英文翻译资料

A. 英文原文

The Future of Web Apps

At Airbnb, we've learned a lot over the past few years while building rich web experiences. We dove into the single-page app world in 2011 with our mobile web site, and have since launched Wish Lists and our newly-redesigned search page, among others. Each of these is a large JavaScript app, meaning that the bulk of the code runs in the browser in order to support a more modern, interactive experience.

This approach is commonplace today, and libraries like Backbone.js, Ember.js, and Angular.js have made it easier for developers to build these rich JavaScript apps. We have found, however, that these types of apps have some critical limitations. To explain why, let's first take a quick detour through the history of web apps.

JavaScript Grows Up

Since the dawn of the Web, the browsing experience has worked like this: a web browser would request a particular page (say, "http://www.geocities.com/"), causing a server somewhere on the Internet to generate an HTML page and send it back over the wire. This has worked well because browsers weren't very powerful and HTML pages represented documents that were mostly static and self-contained. JavaScript, created to allow web pages to be more dynamic, didn't enable much more than image slideshows and date picker widgets.

After years of advances in personal computing, creative technologists have pushed the web to its limits, and web browsers have evolved to keep up. Now, the Web has matured into a fully-featured application platform, and fast JavaScript runtimes and HTML5 standards have enabled developers to create the rich apps that before were only possible on native platforms.

The Single-Page App

It wasn't long before developers started to build out entire applications in the browser using JavaScript, taking advantage of these new capabilities. Apps like Gmail, the classic example of the single-page app, could respond immediately to user interactions, no longer needing to make a round-trip to the server just to render a new page.

Libraries like Backbone.js, Ember.js, and Angular.js are often referred to as client-side MVC (Model-View-Controller) or MVVM (Model-View-ViewModel) libraries.

The bulk of the application logic (views, templates, controllers, models, internationalization, etc.) lives in the client, and it talks to an API for data. The server could be written in any language, such as Ruby, Python, or Java, and it mostly handles serving up an initial barebones page of HTML. Once the JavaScript files are downloaded by the browser, they are evaluated and the client-side app is initialized, fetching data from the API and rendering the rest of the HTML page.

This is great for the user because once the app is initially loaded, it can support quick navigation between pages without refreshing the page, and if done right, can even

work offline.

This is great for the developer because the idealized single-page app has a clear separation of concerns between the client and the server, promoting a nice development workflow and preventing the need to share too much logic between the two, which are often written in different languages.

Trouble in Paradise

In practice, however, there are a few fatal flaws with this approach that prevent it from being right for many use cases.

SEO

An application that can only run in the client-side cannot serve HTML to crawlers, so it will have poor SEO by default. Web crawlers function by making a request to a web server and interpreting the result; but if the server returns a blank page, it's not of much value. There are workarounds, but not without jumping through some hoops.

Performance

By the same token, if the server doesn't render a full page of HTML but instead waits for client-side JavaScript to do so, users will experience a few critical seconds of blank page or loading spinner before seeing the content on the page. There are plenty of studies showing the drastic effect a slow site has on users, and thus revenue. Amazon claims that each 100ms reduction in page load time raises revenue by 1%. Twitter spent a year and 40 engineers rebuilding their site to render on the server instead of the client, claiming a 5x improvement in perceived loading time.

Maintainability

While the ideal case can lead to a nice, clean separation of concerns, inevitably some bits of application logic or view logic end up duplicated between client and server, often in different languages. Common examples are date and currency formatting, form validations, and routing logic. This makes maintenance a nightmare, especially for more complex apps.

Some developers, myself included, feel bitten by this approach-it's often only after having invested the time and effort to build a single-page app that it becomes clear what the drawbacks are.

A Hybrid Approach

At the end of the day, we really want a hybrid of the new and old approaches: we want to serve fully-formed HTML from the server for performance and SEO, but we want the speed and flexibility of client-side application logic.

To this end, we've been experimenting at Airbnb with "Isomorphic JavaScript" apps, which are JavaScript applications that can run both on the client-side and the server-side.

An isomorphic app might look like this, dubbed here "Client-server MVC".In this world, some of your application and view logic can be executed on both the server and the client. This opens up all sorts of doors-performance optimizations, better maintainability, SEO-by-default, and more stateful web apps.

With Node.js, a fast, stable server-side JavaScript runtime, we can now make this dream a reality. By creating the appropriate abstractions, we can write our application logic such that it runs on both the server and the client-the definition of isomorphic

JavaScript.

Isomorphic JavaScript in the Wild

This idea isn't new-Nodejitsu wrote a great description of isomorphic JavaScript architecture in 2011-but it's been slow to adopt. There have been a few isomorphic frameworks to spring up already.

Mojito was the first open-source isomorphic framework to get any press. It's an advanced, full-stack Node.js-based framework, but its dependence on YUI and Yahoo!-specific quirks haven't led to much popularity in the JavaScript community since they open sourced it in April 2012.

Meteor is probably the most well-known isomorphic project today. Meteor is built from the ground up to support real-time apps, and the team is building an entire ecosystem around its package manager and deployment tools. Like Mojito, it is a large, opinionated Node.js framework, however it's done a much better job engaging the JavaScript community, and its much-anticipated 1.0 release is just around the corner. Meteor is a project to keep tabs on-it's got an all-star team, and it's raised $11.2 M from Andreessen Horowitz-unheard of for a company wholly focused on releasing an open-source product.

Asana, the task management app founded by Facebook cofounder Dustin Moskovitz, has an interesting isomorphic story. Not hurting for funding, considering Moskovitz' status as youngest billionaire in the world, Asana spent years in R&D developing their closed-source Luna framework, one of the most advanced examples of isomorphic JavaScript around. Luna, originally built on v8cgi in the days before Node.js existed, allows a complete copy of the app to run on the server for every single user session. It runs a separate server process for each user, executing the same JavaScript application code on the server that is running in the client, enabling a whole class of advanced optimizations, such as robust offline support and snappy real-time updates.

We launched an isomorphic library of our own earlier this year. Called Rendr, it allows you to build a Backbone.js Handlebars.js single-page app that can also be fully rendered on the server-side. Rendr is a product of our experience rebuilding the Airbnb mobile web app to drastically improve pageload times, which is especially important for users on high-latency mobile connections. Rendr strives to be a library rather than a framework, so it solves fewer of the problems for you compared to Mojito or Meteor, but it is easy to modify and extend.

Published in Code
November 11, 2013 by Spike Brehm

B. 原文的翻译

网页应用程序的未来

在 Airbnb（租房网站），过去的几年里，我们学到了很多构建丰富 Web 经验。在 2011 年，我们通过自己的移动网站引入了单页面应用程序世界、已经启动了愿望清单和新近设计的搜索页面等等。在这些项目中，每个都是一个大型的 JavaScript 应用程序，这意味着大量的代码运行在浏览器中为了支持一个更现代，互动体验。

这种方法是在今天是很普遍的，像 Backbone.js、Ember.js 和 Angular.js 这些库文件，使开发人员更容易构建这些功能强大的 JavaScript 应用程序。然而，我们发现，这些类型的应用程序存在一些严重的局限性。究其原因，我们可以通过 Web 应用程序的历史先快速了解。

JavaScript 的发展

从网络发展之初到现在，浏览模式一直是这样的：一个 Web 浏览器请求一个特定的页面(例如“http://www.geocities.com/”)，引起互联网上的某个服务器生成一个 HTML 页面并将其通过网线发回。这个模式效果很好，因为浏览器不是非常强大和 HTML 页面代表文件大多是静态的，独立的。JavaScript，允许 Web 页面显示更为动态，不使用太多的图像幻灯片和日期选择小部件。

在个人电脑行业经过多年的进步，创新技术推动网络的达到它的极限，Web 浏览器一直进化努力跟上。现在，网络已经成为了一个全功能的应用程序平台，快速 JavaScript 运行时间和 HTML5 标准使得开发者能够开发出丰富的应用程序，而之此前只能在本地平台。

单页面应用程序

不久之前，开发人员开始通过在浏览器中使用 JavaScript 构建整个应用程序，利用这些新功能。应用程序例如 Gmail，单页面应用程序的典型例子，可以立即响应用户交互，不再需要到服务器的往返只是呈现一个新页面。

像 Backbone.js、Ember.js 和 Angular.js 这些库文件，经常被称为客户端 MVC(模型-视图-控制器)或 MVVM(Model-View-ViewModel)库。

大部分的应用程序逻辑(视图、模板、控制器、模型、国际化等等)应用于客户端，它通过 API 获得数据。服务器可以用任何语言编写，如 Ruby、Python、或 Java，它主要处理提供一个初始化的 HTML 页面。一旦 JavaScript 文件被浏览器下载，这些文件就会被解析和客户端应用程序被初始化，包括从 API 抓取数据和渲染 HTML 页面的其余部分。

这对于用户非常有利,因为一旦加载初始化应用程序时，它可以支持在没有刷新页面之间进行快速页面导航，如果处理得当，甚至可以离线工作。

这对于开发人员来说是伟大的，因为理想化的单页面应用程序有一个明确的客户端和服务器之间的关注点分离，通常用不同的语言编写来促进一个不错的开发流程和防止两者之间需要分享太多逻辑。

天堂里的烦恼

然而在实践中，这种方法有几个致命的缺陷，防止它适合许多用例。

搜索引擎优化

一个只能运行在客户端不能为 HTML 爬虫的应用程序，所以它有槽糕的默认搜索引擎优化。网络爬虫函数通过一个请求到一个 Web 服务器和解释结果；但如果服务器返回一个空白页，这是没有多少价值的。有解决方案，但不是没有

跳过一些桎梏。

性能

出于同样的原因，如果服务器不呈现完整页面的 HTML，而是等待客户端 JavaScript 去产生，用户将经历几个关键秒的空白页或微调器加载之前看到页面上的内容。有很多研究表明糟糕的影响，缓慢网站给用户使用，从而网站的收入也减少。亚马逊声称每个页面加载时间减少 100ms，收入就会增加 1%。Twitter 花了一年和 40 工程师重建他们的网站呈现在服务器上，而不是客户端，声称提高 5 倍加载时间。

可维护性

在理想的情况下会导致一个不错的，干净的关注点分离，不可避免的一些应用程序逻辑或视图的逻辑通常用不同的语言终结客户端和服务器之间的复制。常见的例子是日期和货币格式，形式验证和路由逻辑。这使得维护成为一个噩梦，尤其是对于更复杂的应用程序。

一些开发人员，包括我自己，觉得这种方法不好—它通常需要投入时间和精力建立一个单页面应用程序，很明显的缺点是什么。

一个混合的方法

在一天结束的时候，我们真的想要一个混合的新老方法：我们想提供十足的 HTML 服务器的性能和搜索引擎优化，但我们想要客提升客户端应用程序逻辑的速度和灵活性。

为此，在 Airbnb 我们用同构的 JavaScript 应用程序一直在努力，这是 JavaScript 应用程序可以运行在客户端和服务器端。

一个同构应用看起来像这样，这里被称为“客户端-服务器 MVC”。在这个世界上，你的一些应用程序和视图逻辑可以在服务器和客户端执行。这开辟了各种各样的门—性能优化，更好的可维护性，默认的引擎优化，更有状态 Web 应用程序。

伴随着 Node.js，一个快速，稳定的服务器端 JavaScript 运行时间的库文件，我们可以让这个梦想成为现实。通过构建合适的抽象，我们可以编写应用程序逻辑，它运行在服务器和客户端 JavaScript 同构的定义。

原生态的同构 JavaScript

这一想法并不新鲜—在 2011 年 Nodejitsu 对同构 JavaScript 架构进行了详细的描述，但它采用的缓慢，有一些同构框架已经涌现。

莫吉托是第一个开源同构框架被曝光。这是一个先进、完整的节点。基于 Node.js 框架，但其依赖 YUI 和 Yahoo!特别的怪癖从而导致其在自 2012 年 4 月开源的 JavaScript 社区不大流行。

流星可能是今天最知名的同构项目。流星是从头构建的支持实时应用程序，和其周围的团队建立一个整个生态系统包管理器和部署工具。和莫吉托一样，它是一个大型的、固执己见的 Node.js 框架，然而它在 JavaScript 社区做了更好的工作、和其备受期待的 1.0 版本是指日可待。流星是被密切关注的项目，它有一个全明星团队来开发，而且从 Andreessen Horowitz 筹集了 11.2 百万美元—公司完全集中在释放一个开源产品，在此前可谓闻所未闻。

Asana 任务管理应用程序由 Facebook 创始人 Dustin Moskovitz 建立，有一个非常有趣同构的故事。不伤害资金，考虑到莫斯科维茨作为世界上最年轻的亿万富翁，Asana 多年研发发展他们闭源 Luna 框架，最先进的同构 JavaScript 的例子之一。月神，原来数天建立了 v8cgi 在 Node.js 存在之前，允许一个完整的副本

在服务器上运行的应用程序的每个用户会话。它为每个用户运行一个单独的服务器进程，执行相同的 JavaScript 应用程序代码在服务器上运行的客户端，使整个类先进的优化，比如强大的离线支持和时髦的实时更新。

今年早些时候，我们推出了一个我们自己的同构库文件。称 Rendr，它允许您构建一个 Backbone.js、Handlebars.js 单页面应用程序，也可以完全呈现在服务器端。Rendr 是经验的产物由 Airbnb 重建的移动 Web 应用程序可以大大提高页面加载时间，在高延迟移动用户连接这是特别重要的。Rendr 趋向是一个库，而不是一个框架，因此相比莫吉托或流星，为你解决更少的问题，但很容易修改和扩展。

在 Code 于 2013 年 11 月 11 日出版

作者 Spike Brehm