

读书笔记 5 NameServer 源码解析

源码结构图



一个 NameServer 的启动过程：

```
public class NamesrvStartup {

    private static InternalLogger log;
    private static Properties properties = null;
    private static CommandLine commandLine = null;

    public static void main(String[] args) { main0(args); }

    public static NamesrvController main0(String[] args) {
        try {
            NamesrvController controller = createNamesrvController(args);
            start(controller);
            String tip = "The Name Server boot success. serializeType=" + RemotingCommand.getSerializeTypeInfini();
            log.info(tip);
        } catch (Exception e) {
            log.error("Name Server boot failed.", e);
            System.exit(-1);
        }
    }
}
```

```

public static NamesrvController createNamesrvController(String[] args) throws IOException, JoranException {
    System.setProperty(RemotingCommand.REMOTING_VERSION_KEY, Integer.toString(MQVersion.CURRENT_VERSION));
    //PackageConflictDetect.detectFastjson();

```

```

    Options options = ServerUtil.buildCommandLineOptions(new Options());
    CommandLine cmdLine = ServerUtil.parseCmdLine("mqnamesrv", args, buildCommandLineOptions(options), new PosixParser());

```

```

    if (null == cmdLine) {
        System.exit(status: -1);
        return null;
    }

```

主要功能，解析命令行参数
&
初始化 controller

```

    final NamesrvConfig namesrvConfig = new NamesrvConfig();
    final NettyServerConfig nettyServerConfig = new NettyServerConfig();
    nettyServerConfig.setListenPort(9876);

```

```

    if (cmdLine.hasOption('c')) {  // 命令行指定配置文件位置

```

```

        String file = cmdLine.getOptionValue('c');
        if (file != null) {
            InputStream in = new BufferedInputStream(new FileInputStream(file));
            Properties properties = new Properties();
            properties.load(in);
            MixAll.properties2Object(properties, namesrvConfig);
            MixAll.properties2Object(properties, nettyServerConfig);

            namesrvConfig.setConfigStorePath(file);

            System.out.printf("load config properties file OK, %s\n", file);
            in.close();
        }
    }

```

```

    if (cmdLine.hasOption('p')) {  // 打印所有配置项的值
        InternalLogger console = InternalLoggerFactory.getLogger(LoggerName.NAMESRV_CONSOLE_NAME);
        MixAll.printObjectProperties(console, namesrvConfig);
        MixAll.printObjectProperties(console, nettyServerConfig);
        System.exit(status: 0);
    }

```

```

    MixAll.properties2Object(ServerUtil.commandLine2Properties(cmdLine), namesrvConfig);

```

```

public static NamesrvController start(final NamesrvController controller) throws Exception {

```

```

    if (null == controller) {
        throw new IllegalArgumentException("NamesrvController is null");
    }

```

```

    boolean initResult = controller.initialize();  // 初始化

```

```

    if (!initResult) {
        controller.shutdown();
        System.exit(status: -3);
    }

```

```

    Runtime.getRuntime().addShutdownHook(new ShutdownHookThread(log, new Callable<Void>() {
        @Override
        public Void call() throws Exception {
            controller.shutdown();
            return null;
        }
    }));

```

```

    controller.start();  // 开始调用服务

```

```

    return controller;
}

```

```
NamesrvController.java x ServiceThread.java x Thread.java x RemotingService.java x NettyRemotingServer.java
log,
this.namesrvConfig, this.nettyServerConfig
);
this.configuration.setStorePathFromConfig(this.namesrvConfig, fieldName: "configStorePath");
}

public boolean initialize() {
    this.kvConfigManager.load(); // 倒入 kv config 配置信息

    this.remotingServer = new NettyRemotingServer(this.nettyServerConfig, this.brokerHousekeepingService); // 初始化 netty 通信模块

    this.remotingExecutor =
        Executors.newFixedThreadPool(nettyServerConfig.getServerWorkerThreads(), new ThreadFactoryImpl( threadNamePrefix: "RemotingExecutorThread_"));
    // 创建默认 8 个线程的线程池

    this.registerProcessor(); // 根据请求调用不同 processor 处理

    this.scheduledExecutorService.scheduleAtFixedRate(new Runnable() {
        @Override
        public void run() { NamesrvController.this.routeInfoManager.scanNotActiveBroker(); }
    }, initialDelay: 5, period: 10, TimeUnit.SECONDS);
    // 扫描非活跃节点的定时线程

    this.scheduledExecutorService.scheduleAtFixedRate(new Runnable() {
        @Override
        public void run() { NamesrvController.this.kvConfigManager.printAllPeriodically(); }
    }, initialDelay: 1, period: 10, TimeUnit.MINUTES);
    // 打印配置的定时线程

    if (TlsSystemConfig.tlsMode != TlsMode.DISABLED) {
        // Register a listener to reload SslContext
        try {
            fileWatchService = new FileWatchService(
                new String[] {
                    TlsSystemConfig.tlsServerCertPath,
                    TlsSystemConfig.tlsServerKeyPath,
                    TlsSystemConfig.tlsServerTrustCertPath
                }
            );
        } catch (Exception e) {
            log.error("Failed to register ssl context listener", e);
        }
    }
}
```

```
private void registerProcessor() {
    if (namesrvConfig.isClusterTest()) {
        this.remotingServer.registerDefaultProcessor(new ClusterTestRequestProcessor( namesrvController: this, namesrvConfig.getProductEnvName()),
            this.remotingExecutor);
    } else {
        this.remotingServer.registerDefaultProcessor(new DefaultRequestProcessor( namesrvController: this), this.remotingExecutor);
    }
}
```

```
@Override
public void registerDefaultProcessor(NettyRequestProcessor processor, ExecutorService executor) {
    this.defaultRequestProcessor = new Pair(NettyRequestProcessor, ExecutorService>(processor, executor);
}
```

@Override

```
25 public interface NettyRequestProcessor {
26     RemotingCommand processRequest(ChannelHandlerContext ctx, RemotingCommand request)
27     throws Exception;
28
29     boolean rejectRequest();
30
31 }
```

```

public class DefaultRequestProcessor extends AsyncNettyRequestProcessor implements NettyRequestProcessor {
    private static InternalLogger log = InternalLoggerFactory.getLogger(LoggerName.NAMESRV_LOGGER_NAME);

    protected final NamesrvController namesrvController;

    public DefaultRequestProcessor(NamesrvController namesrvController) { this.namesrvController = namesrvController; }

    @Override
    public RemotingCommand processRequest(ChannelHandlerContext ctx,
        RemotingCommand request) throws RemotingCommandException {

        if (ctx != null) {
            log.debug("receive request, {} {} {}",
                request.getCode(),
                RemotingHelper.parseChannelRemoteAddr(ctx.channel()),
                request);
        }

        switch (request.getCode()) {
            case RequestCode.PUT_KV_CONFIG:
                return this.putKVConfig(ctx, request);
            case RequestCode.GET_KV_CONFIG:

```

例如这里，实现了 NettyRequestProcessor，上面初始化的时候生成了一个类型的实例，最后走到类似的逻辑，主体是一个 switch 根据 RequestCode 取调不同的函数实现

RocketMq 集群存储状态：

NameServer 协调保存各种数据，通过 RouteInfoManager 实现

```

public void deleteTopic(final String topic) {
    try {
        try {
            this.lock.writeLock().lockInterruptibly();
            this.topicQueueTable.remove(topic);
        } finally {
            this.lock.writeLock().unlock();
        }
    } catch (Exception e) {
        log.error("deleteTopic Exception", e);
    }
}

public byte[] getAllTopicList() {
    TopicList topicList = new TopicList();
    try {
        try {
            this.lock.readLock().lockInterruptibly();
            topicList.getTopicList().addAll(this.topicQueueTable.keySet());
        } finally {
            this.lock.readLock().unlock();
        }
    } catch (Exception e) {
        log.error("getAllTopicList Exception", e);
    }

    return topicList.encode();
}

public RegisterBrokerResult registerBroker(

```

可重入的读写锁

