## Websoket总结

### Websoket之前的技术：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 技术特点 | | 缺点 |
| 轮询 | 客户端以一定的时间间隔向服务器端发出请求，以频繁请求的方式来保持客服端和服务端的同步 | | 当客户端以固定频率向服务器发起请求的时候，服务端的数据可能并没有更新，带来了很多无谓的网络传输， 是一种非常低效率的实时方案 |
| 长轮询 | 轮询的改进，当服务器端没有数据更新的时候，连接会保持一段时间周期直到数据或状态改变或者时间过期  （阻塞模型：没有信息，一直不返回response） | | 当服务端的数据变得非常频繁的话和轮询比起来没有本质上的提高 |
| 总结 | | 并不是真正的实时技术，只是使用了一种技巧来实现的模拟实时，在每次客户端和服务器端交互的时候都是一次 HTTP 的请求和应答的过程，而每一次的 HTTP 请求和应答都带有完整的 HTTP 头信息，这就增加了每次传输的数据量 | |

以上为布道师的总结，下面是自我总结（通俗易懂）：

|  |  |  |  |
| --- | --- | --- | --- |
| 轮询 | 客户端（request：） 服务端（response：没有信息）：一次无效的传输 -loop  直到莫一次服务端返回信息：一次有效地传输 | | 需要服务器有很快的处理速度和资源 |
| 长轮询 | 客户端（request:） 服务端（response: 等待信息）-loop  直到不知多久后服务端收到信息后：完成一次有效通信 | | 需要有很高的并发，也就是说同时接待客户的能力 |
| 总结 | | 都是在不断地建立HTTP连接，然后等待服务端处理，可以体现HTTP协议的另外一个特点，被动性，服务端不能主动联系客户端，只能由客户端发起 | |

### Websoket原理

1. 借用了HTTP的协议来完成一部分握手，服务端可以主动推送信息给客户端，这种程序设计叫回调。
2. 在传统的方式上，要不断的建立，关闭HTTP协议，每次都要重新传输 identity info （鉴别信息），来告诉服务端你是谁。Websocket服务端会一直知道你的信息，直到你关闭请求。
3. 类图：



### Api调研

#### Wesoket事件：

|  |  |  |
| --- | --- | --- |
| open | Socket.onopen | 此事件发生在套接字建立连接。 |
| message | Socket.onmessage | 此事件发生时，客户端收到来自服务器的数据。 |
| error | Socket.onerror | 此事件发生时有任何通信错误。 |
| close | Socket.onclose | 此事件发生在连接关闭。 |

#### Websoket方法：

|  |  |
| --- | --- |
| Socket.send() | send（data）方法用来连接传输数据。 |
| Socket.close() | close（）方法将被用于终止任何现有的连接。 |

#### Websocket 属性：

|  |  |
| --- | --- |
| Socket.readyState | readyState的代表的ReadOnly属性的连接状态。它可以有以下值：   1. 一个0值表示该连接尚未建立。 2. 值为1表示连接建立和沟通是可能的。 3. 值为2表示连接是通过将结束握手。 4. 值为3表示连接已关闭或无法打开。 |
| Socket.bufferedAmount | 读属性的bufferedAmount代表文本的字节数,utf - 8的排队使用send()方法 |

由于Websocket客户端是基于浏览器的api的，所以客户端只要写好js就行，服务端根据websocket api 写好相应的过程，就能实现简单的websocket通信，相关的资料百度上都有，对照大神代码自己做一遍就能对api更加了解.