作用：提供后台命令行

- shutdown停止服务
- restart重启服务
- zk查询ZK节点信息
- count统计连接数、在线用户数
- route查询用户路由信息
- push推送测试消息到客户端(手机)
- conf查询配置
- monitor查询监控信息
- profile禁用启用

初始化控制台服务

```
chain.boot()
        .setNext(new CacheManagerBoot())//1.初始化缓存模块
        .setNext(new ServiceRegistryBoot())//2.启动服务注册与发现模块
        .setNext(new ServiceDiscoveryBoot())//2.启动服务注册与发现模块
        .setNext(new ServerBoot(mPushServer.getConnectionServer(), mPushServer.getConnServerNode()))//3.启动接入服务
        .setNext(() -> new ServerBoot(mPushServer.getWebsocketServer(), mPushServer.getWebsocketServerNode()), wsEnab
        .setNext(() -> new ServerBoot(mPushServer.getUdpGatewayServer(), mPushServer.getGatewayServerNode()), udpGate
        .setNext(() -> new ServerBoot(mPushServer.getGatewayServer(), mPushServer.getGatewayServerNode()), tcpGateway
        .setNext(new ServerBoot(mPushServer.getAdminServer(), null))//7.启动控制台服务
        .setNext(new RouterCenterBoot(mPushServer))//8.启动路由中心组件
        .setNext(new PushCenterBoot(mPushServer))//9.启动推送中心组件
        .setNext(() -> new HttpProxyBoot(mPushServer), CC.mp.http.proxy_enabled)//10.启动http代理服务，dns解析服务
        .setNext(new MonitorBoot(mPushServer))//11.启动监控服务
        .end();
```

服务启动

```
ServerBoot  start()
    @Override
    public void start() {
        server.init();
        server.start(new Listener() {
            @Override
            public void onSuccess(Object... args) {
                Logs.Console.info("start {} success on:{}", server.getClass().getSimpleName(), args[0]);
                if (node != null) {//注册应用到zk
                    ServiceRegistryFactory.create().register(node);
                    Logs.RSD.info("register {} to srd success.", node);
                }
                startNext();
            }

            @Override
            public void onFailure(Throwable cause) {
                Logs.Console.error("start {} failure, jvm exit with code -1", server.getClass().getSimpleNa
                System.exit(-1);
            }
        });
    }

    @Override
    protected void stop() {
        stopNext();
        if (node != null) {
            ServiceRegistryFactory.create().deregister(node);
        }
        Logs.Console.info("try shutdown {}...", server.getClass().getSimpleName());
        server.stop().join();
        Logs.Console.info("{} shutdown success.", server.getClass().getSimpleName());
    }
```

1、调用AdminServer#init()

2、调用AdminServer的父类NettyTCPServer#start()

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

    @Override
    public void init() {
        super.init();  1
        this.adminHandler = new AdminHandler(mPushServer);  2
    }

    @Override
    protected void initPipeline(ChannelPipeline pipeline) {  3
        pipeline.addLast(new DelimiterBasedFrameDecoder(8192, Delimiters.lineDelimiter()));
        super.initPipeline(pipeline);
    }

    @Override
    public ChannelHandler getChannelHandler() {
        return adminHandler;
    }
```

1 调用NettyTCPServer#init()

主要是利用AtomicReference<State>的cas判断netty服务是不是启动，如果已经启动则抛出ServiceException异常；

1.2 调用ServerConnectionManager#init()

这里主要是正常连接管理，该方法里面不做任何事情；

1.3 注册推送消息的处理类

2 调用NettyTCPServer#start()方法，创建ServerBootstrap启动Netty长连接服务；

NettyTCPServer创建netty ServerBootstrap服务时，会调用其子类GatewayServer中方法：

initPipeline()：在已有的Pipeline最前面加入换行解码器DelimiterBasedFrameDecoder；

getChannelHandler(): 设置netty 事件处理类AdminHandler，处理建连、消息、断连、异常事件；