Netty 断线重连解决方案 ☆

尹吉欢 · 2018-03-15 · 3条评论 · 5307人阅读

java (http://cxytiandi.com/article/search/java) netty (http://cxytiandi.com/article/search/netty)

本篇文章是Netty专题的第七篇，前面六篇文章如下：

- 高性能NIO框架Netty入门篇 (http://cxytiandi.com/blog/detail/17345)
- 高性能NIO框架Netty-对象传输 (http://cxytiandi.com/blog/detail/17403)
- 高性能NIO框架Netty-整合kryo高性能数据传输 (http://cxytiandi.com/blog/detail/17436)
- 高性能NIO框架Netty-整合Protobuf高性能数据传输 (http://cxytiandi.com/blog/detail/17469)
- Netty4自带编解码器详解 (http://cxytiandi.com/blog/detail/17547)
- Netty粘包拆包解决方案 (http://cxytiandi.com/blog/detail/17641)

用Netty实现长连接服务，当发生下面的情况时，会发生断线的情况。

- 网络问题
- 客户端启动时服务端挂掉了，连接不上服务端
- 客户端已经连接服务端，服务端突然挂掉了
- 其它问题等...

如何解决上面的问题?

1.心跳机制检测连接存活

长连接是指建立的连接长期保持，不管有无数据包的发送都要保持连接通畅。**心跳**是用来检测一个系统是否存活或者网络链路是否通畅的一种方式，一般的做法是客户端定时向服务端发送心跳包，服务端收到心跳包后进行回复，客户端收到回复说明服务端存活。

通过心跳检测机制，可以检测客户端与服务的长连接是否保持，当客户端发送的心跳包没有收到服务端的响应式，可以认为服务端已经出故障了，这个时候可以重新连接或者选择其他的可用的服务进行连接。

在Netty中提供了一个IdleStateHandler类用于心跳检测，用法如下:

```
1. ch.pipeline().addLast("ping", new IdleStateHandler(60, 20, 60 * 10, TimeUnit.SECONDS));
```

- 第一个参数 60 表示读操作空闲时间
- 第二个参数 20 表示写操作空闲时间
- 第三个参数 60*10 表示读写操作空闲时间
- 第四个参数 单位/秒

在处理数据的ClientPoHandlerProto中增加userEventTriggered用来接收心跳检测结果,event.state()的状态分别对应上面三个参数的时间设置，当满足某个时间的条件时会触发事件。

```
1. public class ClientPoHandlerProto extends ChannelInboundHandlerAdapter {
2.     private ImConnection imConnection = new ImConnection();
3. 
4.     @Override
5.     public void channelRead(ChannelHandlerContext ctx, Object msg) {
6.         MessageProto.Message message = (MessageProto.Message) msg;
7.         System.out.println("client:" + message.getContent());
8.     }
9. 
10.     @Override
11.     public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) {
12.         cause.printStackTrace();
13.         ctx.close();
14.     }
15. 
16.     @Override
17.     public void userEventTriggered(ChannelHandlerContext ctx, Object evt) throws Exception
    {
18.         super.userEventTriggered(ctx, evt);
19.         if (evt instanceof IdleStateEvent) {
20.             IdleStateEvent event = (IdleStateEvent) evt;
21.             if (event.state().equals(IdleState.READER_IDLE)) {
22.                 System.out.println("长期没收到服务器推送数据");
23.                 //可以选择重新连接
24.             } else if (event.state().equals(IdleState.WRITER_IDLE)) {
25.                 System.out.println("长期未向服务器发送数据");
26.                 //发送心跳包
27.                 ctx.writeAndFlush(MessageProto.Message.newBuilder().setType(1));
28.             } else if (event.state().equals(IdleState.ALL_IDLE)) {
29.                 System.out.println("ALL");
30.             }
31.         }
32.     }
33. }
```

服务端收到客户端发送的心跳消息后，回复一条信息

```java
public class ServerPoHandlerProto extends ChannelInboundHandlerAdapter {

    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) {
        MessageProto.Message message = (MessageProto.Message) msg;
        if (ConnectionPool.getChannel(message.getId()) == null) {
            ConnectionPool.putChannel(message.getId(), ctx);
        }
        System.err.println("server:" + message.getId());
        // ping
        if (message.getType() == 1) {
            ctx.writeAndFlush(message);
        }

    }

    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) {
        cause.printStackTrace();
        ctx.close();
    }
}
```

当客户端20秒没往服务端发送过数据，就会触发IdleState.WRITER_IDLE事件，这个时候我们就像服务端发送一条心跳数据，跟业务无关，只是心跳。服务端收到心跳之后就会回复一条消息，表示已经收到了心跳的消息，只要收到了服务端回复的消息，那么就不会触发IdleState.READER_IDLE事件，如果触发了IdleState.READER_IDLE事件就说明服务端没有给客户端响应，这个时候可以选择重新连接。

2.启动时连接重试

在Netty中实现重连的操作比较简单，Netty已经封装好了，我们只需要稍微扩展一下即可。

连接的操作是客户端这边执行的，重连的逻辑也得加在客户端，首先我们来看启动时要是连接不上怎么去重试

增加一个负责重试逻辑的监听器，代码如下：

```java
import java.util.concurrent.TimeUnit;

import com.netty.im.client.ImClientApp;

import io.netty.channel.ChannelFuture;
import io.netty.channel.ChannelFutureListener;
import io.netty.channel.EventLoop;
/**
 * 负责监听启动时连接失败，重新连接功能
 * @author yinjihuan
 *
 */
public class ConnectionListener implements ChannelFutureListener {

    private ImConnection imConnection = new ImConnection();

    @Override
    public void operationComplete(ChannelFuture channelFuture) throws Exception {
        if (!channelFuture.isSuccess()) {
            final EventLoop loop = channelFuture.channel().eventLoop();
            loop.schedule(new Runnable() {
                @Override
                public void run() {
                    System.err.println("服务端链接不上，开始重连操作...");
                    imConnection.connect(ImClientApp.HOST, ImClientApp.PORT);
                }
            }, 1L, TimeUnit.SECONDS);
        } else {
            System.err.println("服务端链接成功...");
        }
    }
}
```

通过channelFuture.isSuccess()可以知道在连接的时候是成功了还是失败了，如果失败了我们就启动一个单独的线程来执行重新连接的操作。

只需要在ConnectionListener添加到ChannelFuture中去即可使用

```
public class ImConnection {

    private Channel channel;

    public Channel connect(String host, int port) {
        doConnect(host, port);
        return this.channel;
    }

    private void doConnect(String host, int port) {
        EventLoopGroup workerGroup = new NioEventLoopGroup();
        try {
            Bootstrap b = new Bootstrap();
            b.group(workerGroup);
            b.channel(NioSocketChannel.class);
            b.option(ChannelOption.SO_KEEPALIVE, true);
            b.handler(new ChannelInitializer<SocketChannel>() {
                @Override
                public void initChannel(SocketChannel ch) throws Exception {

                    // 实体类传输数据，protobuf序列化
                    ch.pipeline().addLast("decoder",
                            new ProtobufDecoder(MessageProto.Message.getDefaultInstance
()));
                    ch.pipeline().addLast("encoder",
                            new ProtobufEncoder());
                    ch.pipeline().addLast(new ClientPoHandlerProto());

                }
            });

            ChannelFuture f = b.connect(host, port);
            f.addListener(new ConnectionListener());
            channel = f.channel();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

}
```

可以按照如下步骤进行测试：

- 直接启动客户端，不启动服务端

- 当连接失败的时候会进入ConnectionListener中的operationComplete方法执行我们的重连逻辑

3.运行中连接断开时重试

使用的过程中服务端突然挂了，就得用另一种方式来重连了，可以在处理数据的Handler中进行处理。

```
public class ClientPoHandlerProto extends ChannelInboundHandlerAdapter {
    private ImConnection imConnection = new ImConnection();

    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) {
        MessageProto.Message message = (MessageProto.Message) msg;
        System.out.println("client:" + message.getContent());
    }

    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) {
        cause.printStackTrace();
        ctx.close();
    }

    @Override
    public void channelInactive(ChannelHandlerContext ctx) throws Exception {
        System.err.println("掉线了...");
        //使用过程中断线重连
        final EventLoop eventLoop = ctx.channel().eventLoop();
        eventLoop.schedule(new Runnable() {
            @Override
            public void run() {
                imConnection.connect(ImClientApp.HOST, ImClientApp.PORT);
            }
        }, 1L, TimeUnit.SECONDS);
        super.channelInactive(ctx);
    }

}
```

在连接断开时都会触发 channelInactive 方法, 处理重连的逻辑跟上面的一样。

可以按照如下步骤进行测试：

- 启动服务端
- 启动客户端，连接成功
- 停掉服务端就会触发channelInactive进行重连操作

源码参考：https://github.com/yinjihuan/netty-im (https://github.com/yinjihuan/netty-im)