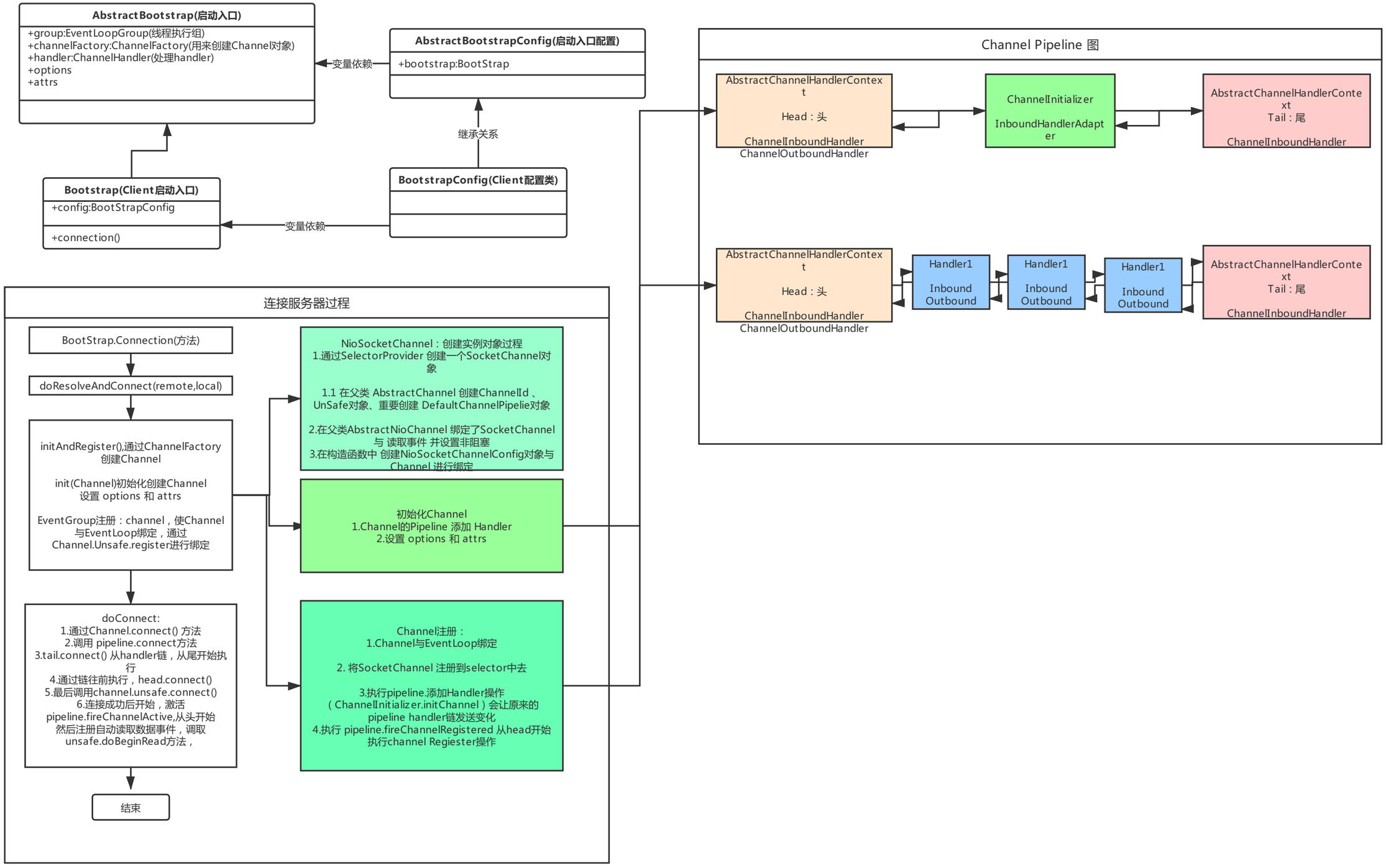
AbstractBootstrap(启动入口)
+group:EventLoopGroup(线程执行组)
+channelFactory:ChannelFactory(用来创建Channel对象)
+handler:ChannelHandler(处理handler)
+options
+attrs
变量依赖
AbstractBootstrapConfig(启动入口配置)
+bootstrap:BootStrap
继承关系
Bootstrap(Client启动入口)
+config:BootStrapConfig
+connection()
变量依赖
BootstrapConfig(Client配置类)
Channel Pipeline 图
AbstractChannelHandlerContext
Head：头
ChannelInboundHandler
ChannelOutboundHandler
ChannelInitializer
InboundHandlerAdapter
AbstractChannelHandlerContext
Tail：尾
ChannelInboundHandler
AbstractChannelHandlerContext
Head：头
ChannelInboundHandler
ChannelOutboundHandler
Handler1
Inbound
Outbound
Handler1
Inbound
Outbound
Handler1
Inbound
Outbound
AbstractChannelHandlerContext
Tail：尾
ChannelInboundHandler
连接服务器过程
BootStrap.Connection(方法)
doResolveAndConnect(remote,local)
initAndRegister(),通过ChannelFactory创建Channel
init(Channel)初始化创建Channel
设置 options 和 attrs
EventGroup注册：channel，使Channel与EventLoop绑定，通过Channel.Unsafe.register进行绑定
doConnect:
1.通过Channel.connect() 方法
2.调用 pipeline.connect方法
3.tail.connect() 从handler链，从尾开始执行
4.通过链往前执行，head.connect()
5.最后调用channel.unsafe.connect()
6.连接成功后开始，激活pipeline.fireChannelActive,从头开始
然后注册自动读取数据事件，调取unsafe.doBeginRead方法，
结束
NioSocketChannel：创建实例对象过程
1.通过SelectorProvider 创建一个SocketChannel对象
1.1 在父类 AbstractChannel 创建ChannelId 、UnSafe对象、重要创建 DefaultChannelPipelie对象
2.在父类AbstractNioChannel 绑定了SocketChannel与 读取事件 并设置非阻塞
3.在构造函数中 创建NioSocketChannelConfig对象与Channel 进行绑定
初始化Channel
1.Channel的Pipeline 添加 Handler
2.设置 options 和 attrs
Channel注册：
1.Channel与EventLoop绑定
2. 将SocketChannel 注册到selector中去
3.执行pipeline.添加Handler操作
（ChannelInitializer.initChannel）会让原来的pipeline handler链发送变化
4.执行 pipeline.fireChannelRegistered 从head开始执行channel Regiester操作