7-12 | im 系统的核心 handler 设计与实现

设计 IM-Handler 工厂

定义了一个统一的 HandlerFacotry 工厂，对外暴露一个 doMsgHandler 接口：

```
Java
package org.qiyu.live.im.core.server.handler;

import io.netty.channel.ChannelHandlerContext;
import org.qiyu.live.im.core.server.common.ImMsg;

/**
 * @Author idea
 * @Date: Created in 20:42 2023/7/6
 * @Description
 */
public interface ImHandlerFactory {

    /**
     * 按照 immsg 的 code 去筛选
     *
     * @param channelHandlerContext
     * @param imMsg
     */
    void doMsgHandler(ChannelHandlerContext channelHandlerContext,
ImMsg imMsg);
}
```

其接口的下边有一个具体的 handler 工厂类：

```
Java
package org.qiyu.live.im.core.server.handler.impl;

import io.netty.channel.ChannelHandlerContext;
import org.qiyu.live.im.core.server.common.ImMsg;
import org.qiyu.live.im.core.server.handler.ImHandlerFactory;
import org.qiyu.live.im.core.server.handler.SimplyHandler;
```

```
import org.qiyu.live.im.interfaces.ImMsgCodeEnum;

import java.util.HashMap;
import java.util.Map;

/**
 * @Author idea
 * @Date: Created in 20:42 2023/7/6
 * @Description
 */
public class ImHandlerFactoryImpl implements ImHandlerFactory {

    private static Map<Integer, SimplyHandler> simplyHandlerMap =
new HashMap<>();

    static {
        //登录消息包，登录 token 认证，channel 和 userId 关联
        //等出消息包，正常断开 im 连接的时候发送的
        //业务消息包，最常用的消息类型，例如我们的 im 发送数据，或者接收
数据的时候会用到
        //心跳消息包，定时会给 im 发送，汇报功能
        simplyHandlerMap.put(ImMsgCodeEnum.IM_LOGIN_MSG.getCode(),
new LoginMsgHandler());

simplyHandlerMap.put(ImMsgCodeEnum.IM_LOGOUT_MSG.getCode(), new
LogoutMsgHandler());
        simplyHandlerMap.put(ImMsgCodeEnum.IM_BIZ_MSG.getCode(),
new BizImMsgHandler());

simplyHandlerMap.put(ImMsgCodeEnum.IM_HEARTBEAT_MSG.getCode(), new
HeartBeatImMsgHandler());
    }

    @Override
    public void doMsgHandler(ChannelHandlerContext
channelHandlerContext, ImMsg imMsg) {
        SimplyHandler simplyHandler =
simplyHandlerMap.get(imMsg.getCode());
        if (simplyHandler == null) {
            throw new IllegalArgumentException("msg code is
error,code is :" + imMsg.getCode());
        }
        simplyHandler.handler(channelHandlerContext, imMsg);
    }
```

```
}
```

这个工厂类，主要是提供给整个 Netty 应用接收外界消息包时候使用的：

```
Java
package org.qiyu.live.im.core.server.handler;

import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.SimpleChannelInboundHandler;
import org.qiyu.live.im.core.server.common.ImMsg;
import
org.qiyu.live.im.core.server.handler.impl.ImHandlerFactoryImpl;

/**
 * im 消息统一 handler 入口
 *
 * @Author idea
 * @Date: Created in 20:31 2023/7/6
 * @Description
 */
public class ImServerCoreHandler extends
SimpleChannelInboundHandler {

    private ImHandlerFactory imHandlerFactory = new
ImHandlerFactoryImpl();

    @Override
    protected void messageReceived(ChannelHandlerContext ctx,
Object msg) throws Exception {
        if (!(msg instanceof ImMsg)) {
            throw new IllegalArgumentException("error msg,msg
is :" + msg);
        }
        ImMsg imMsg = (ImMsg) msg;
        imHandlerFactory.doMsgHandler(ctx,imMsg);
    }
}
```

消息处理器

我们会按照消息内部的 code 值来区分不同业务场景的消息数据包，然后将这些不同场景的消

息数据包转发给不同的 handler 去处理，其相关接口定义如下：

```
Java
package org.qiyu.live.im.core.server.handler;

import io.netty.channel.ChannelHandlerContext;
import org.qiyu.live.im.core.server.common.ImMsg;

/**
 * @Author idea
 * @Date: Created in 20:39 2023/7/6
 * @Description
 */
public interface SimplyHandler {

    /**
     * 消息处理函数
     *
     * @param ctx
     * @param imMsg
     */
    void handler(ChannelHandlerContext ctx, ImMsg imMsg);
}
```

该接口是 ImHandlerFactoryImpl 类中会调用的一个底层接口，其具体实现是交给不同的 handler 具体类去完成。下边我们来定义不同的处理器类。

登录处理器

```
Java
package org.qiyu.live.im.core.server.handler.impl;

import io.netty.channel.ChannelHandlerContext;
import org.qiyu.live.im.core.server.common.ImMsg;
import org.qiyu.live.im.core.server.handler.SimplyHandler;

/**
 * 登录消息的处理逻辑统一收拢到这个类中
 *
 * @Author idea
 * @Date: Created in 20:40 2023/7/6
 * @Description
 */
public class LoginMsgHandler implements SimplyHandler {
```

```
    @Override
    public void handler(ChannelHandlerContext ctx, ImMsg imMsg) {
        System.out.println("[login]:" + imMsg);
        ctx.writeAndFlush(imMsg);
    }
}
```

登出处理器

```
Java
package org.qiyu.live.im.core.server.handler.impl;

import io.netty.channel.ChannelHandlerContext;
import org.qiyu.live.im.core.server.common.ImMsg;
import org.qiyu.live.im.core.server.handler.SimplyHandler;
import org.qiyu.live.im.interfaces.ImMsgCodeEnum;

/**
 * 登出消息的处理逻辑统一收拢到这个类中
 *
 * @Author idea
 * @Date: Created in 20:40 2023/7/6
 * @Description
 */
public class LogoutMsgHandler implements SimplyHandler {

    @Override
    public void handler(ChannelHandlerContext ctx, ImMsg imMsg) {
        System.out.println("[logout]:" + imMsg);
        ctx.writeAndFlush(imMsg);
    }
}
```

用户心跳包处理器

```
Java
package org.qiyu.live.im.core.server.handler.impl;

import io.netty.channel.ChannelHandlerContext;
import org.qiyu.live.im.core.server.common.ImMsg;
```

```
import org.qiyu.live.im.core.server.handler.SimplyHandler;

/**
 * 心跳消息处理器
 *
 * @Author idea
 * @Date: Created in 20:41 2023/7/6
 * @Description
 */
public class HeartBeatImMsgHandler implements SimplyHandler {

    @Override
    public void handler(ChannelHandlerContext ctx, ImMsg imMsg) {
        System.out.println("[heartBeat]:" + imMsg);
        ctx.writeAndFlush(imMsg);
    }
}
```

业务消息处理器

```
Java
package org.qiyu.live.im.core.server.handler.impl;

import io.netty.channel.ChannelHandlerContext;
import org.qiyu.live.im.core.server.common.ImMsg;
import org.qiyu.live.im.core.server.handler.SimplyHandler;

/**
 * 业务消息处理器
 *
 * @Author idea
 * @Date: Created in 20:41 2023/7/6
 * @Description
 */
public class BizImMsgHandler implements SimplyHandler {

    @Override
    public void handler(ChannelHandlerContext ctx, ImMsg imMsg) {
        System.out.println("[bizImMsg]:" + imMsg);
        ctx.writeAndFlush(imMsg);
    }
}
```

最后我们需要在启动类 NettyImServerApplication 的 startApplication 方法中加入 ImServerCoreHandler 这个类：

```
//基于netty去启动一个java进程，绑定监听的端口
public void startApplication(int port) throws InterruptedException {
    setPort(port);
    //处理accept事件
    NioEventLoopGroup bossGroup = new NioEventLoopGroup();
    //处理read&write事件
    NioEventLoopGroup workerGroup = new NioEventLoopGroup();
    ServerBootstrap bootstrap = new ServerBootstrap();
    bootstrap.group(bossGroup, workerGroup);
    bootstrap.channel(NioServerSocketChannel.class);
    //netty初始化相关的handler
    bootstrap.childHandler((ChannelInitializer) (ch) -> {
            //打印日志，方便观察
            LOGGER.info("初始化连接渠道");
            //设计消息体
            //增加编解码器
            ch.pipeline().addLast(new ImMsgDecoder());
            ch.pipeline().addLast(new ImMsgEncoder());
            ch.pipeline().addLast(new ImServerCoreHandler());
            //设置这个netty处理handler
    });
    //基于JVM的钩子函数去实现优雅关闭
    Runtime.getRuntime().addShutdownHook(new Thread(() -> {
        bossGroup.shutdownGracefully();
        workerGroup.shutdownGracefully();
    }));
```