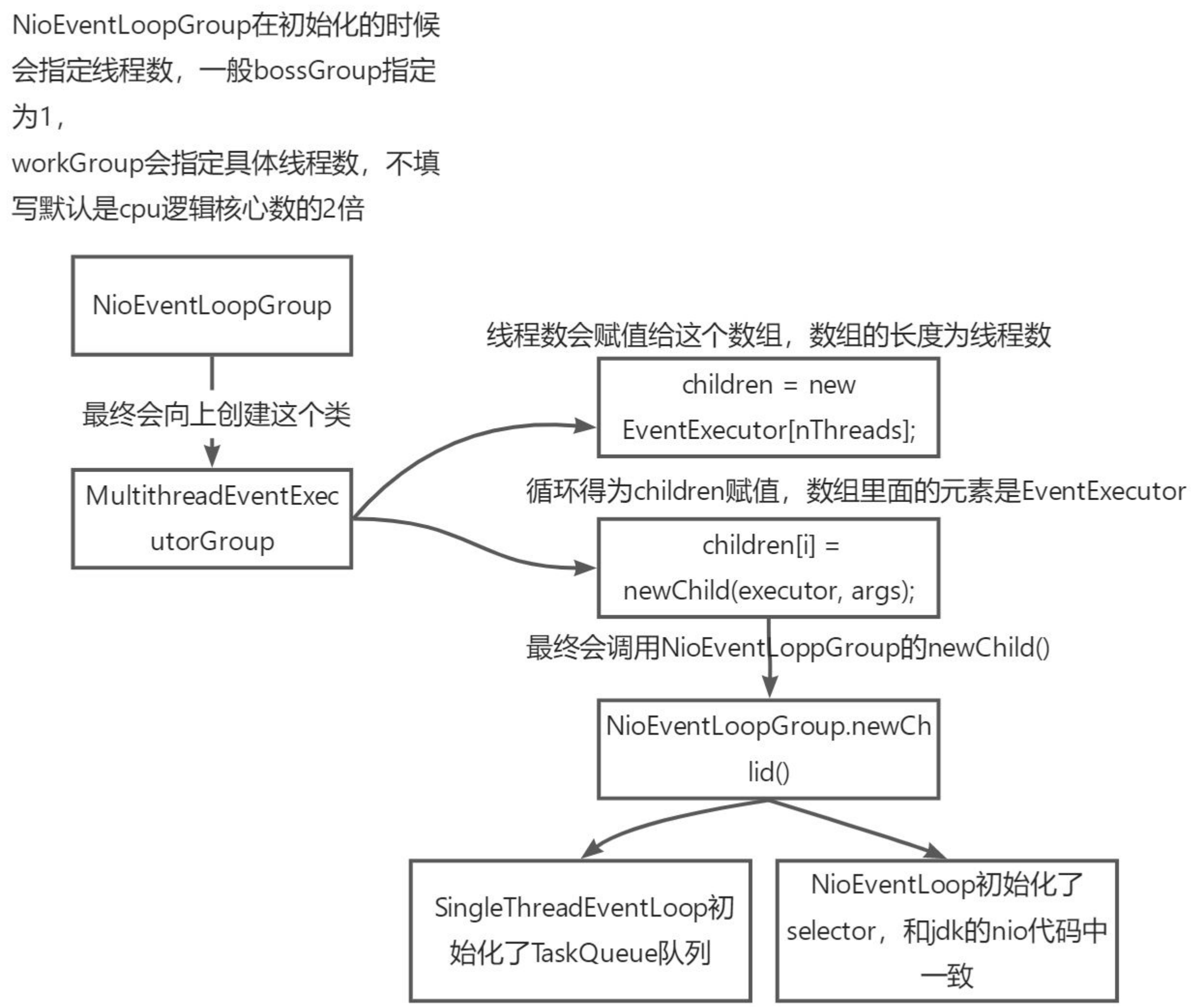
```
mainGroup = new NioEventLoopGroup(tcpConfig.getBossThreadSize());// 处理客户端连接请求数
subGroup = new NioEventLoopGroup(tcpConfig.getBusinessThreadSize());// 真正服务的请求线程数(不填默认是cpu核心数2倍)
```

new NioEventLoopGroup最终会调用到MultithreadEventExecutorGroup()这个类的构造方法。



NioEventLoopGroup在初始化的时候会指定线程数，一般bossGroup指定为1，
workGroup会指定具体线程数，不填写默认是cpu逻辑核心数的2倍
NioEventLoopGroup
最终会向上创建这个类
MultithreadEventExecutorGroup
线程数会赋值给这个数组，数组的长度为线程数
children = new EventExecutor[nThreads];
循环得为children赋值，数组里面的元素是EventExecutor
children[i] = newChild(executor, args);
最终会调用NioEventLoppGroup的newChild()
NioEventLoopGroup.newChild()
SingleThreadEventLoop初始化了TaskQueue队列
NioEventLoop初始化了selector，和jdk的nio代码中一致

```
server = new ServerBootstrap();
server.group(mainGroup, subGroup)

    /**
     * Set the {@link EventLoopGroup} for the parent (acceptor) and the child (client). These
     * {@link EventLoopGroup}'s are used to handle all the events and IO for {@link ServerChannel} and
     * {@link Channel}'s.
     */
    public ServerBootstrap group(EventLoopGroup parentGroup, EventLoopGroup childGroup) {
        super.group(parentGroup);
        if (childGroup == null) {
            throw new NullPointerException("childGroup");
        }
        if (this.childGroup != null) {
            throw new IllegalStateException("childGroup set already");
        }
        this.childGroup = childGroup;
        return this;
    }
```

group方法比较简单，只是把对应的NioEventLoopGroup赋值给对应的成员变量。

```
.channel(NioServerSocketChannel.class) //NioDatagramChannel.class 如果是udp使用这个类 下面设置的option也会不一样


    public B channelFactory(ChannelFactory<? extends C> channelFactory) {
        if (channelFactory == null) {
            throw new NullPointerException("channelFactory");
        }
        if (this.channelFactory != null) {
            throw new IllegalStateException("channelFactory set already");
        }

        this.channelFactory = channelFactory;
        return self();
    }
.channel方法最终会调用到这个方法中来。会创建一个反射工厂。将我们的NioServerSocketChannel.class
赋值给一个成员变量。这个类在后面的源码中会进行初始化，这里只是把它当成成员变量暂时存放起来。
```

channel方法就是将我们传入的class对象保存在一个成员变量中。后面的代码里会通过反射初始化这个类。

```
.option(ChannelOption.SO_BACKLOG, 10240) // 服务端可连接队列大小
```

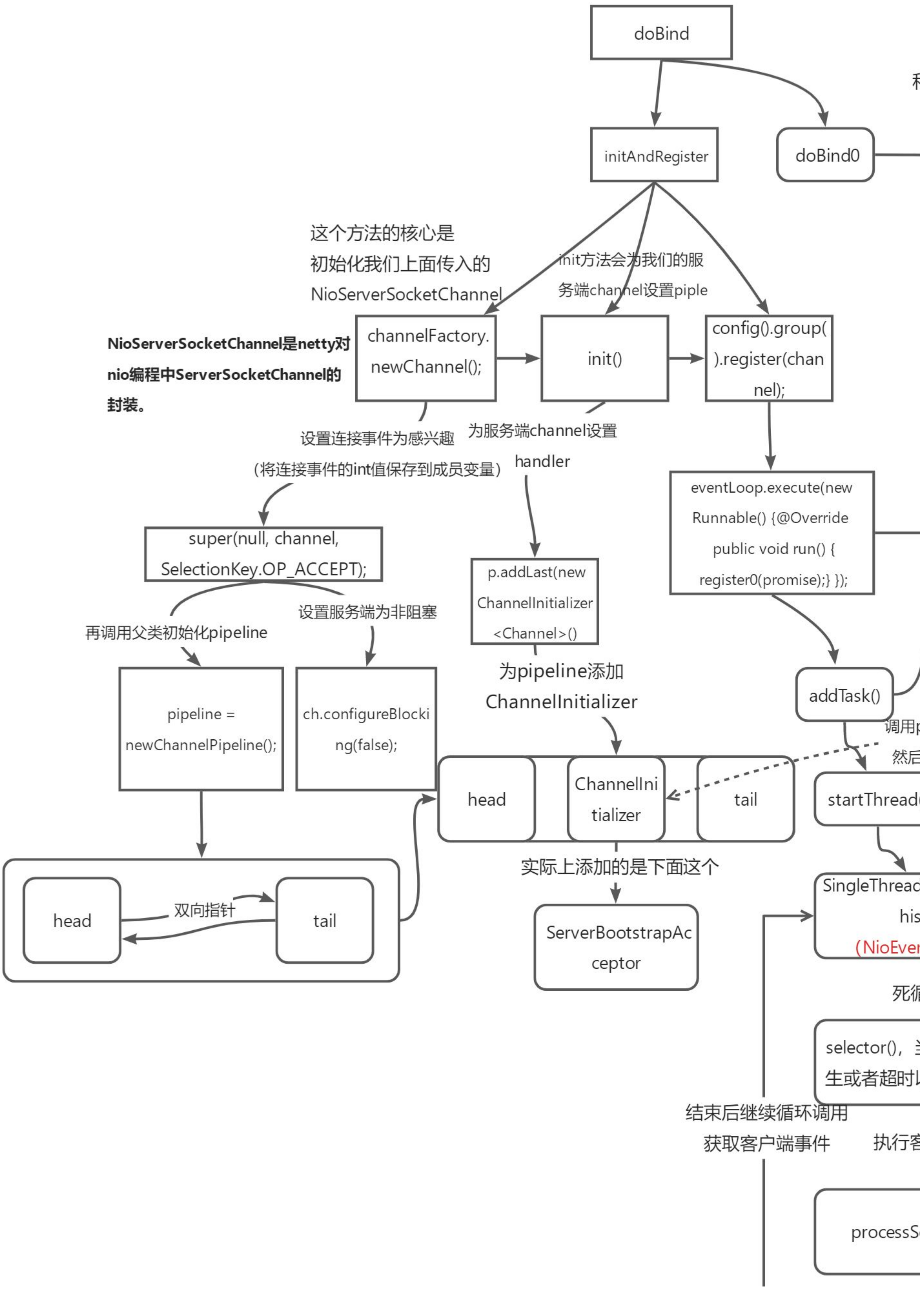
option方法是将我们的一些服务器参数设置到ServerBootstrap中，有点类似于tomcat里面的参数配置。

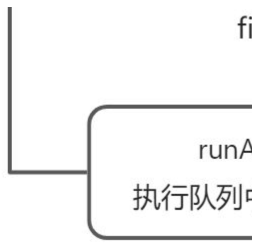
```
.childHandler(new ChannelInitializer<SocketChannel>() {


/**
     * Set the {@link ChannelHandler} which is used to serve the request for the {@link Channel}'s.
     */
    public ServerBootstrap childHandler(ChannelHandler childHandler) {
        if (childHandler == null) {
            throw new NullPointerException("childHandler");
        }
        this.childHandler = childHandler;
        return this;
    }
```

这个代码也是一个简单的成员变量赋值

主线代码到这里就结束了。

```
server.bind(9000);
```




fi
runA
执行队列中