入口

```
Main  main()
 *
 * @author ohun@live.cn (夜色)
 */
public final class Main {

    public static void main(String[] args) {
        Logs.init(); 1
        Logs.Console.info("launch alloc server...");
        AllocServer server = new AllocServer();
        server.start(); 2
        addHook(server);
    }

    private static void addHook(AllocServer server) {
        Runtime.getRuntime().addShutdownHook(
                new Thread(() -> {
                    try {
                        server.stop();
                    } catch (Exception e) {
                        Logs.Console.error("alloc server stop ex", e);
                    }
                    Logs.Console.info("jvm exit, all service stopped...");

                }, "mpush-shutdown-hook-thread")
        );
    }
}
```

1、初始化日志服务

2、初始化AllocServer服务，并调用start()方法启动服务

调用start()，然后会先调用AllocServer#init()方法，然后才是AllocServer#doStart()

```
AllocServer  init()
 */
public final class AllocServer extends BaseService {

    private HttpServer httpServer;
    private AllocHandler allocHandler;
    private PushHandler pushHandler;

    @Override
    public void init() {
        int port = CC.mp.net.cfg.getInt("alloc-server-port");  1
        boolean https = "https".equals(CC.mp.net.cfg.getString("alloc-server-protocol")); 2

        this.httpServer = HttpServerCreator.createServer(port, https); 3
        this.allocHandler = new AllocHandler(); 4
        this.pushHandler = new PushHandler(); 5

    6   httpServer.setExecutor(Executors.newCachedThreadPool());//设置线程池，由于是纯内存操作，不需要队列
        httpServer.createContext("/", allocHandler);//查询mpush机器
        httpServer.createContext("/push", pushHandler);//模拟发送push
        httpServer.createContext("/index.html", new IndexPageHandler());//查询mpush机器
    }
```

1、获取alloc服务的端口9999

```
1  mp.log-level=${log.level}
```

```
mp.net.gateway-server-net=tcp // 网关服务使用的网络 udp/tcp
mp.net.alloc-server-port=9999
mp.net.alloc-server-protocol=http
mp.zk.server-address="127.0.0.1:2181"
mp.redis {// redis 集群配置
 password:""
 nodes:["127.0.0.1:6379"] //格式是ip:port
 cluster-model:single //single, cluster
}
```

2、获取alloc服务的HTTP协议(http 或者 https)

3、用JDK API创建HTTP服务

4、初始化AlloHandler处理器，处理"/"请求，实现HttpHandler，从ZK获取MPUSH服务地址

5、初始化PushHandler处理器，处理“/push” 请求，实现HttpHandler，模拟推送消息给MPUSH

6、初始化IndexPageHandler处理器，处理“/index.html” 请求，实现HttpHandler，模拟发起push请求

```
@Override
protected void doStart(Listener listener) throws Throwable {
    pushHandler.start(); 1
    allocHandler.start(); 2
    httpServer.start();  3
    Logs.Console.info("================================================================");
    Logs.Console.info("====================ALLOC SERVER START SUCCESS====================");
    Logs.Console.info("================================================================");
}

@Override
protected void doStop(Listener listener) throws Throwable {
    httpServer.stop(0);//1 min
    pushHandler.stop();
    allocHandler.stop();
    Logs.Console.info("================================================================");
    Logs.Console.info("====================ALLOC SERVER STOPPED SUCCESS====================");
    Logs.Console.info("================================================================");
}
```

1、启动加载PushClient的一系列服务，如MpushClient、服务发现、缓存、事件BUS

```
PushClient  doStart()

    protected void doStart(Listener listener) throws Throwable {
        if(this.mPushClient == null) {
            this.mPushClient = new MPushClient();
        }

        this.pushRequestBus = this.mPushClient.getPushRequestBus();
        this.cachedRemoteRouterManager = this.mPushClient.getCachedRemoteRouterManager();
        this.gatewayConnectionFactory = this.mPushClient.getGatewayConnectionFactory();
        ServiceDiscoveryFactory.create().syncStart();
        CacheManagerFactory.create().init();
        this.pushRequestBus.syncStart();
        this.gatewayConnectionFactory.start(listener);
    }
```

2、启动缓存服务、发现服务、订阅"/cluster/cs" ZK节点、初始化调度线程池

```
/*package*/ final class AllocHandler implements HttpHandler {

    private List<ServerNode> serverNodes = Collections.emptyList();
    private ScheduledExecutorService scheduledExecutor;
    private final ServiceDiscovery discovery = ServiceDiscoveryFactory.create();
    private final UserManager userManager = new UserManager(null);

    public void start() {
        CacheManagerFactory.create().init(); //启动缓冲服务

        ServiceDiscovery discovery = ServiceDiscoveryFactory.create();// 启动发现服务
        discovery.syncStart();
        discovery.subscribe(ServiceNames.CONN_SERVER, new ConnServerNodeListener());


        scheduledExecutor = Executors.newSingleThreadScheduledExecutor();
        scheduledExecutor.scheduleAtFixedRate(this::refresh, 0, 5, TimeUnit.MINUTES);
    }

    public void stop() {
        discovery.syncStop();
        CacheManagerFactory.create().destroy();
        scheduledExecutor.shutdown();
    }
```

定时调度器，每5分钟去ZK拿到MPUSH服务地址列表，并按在线用户量排序，然后设置到本地缓存中；

待客户端访问alloc服务时，直接从本地缓存中直接获取；

```
AllocHandler  refresh()
    }

    /**
     * 从zk中获取可提供服务的机器,并以在线用户量排序
     */
    private void refresh() {
        //1.从缓存中拿取可用的长链接服务器节点
        List<ServiceNode> nodes = discovery.lookup(ServiceNames.CONN_SERVER);
        if (nodes.size() > 0) {
            //2.对serverNodes可以按某种规则排序,以便实现负载均衡,比如:随机,轮询,链接数量等
            this.serverNodes = nodes
                    .stream()
                    .map(this::convert)
                    .sorted(ServerNode::compareTo)
                    .collect(Collectors.toList());
        }
    }
```

3、启动HTTP服务