## 1.概述

cross-origin是实现了[Cross-Origin Resource Sharing](http://www.w3.org/TR/cors/)（以下简称CORS）规范的服务器端filter。主要实现的功能是根据CORS规范，在跨域请求访问相关资源前进行过滤操作。本文仅对cross-origin filter配置和使用进行介绍，有关CORS的内容，请参考规范文档以及相关PPT。

## 2.配置说明

### 2.1添加依赖

如果是maven工程，在pom.xml文件中加入依赖即可。

<dependency>

<groupId>com.netease.wd</groupId>

<artifactId>cross-origin</artifactId>

<version>0.0.1</version>

</dependency>

如果是非maven工程，将jar包加入到工程的classpath中即可。

### 2.2配置web.xml文件

#### 2.2.1配置filter

在web.xml文件中引入cross-origin filter的配置

<filter>

<filter-name>cross-origin</filter-name>

<filter-class>

com.netease.wd.crossorigin.filter.CrossOriginFilter

</filter-class>

</filter>

并添加需要的初始化参数。初始化参数说明如下：

**allowOrigins:**允许访问资源的origin列表。必须是完整的域名，包含scheme, host和port(如果有的话),以,分隔，且区分大小写。 如：[http://a.b.com](http://a.b.com/), [http://a.b.com:8080](http://a.b.com:8080/)。作用于preflight请求以及后面的正式访问请求。访问的请求必须带有Origin header，并且内容要完全匹配列表中的某个值才允许访问后面的资源。如果对origin无限制，则该项可以配置成\*。

**allowMethods：**允许访问资源的method列表，以,分隔，区分大小写。如：GET, POST等。在preflight请求中，必须带有“Access-Control-Request-Method” header，并且值必须与列表中的某个值匹配，才能允许访问后续资源。如果对Method无限制，可以将该项配置成\*。

**allowHeaders：**访问资源的请求所允许携带的header列表。Filter会检查与preflight请求的“Access-Control-Request-Headers” header，获取请求访问的header列表。Header列表所包含的header必须全部出现在allowHeaders列表中（不区分大小写），才允许访问后续资源。如果对header无限制，可将该项配置为\*。

**supportCredentials：**是否支持credentials选项，可选值为：true或者false。如果为true，则在响应中加入“Access-Control-Allow-Credentials”为true的header。

**maxAge：**指定preflight request缓存的时间长度，单位为秒。配置后，会在响应中加入相应的“Access-Control-Max-Age” header和秒数。

**exposeHeaders：**允许expose的header列表，以,分隔。配置该项后，在正式访问请求的响应中会带上“Access-Control-Expose-Header” header。

**checkReferer：**是否检查referer header，可选值：true或false。该选项主要用于处理IE某些版本下，不支持CORS并且跨域请求不带Origin header的情况。打开该开关后，如果请求没有Origin header，则会比较Referer和Host来判断请求是否存在跨域。如果有跨域，则从Referer中提取出Origin域名，之后的流程则与前面描述的流程一致。

#### 2.2.2 配置url mapping

将需要做CORS处理的资源与filter进行关联即可。

<filter-mapping>

<filter-name>cross-origin</filter-name>

<url-pattern>/path/to/resources</url-pattern>

</filter-mapping>

## 3.使用示例

<filter>

<filter-name>cross-origin</filter-name>

<filter-class>com.netease.wd.crossorigin.filter.CrossOriginFilter</filter-class>

<init-param>

<param-name>allowOrigins</param-name>

<param-value>http://a.b.com, http://a.b.com:8080</param-value>

</init-param>

<init-param>

<param-name>allowMethods</param-name>

<param-value>GET, POST, OPTIONS</param-value>

</init-param>

<init-param>

<param-name>allowHeaders</param-name>

<param-value>\*</param-value>

</init-param>

<init-param>

<param-name>supportCredentials</param-name>

<param-value>true</param-value>

</init-param>

<init-param>

<param-name>maxAge</param-name>

<param-value>60</param-value>

</init-param>

<init-param>

<param-name>exposeHeaders</param-name>

<param-value>Set-Cookie, Max-Age</param-value>

</init-param>

<init-param>

<param-name>checkReferer</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>cross-origin</filter-name>

<url-pattern>/path/to/resources</url-pattern>

</filter-mapping>

## 4.支持nej兼容浏览器

为了配合nej框架兼容不同的浏览器，需要对响应进行不同的加工处理。

对于支持CORS的浏览器，服务器只需要返回json字符串即可；

对于不支持CORS的浏览器，服务器需要返回特定格式的html页面内容。

Cross-origin包提供了简便的工具类对上述流程做了封装。应用程序只需调用工具方法并传递响应的json字符串即可。

**使用示例如下**：

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response) {

String content = "{msg: \"'hello'\"}";

**try** {

CORSResponseUtil.*response*(request, response, content);

} **catch** (IOException e) {

e.printStackTrace();

}

}