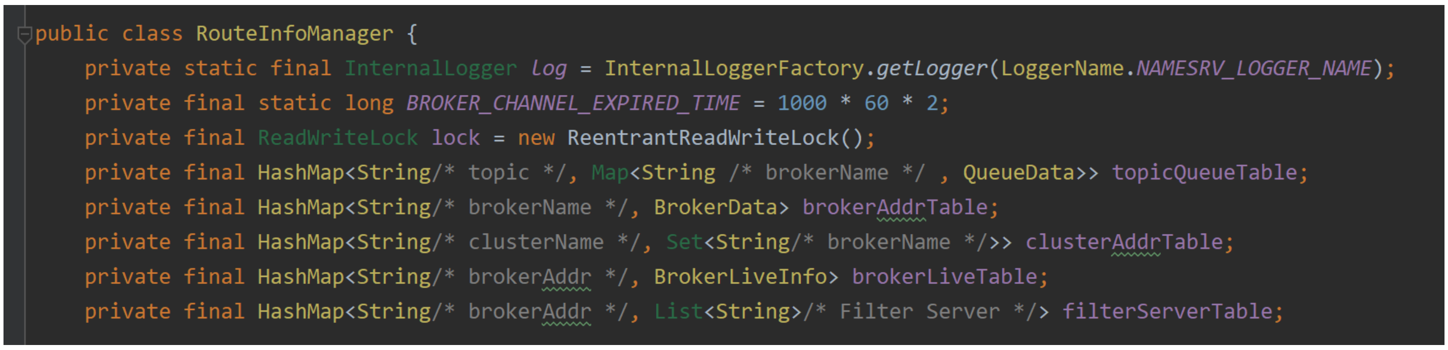
技术文章

分析元数据的生命周期

分析**路由元数据**的生命周期

路由元数据存储于nameserver中的RouteInfoManager，由nameserver进行路由注册和删除工作



```
public class RouteInfoManager {
    private static final InternalLogger log = InternalLoggerFactory.getLogger(LoggerName.NAMESRV_LOGGER_NAME);
    private final static long BROKER_CHANNEL_EXPIRED_TIME = 1000 * 60 * 2;
    private final ReadWriteLock lock = new ReentrantReadWriteLock();
    private final HashMap<String/* topic */, Map<String /* brokerName */ , QueueData>> topicQueueTable;
    private final HashMap<String/* brokerName */, BrokerData> brokerAddrTable;
    private final HashMap<String/* clusterName */, Set<String/* brokerName */>> clusterAddrTable;
    private final HashMap<String/* brokerAddr */, BrokerLiveInfo> brokerLiveTable;
    private final HashMap<String/* brokerAddr */, List<String>/* Filter Server */> filterServerTable;
```

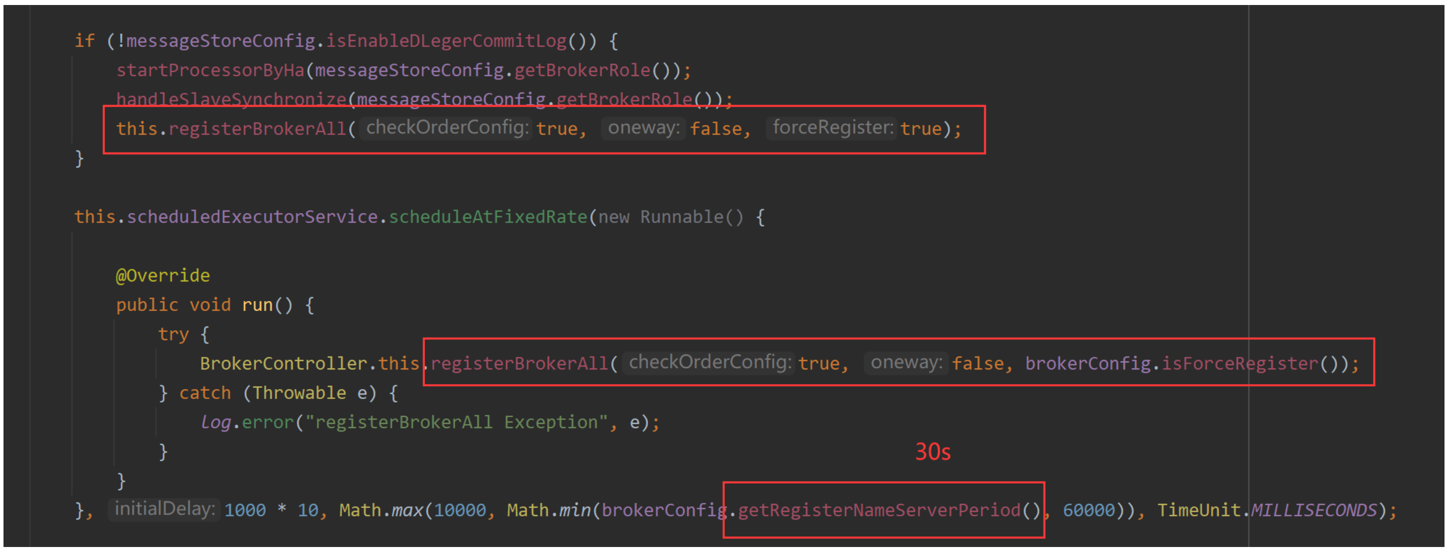
服务端

创建/更新元数据

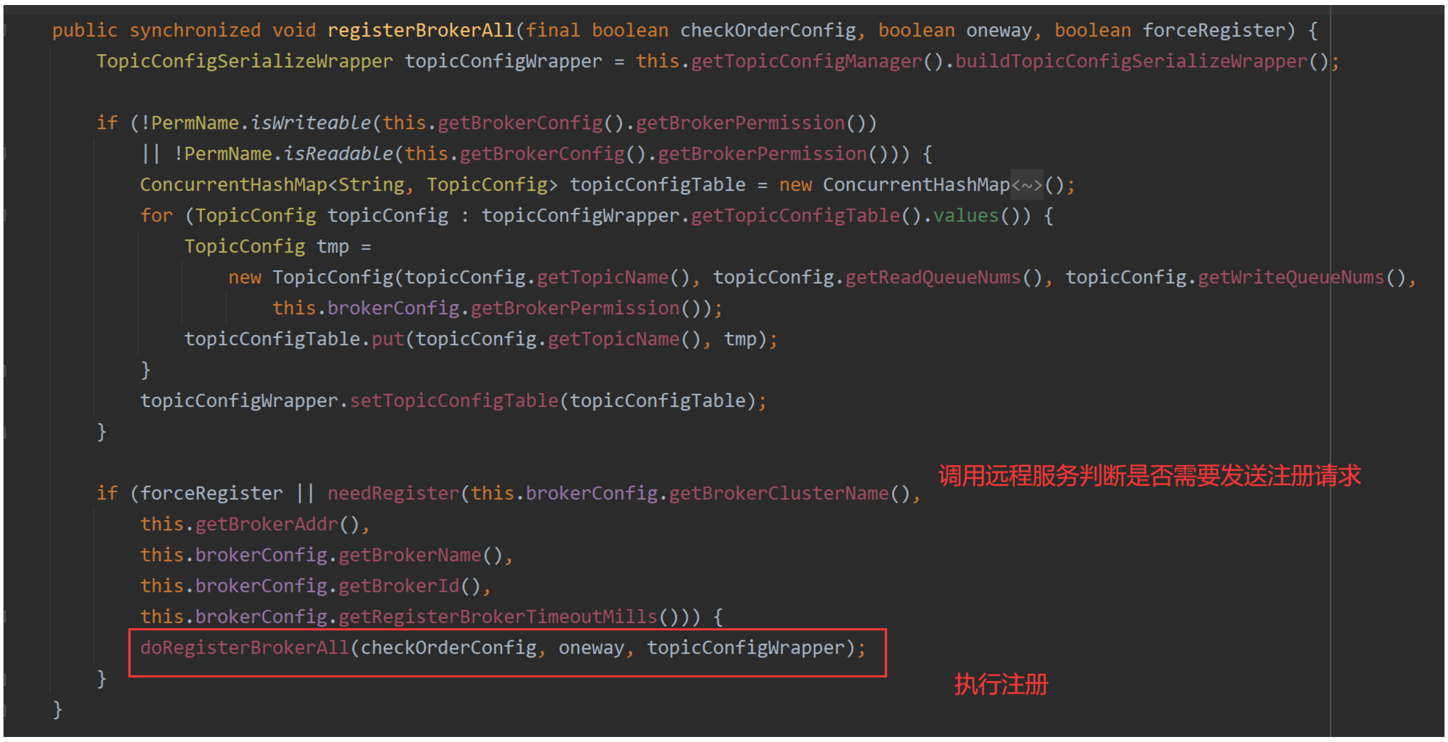
每次启动一个broker，broker会触发路由注册，broker启动时会和所有nameserver创建心跳连接，向NameServer发送Broker的相关信息。在下面【Broker 注册到 Nameserver】可以看到具体流程。

Broker 注册到 Nameserver

在BrokerController的start方法中，可以看到broker在启动时，会向nameserver注册，之后会通过定时任务，每隔30s以心跳的方式上报。



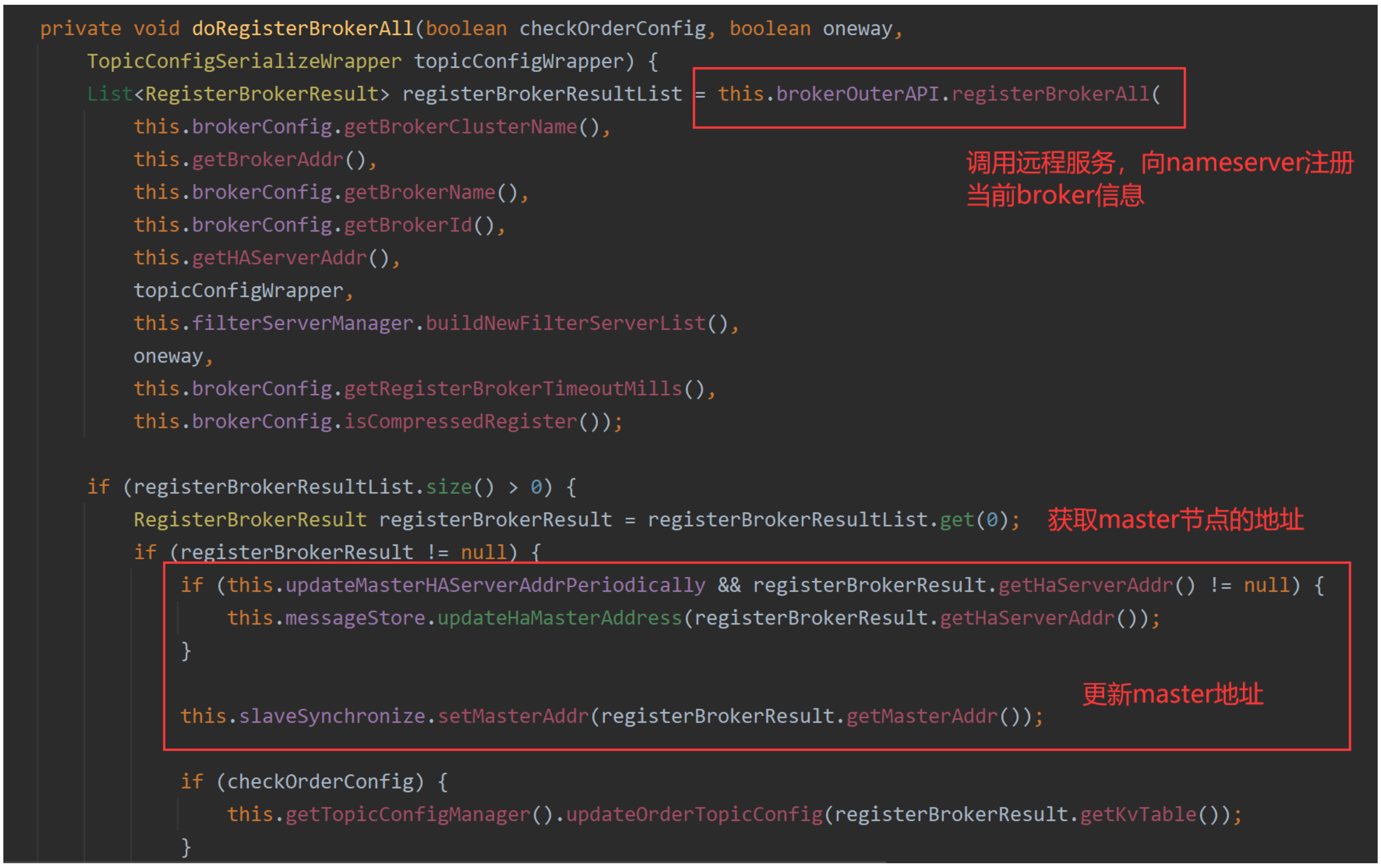
```
    if (!messageStoreConfig.isEnableDLegerCommitLog()) {
        startProcessorByHa(messageStoreConfig.getBrokerRole());
        handleSlaveSynchronize(messageStoreConfig.getBrokerRole());
        this.registerBrokerAll( checkOrderConfig: true,  oneway: false,  forceRegister: true);
    }

    this.scheduledExecutorService.scheduleAtFixedRate(new Runnable() {

        @Override
        public void run() {
            try {
                BrokerController.this.registerBrokerAll( checkOrderConfig: true,  oneway: false, brokerConfig.isForceRegister());
            } catch (Throwable e) {
                log.error("registerBrokerAll Exception", e);
            }
        }
    },  initialDelay: 1000 * 10, Math.max(10000, Math.min(brokerConfig.getRegisterNameServerPeriod(), 60000)), TimeUnit.MILLISECONDS);
```



```java
public synchronized void registerBrokerAll(final boolean checkOrderConfig, boolean oneway, boolean forceRegister) {
    TopicConfigSerializeWrapper topicConfigWrapper = this.getTopicConfigManager().buildTopicConfigSerializeWrapper();

    if (!PermName.isWriteable(this.getBrokerConfig().getBrokerPermission())
        || !PermName.isReadable(this.getBrokerConfig().getBrokerPermission())) {
        ConcurrentHashMap<String, TopicConfig> topicConfigTable = new ConcurrentHashMap<~>();
        for (TopicConfig topicConfig : topicConfigWrapper.getTopicConfigTable().values()) {
            TopicConfig tmp =
                new TopicConfig(topicConfig.getTopicName(), topicConfig.getReadQueueNums(), topicConfig.getWriteQueueNums(),
                    this.brokerConfig.getBrokerPermission());
            topicConfigTable.put(topicConfig.getTopicName(), tmp);
        }
        topicConfigWrapper.setTopicConfigTable(topicConfigTable);
    }

    if (forceRegister || needRegister(this.brokerConfig.getBrokerClusterName(),
        this.getBrokerAddr(),
        this.brokerConfig.getBrokerName(),
        this.brokerConfig.getBrokerId(),
        this.brokerConfig.getRegisterBrokerTimeoutMills())) {
        doRegisterBrokerAll(checkOrderConfig, oneway, topicConfigWrapper);
    }
}
```

调用远程服务判断是否需要发送注册请求

执行注册



```java
private void doRegisterBrokerAll(boolean checkOrderConfig, boolean oneway,
    TopicConfigSerializeWrapper topicConfigWrapper) {
    List<RegisterBrokerResult> registerBrokerResultList = this.brokerOuterAPI.registerBrokerAll(
        this.brokerConfig.getBrokerClusterName(),
        this.getBrokerAddr(),
        this.brokerConfig.getBrokerName(),
        this.brokerConfig.getBrokerId(),
        this.getHAServerAddr(),
        topicConfigWrapper,
        this.filterServerManager.buildNewFilterServerList(),
        oneway,
        this.brokerConfig.getRegisterBrokerTimeoutMills(),
        this.brokerConfig.isCompressedRegister());

    if (registerBrokerResultList.size() > 0) {
        RegisterBrokerResult registerBrokerResult = registerBrokerResultList.get(0);
        if (registerBrokerResult != null) {
            if (this.updateMasterHAServerAddrPeriodically && registerBrokerResult.getHaServerAddr() != null) {
                this.messageStore.updateHaMasterAddress(registerBrokerResult.getHaServerAddr());
            }

            this.slaveSynchronize.setMasterAddr(registerBrokerResult.getMasterAddr());

            if (checkOrderConfig) {
                this.getTopicConfigManager().updateOrderTopicConfig(registerBrokerResult.getKvTable());
            }
```

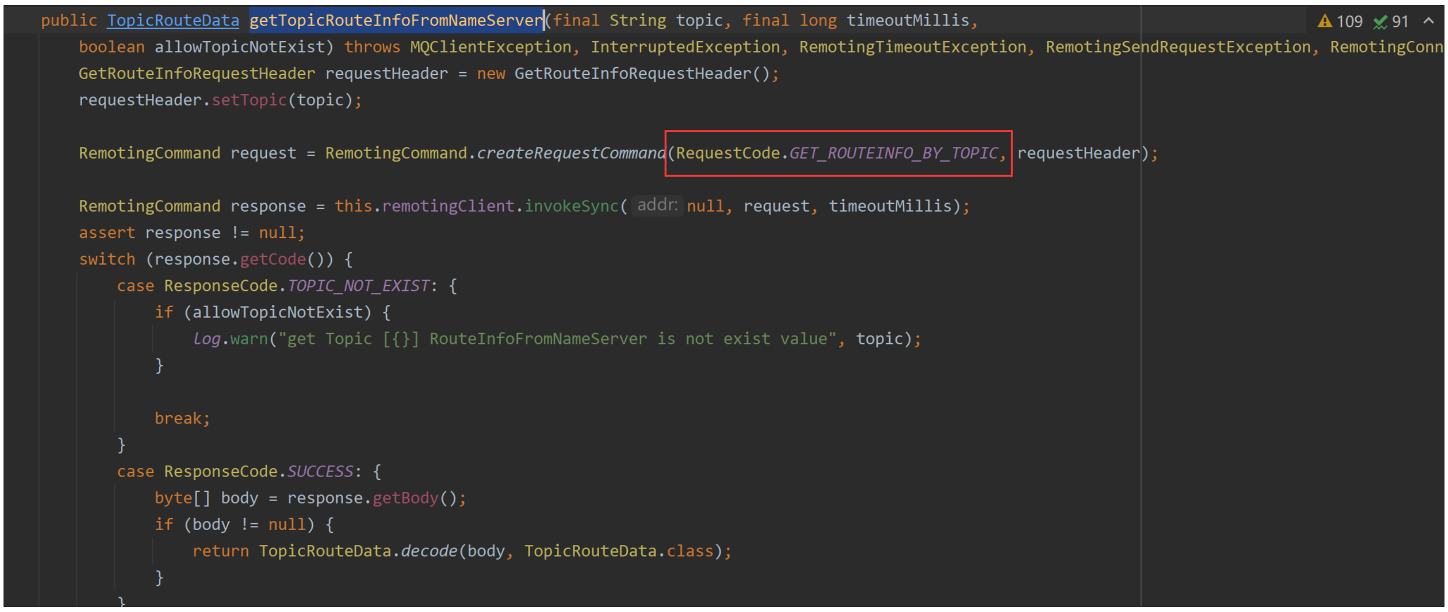
调用远程服务，向nameserver注册当前broker信息

获取master节点的地址

更新master地址

Client 访问 Nameserver

客户端定时向nameserver发起请求GET_ROUTEINFO_BY_TOPIC，获取对应的路由信息

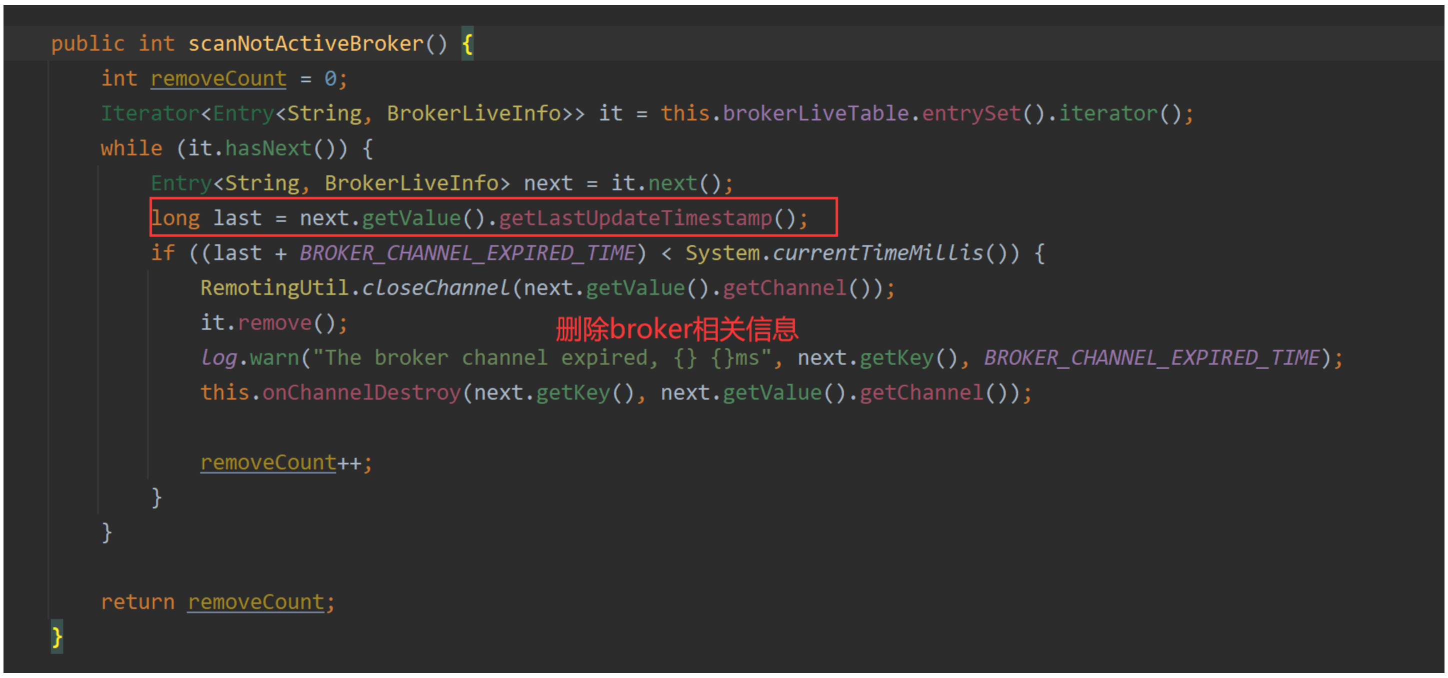


```java
public TopicRouteData getTopicRouteInfoFromNameServer(final String topic, final long timeoutMillis,
    boolean allowTopicNotExist) throws MQClientException, InterruptedException, RemotingTimeoutException, RemotingSendRequestException, RemotingConn
    GetRouteInfoRequestHeader requestHeader = new GetRouteInfoRequestHeader();
    requestHeader.setTopic(topic);

    RemotingCommand request = RemotingCommand.createRequestCommand(RequestCode.GET_ROUTEINFO_BY_TOPIC, requestHeader);

    RemotingCommand response = this.remotingClient.invokeSync( addr: null, request, timeoutMillis);
    assert response != null;
    switch (response.getCode()) {
        case ResponseCode.TOPIC_NOT_EXIST: {
            if (allowTopicNotExist) {
                log.warn("get Topic [{}] RouteInfoFromNameServer is not exist value", topic);
            }

            break;
        }
        case ResponseCode.SUCCESS: {
            byte[] body = response.getBody();
            if (body != null) {
                return TopicRouteData.decode(body, TopicRouteData.class);
            }
```

心跳感知与动态删除

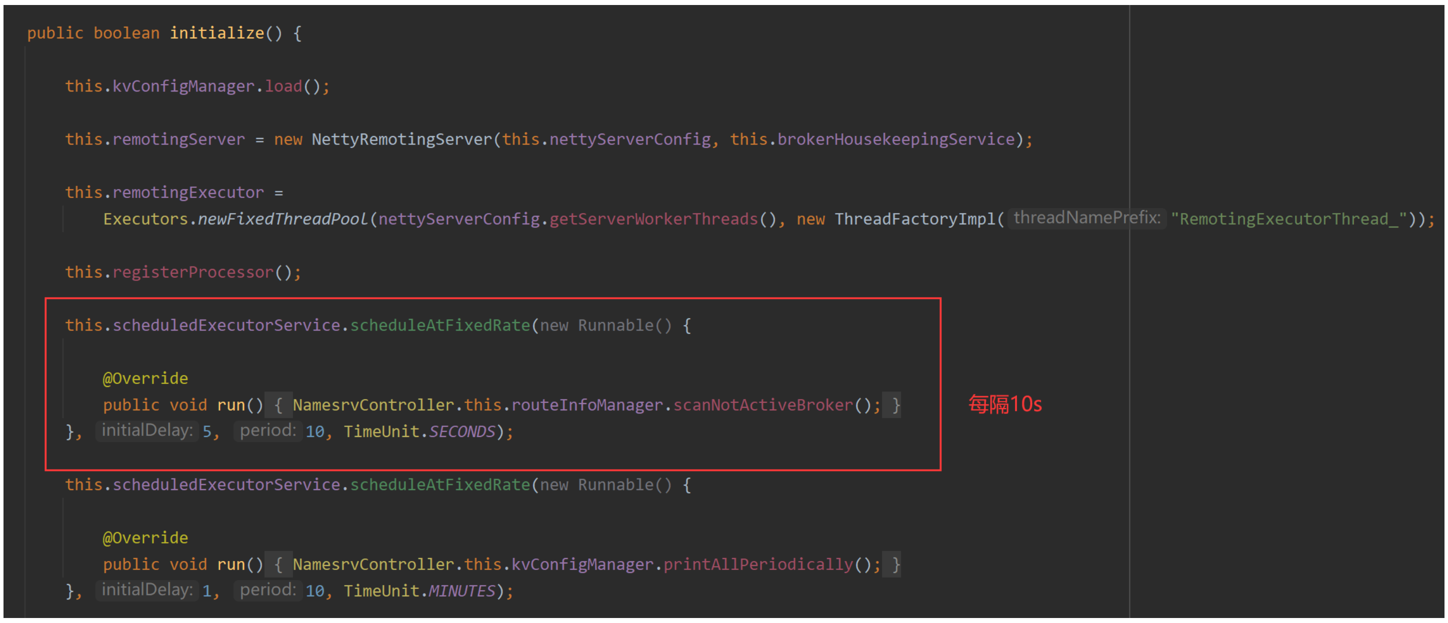
- nameserver启动后，每10s会扫描一次brokerLiveTable，若某个broker的lastUpdateTimestamp在120s没有更新，nameserver会移除该broker的路由信息。

在RouteInfoManager中有方法scanNotActiveBroker()



```java
public int scanNotActiveBroker() {
    int removeCount = 0;
    Iterator<Entry<String, BrokerLiveInfo>> it = this.brokerLiveTable.entrySet().iterator();
    while (it.hasNext()) {
        Entry<String, BrokerLiveInfo> next = it.next();
        long last = next.getValue().getLastUpdateTimestamp();
        if ((last + BROKER_CHANNEL_EXPIRED_TIME) < System.currentTimeMillis()) {
            RemotingUtil.closeChannel(next.getValue().getChannel());
            it.remove();
            log.warn("The broker channel expired, {} {}ms", next.getKey(), BROKER_CHANNEL_EXPIRED_TIME);
            this.onChannelDestroy(next.getKey(), next.getValue().getChannel());

            removeCount++;
        }
    }

    return removeCount;
}
```

在NamesrvController中的initializa()方法中可以看到该心跳感知

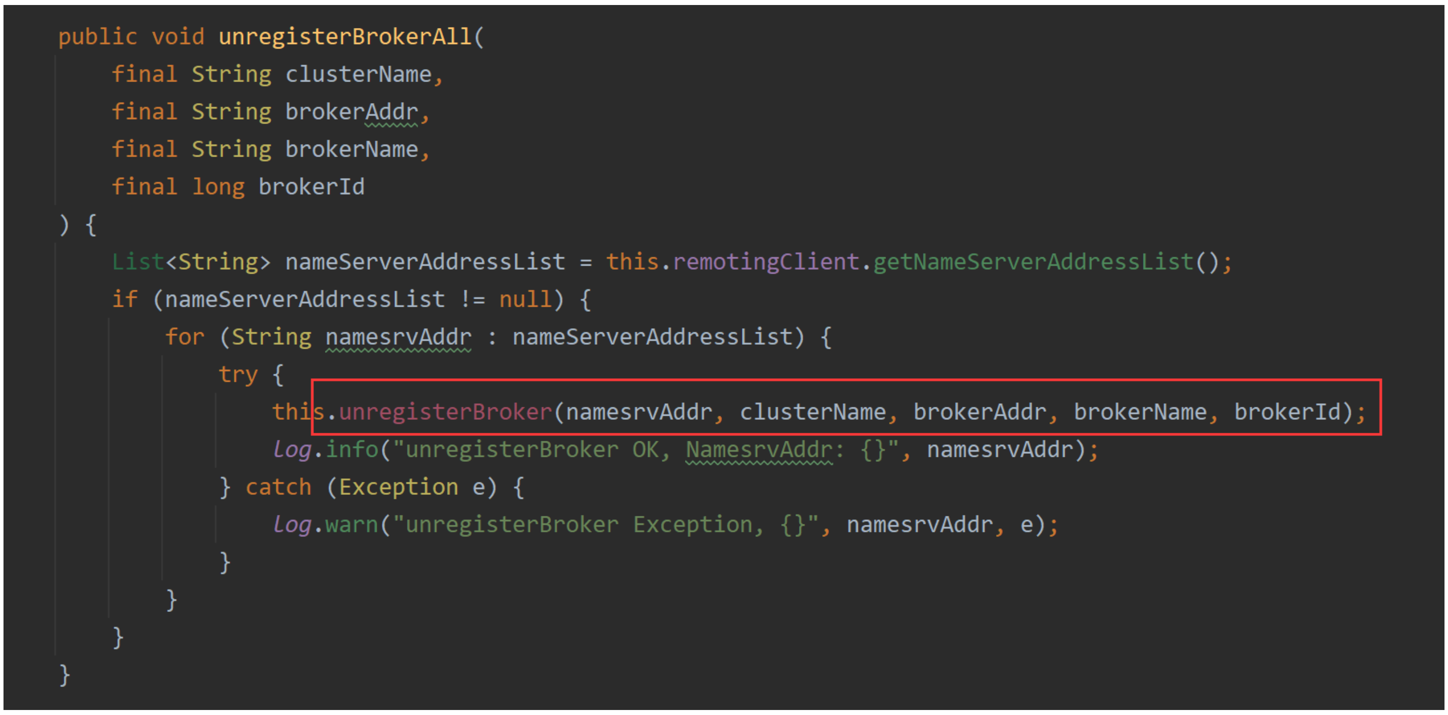


```java
public boolean initialize() {

    this.kvConfigManager.load();

    this.remotingServer = new NettyRemotingServer(this.nettyServerConfig, this.brokerHousekeepingService);

    this.remotingExecutor =
        Executors.newFixedThreadPool(nettyServerConfig.getServerWorkerThreads(), new ThreadFactoryImpl( threadNamePrefix: "RemotingExecutorThread_"));

    this.registerProcessor();

    this.scheduledExecutorService.scheduleAtFixedRate(new Runnable() {

        @Override
        public void run() { NamesrvController.this.routeInfoManager.scanNotActiveBroker(); }
    }, initialDelay: 5, period: 10, TimeUnit.SECONDS);

    this.scheduledExecutorService.scheduleAtFixedRate(new Runnable() {

        @Override
        public void run() { NamesrvController.this.kvConfigManager.printAllPeriodically(); }
    }, initialDelay: 1, period: 10, TimeUnit.MINUTES);
```

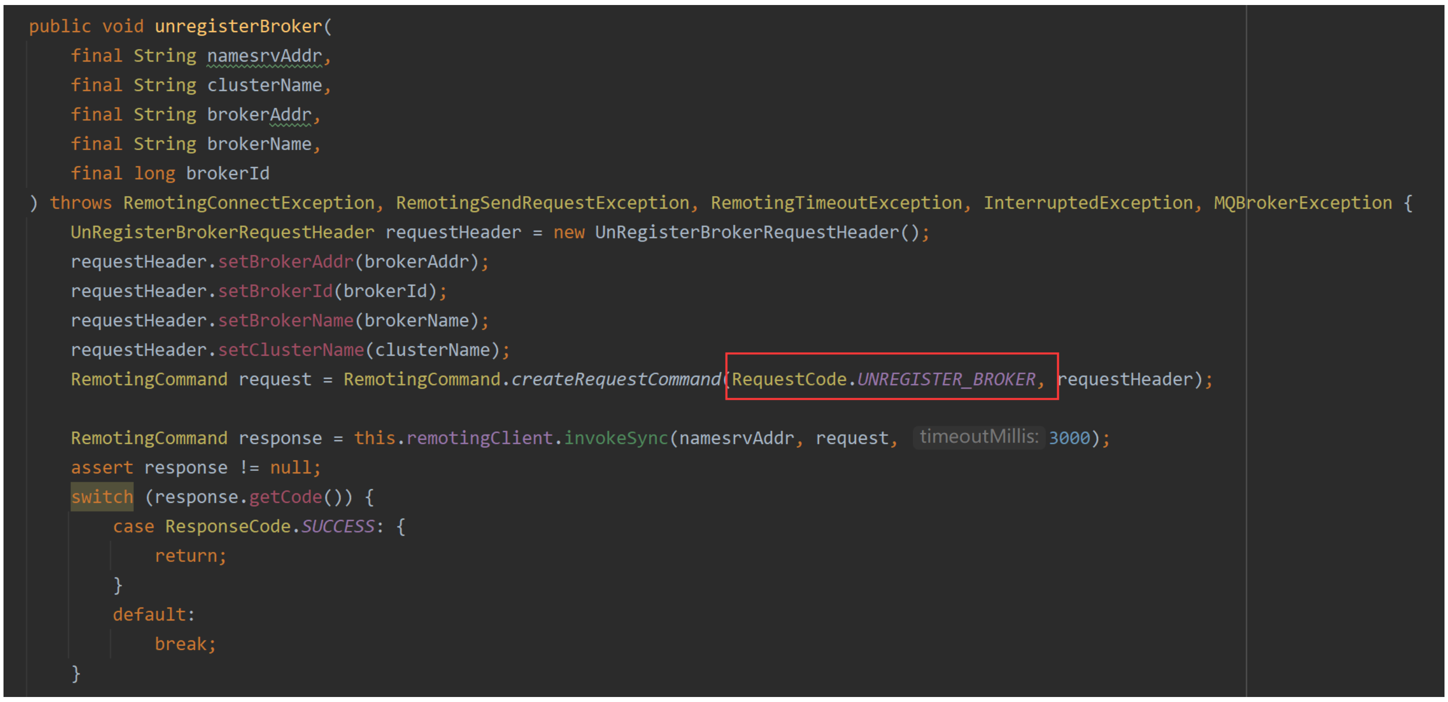
- broker自己关闭时，会发送UNREGISTER_BROKER 命令通知nameserver删除该路由信息

在BrokerController的shutdown方法中会调用unregisterBrokerAll()方法来注销 Broker

unregisterBrokerAll()在BrokerOuterAPI中：



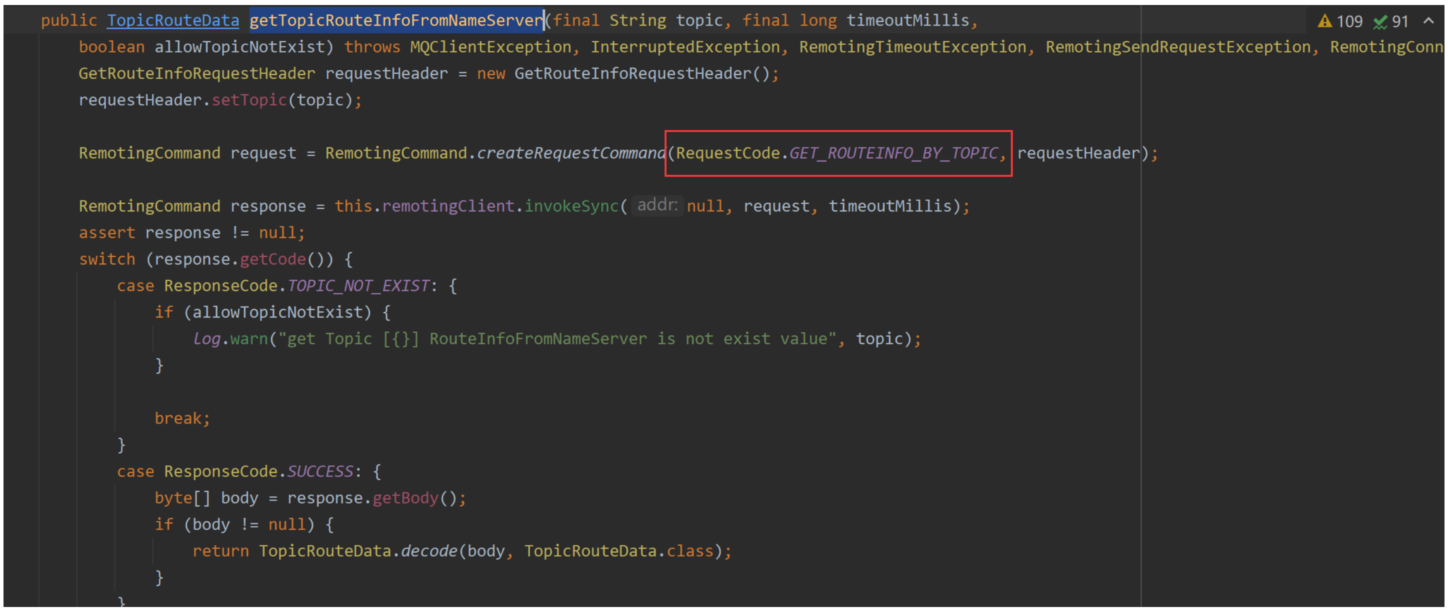
```java
public void unregisterBrokerAll(
    final String clusterName,
    final String brokerAddr,
    final String brokerName,
    final long brokerId
) {
    List<String> nameServerAddressList = this.remotingClient.getNameServerAddressList();
    if (nameServerAddressList != null) {
        for (String namesrvAddr : nameServerAddressList) {
            try {
                this.unregisterBroker(namesrvAddr, clusterName, brokerAddr, brokerName, brokerId);
                log.info("unregisterBroker OK, NamesrvAddr: {}", namesrvAddr);
            } catch (Exception e) {
                log.warn("unregisterBroker Exception, {}", namesrvAddr, e);
            }
        }
    }
}
```


```java
public void unregisterBroker(
    final String namesrvAddr,
    final String clusterName,
    final String brokerAddr,
    final String brokerName,
    final long brokerId
) throws RemotingConnectException, RemotingSendRequestException, RemotingTimeoutException, InterruptedException, MQBrokerException {
    UnRegisterBrokerRequestHeader requestHeader = new UnRegisterBrokerRequestHeader();
    requestHeader.setBrokerAddr(brokerAddr);
    requestHeader.setBrokerId(brokerId);
    requestHeader.setBrokerName(brokerName);
    requestHeader.setClusterName(clusterName);
    RemotingCommand request = RemotingCommand.createRequestCommand(RequestCode.UNREGISTER_BROKER, requestHeader);

    RemotingCommand response = this.remotingClient.invokeSync(namesrvAddr, request,  timeoutMillis: 3000);
    assert response != null;
    switch (response.getCode()) {
        case ResponseCode.SUCCESS: {
            return;
        }
        default:
            break;
    }
```

客户端

获取元数据

客户端定时向nameserver发起请求GET_ROUTEINFO_BY_TOPIC，获取对应的信息。


```java
public TopicRouteData getTopicRouteInfoFromNameServer(final String topic, final long timeoutMillis,
    boolean allowTopicNotExist) throws MQClientException, InterruptedException, RemotingTimeoutException, RemotingSendRequestException, RemotingConn
    GetRouteInfoRequestHeader requestHeader = new GetRouteInfoRequestHeader();
    requestHeader.setTopic(topic);

    RemotingCommand request = RemotingCommand.createRequestCommand(RequestCode.GET_ROUTEINFO_BY_TOPIC, requestHeader);

    RemotingCommand response = this.remotingClient.invokeSync( addr: null, request, timeoutMillis);
    assert response != null;
    switch (response.getCode()) {
        case ResponseCode.TOPIC_NOT_EXIST: {
            if (allowTopicNotExist) {
                log.warn("get Topic [{}] RouteInfoFromNameServer is not exist value", topic);
            }

            break;
        }
        case ResponseCode.SUCCESS: {
            byte[] body = response.getBody();
            if (body != null) {
                return TopicRouteData.decode(body, TopicRouteData.class);
            }
```

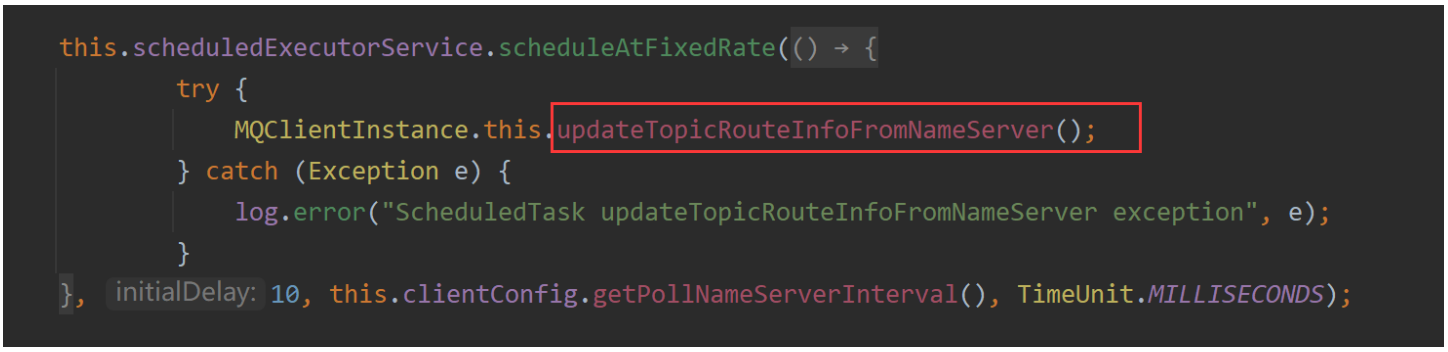
感知元数据的变化

在topic对应的路由信息变化后，nameserver不会通知客户端。需要客户端定时获取topic的最新路由，具体实现如下流程：

producer.start();--->this.defaultMQProducerImpl.start();--->mQClientFactory.start();--->

this.startScheduledTask();--->updateTopicRouteInfoFromNameServer();--->

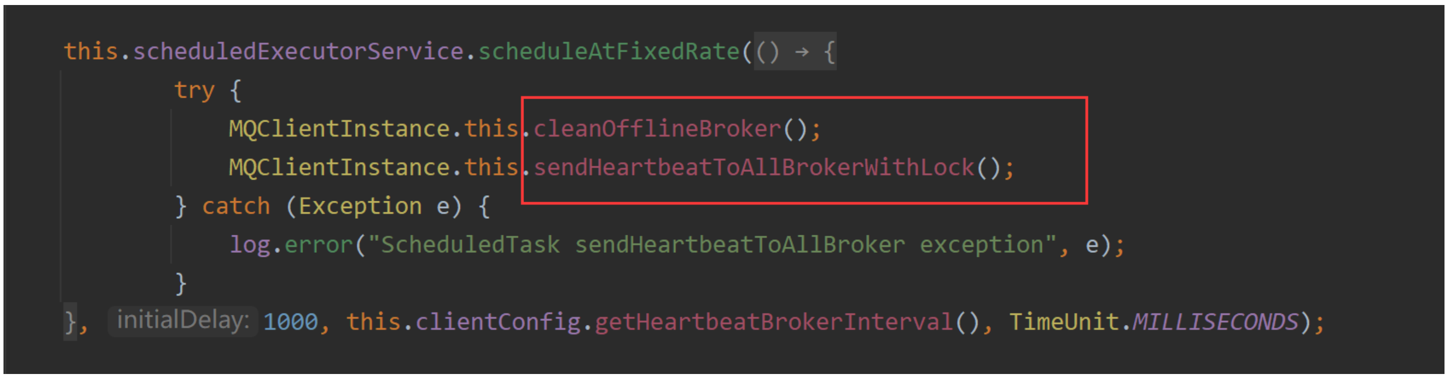
mQClientAPIImpl.getTopicRouteInfoFromNameServer();



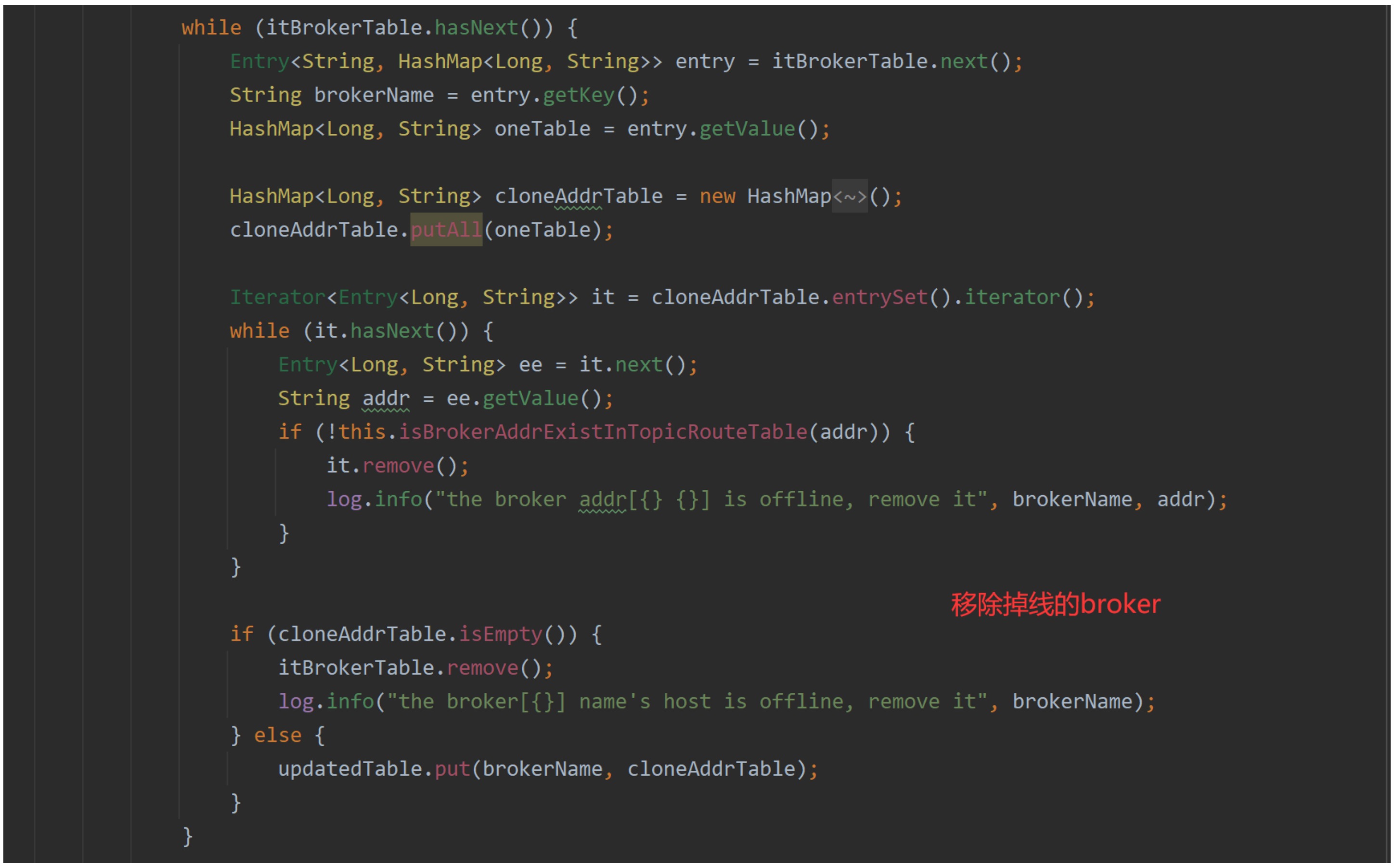
```
this.scheduledExecutorService.scheduleAtFixedRate(() → {
        try {
            MQClientInstance.this.updateTopicRouteInfoFromNameServer();
        } catch (Exception e) {
            log.error("ScheduledTask updateTopicRouteInfoFromNameServer exception", e);
        }
}, initialDelay: 10, this.clientConfig.getPollNameServerInterval(), TimeUnit.MILLISECONDS);
```

清除不必要的元数据

客户端在启动之后，会定时向broker发送心跳，定时检查broker是否掉线，及时清理已经过期的元数据。

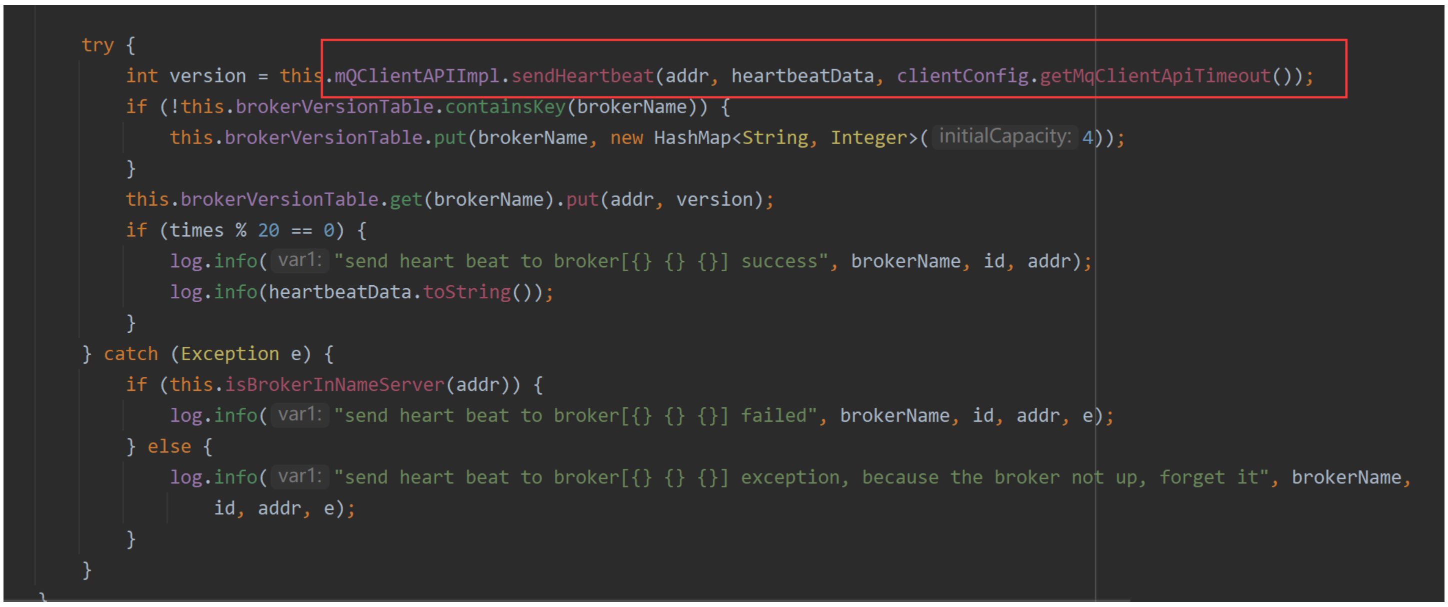


```
this.scheduledExecutorService.scheduleAtFixedRate(() → {
        try {
            MQClientInstance.this.cleanOfflineBroker();
            MQClientInstance.this.sendHeartbeatToAllBrokerWithLock();
        } catch (Exception e) {
            log.error("ScheduledTask sendHeartbeatToAllBroker exception", e);
        }
}, initialDelay: 1000, this.clientConfig.getHeartbeatBrokerInterval(), TimeUnit.MILLISECONDS);
```

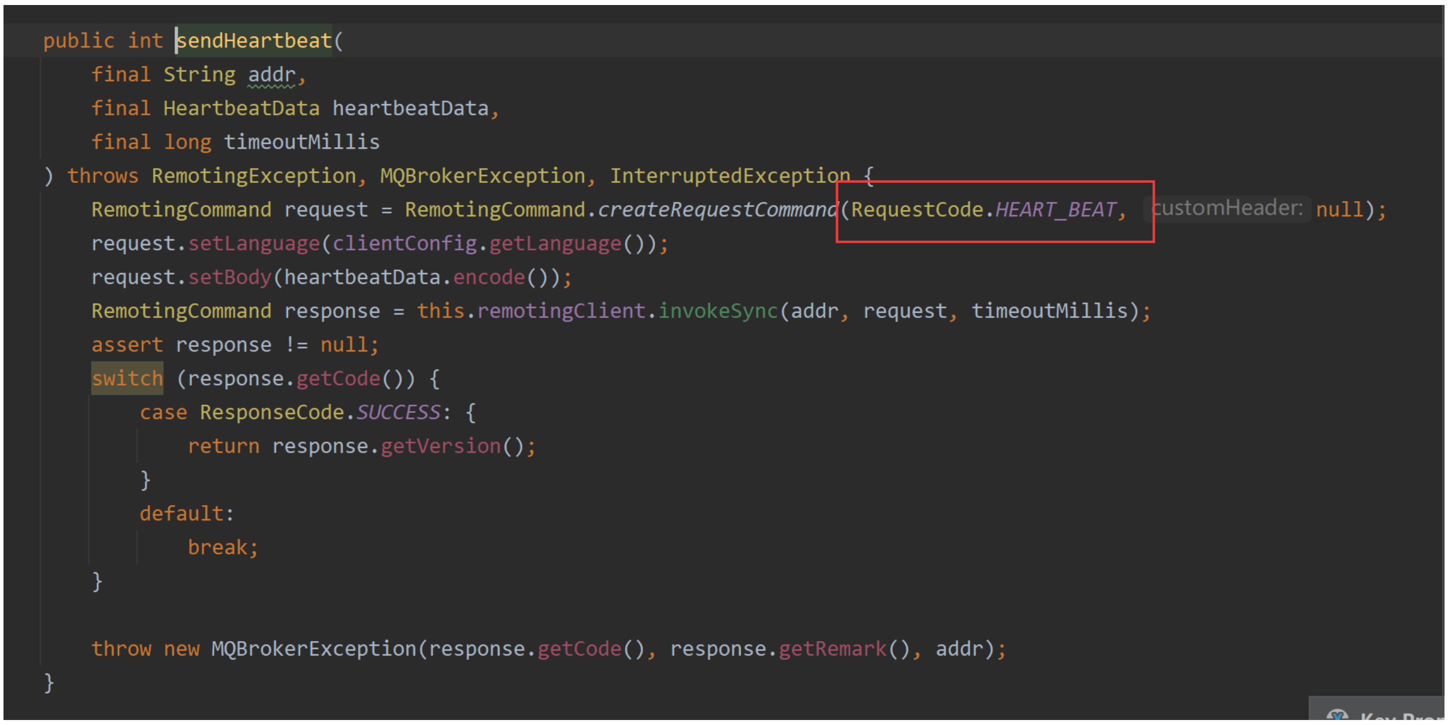


```
while (itBrokerTable.hasNext()) {
    Entry<String, HashMap<Long, String>> entry = itBrokerTable.next();
    String brokerName = entry.getKey();
    HashMap<Long, String> oneTable = entry.getValue();

    HashMap<Long, String> cloneAddrTable = new HashMap<~>();
    cloneAddrTable.putAll(oneTable);

    Iterator<Entry<Long, String>> it = cloneAddrTable.entrySet().iterator();
    while (it.hasNext()) {
        Entry<Long, String> ee = it.next();
        String addr = ee.getValue();
        if (!this.isBrokerAddrExistInTopicRouteTable(addr)) {
            it.remove();
            log.info("the broker addr[{} {}] is offline, remove it", brokerName, addr);
        }
    }

    if (cloneAddrTable.isEmpty()) {
        itBrokerTable.remove();
        log.info("the broker[{}] name's host is offline, remove it", brokerName);
    } else {
        updatedTable.put(brokerName, cloneAddrTable);
    }
}
```

向broker发送心跳



```java
    try {
        int version = this.mQClientAPIImpl.sendHeartbeat(addr, heartbeatData, clientConfig.getMqClientApiTimeout());
        if (!this.brokerVersionTable.containsKey(brokerName)) {
            this.brokerVersionTable.put(brokerName, new HashMap<String, Integer>( initialCapacity: 4));
        }
        this.brokerVersionTable.get(brokerName).put(addr, version);
        if (times % 20 == 0) {
            log.info( var1: "send heart beat to broker[{} {} {}] success", brokerName, id, addr);
            log.info(heartbeatData.toString());
        }
    } catch (Exception e) {
        if (this.isBrokerInNameServer(addr)) {
            log.info( var1: "send heart beat to broker[{} {} {}] failed", brokerName, id, addr, e);
        } else {
            log.info( var1: "send heart beat to broker[{} {} {}] exception, because the broker not up, forget it", brokerName,
                id, addr, e);
        }
    }
```



```java
public int sendHeartbeat(
    final String addr,
    final HeartbeatData heartbeatData,
    final long timeoutMillis
) throws RemotingException, MQBrokerException, InterruptedException {
    RemotingCommand request = RemotingCommand.createRequestCommand(RequestCode.HEART_BEAT,  customHeader: null);
    request.setLanguage(clientConfig.getLanguage());
    request.setBody(heartbeatData.encode());
    RemotingCommand response = this.remotingClient.invokeSync(addr, request, timeoutMillis);
    assert response != null;
    switch (response.getCode()) {
        case ResponseCode.SUCCESS: {
            return response.getVersion();
        }
        default:
            break;
    }

    throw new MQBrokerException(response.getCode(), response.getRemark(), addr);
}
```