http协议

HTTP，英文Hyper Text Transfer Protocol，也就是超文本传输协议的缩写，它主要是为了从Web服务器传输超文本到浏览器而设计的协议，由请求和响应构成。它是一种无连接的协议，也就意味着限制每次连接只处理一个请求，服务端处理完成且收到客户端应答后立即断开连接；同时也是无状态的，也就意味着没有记忆能力，每次连接都需要带上需要的信息。

HTTP1.1版本中给出一种持续连接的机制，绝大多数的Web开发，都是构建在HTTP协议之上的Web应用。

常用的HTTP方法有哪些？

GET： 用于请求访问已经被URI（统一资源标识符）识别的资源，可以通过URL传参给服务器。 **POST：**用于传输信息给服务器，主要功能与GET方法类似，但一般推荐使用POST方式。 **PUT：** 传输文件，报文主体中包含文件内容，保存到对应URI位置。 **HEAD：** 获得报文首部，与GET方法类似，只是不返回报文主体，一般用于验证URI是否有效。 **DELETE：**删除文件，与PUT方法相反，删除对应URI位置的文件。 **OPTIONS：**查询相应URI支持的HTTP方法。

GET方法与POST方法的区别

1. get重点在从服务器上获取资源，post重点在向服务器发送数据；
2. get传输数据是通过URL请求，以field（字段）= value的形式，置于URL后，并用"?"连接，多个请求数据间用"&"连接，如http://127.0.0.1/Test/login.action?name=admin&password=admin，这个过程用户是可见的；post传输数据通过http的post机制，将字段与对应值封存在请求实体中发送给服务器，这个过程对用户是不可见的；
3. get传输的数据量小，因为受URL长度限制，但效率较高;post可以传输大量数据，所以上传文件时只能用post方式；
4. get是不安全的，因为URL是可见的，可能会泄露私密信息，如密码等；post较get安全性较高；
5. get方式只能支持ASCII（音as key）字符，向服务器传的中文字符可能会乱码；post支持标准字符集，可以正确传递中文字符。

HTTP请求报文与响应报文格式

请求报文包含四部分： a、请求行：包含请求方法、URI、HTTP版本信息 b、请求首部字段 c、请求内容实体 d、空行

响应报文包含四部分： a、状态行：包含HTTP版本、状态码、状态码的原因短语 b、响应首部字段 c、响应内容实体 d、空行

常见的HTTP相应状态码

返回的状态

1xx：指示信息--表示请求已接收，继续处理 2xx：成功--表示请求已被成功接收、理解、接受 3xx：重定向--要完成请求必须进行更进一步的操作 4xx：客户端错误--请求有语法错误或请求无法实现 5xx：服务器端错误--服务器未能实现合法的请求

200：请求被正常处理 **204：**请求被受理但没有资源可以返回 **206：**客户端只是请求资源的一部分，服务器只对请求的部分资源执行GET方法，相应报文中通过Content-Range指定范围的资源。 **301：**永久性重定向 **302：**临时重定向 **303：**与302状态码有相似功能，只是它希望客户端在请求一个URI的时候，能通过GET方法重定向到另一个URI上 **304：**发送附带条件的请求时，条件不满足时返回，与重定向无关 **307：**临时重定向，与302类似，只是强

制要求使用POST方法 **400：**请求报文语法有误，服务器无法识别 **401：**请求需要认证 **403：**请求的对应资源禁止被访问 **404：**服务器无法找到对应资源 **500：**服务器内部错误 **503：**服务器正忙

HTTP1.1版本新特性

a、默认持久连接节省通信量，只要客户端服务端任意一端没有明确提出断开TCP连接，就一直保持连接，可以发送多次HTTP请求

b、管线化，客户端可以同时发出多个HTTP请求，而不用一个个等待响应

c、断点续传原理

常见HTTP首部字段

a、通用首部字段（请求报文与响应报文都会使用的首部字段） Date：创建报文时间 Connection：连接的管理 Cache-Control：缓存的控制 Transfer-Encoding：报文主体的传输编码方式

b、请求首部字段（请求报文会使用的首部字段） Host：请求资源所在服务器 Accept：可处理的媒体类型 Accept-Charset：可接收的字符集 Accept-Encoding：可接受的内容编码 Accept-Language：可接受的自然语言

c、响应首部字段（响应报文会使用的首部字段） Accept-Ranges：可接受的字节范围 Location：令客户端重新定向到的URI Server：HTTP服务器的安装信息

d、实体首部字段（请求报文与响应报文的的实体部分使用的首部字段） Allow：资源可支持的HTTP方法 Content-Type：实体主类的类型 Content-Encoding：实体主体适用的编码方式 Content-Language：实体主体的自然语言 Content-Length：实体主体的的字节数 Content-Range：实体主体的位置范围，一般用于发出部分请求时使用

HTTP的缺点与HTTPS

a、通信使用明文不加密，内容可能被窃听 b、不验证通信方身份，可能遭到伪装 c、无法验证报文完整性，可能被篡改

HTTPS就是HTTP加上加密处理（一般是SSL安全通信线路）+认证+完整性保护

HTTP优化

利用负载均衡优化和加速HTTP应用；利用HTTP Cache来优化网站

其他题目

http和https的区别

1. HTTP 的 URL 以 http:// 开头，而 HTTPS 的 URL 以 https:// 开头
2. HTTP 是不安全的，而 HTTPS是安全的
3. HTTP 标准端口是80 ，而 HTTPS 的标准端口是443
4. 在 OSI 网络模型中，HTTP 工作于应用层，而 HTTPS 的安全传输机制工作在传输层
5. HTTP 无法加密，而 HTTPS 对传输的数据进行加密
6. HTTP 无需证书，而 HTTPS 需要 CA 机构 wosign 的颁发的 SSL 证书

什么是http协议是无状态协议?怎么解决http协议无状态?

- 无状态协议对于事务处理没有记忆能力，缺少状态意味着如果后续处理需要前面的信息。也就是说，当客户端一次HTTP请求完成以后，客户端再发送一次HTTP请求，HTTP并不知道当前客户端是一个”老用户“。

- 可以使用Cookie来解决无状态的问题，Cookie就相当于一个通行证，第一次访问的时候服务端给客户端发送一个Cookie，当客户端再次来的时候，拿着Cookie(通行证)，那么服务器就知道这个是"老用户"。

http1.1长连接（持久连接）

如今所使用的HTTP协议基本上都是1.1的，默认为长连接。一个HTTP请求的响应头里面，默认会有这么一行代码：`Connection:keep-alive`。HTTP长连接允许了HTTP设备在请求结束之后将TCP连接保持在打开的状态，以便为之后的HTTP请求重用现存的连接。从本质上来讲，HTTP仅仅是应用层的协议，TCP才是真正的传输层协议，因此TCP才有真正的长连接短连接的说法。

> 短连接的步骤为：建立连接——数据传输——关闭连接...建立连接——数据传输——关闭连接

> 长连接的步骤为：建立连接——数据传输...(保存连接)...数据传输——关闭连接

长连接的带来的好处是显而易见的，它复用了TCP通道。通俗的来说，使用短连接发起多个HTTP请求时，每次都会去重新建立TCP连接，这样会造成严重的资源浪费。若采用长连接，什么三次握手四次挥手可能只需要进行一次，这无疑减少了很多消耗。

当然，长连接也并不是永久无限制的连接。服务器能够对长连接进行限制。

```
Connection:Keep-Alive
Keep-Alive:max=5,timeout=120
```

像上面这样的HTTP响应头，说明服务器最多还会为另外5个事务保存连接打开的状态或者保持连接打开状态至空闲120秒后。

从长连接的角度来说，keep-alive和Websocket有什么区别？

1. 协议不同，一个是http1.1协议，一个是websocket协议
2. http长连接是：客户端往服务器端发送一个http请求，然后服务端等到任务完成之后才返回。这个任务完成的时间可以是几个小时，而一般的http请求得到返回的时间没那么长。客户端得到服务器端返回的信息之后，马上又发一个请求，然后服务器端继续前面的事情，也就是不立即返回，等有了新的信息之后再返回。
3. websocket是指客户端和服务器之间建立一个websockt连接，然后客户端可以给服务器发信息，服务器也可以给客户端发信息。从行为上来说，对比长连接，服务器的不需要把请求hold住，而客户端也不需要接收一次信息后再立刻发送请求。 在没有websocket之前，只有长连接可以选择。
4. 也可以简单的理解为，keep-alive 只是一种为了达到复用TCP连接的"协商"行为，双方并没有建立正真的连接会话，服务端也可以不认可，也可以随时（在任何一次请求完成后）关闭掉。WebSocket 不同，它本身就规定了是正真的、双工的长连接，两边都必须要维持住连接的状态。

http1.1 管线化

通常，http1.0 请求总是顺序发送的，下一个请求只有在当前请求的响应被完全接受的时候才会被发送。由于网络延迟和带宽的限制，这样会导致在服务器发送下一个响应的时候中间有很大的延迟。

http/1.1 允许多个http请求通过一个套接字同时被输出 ，而不用等待相应的响应。然后请求者就会等待各自的响应，这些响应是按照之前请求的顺序依次到达。

http 断点续传（分块传输）

断点续传：指的是在上传/下载时，将任务（一个文件或压缩包）人为的划分为几个部分，每一个部分采用一个线程进行上传/下载，如果碰到网络故障，可以从已经上传/下载的部分开始继续上传/下载未完成的部分，而没有必要从头开始上传/下载。可以节省时间，提高速度。

四个必须的HTTP头域

- Range
  用于请求头中，指定第一个字节的位置和最后一个字节的位置，一般格式：
  Range:(unit=first byte pos)-[last byte pos]
  Range : bytes=50- 从第50个字节开始到最后一个字节
  Range : bytes=-70 最后的70个字节
  Range : bytes=50-100 从第50字节到100字节
- Content-Range
  用于响应头，指定整个实体中的一部分的插入位置，他也指示了整个实体的长度。在服务器向客户返回一个部分响应，它必须描述响应覆盖的范围和整个实体长度。一般格式：
  Content-Range: bytes (unit first byte pos) - [last byte pos]/[entity legth]
- Accept-Ranges
  Accept-Ranges:bytes // 告诉客户端支持断点传输
- Content-Length
  内容长度

http返回码

```
200  OK（一切正常）
206  Partial Content（服务器已经成功处理了部分内容）
416  Requested Range Not Satisfiable 对方（客户端）发来的Range 请求头不合理
```

实例1

```
//请求：
GET /123.zip HTTP/1.1
```

```
//响应：
HTTP/1.1 200 OK
Accept-Ranges : bytes    //告诉客户端支持断点传输
Content-Length : 1900 //文件总大小
Content-Type : image/jpeg //文件类型
```

```
//中间停止下载，重新发起请求
GET /123.zip HTTP/1.1
Range：bytes=580-
```

```
//响应：
HTTP/1.1 206 Partial Content
Accept-Ranges : bytes
Content-Type : image/jpeg //文件类型
Content-Length : （1900 - 580） //长度则不是总长度了，而580到1900共有多少字节。
Content-Range :bytes 580-（1900-1 ） / 1900    //因为到了文件总大小了，所以指定范围永远不能超过等于总大小；但是到了最后一个字节必须输出出去，完全是写法问题。
```

实例 2

```
//请求：
GET /test.rar HTTP/1.1
Connection: close
Host: 116.1.219.219
Range: bytes=0-801 //一般请求下载整个文件是bytes=0- 或不用这个头
```

```
//一般正常回应
HTTP/1.1 200 OK
Content-Length: 801
Content-Type: application/octet-stream
Content-Range: bytes 0-800/801 //801:文件总大小
```

https工作原理

HTTPS其实是有两部分组成：HTTP + SSL / TLS，也就是在HTTP上又加了一层处理加密信息的模块。服务端和客户端的信息传输都会通过TLS进行加密，所以传输的数据都是加密后的数据。

1. 首先HTTP请求服务端生成证书，客户端对证书的有效期、合法性、域名是否与请求的域名一致、证书的公钥（RSA加密）等进行校验；
2. 客户端如果校验通过后，就根据证书的公钥的有效， 生成随机数，随机数使用公钥进行加密（RSA加密）；
3. 消息体产生的后，对它的摘要进行MD5（或者SHA1）算法加密，此时就得到了RSA签名；
4. 发送给服务端，此时只有服务端（RSA私钥）能解密。
5. 解密得到的随机数，再用AES加密，作为密钥（此时的密钥只有客户端和服务端知道）。

一级域名？二级域名指什么？

- .com ：顶级域名
- baidu.com： 一级域名
- tieba.baidu.com / www.baidu.com 二级域名

一个页面从输入URL到页面加载显示完成，这个过程都发生什么？

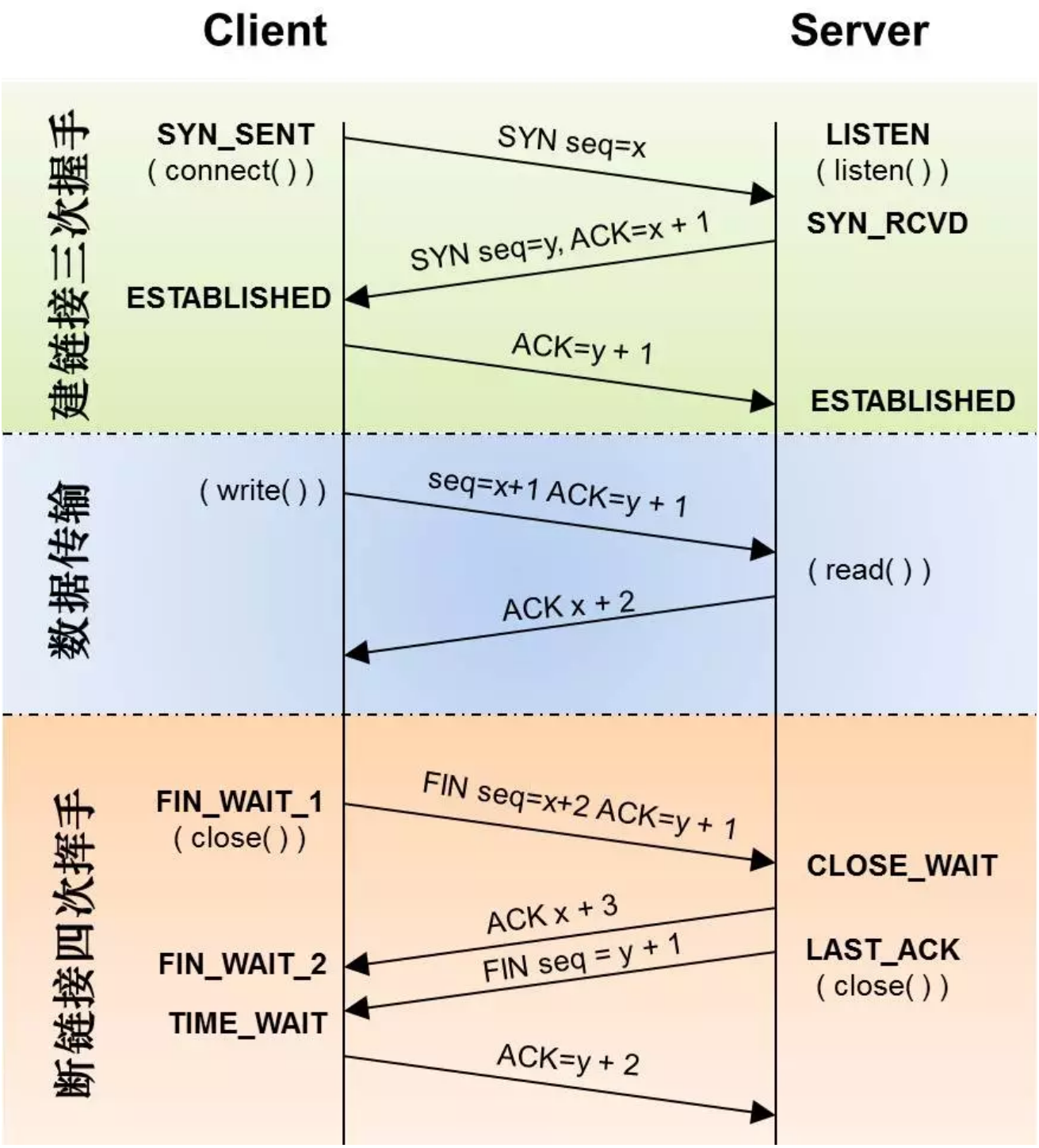
1. 首先，在浏览器地址栏中输入URL
2. 浏览器先查看**浏览器缓存-系统缓存-路由器缓存**，如果缓存中有，会直接在屏幕中显示页面内容。若没有，则跳到第三步操作
3. 在发送HTTP请求前，需要**域名解析**(DNS解析)，解析获取相应的IP地址
4. 浏览器向服务器发起**TCP连接**，与浏览器建立**TCP三次握手**
5. 握手成功后，浏览器向服务器发送HTTP请求，请求数据包
6. 服务器处理收到的请求，将数据返回至浏览器
7. 浏览器收到HTTP响应
8. 读取页面内容，浏览器渲染，解析HTML源码
9. 生成DOM树，解析CSS样式，JS交互
10. 客户端和服务器交互，AJAX查询

步骤2的具体过程是：

- **浏览器缓存**：浏览器会记录DNS一段时间，因此，只是第一个地方解析DNS请求；
- **操作系统缓存：**如果在浏览器缓存中不包含这个记录，则会使系统调用操作系统，获取操作系统的记录(保存最近的DNS查询缓存)；
- **路由器缓存**：如果上述两个步骤均不能成功获取DNS记录，继续搜索路由器缓存（临时文件存储空间）；

- **ISP缓存：**若上述均失败，继续向ISP搜索。

TCP三次握手和四次握手



三次握手

1. 第一次握手：建立连接。客户端发送连接请求报文段，将SYN位置为1，Sequence Number为x；然后，客户端进入SYN_SEND状态，等待服务器的确认。
2. 第二次握手：服务器收到SYN报文段。服务器收到客户端的SYN报文段，需要对这个SYN报文段进行确认，设置Acknowledgment Number为x+1(Sequence Number+1)；同时，自己自己还要发送SYN请求信息，将SYN位置为1，Sequence Number为y；服务器端将上述所有信息放到一个报文段（即SYN+ACK报文段）中，一并发送给客户端，此时服务器进入SYN_RECV状态。

3. 第三次握手：客户端收到服务器的SYN+ACK报文段。然后将Acknowledgment Number设置为y+1，向服务器发送ACK报文段，这个报文段发送完毕以后，客户端和服务器端都进入ESTABLISHED状态，完成TCP三次握手。
   完成了三次握手，客户端和服务器端就可以开始传送数据。以上就是TCP三次握手的总体介绍。

四次握手

当客户端和服务器通过三次握手建立了TCP连接以后，当数据传送完毕，肯定是要断开TCP连接的啊。那对于TCP的断开连接，这里就有了神秘的“四次分手”。

1. 第一次分手：主机1（可以使客户端，也可以是服务器端），设置Sequence Number和Acknowledgment Number，向主机2发送一个FIN报文段；此时，主机1进入FIN_WAIT_1状态；这表示主机1没有数据要发送给主机2了。
2. 第二次分手：主机2收到了主机1发送的FIN报文段，向主机1回一个ACK报文段，Acknowledgment Number为Sequence Number加1；主机1进入FIN_WAIT_2状态；主机2告诉主机1，我“同意”你的关闭请求。
3. 第三次分手：主机2向主机1发送FIN报文段，请求关闭连接，同时主机2进入LAST_ACK状态。
4. 第四次分手：主机1收到主机2发送的FIN报文段，向主机2发送ACK报文段，然后主机1进入TIME_WAIT状态；主机2收到主机1的ACK报文段以后，就关闭连接；此时，主机1等待2MSL后依然没有收到回复，则证明Server端已正常关闭，那好，主机1也可以关闭连接了。

为什么需要“三次握手”

“为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误”

“已失效的连接请求报文段”的产生在这样一种情况下：client发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达server。本来这是一个早已失效的报文段。但server收到此失效的连接请求报文段后，就误认为是client再次发出的一个新的连接请求。于是就向client发出确认报文段，同意建立连接。

Websocket

WebSocket 是什么？

WebSocket 是 HTML5 开始提供的一种在单个 TCP 连接上进行全双工通讯的协议。

为什么需要 WebSocket ？

了解计算机网络协议的人，应该都知道：HTTP 协议是一种无状态的、无连接的、单向的应用层协议。它采用了请求/响应模型。通信请求只能由客户端发起，服务端对请求做出应答处理。

这种通信模型有一个弊端：HTTP 协议无法实现服务器主动向客户端发起消息。

这种单向请求的特点，注定了如果服务器有连续的状态变化，客户端要获知就非常麻烦。大多数 Web 应用程序将通过频繁的异步JavaScript和XML（AJAX）请求实现长轮询。轮询的效率低，非常浪费资源（因为必须不停连接，或者 HTTP 连接始终打开）。

WebSocket 连接允许客户端和服务器之间进行全双工通信，以便任一方都可以通过建立的连接将数据推送到另一端。WebSocket 只需要建立一次连接，就可以一直保持连接状态。这相比于轮询方式的不停建立连接显然效率要大大提高。

WebSocket 如何工作？

Web浏览器和服务器都必须实现 WebSockets 协议来建立和维护连接。由于 WebSockets 连接长期存在，与典型的HTTP连接不同，对服务器有重要的影响。

基于多线程或多进程的服务器无法适用于 WebSockets，因为它旨在打开连接，尽可能快地处理请求，然后关闭连接。**任何实际的 WebSockets 服务器端实现都需要一个异步服务器。**

Websocket的使用及API

```
// 客户端
var ws = new WebSocket('wss://example.com/socket'); //打开新的安全 WebSocket 连接(wss)
ws.onerror = function (error) { ... } ;//可选的回调，在连接出错时调用
ws.onclose = function () { ... } ;//可选的回调，在连接终止时调用
ws.onopen = function () { //可选的回调，在 WebSocket 连接建立时调用
ws.send("Connection established. Hello server!"); //客户端先向服务器发送一条消息
}
ws.onmessage = function(msg) {//回调函数，服务器每发回一条消息就调用一次
    if(msg.data instanceof Blob) {//根据接收到的消息，决定调用二进制还是文本处理逻辑
        processBlob(msg.data);
    } else {
        processText(msg.data);
    }
}
```

HTTP 和 WebSocket 有什么关系

Websocket 其实是一个新协议，跟 HTTP 协议基本没有关系，只是为了兼容现有浏览器的握手规范而已，也就是说它是 HTTP 协议上的一种补充。 HTTP协议的服务端具有“被动性”，不能主动与客户端通信，得先让客户端发出请求，而WebSocket则是具有“主动性”，能主动与客户端通信。