图文 105 NameServer是如何处理Broker的注册请求的?



详情 评论

**NameServer是如何处理Broker的注册请求的?**


继《从零开始带你成为JVM实战高手》后，阿里资深技术专家携新作再度出山，重磅推荐：

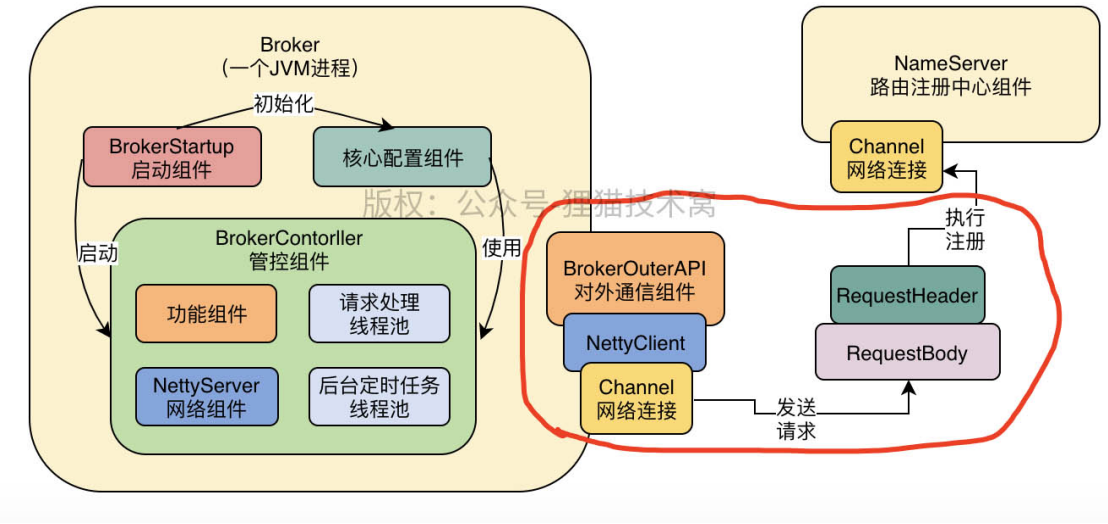
（点击下方蓝字试听）

《从零开始带你成为MySQL实战优化高手》

---

上一次我们分析完了Broker启动的时候是如何通过BrokerOuterAPI发送注册请求到NameServer去的

大家看下图红圈的部分，我们可以回忆一下这个Broker发送注册请求的过程。




今天我们就来研究一下NameServer接收到这个注册请求之后，是如何进行处理的，这里要涉及到Netty网络通信相关的东西，可能很多人没接触过Netty，但是没关系，我尽量弱化掉Netty自身的东西，主要站在通用的网络通信的角度去讲解。

大家如果对Netty不了解的，对一些Netty自己的特殊API也不用过多的去关注，主要了解他的网络通信的流程就可以了。

现在我们回到NamesrvController这个类的初始化的方法里去，也就是NamesrvController.initialize()这个方法

我们看下面的一个源码片段就可以了，我省略了一些无关紧要的代码。

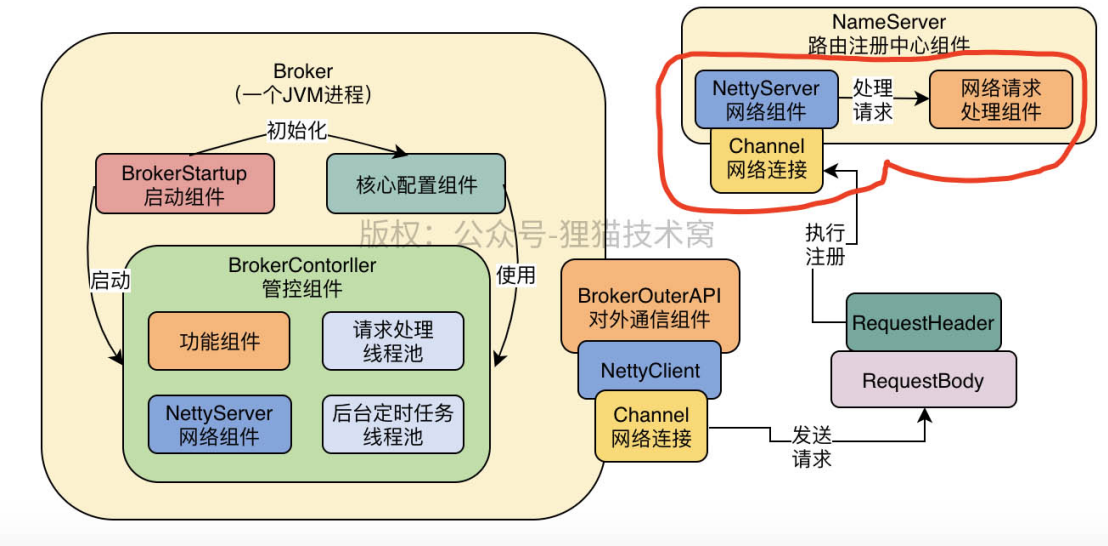
```
public boolean initialize() {

  this.kvConfigManager.load();

  // 在这里初始化了Netty服务器，这个大家应该都没问题了
  this.remotingServer = new NettyRemotingServer(
        this.nettyServerConfig, this.brokerHousekeepingService);

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

  // 非常核心的是下面这行代码，他有一个注册Processor的过程
  // 这个Processor其实就是请求处理器，是NameServer用来处理网络请求的组件
  this.registerProcessor();

  // 省略了一大堆源码

}
```

我们继续看下面的registerProcessor()方法的源码。

```
private void registerProcessor() {
   if (namesrvConfig.isClusterTest()) {

       this.remotingServer.registerDefaultProcessor(new ClusterTestRequestProcessor(this, namesrvConfig.getProductEnvName()),
          this.remotingExecutor);
    } else {
        // 上面那个代码是用于处理测试集群的，核心是下面这行代码
        // 大家可以看到，在这里，把NameServer的默认请求处理组件注册了进去
        // 是注册给了NettyServer的
        // 也就是说，NettyServer接收到的网络请求，都会由这个组件来处理
        this.remotingServer.registerDefaultProcessor(new DefaultRequestProcessor(this), this.remotingExecutor);
    }
}

```

大家看完了上面的源码之后，我来给大家在图里感受一下，在下图中我们可以看到NettyServer是用于接收网络请求的，那么接收到的网络请求给谁处理呢？

其实就是给DefaultRequestProcessor这个请求处理组件来进行处理的。




所以我们如果要知道Broker注册请求是如何处理的，直接就是看DefaultRequestProcessor中的代码就可以了，下面给大家看一下这个类的一些源码片段。

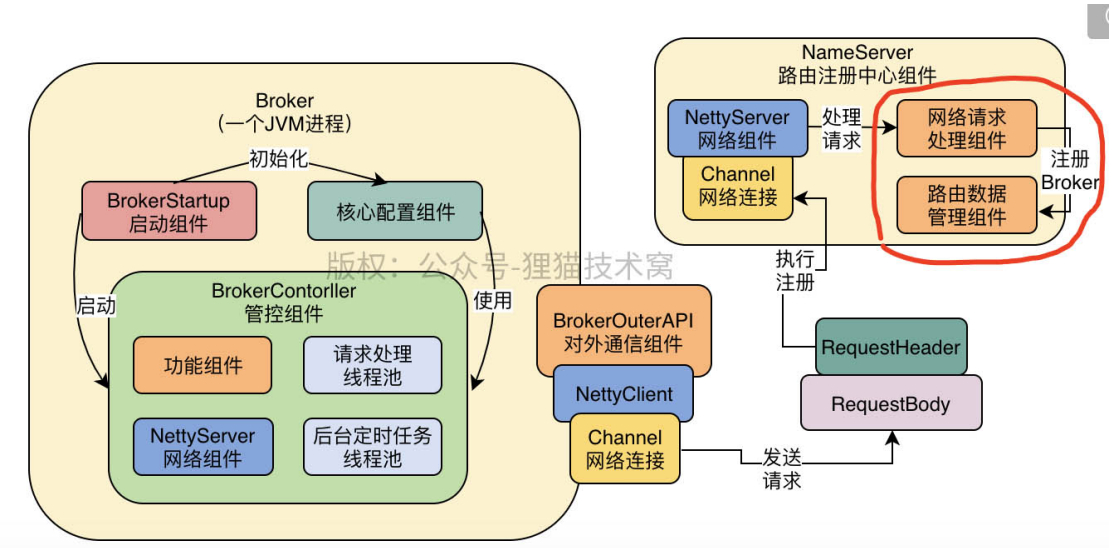
```
@Override
public RemotingCommand processRequest(
        ChannelHandlerContext ctx,
        RemotingCommand request) throws RemotingCommandException {

  // 下面这坨代码就不用管他了，打印调试日志的
  if (ctx != null) {
      log.debug("receive request, {} {} {}",
         request.getCode(),
         RemotingHelper.parseChannelRemoteAddr(ctx.channel()),
         request);
  }

  // 下面就是核心代码了，根据你的请求类型有不同的处理过程
  switch (request.getCode()) {
    case RequestCode.PUT_KV_CONFIG:
      return this.putKVConfig(ctx, request);
    case RequestCode.GET_KV_CONFIG:
      return this.getKVConfig(ctx, request);
    case RequestCode.DELETE_KV_CONFIG:
       return this.deleteKVConfig(ctx, request);
    case RequestCode.QUERY_DATA_VERSION:
       return queryBrokerTopicConfig(ctx, request);
    // 别的请求先不用管了，直接看下面这个好了，就是注册Broker的请求
    case RequestCode.REGISTER_BROKER:
      Version brokerVersion = MQVersion.value2Version(request.getVersion());
      if (brokerVersion.ordinal() >= MQVersion.Version.V3_0_11.ordinal()) {
          return this.registerBrokerWithFilterServer(ctx, request);
      } else {
          // 核心的注册请求处理逻辑，就是在下面的registerBroker方法里
          return this.registerBroker(ctx, request);
      }
     // 省略了一大堆源码
  }
}
```

接着我们进入这个类里的registerBroker()方法，去看看到底如何完成Broker注册的。

```java
public RemotingCommand registerBroker(
        ChannelHandlerContext ctx,
        RemotingCommand request) throws RemotingCommandException {
  final RemotingCommand response = RemotingCommand.createResponseCommand(RegisterBrokerResponseHeader.class);

  // 下面一大坨代码大致看一眼就行了，就是解析注册请求，然后构造返回响应的
  final RegisterBrokerResponseHeader responseHeader = (RegisterBrokerResponseHeader) response.readCustomHeader();
  final RegisterBrokerRequestHeader requestHeader =
     (RegisterBrokerRequestHeader) request.decodeCommandCustomHeader(RegisterBrokerRequestHeader.class);

  if (!checksum(ctx, request, requestHeader)) {
     response.setCode(ResponseCode.SYSTEM_ERROR);
     response.setRemark("crc32 not match");
     return response;
  }

  TopicConfigSerializeWrapper topicConfigWrapper;
  if (request.getBody() != null) {
     topicConfigWrapper = TopicConfigSerializeWrapper.decode(request.getBody(), TopicConfigSerializeWrapper.class);
  } else {
     topicConfigWrapper = new TopicConfigSerializeWrapper();
     topicConfigWrapper.getDataVersion().setCounter(new AtomicLong(0));
     topicConfigWrapper.getDataVersion().setTimestamp(0);
  }
```

```java
  // 核心其实在这里，就是调用了RouteInfoManager这个核心功能组件
  // RouteInfoManager，顾名思义，就是路由信息管理组件，他是一个功能组件
  // 调用了这个功能组件的注册Broker的方法
  RegisterBrokerResult result = this.namesrvController.getRouteInfoManager().registerBroker(
    requestHeader.getClusterName(),
    requestHeader.getBrokerAddr(),
    requestHeader.getBrokerName(),
    requestHeader.getBrokerId(),
    requestHeader.getHaServerAddr(),
    topicConfigWrapper,
    null,
    ctx.channel()
  );

  // 下面都是在构造返回的响应了，大致看一眼就行了
  responseHeader.setHaServerAddr(result.getHaServerAddr());
  responseHeader.setMasterAddr(result.getMasterAddr());

  byte[] jsonValue = this.namesrvController.getKvConfigManager().getKVListByNamespace(NamesrvUtil.NAMESPACE_ORDER_TOPIC_CONFIG);
  response.setBody(jsonValue);
  response.setCode(ResponseCode.SUCCESS);
  response.setRemark(null);
  return response;
}
```

下面我们先在图里给大家体现一下RouteInfoManager这个路由数据管理组件，实际Broker注册就是通过他来做的。



至于RouteInfoManager的注册Broker的方法，我们就不带着大家来看了。这里给大家留一个今天的源码分析小作业，大家可以自己到RouteInfoManager的注册Broker的方法里去看看，最终如何把一个Broker机器的数据放入RouteInfoManager中维护的路由数据表里去的。

其实我这里提示一下，核心思路非常简单，无非就是用一些Map类的数据结构，去存放你的Broker的路由数据就可以了，包括了Broker的clusterName、brokerId、brokerName这些核心数据。

而且在更新的时候，一定会基于Java并发包下的ReadWriteLock进行读写锁加锁，因为在这里更新那么多的内存Map数据结构，必须要加一个写锁，此时只能有一个线程来更新他们才行！

大家看完这个Broker注册的最后一个步骤之后，也就是RouteInfoManager的注册过程之后，有什么源码分析的心得体会，都可以在评论区发表出来，跟大家一起交流。

## End



---



---