初始化HTTP代理服务、DNS解析服务

```
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
        .setNext(() -> new ServerBoot(mPushServer.getWebsocketServer(), mPushServer.getWebsocketServerNode()),
        .setNext(() -> new ServerBoot(mPushServer.getUdpGatewayServer(), mPushServer.getGatewayServerNode()), uc
        .setNext(() -> new ServerBoot(mPushServer.getGatewayServer(), mPushServer.getGatewayServerNode()), tcpGa
        .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();
```

启动服务

```
public final class HttpProxyBoot extends BootJob {

    private final MPushServer mPushServer;

    public HttpProxyBoot(MPushServer mPushServer) {
        this.mPushServer = mPushServer;
    }

    @Override
    protected void start() {
        mPushServer.getHttpClient().syncStart();
        DnsMappingManager.create().start();

        startNext();
    }

    @Override
    protected void stop() {
        stopNext();
        mPushServer.getHttpClient().syncStop();
        DnsMappingManager.create().stop();
    }
}
```

1、初始化NettyHttpClient

获取单例模式的NettyHttpClient实例，然后syncStart()将会调用NettyHttpClient#doStart()

2、初始化DNS

通过SPI，找到mpush-common模块中DnsMappingManager接口的实现类HttpProxyDnsMappingManager，得到HttpProxyDnsMappingManager实例；然后start()将会调用HttpProxyDnsMappingManager#doStart();

HTTP代理服务

```
NettyHttpClient  doStart()

@Override
protected void doStart(Listener listener) throws Throwable {
1   workerGroup = new NioEventLoopGroup(http_work, new DefaultThreadFactory(ThreadNames.T_HTTP_CLIENT));
    b = new Bootstrap();
2   b.group(workerGroup);
    b.channel(NioSocketChannel.class);
    b.option(ChannelOption.SO_KEEPALIVE, true);
    b.option(ChannelOption.TCP_NODELAY, true);
    b.option(ChannelOption.SO_REUSEADDR, true);
    b.option(ChannelOption.CONNECT_TIMEOUT_MILLIS, 4000);
    b.option(ChannelOption.ALLOCATOR, PooledByteBufAllocator.DEFAULT);
    b.handler(new ChannelInitializer<SocketChannel>() {
        @Override
        public void initChannel(SocketChannel ch) throws Exception {
            ch.pipeline().addLast("decoder", new HttpResponseDecoder()); 3
            ch.pipeline().addLast("aggregator", new HttpObjectAggregator(maxContentLength)); 4
            ch.pipeline().addLast("encoder", new HttpRequestEncoder()); 5
            ch.pipeline().addLast("handler", new HttpClientHandler(NettyHttpClient.this)); 6
        }
    });
    timer = new HashedWheelTimer(new NamedThreadFactory(T_HTTP_TIMER), 1, TimeUnit.SECONDS, 64); 7
    listener.onSuccess();
}

@Override
protected void doStop(Listener listener) throws Throwable {
    pool.close();
    workerGroup.shutdownGracefully();
    timer.stop();
    listener.onSuccess();
}
```

1 初始化事件循环组

2 实例化Bootstrap()

ChannelOption.SO_KEEPALIVE 启用心跳保活机制

ChannelOption.TCP_NODELAY 开启低延迟

ChannelOption.SO_REUSEADDR 用于UDP，同一端口绑定多个套接字

ChannelOption.CONNECT_TIMEOUT_MILLIS 连接超时

ChannelOption.ALLOCATOR Netty4使用对象池，重用缓冲区

3 http 响应解码器

4 Http消息聚合，把多个HTTP请求中的数据组装成一个

5 http 请求编码器

6 http处理器

7 初始化时间轮，用于HTTP请求超时控制

DNS服务

HttpProxyDnsMappingManager

```
private Map<String, List<DnsMapping>> available = Maps.newConcurrentMap();

private ScheduledExecutorService executorService;

@Override
protected void doStart(Listener listener) throws Throwable {
    ServiceDiscovery discovery = ServiceDiscoveryFactory.create();  3
    discovery.subscribe(DNS_MAPPING, this);  4
    discovery.lookup(DNS_MAPPING).forEach(this::add);  5

    if (all.size() > 0) {
        executorService = Executors.newSingleThreadScheduledExecutor(  6
                new NamedPoolThreadFactory(ThreadNames.T_HTTP_DNS_TIMER)
        );
        executorService.scheduleAtFixedRate(this, 1, 20, TimeUnit.SECONDS); //20秒 定时扫描dns  7
    }
    listener.onSuccess();
}

@Override
protected void doStop(Listener listener) throws Throwable {
    if (executorService != null) {
        executorService.shutdown();
    }
    listener.onSuccess();
}

@Override
public void init() {
    all.putAll(CC.mp.http.dns_mapping);  1
    available.putAll(CC.mp.http.dns_mapping);  2
}
```

1 获取DNS配置，域名：LIST[IP:PORT]

```
#HTTP代理配置, 见reference.conf
http {
 proxy-enabled=false //启用Http代理
 max-conn-per-host=5 //每个域名的最大链接数, 建议web服务nginx超时时间设长一点, 以便保持长链接
 default-read-timeout=10s //请求超时时间
 max-content-length=5m //response body 最大大小
 dns-mapping { //域名映射外网地址转内部IP, 域名部分不包含端口号
 //"mpush.com":["127.0.0.1:8080", "127.0.0.1:8081"]
 }
}
```

```
// CC.java
interface http {
 Map<String, List<DnsMapping>> dns_mapping = loadMapping();
 static Map<String, List<DnsMapping>> loadMapping() {
```

```
Map<String, List<DnsMapping>> map = new HashMap<>();
cfg.getObject("dns-mapping").forEach((s, v) ->
map.put(s, ConfigList.class.cast(v)
.stream()
.map(cv -> DnsMapping.parse((String) cv.unwrapped()))
 .collect(toCollection(ArrayList::new))
 )
 );
 return map;
 }
}
```

2 获取DNS配置，域名：LIST[IP:PORT]

3 通过SPI，找到mpush-zk模块中ServiceDiscoveryFactory接口的实现类ZKDiscoveryFactory，得到ZKServiceRegistryAndDiscovery实例；

4 订阅事件

String DNS_MAPPING = "/dns/mapping";

5 获取注册在ZK上的DNS_MAPPING 的服务信息

6 创建线程池，用于定时扫描DNS服务做健康检查

7 运行DNS健康检查服务