链滴

5. Netty 初认识 -- 入门应用 - 接收浏览器请求输出固定话术 2

作者：289306290

原文链接：https://ld246.com/article/1582883435865

来源网站：链滴

涉及两个类

SomeServer.java

```
package club.wujingjian.study;

import io.netty.bootstrap.ServerBootstrap;
import io.netty.channel.ChannelFuture;
import io.netty.channel.ChannelInitializer;
import io.netty.channel.ChannelPipeline;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.SocketChannel;
import io.netty.channel.socket.nio.NioServerSocketChannel;
import io.netty.handler.codec.http.HttpServerCodec;

public class SomeServer {
    public static void main(String[] args) throws InterruptedException {

        //用于处理客户端连接请求,将请求发送给childGroup中的EventLoop
        NioEventLoopGroup parentGroup = new NioEventLoopGroup();
        //用于处理客户端请求
        NioEventLoopGroup childGroup = new NioEventLoopGroup();

        try {
            //用于启动ServerChannel
            ServerBootstrap bootstrap = new ServerBootstrap();
            bootstrap.group(parentGroup, childGroup) //指定EventLoopGroup
                    .channel(NioServerSocketChannel.class) //指定使用NIO进行通信
                    .childHandler(new ChannelInitializer<SocketChannel>() {
                        @Override
                        protected void initChannel(SocketChannel ch) throws Exception {
                            // 从Channel中获取ChannelPipeline
                            ChannelPipeline pipeline = ch.pipeline();
                            // 将HttpServerCodec处理器放入pipeline的最后
                            // HttpServerCodec 是什么? 是HttpRequestDecoder(http请求解码器) 与HttpR
esponseEncoder（http编码器）的复合体
                            // 是HttpRequestDecoder 将channel中的bytebuffer数据解码为HttpRequest
象
                            // HttpResponseEncoder 将HttpResponse对象编码为将要在Channel中发送的B
teBuf数据
                            pipeline.addLast(new HttpServerCodec());
                            // 将自定义的处理器放入到pipeline的最后
                            pipeline.addLast(new SomeServerHandler());
                        }
                    }); //指定childGroup中的eventLoop所绑定的线程索要处理的处理器

            // 指定当前服务器所监听的端口号
            // bind()方法的执行是异步的
            //加上.sync()方法后会使bind()操作与后续代码的执行由异步变成同步
            ChannelFuture future = bootstrap.bind(8888).sync();
            // 关闭channel
            // closefuture()的执行是异步的.
            // 当channel调用了close()方法并关闭成功后才会触发closeFurure()方法的执行.
            // 加上.sync()方法后变成同步
```

```
            System.out.println("服务器启动成功,监听的端口号为: 8888");
            future.channel().closeFuture().sync();
        } finally {
            // 优雅关闭
            parentGroup.shutdownGracefully();
            childGroup.shutdownGracefully();
        }
    }
}
```

SomeServerHandler.java

```
package club.wujingjian.study;

import io.netty.buffer.ByteBuf;
import io.netty.buffer.Unpooled;
import io.netty.channel.ChannelFutureListener;
import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.ChannelInboundHandlerAdapter;
import io.netty.handler.codec.http.*;
import io.netty.util.CharsetUtil;

import java.nio.ByteBuffer;

// 自定义的服务端处理器
// 需求:  用户提交一个请求后,在浏览器上就会看到hello netty world!
public class SomeServerHandler extends ChannelInboundHandlerAdapter {

    /**
     * 当Channel中有来自于客户端的数据时就会触发该方法的执行.
     * @param ctx 上下文对象
     * @param msg 来自于Channel中的数据(Channel中的数据是来自于客户端的)
     * @throws Exception
     */
    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
        System.out.println("msg = " + msg.getClass());
        System.out.println("客户端地址 =  " + ctx.channel().remoteAddress());
//        super.channelRead(ctx, msg);
        if (msg instanceof HttpRequest) {
            HttpRequest httpRequest = (HttpRequest) msg;
            System.out.println("请求方式:" + httpRequest.method());
            System.out.println("请求uri:" + httpRequest.uri());

            if (httpRequest.uri().equals("/favicon.ico")) {
                System.out.println("不处理/favicon.ico请求");
                return;
            }

            //构造response的响应体
            ByteBuf body = Unpooled.copiedBuffer("hello netty world!", CharsetUtil.UTF_8) ;
            // 生成响应对象
            DefaultFullHttpResponse response = new DefaultFullHttpResponse(HttpVersion.HTTP
```

```
1_1, HttpResponseStatus.OK, body);
            // 获取到Response的头部后进行初始化.
            HttpHeaders headers = response.headers();
            headers.set(HttpHeaderNames.CONTENT_TYPE, "text/plain");
            headers.set(HttpHeaderNames.CONTENT_LENGTH, body.readableBytes());

            //将响应对象写入到Channel
//            ctx.write(response);
//            ctx.flush();
            ctx.writeAndFlush(response)
                    //添加监听器,响应体发送完毕则直接将Channel关闭
                    .addListener(ChannelFutureListener.CLOSE);
        }
    }

    /**
     *  当Channel中的数据在处理时候出现异常时候会触发该方法的执行
     * @param ctx 上下文
     * @param cause 异常信息
     * @throws Exception
     */
    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) throws Excepti
n {
//        super.exceptionCaught(ctx, cause);
        System.out.println("执行异常了--exceptionCaught" + cause.getMessage());
        // 关闭Channel
        ctx.close();
    }
}
```