链滴

# WebSocket 学习 (一)

作者：shark

原文链接：https://ld246.com/article/1590720997362

来源网站：链滴

# 简介

下一个项目需要做一个对于rabbitmq里队列交换机等资源的监控项目,需要用到的技术有rabbitmq的PI和websocket协议(其实最主要的是前端的大屏,领导喜欢看大屏哈哈哈哈),其实以前也用过websocke,但是都只是使用,并没有对websocket协议本身有过深入的分析了解,趁这次知识储备的机会对这个协做一个简单的了解

# 与HTTP协议的关系

同样作为应用层的协议，WebSocket在现代的软件开发中被越来越多的实践，和HTTP有很多相似的方，这里将它们简单的做一个纯个人、非权威的比较：

# 比较

### 相同点

- 都是基于TCP的应用层协议。
- 都使用Request/Response模型进行连接的建立。
- 在连接的建立过程中对错误的处理方式相同，在这个阶段WS可能返回和HTTP相同的返回码。
- 都可以在网络中传输数据。

### 不同点

- WS使用HTTP来建立连接，但是定义了一系列新的header域，这些域在HTTP中并不会使用。
- WS的连接不能通过中间人来转发，它必须是一个直接连接。
- WS连接建立之后，通信双方都可以在任何时刻向另一方发送数据。
- WS连接建立之后，数据的传输使用帧来传递，不再需要Request消息。
- WS的数据帧有序。

# 依赖关系

### WebSocket依赖于HTTP连接

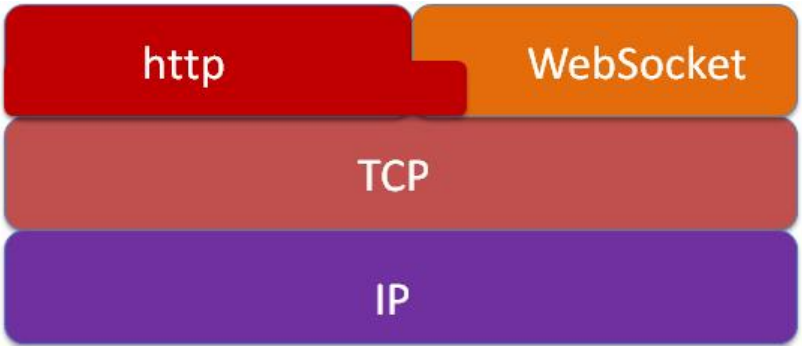
每个WebSocket连接都始于一个HTTP请求。具体来说，WebSocket协议在第一次握手连接时，通过TTP协议在传送WebSocket支持的版本号，协议的字版本号，原始地址，主机地址等等一些列字段给务器端：

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key:dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Version: 13
```

注意，关键的地方是，这里面有个Upgrade首部，用来把当前的HTTP请求升级到WebSocket协议，是HTTP协议本身的内容，是为了扩展支持其他的通讯协议。如果服务器支持新的协议，则必须返回10：

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept:s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

至此，HTTP请求物尽其用，如果成功出发onopen事件，否则触发onerror事件，后面的传输则不再赖HTTP协议。总结一下，这张图比较贴切：




WebSocket设计上就是天生为HTTP增强通信（全双工通信等），所以在HTTP协议连接的基础上是自然的一件事，并因此而能获得HTTP的诸多便利。第二，这诸多便利中有一条很重要，基于HTTP连将获得最大的一个兼容支持，比如即使服务器不支持WebSocket也能建立HTTP通信，只不过返回的onerror而已，这显然比服务器无响应要好的多。

# API介绍

简单来说，WebSocket是一种协议，与HTTP协议一样位于应用层，都是TCP/IP协议的子集。HTTP议是单向通信协议，只有客户端发起HTTP请求，服务端才会返回数据。而WebSocket协议是双向通协议，在建立连接之后，客户端和服务器都可以**主动**向对方发送或接受数据。WebSocket协议建立前提需要借助HTTP协议，建立连接之后，持久连接的双向通信就与HTTP协议无关了。

**WebSocket协议的目标是在一个独立的持久连接上提供全双工双向通信。客户端和服务器可以向对主动发送和接受数据。在JS中创建WebSocket后，会有一个HTTP请求发向浏览器以发起请求。在得服务器响应后，建立的连接会使用HTTP升级将HTTP协议转换为WebSocket协议。也就是说，使标准的HTTP协议无法实现WebSocket，只有支持那些协议的专门浏览器才能正常工作。**

由于WebScoket使用了自定义协议，所以URL与HTTP协议略有不同。未加密的连接为ws://，而不是http://。加密的连接为wss://，而不是https://。

使用JavaScript是实现WebScoket协议相对简单，以下是WebSocket APIs

```
// 打开WebSocket, 传递的参数url没有同源策略的限制。
let websocket = new WebSocket(url)

// 监听open事件，在成功建立websocket时向url发送纯文本字符串数据(如果是对象则必须序列化处
)。
websocket.onopen = () => {
  if (websocket.readyState === WebSocket.OPEN) {
    websocket.send('hello world')
  }
}

// 监听message事件，在服务器响应时接受数据。返回的数据存储在事件对象中。
websocket.onmessage = e => {
  let data = e.data
```

```
  console.log(data)
}

// 监听error事件，在发生错误时触发，连接不能持续。
websocket.onerror = () => {
  console.log('websocket connecting error!!')
}

// 监听close事件，在连接关闭时触发。只有close事件的事件对象拥有额外的信息。可以通过这些信
来查看关闭状态
websocket.onclose = e => {
  let clean = e.wasClean // 是否已经关闭
  let code = e.code // 服务器返回的数值状态码。
  let reason = e.reason //服务器返回的消息。
```

**注意，WebScoket不支持DOM2语法为事件绑定事件处理程序，因此必须使用DOM0级语法来每个件绑定事件处理程序。**

```
// correct!
websocket.onerror = () => {}
// error!
websocket.addEventListener('error', () => {})
```

WebSocket是应用层协议，是TCP/IP协议的子集，通过HTTP/1.1协议的101状态码进行握手。**也就说，WebSocket协议的建立需要先借助HTTP协议，在服务器返回101状态码之后，就可以进行web ocket全双工双向通信了，就没有HTTP协议什么事情了。**

参照wiki握手协议的例子：并对一些字段进行说明。

**Connection:**Connection必须设置为Upgrade，表示客户端希望连接升级

**Upgrade:**Upgrade必须设置为WebSocket，表示在取得服务器响应之后，使用HTTP升级将HTT 协议转换(升级)为WebSocket协议。

**Sec-WebSocket-key:**随机字符串，用于验证协议是否为WebSocket协议而非HTTP协议

**Sec-WebSocket-Version:**表示使用WebSocket的哪一个版本。

**Sec-WebSocket-Accept:**根据Sec-WebSocket-Accept和特殊字符串计算。验证协议是否为Web ocket协议。

**Sec-WebSocket-Location:**与Host字段对应，表示请求WebSocket协议的地址。

**HTTP/1.1 101 Switching Protocols:**101状态码表示升级协议，在返回101状态码后，HTTP协议 成工作，转换为WebSocket协议。此时就可以进行全双工双向通信了。

# 总结

你可以把 WebSocket 看成是 HTTP 协议为了支持长连接所打的一个大补丁，它和 HTTP 有一些共性 是为了解决 HTTP 本身无法解决的某些问题而做出的一个改良设计。在以前 HTTP 协议中所谓的 keep alive connection 是指在一次 TCP 连接中完成多个 HTTP 请求，但是对每个请求仍然要单独发 heade ；所谓的 polling 是指从客户端（一般就是浏览器）不断主动的向服务器发 HTTP 请求查询是否有新 据。这两种模式有一个共同的缺点，就是除了真正的数据部分外，服务器和客户端还要大量交换 HTTP header，信息交换效率很低。它们建立的“长连接”都是伪.长连接，只不过好处是不需要对现有的 H

TP server 和浏览器架构做修改就能实现。

WebSocket 解决的第一个问题是，通过第一个 HTTP request 建立了 TCP 连接之后，之后的交换数都不需要再发 HTTP request了，使得这个长连接变成了一个真.长连接。但是不需要发送 HTTP head r就能交换数据显然和原有的 HTTP 协议是有区别的，所以它需要对服务器和客户端都进行升级才能现。在此基础上 WebSocket 还是一个双通道的连接，在同一个 TCP 连接上既可以发也可以收信息此外还有 multiplexing 功能，几个不同的 URI 可以复用同一个 WebSocket 连接。这些都是原来的TTP 不能做到的。

另外说一点技术细节，因为看到有人提问 WebSocket 可能进入某种半死不活的状态。这实际上也是有网络世界的一些缺陷性设计。上面所说的 WebSocket 真.长连接虽然解决了服务器和客户端两边的题，但坑爹的是网络应用除了服务器和客户端之外，另一个巨大的存在是中间的网络链路。一个 HTTP WebSocket 连接往往要经过无数的路由，防火墙。你以为你的数据是在一个“连接”中发送的，实上它要跨越千山万水，经过无数次转发，过滤，才能最终抵达终点。在这过程中，中间节点的处理方很可能会让你意想不到。

比如说，这些坑爹的中间节点可能会认为一份连接在一段时间内没有数据发送就等于失效，它们会自主张的切断这些连接。在这种情况下，不论服务器还是客户端都不会收到任何提示，它们只会一厢情的以为彼此间的红线还在，徒劳地一边又一边地发送抵达不了彼岸的信息。而计算机网络协议栈的实中又会有一层套一层的缓存，除非填满这些缓存，你的程序根本不会发现任何错误。这样，本来一个好的 WebSocket 长连接，就可能在毫不知情的情况下进入了半死不活状态。

而解决方案，WebSocket 的设计者们也早已想过。就是让服务器和客户端能够发送 Ping/Pong Fram（RFC 6455 - The WebSocket Protocol）。这种 Frame 是一种特殊的数据包，它只包含一些元数而不需要真正的 Data Payload，可以在不影响 Application 的情况下维持住中间网络的连接状态。

总结参考 https://www.zhihu.com/question/20215561/answer/40250050