基于websocket的实时单页应用开发框架

Introduction

Background

目前绝大部分web应用的都是基于HTTP协议通讯，及由客户端主动发起请求的方式进行数据的交互，这种方式在基本上能满足大部分网站的功能需求，是非常完善的web应用解决方案。随着网络的普及和快速发展，数据的变化变得越来越快，用户的数据的实时性要求也越来越高，在对实时性要求较高的web应用上，基于HTTP通讯的方式就不太实用，因为HTTP请求只能从客户端主动发出，服务端的有新的数据的时候，客户端并不能第一时间获取得到。为了解决这一问题，基于HTTP的解决方案主要有轮询、长轮询和Iframe流。这种三种解决方案缺点就是会浪费很多带宽资源和服务器资源。

At present, most of the web applications are based on the HTTP protocol and the data is exchanged by the client to initiate the request. This method can basically meet the functional requirements of most websites and is a very complete web application solution.With the popularization and rapid development of networks, data changes have become faster and faster. The real-time requirements of users' data have also become higher and higher. For web applications that require high real-time performance, HTTP-based communication is not practical. Because HTTP requests can only be initiated from the client, new data on the server side cannot be obtained by the client for the first time.To solve this problem, HTTP-based solutions mainly include polling, long polling, and Iframe streaming. The disadvantage of these three solutions is that it will waste a lot of internet resources and server resources.

HTML5标准中定义了新的通讯方式：WebSocket[1]，该通讯协议目的解决客户端和服务端之间的双向通讯问题，而且在主流浏览器中得到了广泛的支持。目前websocket在web上的应用主要有实时聊天、实时监控、游戏等。事实证明是一个非常可靠的技术。

The HTML5 standard defines a new communication method: WebSocket[1]. The purpose of this communication protocol is to solve the full duplex communication problem between the client and the server, and it has been widely supported in mainstream browsers. Currently websocket applications on the web mainly include real-time chat, real-time monitoring, and games. It turns out to be a very reliable technology.

单页应用是现在流行的页面开发模式，它可以在一个页面中完成原本多个页面才能完成的功能，而且性能更好，大大提升了用户体验。如果结合WebSocket技术，可以构建出的实时单页web应用，能够进一步提升效率，适应更多复杂的数据需求。

Single-page application is a popular page development mode, which can complete the functions of multiple pages in one page, and has better performance and greatly enhances the user experience. If combined with WebSocket technology, real-time single-page web applications can be built to further increase efficiency and adapt to more complex data needs.

Motivation

现在WebSocket的应用还不是非常广泛，一般在特定实时场景下才会在web应用中部分采用，相对HTTP请求，WebSocket模式有很大不同，在过于复杂的场景下，会增加开发难度，而且也缺少成熟的解决方案和参考资料。

Currently, the application of WebSocket is not very extensive, and it is generally adopted in web applications in certain real-time scenarios. Compared to HTTP requests, the WebSocket mode is very different. In a complicated scenario, it will increase the difficulty of development, and it is also lacking mature solutions and references.

如果系统在不增加太多开发复杂度的情况下全站采用websocket进行通讯，并结合单页应用的特点，用户的浏览体验会提高很多。在这种模式下，一个页面就完成所有功能，前后端只需要建立一个websocket链接就可以完成所有的数据交互。

If the system uses websocket for communication without increasing the development complexity, and combined with the characteristics of the single-page application, the user's browsing experience will be much improved. In this mode, all functions are completed on one page, and all data interactions can be completed only by creating a websocket link on the front and back ends.

所以我打算开发出一个单页应用+WebSocket通讯的web系统开发框架，让开发者可以快速开发出功能复杂的、高效的实时web应用，提供一个此类应用的解决方案和开发模式。

So I plan to develop a web system development framework for single-page application that base on WebSocket communication, providing a solution and development model for such applications, allowing developers to quickly develop functional and efficient real-time web applications.

Current Methods

基于HTTP的实时通讯方案

轮询

客户端在每隔一段较短的时间里发送一次新的HTTP请求，服务器无论是否有新的数据的时候都会立即返回结果。这种方式在服务端没有新的数据的时候会产生很多无效的请求，从而浪费带宽和服务器资源。

The client sends a new HTTP request every short period of time. The server will return the result immediately whenever there is new data. This method will generate many invalid requests when there is no new data on the server, thus wasting bandwidth and server resources.

长轮询

客户端发起的请求，服务端在没有新的数据下不会立即返回，而是处于挂载状态，直到有新的数据的时候再响应这次请求，或者挂载时间达到服务器时间限制时候，才会立即返回，告诉客户端继续发送下一个请求，如此循环。这种方案相对“轮询”方案可以减少HTTP请求次数，但是服务端在挂载请求的时候，依然需要消耗额外的资源，而且难以管理维护。

The server will not immediately return the client-initiated request without new data, but will be in the mounted state until there is new data and then respond to this request, or when the mount time reaches the server time limit, the server will return immediately, telling the client to continue sending the next request, and so on. This scheme can reduce the number of HTTP requests compared to the "polling" scheme, but the server still needs to consume extra resources when handling the mount request, and it is difficult to manage and maintain.

iframe流

该方案是在页面中插入一个隐藏的iframe，其src属性是一个长链接请求，服务端可以不断的传入数据，客户端通过JavaScript进行处理，从而获取到持续更新的数据。这种方案的缺点依然是需要花费服务端额外的资源去维护长连接。

The solution is to insert a hidden iframe in the page. The src attribute is a long link request. The server can continuously transfer data, and the client processes it through JavaScript to obtain continuously updated data. The disadvantage of this solution is still the need to spend extra resources on the server to maintain long connections.

多页面应用

多页面应用是将一个web应用根据功能划分成多个html页面，每个页面完成部分功能，用户需要在多个页面之间跳转完成工作。每次页面跳转需要重新请求html文件以及相关的css和JavaScript文件，然后进行渲染。这是目前最常见的web应用架构模式，其缺点是页面之间跳转会因为需要重新下载资源和重新渲染缺乏连贯性，影响用户浏览体验，而且会增加额外的HTTP请求和页面再次渲染的资源消耗，多个页面之间的数据也难以在浏览器端实现共享，需要借助服务的去维护，从而增加前后端开发的耦合度。优点是开发简单，利于搜索引擎检索。

A multi-page application divides a web application into multiple html pages according to functions. Each page completes part of the function. The user needs to jump between multiple pages to complete the work. Every time the page jumps, it needs to re-request the html file and related css and JavaScript files, and then render. This is the most common mode of web application architecture. The disadvantage is that the jump between pages will result in a lack of coherence due to the need to re-download resources and re-render, affecting the user browsing experience, and adding additional HTTP requests and resources for page rendering again. Consumption, data between multiple pages is also difficult to share on the browser side, need to use services to maintain, thereby increasing the coupling of the front-end development. The advantage is that the development is simple and conducive to search engine search.

Contributions

我所我打算实现一个基于websocket通讯的单页应用Web开发框架，方便开发者简单高效的开发出高性能的、实时的web应用[3]。我们框架的主要特点有：

1. 快速上手：不会引入太多复杂的概念，便于开发者快速从HTTP开发模式中过渡到新框架。
   . 简单：降低websocket的开发复杂度。
   . 双向通讯：前后端可以方便主动推送数据，而不需要关心websocket的工作原理。
   . 高效：数据交互速度更快，更加节约资源
   . 低耦合：能够方便的开发出适合更多场景的web应用，甚至是app应用。

In my opinion, I intend to implement a single-page application web development framework based on websocket communication, so that developers can easily and efficiently develop high-performance, real-time web applications. The main features of our framework are:

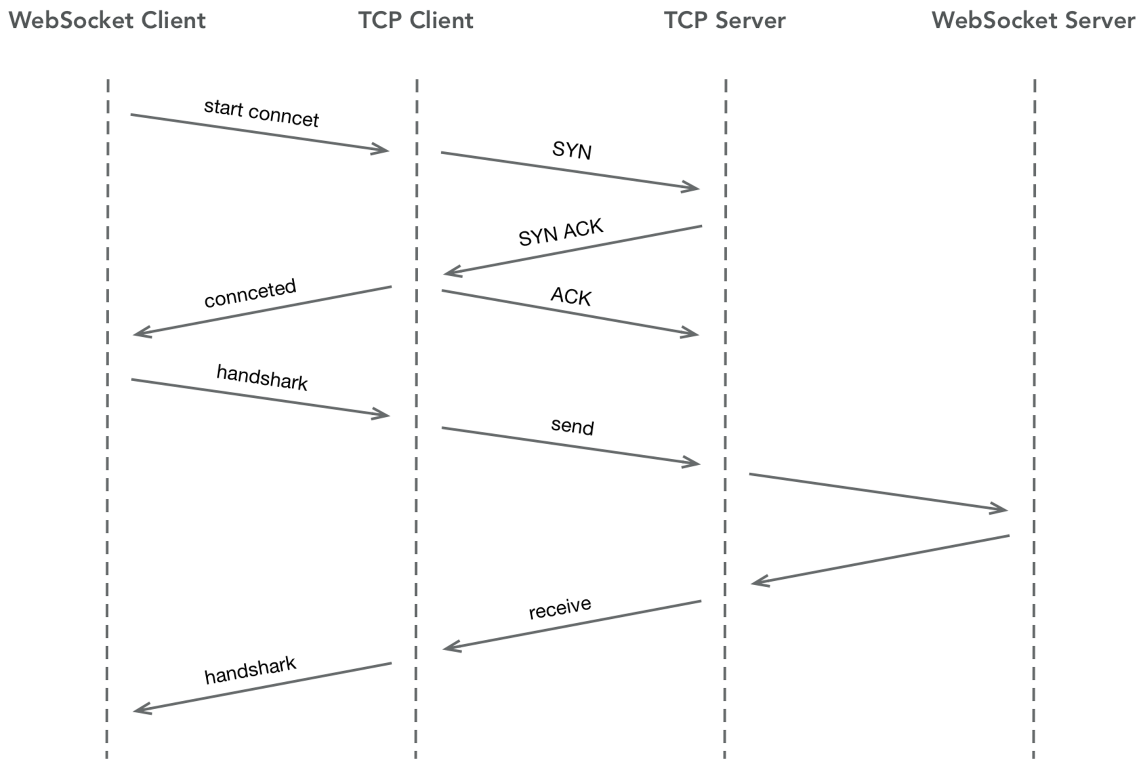
1. Get started quickly: There aren't too many complex concepts introduced, making it easy for developers to quickly transition from the HTTP development model to the new framework.
2. Simple: Reduce websocket development complexity.
3. Two-way communication: Front-end and back-end can facilitate the initiative to push data, without having to care about the working principle of websocket.
4. Efficient: faster data exchange, more resource-efficient
5. Low coupling: It is easy to develop web applications that are suitable for more scenarios, even app applications.

Related Work

WebSocket

绝大部分的web应用都是基于HTTP协议的，它简单可靠，经过多年发展，现在非常完善，但是由于其无连接的特点，所以每次只能处理一个请求，处理结束之后便断开，而且服务端无法主动发起请求，只能被动的接受请求。为了解决这个问题，于是IETF创建了WebSocket通讯协议，其主要特点是在单个TCP链接上进行全双工通讯，使得客户端和服务端都能够互相主动的推送数据[]。WebSocket建立链接的时候首先采用HTTP协议进行协商升级协议，后续数据传输再通过WebSocket协议去实现。建立WebSocket链接的过程如图所示。

Most of the web applications are based on the HTTP protocol. It is simple and reliable. After years of development, it is very complete. However, because of its connectionless nature, it can only handle one request at a time, and it is disconnected after the processing is over. The server cannot initiate the request, but can only passively accept the request. In order to solve this problem, the IETF created the WebSocket communication protocol. Its main feature is to perform full-duplex communication on a single TCP link, so that both the client and the server can push data to each other. When the WebSocket establishes a link, the protocol is first negotiated and upgraded using the HTTP protocol, and subsequent data transmission is implemented through the WebSocket protocol. The process of establishing a WebSocket link is shown in the Figure 1.



成功建立链接后客户端和服务端就可以随时进行双向数据通信，后续每次通信都不需要携带完整头部信息，直接传输数据文本，更加节省带宽资源，而且还支持拓展子协议。

After successfully establishing the link, the client and the server can perform two-way data communication at any time[5], and each subsequent communication can directly transmit the data text without carrying the complete header information, thereby further saving the bandwidth resources and supporting the development of the sub-protocol.

React

React是用于构建用户界面的开源JavaScript库[2]，诞生于FaceBook公司。React将数据和html封装成一个一个的组件，从而构成完整的页面，然后通过更改组件的state数据，使对应的html结构自动实现插入、删除、更改等操作，开发者不需要关注DOM的操作，React能够在O(n)的时间复杂度内准确的实现复杂的DOM更新，再结合浏览器的history api便能开发出功能复杂的web应用，用户也因此不需要频繁的切换页面便可完成复杂的业务需求。React的核心是虚拟DOM技术和Diff算法。

React is an open source JavaScript library for building user interfaces and was born at FaceBook[2]. React encapsulates data and html into components one by one, thus forming a complete page. Then, by changing the component's state data, the corresponding html structure is automatically inserted, deleted, changed, etc. The developer does not need to pay attention to the DOM operation[4]. React can accurately implement complex DOM updates within the time complexity of O(n). Combined with the browser's history api, React can develop complex web applications. Users can therefore complete complexities without frequent page switching. Business needs. The core of React is virtual DOM technology and Diff algorithm.

组件化

React将html代码封装成独立的UI组件，组件之间可以相互嵌套构成复杂的组件，最终构成整个页面，可以简单理解成强化版的html标签。如下代码是一个结合jsx语法的组件实现，该组件是继承React.Conponent的类，其render方法返回的是要渲染的html标签，这种类型的组件可以获取到父组件传递的数据，也可以维护自己的数据，同时拥有生命周期事件和对应的方法。组件化让UI更加灵活，易于维护。

React encapsulates html code into individual UI components that can be nested together to form complex components that ultimately form the entire page and can be easily understood as an enhanced html tag. The following code is a component implementation that combines jsx syntax. This component is a class that inherits from React.Conponent. The render method returns the html tag to be rendered. This type of component can obtain the data passed by the parent component and can also be maintained. Your own data, while having lifecycle events and corresponding methods. Componentization makes the UI more flexible and easier to maintain.

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}
```

虚拟DOM Virtual Document Object Model

React在数据更新的时候，根据更新后的状态用JavaScript构建DOM树，然后跟上次DOM树进行对比，找到两棵DOM树不同的地方，最后在浏览器真实DOM上对不同的地方进行更新，这样能够以最小改动去更新真实DOM结构，从而提高页面的性能。

React builds the DOM tree with JavaScript based on the updated state when the data is updated, then compares it with the last DOM tree, finds the two DOM trees different places, and finally updates the different places in the browser's real DOM. This can update the real DOM structure with minimal changes, thereby improving the performance of the page.

Diff算法 Diff Algorithm

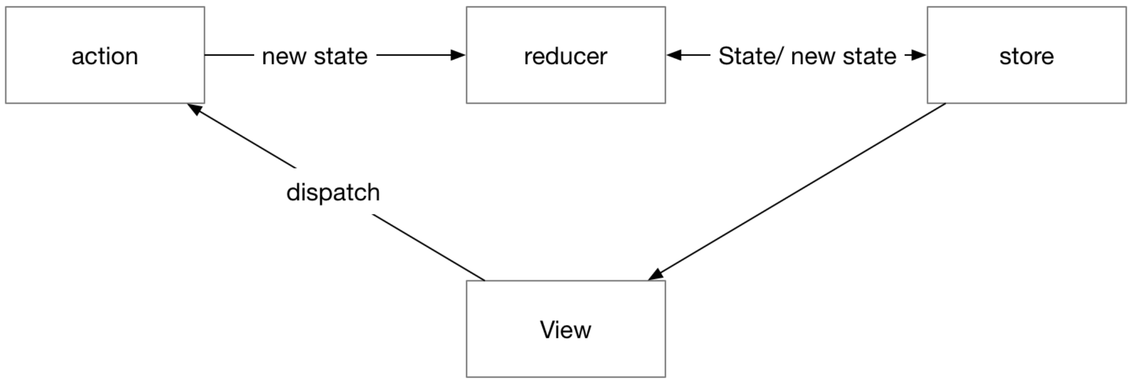
DOM树的比对虽然是在内存中进行，但是比对操作会在频繁的触发，所以依然需要保证其高效性。直接找出两棵树的不同处的时间复杂度是O(n^3)。React在比对过程中，从上到下逐层比较，同一层级的节点比较时，如果是同一类型的组件，继续进行分层比较，如果组件类型不同，直接替换整个节点及子节点。这样只需要遍历一次树，就可以完成DOM树的比对，将算法复杂度降低到O(n)。

Although the comparison of the DOM tree is performed in memory, the comparison operation is triggered frequently, so it is still necessary to ensure its high efficiency. The time complexity of directly finding the difference between two trees is O(n^3). React during the comparison process, from top to bottom layer by layer comparison, when the nodes of the same level are compared, if it is the same type of component, continue to hierarchical comparison, if the component types are different, directly replace the entire node and child nodes. This requires only traversing the tree once to complete the DOM tree alignment, reducing the algorithm complexity to O(n).

Redux

Redux是一个开源的应用状态管理JavaScript库，提供可预测化的状态管理，Redux的思想是将视图和数据进行分离，从而实现前端的MVC模式开发[?]，能够使程序更加直观、低耦合。Redux可以配合多种UI库使用，在配合React使用的时候，其前端结构图如图所示。

Redux is an open source application state management JavaScript library that provides predictable state management. Redux's idea is to separate view and data, so as to achieve front end MVC pattern development [?], which can make the program more intuitive and low coupling. Redux can be used with a variety of UI libraries. When used with React, the front-end structure diagram is as shown Figure 2.



我们的框架采用redux作为数据管理器，在任何时候都可以方便的对组件的状态进行同步，让数据可以预测和维护。

Our framework uses redux as a data manager. It is easy to synchronize the state of components at any time so that data can be predicted and maintained.

Our framework

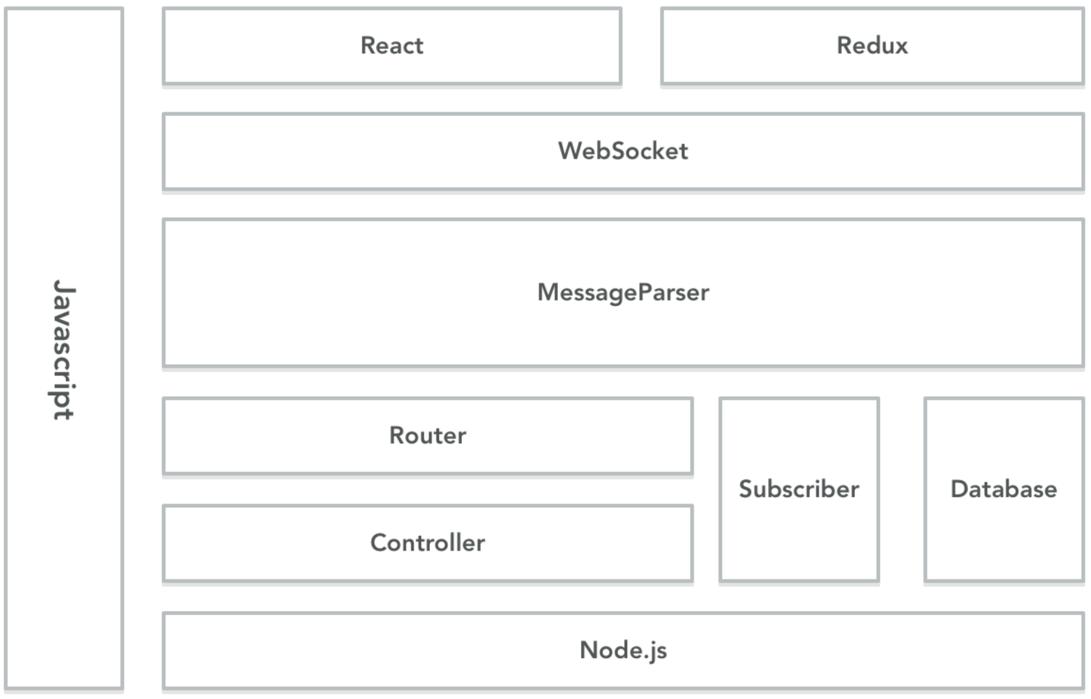
Architecture

本框架分为客户端部分和服务端部分。我们框架包含两种消息通讯机制，分别是立即消息和订阅消息，前者是用websocket实现类似HTTP请求的一对一消息，后者基本实现思想是发布和订阅模式，客户端自动通知服务端订阅内容，服务端在有数据更新的时候通知订阅的用户。

This framework is divided into a client part and a server part. Our framework contains two message communication mechanisms, namely, immediate messages and subscription messages. The former is a one-to-one message that uses websocket to implement HTTP-like requests. The basic idea of the latter is the publish and subscribe mode. The client automatically notifies the server of subscription content. The server informs the subscribed user when data is updated.

客户端是基于react和redux构建的单页应用，由携带数据的action触发store的更新，store更新触发视图的自动重新渲染，用户在视图上传发起action，从而形成一个闭环。我们框架在对react的组件进一步包装，使其在挂载和销毁的时候自动通过WebSocket通知服务端更新用户和redux的事件之间的订阅关系。同时服务端在WebSocket基础上实现了类似HTTP请求的立即消息机制，使得客户端可以发起能得到立即回复的请求，服务端有对应路由匹配和控制器处理请求，并可以处理服务的主动发布的逻辑，即根据订阅器里的用户订阅事件进行消息推送，消息数据内容为redux action 对象，被订阅的组件能够准确的收到消息并自动触发事件，从而更新store，最终自动更新视图。系统架构下图所示。

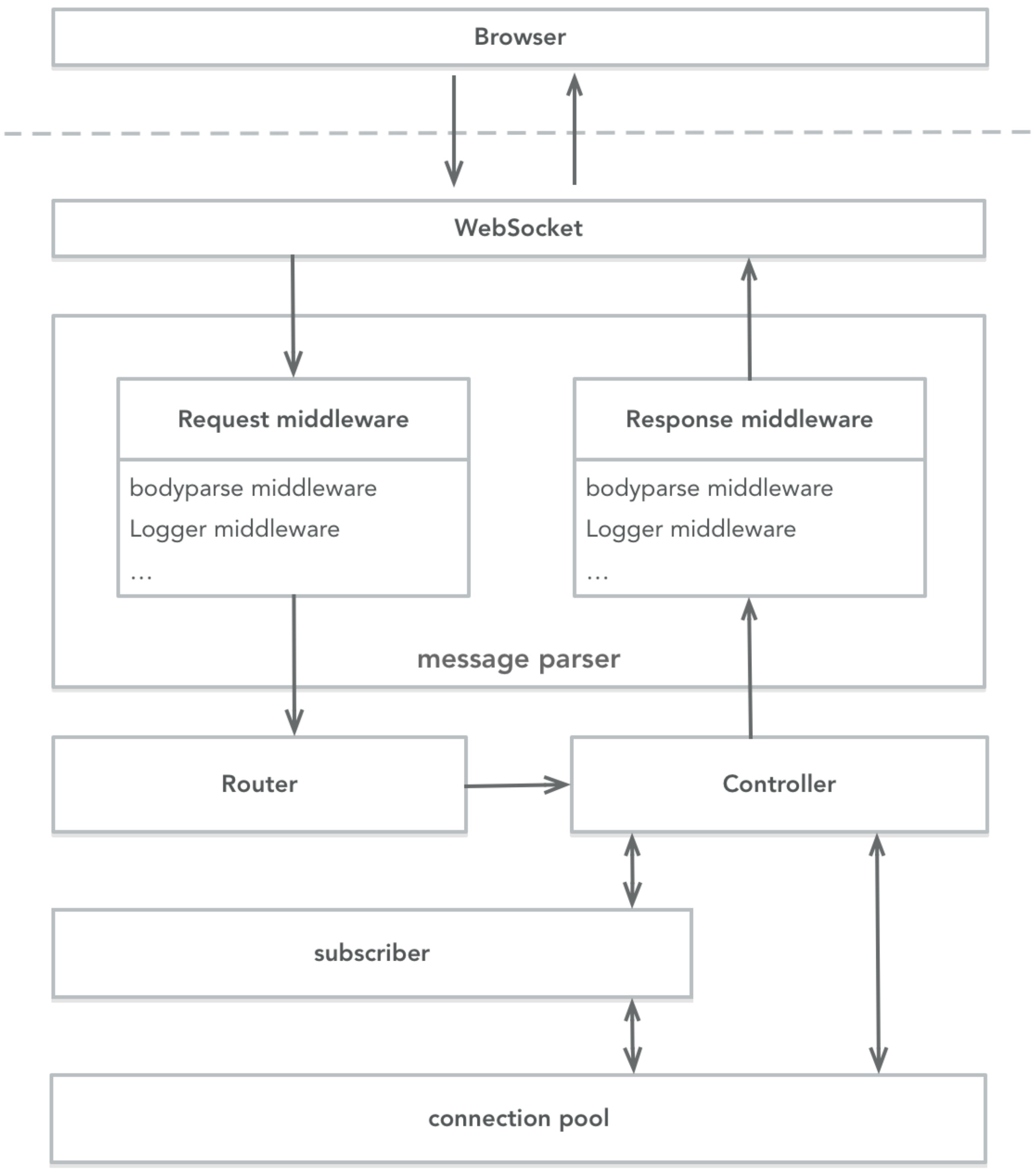
The client is a single-page application constructed based on react and redux. The action that carries the data triggers the update of the store. The store update triggers the automatic re-rendering of the view. The user initiates the action in the view upload, thus forming a closed loop. Our framework further packages the components of react so that when it is mounted and destroyed, it automatically informs the server through WebSocket to update the subscription relationship between the user and redux events. At the same time, the server implements an immediate message mechanism similar to an HTTP request on the basis of WebSocket, so that the client can initiate a request for immediate reply, the server has corresponding route matching and the controller processes the request, and can handle the active publishing logic of the service. That is, the message is pushed according to the user subscription event in the subscriber. The content of the message data is a redux action object, and the subscribed component can receive the message accurately and trigger the event automatically, thereby updating the store, and finally automatically updating the view. The system architecture is shown in the Figure 3.



服务端架构 Server Architecture

服务端主要由消息解析器、路由、控制器、订阅器和连接池组成。WebSocket接受到消息后会经过消息解析器中的请求中间件层层处理，然后构造成一个请求对象，交给相匹配的路由处理，路由会调用相关的控制器对当前请求进行业务逻辑处理，例如进行数据库操作等，处理完后，控制器可以根据订阅器中的用户订阅情况和连接池中的活跃用户构造返回内容，返回的内容依然会经过消息解析器中的返回中间件层层处理，最后通过WebSocket发送给用户。架构图所示。

The server consists of message parsers, routes, controllers, subscribers, and connection pools. After receiving the message, WebSocket will process the request middleware in the message parser, and then construct a request object and send it to the matching route. The route will call the related controller to process the current request for business logic, for example, Database operations, etc. After processing, the controller can return content based on the user's subscription status in the subscriber and active user constructs in the connection pool. The returned content will still be processed through the middle layer returned by the message parser, and finally Send it to users via WebSocket. The architecture diagram shows in Figure 4.



我们框架中提到的路由不是传统框架中的路由，但是实现的是相同的功能，路由的作用是对HTTP请求的路径进行匹配，然后进行响应，事实证明这种方式能够有效的给每一个HTTP请求添加唯一标识。我们框架也借鉴这种方式，对客户端的每一条消息添加一个头部对象，该对象就包含url属性，服务端根据url建立路由机制，使框架能够处理多样的websocket消息。

The route mentioned in our framework is not a route in the traditional framework, but the same function is implemented. The role of the route is to match the path of the HTTP request and then respond. It is proved that this method can effectively give each HTTP Request to add a unique identifier. Our framework also learns from this approach by adding a header object to each message on the client. The object contains the url attribute. The server creates a routing mechanism based on the url so that the framework can handle a variety of websocket messages.

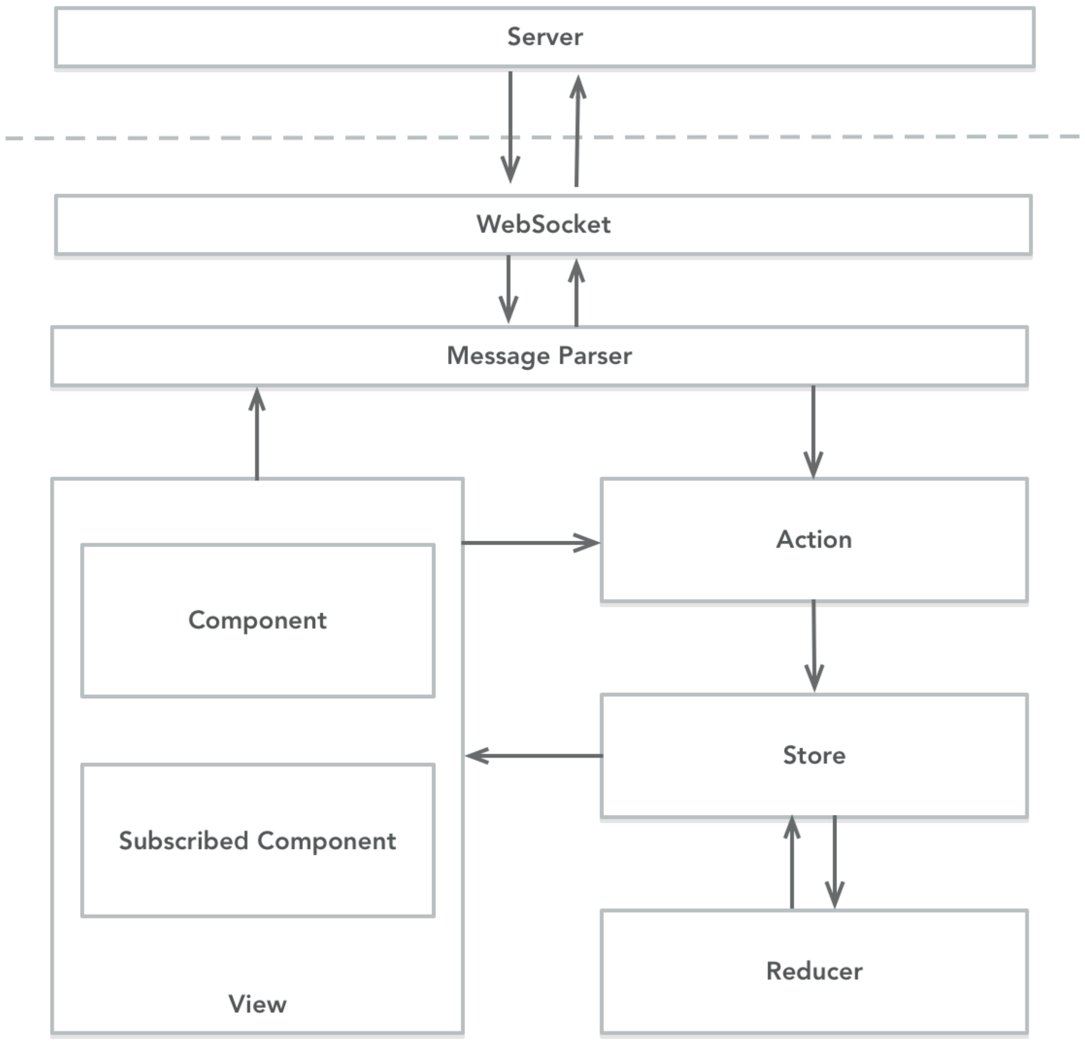
结合中间件和路由就可以完成多种业务需求，为了实现自动推送新消息功能，我们用订阅器实现用户以及其订阅关系，框架内置一个值为/subsribe的路由，该路由接受body为redux事件名的请求，并添加当前用户与订阅事件到订阅器中，与之对应的/unsubscribe路由则是删除订阅器中当前用户对该事件的订阅。开发者可以创建任何需要的路由以及控制器实现需要的功能，例如登录、获取文章等需要立即返回的数据请求，客户端会通过请求的唯一标示来保证同一次请求与返回。

Combining middleware and routing can accomplish various business requirements. In order to implement the function of automatically pushing new messages, we use a subscriber to implement the user and its subscription relationship. The framework has a route with a value of /subsribe, and the route accepts the body as the redux event name. The request is added, and the current user and the subscription event are added to the subscriber, and the corresponding/unsubscribe route is to delete the current subscriber's subscription to the event in the subscriber. Developers can create any needed routes and controllers to implement the required functions, such as logging in, getting articles, and other data requests that need to be returned immediately. The client will ensure the same request and return through the unique identifier of the request.

客户端架构 Client Architecture

客户端主要包括三个部分，视图、状态管理器和WebSocket消息处理器。我们框架的前端部分分两个子框架去实现，分别是jayce和jayce-dom，这样能够将视图层与数据处理层分离，以实现与多种视图框架配合使用，例如vue和angular等。首先我们框架通过传入redux的store对象和配置信息生成一个全局唯一的Jayce实例，该实例包含WebSocket执行方法和redux事件执行方法，实例化Jayce实例后，就会建立WebSocket链接，对于需要服务端主动推送新数据的内容，开发者在编写react组件的时候，可以调用jayce-dom的jayceSubscribe方法将任意组件包装成订阅组件，该组件在生命周期内会通知服务端当前用户订阅redux事件，在组件销毁的时候也请求服务端当前用户取消redux事件的订阅。对于立即消息，用户可以调用Jayce实例的send方法发送请求到服务端，其回调方法里会得到服务端返回的数据，编写上跟AJAX请求无异。前端架构如图所示。

The client consists of three parts, the view, the state manager, and the WebSocket message processor. The front end of our framework is implemented in two sub-frameworks, namely jayce and jayce-dom, which can separate the view layer from the data processing layer to enable the use of multiple view frames, such as vue and angular. First, our framework generates a globally unique Jayce instance by passing in redux's store object and configuration information. This instance contains the WebSocket execution method and the redux event execution method. After the Jayce instance is instantiated, a WebSocket link will be established for the server-needed initiative. Pushing the contents of new data, the developer can write jayce-dom's jayceSubscribe method to wrap any component into a subscription component when writing the react component. This component will notify the server's current user to subscribe to the redux event during the life cycle, and destroy the component. Also requests the current user of the server to cancel the redux event subscription. For the immediate message, the user can call the send method of the Jayce instance to send the request to the server. The callback method will get the data returned by the server. The write is the same as the AJAX request. The front-end architecture is as shown in the Figure 5.



消息协议 Message Protocol

除了浏览器初次渲染需要的文件通过HTTP请求外，我们让后续的所有的数据交互都通过WebSocket实现，而不同的类型的消息会有不同的处理逻辑，为了保证不同类型的消息能够被正确处理以及系统的可拓展性，我们在WebSocket的消息基础上建立了一个简单的协议。WebSocket传输的数据内容是字符串文本，在进行消息传递前，都需要经过message parser进行解析。发送端封构造json格式的请求对象，然后转换成json字符串交给WebSocket传输，接收端再解析成json对象，由于客户端和服务端都是基于JavaScript开发，所以json能够直接被处理。每个消息对象包含两个属性，header 和 body。body为数据内容，header有两个固有属性url和type，前者是路由标识，告诉服务端用哪个路由处理器处理，后者是请求类型，框架内置三种类型：IMMEDIATELY、SUBSCRIBE、UNSUBSCRIBE，分别是立即型消息、订阅型消息、取消订阅型消息。除此外，应用也可以根据需要添加其他header信息，从而实现更灵活的业务。

In addition to the files required for the initial rendering of the browser through the HTTP request, we allow all subsequent data interactions to be implemented through WebSocket, and different types of messages will have different processing logic, in order to ensure that different types of messages can be processed correctly and With the scalability of the system, we have established a simple protocol based on the

WebSocket message. WebSocket transmission of data content is a string of text, in the message before passing, need to be parsed by the message parser. The sender constructs the request object in json format, and then converts it into a json string for transmission to the WebSocket. The receiver then parses it into a json object. Since both the client and the server are based on JavaScript, json can be processed directly. Each message object contains two attributes, header and body. The body is the data content. The header has two intrinsic properties: url and type. The former is the route identifier, which tells the server which route processor to use for processing. The latter is the request type. The frame has three built-in types: IMMEDIATELY, SUBSCRIBE, and UNSUBSCRIBE. Immediate messages, subscription messages, and unsubscribe messages. In addition, applications can also add other header information as needed to achieve more flexible services.

自动订阅组件 Automatic Subscription Component

React通过编写组件的方式去构建浏览器DOM结构，每个组件都有自己的生命周期方法，我们框架在React组件的基础上进行了封装，封装通过`jayceSubscribe()`方法实现。该方法接受两个参数，第一个参数是需要订阅的redux事件数组，第二参数是实例化的Jayce对象，调用`jayceSubscribe()`方法后返回的是一个匿名方法，接受一个react 组件为参数，返回包装后的Jayce 组件。调用示例：

React builds the browser DOM structure by writing components. Each component has its own lifecycle method. Our framework encapsulates the React component and the package is implemented using the `jayceSubscribe()` method. This method accepts two arguments. The first argument is an array of redux events to be subscribed to. The second argument is an instantiated Jayce object. Calling the `jayceSubscribe()` method returns an anonymous method accepting a react component as an argument. , Return the packaged Jayce component. Invoking the example:

```
export default jayceSubscribe(['GET_NEW_ARTICLE'], jayce)(Article);
```

其中Article为需要订阅的组件，最终导出的是经过Jayce包装的组件。 Jayce包装组件的过程：

1. 创建一个新组件
2. 在componentWillMount的生命周期事件里执行请求订阅方法
3. 在componentWillUnmount的生命周期事件里执行取消订阅请求方法
4. 添加需要被包装的组件作为子组件
5. 返回这个新组件

Which Article is the need to subscribe to the components, the final export is Jayce packaged components. The Jayce packaging component process:

1. Create a new component
2. Perform a request subscription method in the lifecycle event of componentWillMount
3. Perform unsubscribe request method in lifecycle event of componentWillUnmount
4. Add components that need to be packaged as subcomponents
5. Return to this new component

订阅器 Subscriber

为了实现服务端能够准确、主动的推送实时性数据，所以服务端需要维护一个用户的store订阅器。订阅器保存的是客户端的redux事件与用户的对用关系，数据结构为一个JavaScript对象，属性名为事件名称，值为订阅该事件的用户连接对象数组，框架内置的/subsribe和/unsubscribe路由及相关控制器实现了对订阅器的自动管理，对开发者而言无需关心订阅器内容，只需根据业务订阅和发布事件，对应的用户都能够自动获取到数据和触发redux事

件。

In order to enable the server to accurately and proactively push real-time data, the server needs to maintain a user's store subscriber. The subscriber saves the client's redux event and the user's use relationship, the data structure is a JavaScript object, the property name is the event name, and the value is the user connection object array that subscribes to the event, the built-in /subsribe and/unsubscribe routes of the framework and The related controller implements the automatic management of the subscriber, and does not need to care about the subscriber content for the developer. According to the business subscription and the publishing event, the corresponding user can automatically obtain the data and trigger the redux event.

Message Parse

Message Parser是服务端和客户端可靠通信的枢纽，WebSocket只是建立服务端和客户端之间的全双工通信的渠道，所以我们在需要一个方便，可拓展的消息解析功能，使这些文本消息能够被框架正确识别和处理。通过上文约定的消息协议，在服务端，Message Parser会在服务启动的时候注册系统级和用户级中间件，接收到消息的时候Message Parser会根据消息内容构造json格式的请求对象，然后交个每个'request'型的中间件处理，最后到达路由处理器，服务端返回消息到客户端的时候，返回的消息对象依然会经过消息处理器中的'response'中间件处理，最后处理成为符合消息协议规定的文本内容交给WebSocket发送给客户端。

Message Parser is the hub of reliable communication between server and client. WebSocket is only a channel for establishing full-duplex communication between server and client. Therefore, we need a convenient and expandable message parsing function to enable these text messages to It is correctly identified and processed by the framework. Through the message protocol agreed above, on the server, Message Parser will register system-level and user-level middleware when the service is started. When receiving the message, Message Parser will construct the request object in json format according to the message content, and then submit a message. Each 'request' type of middleware is processed and finally reaches the route processor. When the server returns a message to the client, the returned message object will still be processed by the 'response' middleware in the message handler, and finally processed into a match message. The text content specified in the agreement is sent to the client by WebSocket.

Middleware

中间件是框架的一个重要组成部分。每一个消息都会流经每一个中间件，这样创建合理的中间件能够完成各种业务需求，例如身份认证、统计分析、消息拦截等。一个中间就是一个方法，接受两个参数，第一个请求对象，第二个是执行下一个中间件的方法，当中间件被注册到框架后，该中间件方法接受到的分别是上一个中间件处理完后的请求对象和调用下一个中间件的方法。如下是框架内置的构建请求对象的中间件示例，它是请求阶段第一个中间件。

Middleware is an important part of the framework. Every message flows through every middleware, so creating reasonable middleware can fulfill various business requirements, such as identity authentication, statistical analysis, message interception, and so on. A middle is a method that accepts two parameters. The first is the request object. The second is the method to execute the next middleware. After the middleware is registered in the framework, the middleware method receives the previous one. The middleware after processing the request object and the method to call the next middleware. The following is an example of a middleware for a build request object built into the framework. It is the first middleware in the request phase.

```
function requestBodyParse(ctx, next) {
  let req = JSON.parse(ctx.message);
  if(req.header && req.body){
    ctx.req = req;
    next();
  } else {
    ctx.req = {
      header: {
        url: '/error'
      },
      body: ''
    }
    return;
  }
}

module.exports = requestBodyParse
```

该中间件将接收到的消息字符根据消息协议规定格式串转换成对象，处理完后调用next()方法，执行下一个中间件。如果转换失败，则交给框架内置的/error路由处理。然后在框架实例对象上调用use方法注册到消息解析器中。

The middleware converts the received message character into an object string according to the format specified by the message protocol. After processing, the next() method is called to execute the next middleware. If the conversion fails, it is passed to the framework's built-in /error routing process. Then call the use method on the framework instance object to register with the message parser.

Process

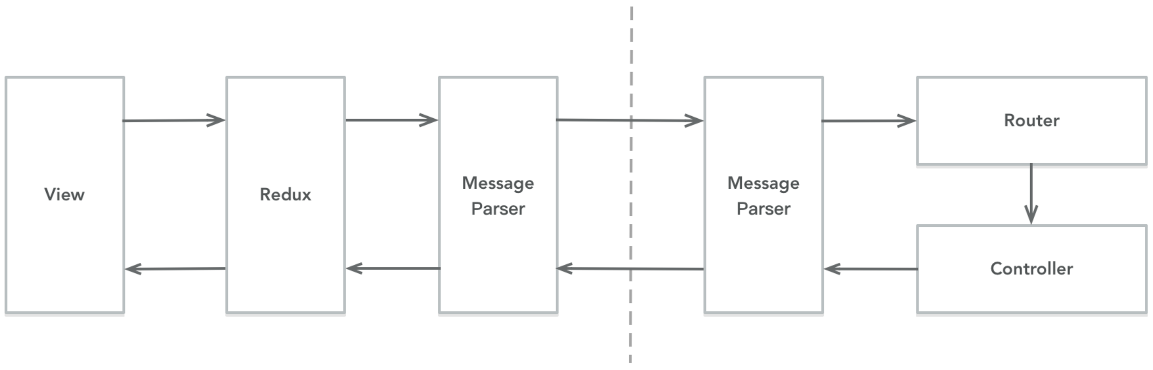
在实际使用中，主要存在三个服务端与客户端的交互方式，分别是客户端发送后立即得到返回的消息、客户端发送订阅redux和取消订阅事件的消息、服务端发送执行redux事件的消息。

In practical use, there are mainly three interaction modes between the server and the client. They are the message returned immediately after sending by the client, the message sent by the client to subscribe to the redux and unsubscribing event, and the message sent by the server to execute the redux event.

立即消息 Immediate message

一个web应用中会有大部分操作是立即返回操作，需要马上得到结果，例如提交表单、打开文章详情页面等。这种消息需要保证请求与返回一一对应，即每一次客户端发送消息都会有服务端唯一应答，跟HTTP请求的特点一样。为了实现这一特性，客户端每次发出消息都会在请求对象头部添加唯一标识，并将其添加到请求等待队列，服务端处理请求后返回携带这个标识的消息对象，客户端得到返回后会去请求对待队列中查找并执行相关回调方法。流程简化后如图所示。

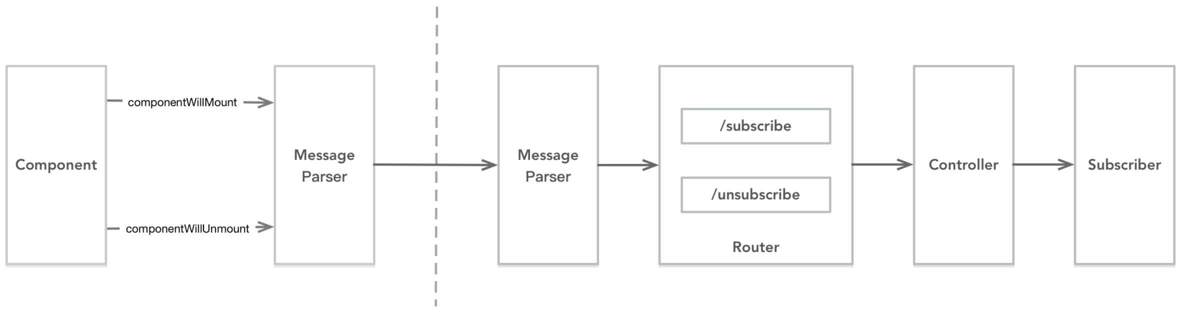
Most of the operations in a web application are immediate return operations, ie, immediate results, such as submitting a form, opening an article details page, etc. This kind of message needs to ensure that the request and return are in one-to-one correspondence. That is, each time the client sends a message, it will have a unique reply from the server, which is the same as the characteristics of the HTTP request. In order to achieve this feature, each time the client issues a message, it adds a unique identifier to the request object's header and adds it to the request waiting queue. After the server processes the request, it returns the message object carrying the identifier. After the client is returned, the client will be returned. Go to the request queue to find and execute related callback methods. The process is simplified as shown in Figure 6.



自动订阅 Automatic subscription

自动订阅消息是组件自动完成的，组件会在相关的生命周期方法里发送消息通知服务端订阅和取消订阅redux事件，然后服务端在订阅器中更新该用户对redux事件的订阅关系。

The automatic subscription message is automatically completed by the component. The component sends a message in the relevant life cycle method to notify the server to subscribe and unsubscribe the redux event, and then the service side updates the subscriber's subscription relationship to the redux event in the subscriber.The subscription process is shown in Figure 7.



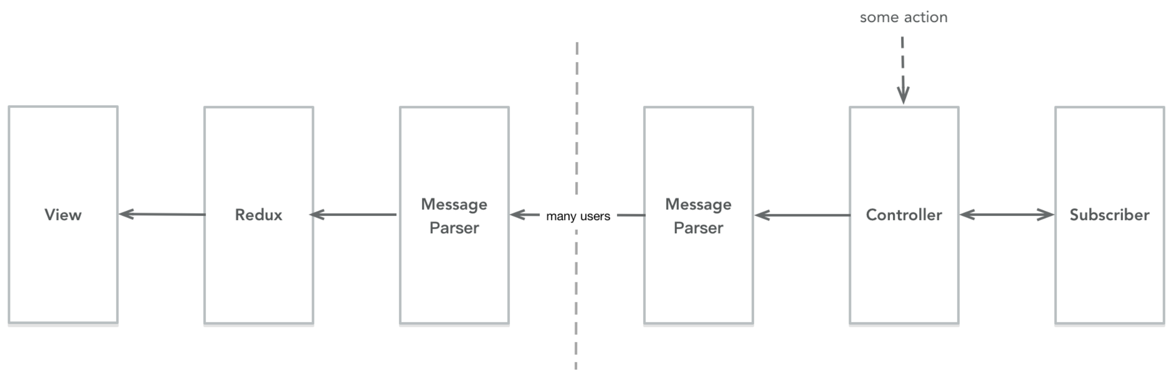
订阅流程的实现跟立即消息原理相似，框架封装了发送订阅相关的方法和路由控制器，订阅相关的路由控制器接收到订阅请求后不需要返回消息，直接去更新订阅器。用户也可以重写订阅相关的路由控制器，或者添加额外的订阅器路由实现更灵活的功能。

The implementation of the subscription process is similar to the principle of the immediate message. The framework encapsulates the method of sending subscriptions and the routing controller. The subscription-related routing controller does not need to return a message after receiving the subscription request and directly updates the subscriber. Users can also override the subscription-related routing controllers or add additional subscriber routes for more flexible functionality.

服务端推送 Server push

触发服务端主动推送操作可以来自多种行为，例如其他用户提交数据操作、服务端定时任务等。其流程如图所示。

Triggering the server's active push operation can come from a variety of behaviors, such as other users submitting data operations, server-side timing tasks, and so on. The process is shown in the Figure 8.



只要能获取到框架实例对象，就可以随时在需要的逻辑中调用该实例触发广播、多播、单播操作。 As long as you can get the framework instance object, you can call the instance at any time in the required logic to trigger broadcast, multicast, and unicast operations.

Experimental results

相对于HTTP模式的web应用，我们框架的主要优势在实现了客户端和服务端的全双工通讯和更小的数据传输量上。因此我们通过两个实验对比HTTP服务器和我们框架服务器的性能[6]。以下实验的服务器配置都是1核心处理器、2G运行内存。客户端为 i7 4460。

Compared to HTTP mode web applications, the main advantage of our framework is to achieve full-duplex communication and smaller data transfer between the client and server. So we compare the performance of the HTTP server and our framework server through two experiments. The server is hosted on aliyun single core processor and 1G RAM, the client is run on Intel Core i7-8550U and 8GB of RAM in the chrome browser.

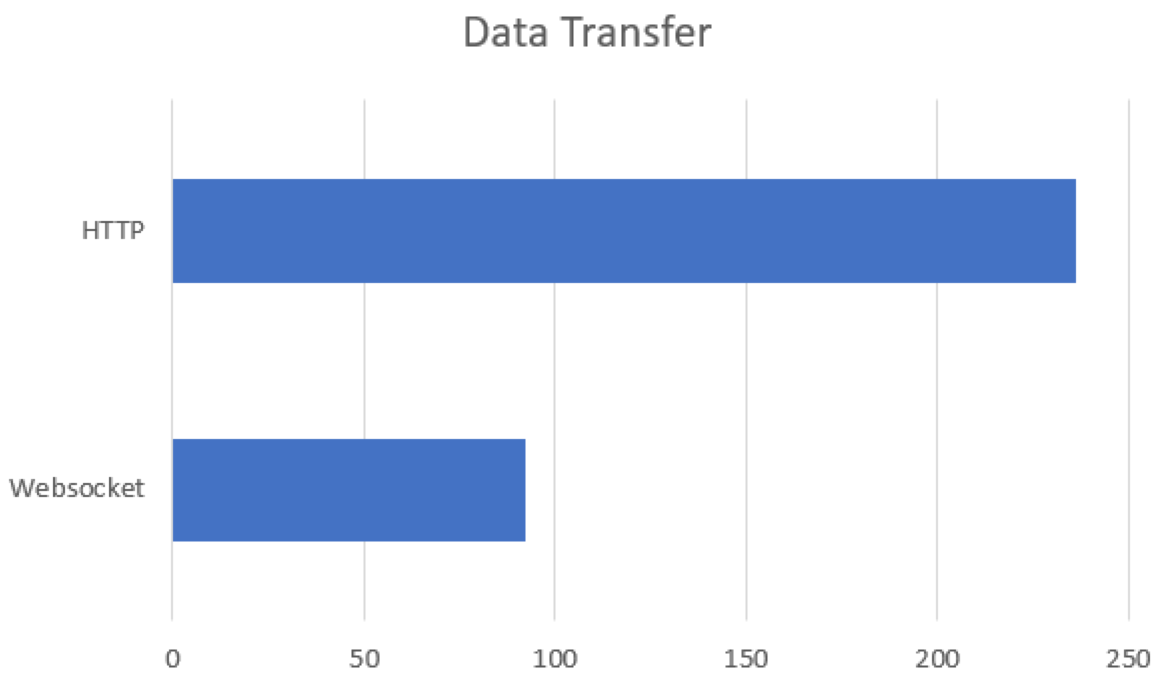
Express 是一个轻量级的基于Node.js的HTTP协议服务端框架，我们分别用我们的框架和express框架搭建两个服务端应用，客户端分别通过HTTP的方式和Webscoket的方式向服务端获取相同的json数据文本。

Express is a lightweight Node.js-based HTTP protocol server framework. We use our framework and express framework to build two server applications. The client requests the same json data text from the server through HTTP and Webscoket respectively.

相同请求内容下数据传输量对比

首先我们对比一下两种方式的下的数据传输量，websocket建立链接后每次请求只会传递消息内容，而不需要传递HTTP那样的完整头部信息，可以预想前者的数据传输量比后者小。图是两者的对比。

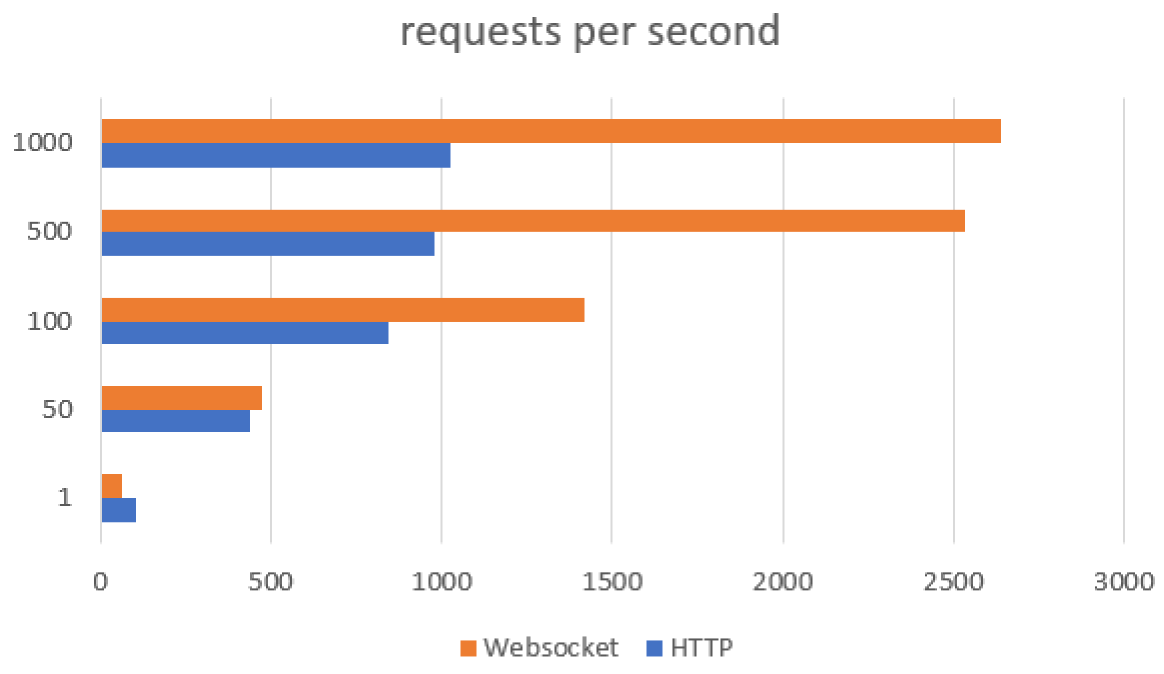
First, we compare the amount of data transmission under the two modes. After websocket establishes a link, each request only passes the message content. Instead of passing the complete header information such as HTTP, the amount of data transmission of the former is expected to be smaller than the latter.The Figure 9 is a comparison of both.



相同数量用户并发下服务端响应性能

然后我们对比两种模式的并发能力。我们模拟100个用户发起请不同数量请求下服务端每秒响应的请求数，图1是每个用户发起1个、50个、100个请求时测试的结果。其中websocket测试结果包含websocket建立阶段。

Then we compare the concurrent capabilities of the two modes. We simulated the number of requests that the 100 users responded to each request for different numbers of requests. Figure 10 shows the results of each user's test when they initiated 1, 50, and 100 requests. The websocket test result includes the websocket creation phase.



每个用户只发起一个请求时，HTTP比websocket的响应速度快一倍，由于websocet建立后并不会马上释放，服务端需要消耗资源去维护这些链接。但是随着每个用户的请求数的增加，websocket的优势就体现出来了，频繁的请求让websocket能够充分利用建立起来的链接，以最小的开销去传递数据。当每个用户的请求数达到500时，两者的服务器资源已经被彻底消耗，所以请求处理数量无法进一步提升。

When each user only initiates one request, HTTP responds twice as fast as websocket. Since websocet is not released immediately after it is established, the server needs to consume resources to maintain these links. However, as the number of requests from each user increases, the advantages of websocket are reflected. Frequent requests allow websocket to make full use of the established links and transfer data with minimal overhead. When the number of requests per user reaches 500, the server resources of both are completely consumed, so the number of request processing cannot be further improved.

Conclusions

这篇文章介绍了一个基于WebSocket的单页应用开发框架，在实时性要求越来越高和单页应用成为主流开发方式的环境下，我们框架为开发者提供了一个满足这两种需求的解决方案和实现，能够让开发者快速高效的开发出实时单页web应用，大大提高用户使用体验，而且对于HTTP这种数据获取方式，我们框架也同样支持，从而满足多种功能需求。实验证明，我们框架能有效降低带宽、服务器等资源消耗，从而降低运维成本。为了适应快速的技术的变化，我们框架尽可能降低各模块的耦合度，例如开发者可以重写组件包装器来搭配各种视图框架，也可以将订阅器用redis数据库替代。

This article introduces a single-page application development framework based on WebSocket. In an environment where real-time requirements are becoming higher and higher and single-page applications have become mainstream development methods, our framework provides developers with a solution that satisfies both requirements. The solution and implementation can enable developers to quickly and efficiently develop real-time single-page web applications, greatly improving the user experience, and for the HTTP data acquisition method, our framework also supports it to meet a variety of functional requirements. Experiments have proved that our framework can effectively reduce the bandwidth, server and other resource consumption, thereby reducing the operation and maintenance costs. In order to adapt to rapid technological changes, our framework minimizes the coupling of modules. For example, developers can rewrite component wrappers to match various view frameworks, and can also replace subscribers with redis database.

我们框架用websocket替代了绝大部分HTTP请求，虽然底层变化较大，但是对开发者而言跟传统web应用开发模式变化不大，我们框架保持了路由、控制器、请求、返回、连接池等相关概念和功能，学习简单但是功能强大。

Our framework replaces most of the HTTP requests with websockets. Although the underlying changes are relatively large, there is little change for developers from traditional web application development models. Our framework maintains routes, controllers, requests, returns, connection pools, etc. Related concepts and functions, learning is simple but powerful.

参考文献

[1]February. The WebSocket protocol[J]. 2011. [2]Speaker-Hunt P, Speaker-O'Shannessy P, Speaker-Smith D, et al. React: Facebook's Functional Turn on Writing JavaScript[J]. Queue, 2016, 14(4):40. [3]Pimentel V, Nickerson B G. Communicating and Displaying Real-Time Data with WebSocket[J]. IEEE Internet Computing, 2012, 16(4):45-53. [4]Yan X Q, Bai J F. React Refresh Mechanism Analysis Based on Virtual Dom Diff Algorithm[J]. Computer Knowledge & Technology, 2017. [5]Rao S S, Vin H M, Tarafdar A. Comparative Evaluation of Server-push and Client-pull Architectures for Multimedia Servers[C]// 1996:45--48. [6]Subraya

B M, Subrahmanya S V. Object driven performance testing of Web applications[C]// Quality Software, 2000. Proceedings. First Asia-Pacific Conference on. IEEE, 2000:17-26.