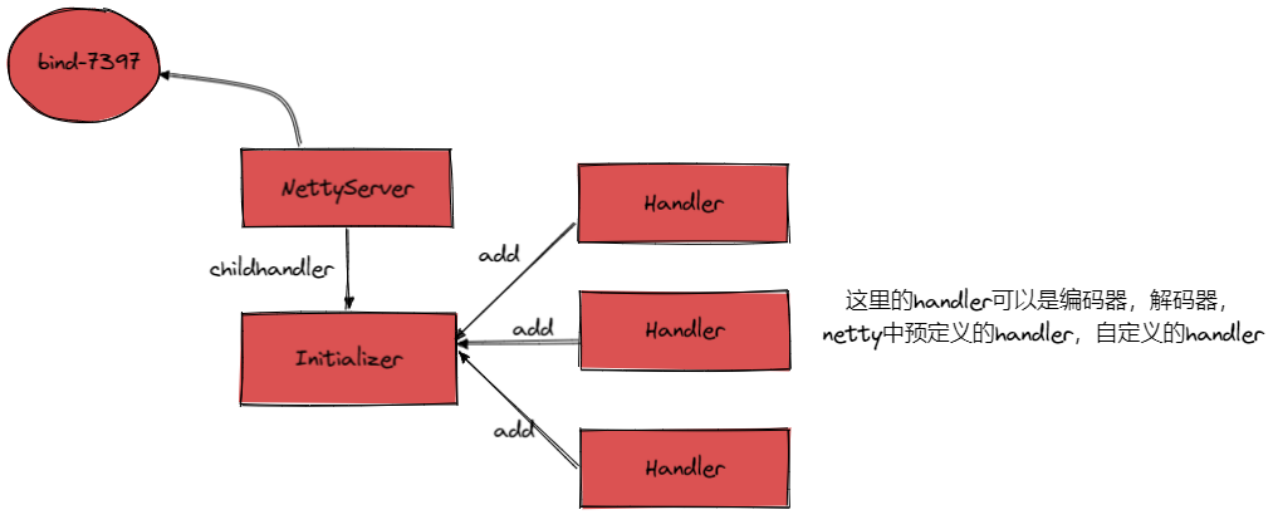
API网关-第一节《处理HTTP请求并处理session会话》

我印象里的Netty使用

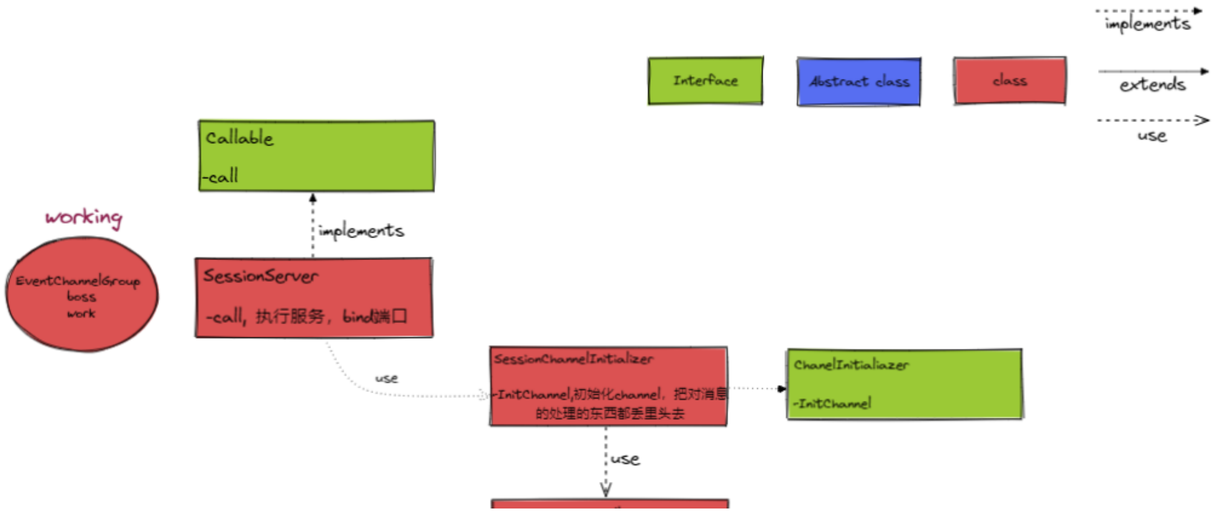
之前的项目中，自己看着小傅哥的小册也摸索着使用过netty，使用基本上也就是以下的流程，个人使用的是继承ChannelInboundHandlerAdapter的handler，可以处理用户端传来的消息，并在连接、收到消息、关闭时做出回应。

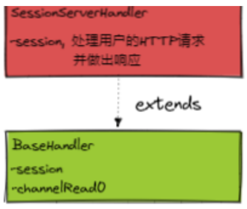


API网关中的Session基础设计

第一节的基础类图

这一块算是比较基础的netty操作，NettyServer实现了Callable接口是为了后续的call调用，这里的端口是硬编码的，我觉得后续可以提供出另外的方法去读取yaml或者xml中的设置，比如新增一个init方法，server方法，给用户留有空间在call里面init，否则就用默认。

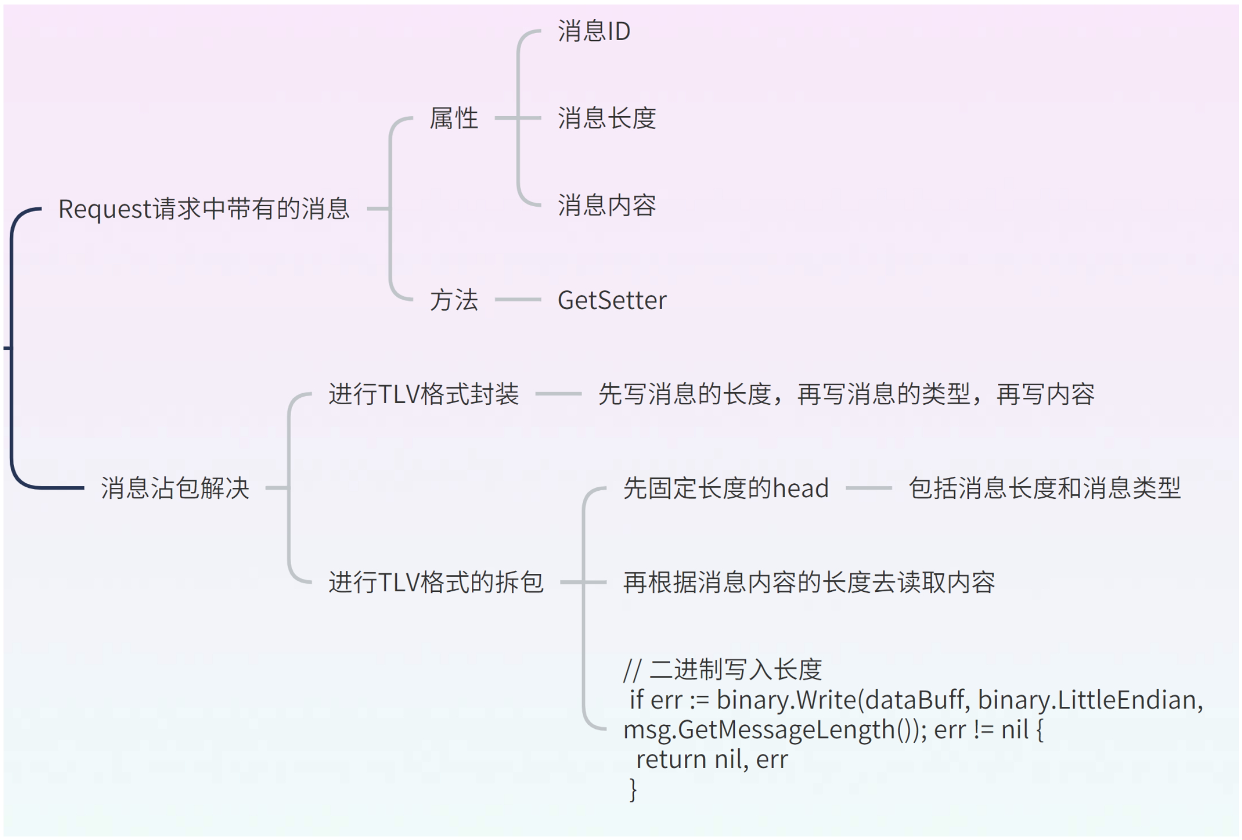




代码实现

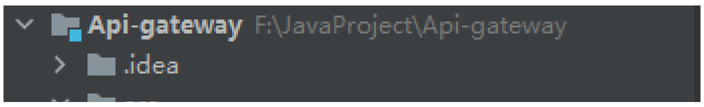
大纲：

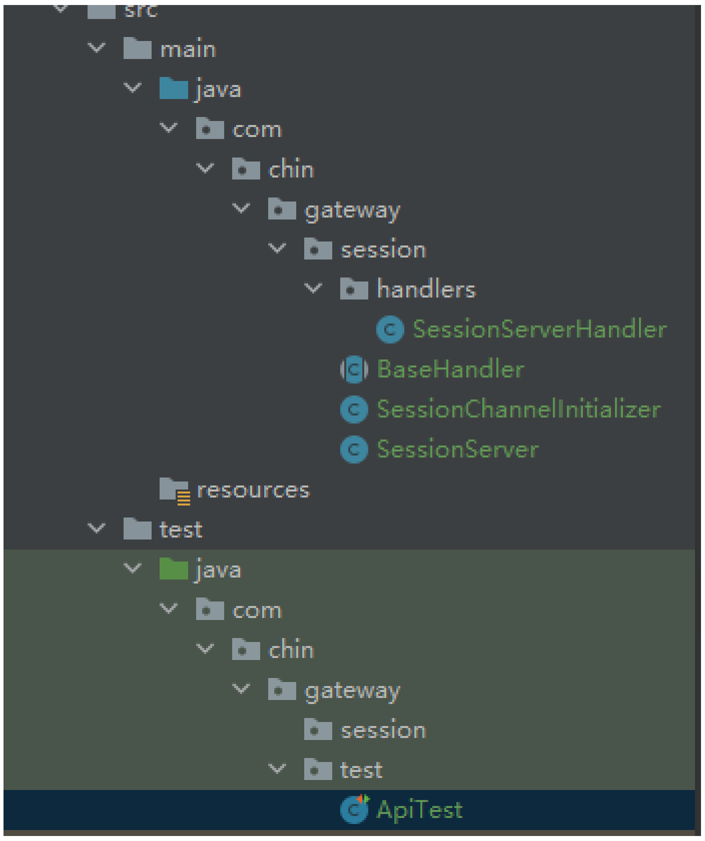
- SessionServer：实现Callable接口，实现call方法，用bind去监听端口，并.childChannel加入SessionChannelInitializer。
- SessionChannelInitializer：一个基础的使用，实现了Netty的ChannelInitializer的initChannel方法。加入了编码解码器处理沾包问题（去回顾一下沾包和半包，TCP的经典问题），最后加入我们的处理逻辑SessionServerHandler
- SessionServerHandler：这里实现一个session方法，也就是对方HTTP请求打过来以后，我该怎么处理，或者怎么回复对方。
- 回顾自己实现的服务器是怎么处理消息的：使用TLV协议，手动的进行拆包，封包，提取出其中的消息



```
// 二进制写入长度
 if err := binary.Write(dataBuff, binary.LittleEndian, msg.GetMessageLength()); err != nil {
  return nil, err
 }
```

代码架构





```
src
  main
    java
      com
        chin
          gateway
            session
              handlers
                SessionServerHandler
              BaseHandler
              SessionChannelInitializer
              SessionServer
    resources
  test
    java
      com
        chin
          gateway
            session
            test
              ApiTest
```

依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/mave
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>Api-gateway</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>io.netty</groupId>
            <artifactId>netty-all</artifactId>
            <version>4.1.77.Final</version>
        </dependency>
        <dependency>
```

```xml
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>2.0.7</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.7.36</version>
            <type>jar</type>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-core</artifactId>
            <version>1.2.11</version>
            <type>jar</type>
        </dependency>
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>1.2.11</version>
            <type>jar</type>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.dubbo</groupId>
            <artifactId>dubbo</artifactId>
            <version>2.7.5</version>
        </dependency>
        <dependency>
            <groupId>org.apache.zookeeper</groupId>
            <artifactId>zookeeper</artifactId>
            <version>3.4.13</version>
        </dependency>
        <dependency>
            <groupId>org.apache.curator</groupId>
            <artifactId>curator-framework</artifactId>
            <version>4.0.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.curator</groupId>
            <artifactId>curator-recipes</artifactId>
            <version>4.0.1</version>
        </dependency>
        <dependency>
            <groupId>cglib</groupId>
            <artifactId>cglib</artifactId>
            <version>3.3.0</version>
        </dependency>
        <dependency>
            <groupId>commons-httpclient</groupId>
            <artifactId>commons-httpclient</artifactId>
            <version>3.1</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>${maven.compiler.source}</source>
                    <target>${maven.compiler.target}</target>
                    <encoding>${project.build.sourceEncoding}</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

SessionServer：

```java
package com.chin.gateway.session;


import io.netty.bootstrap.ServerBootstrap;
import io.netty.channel.Channel;
import io.netty.channel.ChannelFuture;
import io.netty.channel.ChannelOption;
import io.netty.channel.EventLoopGroup;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.nio.NioServerSocketChannel;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.net.InetSocketAddress;
import java.util.concurrent.Callable;
```

```java
/**
 * @author qi
 */
public class SessionServer implements Callable<Channel> {

    private Logger logger = LoggerFactory.getLogger(SessionServer.class);

    private final EventLoopGroup boss = new NioEventLoopGroup(1);
    private final EventLoopGroup work = new NioEventLoopGroup();
    private Channel channel;

    @Override
    public Channel call() throws Exception {
        ChannelFuture channelFuture = null;
        try {
            ServerBootstrap b = new ServerBootstrap();
            b.group(boss, work).channel(NioServerSocketChannel.class)
                    .option(ChannelOption.SO_BACKLOG, 128)
                    .childHandler(new SessionChannelInitializer());
            channelFuture = b.bind(new InetSocketAddress(7397)).syncUninterruptibly();
            this.channel = channelFuture.channel();
        } catch (Exception e) {
            logger.error("socket server start error.", e);
        } finally {
            if (null != channelFuture && channelFuture.isSuccess()) {
                logger.info("socket server start done.");
            } else {
                logger.error("socket server start error");
            }
        }
        return channel;
    }
}
```

SessionChannelInitializer

```java
package com.chin.gateway.session;

import com.chin.gateway.session.handlers.SessionServerHandler;
import io.netty.channel.Channel;
import io.netty.channel.ChannelInitializer;
import io.netty.handler.codec.http.HttpObjectAggregator;
import io.netty.handler.codec.http.HttpRequestDecoder;
import io.netty.handler.codec.http.HttpResponseDecoder;
import io.netty.handler.codec.http.HttpResponseEncoder;

/**
 * @author qi
 * 初始Server时定义的childChannel，每个消息传过来的处理
 */
public class SessionChannelInitializer extends ChannelInitializer {

    @Override
    protected void initChannel(Channel channel) throws Exception {
        channel.pipeline().addLast(new HttpRequestDecoder());
        channel.pipeline().addLast(new HttpResponseEncoder());
        channel.pipeline().addLast(new HttpObjectAggregator(1024 * 1024));
        channel.pipeline().addLast(new SessionServerHandler());
    }
}
```

BaseHandler

```java
package com.chin.gateway.session;

import io.netty.channel.Channel;
import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.SimpleChannelInboundHandler;

/**
 * @author qi
 * @description Handler的抽象类
 */
public abstract class BaseHandler<T> extends SimpleChannelInboundHandler<T> {

    @Override
    protected void channelRead0(ChannelHandlerContext channelHandlerContext, T msg) throws Exce
        // 读取channel中的东西再去处理，这里是模板方法的模式，后期容易扩充。
        session(channelHandlerContext, channelHandlerContext.channel(), msg);
    }

    /**
     * 处理channel的msg的具体逻辑
     * @param channelHandlerContext
     * @param channel
     * @param request
     */
    protected abstract void session(ChannelHandlerContext channelHandlerContext, final Channel
}
```

SessionServerHandler

```
package com.chin.gateway.session.handlers;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.serializer.SerializerFeature;
import com.chin.gateway.session.BaseHandler;
import io.netty.channel.Channel;
import io.netty.channel.ChannelHandlerContext;
import io.netty.handler.codec.http.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * @author qi
 */
public class SessionServerHandler extends BaseHandler<FullHttpRequest> {

    private final Logger logger = LoggerFactory.getLogger(SessionServerHandler.class);

    @Override
    protected void session(ChannelHandlerContext channelHandlerContext, Channel channel, FullHt
        logger.info("receive msg from {}, uri {}, method: {}", channel.remoteAddress(), request

        // 返回的response
        DefaultFullHttpResponse response = new DefaultFullHttpResponse(HttpVersion.HTTP_1_1, Ht
        // 返回的信息
        response.content().writeBytes(JSON.toJSONBytes("你访问的路径被Chin的网关代理， URI:" + re
        // Header
        HttpHeaders heads = response.headers();
        // 返回内容的类型
        heads.add(HttpHeaderNames.CONTENT_TYPE, HttpHeaderValues.APPLICATION_JSON + "; charset=
        // 响应体的长度
        heads.add(HttpHeaderNames.CONTENT_LENGTH, response.content().readableBytes());
        // 配置持久连接
        heads.add(HttpHeaderNames.CONNECTION, HttpHeaderValues.KEEP_ALIVE);
        // 配置跨域访问
        heads.add(HttpHeaderNames.ACCESS_CONTROL_ALLOW_ORIGIN, "*");
        heads.add(HttpHeaderNames.ACCESS_CONTROL_ALLOW_HEADERS, "*");
        heads.add(HttpHeaderNames.ACCESS_CONTROL_ALLOW_METHODS, "GET, POST, PUT, DELETE");
        heads.add(HttpHeaderNames.ACCESS_CONTROL_ALLOW_CREDENTIALS, "true");

        channel.writeAndFlush(response);
    }
}
```

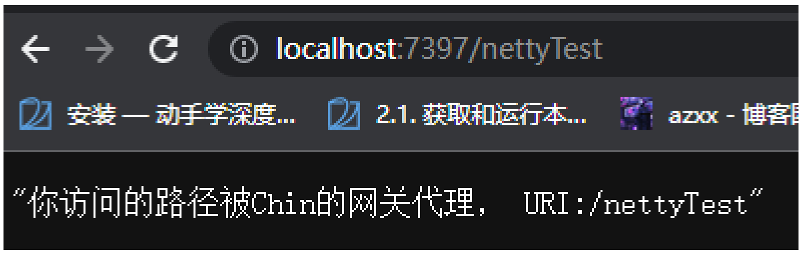
注意的点：小傅哥在类图上画的是实现BaseHandler，但是写的时候发现这很明显的是一个模板模式，从名字也看出来了，是一个小错误。

测试

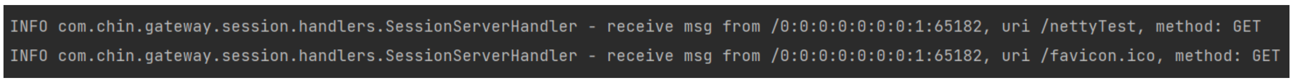
代码

```
package com.chin.gateway.test;

import com.chin.gateway.session.SessionServer;
import io.netty.channel.Channel;
import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class ApiTest {

    private final Logger logger = LoggerFactory.getLogger(ApiTest.class);

    @Test
    public void test() throws ExecutionException, InterruptedException {
        SessionServer server = new SessionServer();
        Future<Channel> future = Executors.newFixedThreadPool(2).submit(server);
        Channel channel = future.get();
        if (null == channel) throw new RuntimeException("netty server start error channel is nu

        while (!channel.isActive()) {
            logger.info("Netty Start");
            Thread.sleep(500);
        }
        logger.info("NettyServer Started ok bind:{}", channel.localAddress());

        for (;;){

        }
    }
}
```

结果



"你访问的路径被Chin的网关代理， URI:/nettyTest"



```
INFO com.chin.gateway.session.handlers.SessionServerHandler - receive msg from /0:0:0:0:0:0:0:1:65182, uri /nettyTest, method: GET
INFO com.chin.gateway.session.handlers.SessionServerHandler - receive msg from /0:0:0:0:0:0:0:1:65182, uri /favicon.ico, method: GET
```

一个基础的Netty服务器搭建就建立啦。这里我猜测一下，获取到了请求的路径那就可以有后续的处理咯，可能进行一些基础的路由操作。

总结

学习过程中，我也一直在复习我之前做的WebSocket，其实个人觉得Netty的一些操作比较固定，记忆一下或者google、百度一搜都有，主要还是在处理的逻辑是要自己写的。

- 中途发现其实HTTP请求和自己的WebSocket操作是不太一样的，WebSocket其实是要经过多一次的握手去进行协议的转化，然后发送的报文格式也是不一样的，之前发送的是TextWebSocketFrame，现在是DefaultFullHttpResponse。