

初始化长连接服务

```
chain.boot()
    .setNext(new CacheManagerBoot())//1.初始化缓存模块
    .setNext(new ServiceRegistryBoot())//2.启动服务注册与发现模块
    .setNext(new ServiceDiscoveryBoot())//2.启动服务注册与发现模块
    .setNext(new ServerBoot(mPushServer.getConnectionServer(), mPushServer.getConnServerNode()))//3.启动接入服务
    .setNext(() -> new ServerBoot(mPushServer.getWebsocketServer(), mPushServer.getWebsocketServerNode()), wsEnabled())
    .setNext(() -> new ServerBoot(mPushServer.getUdpGatewayServer(), mPushServer.getGatewayServerNode()), udpGatewayEnabled())
    .setNext(() -> new ServerBoot(mPushServer.getGatewayServer(), mPushServer.getGatewayServerNode()), tcpGatewayEnabled())//
    .setNext(new ServerBoot(mPushServer.getAdminServer(), null))//7.启动控制台服务
    .setNext(new RouterCenterBoot(mPushServer))//8.启动路由中心组件
    .setNext(new PushCenterBoot(mPushServer))//9.启动推送中心组件
    .setNext(() -> new HttpProxyBoot(mPushServer), CC.mp.http.proxy_enabled)//10.启动http代理服务, dns解析服务
    .setNext(new MonitorBoot(mPushServer))//11.启动监控服务
    .end();
```

服务启动

```
ServerBoot start()
@Override
public void start() {
    1 server.init();
    2 server.start(new Listener() {
        @Override
        public void onSuccess(Object... args) {
            Logs.Console.info("start {} success on:{}", server.getClass().getSimpleName(), args[0]);
            if (node != null) { //注册应用到zk
                3 ServiceRegistryFactory.create().register(node);
                Logs.RSD.info("register {} to srd success.", node);
            }
            startNext();
        }

        @Override
        public void onFailure(Throwable cause) {
            Logs.Console.error("start {} failure, jvm exit with code -1", server.getClass().getSimpleName());
            System.exit(-1);
        }
    });
}

@Override
protected void stop() {
    stopNext();
    if (node != null) {
        ServiceRegistryFactory.create().deregister(node);
    }
    Logs.Console.info("try shutdown {}...", server.getClass().getSimpleName());
    server.stop().join();
    Logs.Console.info("{} shutdown success.", server.getClass().getSimpleName());
}
```

- 1、调用ConnectionServer#init()
- 2、调用ConnectionServer#start()
- 3、将CS节点信息注册到Zookeeper

```

ConnectionServer start()

@Override
public void init() {
    super.init(); 1.1
    connectionManager.init(); 1.2
    messageDispatcher.register(Command.HEARTBEAT, HeartBeatHandler::new); 1.3
    messageDispatcher.register(Command.HANDSHAKE, () -> new HandshakeHandler(mPushServer));
    messageDispatcher.register(Command.BIND, () -> new BindUserHandler(mPushServer));
    messageDispatcher.register(Command.UNBIND, () -> new BindUserHandler(mPushServer));
    messageDispatcher.register(Command.FAST_CONNECT, () -> new FastConnectHandler(mPushServer));
    messageDispatcher.register(Command.PUSH, PushHandlerFactory::create);
    messageDispatcher.register(Command.ACK, () -> new AckHandler(mPushServer));
    messageDispatcher.register(Command.HTTP_PROXY, () -> new HttpProxyHandler(mPushServer), CC.mp.http.proxy);

    if (CC.mp.net.traffic_shaping.connect_server.enabled) { //启用流量整形, 限流
        trafficShapingExecutor = Executors.newSingleThreadScheduledExecutor(new NamedPoolThreadFactory(T_TRAFFIC_SHAPING));
        trafficShapingHandler = new GlobalChannelTrafficShapingHandler(
            trafficShapingExecutor,
            write_global_limit, read_global_limit,
            write_channel_limit, read_channel_limit,
            check_interval);
    }
}

@Override
public void start(Listener listener) {
    super.start(listener); 2.1
    if (this.workerGroup != null) { // 增加线程池监控
        mPushServer.getMonitor().monitor("conn-worker", this.workerGroup); 2.2
    }
}

```

1.1 调用NettyTCPServer#init()

主要是利用AtomicReference<State>的cas判断netty服务是不是启动，如果已经启动则抛出ServiceException异常；

1.2 调用ServerConnectionManager#init()

这里主要是心跳连接管理，该方法里面会初始化HashedWheelTimer用于连接超时管理；

1.3 注册各种消息的处理类

2.1 调用NettyTCPServer#start()方法，创建ServerBootstrap启动Netty长连接服务；

NettyTCPServer创建netty ServerBootstrap服务时，会调用其子类

ConnectionServer中方法：

initPipeline()：在已有的Pipeline最前面加入trafficShapingHandler限流处理；

initOptions(): 设置发送，接收BUF缓冲区大小、设置Bytebuf高低水位，用于控制写入数据的量；

getChannelHandler(): 设置netty 事件处理类ServerChannelHandler，处理建连、消息、断连、异常事件；

2.2 将workerGroup线程池加入到监控中；