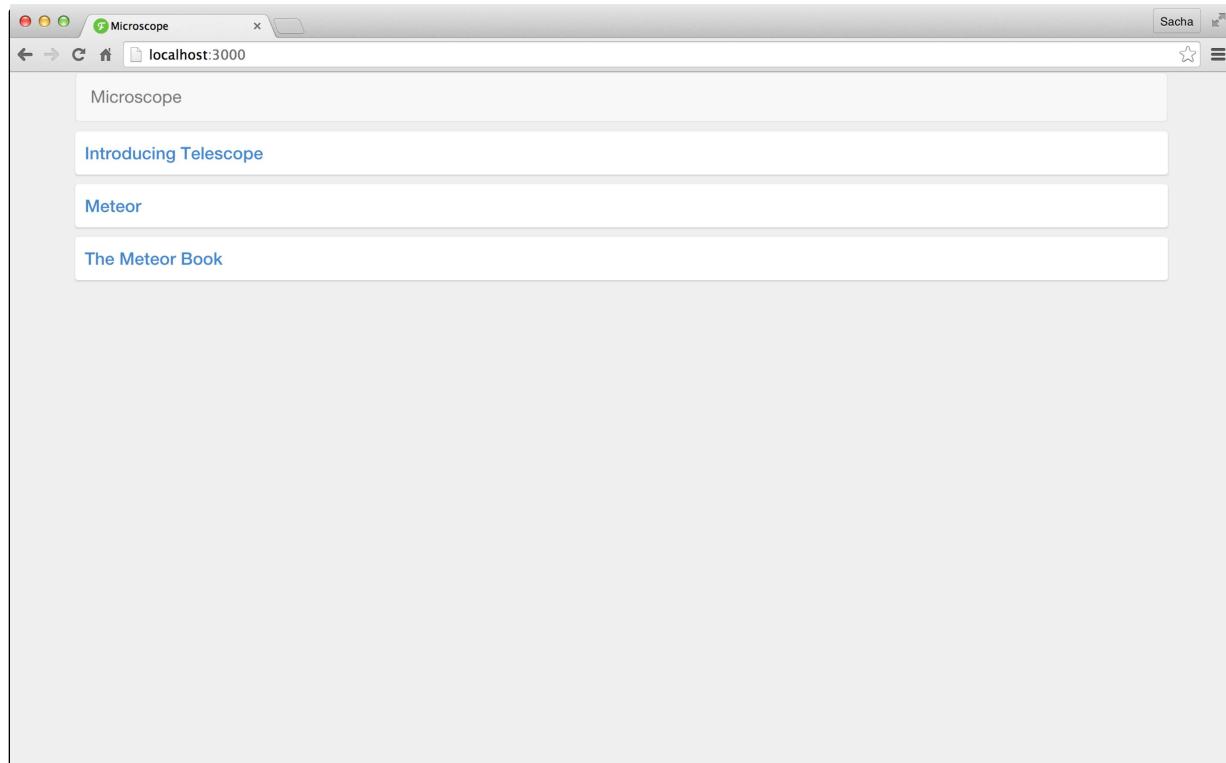
为简单起见，我们将采用与模板同名的方式来命名包含其 helper 的文件，区别是 `.js` 扩展名。那好让我们马上在 `/client/templates/posts` 目录下创建 `posts_list.js` 文件，开始构建

我们第一个 helper:

```
var postsData = [
  {
    title: 'Introducing Telescope',
    url: 'http://sachagreif.com/introducing-telescope/'
  },
  {
    title: 'Meteor',
    url: 'http://meteor.com'
  },
  {
    title: 'The Meteor Book',
    url: 'http://themeteorbook.com'
  }
];
Template.postsList.helpers({
  posts: postsData
});
```

client/templates/posts/posts_list.js

如果你运行是正确的，你现在应该在浏览器中看到这样的画面：



我们的第一个带静态数据的模板

我们刚刚在做两件事情。首先我们放置一些虚拟的基本数据到 `postsData` 数组中。原本数据应该是来自数据库的，但由于我们还没有学习到该怎么做（下一章揭晓），所以我们先通过使用静态数据来“作弊”。

然后，我们使用的是 Meteor 的 `Template.postsList.helpers()` 函数，建立了 `posts` 模板 `helper` 来返回刚刚定义的 `postsData` 数组。

如果你记得，现在我们在 `postsList` 模板中使用这个 `posts helper`:

```
<template name="postsList">
  <div class="posts">
    {{#each posts}}
      {{> postItem}}
    {{/each}}
  </div>
</template>
```

`client/templates/posts/posts_list.html`

定义 `posts helper` 就让我们的模板可以使用它，所以模板就可以遍历 `postsData` 数组并将里面的每个对象发送到 `postItem` 模板中。

提交 3-1

添加了基本的 `post` 列表模板和静态数据。

[在 GitHub 中查看](#)

[查看演示](#)

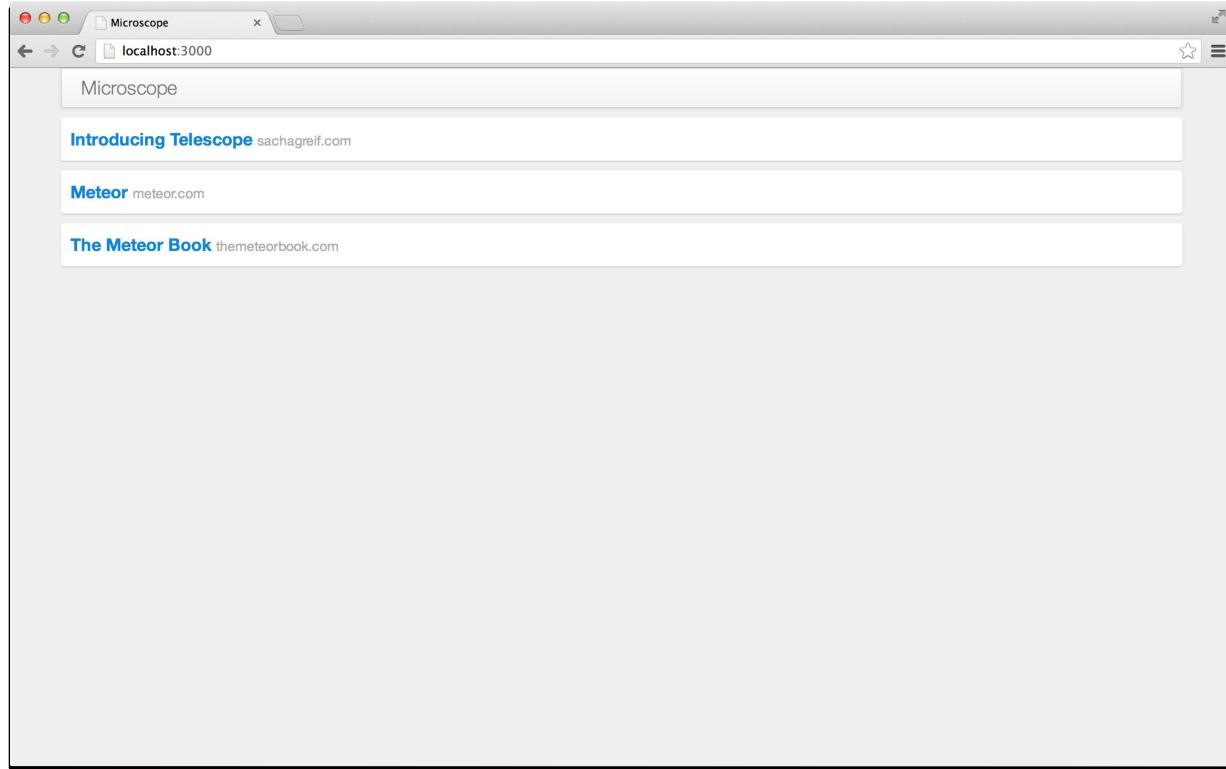
关于 `domain Helper`

类似地，我们现在创建一个 `post_item.js` 文件来包含 `postItem` 模板的逻辑：

```
Template.postItem.helpers({  
  domain: function() {  
    var a = document.createElement('a');  
    a.href = this.url;  
    return a.hostname;  
  }  
});
```

client/templates/posts/post_item.js

这一次我们 `domain` helper 的值不再是一个数组，而是一个匿名函数。相比起我们之前简化的虚拟数据的例子，这种模式更为常见（而且更有用）。



显示每个链接的域名。

这个 `domain` helper 方法通过 JavaScript 来获取一个 URL 地址并返回其域名。但是它一开始是从哪里获得 URL 地址呢？

为了回答这个问题，我们需要回到我们的 `posts_list.html` 模板。`{{{#each}}}` 代码块不仅遍历我们数组，它还在代码块范围内将 `this` 的值赋予被遍历的对象。

这意味着在 `{{{#each}}}` 标记之间，每个 `post` 都可以通过 `this` 依次访问，并且一直延伸到模板 helper (`post_item.js`) 中。

我们现在明白了为什么 `this.url` 会返回当前 `post` 的 URL。而且，如果我们在 `post_item.html` 模板里面使用 `{{{title}}}` 和 `{{{url}}}`，Meteor 就会知道需要去调用 `this.title` 和 `this.url` 去返回我们想要的正确值。

提交 3-2

设置 `postItem` 的 `domain` helper。

在 [GitHub](#) 中查看

[查看演示](#)

神奇的 JavaScript

尽管这对于 Meteor 来说并不特别，这里会简单解释一下上面“神奇的 JavaScript 代码”。首先，我们创建一个空的锚 (`a`) HTML 标签并储存在内存中。

然后我们将其 `href` 属性设置为当前 `post` 的 URL（正如我们刚刚讲到的，`this` 在 helper 中正是当前被操作的对象）。

最后，我们利用 `a` 标签的特别的 `hostname` 属性来返回 URL 的域名。

如果你正确地紧跟着我们的进度，这时候你就会在浏览器中看到一个 `post` 列表。但这个列表只是调用了静态数据，它没有利用到 Meteor 实时性的特点。我们将在下一章向你展示该如何去修改！

动态代码重载

你可能已经注意到，当你修改文件的时候，不需要手动刷新页面，浏览器就会自动重新加载。

这是因为 **Meteor** 跟踪了项目目录下的所有文件，当检测到其中一个文件发生改变，它就会自动刷新你的浏览器。

Meteor 的动态代码重载是相当智能的，它甚至可以在两个刷新动作之间保存你的 App 状态。

使用 Git 和 GitHub

SIDEBAR

3.5

GitHub 是一个开源项目的社交化代码存储空间，基于 **Git** 作为版本控制系统。它的首要功能就是代码共享和项目协作。在本章你可以快速找到用 GitHub 学习本书的一些方法。

本章节假设你不太了解 Git 和 GitHub。如果你已经熟悉他们了，你可以直接跳到下一章！

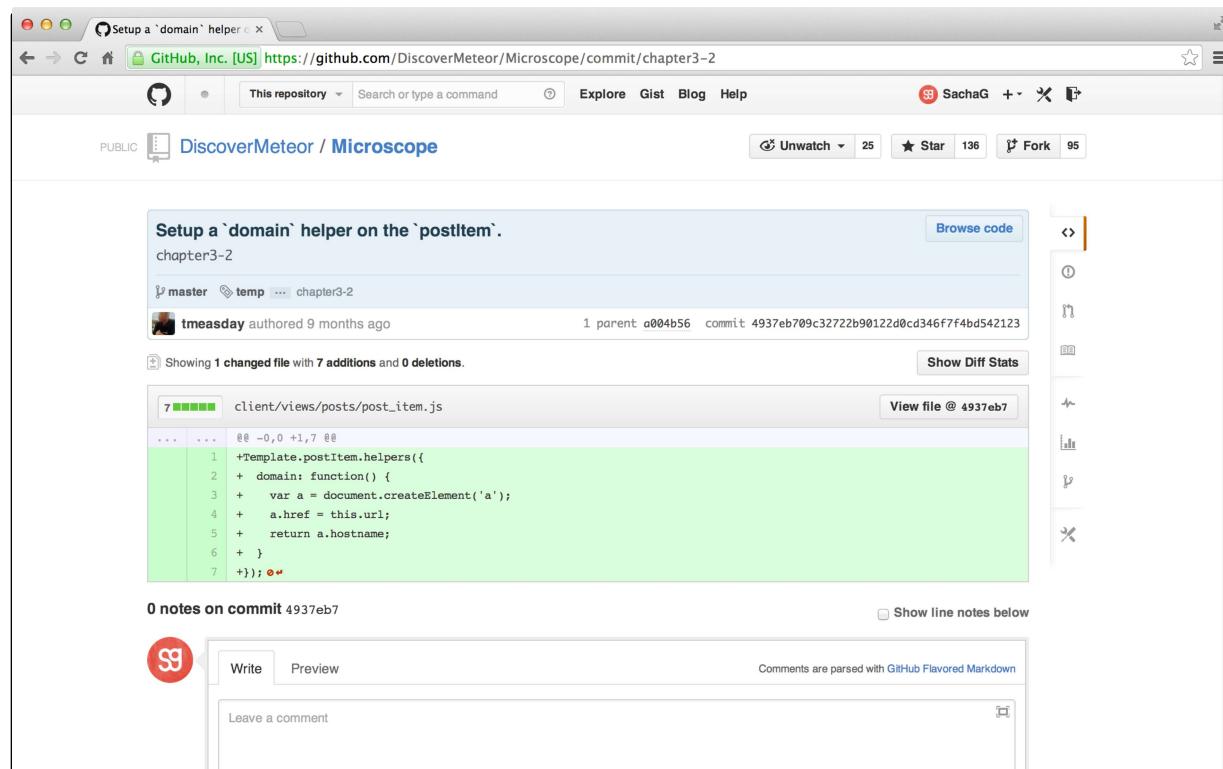
代码提交

Git 最基本的工作单元是提交。你可以把提交设想为你的代码库在某个特定时间的一个快照。

与其简单地给你一个 Microscope 项目的最终代码版本，我们更愿意把开发过程中每一步的快照都提供出来，这样你在 Github 上在线看到。

比如，这个就是我们上一章最后一次提交

(<https://github.com/DiscoverMeteor/Microscope/commit/chapter3-2>) 看起来是这样的：



GitHub 上的一次代码提交

你可以看到 `post_item.js` 这个文件的“diff”（差异），换句话说就是这次提交改动了这个文件的什么地方。因为我们这个文件是新建的，所以你可以看到所有内容都是绿色高亮。

让我们对比一下另外一个例子本书中以后会用到的文件：

Added basic upvoting algorithm.

chapter13-1

master chapter14-2 ... chapter13-1

tmeasday authored 9 months ago 1 parent 5b2a619 commit abfd3124f9154eeac51d43ef5c0ddc4a5314d306

Showing 4 changed files with 38 additions and 5 deletions. [View file @ abfd312](#) [Show Diff Stats](#)

client/views/posts/post_item.html	View file @ abfd312
-----------------------------------	---------------------

```

@@ -1,8 +1,10 @@
 1   1     <template name="postItem">
 2   2       <div class="post">
 3   3         + <a href="#" class="upvote btn">↑</a>
 4   4           <div class="post-content">
 5   5             <h3><a href="{{url}}>{{title}}</a><span>{{domain}}</span></h3>
 6   6             <p>
 7   7               {{votes}} Votes,
 8   8               submitted by {{author}},
 9   9               <a href="{{pathFor 'postPage'}}">{{commentsCount}} comments</a>
10  10               {{#if ownPost}}<a href="{{pathFor 'postEdit'}}>Edit</a>{{/if}}</p>

```

client/views/posts/post_item.js	View file @ abfd312
---------------------------------	---------------------

```

@@ -7,4 +7,11 @@
 7   7     a.href = this.url;
 8   8     return a.hostname;

```

修改代码

这次，只有修改的代码行被高亮为绿色了。

当然，有时候我们并不增加和修改代码，而是直接删除某些行：

The screenshot shows a GitHub pull request page. The title of the pull request is "Augmented the postsList route to take a limit" and it is associated with the commit "chapter12-2". The author is tmeasday, who authored the code 9 months ago. The pull request has 1 parent, commit c7af59e, and 17 additions and 10 deletions across 3 changed files. The main part of the screenshot displays two code diff panels. The first panel shows changes in "client/views/posts/posts_list.js", where line 5 has been deleted. The second panel shows changes in "lib/router.js", where line 5 has been added. The GitHub interface includes a sidebar with various icons and a vertical scroll bar.

删除代码

好了，我们现在看到 GitHub 的第一个好处：一览代码的改动。

浏览提交的代码

Git 的提交视图给我们显示了本次提交的代码改动，但是有时候我们还是需要看看没有修改的那些代码，从而确认他们的代码看起来合理。

好，让 GitHub 再次来解决这个问题。在提交页面上点击 **Browse code**（浏览代码）按钮：

The screenshot shows a GitHub commit detail page. The commit title is "Setup a `domain` helper on the `postItem`." and it's associated with chapter3-2. The commit was authored by tmeasday 9 months ago, with a parent commit a004b56 and a commit hash 4937eb709c32722b90122d0cd346f7f4bd542123. The file client/views/posts/post_item.js has been modified with 7 additions and 0 deletions. A red box highlights the "Browse code" button. Below the code editor, there are notes on the commit and a comment section.

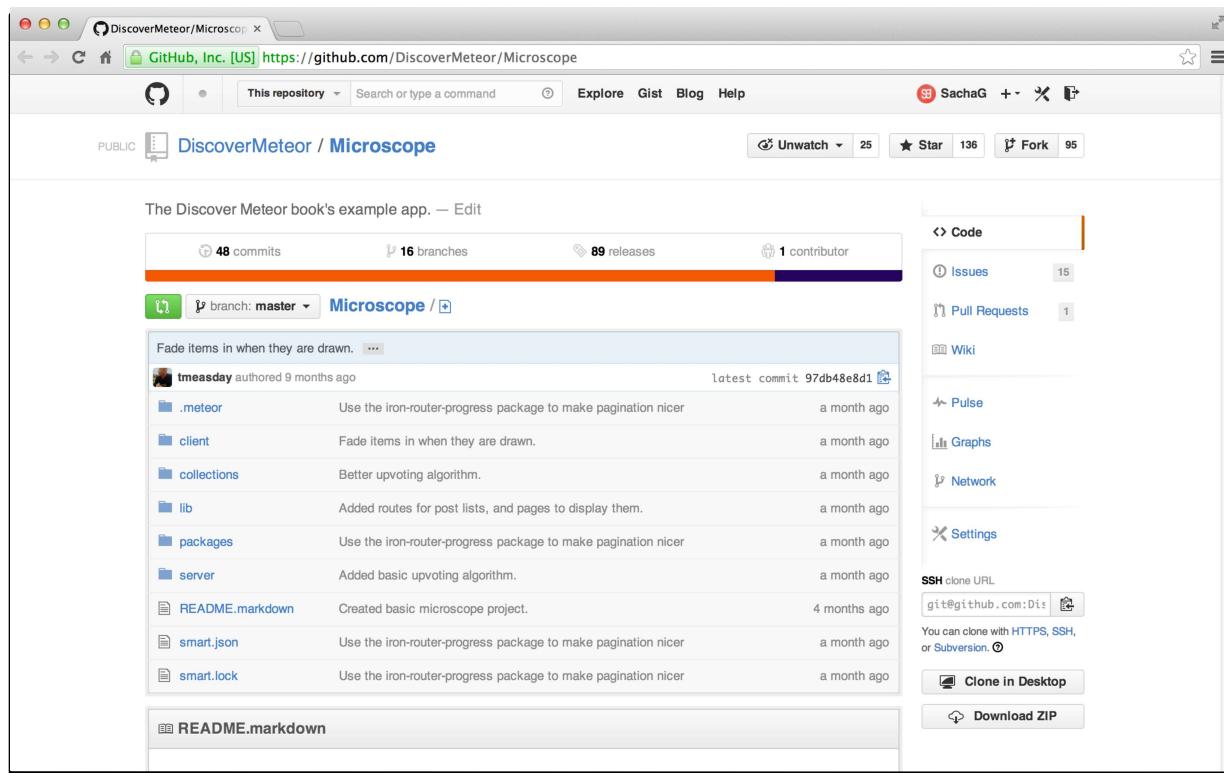
浏览代码按钮

你现在可以看到某次提交的时候代码库的样子了。

The screenshot shows the GitHub repository tree for the commit 4937eb709c32722b90122d0cd346f7f4bd542123. The repository name is DiscoverMeteor / Microscope. The tree view shows 5 commits, 16 branches, 89 releases, and 1 contributor. The latest commit is 4937eb709c. The commit details show "Setup a `domain` helper on the `postItem`." by tmeasday 9 months ago. The commit message also includes "Added bootstrap package." and "Created basic microscope project." The repository sidebar shows issues (15), pull requests (1), wiki, pulse, graphs, network, and settings. It also provides an SSH clone URL and download options.

提交编号 3-2 的代码库。

GitHub 没有给我们足够的视觉提示让我们知道我们正在看一个提交，不过你可以通过与“正常”的主视图进行比较，就会发现文件结构的不同了：



提交编号 14-2 的代码库。

访问本地提交

我们刚刚看到在 GitHub 上如何在线浏览某个提交的整个代码。但是你是否想在本地也看到呢？比如你也许想退回到某次提交的代码状态测试一下那时候运行的样子。

想做到这个你需要首次（也许你早已经不是首次了，但是至少在本书中是第一次用命令行）用 git 命令行。对于新手首先要确定你已经安装了 Git。然后克隆 **Clone**（或者叫下载一份本地拷贝）一份 Microscope 的代码库。命令如下：

```
git clone git@github.com:DiscoverMeteor/Microscope.git github_microscope
```

命令行最后的 `github_microscope` 实际上就是将会存代码库的本地目录名。假设你已有了

microscope 文件夹，那就随便起一个新名字（不是必须用和 GitHub 代码库相同的名字）。

让我们 `cd` 进这个代码库，然后就可以使用 `git` 命令了。

```
cd github_microscope
```

现在我们已经从 GitHub 克隆了代码库，我们已经下载了这个应用的全部代码，也就是说我们正在看的是最后一次提交。

值得感激的是我们有办法 `check out` (签出) 代码回退到某一个特定的提交，而不会影响到其他提交。让我们试一下：

```
git checkout chapter3-1
Note: checking out 'chapter3-1'.
```

你现在是在 '`detached HEAD`' 状态。你可以随便看看做点实验性的修改并提交。你只需再次签出代码即可放弃你的提交，而不会影响的任何其他分支。

如果你想建立新的分支来保留你的提交，你可以这样做（现在或者以后也行），迁出代码的时候使用 `-b` 参数。

举例：

```
git checkout -b new_branch_name
HEAD is now at a004b56... Added basic posts list template and static data.
```

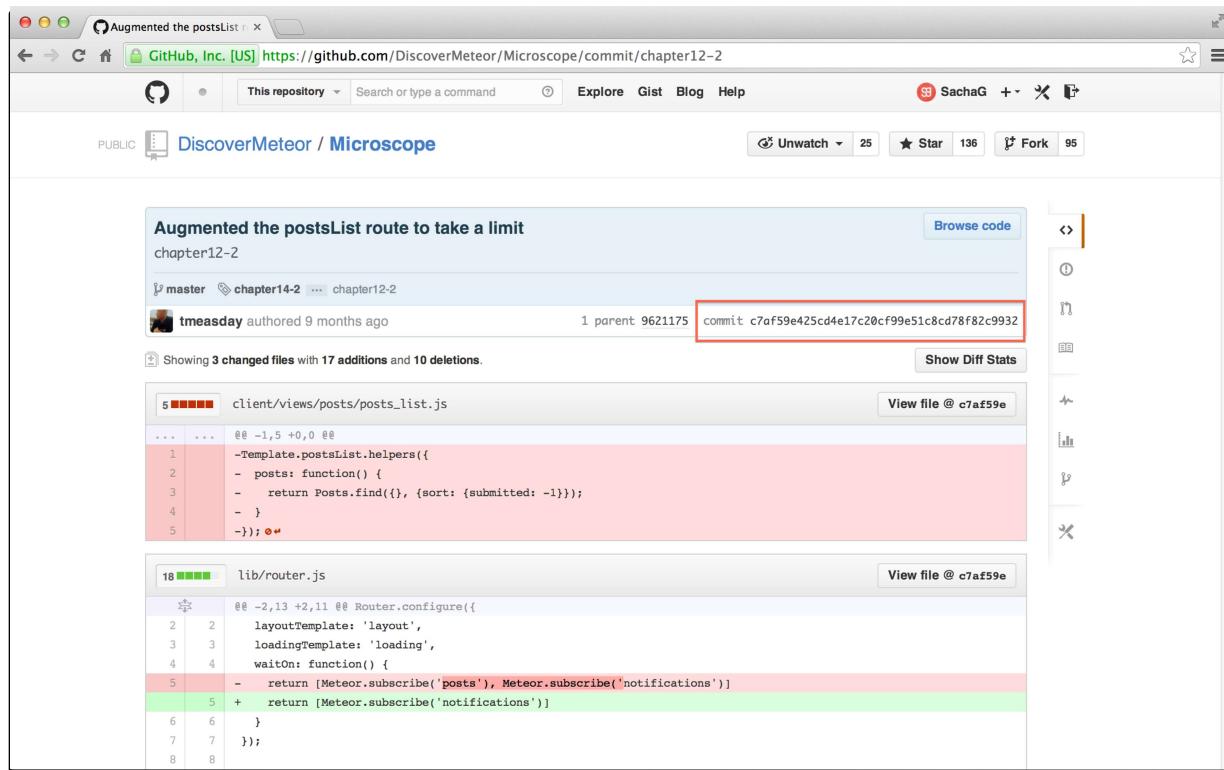
Git 通知你现在在‘分离头’“`detached HEAD`”状态，也就是说就 Git 而言，你可以发现过去的历史提交但是不能修改。你可以想象一个巫婆用水晶球回顾历史。

（注意 Git 还有让你改变历史提交的命令。这更像是时间旅行，不过那些不是我们本书需要讨论的范畴了。）

你能够简单地输入 `chapter3-1` 是因为我们实现已经把所有的 Microscope 的提交都打了标签

了。如果你的提交没有标签，那么你需要先找到你的提交的哈希码 **hash**，或者ID。

再一次 GitHub 让我们可以轻松地在提交的右下角看到蓝色提交框中提交哈希码。如图所示：



找到提交的哈希码

这一次让我们试试哈希码而不是标签：

```
git checkout c7af59e425cd4e17c20cf99e51c8cd78f82c9932
Previous HEAD position was a004b56... Added basic posts list template and static data.
HEAD is now at c7af59e... Augmented the postsList route to take a limit
```

好了，别再用水晶球看历史提交了，让我们想看看最新的代码状态，我们可以让 Git 签出主分支**master**：

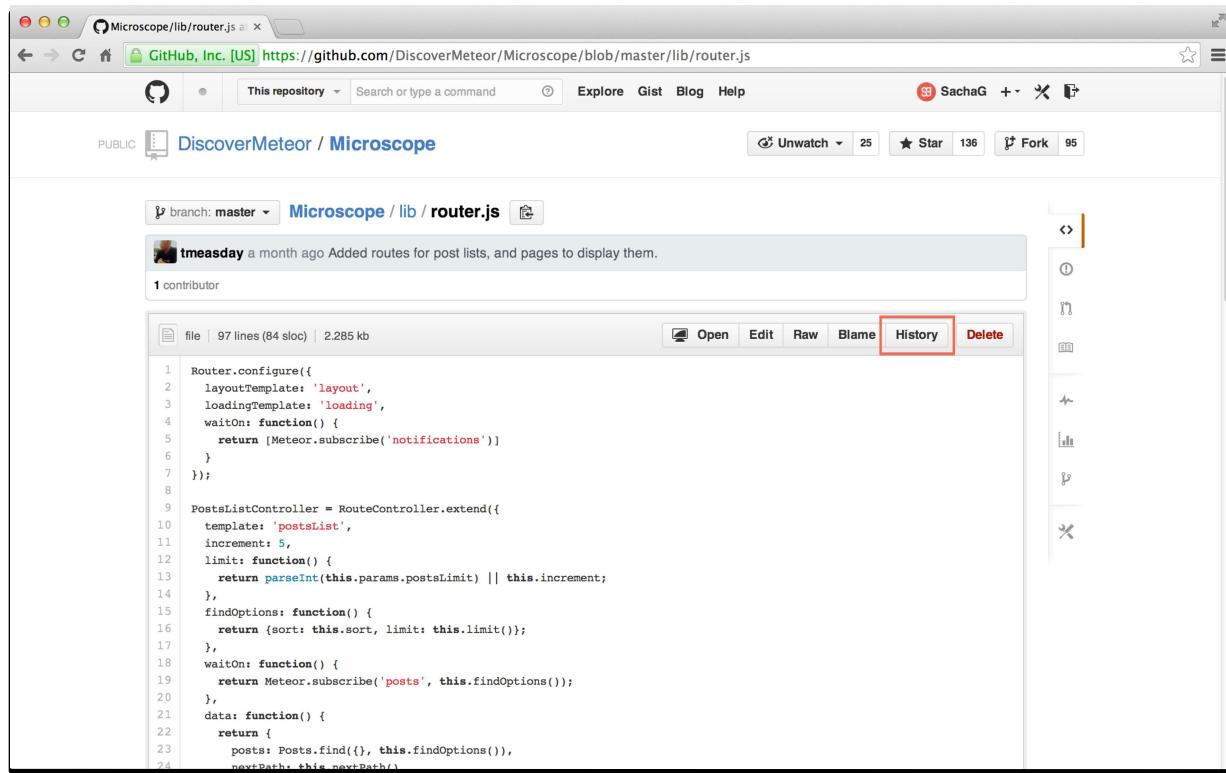
```
git checkout master
```

注意，你也可以现在运行 `meteor` 命令，即使是在“detached HEAD”的状态下。你也许首先需要运行一下 `meteor update` 命令，如果 Meteor 提示有丢失的代码包，因为 Microscope 的 Git 代码库没有包括包代码。

Historical Perspective

这里还有一种常见的情形：你看代码文件的时候发现有一些你以前没有见到过的改动。大多数情况是你不记得什么时候你改了这个文件。你可以逐个检查每个提交直到你发现某个提交造成了这个改动，不过还有一种更好的方法，那就是 GitHub 的历史功能。

首先，在 GitHub 上找一个代码库里的文件，然后找到“历史”按钮：



GitHub 的历史按钮

你现在可以看到所有影响这个文件的提交历史：

The screenshot shows a GitHub repository page for 'DiscoverMeteor / Microscope'. The specific file 'lib/router.js' is selected. The commit history for this file is displayed, dated Dec 11, 2013. The commits are listed in reverse chronological order, starting with 'Added routes for post lists, and pages to display them.' and ending with 'Monitor which errors have been seen, and clear on routing.'. Each commit includes a small profile picture of the author, the commit message, the date it was authored, and a 'Browse code' link.

Date	Commit Message	Author	Code Link
Dec 11, 2013	Added routes for post lists, and pages to display them.	tmeasday	6c6db6d1f12
	Use a single post subscription to ensure that we can always see the r...	tmeasday	5b2a619632
	Use the iron-router-progress package to make pagination nicer	tmeasday	51c772d39c
	Added nextPath() to the controller and use it to step through posts.	tmeasday	46127f19cd
	Refactored postsLists route into a RouteController	tmeasday	64dfeb0000
	Augmented the postsList route to take a limit	tmeasday	c7af59e425
	Added basic notifications collection.	tmeasday	3234b21edd
	Made a simple publication/subscription for comments.	tmeasday	a179bec0cb
	Added comments collection, pub/sub and fixtures.	tmeasday	c702efa8fe
	Monitor which errors have been seen, and clear on routing.	tmeasday	09259b59b9

显示一个文件的历史

问责游戏

让我们看看问责：

This screenshot shows a GitHub repository page for the 'Microscope' project. The file 'posts.js' is open, displaying its code. At the top of the code editor, there are several tabs: 'Edit', 'Raw', 'Blame' (which is highlighted with a red box), and 'History'. The code itself is a Meteor Collection named 'Posts' with various methods and permissions defined.

```

1 Posts = new Meteor.Collection('posts');
2
3 Posts.allow({
4   update: ownsDocument,
5   remove: ownsDocument
6 });
7
8 Posts.deny({
9   update: function(userId, post, fieldNames) {
10     // may only edit the following two fields:
11     return (_.without(fieldNames, 'url', 'title').length > 0);
12   }
13 });
14
15 Meteor.methods({
16   post: function(postAttributes) {
17     var user = Meteor.user(),
18       postWithSameLink = Posts.findOne({url: postAttributes.url});
19
20     // ensure the user is logged in
21     if (!user)
22       throw new Meteor.Error(401, "You need to login to post new stories");
23
24     // ensure the post has a title

```

GitHub 的问责按钮

这个简洁的视图给我们逐行显示了谁修改过这个文件，以及在哪次提交中修改的。（换句话说，知道软件搞坏了该找谁算账）：

This screenshot shows the 'Blame' view for the 'posts.js' file. The 'Blame' tab is selected, showing a history of commits and their changes. Each commit is listed with its author, date, and a brief description of the changes made. The code is shown in a split-view format, with the original code on the left and the modified code on the right.

Commit	Date	Author	Description	Modified Lines
589ef5d5	2013-04-07	tmeasday	Added a posts collection	1
7b001288	2013-04-07	tmeasday	Removed insecure, and allowe...	2
46fb2c5c	2013-04-07	tmeasday	Added basic permission to ch...	3
ff6b77cb	2013-04-07	tmeasday	Only allow changing certain ...	8
46fb2c5c	2013-04-07	tmeasday	Added basic permission to ch...	14
c71f9b52	2013-04-07	tmeasday		15

GitHub 问责视图

现在的 Git 已经是一个相当复杂的工具了 - GitHub 也一样 -，所以你不可能在这里覆盖所有功能。事实上，我们只是学习了一点皮毛。尽管如此，这点皮毛已经够我们在本书的学习中用了。

集合

在第一章我们提到了 Meteor 的核心功能，那就是服务器端和客户端的自动数据同步。

在这一章我们要仔细了解一下它是如何运作的，以及研究那个让它得以运行的关键技术：Meteor 集合（**Collection**）。

集合是一个特殊的数据结构，它将你的数据存储到持久的、服务器端的 MongoDB 数据库中，并且与每一个连接的用户浏览器进行实时地同步。

我们想让我们的 `post` 永久保存并且要在用户之间共享，所以我们一开始要新建一个叫做 `Posts` 的 collection 来保存它们。

我们现在做一个社交新闻应用，所以第一件事儿就是做一个人们贴上来的帖子的连接列表。我们叫它 ‘post’

很自然，我们需要把它们存起来。Meteor 绑定了 MongoDB 运行在服务器上作为持久化存储。

因此，尽管一个用户在浏览器上有各种状态(比如他们正在阅读哪一页，或者正在输入那一条评论)，而服务器上，尤其是 Mongo，保存的是永久保留的一致数据。说到一致，我们是指对于所有用户来说都是一样的数据：每个用户也许在看不同的页面，但是帖子 Post 的主列表对所有用户来说却始终是一样的。

这些数据在 Meteor 中被存储在集合（**Collection**）中。集合是一种特殊的数据结构，通过发布（`publications`）和订阅（`subscriptions`）机制把数据实时同步上行或者下行到连接着的各个用户的浏览器或者 Mongo 数据库中。让我们看看如何做到的。

我们希望我们的帖子 Post 可以持久存储并分享给用户们，所以我们一开始就要建立一个叫 `Posts` 的集合来存储他们。如果你还没有在根文件夹建立一个叫做 `collections/` 的文件夹，并在里面放一个 `posts.js` 的文件的话，那现在就加上。

```
Posts = new Mongo.Collection('posts');
```

lib/collections/posts.js

提交 4-1 增加一个 post 集合

[在 GitHub 中查看](#) [查看演示](#)

代码所在的目录既不是 `client/` 也不是 `server/` 所以 `Posts` 会共同存在运行在服务器和客户端。然而，这个集合的使用在两种环境下十分不同。

要 Var 还是不要 Var?

在 Meteor 中，关键字 `var` 限制对象的作用域在文件范围内。我们想要 `Posts` 作用于整个应用范围内，因此我们在这里不要 Var 这个关键字。

存储数据

网络应用有三种基本方式保存数据，各种方式有不同的角色：

- 浏览器内存：像 JavaScript 变量的这些数据会保存在浏览器内存中，意味着他们不是永久性的：它们存在于当前浏览器标签中，当标签关闭后它们会消失。
- 浏览器存储：浏览器也可存储较为永久性的数据，使用 `cookies` 或本地存储 **LocalStorage**。虽然数据会在不同 `session` 间保持，但是只是针对于当前用户（包括标签之间）但不能轻易地共享给其他用户。
- 服务器端数据库：你想永久保存数据并且提供给多个用户的最好方法是数据库

(MongoDB 是 Meteor 应用默认的方案)。

Meteor 使用所有三种方式，有时会从一个地方同步数据到另一个地方（我们会马上看到）。话虽如此，数据库仍然是包含数据主副本的“规范化的”数据源。

客户端与服务器

不在 `client/` 或 `server/` 文件夹中代码会在客户端和服务器端运行。所以 `Posts` 集合在客户端和服务器端都可用。但是，在各自环境下所起的作用有很大不同。

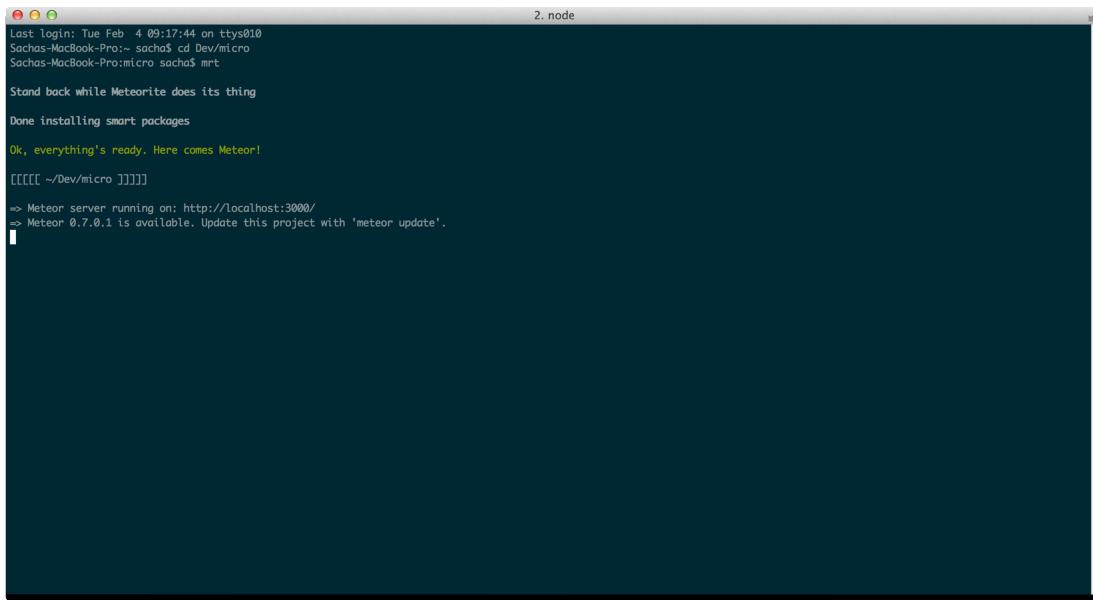
在服务器，集合有一个任务就是和 Mongo 数据库联络，读取任何数据变化。在这种情况下，它可以比对标准的数据库。

在客户端，集合是一个安全拷贝来自于实时一致的数据子集。客户端的集合总是（通常）透明地实时更新数据子集。

Console, Console 与 Console

在这一章，我们开始使用浏览器控制台，不过不要和终端、**Meteor Shell** 或者 **Mongo Shell** 搞混了。现在对它们做个比对。

终端命令行（Terminal）



```
Last login: Tue Feb  4 09:17:44 on ttys010
Sachas-MacBook-Pro:~ sachas$ cd Dev/micro
Sachas-MacBook-Pro:micro sachas$ mrt

Stand back while Meteorite does its thing

Done installing smart packages

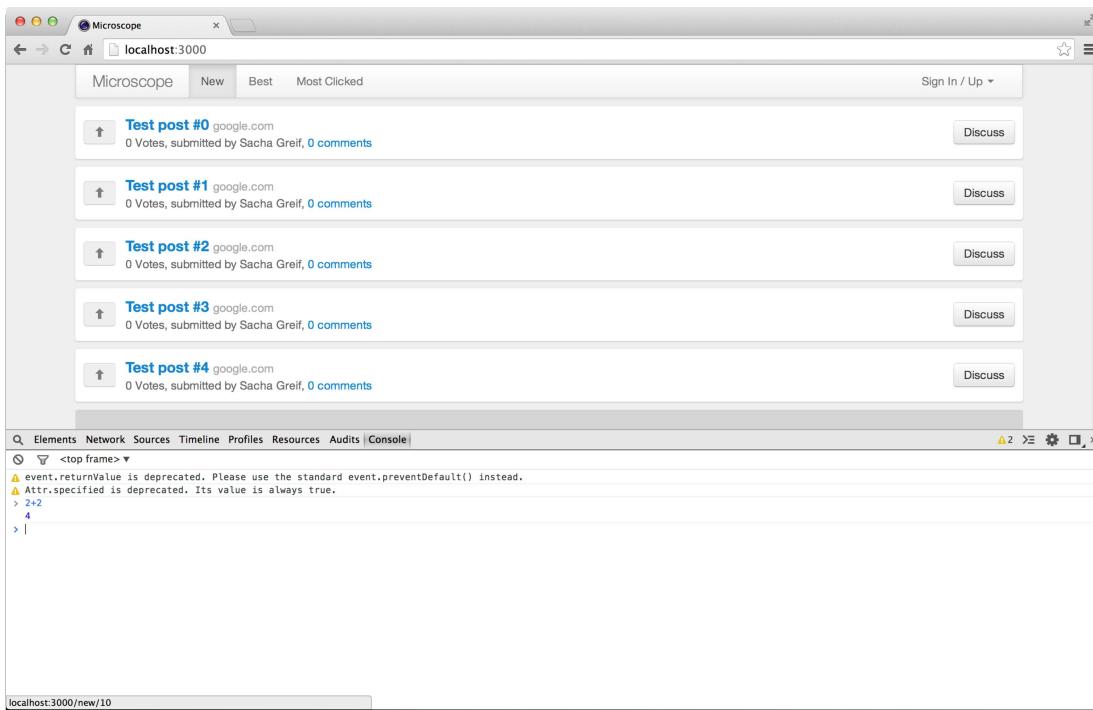
Ok, everything's ready. Here comes Meteor!
[[[[[[ ~/Dev/micro ]]]]]]

=> Meteor server running on: http://localhost:3000/
=> Meteor 0.7.0.1 is available. Update this project with 'meteor update'.
```

终端命令行

- 由操作系统启动
- 服务器端 `console.log()` 会输出到这里
- 有 `$` 提示符
- 通常也被成为外壳程序 Shell, Bash

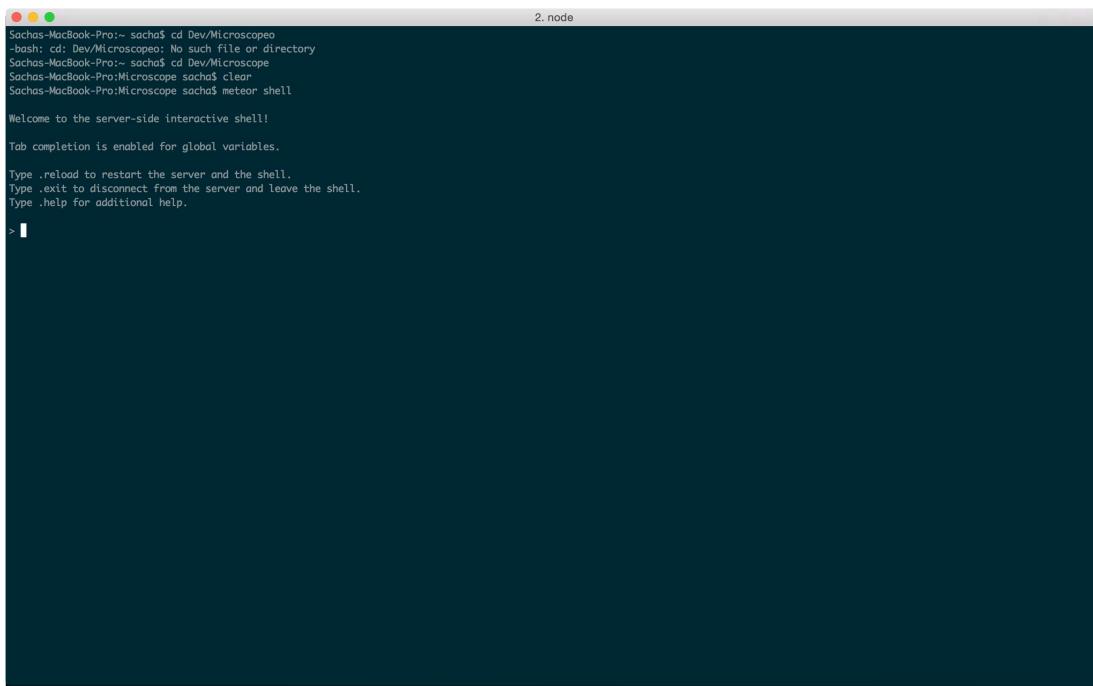
浏览器控制台（Browser Console）



The Browser Console

- 在浏览器内启动，执行 Javascript 代码
- 客户端的 `console.log()` 会输出到这里
- 提示符是 `>`
- 也通常被称作 Javascript 控制台或者开发工具控制台（DevTools Console）

Meteor Shell



Meteor Shell

- 在 Terminal 用 `meteor shell` 调用。
- 使你直接接触到应用的服务器端代码。
- 提示符： `>`。

Mongo 外壳程序 (Mongo Shell)

```

node                               node
Sachas-MacBook-Pro:- sachas$ cd Dev/micro
Sachas-MacBook-Pro:micro sachas$ mrt mongo
Stand back while Meteorite does its thing
Done installing smart packages
Ok, everything's ready. Here comes Meteor!

MongoDB shell version: 2.4.4
connecting to: 127.0.0.1:3002/meteor
> db.posts.find()
[{"title": "Introducing Telescope", "userId": "3w4cWap35gyt2wb7M", "author": "Sacha Greif", "url": "http://sachagreif.com/introducing-telescope/", "submitted": 1391453182226, "commentsCount": 2, "upvoters": [], "votes": 0, "_id": "p5NipPM4JdltFvQlo"}, {"title": "Meteor", "userId": "b43r74nbK858CtY4j", "author": "Tom Coleman", "url": "http://meteor.com", "submitted": 1391442382226, "commentsCount": 0, "upvoters": [], "votes": 0, "_id": "3Fe3umbijZed6nH82"}, {"title": "The Meteor Book", "userId": "b43r74nbK858CtY4j", "author": "Tom Coleman", "url": "http://themeteorbook.com", "submitted": 139145182226, "commentsCount": 0, "upvoters": [], "votes": 0, "_id": "wJhu0HMiTyuzhzE5Lm"}, {"title": "Test post #0", "author": "Sacha Greif", "userId": "3w4cWap35gyt2wb7M", "url": "http://google.com/?q=test-0", "submitted": 1391478382227, "commentsCount": 0, "upvoters": [], "votes": 0, "_id": "f29hT8merJpc0ay"}, {"title": "Test post #1", "author": "Sacha Greif", "userId": "3w4cWap35gyt2wb7M", "url": "http://google.com/?q=test-1", "submitted": 1391474782227, "commentsCount": 0, "upvoters": [], "votes": 0, "_id": "2mbaZL6msmNtnou"}, {"title": "Test post #2", "author": "Sacha Greif", "userId": "3w4cWap35gyt2wb7M", "url": "http://google.com/?q=test-2", "submitted": 1391471182227, "commentsCount": 0, "upvoters": [], "votes": 0, "_id": "NKemnBRtpsaGhfly"}, {"title": "Test post #3", "author": "Sacha Greif", "userId": "3w4cWap35gyt2wb7M", "url": "http://google.com/?q=test-3", "submitted": 1391467582227, "commentsCount": 0, "upvoters": [], "votes": 0, "_id": "HNGEfF6z2PrdWsvsk"}, {"title": "Test post #4", "author": "Sacha Greif", "userId": "3w4cWap35gyt2wb7M", "url": "http://google.com/?q=test-4", "submitted": 1391463982227, "commentsCount": 0, "upvoters": [], "votes": 0, "_id": "TmW3NQ6r78nCKx8A"}, {"title": "Test post #5", "author": "Sacha Greif", "userId": "3w4cWap35gyt2wb7M", "url": "http://google.com/?q=test-5", "submitted": 1391460382227, "commentsCount": 0, "upvoters": [], "votes": 0, "_id": "T9EKnx23zGg5o2EM"}, {"title": "Test post #6", "author": "Sacha Greif", "userId": "3w4cWap35gyt2wb7M", "url": "http://google.com/?q=test-6", "submitted": 1391456782227, "commentsCount": 0, "upvoters": [], "votes": 0, "_id": "itKpRWyy8q4nBjXK"}, {"title": "Test post #7", "author": "Sacha Greif", "userId": "3w4cWap35gyt2wb7M", "url": "http://google.com/?q=test-7", "submitted": 1391453182227, "commentsCount": 0, "upvoters": [], "votes": 0, "_id": "FRKRY0q9pxFuLFzS"}, {"title": "Test post #8", "author": "Sacha Greif", "userId": "3w4cWap35gyt2wb7M", "url": "http://google.com/?q=test-8", "submitted": 1391449582227, "commentsCount": 0, "upvoters": [], "votes": 0, "_id": "rqWhxxyCyHjy3hNS"}, {"title": "Test post #9", "author": "Sacha Greif", "userId": "3w4cWap35gyt2wb7M", "url": "http://google.com/?q=test-9", "submitted": 1391445982227, "commentsCount": 0, "upvoters": [], "votes": 0, "_id": "cFPPrvFsRtJap2q2M"}]
> |
```

Mongo 外壳

- 从终端由 `meteor mongo` 或者 `mrt mongo` 来启动
- 你可以在这里直接操作 App 的数据库
- 提示符 `>`
- 也被称作 Mongo 控制台 (Mongo Console)

注意在各种情况下你都不需要敲提示符（`$` 或 `>`）在命令前面。而且你可以认定任何不是用提示符起始的行都是前一个命令的输出结果。

服务器端的集合

在服务器端，集合可以像 API 一样操作 Mongo 数据库。在服务器端的代码，你可以写像 `Posts.insert()` 或 `Posts.update()` 这样的 Mongo 命令，来对 Mongo 数据库中的 `posts`

集合进行操作。

如果想直接看看 MongoDB 数据库，可以打开第二个终端窗口（这时候 Meteor 还在第一个终端窗口继续运行呢），在你应用的目录，输入命令 `meteor mongo` 启动 Mongo Shell 外壳程序。现在你可以输入标准的 Mongo 命令（如同以往，你可以敲 `ctrl+c` 快捷键退出）。比如让我们插入一个新的 post：

```
meteor mongo

> db.posts.insert({title: "A new post"});

> db.posts.find();
{ "_id": ObjectId("..."), "title" : "A new post"};
```

Mongo Shell

Meteor.com 上的 Mongo

注意如果你把应用部署在 `*.meteor.com` 上，你一样可以通过 `meteor mongo myApp` 的方式进入你应用的 Mongo shell 进行操作。

而且你还可以输入 `meteor logs myApp` 得到你应用的 log 日志。

Mongo 的语法由于借鉴了 Javascript 的语法所以十分熟悉。我们现在在 Mongo 外壳里不做过多的数据操作，不过我们可以随时来这里检查数据确保他们正常存在。

客户端集合

客户端的集合更加有趣。当你在客户端申明 `Posts = new Mongo.Collection('posts');`，你实际上是创建了一个本地的，在浏览器缓存中的真实的 Mongo 集合。当我们说客户端集合被“缓存”是指它保存了你数据的一个子集，而且对这些数据提供了十分快速的访问。

有一点我们必须要明白，因为这是 Meteor 工作的一个基础：通常说来，客户端的集合的数据是你 Mongo 数据库的所有数据的一个子集（毕竟我们不会想把整个数据库的数据全传到客户端来）。

第二，那些数据是被存储在浏览器内存中的，也就是说访问这些数据几乎不需要时间，不像去服务器访问 `Posts.find()` 那样需要等待，因为数据事实上已经载入了。

介绍 MiniMongo

Meteor 的客户端 Mongo 的技术实现被称为 MiniMongo。它目前还不是一个完美的实现，而且你会发现偶尔 Mongo 的功能在这里不能实现。不过本书中涉及到的功能都是可以在 Mongo 和 MiniMongo 中实现的。

客户端-服务器通讯

这一切最关键的是如何让客户端的集合数据与服务器端同名的集合数据同步（以我们现在这个例子来说是 `posts`）。

与其现在就解释细节不如让我们先来看看发生了什么。

现在我们打开两个浏览器窗口，分别打开他们的浏览器控制台。然后在终端命令行打开 Mongo 外壳程序。

现在可以在这三个地方看到我们早前时候建立的那个文档。（注意，我们应用的用户界面依然显示着我们之前的三个演示 post，请忽略它们。）

```
> db.posts.find();
{title: "A new post", _id: ObjectId("...")};
```

Mongo 外壳

```
> Posts.findOne();
{title: "A new post", _id: LocalCollection._ObjectID};
```

第一个浏览器控制台

让我们来创建一个帖子。在其中一个浏览器窗口中运行这个插入命令：

```
> Posts.find().count();
1
> Posts.insert({title: "A second post"});
'xxx'
> Posts.find().count();
2
```

第一个浏览器控制台

毫无疑问，这个帖子被加入到本地集合中。现在让我们查看一下 Mongo：

```
> db.posts.find();
{title: "A new post", _id: ObjectId("...")};
{title: "A second post", _id: 'yyy'};
```

Mongo 外壳

如同你所看见的那样，这个帖子一路上行一直到 Mongo 数据库中，而我们却没有为这个连接客户端和服务器的过程写任何一行代码。（严格地说，我们的确写了一行代码： `new Mongo.Collection('posts')`）。但是这没关系！

现在到第二个浏览器窗口的控制台中输入这个命令：

```
› Posts.find().count();
2
```

第二个浏览器的控制台

这个帖子居然也在这儿！甚至于我们连刷新都没有在第二个浏览器做过，更何况我们也没有写任何代码来推送更新。这一切像魔术一般 - 而且是即时的，尽管这一切以后看起来都很显而易见。

实际情况是服务器端的集合被客户端的集合通知说有一个新帖子，然后执行了一个任务把这个帖子放入 Mongo 数据库，进而会送到所有连接着的 `post` 即可。

在浏览器的控制台取出所有的帖子没什么用处。我们以后会学习如何把这些数据显示在模板中，并把这个简单的 HTML 原型变成一个有用的实时 Web 应用。

保持实时

从浏览器控制台看到集合算是一件事儿，我们更应该关注的是能在屏幕上显示数据和数据的变化。要做到这一点，我们需要把我们的应用从一个单一显示静态数据的 页面 变成可以实时动态数据的 应用 。

让我们看怎么做。

从数据库提取数据

首先我们先放点数据在数据库里。我们要做的是让服务器第一次初始启动的时候从一个数据文件中读取数据结构存在 `Posts` 集合中。

首先我们要确保数据库中没有数据。我们使用 `meteor reset` 命令清空数据库初始化我们的项目。当然，如果在真实的正在运行的正式项目上请务必十分小心。

停止 Meteor 服务（通过键入 `ctrl-c`）然后在命令行输入：

```
meteor reset
```

这个 `reset` 命令彻底地把 Mongo 数据库清空了。在开发的时候这个命令很有用，尤其当我们的数据库发生数据混乱的时候。

现在重启我们的 Meteor 应用：

```
meteor
```

现在数据库已经清空，我们可以增加下面的代码以便在服务器启动时候检查数据库 `Posts` 集合，如果为空则载入三条帖子。

```
if (Posts.find().count() === 0) {
  Posts.insert({
    title: 'Introducing Telescope',
    url: 'http://sachagreif.com/introducing-telescope/'
  });

  Posts.insert({
    title: 'Meteor',
    url: 'http://meteor.com'
  );

  Posts.insert({
    title: 'The Meteor Book',
    url: 'http://themeteorbook.com'
  });
}
```

`server/fixtures.js`

提交 4-2 在 posts 集合中加入数据.

在 GitHub 中查看

查看演示

我们把这个文件放到了 `server/` 目录中，因此永远不会被加载到任何用户的浏览器中。这段代码在服务器启动的时候会立即运行，然后调用 插入 功能在数据库的 `posts` 集合中插入三条简单的帖子。因为我们还没有加入任何数据安全功能，所以无论在服务器还是在客户端运行这个文件都事实上没有区别的。

现在我们用 `meteor` 命令启动服务，这三条帖子会被装在到数据库中。

动态数据

现在如果我们打开一个浏览器的控制台，我们可以看到这三个帖子都被转载到 MiniMongo 中了：

```
› Posts.find().fetch();
```

浏览器控制台

要把这些 `post` 渲染到 HTML 中，我们需要用模板 `helper`。

在第三章中，我们看到 Meteor 允许我们把 数据上下文 绑定到我们的 Spacebars 模板上，从而用 HTML 视图显示这些简单数据结构。我们可以同样把我们的集合数据绑定起来。我们马上就替换掉静态的 `postsData` Javascript 对象成为一个动态地集合。

现在请随手删掉 `postsData` 代码。下面是 `posts_list.js` 修改后的样子：

```
Template.postsList.helpers({  
  posts: function() {  
    return Posts.find();  
  }  
});
```

client/templates/posts/posts_list.js

提交 4-3

把集合连接到 `postsList` 模板上。

在 [GitHub](#) 中查看

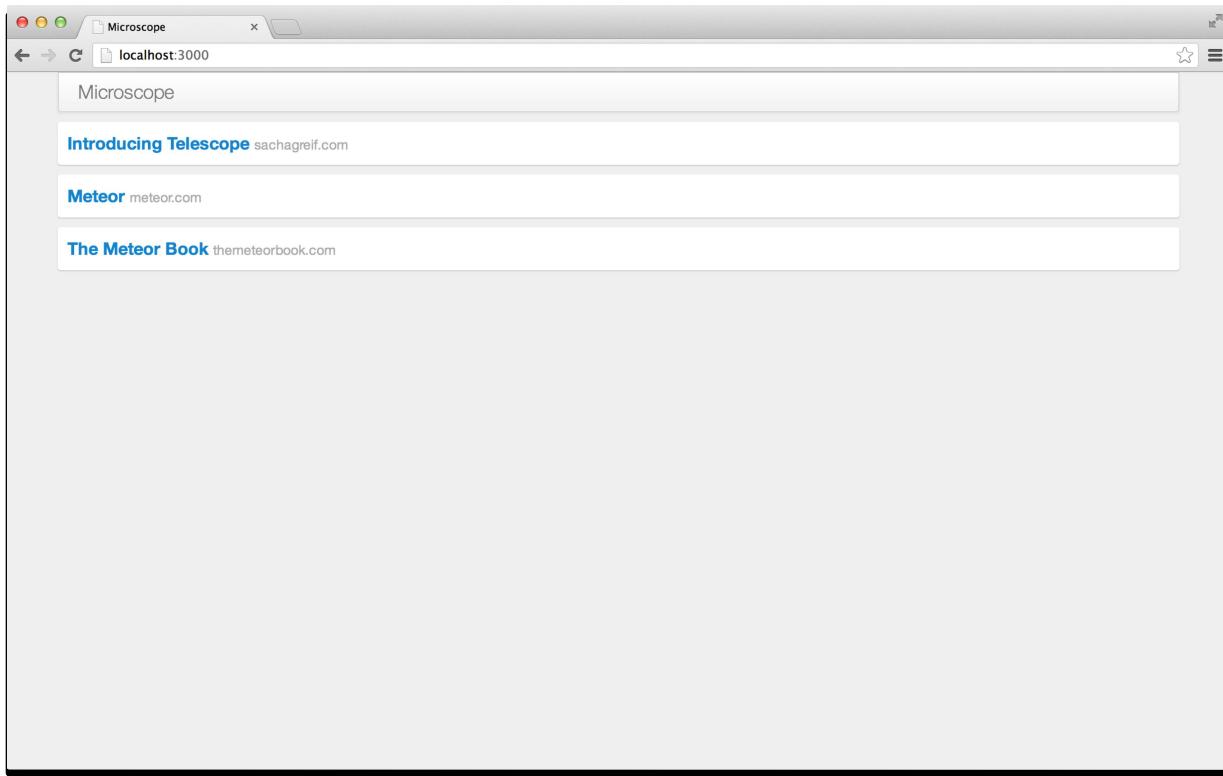
[查看演示](#)

查找与提取

在 Meteor 中，`find()` 返回值是一个游标。游标是一种从动数据源。如果你想输出内容，你可以对游标使用 `fetch()` 来把游标转换成数组。

Meteor 十分智能地在应用中保持游标状态而避免动不动就把游标变成数组。这就造成了你不会经常在 Meteor 代码中看到 `fetch()` 被调用（基于同样原因，我们在上述例子中也没有使用 `fetch()`）。

现在，与其把帖子们变成静态的数组，不如直接把游标赋给 `posts` 帮助方法。但是如何做到呢？如果我们回到浏览器上，我们可以看到：



使用活数据

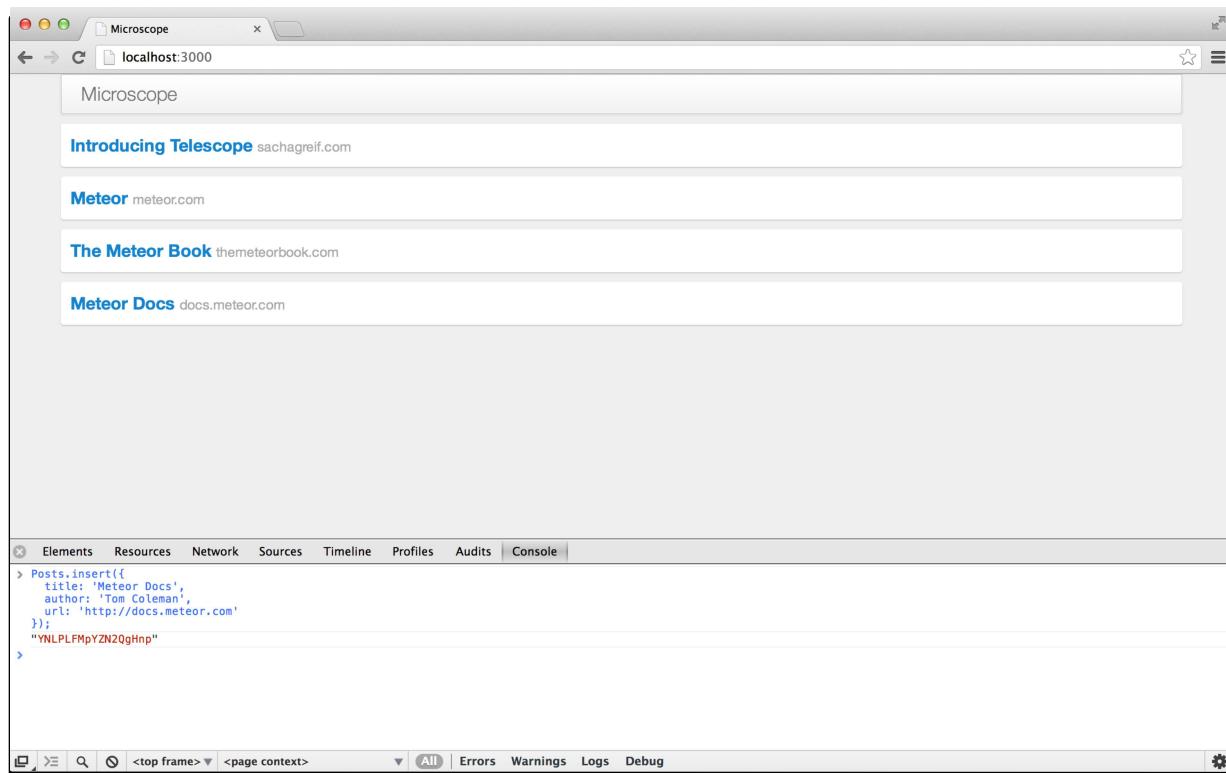
我们可以清晰地看到 `{{#each}}` 帮助方法已经枚举了 `Posts` 中的所有帖子，而且显示到屏幕上。服务器端的集合从 `Mongo` 数据库中取出贴子数据，通过网络传到客户端的集合中，进而 `handlers` 的帮助方法把这些数据加载到模板中。

现在我们只需要再走一步；让我们通过控制台增加另一个帖子：

```
Posts.insert({  
  title: 'Meteor Docs',  
  author: 'Tom Coleman',  
  url: 'http://docs.meteor.com'  
});
```

浏览器控制台

再看浏览器 - 你会看到这些：



通过控制台增加帖子

你刚才第一次看到从动功能生效了。当我们告诉 handlebars 去枚举 `Posts.find()` 游标的时候，它自己知道如何发现游标的变动，从而用最简单的方式将变化后的正确数据显示到屏幕上。

检查 DOM 变动

在目前的情况下，最简单的变动应该就是增加一个 `<div class="post">...</div>`。如果你想看看是否的确如此，你可以打开 DOM 检查器然后选择某个已经存在的帖子的 `<div>`。

现在在 Javascript 控制台，插入另外一个帖子。当你回到检查器，会发现一条新的 `<div>` 对应了新增的那个帖子。同时，原先选中的那个旧的 `<div>` 仍然存在。这是一种判断元素是否被重新渲染的有效方式。

连接集合: 发布与订阅

到此为止，我们仍然用着 `autopublish` 这个包，这个包并不是为正式产品化的应用程序准备的。正如它的名字陈述的那样，它简单地把整个集合分享给所有连接的客户端。这个可不是我们期望的样子，所以让我们去掉它。

打开一个终端窗口，输入：

```
meteor remove autopublish
```

这个操作有了立即的反应。当你打开浏览器，你会发现所有的帖子都不见了！这是因为我们一直依赖于 `autopublish` 来让我们的客户端可以镜像般地得到数据库中的所有帖子。

最终我们需要做得到我们仅仅把我们客户端需要看到的帖子传输过来（需要考虑分页的情况）。不过暂时我们可以先设置把 `Posts` 所有帖子都发布出来。

为达到这个目的，我们建立一个简单的 `Publish()` 函数，它仅仅返回一个反映所有帖子的游标。

```
Meteor.publish('posts', function() {
  return Posts.find();
});
```

server/publications.js

在客户端我们需要订阅这个发布。我们仅仅需要增加这样一行到 `main.js` 文件中：

```
Meteor.subscribe('posts');
```

client/main.js

提交 4-4

删除 `autopublish` 并建立基本的发布功能

[在 GitHub 中查看](#)

[查看演示](#)

如果你现在看一眼浏览器，发现帖子都回来了。哇！好险啊！

总结

我们都做了什么？尽管我们还没有用户界面，至少我们已经有了一个能用的应用。我们可以把这个应用部署到网络上，（使用浏览器的控制台）发帖子，并看到帖子显示在其他用户的浏览器上。

发布（Publication）和订阅（Subscription）是 Meteor 的最基本最重要的概念之一，但是如果你是刚刚开始接触 Meteor 的话，也是有些难度的。

这已经导致不少误解，比如认为 Meteor 是不安全的，或者说 Meteor 应用无法处理大量数据等等。

人们起初会感觉这些概念很迷惑很大程度上是因为 Meteor 像变魔法一样替你做了很多事儿。尽管这些魔法最终看起来很有效，但是它们掩盖了后台真正做的工作（好像魔术一样）。所以让我们剥去魔法的外衣来看看究竟发生了什么。

过去的日子

首先，让我们回顾一下2011年之前，当 Meteor 还没有诞生的时候的老日子。比如说我们要建立一个简单的 Rails app。当用户来我们的站点，客户端（举例说浏览器）向我们的服务器端的 app 发送请求。

App 的第一个任务就是搞清楚这个客户请求什么数据。这个可能是搜索结果的第12页、玛丽的用户信息、鲍勃的最新20条微博，等等等等。你可以想想成为一个书店的伙计在书架之间帮你寻找你要的书。

当正确的数据被找到，这个 App 的下一个任务就是把数据转换成好看的，人类可读的 HTML 格式（对于 API 而言是 JSON 串）。

用书店来举例，那就相当于是把你刚买的书包好，然后装入一个漂亮的袋子。这就是著名的 MVC（模型-视图-控制器）模式中的视图部分。

最终，App 把 HTML 代码送到客户端。这个 App 的任务也就交差了。它可以去买瓶啤酒然后等着下一个请求。

Meteor 的方式

让我们看看 Meteor 相对之下是多么的特别。正如我们看到的，Meteor 的关键性创新在于 Rails 程序只跑在服务器上，而一个 Meteor App 还包括在客户端（浏览器）上运行的客户端组件。

