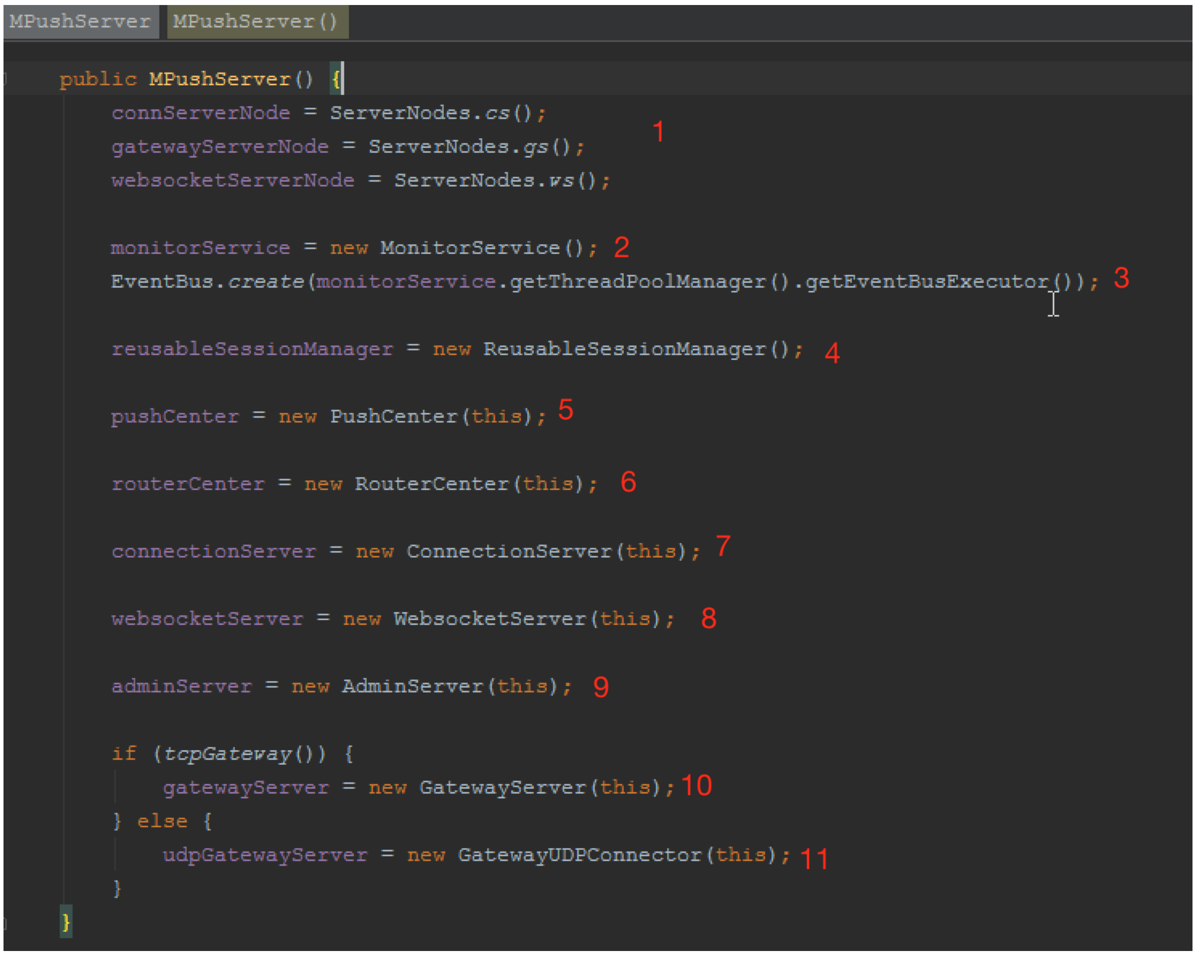

MPushServer MPushServer()

```
public MPushServer() {
    connServerNode = ServerNodes.cs();
    gatewayServerNode = ServerNodes.gs();          1
    websocketServerNode = ServerNodes.ws();

    monitorService = new MonitorService(); 2
    EventBus.create(monitorService.getThreadPoolManager().getEventBusExecutor()); 3

    reusableSessionManager = new ReusableSessionManager();  4

    pushCenter = new PushCenter(this); 5

    routerCenter = new RouterCenter(this);  6

    connectionServer = new ConnectionServer(this); 7

    websocketServer = new WebsocketServer(this);  8

    adminServer = new AdminServer(this);  9

    if (tcpGateway()) {
        gatewayServer = new GatewayServer(this);10
    } else {
        udpGatewayServer = new GatewayUDPConnector(this); 11
    }
}
```

1、初始化长连接、websocket服务、网关服务对象(暂时不清楚初始化这几个有什么作用)

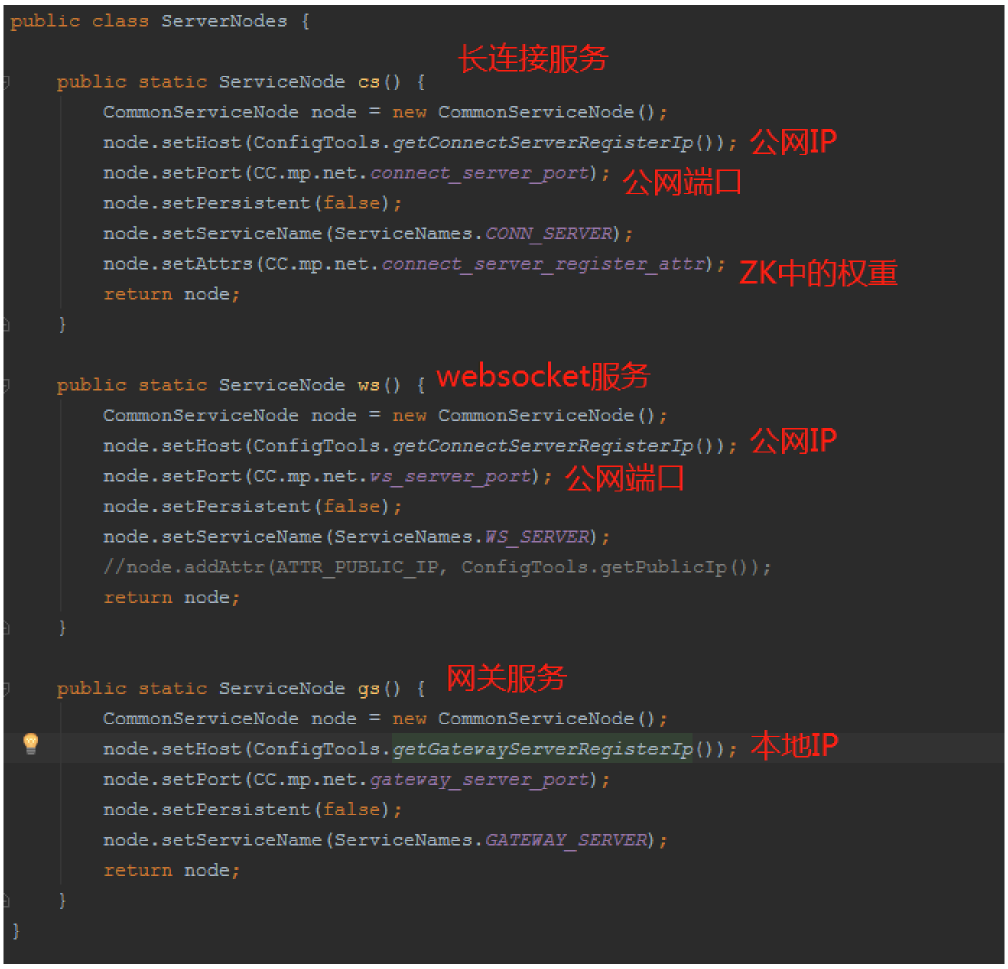
String CONN_SERVER = "/cluster/cs";

String WS_SERVER = "/cluster/ws";

String GATEWAY_SERVER = "/cluster/gs";

String DNS_MAPPING = "/dns/mapping";

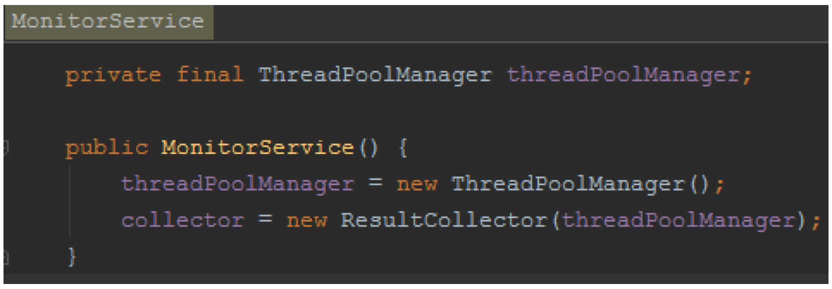
String ATTR_PUBLIC_IP = "public_ip";



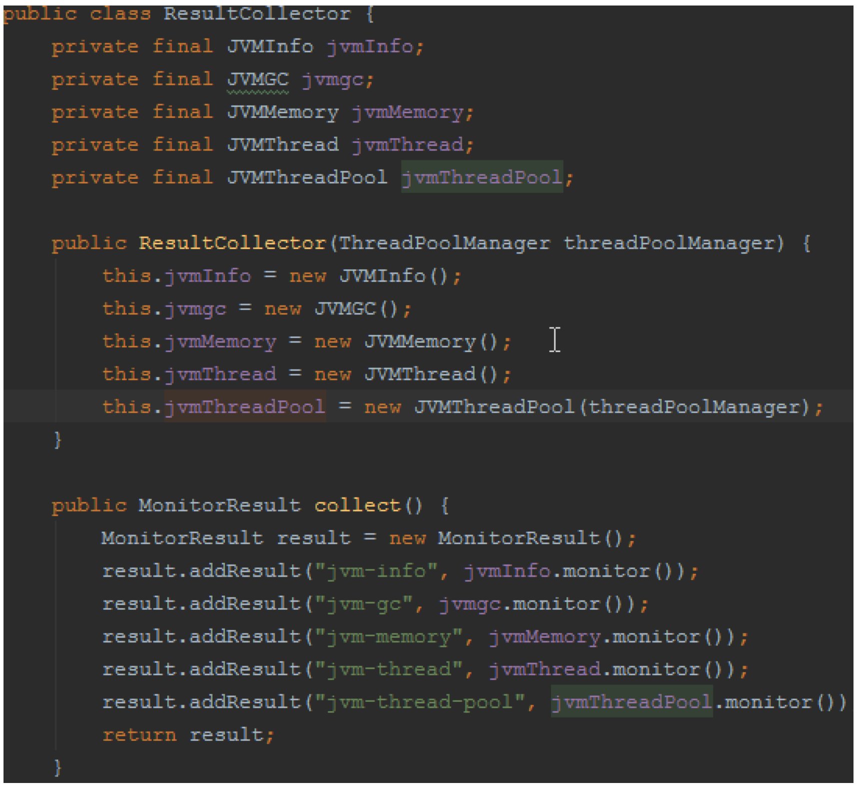
```java
public class ServerNodes {

    public static ServiceNode cs() {
        CommonServiceNode node = new CommonServiceNode();
        node.setHost(ConfigTools.getConnectServerRegisterIp());
        node.setPort(CC.mp.net.connect_server_port);
        node.setPersistent(false);
        node.setServiceName(ServiceNames.CONN_SERVER);
        node.setAttrs(CC.mp.net.connect_server_register_attr);
        return node;
    }

    public static ServiceNode ws() {
        CommonServiceNode node = new CommonServiceNode();
        node.setHost(ConfigTools.getConnectServerRegisterIp());
        node.setPort(CC.mp.net.ws_server_port);
        node.setPersistent(false);
        node.setServiceName(ServiceNames.WS_SERVER);
        //node.addAttr(ATTR_PUBLIC_IP, ConfigTools.getPublicIp());
        return node;
    }

    public static ServiceNode gs() {
        CommonServiceNode node = new CommonServiceNode();
        node.setHost(ConfigTools.getGatewayServerRegisterIp());
        node.setPort(CC.mp.net.gateway_server_port);
        node.setPersistent(false);
        node.setServiceName(ServiceNames.GATEWAY_SERVER);
        return node;
    }
}
```

2、初始化监控服务

实例化ResultCollector对象，用来收集JVM、线程池信息

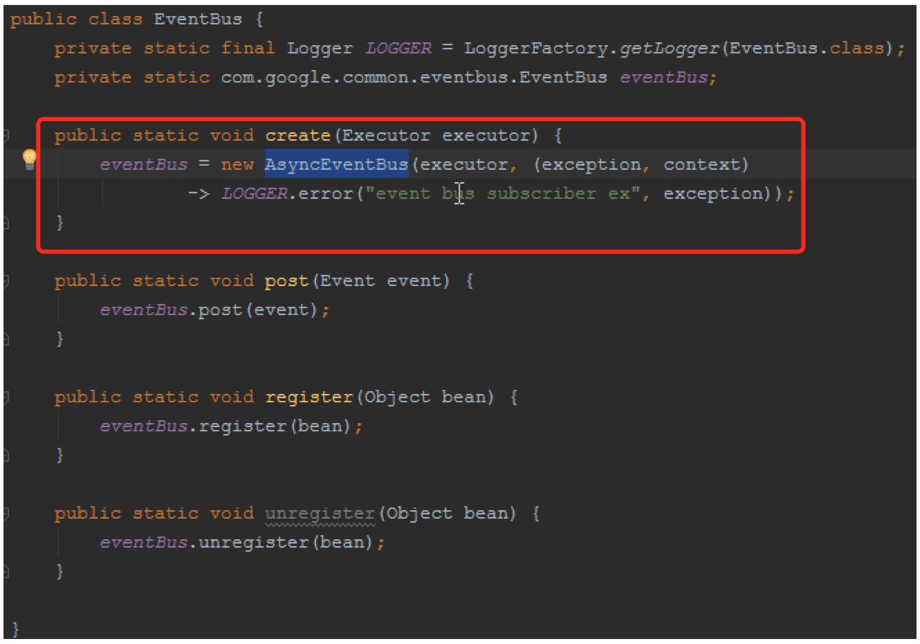


```java
MonitorService

    private final ThreadPoolManager threadPoolManager;

    public MonitorService() {
        threadPoolManager = new ThreadPoolManager();
        collector = new ResultCollector(threadPoolManager);
    }
```



```java
public class ResultCollector {
    private final JVMInfo jvmInfo;
    private final JVMGC jvmgc;
    private final JVMMemory jvmMemory;
    private final JVMThread jvmThread;
    private final JVMThreadPool jvmThreadPool;

    public ResultCollector(ThreadPoolManager threadPoolManager) {
        this.jvmInfo = new JVMInfo();
        this.jvmgc = new JVMGC();
        this.jvmMemory = new JVMMemory();
        this.jvmThread = new JVMThread();
        this.jvmThreadPool = new JVMThreadPool(threadPoolManager);
    }

    public MonitorResult collect() {
        MonitorResult result = new MonitorResult();
        result.addResult("jvm-info", jvmInfo.monitor());
        result.addResult("jvm-gc", jvmgc.monitor());
        result.addResult("jvm-memory", jvmMemory.monitor());
        result.addResult("jvm-thread", jvmThread.monitor());
        result.addResult("jvm-thread-pool", jvmThreadPool.monitor())
        return result;
    }
```

3、初始化事件总线AsyncEventBus

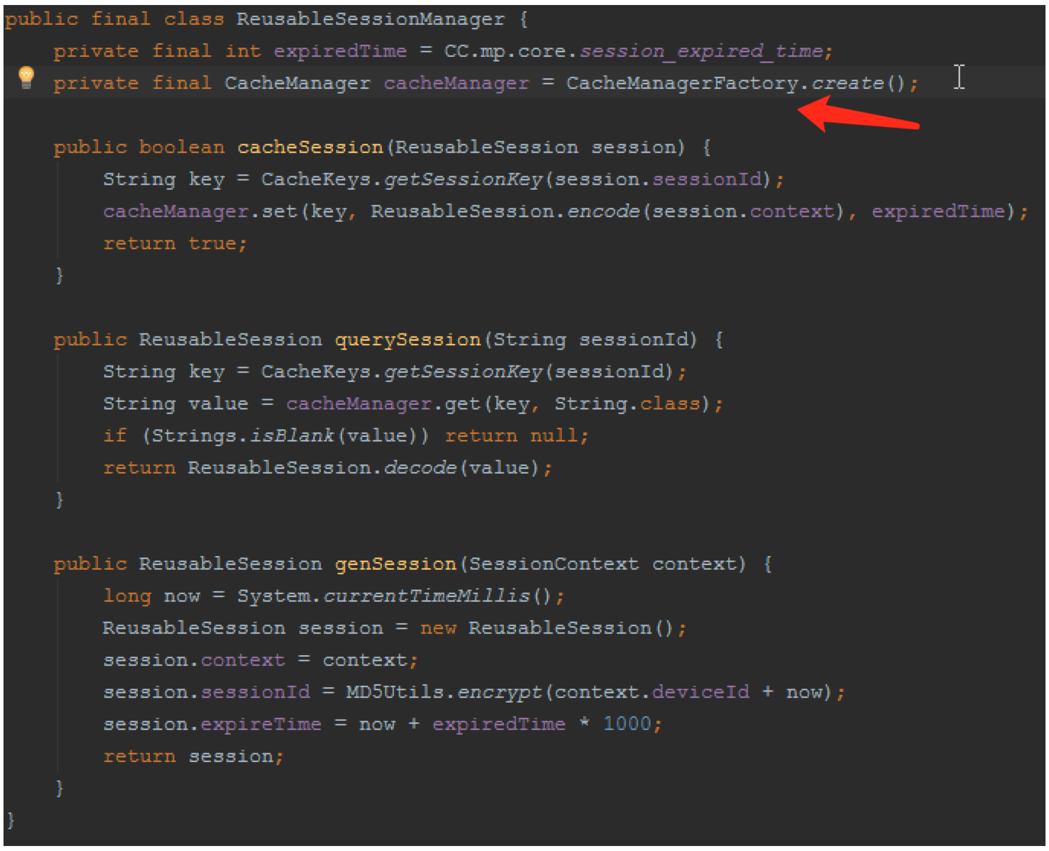


```java
public class EventBus {
    private static final Logger LOGGER = LoggerFactory.getLogger(EventBus.class);
    private static com.google.common.eventbus.EventBus eventBus;

    public static void create(Executor executor) {
        eventBus = new AsyncEventBus(executor, (exception, context)
                -> LOGGER.error("event bus subscriber ex", exception));
    }

    public static void post(Event event) {
        eventBus.post(event);
    }

    public static void register(Object bean) {
        eventBus.register(bean);
    }

    public static void unregister(Object bean) {
        eventBus.unregister(bean);
    }

}
```

4、初始化session管理器，初始化缓存实例(redis)

该管理器主要功能是用redis管理session；

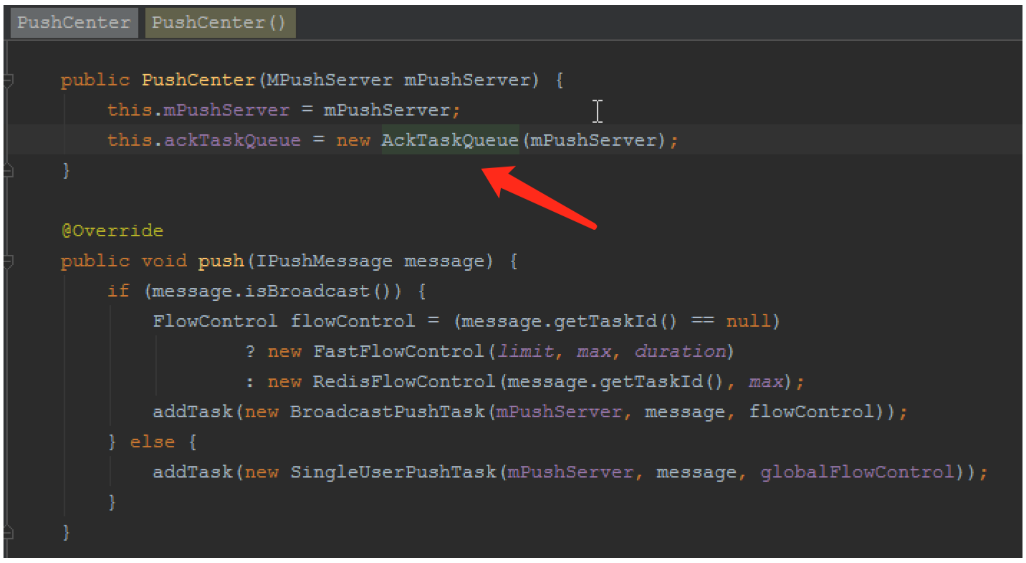
根据SPI，找到CacheManagerFactory的实现RedisCacheManagerFactory，得到RedisManager实例；



```java
public final class ReusableSessionManager {
    private final int expiredTime = CC.mp.core.session_expired_time;
    private final CacheManager cacheManager = CacheManagerFactory.create();

    public boolean cacheSession(ReusableSession session) {
        String key = CacheKeys.getSessionKey(session.sessionId);
        cacheManager.set(key, ReusableSession.encode(session.context), expiredTime);
        return true;
    }

    public ReusableSession querySession(String sessionId) {
        String key = CacheKeys.getSessionKey(sessionId);
        String value = cacheManager.get(key, String.class);
        if (Strings.isBlank(value)) return null;
        return ReusableSession.decode(value);
    }

    public ReusableSession genSession(SessionContext context) {
        long now = System.currentTimeMillis();
        ReusableSession session = new ReusableSession();
        session.context = context;
        session.sessionId = MD5Utils.encrypt(context.deviceId + now);
        session.expireTime = now + expiredTime * 1000;
        return session;
    }
}
```

5、初始化推送服务，初始化ackTaskQueue

ackTaskQueue主要用于异步执行ACK任务；

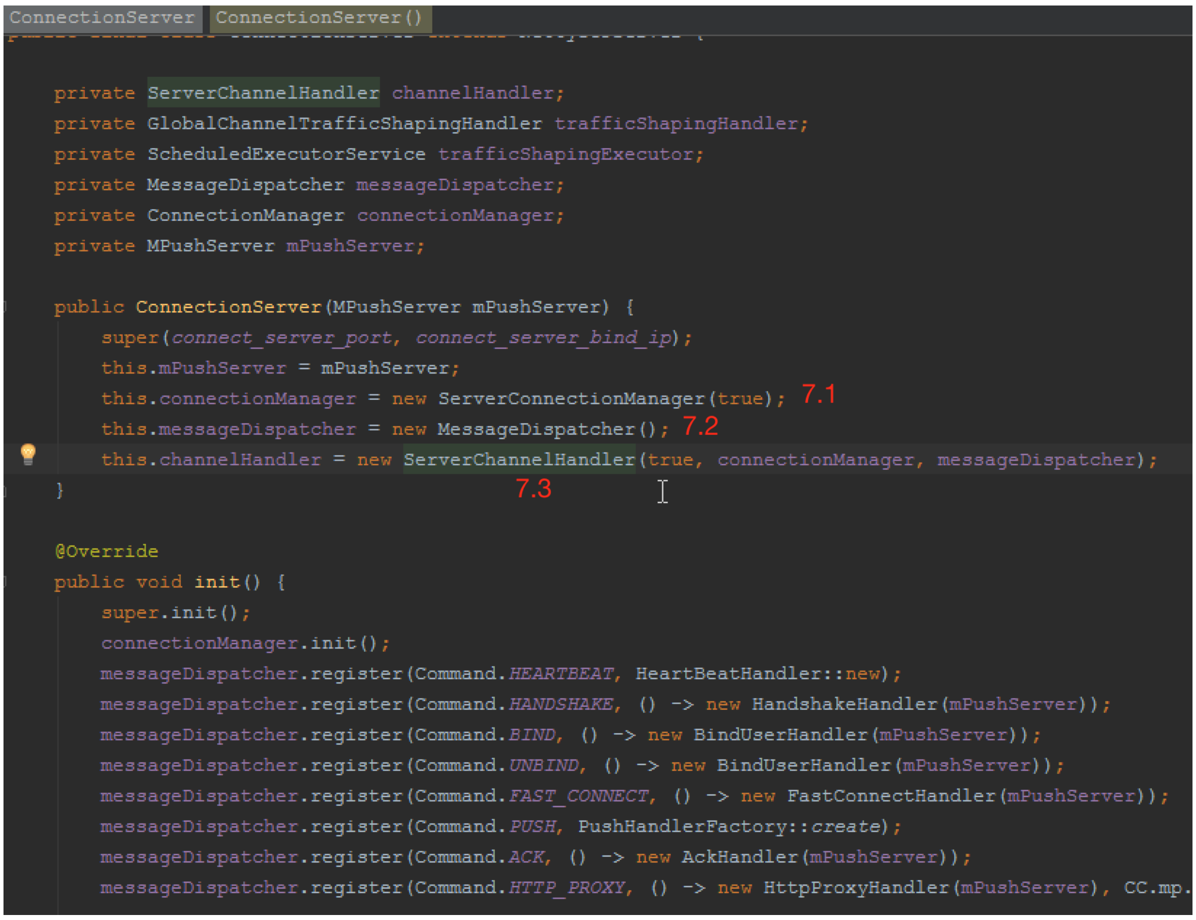


```java
    public PushCenter(MPushServer mPushServer) {
        this.mPushServer = mPushServer;
        this.ackTaskQueue = new AckTaskQueue(mPushServer);
    }

    @Override
    public void push(IPushMessage message) {
        if (message.isBroadcast()) {
            FlowControl flowControl = (message.getTaskId() == null)
                    ? new FastFlowControl(limit, max, duration)
                    : new RedisFlowControl(message.getTaskId(), max);
            addTask(new BroadcastPushTask(mPushServer, message, flowControl));
        } else {
            addTask(new SingleUserPushTask(mPushServer, message, globalFlowControl));
        }
    }
```

6、初始化路由服务

7、初始化长连接服务(这里比较重要)

初始化Netty conn连接管理器、消息转发器、服务处理器



```
    private ServerChannelHandler channelHandler;
    private GlobalChannelTrafficShapingHandler trafficShapingHandler;
    private ScheduledExecutorService trafficShapingExecutor;
    private MessageDispatcher messageDispatcher;
    private ConnectionManager connectionManager;
    private MPushServer mPushServer;

    public ConnectionServer(MPushServer mPushServer) {
        super(connect_server_port, connect_server_bind_ip);
        this.mPushServer = mPushServer;
        this.connectionManager = new ServerConnectionManager(true); 7.1
        this.messageDispatcher = new MessageDispatcher(); 7.2
        this.channelHandler = new ServerChannelHandler(true, connectionManager, messageDispatcher);
    }                                   7.3

    @Override
    public void init() {
        super.init();
        connectionManager.init();
        messageDispatcher.register(Command.HEARTBEAT, HeartBeatHandler::new);
        messageDispatcher.register(Command.HANDSHAKE, () -> new HandshakeHandler(mPushServer));
        messageDispatcher.register(Command.BIND, () -> new BindUserHandler(mPushServer));
        messageDispatcher.register(Command.UNBIND, () -> new BindUserHandler(mPushServer));
        messageDispatcher.register(Command.FAST_CONNECT, () -> new FastConnectHandler(mPushServer));
        messageDispatcher.register(Command.PUSH, PushHandlerFactory::create);
        messageDispatcher.register(Command.ACK, () -> new AckHandler(mPushServer));
        messageDispatcher.register(Command.HTTP_PROXY, () -> new HttpProxyHandler(mPushServer), CC.mp.
```

7.1 初始化Netty conn连接管理器(添加、获取、关闭连接)

ServerConnectionManager(true) ，每建立一个连接保存到缓存中，并且添加netty 连接超时检测，用的是netty自带的HashedWheelTimer；检测到超时，则关闭超时连接；

何时添加连接？

建立连接，ServerChannelHandler#channelActive被调用时；

何时获取连接？

读取消息事件，ServerChannelHandler#channelRead被调用时；

异常事件，ServerChannelHandler#exceptionCaught被调用时；

何时关闭连接？

断开连接事件，ServerChannelHandler#channelInactive被调用时；

7.2 初始化消息转发器

MessageDispatcher：消息分发处理，提供各种消息的处理注册；

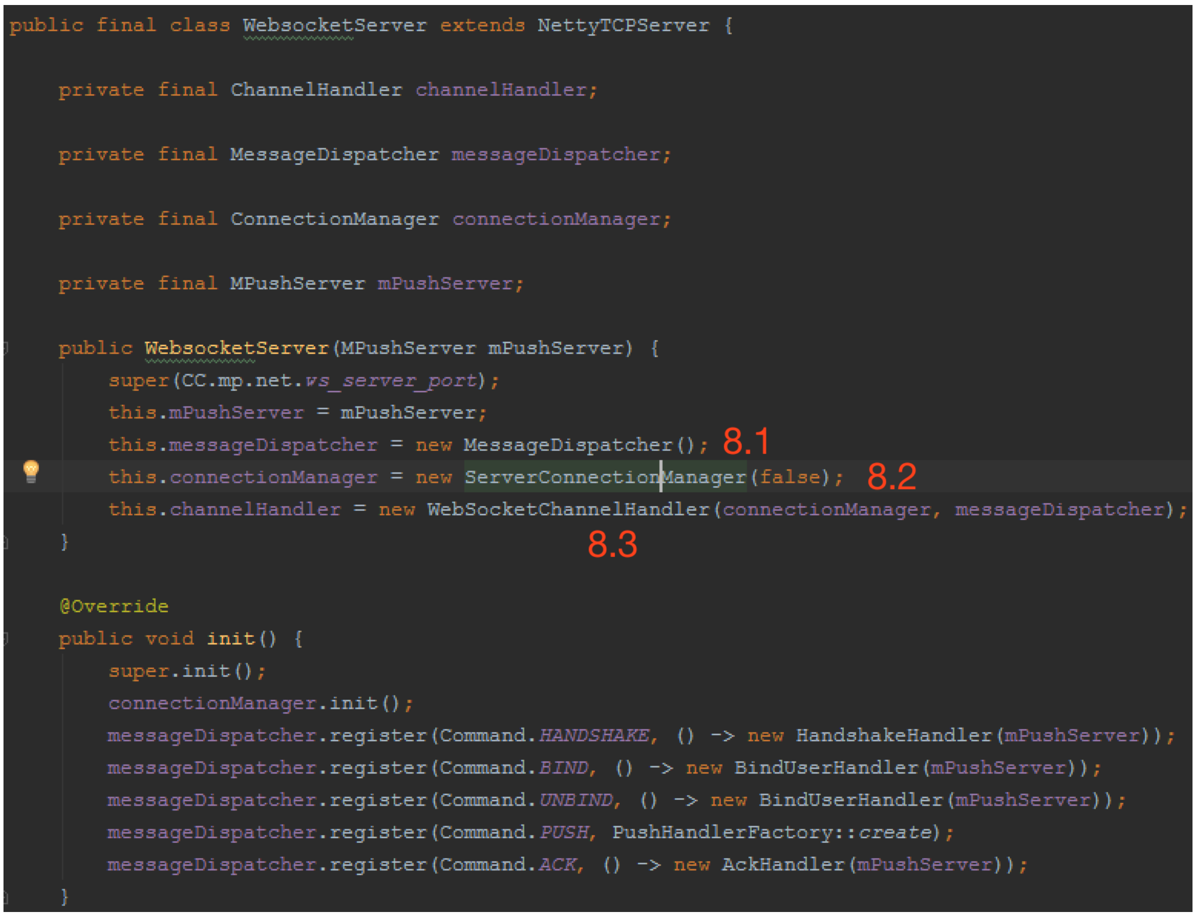
- 心跳
- 握手
- 用户绑定
- 解绑
- 快速连接
- 推送
- 消息确认

- 代理

7.3 初始化长连接服务处理器

ServerChannelHandler：Netty各种事件处理类；

8、websocket服务



```java
public final class WebsocketServer extends NettyTCPServer {

    private final ChannelHandler channelHandler;

    private final MessageDispatcher messageDispatcher;

    private final ConnectionManager connectionManager;

    private final MPushServer mPushServer;

    public WebsocketServer(MPushServer mPushServer) {
        super(CC.mp.net.ws_server_port);
        this.mPushServer = mPushServer;
        this.messageDispatcher = new MessageDispatcher(); 8.1
        this.connectionManager = new ServerConnectionManager(false); 8.2
        this.channelHandler = new WebSocketChannelHandler(connectionManager, messageDispatcher);
    }                                       8.3

    @Override
    public void init() {
        super.init();
        connectionManager.init();
        messageDispatcher.register(Command.HANDSHAKE, () -> new HandshakeHandler(mPushServer));
        messageDispatcher.register(Command.BIND, () -> new BindUserHandler(mPushServer));
        messageDispatcher.register(Command.UNBIND, () -> new BindUserHandler(mPushServer));
        messageDispatcher.register(Command.PUSH, PushHandlerFactory::create);
        messageDispatcher.register(Command.ACK, () -> new AckHandler(mPushServer));
    }
```

8.1 初始化消息转发器

MessageDispatcher：消息分发处理，提供各种消息的处理注册；

- 握手
- 用户绑定
- 解绑
- 推送
- 消息确认

8.2 初始化Netty conn连接管理器(添加、获取、关闭连接)

ServerConnectionManager(false) ，每建立一个连接保存到缓存中;

何时添加连接？

建立连接，WebSocketChannelHandler#channelActive被调用时；

何时获取连接？

读取消息事件，WebSocketChannelHandler#channelRead0被调用时；

异常事件，WebSocketChannelHandler#exceptionCaught被调用时；

何时关闭连接？

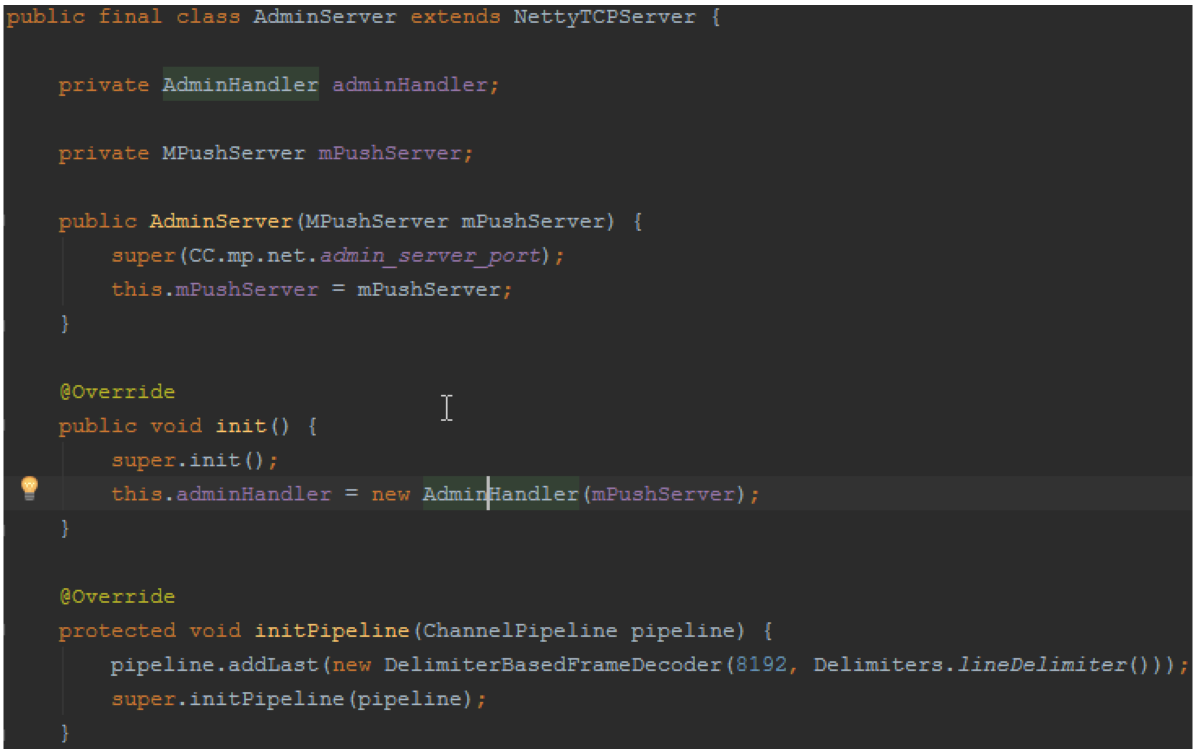
断开连接事件，WebSocketChannelHandler#channelInactive被调用时；

8.3 初始化长连接服务处理器

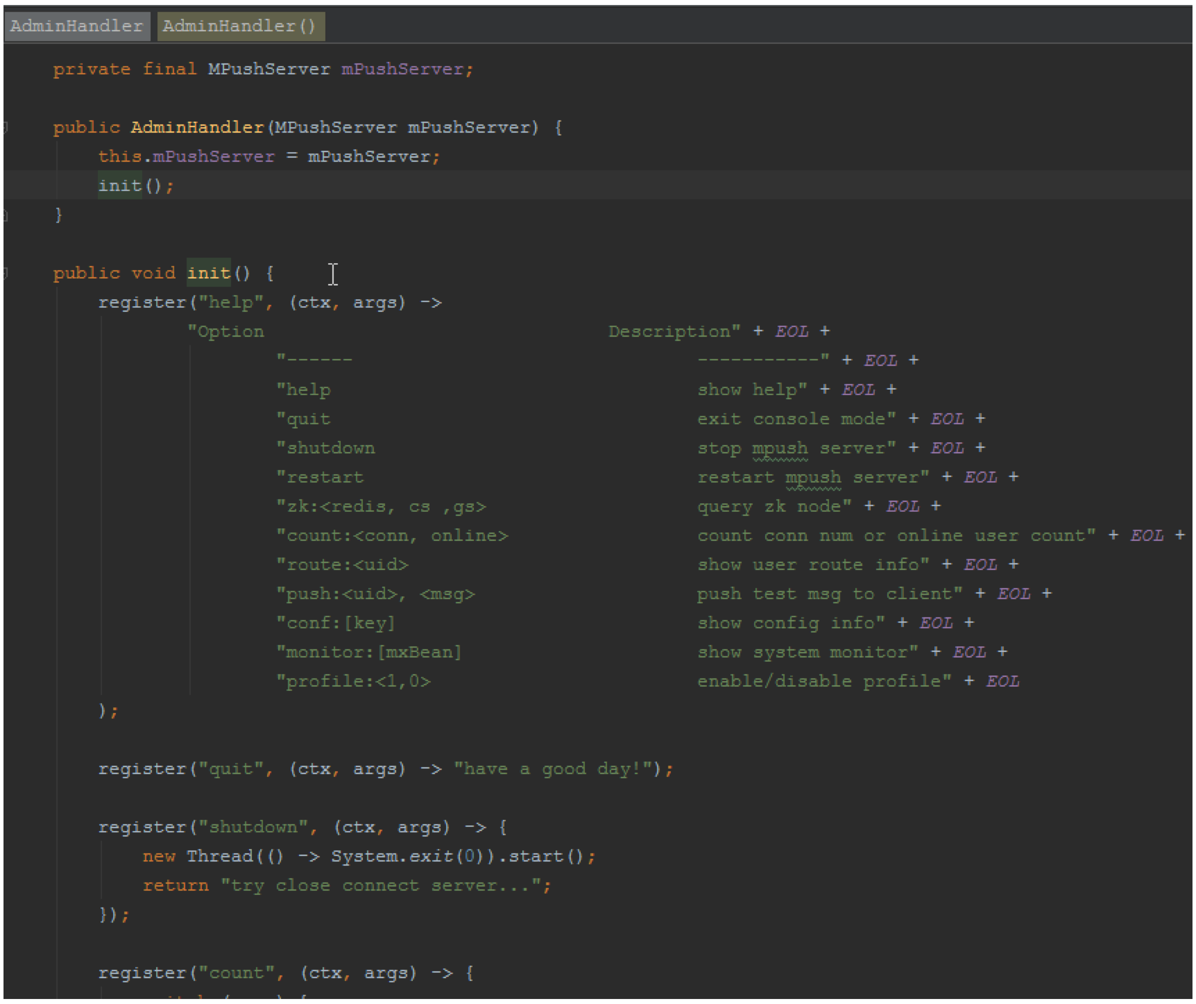
WebSocketChannelHandler：Netty各种事件处理类；

9、初始化admin服务

提供服务命令行，可以查询、操作服务；

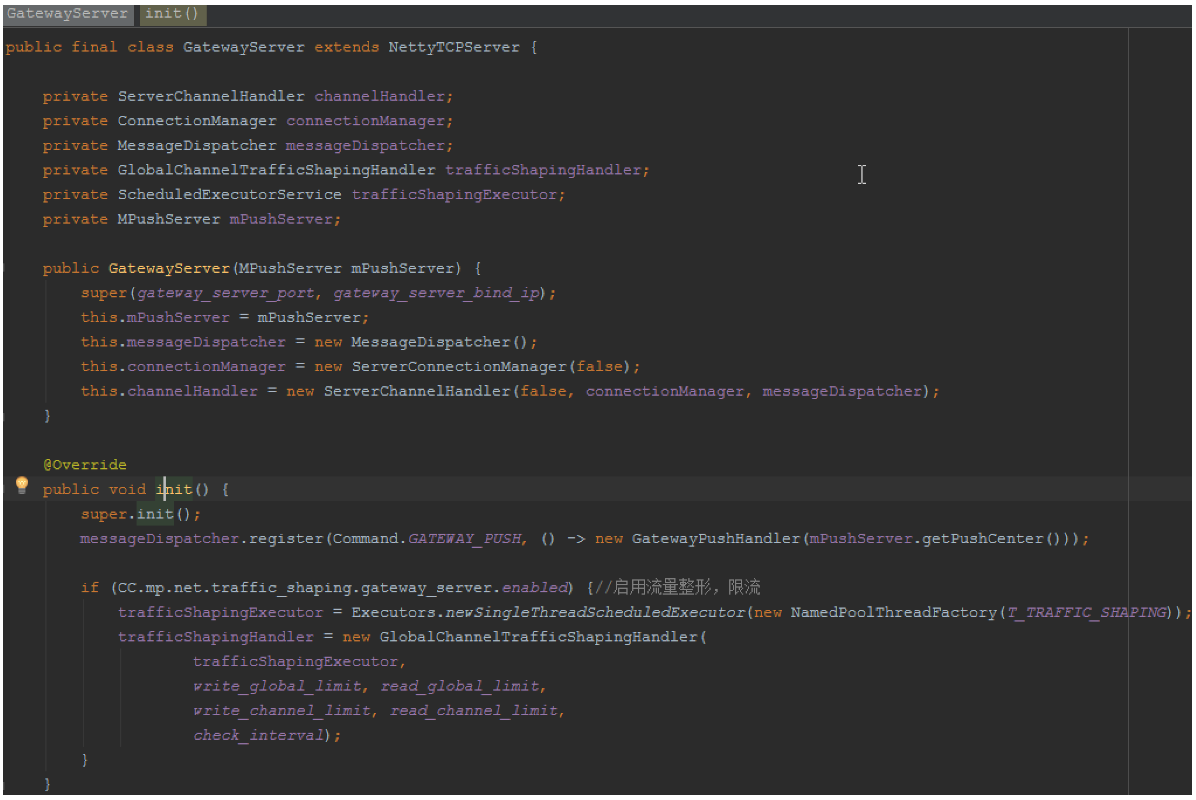


```java
public final class AdminServer extends NettyTCPServer {

    private AdminHandler adminHandler;

    private MPushServer mPushServer;

    public AdminServer(MPushServer mPushServer) {
        super(CC.mp.net.admin_server_port);
        this.mPushServer = mPushServer;
    }

    @Override
    public void init() {
        super.init();
        this.adminHandler = new AdminHandler(mPushServer);
    }

    @Override
    protected void initPipeline(ChannelPipeline pipeline) {
        pipeline.addLast(new DelimiterBasedFrameDecoder(8192, Delimiters.lineDelimiter()));
        super.initPipeline(pipeline);
    }
```



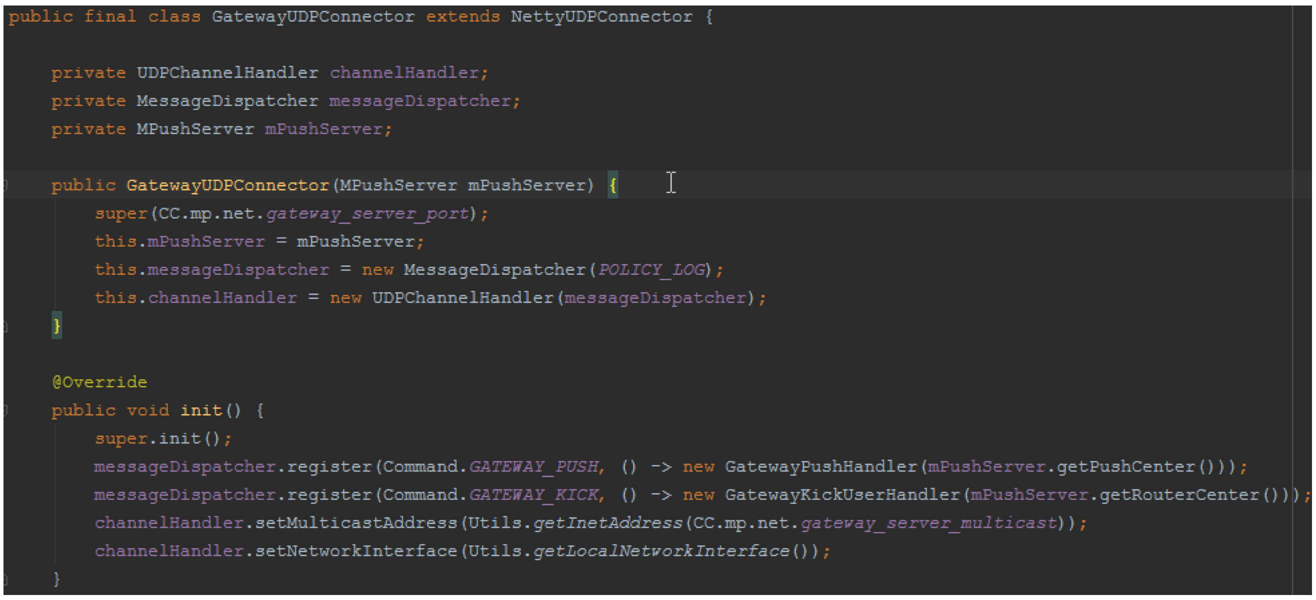
```java
    private final MPushServer mPushServer;

    public AdminHandler(MPushServer mPushServer) {
        this.mPushServer = mPushServer;
        init();
    }

    public void init() {
        register("help", (ctx, args) ->
                "Option                                  Description" + EOL +
                        "------                                  -----------" + EOL +
                        "help                                    show help" + EOL +
                        "quit                                    exit console mode" + EOL +
                        "shutdown                                stop mpush server" + EOL +
                        "restart                                 restart mpush server" + EOL +
                        "zk:<redis, cs ,gs>                      query zk node" + EOL +
                        "count:<conn, online>                    count conn num or online user count" + EOL +
                        "route:<uid>                             show user route info" + EOL +
                        "push:<uid>, <msg>                       push test msg to client" + EOL +
                        "conf:[key]                              show config info" + EOL +
                        "monitor:[mxBean]                        show system monitor" + EOL +
                        "profile:<1,0>                           enable/disable profile" + EOL
        );

        register("quit", (ctx, args) -> "have a good day!");

        register("shutdown", (ctx, args) -> {
            new Thread(() -> System.exit(0)).start();
            return "try close connect server...";
        });

        register("count", (ctx, args) -> {
```

10、初始化网关服务



```java
public final class GatewayServer extends NettyTCPServer {

    private ServerChannelHandler channelHandler;
    private ConnectionManager connectionManager;
    private MessageDispatcher messageDispatcher;
    private GlobalChannelTrafficShapingHandler trafficShapingHandler;
    private ScheduledExecutorService trafficShapingExecutor;
    private MPushServer mPushServer;

    public GatewayServer(MPushServer mPushServer) {
        super(gateway_server_port, gateway_server_bind_ip);
        this.mPushServer = mPushServer;
        this.messageDispatcher = new MessageDispatcher();
        this.connectionManager = new ServerConnectionManager(false);
        this.channelHandler = new ServerChannelHandler(false, connectionManager, messageDispatcher);
    }

    @Override
    public void init() {
        super.init();
        messageDispatcher.register(Command.GATEWAY_PUSH, () -> new GatewayPushHandler(mPushServer.getPushCenter()));

        if (CC.mp.net.traffic_shaping.gateway_server.enabled) {//启用流量整形，限流
            trafficShapingExecutor = Executors.newSingleThreadScheduledExecutor(new NamedPoolThreadFactory(T_TRAFFIC_SHAPING));
            trafficShapingHandler = new GlobalChannelTrafficShapingHandler(
                    trafficShapingExecutor,
                    write_global_limit, read_global_limit,
                    write_channel_limit, read_channel_limit,
                    check_interval);
        }
    }
```

11、初始化UDP网关服务



```java
public final class GatewayUDPConnector extends NettyUDPConnector {

    private UDPChannelHandler channelHandler;
    private MessageDispatcher messageDispatcher;
    private MPushServer mPushServer;

    public GatewayUDPConnector(MPushServer mPushServer) {
        super(CC.mp.net.gateway_server_port);
        this.mPushServer = mPushServer;
        this.messageDispatcher = new MessageDispatcher(POLICY_LOG);
        this.channelHandler = new UDPChannelHandler(messageDispatcher);
    }

    @Override
    public void init() {
        super.init();
        messageDispatcher.register(Command.GATEWAY_PUSH, () -> new GatewayPushHandler(mPushServer.getPushCenter()));
        messageDispatcher.register(Command.GATEWAY_KICK, () -> new GatewayKickUserHandler(mPushServer.getRouterCenter()));
        channelHandler.setMulticastAddress(Utils.getInetAddress(CC.mp.net.gateway_server_multicast));
        channelHandler.setNetworkInterface(Utils.getLocalNetworkInterface());
    }
```