技术大佬灵魂质问：你们对外提供的API接口竟然没加黑白名单限制！！！
==========

技术大佬灵活质问：你们对外提供的API接口竟然没加黑白名单限制！！！
==========

1.概述
---

IP 黑白名单是一种常见的网络安全机制，用于控制对特定资源或服务的访问。黑名单列出了被拒绝访问的 IP 地址，而白名单列出了被允许访问的 IP 地址。通过这种方式，可以有效地防止未经授权的访问，提高系统的安全性。对于对外提供的API接口，通过设置IP黑白名单，确保只有经过认证或信任的系统和客户端才能调用接口。可以阻止已知的恶意IP地址或曾经尝试攻击系统的IP地址，防止这些来源对服务器进行未经授权的访问、扫描、攻击等行为。

**在项目系统服务代码中实现IP黑白名单和在Nginx中设置IP黑白名单都是可行的方式**，具体取决于您的需求和架构。以下是它们各自的优缺点和适用场景：

**在项目中实现IP黑白名单：**

优点：

1. 更灵活：您可以根据项目的需要实现自定义的逻辑，例如根据用户角色、请求路径等动态设置IP黑白名单。
2. 可扩展性：您可以轻松地将IP黑白名单与项目中的其他功能集成，实现更多定制化的安全策略。
3. 方便调试：由于黑白名单是在项目内部实现的，因此您可以更方便地调试和查看日志，进行故障排除。

缺点：

1. 效率问题：如果IP黑白名单需要频繁更新或包含大量IP地址，可能会影响项目的性能和响应速度。
2. 需要额外开发工作：需要开发人员编写代码来实现IP黑白名单功能，增加了开发成本和时间成本。

适用场景：

* 当您需要根据项目的特定需求或逻辑实现IP黑白名单时。
* 当您希望将IP黑白名单与项目中的其他功能集成时。

**在Nginx中设置IP黑白名单：**

优点：

1. 高效：Nginx具有高性能和高并发处理能力，可以在网络层面上快速地过滤和阻止恶意IP地址。
2. 简单快捷：Nginx配置简单，您只需编辑配置文件即可设置IP黑白名单，无需修改项目代码。
3. 减轻服务器负载：通过Nginx过滤IP黑白名单可以减少请求到达项目的数量，减轻了项目服务器的负载压力。

缺点：

1. 不灵活：Nginx的配置是静态的，无法根据项目的业务逻辑动态调整IP黑白名单。
2. 调试困难：由于IP黑白名单是在网络层面实现的，调试和日志记录相对困难，难以进行精细化的排查和调整。

适用场景：

* 当您需要在网络层面快速过滤恶意IP地址时。
* 当您希望简单快捷地实现IP黑白名单功能，而不涉及项目代码的修改和开发工作。

2.Spring Boot基于拦截器实现黑白名单限制
---

**实现思路：** 添加拦截器，拦截项目所有的接口请求，获取请求的网络IP，查询IP是否在白名单之中，有就通过无拒绝访问，匹配黑名单则反之，黑白名单可以放到项目配置文件、缓存redis、数据库MySQL等等。还可以结合登录用户角色相关逻辑进行黑白名单限制。

**获取请求的网络IP的工具类**

```
public class IpUtil {
    private static final String UNKNOWN = "unknown";

    public static String getIpAddress() {
        HttpServletRequest request = ((ServletRequestAttributes) RequestC
ontextHolder.getRequestAttributes()).getRequest();
        String ip = request.getHeader("x-forwarded-for");
        if (StringUtils.isBlank(ip) || UNKNOWN.equalsIgnoreCase(ip)) {
            ip = request.getHeader("Proxy-Client-IP");
        }
        if (StringUtils.isBlank(ip) || UNKNOWN.equalsIgnoreCase(ip)) {
            ip = request.getHeader("WL-Proxy-Client-IP");
        }
        if (StringUtils.isBlank(ip) || UNKNOWN.equalsIgnoreCase(ip)) {
            ip = request.getRemoteAddr();
        }

        // 多个代理的情况，第一个IP为客户端真实IP,多个IP按照','分割
        if (StringUtils.isNotBlank(ip) && ip.indexOf(',') > 0) {
            ip = ip.substring(0, ip.indexOf(","));
        }
        return ip;
    }
}
```

添加拦截器拦截所有接口进行请求ip黑白名单验证，这里可根据相应需求灵活实现判断。

```
@Component
@Slf4j
public class IpAccessInterceptor implements HandlerInterceptor {


    @Value("${ip.black}")
    private Set<String> blackIpList;
```

```
    @Value("${ip.white}")
    private Set<String> whiteIpList;


    private final static String LOCAL = "127.0.0.1";
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletRes
ponse response, Object handler) throws Exception {
        String ip = IpUtil.getIpAddress();
        if (Objects.equals(ip, LOCAL)) {
            // 本机直接放过
            return true;
        }
        // 获取黑白名单：从配置文件、或者缓存、数据库都可以，这里还可以
定制化判断，比如结合登录人角色一起判断都可以
        // 这里为了方便演示，我从配置文件读取
        if (blackIpList.contains(ip)) {
            // 在黑名单中直接拒绝访问
            log.info("ip:{} 在黑名单中拒绝访问.....", ip);
            return false;
        }
        if (!whiteIpList.contains(ip)) {
            // 不在白名单中，也拒绝访问
            log.info("ip:{} 不在白名单中拒绝访问.....", ip);
            return false;
        }
        // 验证通过
        return true;
    }
}

```

配置拦截器

```
@Configuration
public class InterceptorConfig implements WebMvcConfigurer {
    @Resource
    private IpAccessInterceptor ipAccessInterceptor;

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        // registry.addInterceptor(new IpAccessInterceptor()); new 拦截器会
导致拦截器不能成功依赖注入
        registry.addInterceptor(ipAccessInterceptor).addPathPatterns("/**"
```

```
);
    }
}
```

测试接口：

```
    @GetMapping("/filter")
    public String testFilter() {
        log.info("controller业务方法执行了");
        return "hello";
    }
```

配置文件：

```
ip:
  black: 10.0.0.10, 10.0.0.18, 10.33.194.35
  white: 10.33.194.36, 10.33.194.10
```

请求接口：[http://10.33.194.35:8080/test/filter控制台输出如下：](http://cxyroad.com/"http://10.33.194.35:8080/test/filter%E6%8E%A7%E5%88%B6%E5%8F%B0%E8%BE%93%E5%87%BA%E5%A6%82%E4%B8%8B%EF%BC%9A")

```
INFO com.shepherd.basedemo.interceptor.IpAccessInterceptor preHandle [http-nio-8080-exec-1@82013] : ip:10.33.194.35 在黑名单中拒绝访问.....
```

> **项目推荐**：基于SpringBoot2.x、SpringCloud和SpringCloudAlibaba企业级系统架构底层框架封装，解决业务开发时常见的非功能性需求，防止重复造轮子，方便业务快速开发和企业技术栈框架统一管理。引入组件化的思想实

现高内聚低耦合并且高度可配置化，做到可插拔。严格控制包依赖和统一版本管理，做到最少化依赖。注重代码规范和注释，非常适合个人学习和企业使用
>
>
> **Github地址**：[github.com/plasticene/...](http://cxyroad.com/”https://github.com/plasticene/plasticene-boot-starter-parent”)
>
>
> **Gitee地址**：[gitee.com/plasticene3...](http://cxyroad.com/”https://gitee.com/plasticene3/plasticene-boot-starter-parent”)
>
>
> **公众号**：**Shepherd进阶笔记**
>
>
> **交流探讨qun：Shepherd_126**

3.在 Nginx 中设置 IP 黑白名单
---

你也可以在 Nginx 中配置 IP 黑白名单。以下是一个简单的示例：

##### 配置 Nginx

打开 Nginx 配置文件（例如：`/etc/nginx/nginx.conf`），添加以下配置：

```
nginx
复制代码
http {
    ...

    # 允许的 IP 白名单
    allow 192.168.1.100;
    allow 192.168.1.101;

    # 拒绝的 IP 黑名单
    deny 192.168.1.200;
    deny 192.168.1.201;

    server {
        listen 80;
```

```
        server_name example.com;

        location / {
            # 如果 IP 地址不在白名单内且不在黑名单内，则拒绝访问
            allow all;
            deny all;

            proxy_pass http://your_backend_server;
        }
    }

    ...
}

```

然后使用`nginx -s reload` 重新加载配置文件即可

4.总结
---

不知道你有没有调过外部open接口？一般这些接口都是不能直接访问，服务方需要你提供访问接口的ip给你加白名单，这个ip也就是我们访问open接口的服务器的出口ip，使用命令`curl ip.me`查看即可，注意这个出口ip不固定的话请联系网管操作固定，不然动态的话配置黑白名单就无法操作判断了，至于服务方是通过Nginx限制还是代码拦截器限制都可以，可以灵活实现。到这里黑白名单限制就算总结完了，业务系统进行黑白名单限制也是拒绝接口裸奔，保证系统安全的常见措施之一，需了然于胸。

原文链接: https://juejin.cn/post/7369582512236167194