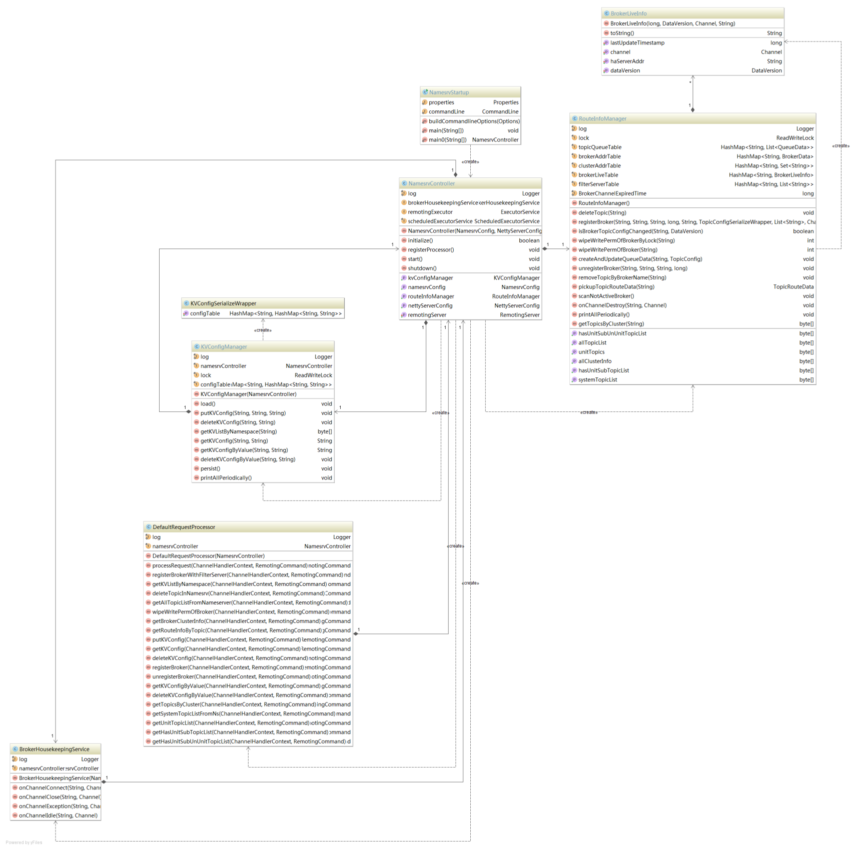
原 Rocketmq之namesrv启动流程源码详解分析

tantexian 发表于 4个月前 阅读 260 收藏 2 点赞 0 评论 0 收藏

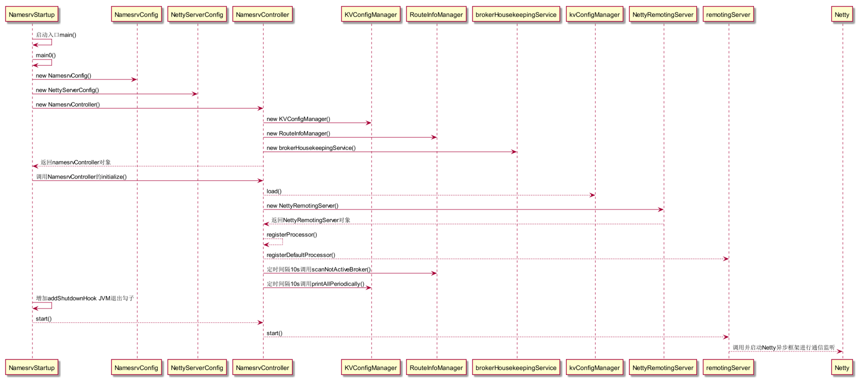
云笔记版本地址：http://note.youdao.com/yws/public/redirect/share?id=86ccd81cf192ebe8af5bf8742aa60c84&type=false

一、namesrv整体类图：

```
NamesrvStartup：namesrv启动的入口类。
NamesrvConfig：namesrv配置文件类
NettysystemConfig：Netty配置文件类设置监听接口及发送接收信息用于与其他模块的远程通信
NamesrvController：namesrv初始化服务控制对象（核心类）
RouteInfoManager：namesrv管理所有broker的路由及topic配置信息类
DefaultRequestProcessor：对netty的封装（封装通信交互协议），管理和启动netty网络通信框架
```



二、namesrvStartup时序图：


NamesrvStartup
NamesrvConfig
NettyServerConfig
NamesrvController
KVConfigManager
RouteInfoManager
brokerHousekeepingService
kvConfigManager
NettyRemotingServer
remotingServer
Netty
启动入口main()
main0()
new NamesrvConfig()
new NettyServerConfig()
new NamesrvController()
new KVConfigManager()
new RouteInfoManager()
new brokerHousekeepingService()
返回namesrvController对象
调用NamesrvController的initialize()
load()
new NettyRemotingServer()
返回NettyRemotingServer对象
registerProcessor()
registerDefaultProcessor()
定时间隔10s调用scanNotActiveBroker()
定时间隔10s调用printAllPeriodically()
增加addShutdownHook JVM退出勾子
start()
start()
调用并启动Netty异步框架进行通信监听

三、源码分析详解：

根据时序图来讲解namesrv启动流程（核心方法讲解）：

1、执行NamesrvStartup类的启动入口main方法，再调用main0

```
// 此处为Namesrv启动的入口函数 2016/7/6 Add by tantexixan
public static void main(String[] args) {
main0(args);// 调用main0函数
}
```

2、main0方法中获取NamesrvConfig、NettyServerConfig配置：

```
// 初始化配置文件
final NamesrvConfig namesrvConfig = new NamesrvConfig();// 加载namesrv的相关配置项
// 加载nettyServerConfig配置项( Netty是一个高性能、异步事件驱动的NIO框架，用来与其他模块通信交互，例如broker模块通信）
// netty具体的启动工作为后续函数controller.start();
final NettyServerConfig nettyServerConfig = new NettyServerConfig();
// 设置netty监听端口为9876
nettyServerConfig.setListenPort(9876);
```

3、创建NamesrvController对象，并执行initialize方法：

```
// 初始化服务控制对象（NamesrvController为核心类，保存了大量的namesrv需要的信息）
final NamesrvController controller = new NamesrvController(namesrvConfig, nettyServerConfig);
// initialize核心函数，其中主要作用为：
// 1、初始化netty相关配置
// 2、定义broker与namesrv通过netty进行通信，的通信协议（即请求中带上code，来代表对应调用哪个方法函数）
// 3、定时每10s扫描broker信息，如果过期则移除
// 4、定时每10s将configTable的信息记录到日志文件中
boolean initResult = controller.initialize();
if (!initResult) {
    controller.shutdown();
System.exit(-3);
}
```

3-1、跟进到NamesrvController构造函数：（创建KVConfigManager、RouteInfoManager、BrokerHousekeepingService对象）

Rocketmq之namesrv启动流程源码详解分析

tantexian 发表于4个月前

```
)
NettyServerConfig nettyServerConfig) {
关配置
/配置管理
信息、topi　息管理
Service(this); // broker管理服务
}
```

其中此处重点应该关注的是RouteInfoManager类：

3-1-1、跟进，详解RouteInfoManager的registerBroker方法：

```
/**
* broker向namesrv注册函数
* 主要功能步骤包括:
* 1、将当前请求注册的broker信息保存或者更新到clusterAddrTable、brokerAddrTable中
* 2、将当前请求注册的broker的topic信息，保存或者更新到topicQueueTable中
* -- 其中isBrokerTopicConfigChanged用来判断当前请求broker信息是否为最新版本，如果是则替换，不是则跳过
* -- createAndUpdateQueueData为具体觉得创建还是更新topicQueueTable
* -- 其中topicQueueTable中保存了对应topic的queueDate，queueDate保存了broker的name、write及read的queue数量，及topicSy
* 3、如果当前broker为master节点，则直接按照上述步骤更新，如果为slave节点，则将haServerAddr、masterAddr等信息设置到resu
* @author tantexian
* @since 2016/7/6
* @return 如果是slave，则返回master的ha地址
*/

public RegisterBrokerResult registerBroker(//
```

```
final String clusterName,// 1
final String brokerAddr,// 2
final String brokerName,// 3
final long brokerId,// 4
final String haServerAddr,// 5
final TopicConfigSerializeWrapper topicConfigWrapper,// 6
final List<String> filterServerList, // 7
final Channel channel// 8
) {
RegisterBrokerResult result = new RegisterBrokerResult();
try {
try {
// 加写锁
this.lock.writeLock().lockInterruptibly();

// 更新集群信息（根据集群名字，获取当前集群下面的所有brokerName）
Set<String> brokerNames = this.clusterAddrTable.get(clusterName);
// 如果当前集群下面brokerNames为空，则将当前请求broker加入到clusterAddrTable中
if (null == brokerNames) {
brokerNames = new HashSet<String>();
this.clusterAddrTable.put(clusterName, brokerNames);
}
brokerNames.add(brokerName);

boolean registerFirst = false;

// 更新主备信息（在brokerAddrTable中获取所有的brokerDAte）
BrokerData brokerData = this.brokerAddrTable.get(brokerName);
// 如果当前不存在brokerDate，即还没有broker向namesrv注册，则直接将当前broker信息put加入
if (null == brokerData) {
registerFirst = true;
brokerData = new BrokerData();
brokerData.setBrokerName(brokerName);
HashMap<Long, String> brokerAddrs = new HashMap<Long, String>();
brokerData.setBrokerAddrs(brokerAddrs);

this.brokerAddrTable.put(brokerName, brokerData);
}
// 获取当前注册broker的brokerAddr地址
String oldAddr = brokerData.getBrokerAddrs().put(brokerId, brokerAddr);
registerFirst = registerFirst || (null == oldAddr);
// 更新Topic信息
if (null != topicConfigWrapper //如果topicConfigWrapper不为空，且当前brokerId == 0，即为当前broker为master
&& MixAll.MASTER_ID == brokerId) {
// 如果Topic配置信息发生变更或者该broker为第一次注册
if (this.isBrokerTopicConfigChanged(brokerAddr, topicConfigWrapper.getDataVersion())//
|| registerFirst) {
// 获取所有topic信息
ConcurrentHashMap<String, TopicConfig> tcTable =
topicConfigWrapper.getTopicConfigTable();
if (tcTable != null) {
// 遍历所有Topic
for (String topic : tcTable.keySet()) {
TopicConfig topicConfig = tcTable.get(topic);
// 根据brokername及topicconfig（read、write queue数量等）新增或者更新到topicQueueTable中
this.createAndUpdateQueueData(brokerName, topicConfig);
}
}
}
}

// 更新最后变更时间（将brokerLiveTable中保存的对应的broker的更新时间戳，设置为当前时间）
BrokerLiveInfo prevBrokerLiveInfo = this.brokerLiveTable.put(brokerAddr, //
new BrokerLiveInfo(//
System.currentTimeMillis(), //
topicConfigWrapper.getDataVersion(),//
channel, //
haServerAddr));
if (null == prevBrokerLiveInfo) {
log.info("new broker registerd, {} HAServer: {}", brokerAddr, haServerAddr);
}

// 更新Filter Server列表
if (filterServerList != null) {
if (filterServerList.isEmpty()) {
this.filterServerTable.remove(brokerAddr);
}
else {
this.filterServerTable.put(brokerAddr, filterServerList);
}
}

// 返回值（如果当前broker为slave节点）则将haServerAddr、masterAddr等信息设置到result返回值中
if (MixAll.MASTER_ID != brokerId) {
// 通过brokename的brokedate获取当前slave节点的master节点addr
String masterAddr = brokerData.getBrokerAddrs().get(MixAll.MASTER_ID);
if (masterAddr != null) {
BrokerLiveInfo brokerLiveInfo = this.brokerLiveTable.get(masterAddr);
if (brokerLiveInfo != null) {
result.setHaServerAddr(brokerLiveInfo.getHaServerAddr());
result.setMasterAddr(masterAddr);
}
}
}
}
```

finally {// 释放写锁

this.lock.writeLock().unlock();

}

}

catch (Exception e) {

log.error("registerBroker Exception", e);

}

return result;

}

3-2、跟进到controller.initialize()：

```
public boolean initialize() {
// 加载KV配置
this.kvConfigManager.load();

// 初始化通信层
this.remotingServer = new NettyRemotingServer(this.nettyServerConfig, this.brokerHousekeepingService);

// 初始化线程池（根据getServerWorkerThreads值，启动相应数量线程）
this.remotingExecutor = Executors.newFixedThreadPool(nettyServerConfig.getServerWorkerThreads(),
        new ThreadFactoryImpl("RemotingExecutorThread_"));

// 此注册函数主要作用就是，定义RequestCode，用来作为netty的通信协议字段
// 即：如果broker通过netty发送通信请求，其中请求信息中带有code == RequestCode.REGISTER_BROKER，
//那么在namesrv的netty端接收到该通信连接时候，
// 则对应调用namesrv的DefaultRequestProcessor类下面的registerBroker方法，从而完成broker向namesrv注册
// 具体请参考com.alibaba.rocketmq.namesrv.processor.DefaultRequestProcessor类
// 更多关于netty在gmq中的通信机制及原理，请关注后续博文(博客地址为：http://my.oschina.net/tantexian)
this.registerProcessor();

// 增加定时任务（延时5秒，每间隔10s钟，定时扫描一次）
this.scheduledExecutorService.scheduleAtFixedRate(new Runnable() {

@Override
        public void run() {
// 定时扫描notActive的broker（若发现broker过期，则清除该broker与namesrv之间的socketChanel通道）
NamesrvController.this.routeInfoManager.scanNotActiveBroker();
}
    }, 5, 10, TimeUnit.SECONDS);

// 增加定时任务（延时1秒，每间隔10s钟，定时扫描一次）
this.scheduledExecutorService.scheduleAtFixedRate(new Runnable() {

@Override
        public void run() {
// 定时将configTable相关信息记录到日志文件中
NamesrvController.this.kvConfigManager.printAllPeriodically();
}
    }, 1, 10, TimeUnit.MINUTES);

// this.scheduledExecutorService.scheduleAtFixedRate(new Runnable() {
    //
    // @Override
    // public void run() {
    // NamesrvController.this.routeInfoManager.printAllPeriodically();
    // }
    // }, 1, 5, TimeUnit.MINUTES);

return true;
}
```

3-2-1、跟进到this.remotingServer = newNettyRemotingServer(this.nettyServerConfig, this.brokerHousekeepingService);

下述函数为对netty的封装，其中new NioEventLoopGroup为启动一定数量的netty线程来用来与broker通信

```
public NettyRemotingServer(final NettyServerConfig nettyServerConfig,
        final ChannelEventListener channelEventListener) {
super(nettyServerConfig.getServerOnewaySemaphoreValue(), nettyServerConfig
.getServerAsyncSemaphoreValue());
```

```
    this.serverBootstrap = new ServerBootstrap();
    this.nettyServerConfig = nettyServerConfig;
    this.channelEventListener = channelEventListener;

    int publicThreadNums = nettyServerConfig.getServerCallbackExecutorThreads();
    if (publicThreadNums <= 0) {
        publicThreadNums = 4;
}

this.publicExecutor = Executors.newFixedThreadPool(publicThreadNums, new ThreadFactory() {
private AtomicInteger threadIndex = new AtomicInteger(0);


        @Override
        public Thread newThread(Runnable r) {
return new Thread(r, "NettyServerPublicExecutor_" + this.threadIndex.incrementAndGet());
}
    });

    this.eventLoopGroupBoss = new NioEventLoopGroup(1, new ThreadFactory() {
private AtomicInteger threadIndex = new AtomicInteger(0);


        @Override
        public Thread newThread(Runnable r) {
return new Thread(r,
String.format("NettyBossSelector_%d", this.threadIndex.incrementAndGet()));
}
    });

    this.eventLoopGroupWorker =
            new NioEventLoopGroup(nettyServerConfig.getServerSelectorThreads(), new ThreadFactory() {
private AtomicInteger threadIndex = new AtomicInteger(0);
                private int threadTotal = nettyServerConfig.getServerSelectorThreads();


                @Override
                public Thread newThread(Runnable r) {
return new Thread(r, String.format("NettyServerSelector_%d_%d", threadTotal,
                        this.threadIndex.incrementAndGet()));
}
            });
}
```

3-2-2、this.registerProcessor();

```
// 此注册函数主要作用就是，定义RequestCode，用来作为netty的通信协议字段
// 即：如果broker通过netty发送通信请求，其中请求信息中带有code == RequestCode.REGISTER_BROKER，
//那么在namesrv的netty端接收到该通信连接时候，
// 则对应调用namesrv的DefaultRequestProcessor类下面的registerBroker方法，从而完成broker向namesrv注册
// 具体请参考com.alibaba.rocketmq.namesrv.processor.DefaultRequestProcessor类
// 更多关于netty在gmq中的通信机制及原理，请关注后续博文(博客地址为：http://my.oschina.net/tantexian)
this.registerProcessor();
```

3-2-3、增加定时任务（延时5秒，每间隔10s钟，定时扫描一次），扫描notActive的broker（若发现broker过期，则清除该broker与namesrv之间的socketChanel通道）

```
// 增加定时任务（延时5秒，每间隔10s钟，定时扫描一次）
this.scheduledExecutorService.scheduleAtFixedRate(new Runnable() {

@Override
        public void run() {
// 定时扫描notActive的broker（若发现broker过期，则清除该broker与namesrv之间的socketChanel通道）
NamesrvController.this.routeInfoManager.scanNotActiveBroker();
}
    }, 5, 10, TimeUnit.SECONDS);
```

3-2-4、增加定时任务，每10s定时，将configTable相关信息记录到日志文件中：

```
// 增加定时任务（延时1秒，每间隔10s钟，定时扫描一次）
this.scheduledExecutorService.scheduleAtFixedRate(new Runnable() {

@Override
```

```
    public void run() {
// 定时将configTable相关信息记录到日志文件中
NamesrvController.this.kvConfigManager.printAllPeriodically();
}
}, 1, 10, TimeUnit.MINUTES);
```

4、设置一个jvm退出勾子函数，即jvm退出时，此处线程调用controller.shutdown()，清理controller相关资源：

```
// 设置一个jvm退出勾子函数，即jvm退出时，此处线程调用controller.shutdown()，清理controller相关资源
Runtime.getRuntime().addShutdownHook(new Thread(new Runnable() {
private volatile boolean hasShutdown = false;
    private AtomicInteger shutdownTimes = new AtomicInteger(0);


    @Override
    public void run() {
synchronized (this) {
log.info("shutdown hook was invoked, " + this.shutdownTimes.incrementAndGet());
            if (!this.hasShutdown) {
this.hasShutdown = true;
                long begineTime = System.currentTimeMillis();
controller.shutdown();
                long consumingTimeTotal = System.currentTimeMillis() - begineTime;
log.info("shutdown hook over, consuming time total(ms): " + consumingTimeTotal);
}
        }
    }
}, "ShutdownHook"));
```

5、启动服务（主要就是启动netty监听网络通信请求，即初始化netty启动异步通信server）：

```
// 启动服务（主要就是启动netty监听网络通信请求，即初始化netty启动异步通信server）
controller.start();
```

5-1、跟进到nameController的start方法：

```
public void start() throws Exception {
this.remotingServer.start();
}
```

5-1-1、跟进到NettyRemotingServer的start方法：

主要功能即，通过封装调用netty启动netty异步通信框架。

```
@Override
public void start() {
this.defaultEventExecutorGroup = new DefaultEventExecutorGroup(//
nettyServerConfig.getServerWorkerThreads(), //
new ThreadFactory() {

private AtomicInteger threadIndex = new AtomicInteger(0);


            @Override
            public Thread newThread(Runnable r) {
return new Thread(r, "NettyServerWorkerThread_" + this.threadIndex.incrementAndGet());
}
        });

ServerBootstrap childHandler = //
this.serverBootstrap.group(this.eventLoopGroupBoss, this.eventLoopGroupWorker)
.channel(NioServerSocketChannel.class)
//
.option(ChannelOption.SO_BACKLOG, 1024)
//
.option(ChannelOption.SO_REUSEADDR, true)
//
.option(ChannelOption.SO_KEEPALIVE, false)
//
.childOption(ChannelOption.TCP_NODELAY, true)
//
.option(ChannelOption.SO_SNDBUF, nettyServerConfig.getServerSocketSndBufSize())
//
.option(ChannelOption.SO_RCVBUF, nettyServerConfig.getServerSocketRcvBufSize())
//
.localAddress(new InetSocketAddress(this.nettyServerConfig.getListenPort()))
.childHandler(new ChannelInitializer<SocketChannel>() {
@Override
                    public void initChannel(SocketChannel ch) throws Exception {
ch.pipeline().addLast(
//
defaultEventExecutorGroup, //
new NettyEncoder(), //
new NettyDecoder(), //
new IdleStateHandler(0, 0, nettyServerConfig
.getServerChannelMaxIdleTimeSeconds()),//
new NettyConnetManageHandler(), //
new NettyServerHandler());
}
                });

    if (nettyServerConfig.isServerPooledByteBufAllocatorEnable()) {
// 这个选项有可能会占用大量堆外内存，暂时不使用。
childHandler.childOption(ChannelOption.ALLOCATOR, PooledByteBufAllocator.DEFAULT);
}

try {
ChannelFuture sync = this.serverBootstrap.bind().sync();
InetSocketAddress addr = (InetSocketAddress) sync.channel().localAddress();
        this.port = addr.getPort();
}
catch (InterruptedException e1) {
throw new RuntimeException("this.serverBootstrap.bind().sync() InterruptedException", e1);
}

if (this.channelEventListener != null) {
this.nettyEventExecuter.start();
}

// 每隔1秒扫描下异步调用超时情况
this.timer.scheduleAtFixedRate(new TimerTask() {

@Override
        public void run() {
try {
NettyRemotingServer.this.scanResponseTable();
}
catch (Exception e) {
log.error("scanResponseTable exception", e);
}
        }
    }, 1000 * 3, 1000);
}
```

分类：RocketMQ相关　字数：2376

打赏　点赞　收藏　分享

tantexian

架构师　成都

+ 关注

粉丝 82 | 博文 364 | 码字总数 529016

相关博客

RocketMQ安装、启动
站在巨人的肩膀上奋斗
69 2

RocketMQ Filtersrv详解
Bieber
1486 1

RocketMQ-debug笔记
强子哥哥
47 0

评论 (0)

Ctrl+Enter 发表评论