## 长连接

长链接更快更实时

解决的问题是就是服务端有需求主动往下面发东西，比如新的新闻内容，咋弄呢，平时http协议都是浏览器主动的？

而且还要很实时。

常用技术有

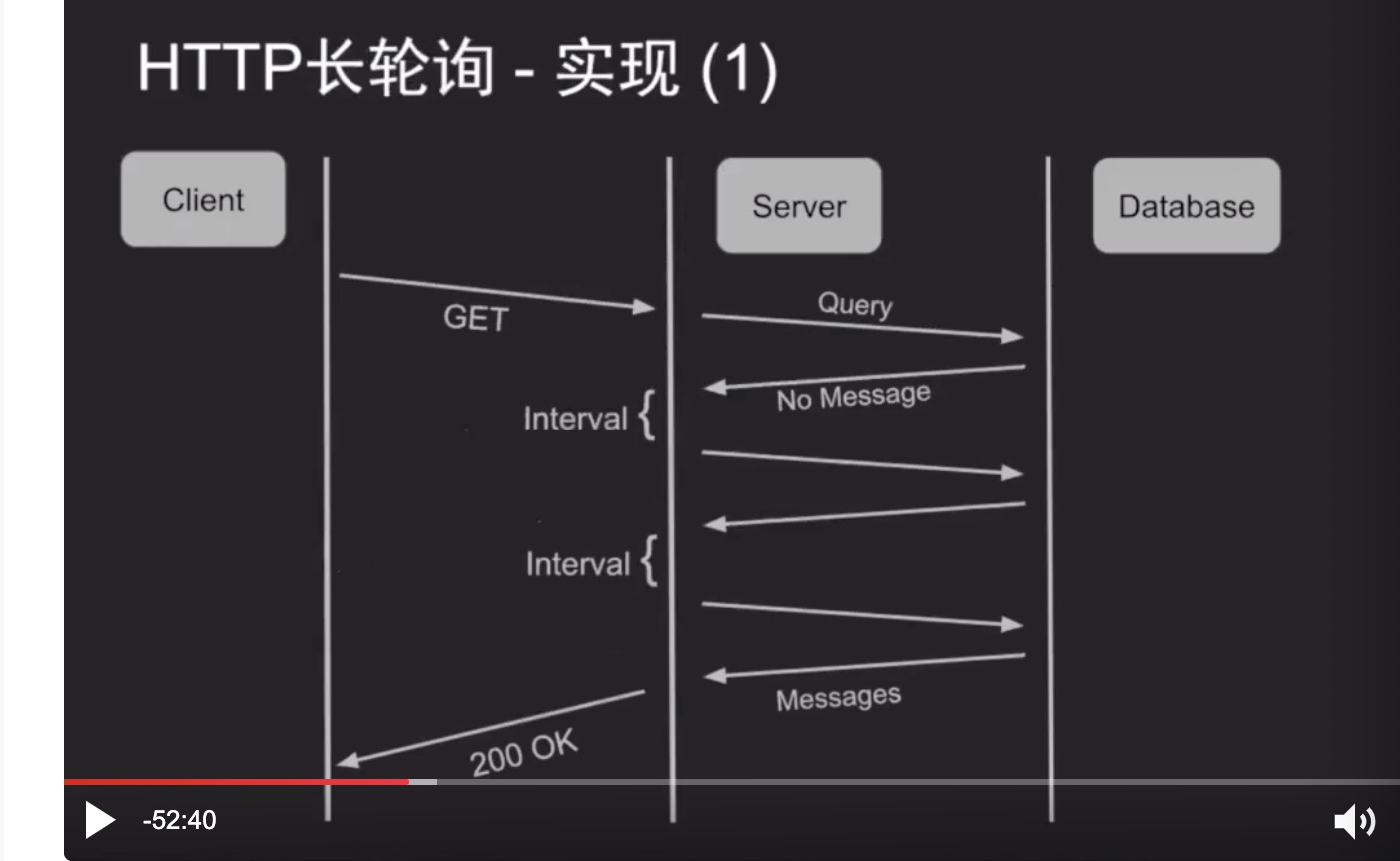


### http轮询

短轮询

就是客户端过一会就去服务器问一次，有没有新的东西，这样的话比较轻量，兼容性比较好，缺点就是实时性不好。

如果你降低间隔的话负载就会增大，还会增加客户端的能耗。



这个就是不停的去问的这个角色放在了服务端，而不是客户端了，是服务端内部去轮询有没有消息。这种长轮询肯定中间客户端和服务端的http连接肯定是长链接了，长tcp协议了。

这种对于强实时性的肯定是不行。

### WebSocket

WebSocket是HTML5新增的协议，它的目的是在浏览器和服务器之间建立一个不受限的双向通信的通道，比如说，服务器可以在任意时刻发送消息给浏览器。

为什么传统的HTTP协议不能做到WebSocket实现的功能？这是因为HTTP协议是一个请求－响应协议，请求必须先由浏览器发给服务器，服务器才能响应这个请求，再把数据发送给浏览器。换句话说，浏览器不主动请求，服务器是没法主动发数据给浏览器的。

这样一来，要在浏览器中搞一个实时聊天，在线炒股（不鼓励），或者在线多人游戏的话就没法实现了，只能借助Flash这些插件。

也有人说，HTTP协议其实也能实现啊，比如用轮询或者Comet。轮询是指浏览器通过JavaScript启动一个定时器，然后以固定的间隔给服务器发请求，询问服务器有没有新消息。这个机制的缺点一是实时性不够，二是频繁的请求会给服务器带来极大的压力。

Comet本质上也是轮询，但是在没有消息的情况下，服务器先拖一段时间，等到有消息了再回复。这个机制暂时地解决了实时性问题，但是它带来了新的问题：以多线程模式运行的服务器会让大部分线程大部分时间都处于挂起状态，极大地浪费服务器资源。另外，一个HTTP连接在长时间没有数据传输的情况下，链路上的任何一个网关都可能关闭这个连接，而网关是我们不可控的，这就要求Comet连接必须定期发一些ping数据表示连接“正常工作”。

以上两种机制都治标不治本，所以，HTML5推出了WebSocket标准，让浏览器和服务器之间可以建立无限制的全双工通信，任何一方都可以主动发消息给对方。

WebSocket协议

WebSocket并不是全新的协议，而是利用了HTTP协议来建立连接。我们来看看WebSocket连接是如何创建的。

首先，WebSocket连接必须由浏览器发起，因为请求协议是一个标准的HTTP请求，格式如下：

GET ws://localhost:3000/ws/chat HTTP/1.1

Host: localhost

Upgrade: websocket

Connection: Upgrade

Origin: http://localhost:3000

Sec-WebSocket-Key: client-random-string

Sec-WebSocket-Version: 13

该请求和普通的HTTP请求有几点不同：

GET请求的地址不是类似/path/，而是以ws://开头的地址；

请求头Upgrade: websocket和Connection: Upgrade表示这个连接将要被转换为WebSocket连接；

Sec-WebSocket-Key是用于标识这个连接，并非用于加密数据；

Sec-WebSocket-Version指定了WebSocket的协议版本。

随后，服务器如果接受该请求，就会返回如下响应：

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: server-random-string

该响应代码101表示本次连接的HTTP协议即将被更改，更改后的协议就是Upgrade: websocket指定的WebSocket协议。

版本号和子协议规定了双方能理解的数据格式，以及是否支持压缩等等。如果仅使用WebSocket的API，就不需要关心这些。

现在，一个WebSocket连接就建立成功，浏览器和服务器就可以随时主动发送消息给对方。消息有两种，一种是文本，一种是二进制数据。通常，我们可以发送JSON格式的文本，这样，在浏览器处理起来就十分容易。

为什么WebSocket连接可以实现全双工通信而HTTP连接不行呢？实际上HTTP协议是建立在TCP协议之上的，TCP协议本身就实现了全双工通信，但是HTTP协议的请求－应答机制限制了全双工通信。WebSocket连接建立以后，其实只是简单规定了一下：接下来，咱们通信就不使用HTTP协议了，直接互相发数据吧。

安全的WebSocket连接机制和HTTPS类似。首先，浏览器用wss://xxx创建WebSocket连接时，会先通过HTTPS创建安全的连接，然后，该HTTPS连接升级为WebSocket连接，底层通信走的仍然是安全的SSL/TLS协议。

浏览器

很显然，要支持WebSocket通信，浏览器得支持这个协议，这样才能发出ws://xxx的请求。目前，支持WebSocket的主流浏览器如下：

Chrome

Firefox

IE >= 10

Sarafi >= 6

Android >= 4.4

iOS >= 8

服务器

由于WebSocket是一个协议，服务器具体怎么实现，取决于所用编程语言和框架本身。Node.js本身支持的协议包括TCP协议和HTTP协议，要支持WebSocket协议，需要对Node.js提供的HTTPServer做额外的开发。已经有若干基于Node.js的稳定可靠的WebSocket实现，我们直接用npm安装使用即可。