xNPE Lv2
2018年08月11日 阅读 3211

WebSocket的故事（一）—— WebSocket的由来

概述

微信小程序、小游戏的火爆，都让WebSocket的应用变得无处不在。针对这个主题，笔者打算做一个系列博客，旨在由浅入深的介绍WebSocket以及在Springboot和JS中如何快速构建和使用WebSocket提供的能力。

本系列计划包含如下几篇文章：

第一篇，什么是WebSocket以及它的用途。
第二篇，Spring中如何利用STOMP快速构建WebSocket广播式消息模式
第三篇，Springboot中，如何利用WebSocket和STOMP快速构建点对点的消息模式(1)
第四篇，Springboot中，如何利用WebSocket和STOMP快速构建点对点的消息模式(2)
第五篇，Springboot中，实现网页聊天室之自定义WebSocket消息代理
第六篇，Springboot中，实现更灵活的WebSocket

本篇的主线

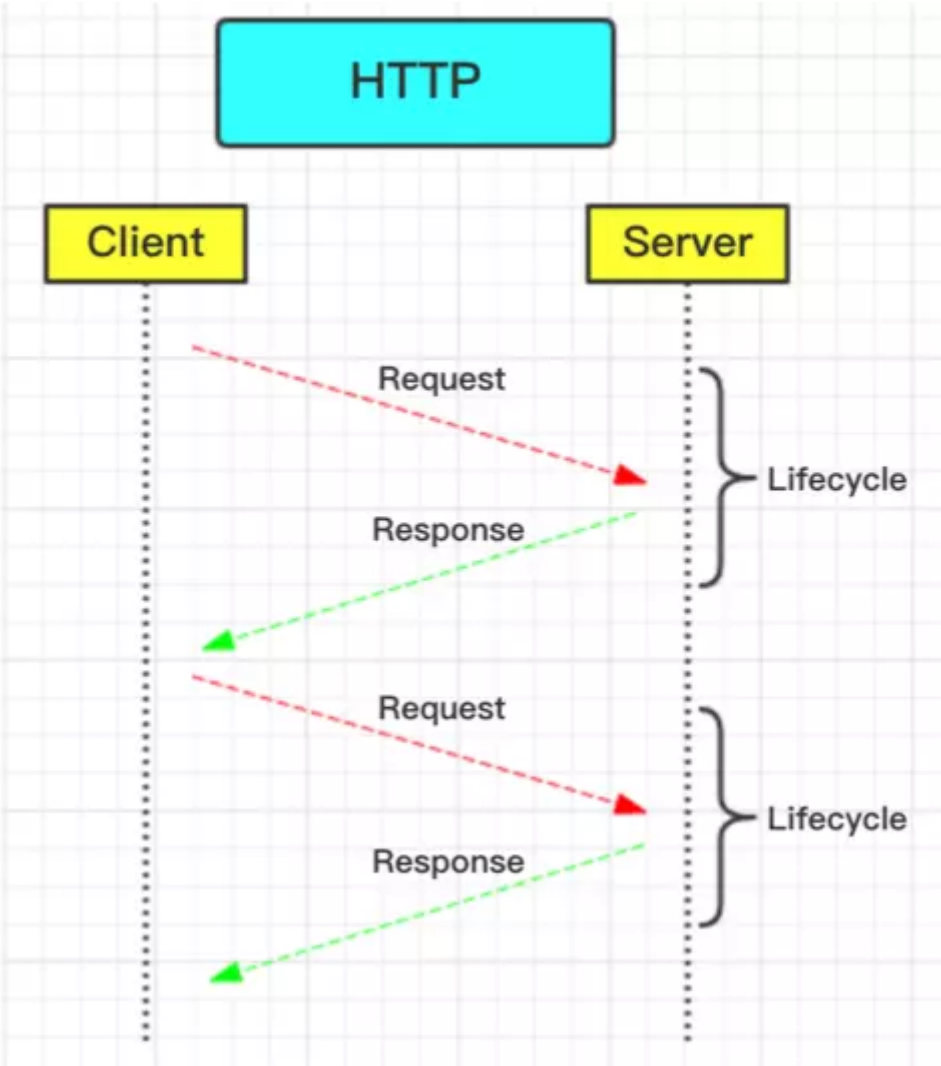
首先由一个典型场景引出WebSocket的需求场景，进而阐述WebSocket协议本身。包括其定义，特点以及握手过程报文的解读。最后，再次从**协议维度**和**实现长连接的方法**两个方面，对比了HTTP与WebSocket的异同，让读者对WebSocket有更深的认识和理解。

本篇适合的读者

为了照顾到刚接触前/后端开发的新手，作为系列的开篇文章，本着由浅入深的目的，本文采用了较为详尽的解读方式，老鸟亦欢迎收藏参考。后续篇章也会陆续更新上线，敬请期待。

由一个场景说起

那么这时查询软件与服务器交互如下图：



很容易理解，每一次航班动态查询，client都需要向server发起请求，然后等待server端的响应结果。当client收到响应后，本次通信的生命周期即宣告结束。

可是小铭说：**我希望只查询一次航班动态，当航班有更新时，服务器可以主动把最新的航班动态信息推送给我！**

怎么办？聪明的程序猿想到了如下的办法：

- **轮询（如ajax的轮询）方式**

即程序内部在小铭第一次请求时，记录下这个请求信息和响应信息，每隔固定时间（例如1分钟）一次服务器，服务器返回当前最新状态，对比之前收到的信息，如果相比有变更，则通知小铭；

> 客户端：啦啦啦，有没有新动态(Request)
> 服务端：正常起飞。。（Response）
> 客户端：有没有新动态(Request)
> 服务端：你好烦啊，正常起飞。。（Response）
> 客户端：有没有新动态（Request）
> 服务端：好啦好啦，有啦给你，延误30分钟。。（Response）
> 客户端：有没有新动态（Request）
> 服务端：没有。。。（Response）

- **服务端增加延迟答复(长连接)**

即程序内部依然采用轮询方式，不过比上一个方案相比，采取了阻塞方式。（一直打电话，没收到就不挂电话），也就是说，客户端发起连接后，如果服务端没消息，就一直不返回Response给客户端。直到有消息才通知小铭，之后客户端再次建立连接，周而复始。

> 客户端：有没有新动态，没有的话就等有了才返回给我吧（Request）
> 服务端：等到有动态的时候再告诉你。（过了一会儿）来了，给你，延误30分钟（Response）
> 客户端：有没有新动态，没有的话就等有了才返回给我吧（Request）

从整个交互的过程来看，这两种都是非常消耗资源的。

- 第一种方案,即轮询，需要服务器有很快的处理速度和处理器资源。（训练有素的接线员）
- 第二种方案，即HTTP长连接（后文还会介绍），需要有很高的并发，也就是说并行处理的能力。（足够多的接线员）

所以它们都有可能发生下面这种情况：

> 客户端：有新动态么？
> 服务端：问的人太多了，线路正忙，请稍后再试（503 Server Unavailable）
> 客户端：。。。。好吧，有新动态么？
> 服务端：问的人太多了，线路正忙，请稍后再试（503 Server Unavailable）
> 客户端：。。。。服务端你到底行不行啊。。!@#$%$^&

通过上面这个例子，总结一下我们可以看出，这两种采用HTTP的方式都不是最好的方式，体现在：

- **HTTP的无状态性**：由于接线员只管接电话和处理请求内容，并不会去记录是谁给他们打了电话，每次打电话，都要重新告诉一遍接线员你是谁和你的请求内容是什么。

那现在想要达到小铭的要求，该怎么办呢?

WebSocket的真身

说了这么半天了，让我们言归正传。基于上述的需求和矛盾，WebSocket出现了。

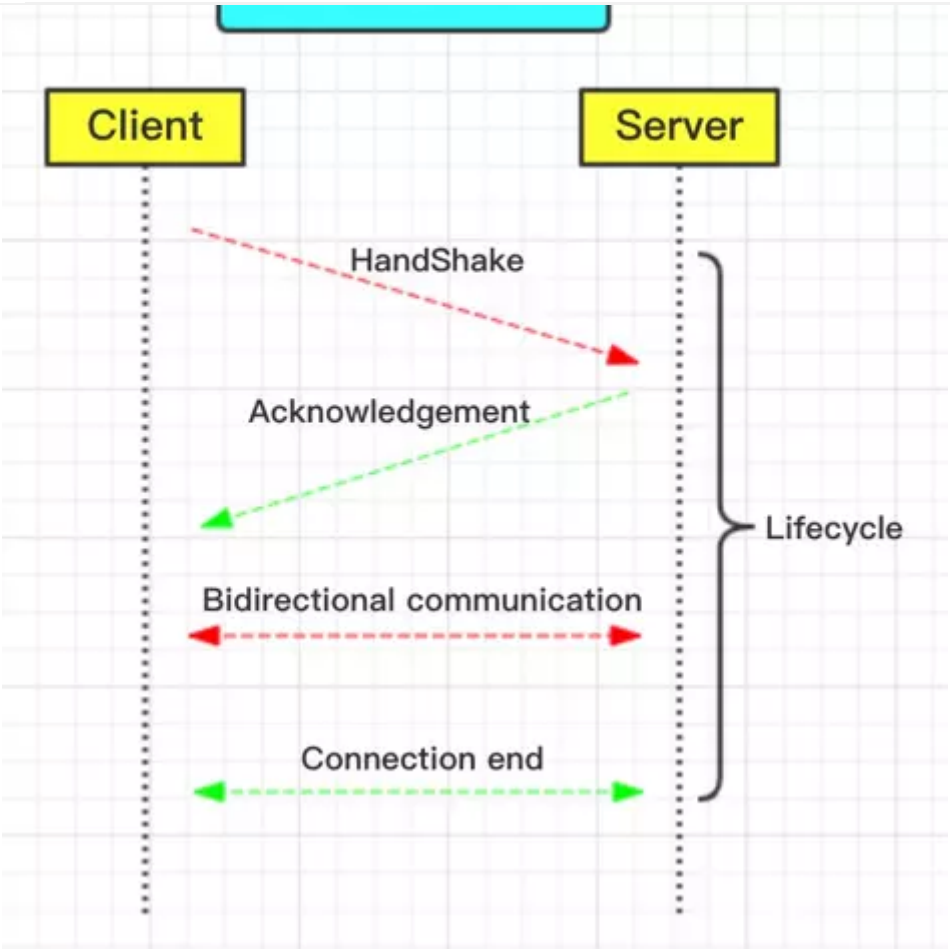
让我们先来看看，使用了WebSocket以后，上面的场景会变成怎样的流程：

> 客户端：我要开始使用WebSocket协议，需要的服务：chat(查动态)，WebSocket协议版本：13（HTTP Request）
> 服务端：没问题，已升级为WebSocket协议（HTTP Protocols Switched）
> 客户端：麻烦航班动态有更新的时候推送通知给我。
> 服务端：没问题。
> （.....过了10分钟）
> 服务端：有动态啦，延误30分钟！
> （.....过了30分钟）
> 服务端：有动态啦，现在开始登机！

由此可见，

- 当使用WebSocket时，服务端可以主动推送信息给客户端了，不必在意客户端等待了多久，不必担心超时断线，解决了被动性问题。
- Websocket只需要一次HTTP交互，来进行协议上的切换，整个通讯过程是建立在一次连接/状态中，也就避免了HTTP的无状态性，服务端会一直知道你的信息，直到你关闭请求，这样就解决了服务端要反复解析HTTP请求头的问题。

如下图所示：



WebSocket的出生

WebSocket是HTML5提出的一个协议规范（2011年）附上协议链接：

The WebSocket Protocol RFC6455

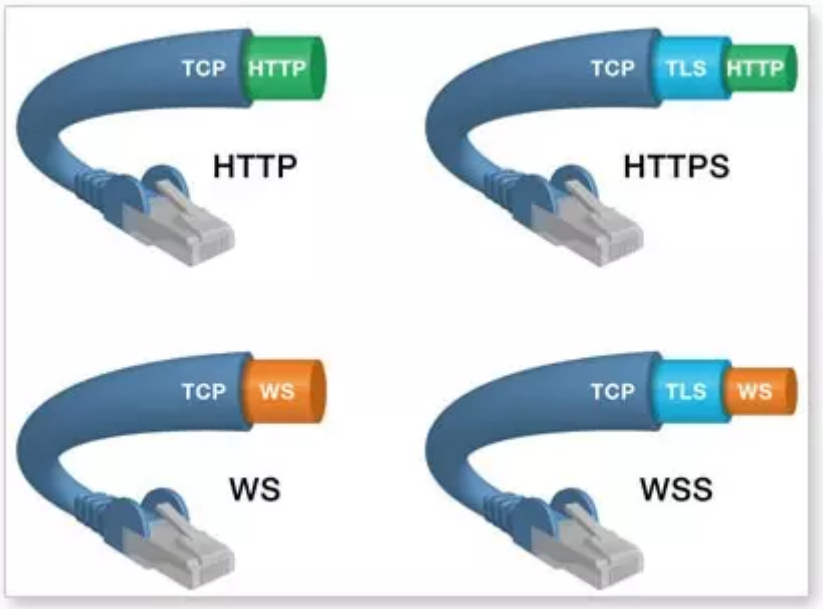
WebSocket约定了一个通信的规范，通过一个握手的机制，客户端（如浏览器）和服务器（WebServer）之间能建立一个类似Tcp的连接，从而方便C-S之间的通信。

WebSocket协议的特点

- 建立在 TCP 协议之上，它需要通过握手连接之后才能通信，服务器端的实现比较容易。
- 与 HTTP 协议有着良好的兼容性。默认端口也是80或443，并且握手阶段采用 HTTP 协议，因此握手时不容易屏蔽，能通过各种 HTTP 代理服务器。
- 数据格式比较轻量，性能开销小，通信高效。可以发送文本，也可以发送二进制数据。
- 没有同源限制，客户端可以与任意服务器通信。

- 它是一种双向通信协议，采用异步回调的方式接受消息，当建立通信连接，可以做到持久性的连接，WebSocket服务器和Browser都能主动的向对方发送或接收数据，实质的推送方式是服务器主动推送，只要有数据就推送到请求方。

用一张图来描述各个协议的关系：



WebSocket的通信建立——握手过程

WebSocket的握手使用HTTP来实现，客户端发送带有Upgrade头的HTTP Request消息。服务端根据请求，做Response。

请求报文：

```
GET wss://www.example.cn/webSocket HTTP/1.1
Host: www.example.cn
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Version: 13
Origin: http://example.cn
Sec-WebSocket-Key: afmbhhBRQuwCLmnWDRWHxw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits
```

详细解释一下：

- **第3行**：Connection：HTTP1.1中规定Upgrade只能应用在直接连接中。带有Upgrade头的HTTP1.1消息必须含有Connection头，因为Connection头的意义就是，任何接收到此消息的人（往往是代理服务器）都要在转发此消息之前处理掉Connection中指定的域（即不转发Upgrade域）。
- **第4行**：Upgrade是HTTP1.1中用于定义转换协议的header域。 如果服务器支持的话，客户端希望使用已经建立好的HTTP（TCP）连接，切换到WebSocket协议。
- **第5行**：Sec-WebSocket-Version标识了客户端支持的WebSocket协议的版本列表。
- **第6行**：Origin为安全使用，防止跨站攻击，浏览器一般会使用这个来标识原始域。
- **第7行**：Sec-WebSocket-Key是一个Base64encode的值，这个是客户端随机生成的，用于服务端的验证，服务器会使用此字段组装成另一个key值放在握手返回信息里发送客户端。
- **第8行**：Sec_WebSocket-Protocol是一个用户定义的字符串，用来区分同URL下，不同的服务所需要的协议，标识了客户端支持的子协议的列表。
- **第9行**：Sec-WebSocket-Extensions是客户端用来与服务端协商扩展协议的字段，permessage-deflate表示协商是否使用传输数据压缩，client_max_window_bits表示采用LZ77压缩算法时，滑动窗口相关的SIZE大小。

注：如果对压缩扩展协商的细节感兴趣，可参考下面的RFC7692了解更多细节。 Compression Extensions for WebSocket RFC7692

响应报文：

```
HTTP/1.1 101
Server: nginx/1.12.2
Date: Sat, 11 Aug 2018 13:21:27 GMT
Connection: upgrade
Upgrade: websocket
Sec-WebSocket-Accept: sLMyWetYOwus23qJyUD/fa1hztc=
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Extensions: permessage-deflate;client_max_window_bits=15
```

详细解释一下：

- **第1行**：HTTP的版本为HTTP1.1，返回码是101，开始解析Header域（不区分大小写）。
- **第2,3行**：服务器信息与时间。
- **第4行**：Connection字段，包含Upgrade。
- **第5行**：Upgrade字段，包含websocket。
- **第6行**：Sec-WebSocket-Accept字段，详细介绍一下：

> 1. 将Sec-WebSocket-Key与协议中已定义的一个GUID “258EAFA5-E914-47DA-95CA-C5AB0DC85B11”进行拼接。
> 2. 将步骤1中生成的字符串进行SHA1编码。
> 3. 将步骤2中生成的字符串进行Base64编码。

> 客户端通过验证服务端返回的Sec-WebSocket-Accept的值, 来确定两件事情:

> 1. 服务端是否理解WebSocket协议, 如果服务端不理解,那么它就不会返回正确的Sec-WebSocket-Accept，则建立WebSocket连接失败。
> 2. 服务端返回的Response是对于客户端的此次请求的,而不是之前的缓存。 主要是防止有些缓存服务器返回缓存的Response.

- **第7行**：Sec-WebSocket-Protocol字段，要判断是否之前的Request握手带有此协议，如果没有，则连接失败。
- **第8行**：扩展协议协商，支持压缩，且LZZ的滑动窗口大小为15。

至此，握手过程就完成了，此时的TCP连接不会释放。客户端和服务端可以互相通信了。

HTTP1.1与WebSocket的异同

最后，作为总结，让我们再来回顾一下HTTP1.1与WebSocket的相同与不同。加深对WebSocket的理解。

协议层面的异同

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WebSocket可能返回和HTTP相同的返回码。

不同点

- WebSocket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。换言之，二者的请求头不同。
- WebSocket的连接不能通过中间人来转发，它必须是一个直接连接。如果通过代理转发，一个代理要承受如此多的WebSocket连接不释放，就类似于一次DDOS攻击了。
- WebSocket在建立握手连接时，数据是通过HTTP协议传输的，但在建立连接之后，真正的数据传输阶段是不需要HTTP协议参与的。
- WebSocket传输的数据是二进制流，是以帧为单位的，HTTP传输的是明文传输，是字符串传输，WebSocket的数据帧有序。

HTTP的长连接与WebSocket的持久连接的异同

HTTP的两种长连接

一、HTTP1.1的连接默认使用长连接（Persistent connection）

即在一定的期限内保持链接，客户端会需要在短时间内向服务端请求大量的资源，保持TCP连接不断开。客户端与服务器通信，必须要有客户端发起然后服务器返回结果。客户端是主动的，服务器是被动的。在一个TCP连接上可以传输多个Request/Response消息对，所以本质上还是Request/Response消息对，仍然会造成资源的浪费、实时性不强等问题。如果不是持续连接，即短连接，那么每个资源都要建立一个新的连接，HTTP底层使用的是TCP，那么每次都要使用三次握手建立TCP连接，即每一个request对应一个response，将造成极大的资源浪费。

二、“长轮询”

即客户端发送一个**超时时间很长**的Request，服务器保持住这个连接，在有新数据到达时返回Response

WebSocket的持久连接

只需建立一次Request/Response消息对，之后都是TCP连接，避免了需要多次建立Request/Response消息对而产生的冗余头部信息。节省了大量流量和服务器资源。因此被广泛应用于线上WEB游戏和线上聊天室的开发。

下一篇内容前瞻

下一篇中，笔者将使用JS（前端）和Springboot（后端），详细介绍如何利用Springboot框架，构建一个基于STOMP的简单WebSocket通信系统。敬请关注。

欢迎关注xNPE技术论坛，更多原创干货每日推送。

关注下面的标签，发现更多相似文章

后端 Spring WebSocket

xNPE Lv2
非典型程序员 @ 自由开发者
获得点赞 452 · 获得阅读 23,327
关注

评论

输入评论...

你为什么不讲道理

博主，你好，写的真的很棒，我想把你的内容记录下来，防止以后忘记，我可以转载你这的关于WebSocket文章吗？我会标注转载地址，谢谢了

2月前 回复

森酱可乐 前端

赞

2月前 回复

恨伊不似红萼 IT

丿Corn Java

赞

1年前 回复

Baidu

好

1年前 1 回复

相关推荐

专栏 · 叁公子KCN · 12分钟前 · 后端 / Python

图像搜索：给你爬的美女图建一个搜索引擎

专栏 · 江五渣 · 12小时前 · Go / 后端

聊聊 Go 语言中的字符表示与字符串遍历

7 7

专栏 · 腾讯IVWEB团队 · 2天前 · 后端 / Serverless

「NGW」前端新技术赛场：Serverless SSR 技术内幕

43 3

专栏 · CoderZS · 2天前 · 后端

美丽的一致性Hash算法

30 1

专栏 · 码农小胖哥 · 2天前 · Docker / 后端

开源模式面临的生存危机

12 3

专栏 · shanyue · 4天前 · Node.js / WebSocket

关于一个 websocket 多节点分布式问题的头条前端面试题

47 4

专栏 · 漫话编程 · 27天前 · 后端 / Java

业务复杂=if else？刚来的大神竟然用策略+工厂彻底干掉了他们！

专栏 · 何甜甜在吗 · 10天前 · 后端 / Java

为什么阿里巴巴要禁用Executors创建线程池?

83 19

专栏 · 小姐姐味道 · 17天前 · 前端 / 后端

运营商劫持狠起来，连json都改

145 32