客户端访问alloc服务，获取服务列表：

GET http://xxxx:9999/

返回内容：ip:port, ip:port

JDK HTTP server在接收到请求时，会根据context找到相应HttpHanlder，调用AllocHandler#handle()

AllocServer init()

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

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

    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
}
```

```
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();
    }

    public void handle(HttpExchange httpExchange) throws IOException {
        //3.格式组装 ip:port,ip:port
```

```
        StringBuilder sb = new StringBuilder();
        Iterator<ServerNode> it = serverNodes.iterator();
        if (it.hasNext()) {
            ServerNode node = it.next();
            sb.append(node.host).append(':').append(node.port);
        }

        while (it.hasNext()) {
            ServerNode node = it.next();
            sb.append(',').append(node.host).append(':').append(node.port);
        }

        byte[] data = sb.toString().getBytes(Constants.UTF_8);
        httpExchange.sendResponseHeaders(200, data.length);//200, content-length
        OutputStream out = httpExchange.getResponseBody();
        out.write(data);
        out.close();
        httpExchange.close();
    }

    /**
     * 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());
        }
    }

    private long getOnlineUserNum(String publicIP) {
        return userManager.getOnlineUserNum(publicIP);
    }

    private ServerNode convert(ServiceNode node) {
        String public_ip = node.getAttr(ServiceNames.ATTR_PUBLIC_IP);
        if (public_ip == null) {
            public_ip = node.getHost();
        }
        long onlineUserNum = getOnlineUserNum(public_ip);
        return new ServerNode(public_ip, node.getPort(), onlineUserNum);
    }

    private class ConnServerNodeListener implements ServiceListener {

        @Override
        public void onServiceAdded(String s, ServiceNode serviceNode) {
            refresh();
        }

        @Override
        public void onServiceUpdated(String s, ServiceNode serviceNode) {
            refresh();
        }
```

```
        @Override
        public void onServiceRemoved(String s, ServiceNode serviceNode) {
            refresh();
        }
    }

    private static class ServerNode implements Comparable<ServerNode> {
        long onlineUserNum = 0;
        String host;
        int port;

        public ServerNode(String host, int port, long onlineUserNum) {
            this.onlineUserNum = onlineUserNum;
            this.host = host;
            this.port = port;
        }

        @Override
        public int compareTo(ServerNode o) {
            return Long.compare(onlineUserNum, o.onlineUserNum);
        }
    }
}
```