# 实现UDP单播和广播

UDP 这样的无连接协议中，并没有持久化连接这样的概念，并且每个消息（一个UDP 数据报）都是一个单独的传输单元。此外，UDP 也没有TCP 的纠错机制。

通过类比，TCP 连接就像打电话，其中一系列的有序消息将会在两个方向上流动。相反，UDP 则类似于往邮箱中投入一叠明信片。你无法知道它们将以何种顺序到达它们的目的地，或者它们是否所有的都能够到达它们的目的地。

UDP的这些方面可能会让你感觉到严重的局限性，但是它们也解释了为何它会比TCP快那么多：所有的握手以及消息管理机制的开销都已经被消除了。显然，UDP很适合那些能够处理或者容忍消息丢失的应用程序，但可能不适合那些处理金融交易的应用程序。

本身作为无连接的不可靠的传输协议（适合频繁发送较小的数据包），他不会对数据包进行合并发送（也就没有Nagle算法之说了），他直接是一端发送什么数据，直接就发出去了，既然他不会对数据合并，每一个数据包都是完整的（数据+UDP头+IP头等等发一次数据封装一次）也就**没有粘包**一说了。

单播的传输模式，定义为发送消息给一个由唯一的地址所标识的单一的网络目的地。面向连接的协议和无连接协议都支持这种模式。

广播——传输到网络（或者子网）上的所有主机。

## Netty 的UDP 相关类

interface **AddressedEnvelope**<M, A extends SocketAddress>extends ReferenceCounted

定义一个消息，其包装了另一个消息并带有发送者和接收者地址。其中M 是消息类型；A 是

地址类型

class **DefaultAddressedEnvelope**<M, A extends SocketAddress>implements AddressedEnvelope<M,A>提供了interface AddressedEnvelope的默认实现

class **DatagramPacket** extends DefaultAddressedEnvelope<ByteBuf, InetSocketAddress> implements ByteBufHolder

扩展了DefaultAddressedEnvelope 以使用ByteBuf 作为消息数据容器。DatagramPacket是final类不能被继承，只能被使用。

**通过content()来获取消息内容**

**通过sender();来获取发送者的消息**

**通过recipient();来获取接收者的消息。**

interface **DatagramChannel** extends Channel

扩展了Netty 的Channel 抽象以支持UDP 的多播组管理

class **NioDatagramChannnel** extends AbstractNioMessageChannel implements DatagramChannel

定义了一个能够发送和接收Addressed-Envelope 消息的Channel 类型

Netty 的DatagramPacket 是一个简单的消息容器，DatagramChannel 实现用它来和远程节点通信。类似于在我们先前的类比中的明信片，它包含了接收者（和可选的发送者）的地址

以及消息的有效负载本身。

## UDP单播

看代码:udp工程下unicast包

## UDP广播

看代码:udp工程下broadcast包

# 服务器推送技术-短轮询和Comet

服务器推送技术干嘛用？就是让用户在使用网络应用的时候，不需要一遍又一遍的去手动刷新就可以及时获得更新的信息。大家平时在上各种视频网站时，对视频节目进行欢乐的吐槽和评论，会看到各种弹幕，当然，他们是用flash技术实现的，对于我们没有用flash的应用，一样可以实现弹幕。又比如在股票网站，往往可以看到，各种股票信息的实时刷新，上面的这些都是基于服务器推送技术。

## Ajax短轮询

就是用一个定时器不停的去网站上请求数据。

## Comet

“[服务器推](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8%E6%8E%A8)”是一种很早就存在的技术，以前在实现上主要是通过客户端的套接口，或是服务器端的远程调用。因为浏览器技术的发展比较缓慢，没有为“服务器推”的实现提供很好的支持，在纯浏览器的应用中很难有一个完善的方案去实现“服务器推”并用于商业程序。，因为 AJAX 技术的普及，gmail等等在实现中使用了这些新技术；同时“服务器推”在现实应用中确实存在很多需求。称这种基于 HTTP[长连接](https://baike.baidu.com/item/%E9%95%BF%E8%BF%9E%E6%8E%A5)、无须在浏览器端安装插件的“服务器推”技术为“Comet”。

#### 基于 AJAX 的长轮询

##### DeferredResult:

Spring mvc的控制层接收用户的请求之后，如果要采用异步处理，那么就要返回DeferedResult<>泛型对象。在调用完控制层之后，立即回返回DeferedResult对象，此时驱动控制层的容器主线程，可以处理更多的请求。

可以将DeferedResult对象作为真实响应数据的代理，而真实的数据是该对象的成员变量result，它可以是String类型，或者ModelAndView类型等。

业务处理完毕之后，要执行setResult方法，将真实的响应数据赋值到DeferedResult对象中。此时，异步线程会唤醒容器主线程。那么容器主线程会继续执行getResult方法，将真实数据响应到客户端。

#### SSE

严格地说，[HTTP 协议](http://www.ruanyifeng.com/blog/2016/08/http.html)无法做到服务器主动推送信息。但是，有一种变通方法，就是服务器向客户端声明，接下来要发送的是流信息（streaming）。

也就是说，发送的不是一次性的数据包，而是一个数据流，会连续不断地发送过来。这时，客户端不会关闭连接，会一直等着服务器发过来的新的数据流，视频播放就是这样的例子。本质上，这种通信就是以流信息的方式，完成一次用时很长的下载。

SSE 就是利用这种机制，使用流信息向浏览器推送信息。它基于 HTTP 协议，目前除了 IE/Edge，其他浏览器都支持。

SSE 与 WebSocket 作用相似，都是建立浏览器与服务器之间的通信渠道，然后服务器向浏览器推送信息。

总体来说，WebSocket 更强大和灵活。因为它是全双工通道，可以双向通信；SSE 是单向通道，只能服务器向浏览器发送，因为流信息本质上就是下载。如果浏览器向服务器发送信息，就变成了另一次 HTTP 请求。

SSE 也有自己的优点。

* SSE 使用 HTTP 协议，现有的服务器软件都支持。WebSocket 是一个独立协议。
* SSE 属于轻量级，使用简单；WebSocket 协议相对复杂。
* SSE 默认支持断线重连，WebSocket 需要自己实现。
* SSE 一般只用来传送文本，二进制数据需要编码后传送，WebSocket 默认支持传送二进制数据。
* SSE 支持自定义发送的消息类型。

##### HTTP 头信息

服务器向浏览器发送的 SSE 数据，必须是 UTF-8 编码的文本，具有如下的 **HTTP 头信息。**

Content-Type: text/event-stream

Cache-Control: no-cache

Connection: keep-alive

上面三行之中，第一行的Content-Type必须指定 MIME 类型为event-steam。

##### 信息格式

每一次发送的信息，由若干个message组成，每个message之间用\n\n分隔。每个message内部由若干行组成，每一行都是如下格式。

[field]: value\n

上面的field可以取四个值。

* data
* event
* id
* retry

此外，还可以有冒号开头的行，表示注释。通常，服务器每隔一段时间就会向浏览器发送一个注释，保持连接不中断。例子 : this is a test stream\n\n

###### data 字段

数据内容用data字段表示。

data: message\n\n

如果数据很长，可以分成多行，最后一行用\n\n结尾，前面行都用\n结尾。

data: begin message\n

data: continue message\n\n

下面是一个发送 JSON 数据的例子。

data: {\n

data: "foo": "bar",\n

data: "baz", 555\n

data: }\n\n

###### id 字段

数据标识符用id字段表示，相当于每一条数据的编号。

id: msg1\n

data: message\n\n

浏览器用lastEventId属性读取这个值。一旦连接断线，浏览器会发送一个 HTTP 头，里面包含一个特殊的Last-Event-ID头信息，将这个值发送回来，用来帮助服务器端重建连接。因此，这个头信息可以被视为一种同步机制。

###### event 字段

event字段表示自定义的事件类型，默认是message事件。浏览器可以用addEventListener()监听该事件。

event: foo\n

data: a foo event\n\n

data: an unnamed event\n\n

event: bar\n

data: a bar event\n\n

上面的代码创造了三条信息。第一条的名字是foo，触发浏览器的foo事件；第二条未取名，表示默认类型，触发浏览器的message事件；第三条是bar，触发浏览器的bar事件。

###### retry 字段

服务器可以用retry字段，指定浏览器重新发起连接的时间间隔。

retry: 10000\n

两种情况会导致浏览器重新发起连接：一种是时间间隔到期，二是由于网络错误等原因，导致连接出错。

## 技术比较

**京东用的什么？Ajax短轮询，**这说明什么？这些技术并没有什么优劣之分，只有合不合适业务的问题。京东的痛点是什么？要用有限的资源来为千万级甚至上亿的用户提供服务，如果是用长连接，对于接入的服务器，比如说Nginx，是很大的压力，光是为用户维持这个长连接都需要成百上千的Nginx的服务器，这是很划不来的。因为对于京东这类购物网站来说，用户的浏览查询量是远远大于用户下单量的，京东需要注重的是服务更多的用户，而且相对于用户浏览页面的图片等等的流量而言，这点带宽浪费占比是很小的。所以我们看京东的付款后的实现，是用的短轮询机制，而且时长放大到了5秒。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **短轮询** | **Serlvet异步** | **SSE** | **WebSocket** |
| 浏览器支持度 | 最高 | 高 | 中 | 低 |
| 实时性 | 最低 | 较高 | 高 | 最高 |
| 代码实现复杂度 | 最容易 | 较容易 | 容易 | 最复杂 |

SSE和WebSocket相比的优势。最大的优势就是便利：不需要添加任何新组件，用任何你习惯的后端语言和框架就能继续使用。你不用为新建虚拟机、弄一个新的IP或新的端口号而劳神，就像在现有网站中新增一个页面那样简单。可以称为既存基础设施优势。

SSE的第二个优势是服务端的简洁。相对而言，WebSocket则很复杂，不借助辅助类库基本搞不定。WebSocket能做的，SSE也能做，反之亦然，但在完成某些任务方面，它们各有千秋。WebSocket是一种更为复杂的服务端实现技术，但它是真正的双向传输技术，既能从服务端向客户端推送数据，也能从客户端向服务端推送数据。

# WebSocket通信

**什么是WebSocket？**

WebSocket ——一种在2011 年被互联网工程任务组（IETF）标准化的协议。

WebSocket解决了一个长期存在的问题：既然底层的协议（HTTP）是一个请求/响应模式的交互序列，那么如何实时地发布信息呢？AJAX提供了一定程度上的改善，但是数据流仍然是由客户端所发送的请求驱动的。还有其他的一些或多或少的取巧方式(Comet)

WebSocket规范以及它的实现代表了对一种更加有效的解决方案的尝试。简单地说，WebSocket提供了“在一个单个的TCP连接上提供双向的通信……结合WebSocket API……它为网页和远程服务器之间的双向通信提供了一种替代HTTP轮询的方案。”

，但是最终它们仍然属于扩展性受限的变通之法。也就是说，WebSocket 在客户端和服务器之间提供了真正的双向数据交换。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。

Web浏览器和服务器都必须实现 WebSockets 协议来建立和维护连接。

**特点**

* HTML5中的协议，实现与客户端与服务器双向，基于消息的文本或二进制数据通信
* 适合于对数据的实时性要求比较强的场景，如通信、直播、共享桌面，特别适合于客户与服务频繁交互的情况下，如实时共享、多人协作等平台。
* 采用新的协议，后端需要单独实现
* 客户端并不是所有浏览器都支持

# WebSocket通信握手

**客户端的请求：**

Connection 必须设置 Upgrade，表示客户端希望连接升级。

Upgrade 字段必须设置 Websocket，表示希望升级到 Websocket 协议。

Sec-WebSocket-Key 是随机的字符串，服务器端会用这些数据来构造出一个 SHA-1 的信息摘要。把 “Sec-WebSocket-Key” 加上一个特殊字符串 “258EAFA5-E914-47DA-95CA-C5AB0DC85B11”，然后计算 SHA-1 摘要，之后进行 BASE-64 编码，将结果做为 “Sec-WebSocket-Accept” 头的值，返回给客户端。如此操作，可以尽量避免普通 HTTP 请求被误认为 Websocket 协议。

Sec-WebSocket-Version 表示支持的 Websocket 版本。RFC6455 要求使用的版本是 13，之前草案的版本均应当弃用。

**服务器端：**

Upgrade: websocket

Connection: Upgrade

依然是固定的，告诉客户端即将升级的是 Websocket 协议，而不是mozillasocket，lurnarsocket或者shitsocket。

然后， Sec-WebSocket-Accept 这个则是经过服务器确认，并且加密过后的 Sec-WebSocket-Key 。

后面的， Sec-WebSocket-Protocol 则是表示最终使用的协议。

至此，HTTP已经完成它所有工作了，接下来就是完全按照Websocket协议进行

# WebSocket通信-STOMP

WebSocket是个规范，在实际的实现中有HTML5规范中的WebSocket API、WebSocket的子协议STOMP。

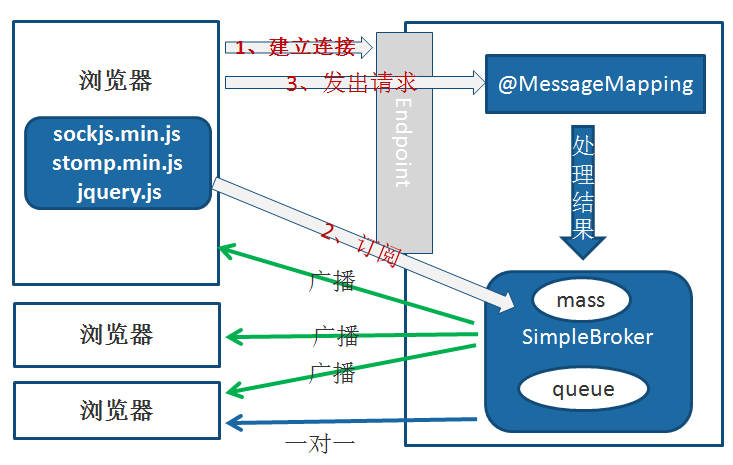
STOMP(Simple Text Oriented Messaging Protocol)

* 简单(流)文本定向消息协议
* STOMP协议的前身是TTMP协议（一个简单的基于文本的协议），专为消息中间件设计。是属于消息队列的一种协议, 和AMQP, JMS平级. 它的简单性恰巧可以用于定义websocket的消息体格式. STOMP协议很多MQ都已支持, 比如RabbitMq, ActiveMq。
* 生产者（发送消息）、消息代理、消费者（订阅然后收到消息）
* STOMP是基于帧的协议

# WebSocket通信实现

## SpringBoot

#### 基于Stomp的聊天室/IM的实现



具体实现：参考 stomp模块下的代码

和WebSocket的集成