WebSocket和Stomp协议

JimmyOu 关注
0.378 2019.03.15 15:57:45 字数 3,639 阅读 3,185

1. webSocket介绍

我们知道http协议是无状态协议，每次请求都只能由客户端发起，服务器进行响应。但是服务器不能主动发送消息给客户端。这种单向的协议在很多的业务场景中不适用，比如消息推送，实时消息详情等。在使用websocket前，我们通常可以使用轮询或者长链接来实现这种实时消息的需求。

1.1. 轮询

由客户端或者浏览器定时发request，然后服务器返回最新的数据给客户端。缺点很明显，浏览器需要不断向服务器发送请求，然而http的request的header非常长，但是实际需要的业务数据却是一个很小的值。需要消耗很多服务器资源和带宽资源。

JimmyOu 关注
总资产3 (约0.37元)

WebSocket和Stomp协议
阅读 3,183

gif展示性能研究
阅读 101

推荐阅读

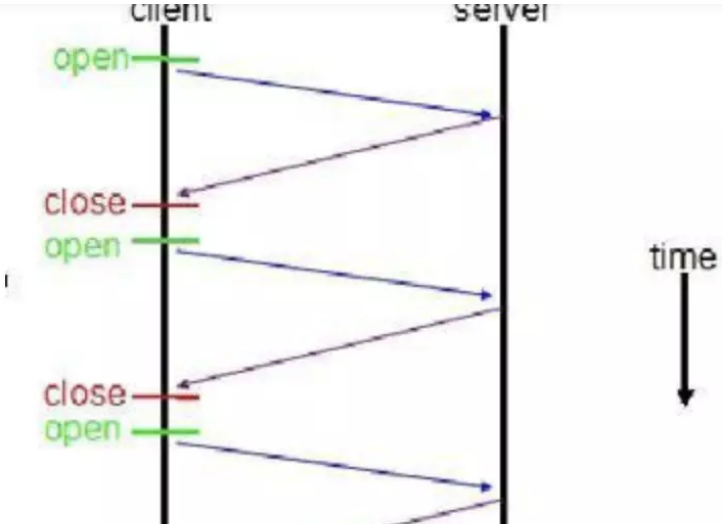
最近央视好像混进了奇怪的东西（截图已传疯）
阅读 15,130

20年前最贵的香港电影，全都给了郑伊健?
阅读 2,959

中国夫妻真实生活曝光，我想起了王家卫说的那句话
阅读 6,410

“最懂事的人,手术都一个人做”
阅读 4,575

“我妈昨晚走了，到死也没见过大海”
阅读 5,743



01

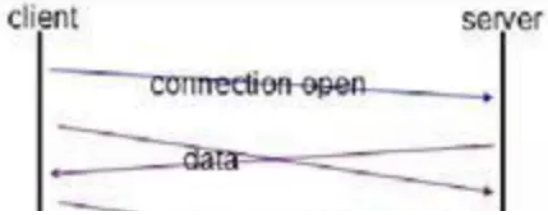
1.2. 长链接

http 1.1 默认保持长链接，数据传输完成保持tcp链接不断开，等待在相同域名下继续使用这个通道传输数据。客户端的长链接不能无限期的拿着，会有一个超时时间，服务器有时候会告诉客户端超时时间，下图中的Keep-Alive: timeout=20，表示这个TCP通道可以保持20秒。另外还可能有max=XXX，表示这个长连接最多接收XXX次请求就断开。对于客户端来说，如果服务器没有告诉客户端超时时间也没关系，服务端可能主动发起四次握手断开TCP连接，客户端能够知道该TCP连接已经无效；另外TCP还有心跳包来检测当前连接是否还活着。

02

1.3. websocket

Websocket是html5提出的一个协议规范，是为解决客户端与服务端实时通信。本质上是一个基于tcp，先通过HTTP/HTTPS协议发起一条特殊的http请求进行握手后创建一个用于交换数据的TCP连接。只需要要做一个握手的动作，在建立连接之后，双方可以在任意时刻，相互推送信息。同时，服务器与客户端之间交换的头信息很小。



03

看起来websocket和http的长链接都可以应用于实时消息的场景，但是http的长链接对每个请求仍然要单独发 header，Keep-Alive不会永久保持连接，它有一个保持时间，可以在不同的服务器软件（如Apache，默认为15s）中设定这个时间。keep-alive双方并没有建立正真的连接会话，服务端可以在任何一次请求完成后关闭。而WebSocket 它本身就规定了是真正的、双工的长连接，两边都必须要维持住连接的状态。

通过服务端和客户端的交互报文来对比webSocket通讯和传统的http的不同：

在客户端，实例化一个新的WebSocket客户端对象，请求类似的 `wss:yourdomain:port/path` 的webSocket url。客户端的webScoket对象会自动解析并识别为WebSocket请求，并链接服务端口，执行双方握手过程，客户端发送数据格式类似：

```
GET /webfin/websocket/ HTTP/1.1
Host: localhost
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: xqBt3ImNzJbYqRINxEFlkg==
Origin: http://localhost:8080
Sec-WebSocket-Version: 13
```

客户端发起的websocket链接报文类似于传统的http报文， `Upgrade: websocket` 表明这是一个websocket请求， `Sec-WebSocket-Key: xqBt3ImNzJbYqRINxEFlkg==` 是客户端发送的一个base64编码秘文

服务端收到报文后返回类似的数据格式：

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: K7DJLdLooIwIG/MOpvWFB3y3FE8=
```

两端的WebSocket连接握手成功, 后续就可以进行TCP通讯了.查阅WebSocket协议栈了解WebSocket客户端和服务端更详细的交互数据格式。

`PingPong` ： 一个关于维持链接的websocket设计技术细节

虽然说webSocket解决了服务器和客户端的链接问题，但是网络应用除了客户端和服务器还存在中间的网络链路，一个http/websocket链接往往还需要进过无数的路由，防火墙，在这个过程中，中间节点的处理方法可能会让人想不到，这些坑爹的中间节点可能会认为一份连接在一段时间内没有数据发送就等于失效，它们会自作主张的切断这些连接。在这种情况下，不论服务器还是客户端都不会收到任何提示，它们只会一厢情愿的以为彼此间的红线还在，徒劳地一边又一边地发送抵达不了彼岸的信息。而计算机网络协议栈的实现中又会有一层套一层的缓存，除非填满这些缓存，你的程序根本不会发现任何错误。

解决方案:

总结：在实时消息的应用场景下，比起轮询，长链接等方案，webSocket确实给我们提供了一个比较完美的解决方案。

2.STOMP传输协议介绍

`STOMP` 中文为: 面向消息的简单文本协议

`websocket` 定义了两种传输信息类型:**文本信息和二进制信息**。类型虽然被确定，但是他们的传输体是没有规定的。所以，需要用一种简单的文本传输类型来规定传输内容，它可以作为通讯中的文本传输协议。

STOMP是基于帧的协议，客户端和服务器使用STOMP帧流通讯

一个STOMP客户端是一个可以以两种模式运行的用户代理，可能是同时运行两种模式。

- 作为生产者，通过 `SEND` 框架将消息发送给服务器的某个服务
- 作为消费者，通过 `SUBSCRIBE` 制定一个目标服务，通过 `MESSAGE` 框架，从服务器接收消息。

例如:

```
COMMAND
header1:value1
header2:value2

Body^@
```

注：帧以commnand字符串开始，以EOL结束。其中包括可选回车符（13字节），紧接着是换行符（10字节）。command下面是0个或多个 `<key>:<value>` 格式的header条目, 每个条目由EOL结束。一个空白行（即额外EOL）表示header结束和body开始。body连接着NULL字节。本文档中的例子将使用 `^@` 代表NULL字节。NULL字节可以选择跟多个EOLs。欲了解更多关于STOMP帧的详细信息，请参阅STOMP1.2协议规范。

2.1 STOMP 1.2 协议

`STOMP 1.2` clients 必须设置以下headers:

1. `accept-version:` clients支持的STOMP的版本号。

2. `host:` client希望连接的虚拟主机名字

可选择设置以下headers:

1. `login:` 用于在server验证的用户id

2. `passcode:` 用于在server验证的密码

3. `heart-beat:` 心跳设置

写下你的评论... 评论1 赞3 ···

- CONNECT
- CONNECTED
- SEND
- SUBSRIBE
- UNSUBSRIBE
- BEGIN
- COMMIT
- ABORT
- ACK
- NACK
- DISCONNECT

2.2.1 CONNECT

STOMP客户端通过初始化一个数据流或者TCP链接发送CONNECT帧到服务端，例如：

```
CONNECT

accept-version:1.2
host:stomp.test

^@
```

2.2.2 CONNECTED

如果服务端接收了链接意图，它回回复一个CONNECTED帧：

```
CONNECTED

version:1.2

^@
```

正常链接后客户端和服务端就可以正常收发信息了。

2.2.3 SEND

客户端主动发送消息到服务器，例如：

```
SEND
destination:/queue/a
content-type:text/plain

I am send body
^@
```

注： 必须包含 `destination` 目标地址，如果没有 `content-type` ,默认表示传递的二进制.

2.2.4 SUBSCRIBE

客户端注册给定的目的地，被订阅的目的地收到的任何消息将通过 `MESSAGE` Frame发送给client。
`ACK` 控制着确认模式。

```
SUBSCRIBE
id:0
destination:/queue/foo
```

id: 一个单连接可以对应多个开放的servers订阅,这个id用来客户端和服务端处理与订阅消息和取消订阅相关的动作。

ack: 可用的值有 auto , client , client-individual , 默认为 auto .

> 当 ack 为 auto 时，client收到server发来的消息后不需要回复 ACK帧 .server假定消息发出去后client就已经收到。这种模式下可能导致服务端向客户端发送的消息丢失

> 当 ack 为 client 时, 客户端收到服务端信息之后必须回复 ACK帧 。如果在收到客户端回复的ACK之前连接断开，服务端会认为这个消息没有被处理而改发给其他客户端。客户端回复的 ACK 会被当做累加的处理。这意味着对信息的确认操作不仅仅是确认了这单个的消息，还确认了这个订阅之前发送的所有消息（即接收到一个确认消息就会把之前的消息一起确认掉，批量操作）。

> 由于client不能处理某些消息，所以client应该发送 NACK帧 去告诉server它不能消费这些消息。

> 当 ack 模式是 client-individual ，确认操作就跟client模式一样，除了 ACK 和 NACK 不是累加的。这意味着当后来的一个消息得到 ACK 或 NACK 之后，之前的那个消息没有被 ACK 或 NACK ，它需要单独的确认。

2.2.5 UNSUBSRIBE

UNSUBSCRIBE用来移除一个已经存在订阅，一旦一个订阅被从连接中取消，那么客户端就再也不会收到来自这个订阅的消息。

```
UNSUBSCRIBE

id:0

^@
```

由于一个连接可以添加多个服务端的订阅，所以id头是 UNSUBSCRIBE 必须包含的，用来唯一标示要取消的是哪一个订阅。 id 的值必须是一个已经存在的订阅的标识。

2.2.6 ACK

ACK是用来在 client 和 client-individual 模式下确认已经收到一个订阅消息的操作。在上述模式下任何订阅消息都被认为是没有被处理的，除非客户端通过回复ACK确认。

```
ACK

id:12345

transaction:tx1

^@
```

2.2.7 NACK

NACK是ACK的反向，它告诉服务端客户端没有处理该消息。服务端可以选择性的处理该消息，重新发送到另一个客户端或者丢弃它或者把他放到无效消息队列中记录。

`NACK` 包含和 `ACK` 相同的头信息： `id` （必须）和 `transaction` （非必须）。

2.2.8 BEGIN

BEGIN用于开启一个事务- `transaction` 。这种情况下的事务适用于发送消息和确认已经收到的消息。在一个事务期间，任何发送和确认的动作都会被当做事务的一个原子操作。

```
BEGIN

transaction:tx1

^@
```

帧中 `transaction` 头是必须的，并且 `transaction` 的标示会被用在 `SEND` 、 `COMMIT` 、 `ABORT` 、 `ACK` 和 `NACK` 中，使之与该事务绑定。同一个链接中的不同事务必须使用不同的标示。
当客户端发送一个 `DISCONNECT` 或者 `TCP` 链接由于任何原因断开时，任何打开的但是还没有被提交的事务都会被默认的立即中断。

2.2.9 COMMIT

用来提交一个事务到处理队列中,帧中的 `transaction` 头是必须得，用以标示是哪个事务被提交。

```
COMMIT

transaction:tx1

^@
```

2.2.10 ABORT

`ABORT` 用于中止正在执行的事务,帧中的 `transaction` 头是必须得，用以标示是哪个事务被终止。

```
ABORT

transaction:tx1

^@
```

2.2.11 DISCONNECT

客户端可以通过 `DISCONNECT` 帧表示正常断开链接

2.2.12 MESSAGE

`MESSAGE` 用于传输从服务端订阅的消息到客户端。

`MESSAGE` 中必须包含 `destionation` 头，用以表示这个消息应该发送的目标。如果这个消息被使用STOMP发送，那么这个 `destionation` 应该与相应的 `SEND帧` 中的目标一样。

`MESSAGE` 中必须包含 `message-id` 头，用来唯一表示发送的是哪一个消息，以及 `subscription` 头用来

MESSAGE 如果有body内容，则必须包含 content-length 和 content-type 头。

```
MESSAGE

content-length:100

content-type:text/plain

destination:/queue/a

message-id:007

subscription:0

Hello queue a

^@
```

2.2.13 ERROR

如果连接过程中出现什么错误，服务端就会发送 ERROR 。在这种情况下，服务端发出 ERROR 之后必须马上断开连接。

2.2.14 RECEIPT

每当服务端收到来自客户端的需要 receipt 的帧时发送给客户端

3.结合webScocket和Stomp协议

蜗牛在正文的实时消息模块就结合了webScocket和Stomp。基于websocket的一层STOMP封装，让业务端只需关心数据本身，不需要太过关心文本协议。当然还是需要了解一些STOMP协议各个Frame的概念和应用场景。

现将项目中的工程抽成一个公有库，NESTOMP内含接入文档和Demo。希望以后在有类似于实时消息场景的需求上给大家一个方案预选。

非常赞👍

赞 回复

被以下专题收入，发现更多相似内容

+收入我的专题

推荐阅读

更多精彩内容>

Feign拦截器传递请求头信息

方式一： 1.创建FeignInterceptor配置类 2.在启动类上加 @Import(FeignInterc...

EnchantF 阅读 139 评论 0 赞 0

idea 显示RunDashboard

修改ideaworkspace.xml 增加

胡济川 阅读 86 评论 0 赞 0

退出登录使JWT失效的解决方案

项目引入了JWT，在实现退出功能的时候，发现即使调用了相关接口废弃令牌，但是令牌仍然可以使用。查看原码才知道，使用...

朱付生 阅读 518 评论 3 赞 10

SpringBoot+Shiro+Jwt实现登录认证

1. 概述 1.1 SpringBoot 这个就没什么好说的了，能看到这个教程的，估计都是可以说精通了Spring...

coderymy 阅读 329 评论 0 赞 1

springboot集成mqtt

1开发环境 采用maven仓库，初始化一个springboot的初始工程，pom文件额外导入spring inte...

MrJK_3539 阅读 45 评论 0 赞 0