HTML5程序设计基础

第七章 数据通信

主要内容

01 Web中传统数据通信方式的缺陷

02 让网页实时通信（WebSocket）

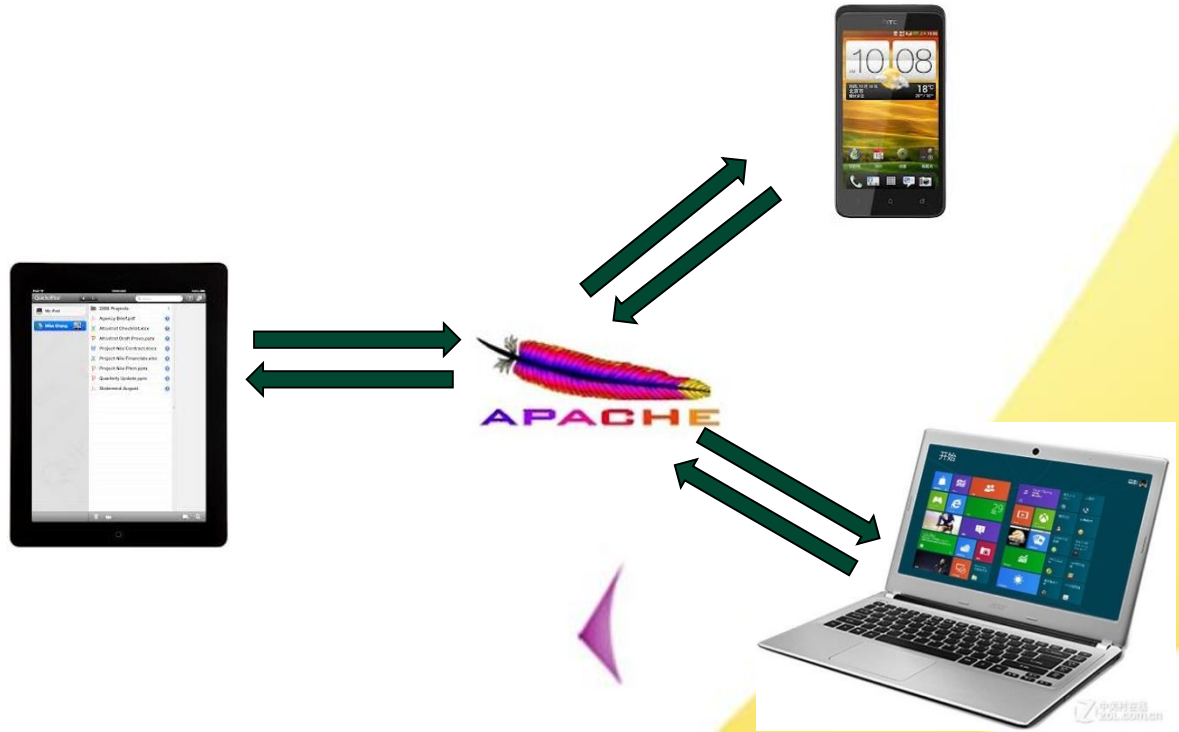
03 WebSocket客户端

04 WebSocket服务端

01 Web中传统数据通信方式的缺陷

数据通信

- Web应用需要从服务器获取页面或数据，也需要和其他用户进行数据交换。这些都是Web中的数据通信。
- Web客户端程序和Web服务器如何进行通信？依据什么协议？



数据通信

Web应用的数据通信方式

HTTP协议

- 一种无状态的、无连接的、单向的应用层协议。
- 采用了请求/响应模型，Request = Response 。
- 通信请求只能由客户端发起，服务端对请求做出应答处理。

Ajax

- 由JavaScript发起请求，局部更新页面/应用。

HTTP协议的缺陷

- 服务器能否推送数据？
  - 通信只能由客户端发起。服务器总是被动的！

如何实现：
微博私信、网页斗地主、在线证券...

Ajax轮询与长轮询的原理

服务器推送数据的解决方案

- 轮询（Ajax）、长轮询（long poll）
- ajax轮询，即让浏览器隔几秒就发送一次请求，询问服务器是否有新信息。
- 长轮询，采取的是阻塞模型，即客户端发起连接后，如果没消息，就一直不返回Response给客户端，直到有消息才返回。返回完之后，客户端再次建立连接，周而复始。

场景再现：

客户端：啦啦啦，有没有新信息(Request)

服务端：没有（Response）

客户端：啦啦啦，有没有新信息(Request)

服务端：没有。。（Response）

客户端：啦啦啦，有没有新信息(Request)

服务端：你好烦啊，没有啊。。（Response）

客户端：啦啦啦，有没有新消息（Request）

服务端：好啦好啦，有啦给你。（Response）

客户端：啦啦啦，有没有新消息（Request）

服务端：。。。。。没。。。。没。。。没有（Response）

场景再现

客户端：啦啦啦，有没有新信息，没有的话就等有了才返回给我吧（Request）

服务端：额。。等待到有消息的时候。。来 给你（Response）

客户端：啦啦啦，有没有新信息，没有的话就等有了才返回给我吧（Request）

数据通信

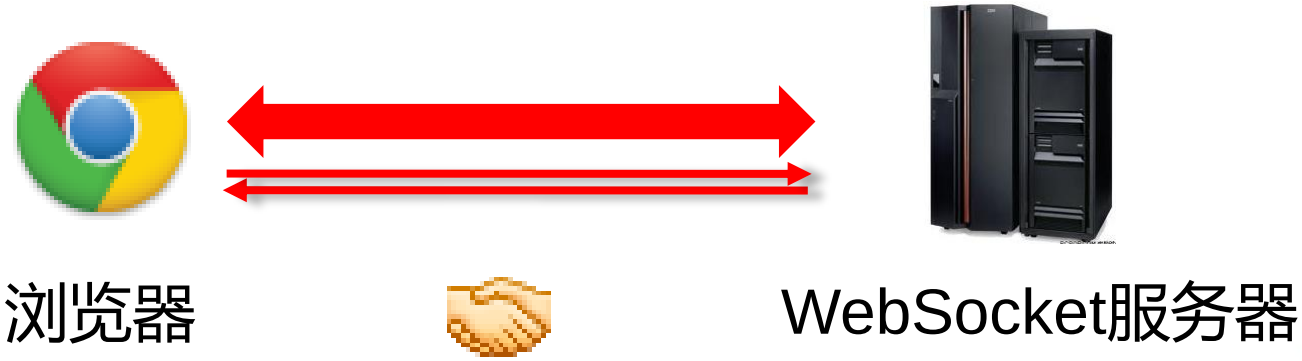
- 在构建交互更丰富的Web应用时，他们都有一定缺陷。
  - 非协议原生支持
  - 效率低、实时性低
  - 服务器资源浪费、网络带宽浪费
    - ajax轮询 需要服务器有很快的处理速度和资源
    - long poll 需要有很高的并发

02

让网页实时通信（WebSocket）

WebSocket简介

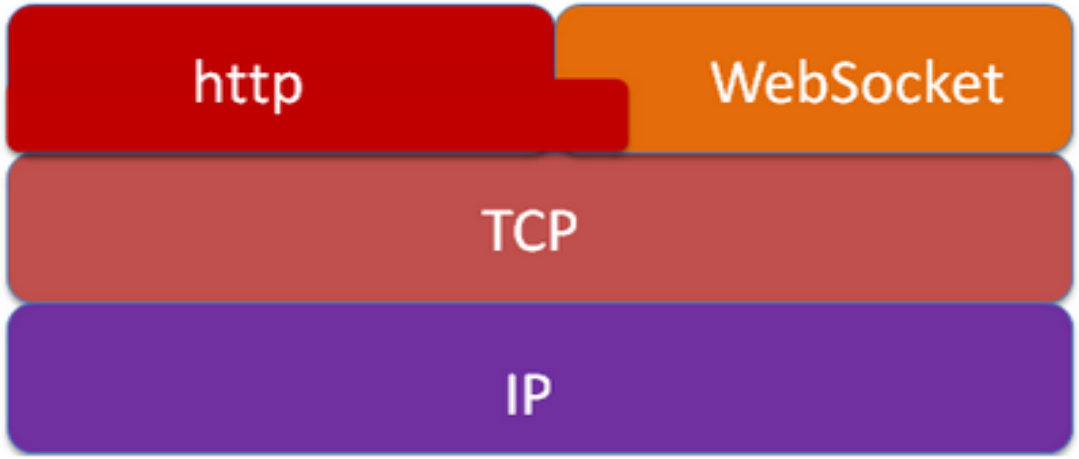
- WebSocket是HTML5提供的一种浏览器与服务器间进行全双工通讯的网络通讯协议。
- WebSocket原理



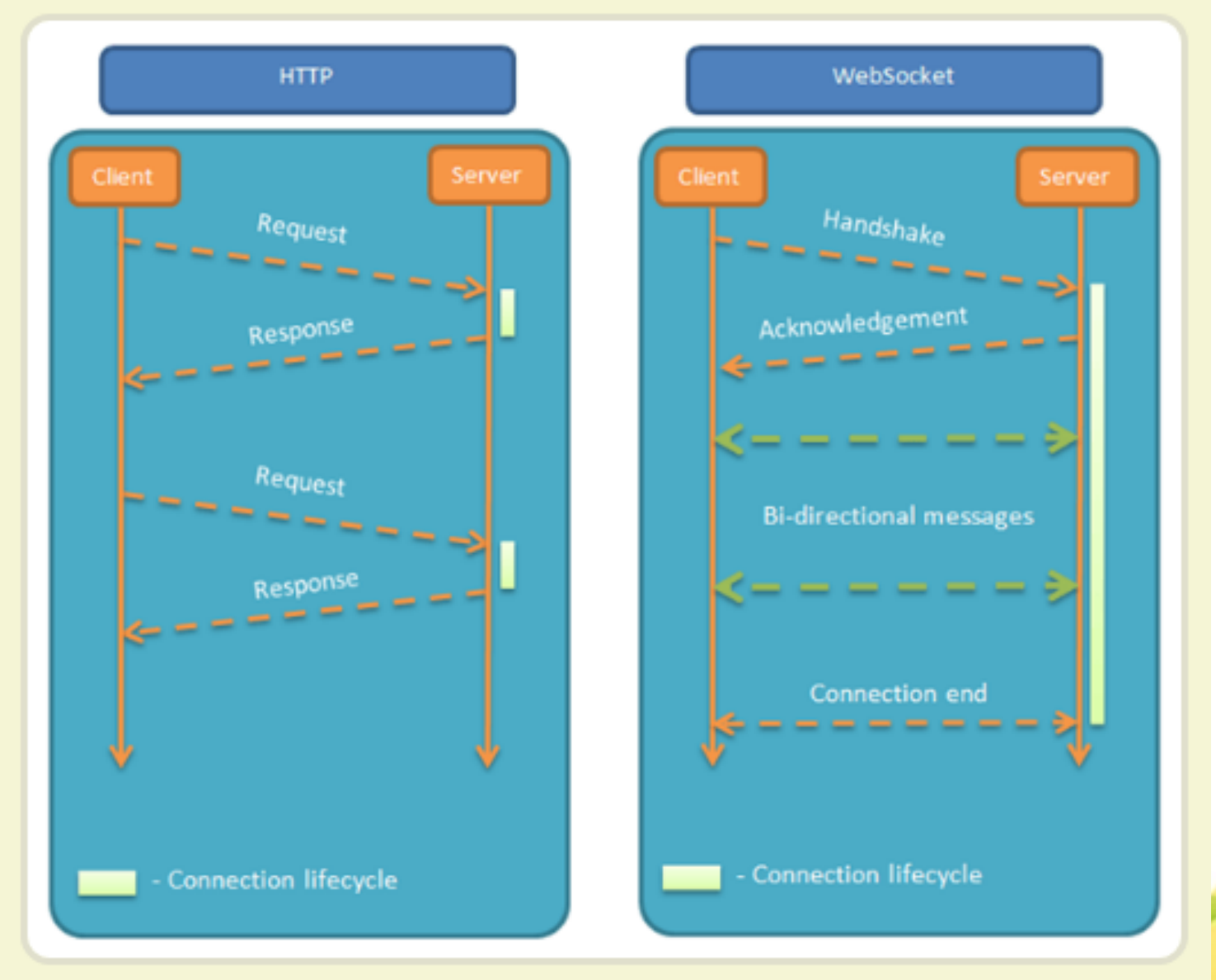
浏览器和服务器只需要做一个握手的动作，便形成了一条快速通道。两者之间就直接可以数据互相传送。

WebSocket、HTTP与TCP

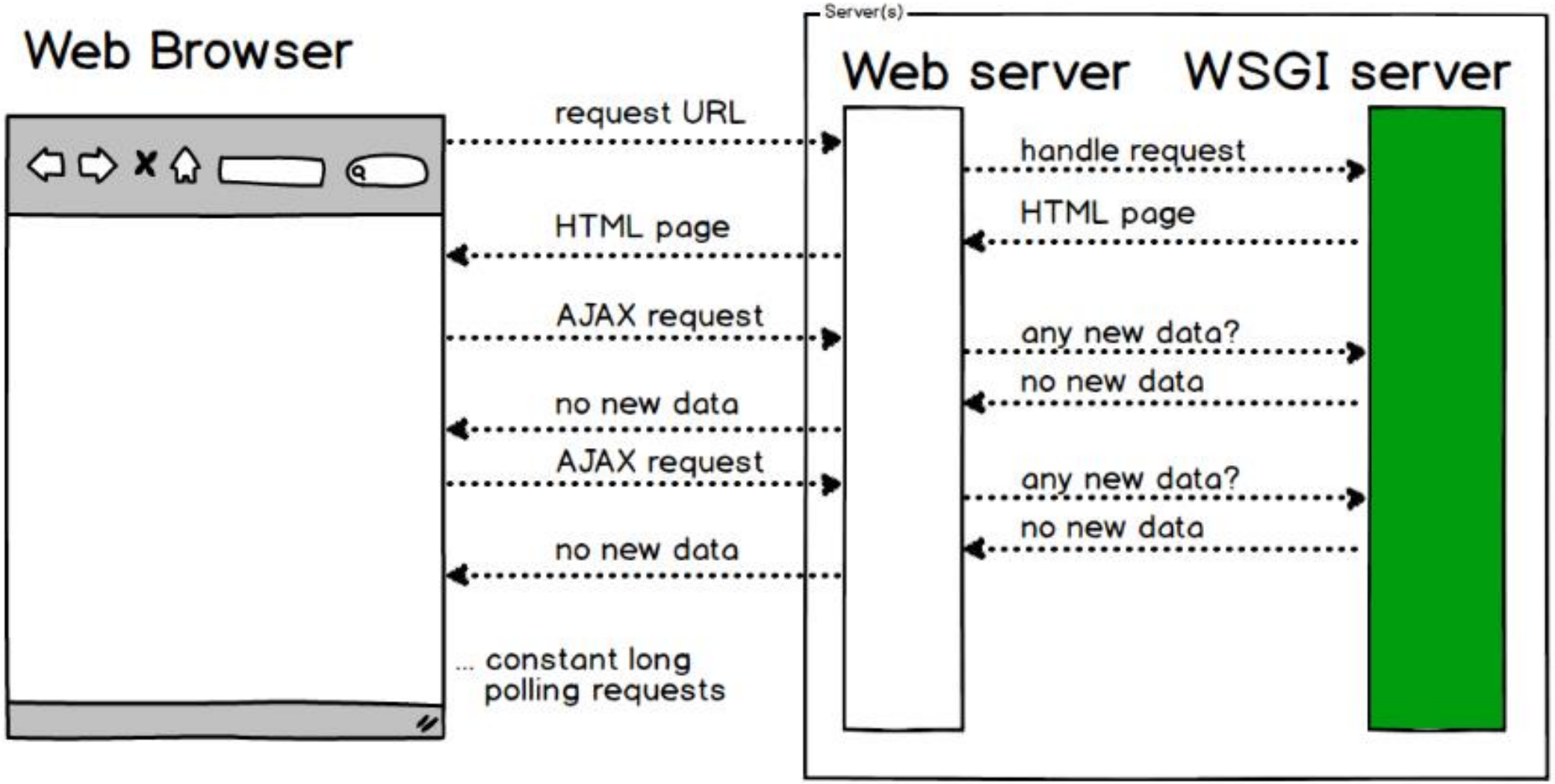
- WebSocket和HTTP属于应用层协议，都是通过TCP协议传输数据。
- WebSocket是全双工通信协议，HTTP是单向的通信协议。
- 对于WebSocket来说，它必须依赖HTTP协议进行一次握手，握手成功后，数据就直接从TCP通道传输，此后就与HTTP无关了。



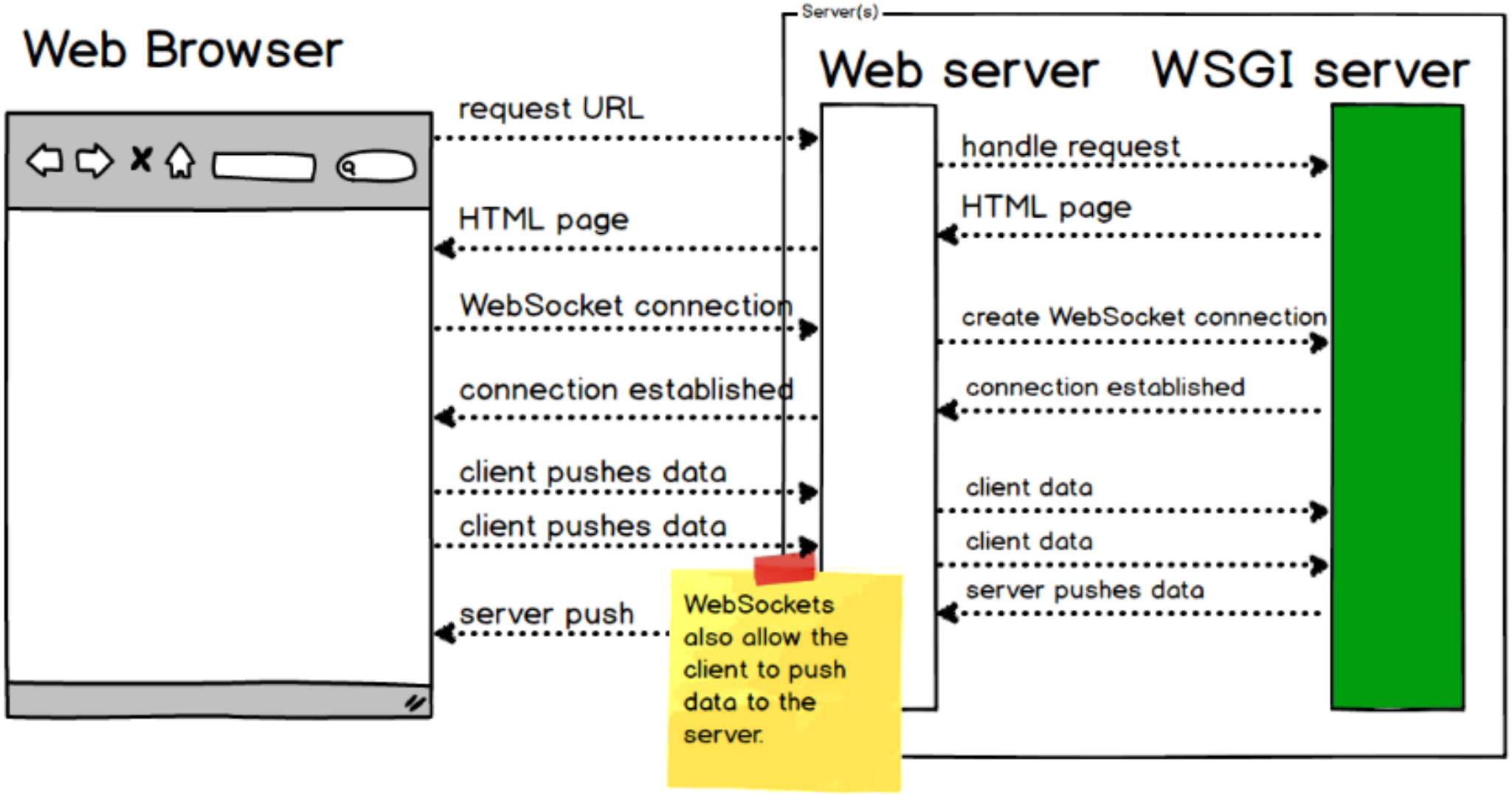
HTTP与WebSocket



通过Ajax长轮询


Server(s)
Web Browser
Web server
WSGI server
request URL
handle request
HTML page
HTML page
AJAX request
any new data?
no new data
no new data
AJAX request
any new data?
no new data
no new data
... constant long
polling requests

WebSocket


Web Browser
Server(s)
Web server
WSGI server
request URL
handle request
HTML page
HTML page
WebSocket connection
create WebSocket connection
connection established
connection established
client pushes data
client data
client pushes data
client data
server pushes data
server push
WebSockets also allow the client to push data to the server.

WebSocket特点

- （1）建立在 TCP 协议之上，服务器端的实现比较容易。
- （2）与 HTTP 协议有着良好的兼容性。默认端口也是80和443，握手阶段采用 HTTP 协议，不容易屏蔽，能通过各种 HTTP 代理服务器。
- （3）数据格式比较轻量，性能开销小，通信高效。
- （4）可以发送文本，也可以发送二进制数据。
- （5）没有同源限制，客户端可以与任意服务器通信。
- （6）协议标识符是ws（如果加密，则为wss），服务器网址就是 URL。

ws://example.com:80/some/path

WebSocket意义

- WebSocket 是HTML5中最强大的通信功能，仅通过一个Socket就可以进行通信，不仅是对常规HTTP通信的一种增量加强，它更代表着一次巨大的进步，对实时的、事件驱动的Web应用程序更是如此。

  --Peter Lubbers

- 有了WebSocket以后：
  - 数据双向实时传输
  - 在Web中轻松实现：
    - 网游、在线聊天
    - 股票、位置监控...

03 WebSocket客户端

使用WebSocket

WebSocket 构造函数

- WebSocket 对象作为一个构造函数，用于新建 WebSocket 实例。
- var Socket = new WebSocket(url, [protocol]);
- 第一个参数 url, 指定连接的 URL。第二个参数 protocol 是可选的，指定了可接受的子协议。

```
var Socket = new WebSocket('ws://localhost:8080');
```

WebSocket属性

属性	描述
WebSocket.readyState	返回实例对象的当前状态，共有四种。 CONNECTING：值为0，表示正在连接。 OPEN：值为1，表示连接成功，可以通信了。 CLOSING：值为2，表示连接正在关闭。 CLOSED：值为3，表示连接已经关闭，或者打开连接失败。
WebSocket.bufferedAmount	表示已被 send() 放入正在队列中等待传输，但是还没有发出二进制数据。

WebSocket属性

```
switch (ws.readyState) {
      case WebSocket.CONNECTING:
        // do something
        break;
      case WebSocket.OPEN:
        // do something
        break;
      case WebSocket.CLOSING:
        // do something
        break;
      case WebSocket.CLOSED:
        // do something
        break;
      default:
        // this never happens
        break;
}
```

WebSocket属性

- 实例对象的bufferedAmount属性可以用来判断发送是否结束。

```
var socket = new WebSocket(url, [protocol] );
var data = new ArrayBuffer(10000000);
socket.send(data);

if (socket.bufferedAmount === 0) {
  // 发送完毕
} else {
  // 发送还没结束
}
```

WebSocket事件

事件	事件处理程序	描述
open	WebSocket.onopen	连接建立时触发
message	WebSocket.onmessage	客户端接收服务端数据时触发
error	WebSocket.onerror	通信发生错误时触发
close	WebSocket.onclose	连接关闭时触发

使用WebSocket

- window.WebSocket.onopen
  - 实例对象的onopen属性，用于指定连接成功后的回调函数。
- window.WebSocket.onmessage
  - 实例对象的onmessage属性，用于指定收到服务器数据后的回调函数。
- window.WebSocket.onclose
  - 实例对象的onclose属性，用于指定连接关闭后的回调函数。

使用WebSocket

```
var ws = new WebSocket('ws://localhost:8080');
```

```
ws.onopen = function () {
    ws.send('Hello Server!');
}
```

```
ws.addEventListener('open', function (event) {
    ws.send('Hello Server!');
});
```

```
ws.onclose = function(event) {
    var code = event.code;
    var reason = event.reason;
    var wasClean = event.wasClean;
    // handle close event
};
```

```
ws.onmessage = function(event) {
    var data = event.data;
    // 处理数据
};
```

WebSocket方法

方法	描述
WebSocket.send( )	使用连接发送数据
WebSocket.close( )	关闭连接

- window.WebSocket.send()
  - 实例对象的send()方法用于向服务器发送数据。
- window.WebSocket.close()
  - 实例对象的close()方法用于关闭连接。

使用WebSocket

```
//设定WebSocket服务器地址
var url = "ws://localhost:8888";
//建立WebSocket连接，返回连接的WebSocket对象
var ws = new WebSocket(url);
//连接事件
ws.onopen = function(){
    console.log("Connection open");
}
//消息事件
ws.onmessage = function(e){
    //事件对象的data属性保存了服务器传递来的具体数据
    console.log(e.data);
}
//关闭连接事件
ws.oncolse = function(){
    console.log("Closed");
}

document.getElementById("sendButton").onclick = function(){
    //WebSocket对象发送数据
    ws.send("Send Message");
}
```

demo7-1.html

04 WebSocket服务端

WebSocket服务端

- WebSocket服务器：
  - WebSocket 在服务端的实现非常丰富。Node.js、Java、C++、Python 等多种语言都有自己的解决方案。
  - 对WebSocket支持的服务器：
    - 基于php - http://code.google.com/p/phpwebsocket/
    - 基于ruby - http://github.com/gimite/web-socket-ruby
    - 基于Tomcat - http://tomcat.apache.org/ (7.0.26支持)
    - 基于node.js - https://github.com/Worlize/WebSocket-Node

WebSocket服务端

- 常用的 Node 实现有以下三种：
  - µWebSockets
  - Socket.IO
  - WebSocket-Node

使用WebSocket

使用WebSocket

- 基于WebSocket实现用户和用户聊天
- 提示：
  - 在WebSocket服务器端，每个用户连接之后都创建一个connection对象。使用该对象向此用户发送数据。
  - 要传输包含几项内容的数据时，可以使用JSON进行传输（JSON和字符串可以互相转换）

THANKYOU