高性能NIO框架Netty-整合Protobuf高性能数据传输 ☆

尹吉欢 · 2018-03-05 · 1条评论 · 4540人阅读

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

前言

本篇文章是Netty专题的第四篇，前面三篇文章如下：

- 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)

上篇文章我们整合了kryo来进行数据的传输编解码，今天将继续学习使用Protobuf来编解码。Netty对Protobuf的支持比较好，还提供了Protobuf的编解码器，非常方便。

Protobuf介绍

GitHub地址：https://github.com/google/protobuf (https://github.com/google/protobuf)

Protobuf是google开源的项目，全称 Google Protocol Buffers，特点如下：

- 支持跨平台多语言，支持目前绝大多数语言例如C++、C#、Java、pthyon等
- 高性能，可靠性高，google出品有保障
- 使用protobuf编译器能自动生成代码，但需要编写proto文件，需要一点学习成本

Protobuf使用

Protobuf是将类的定义使用.proto文件进行描述，然后通过protoc.exe编译器，根据.proto自动生成.java文件，然后将生成的.java文件拷贝到项目中使用即可。

在Github主页我们下周Windows下的编译器，可以在releases页面下载：
https://github.com/google/protobuf/releases (https://github.com/google/protobuf/releases)

 protoc.exe编译器下载

下载完成之后放到磁盘上进行解压，可以将protoc.exe配置到环境变量中去，这样就可以直接在cmd命令行中使用protoc命令，也可以不用配置，直接到解压后的protoc\bin目录下进行文件的编译。

下面我们基于之前的Message对象来构建一个Message.proto文件。

```
syntax = "proto3";
option java_outer_classname = "MessageProto";
message Message {
   string id = 1;
   string content = 2;
}
```

syntax 声明可以选择protobuf的编译器版本(v2和v3)

- syntax=”proto2”;选择2版本
- syntax=”proto3”;选择3版本

option java_outer_classname=”MessageProto”用来指定生成的java类的类名。
message相当于c语言中的struct语句，表示定义一个信息，其实也就是类。
message里面的信息就是我们要传输的字段了，子段后面需要有一个数字编号，从1开始递增

.proto文件定好之后就可以用编译器进行编译，输出我们要使用的Java类，我们这边不配置环境变量，直接到解压包的bin目录下进行操作

首先将我们的Message.proto文件复制到bin目录下，然后在这个目录下打开CMD窗口，输入下面的命令进行编译操作：

```
protoc ./Message.proto --java_out=./
```

—java_out是输出目录，我们就输出到当前目录下，执行完之后可以看到bin目录下多了一个MessageProto.java文件，把这个文件复制到项目中使用即可。

Nettty整合Protobuf

首先加入Protobuf的Maven依赖

```
<!-- https://mvnrepository.com/artifact/com.google.protobuf/protobuf-java -->
<dependency>
    <groupId>com.google.protobuf</groupId>
    <artifactId>protobuf-java</artifactId>
    <version>3.5.1</version>
</dependency>
```

创建一个Proto的Server数据处理类，之前的已经不能用了，因为现在传输的对象是MessageProto这个对象了

```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());
        ctx.writeAndFlush(message);
    }

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

改造服务端启动代码，增加protobuf编解码器，是Netty自带的，不用我们去自定义了

```
public class ImServer {

    public void run(int port) {
        EventLoopGroup bossGroup = new NioEventLoopGroup();
        EventLoopGroup workerGroup = new NioEventLoopGroup();

        ServerBootstrap bootstrap = new ServerBootstrap();
        bootstrap.group(bossGroup, workerGroup)
                .channel(NioServerSocketChannel.class)
                .childHandler(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 ServerPoHandlerProto());

                    }
                })
                .option(ChannelOption.SO_BACKLOG, 128)
                .childOption(ChannelOption.SO_KEEPALIVE, true);

        try {
            ChannelFuture f = bootstrap.bind(port).sync();
             f.channel().closeFuture().sync();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            workerGroup.shutdownGracefully();
            bossGroup.shutdownGracefully();
        }
    }

}
```

服务端改造完了，下面需要把客户的的Handler和编解码器也改成protobuf的就行了，废话不多说，直接上代码

```java
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).sync();
            channel = f.channel();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

}
```

客户的数据处理类：

```
public class ClientPoHandlerProto extends ChannelInboundHandlerAdapter {

    @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();
    }

}
```

最后一步就开始测试了，需要将客户的发送消息的地方改成MessageProto.Message对象，代码如下：

```
/**
 * IM 客户端启动入口
 * @author yinjihuan
 */
public class ImClientApp {
    public static void main(String[] args) {
        String host = "127.0.0.1";
        int port = 2222;
        Channel channel = new ImConnection().connect(host, port);
        String id = UUID.randomUUID().toString().replaceAll("-", "");
        // protobuf
        MessageProto.Message message = MessageProto.Message.newBuilder().setId(id).setCont
ent("hello yinjihuan").build();
        channel.writeAndFlush(message);
    }
}
```

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

欢迎加入我的知识星球，一起交流技术，免费学习猿天地的课程（http://cxytiandi.com/course） (http://cxytiandi.com/course）)

PS：目前星球中正在星主的带领下组队学习Sentinel，等你哦！