基于 HTTP 协议的 3 种实时数据获取技术

五月的仓颉 芋道源码 2019-04-16

HTTP协议

HTTP协议大家都很熟悉了，开始本文之前，首先简单回顾一下HTTP协议。

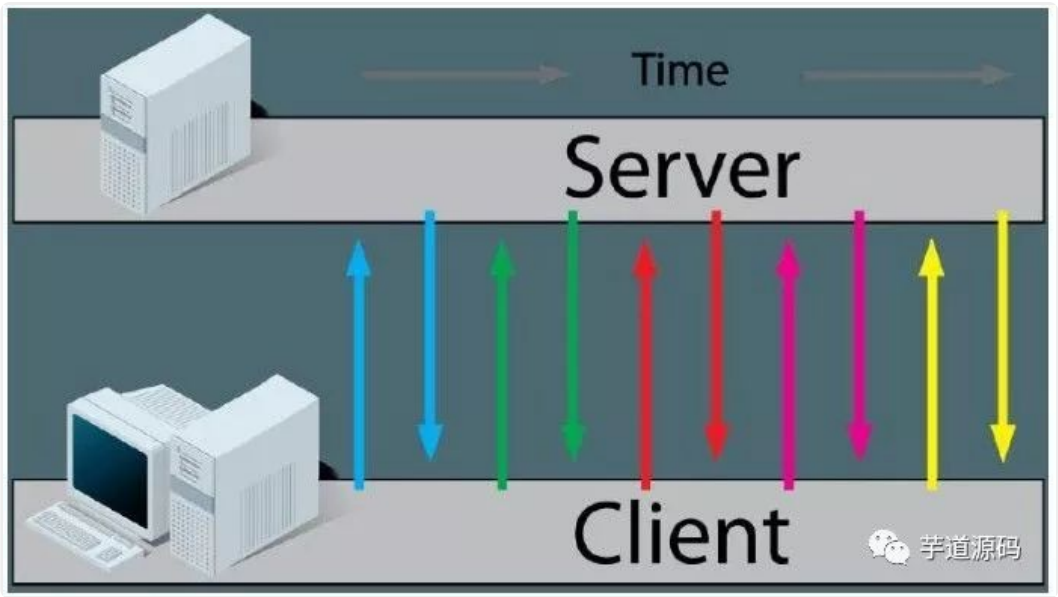
HTTP协议是建立在TCP协议上的应用层协议，协议的本质是请求----应答：

即对于HTTP协议来说，服务端给一次响应后整个请求就结束了，这是HTTP请求最大的特点，也是由于这个特点，HTTP请求无法做到的是服务端向客户端主动推送数据。

但由于HTTP协议的广泛应用，很多时候确实又想使用HTTP协议去实现实时的数据获取，这种时候应当怎么办呢？下面首先介绍几种基于HTTP协议的实时数据获取方法。

方式一：短轮询

轮询是最普遍的基于HTTP协议获取实时数据的方式，轮询又分为短轮询和长轮询。短轮询非常简单，用一张图表示一下：

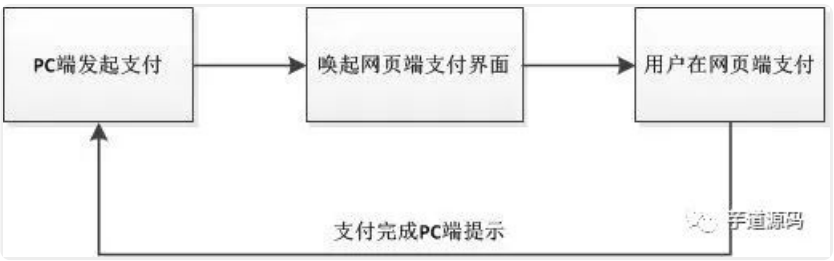


客户端向服务端请求数据，服务端立即将数据返回给客户端，客户端没有拿到想要的数据（比如返回结果告诉客户端，数据处理中），客户端继续发请求，服务端继续立即响应，周而复始。

这种实时数据获取的方式比较粗暴，优点在于编程简单，客户端发请求，服务端实时回响应即可。缺点主要有两个：

- 无效请求多，每一次无效请求都在浪费带宽和服务器的计算资源
- 对服务器压力大，定时发请求，并发一高，可能服务端瞬间会收到成千上万个请求，很容易拖垮服务器甚至导致宕机

那么短轮询适合哪种使用场景呢，按照我的理解如果数据变化比较频繁或者能预期到数据在短时间内会发生一次变化的场景可以使用短轮询，比如：

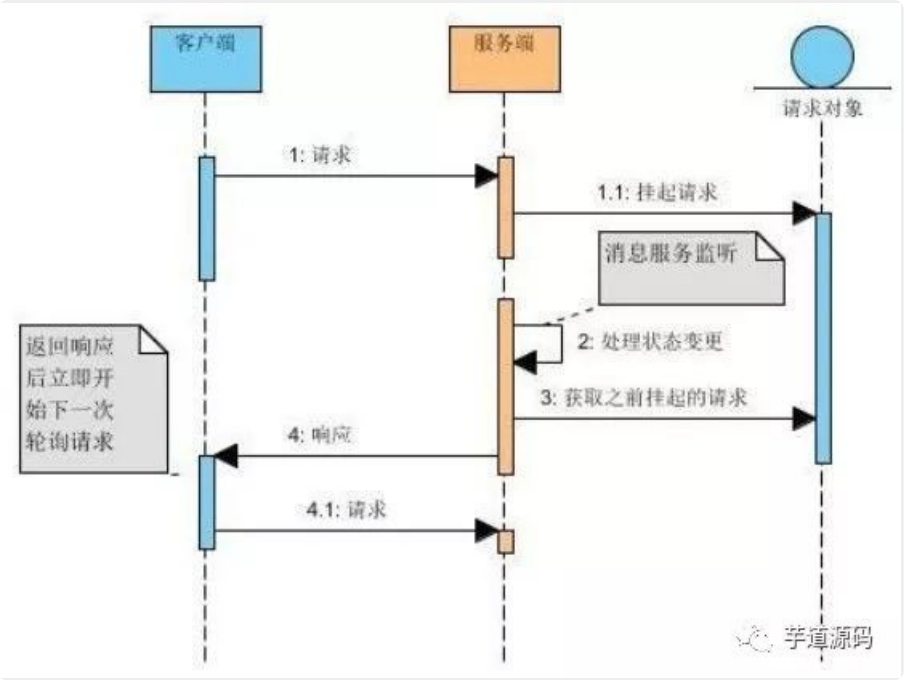


用户在PC端买了一个东西唤起网页端，由于PC端和网页端是不通的，我们预期到用户应该很快会完成付款，这种时候为了开发简单短轮询是一种可以使用的方式，直接服务端提供一个接口告诉客户端订单状态，客户端每5秒请求一次即可，拿到结果就可以不用请求了。

使用短轮询注意要做好请求次数上限的控制，比如请求100次还没检测到用户付款，可以弹窗"请完成付款后去我的订单页面查询"就可以不用请求了。

方式二：长轮询

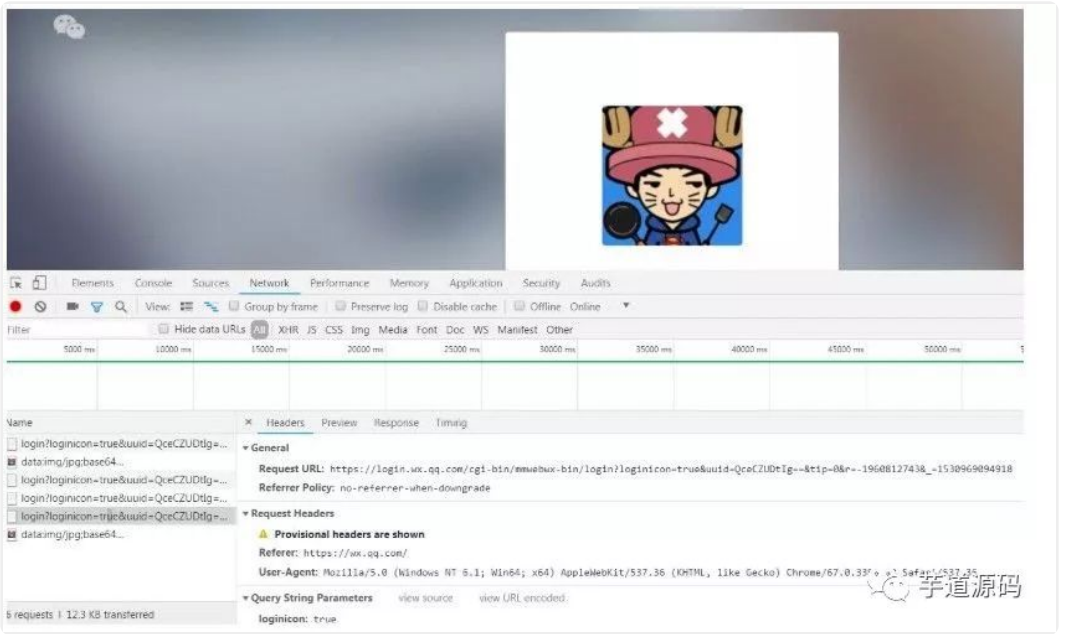
长轮询是另一种实时获取数据的方式，看一下流程：



本质上没有改变，依然是客户端在没有收到自己想要数据的情况下不断发送请求给服务端，差别在于服务端收到请求不再直接给响应，而是将请求挂起，自己去定时判断数据的变化，有变化就立马返回给客户端，没有就等到超时为止。

可以很明显的看到，长轮询的优点就是客户端的请求少了很多避免了无谓的客户端请求，缺点则是服务端会挂起大量请求增加资源消耗且服务器对HTTP请求并发数量是有限制的。

微信网页版的登陆是一个典型的长轮询的例子：



从图上看，客户端不断发送请求到服务器，服务器第一时间并没有给出回应，于是客户端等待，在超时的情况下继续发送请求。

总的来说我理解一般使用长轮询会更多一点，短轮询更加看重的是编程简单，适合小型应用。像微信网页端登录这种，成千上万个用户同时登陆，隔一段时间服务端收成千上个请求去处理哪里受得了，堆机器分摊每台服务器上处理请求的数量终究不是解决问题的办法。

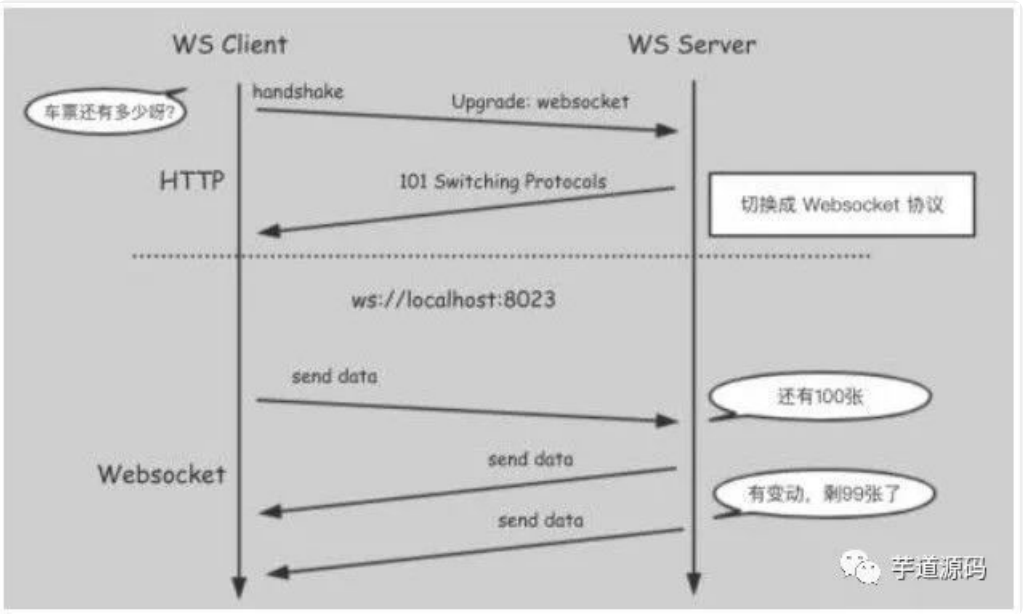
方式三：WebSocket

上面介绍了两种轮询方式，但是两种综合起来都有比较明显的缺点，总结起来有以下几个：

- 伪实时，即上述两种方式都不是真正的实时，无论短轮询的客户端轮询时间多短，还是长轮询的服务端轮询时间多短，都存在一定程度的延时
- 所有的轮询只要没有需要的数据返回，都是对计算资源的一种浪费
- HTTP协议本身是一个重的协议，每一次都必须带有HTTP首部+HTTP头部，实际上对我们来说需要的只是HTTP Body而已，多余的数据都是对带宽的一种浪费

因此，最好我们可以做到的事情是：客户端和服务端之间有一条通路，当服务端数据有变化的时候，服务端可以主动推送到客户端。WebSocket就是HTML5之后为了做到这一点而诞生的一种协议，虽然这是一种新的协议，但也是基于HTTP协议的。

看一下WebSocket的原理，很简单：

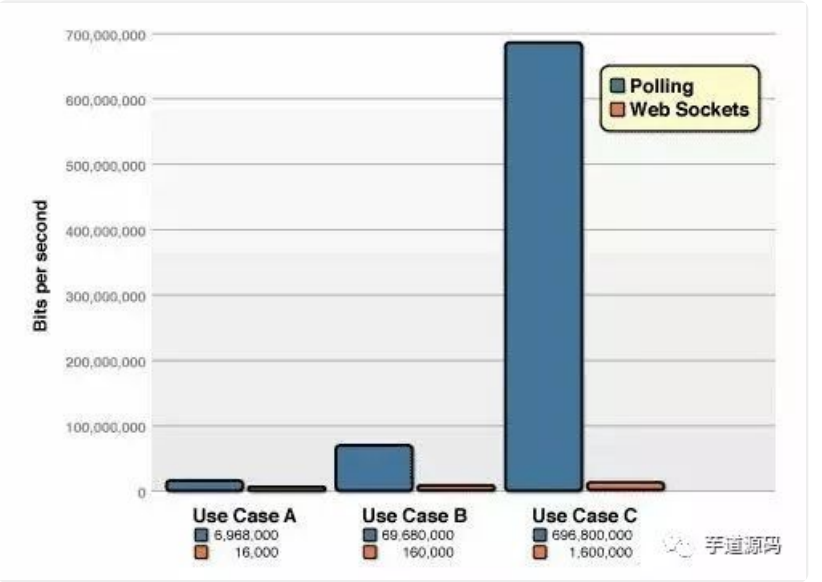


WebSocket客户端首先通过HTTP协议发送几个特别的header到服务端，告诉服务端现在我发起的是HTTP请求，但我要升级到WebSocket了：

- Upgrade:websocket
- Connection：Upgrade
- Sec-WebSocket-Key: XXX
- Sec-WebSocket-Protocol: chat, superchat
- Sec-WebSocket-Version: XX

只要服务器支持WebSocket协议（Tomcat7、Jetty7之后都是支持WebSocket的），那么服务端收到请求且建立连接成功后会返回Sec-WebSocket-Accept、Sec-WebSocket-Protocol这两个header给客户端，且Http Status为101表示协议切换成功，这样客户端和服务端只要任意一方没有断开连接，就可以基于这一条通路进行通讯了。

再谈一下之前提的WebSocket相比长短轮询对于带宽资源的节省。有一个测试，假设HTTP Header是871字节，WebSocket由于数据传输是基于帧的，帧传输更加高效，对比长短轮询，2个字节即可代替871个字节的Header，测试结果为：

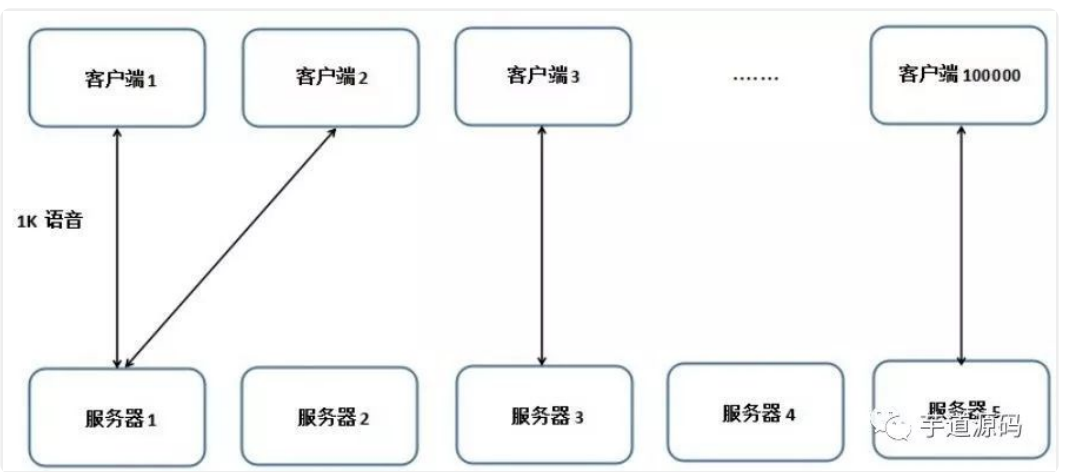


相同的每秒客户端轮询的次数，当次数高达10W/s的高频率次数的时候，轮询需要消耗665Mbps，而WebSocket仅仅只花费了1.526Mbps，将近435倍。

WebSocket做到了真正的实时且大量节省带宽资源，但是我理解也有自己的问题，就是开发成本比较高，这里的开发成本倒不是说自己去实现WebSocket，这个在Java语言层面上直接使用Netty-Socketio即可，API很简单，提供了对WebSocket完整的实现，真正的开发成本在于分布式环境下的数据同步问题。

举个例子，有一个在线聊天系统10W人同时在线，此时有一个用户发了一条1K的语音消息，单机保持10W的连接倒是可以（这里不是HTTP请求，因此不受连接池数影响），问题在于带宽。单机同时向10W用户推送1K语音消息，需要的带宽至少10M，这还只是纯粹推送数据出去，没有考虑到数据进来的场景，实际运行过程中需要的带宽会更多，对于企业来说这是一笔非常大的成本。

因此，大量连接的场景下都会做集群（实际就算没有大量连接，为了高可用性，也会做集群），10W并发分出5台机器，平均每台机器有2W连接，考虑集群下会出现的问题：



客户端1把数据发送到服务器1，服务器1连接的所有客户端都可以推送该条语音，但是问题在于：

- 服务器2~服务器5连的所有客户端如何拿到数据？简单的一种方式是使用消息队列，将数据通过消息队列发送到所有订阅的服务器上
- 那如果传输的是一张1M的图片，数据太大不适合使用消息队列怎么办，可以先将数据存储下来，消息队列只发送id，收到消息的服务器再根据id去取真正的数据并推送
- 如果依赖消息队列，那么不仅仅需要对应用进行代码开发，还需要对消息服务器做分布式集群、做压力测试，保证高可用
- 2W连接正常预计发送1K的消息是没问题的，但是万一用户发送了1M图片导致远超预估带宽怎么办，是业务上取舍不能发送超过XXX的数据还是技术上处理

其他太多需要考虑的问题没有列出来，总而言之，用WebSocket在大量请求、高并发的场景下，代码开发成本是非常高的。但是由于WebSocket可以做到真正的实时服务端对客户端的数据推送且对带宽资源有大量的节省，因此很多IM、音视频、弹幕等应用都会使用WebSocket。

欢迎加入我的知识星球，一起探讨架构，

已在知识星球更新源码解析如下：

《精尽面试题（附答案）》	《精尽学习指南（附视频）》
01. Dubbo 面试题	00. 精尽学习指南 —— 路线
02. Netty 面试题	01. Dubbo 学习指南
03. Spring 面试题	02. Netty 学习指南
04. Spring MVC 面试题	03. Spring 学习指南
05. Spring Boot 面试题	04. Spring MVC 学习指南
06. Spring Cloud 面试题	05. Spring Boot 学习指南
07. MyBatis 面试题	06. Spring Cloud 学习指南
08. 消息队列面试题	07. MyBatis 学习指南
09. RocketMQ 面试题	08. RocketMQ 学习指南
10. RabbitMQ 面试题	09. RabbitMQ 学习指南
11. Kafka 面试题	10. Kafka 学习指南
12. 缓存面试题	11. Redis 学习指南
13. Redis 面试题	12. MySQL 学习指南
14. MySQL 面试题	13. MongoDB 学习指南
15.【分库分表】面试题	14. Elasticsearch 学习指南
16.【分布式事务】面试题	15. 设计模式学习指南
17. Elasticsearch 面试题	16. Java【基础】学习指南
18. MongoDB 面试题	17. Java【并发】学习指南
19. 设计模式面试题	18. Java【虚拟机】学习指南
20. Java【基础】面试题	19. Linux 学习指南
21. Java【集合】面试题	20. 数据结构与算法学习指南
22. Java【并发】面试题	21. 计算机网络学习指南
23. Java【虚拟机】面试题	22. Maven 学习指南
24. Linux 面试题	23. Jenkins 学习指南
25. Git 面试题	24. Git 学习指南
26. 计算机网络面试题	25. Intellij IDEA 学习指南
27. Maven 面试题	26. Docker 学习指南
28. Jenkins 面试题	27. Kubernetes 学习指南
29. Zookeeper 面试题	28. Zookeeper 学习指南
30. Nginx 面试题	29. Nginx 学习指南

《Dubbo 源码解析》

01. 调试环境搭建
02. 项目结构一览
03. 配置 Configuration
04. 核心流程一览
05. 拓展机制 SPI
06. 线程池 ThreadPool
07. 服务暴露 Export
08. 服务引用 Refer
09. 注册中心 Registry
10. 动态编译 Compile
11. 动态代理 Proxy
12. 服务调用 Invoke
13. 调用特性
14. 过滤器 Filter
15. NIO 服务器
16. P2P 服务器
17. HTTP 服务器
18. 序列化 Serialization
19. 集群容错 Cluster
20. 优雅停机 Shutdown
21. 日志适配 Logging
22. 状态检查 Status
23. 监控中心 Monitor
24. 管理中心 Admin
25. 运维命令 QOS
26. 链路追踪 Tracing
27. Spring Boot 集成
28. Spring Cloud 集成

... 一共 **73+** 篇

《Spring Cloud 源码解析》

01. 网关 Spring Cloud Gateway 25 篇
02. 注册中心 Eureka 23 篇
03. 熔断器 Hystrix 9 篇
04. 配置中心 Apollo 32 篇
05. 链路追踪 SkyWalking 38 篇
06. 调度中心 Elastic Job 24 篇

《Netty 源码解析》

01. 调试环境搭建
02. NIO 基础
03. Netty 简介
04. 启动 Bootstrap
05. 事件轮询 EventLoop
06. 通道管道 ChannelPipeline
07. 通道 Channel
08. 字节缓冲区 ByteBuf
09. 通道处理器 ChannelHandler
10. 编解码 Codec
11. 工具类 Util

... 一共 **61+** 篇

《MyBatis 源码解析》

01. 调试环境搭建
02. 项目结构一览
03. MyBatis 初始化
04. SQL 初始化
05. SQL 执行
06. 插件体系
07. Spring 集成

... 一共 **34+** 篇

《Spring 源码解析》

01. 调试环境搭建
02. IoC Resource 定位
03. IoC BeanDefinition 载入
04. IoC BeanDefinition 注册
05. IoC Bean 获取
06. IoC Bean 生命周期
07. AOP 源码导读
08. Transaction 源码导读

... 一共 **46+** 篇

《Spring Boot 源码解析》

01. 调试环境搭建
02. 项目结构一览
03. SpringApplication
04. 自动配置
05. Condition
06. ServletWebServerApplicationContext
07. ReactiveWebServerApplicationContext
08. ApplicationContextInitializer
09. ApplicationListener

... 一共 **15+** 篇（**努力更新中**）

《Spring MVC 源码解析》

01. 调试环境搭建
02. 容器的初始化
03. 组件一览
04. 请求处理一览
05. HandlerMapping 组件
06. HandlerAdapter 组件
07. HandlerExceptionResolver 组件
08. RequestToViewNameTranslator 组件
09. LocaleResolver 组件
10. ThemeResolver 组件
11. ViewResolver 组件
12. MultipartResolver 组件
13. FlashMapManager 组件

... 一共 **24+** 篇

《数据库实体设计》

01. 商品模块
02. 交易模块
03. 营销模块
04. 公用模块

... 一共 **17+** 篇

如果你喜欢这篇文章，喜欢，转发。
生活很美好，明天见(。·ω·。)/♡

阅读全文