# url解析：

http://mail.163.com/index.html

1)http://:这个是协议，也就是HTTP超文本传输协议，也就是网页在网上传输的协议。

2）mail：这个是服务器名，代表着是一个邮箱服务器，所以是mail.

3)163.com:这个是域名，是用来定位网站的独一无二的名字。

4)mail.163.com：这个是网站名，由服务器名+域名组成。

5）/：这个是根目录，也就是说，通过网站名找到服务器，然后在服务器存放网页的根目录

6:）index.html：这个是根目录下的默认网页（当然，163的默认网页是不是这个我不知道，只是大部分的默认网页，都是index.html）

7）http://mail.163.com/index.html:这个叫做URL，统一资源定位符，全球性地址，用于定位网上的资源。

# 域名等级：

域名是由一串点分隔的字符组成的internet上某一计算机的名字

顶级域名就是一级域名，比如 .com .org .cn  
N级域名就是在N-1级域名前追加一级。  
比如二级域名是在一级域名前加一级，二级域名示例：[baidu.com](https://link.zhihu.com/?target=http%3A//baidu.com) [zhihu.com](http://zhihu.com) [qq.com](https://link.zhihu.com/?target=http%3A//qq.com)

注： 有些人会说 [baidu.com](https://link.zhihu.com/?target=http%3A//baidu.com) [zhihu.com](http://zhihu.com) [qq.com](https://link.zhihu.com/?target=http%3A//qq.com) 是一级域名，虽然是错误的-但可以理解(说的人多了也就是对的了……)，这是站在使用者/购买者角度看的，对于购买域名者来说 [xxx.com](https://link.zhihu.com/?target=http%3A//xxx.com) [xxx.com.cn](https://link.zhihu.com/?target=http%3A//xxx.com.cn)就相当一级域名，但是从真正的域名分级看，它们俩分别是二级域名、三级域名。

[www.baidu.com](http://www.baidu.com) 是三级域名

计算机host文件：在访问请求是先到本地hosts中根据域名去映射对应的ip地址，然后去请该地址。

# 浏览器同源策略

同源：URL由协议、域名、端口和路径组成，如果两个URL的协议、域名和端口相同，则表示他们同源。

同源策略:

浏览器的同源策略，限制了来自不同源的"document"或脚本，对当前"document"读取或设置某些属性。 （白帽子讲web安全[1]）从一个域上加载的脚本不允许访问另外一个域的文档属性。即使用js脚本不能获取另外域的内容。

在浏览器中，<script>、<img>、<iframe>、<link>等标签都可以加载跨域资源，而不受同源限制，但浏览器限制了JavaScript的权限使其不能读、写加载的内容。

例子：

1. http://localhost:8080/test.html
2. **<html>**
3. **<head><title>**test same origin policy**</title></head>**
4. **<body>**
5. **<iframe** id="test" src="http://localhost:8081/test2.html"**></iframe>**
6. **<script** type="text/javascript"**>**
7. document.getElementById("test").contentDocument.body.innerHTML = "write somthing";
8. **</script>**
9. **</body>**
10. **</html>**
12. http://localhost:8081/test2.html
13. **<html>**
14. **<head><title>**test same origin policy**</title></head>**
15. **<body>**
16. Testing.
17. **</body>**
18. **</html>**

站点 http://domain-a.com 的某 HTML 页面通过 [<img> 的 src](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/Img#Attributes) 请求 http://domain-b.com/image.jpg。网络上的许多页面都会加载来自不同域的CSS样式表，图像和脚本等资源。出于安全考虑，浏览器会限制从脚本内发起的跨域HTTP请求。（跨域并~~非~~不一定是浏览器限制了发起跨站请求，~~而~~也可能是跨站请求可以正常发起，但是返回结果被浏览器拦截了。最好的例子是 CSRF 跨站攻击原理，请求是发送到了后端服务器无论是否跨域！注意：有些浏览器不允许从 HTTPS 的域跨域访问 HTTP，比如  Chrome 和 Firefox，这些浏览器在请求还未发出的时候就会拦截请求，这是一个特例。）

补充：

Access-Control-Allow-Origin: \* 表明，该资源可以被**任意**外域访问。如果服务端仅允许来自 http://foo.example 的访问，该首部字段的内容如下： Access-Control-Allow-Origin: http://foo.example

# 跨域问题：

跨域：只有当协议，域名，端口相同的时候才算是同一个域名，否则均认为需要做跨域的处理。

**后台cookie跨域：（Tomcat服务器）**

Cookie跨二级以上域：

Cookie中的同源只关注域名，忽略协议和端口。

Domain指令规定了需要发送Cookie的主机名。如果没有指定，默认为当前的文档地址上的主机名（但是不包含子域名）。如果指定了Domain，则一般包含子域名。如果设置了Domain=mozilla.org，则Cookie包含在子域名中（如developer.mozilla.org）。

Path指令表明需要发送Cookie的URL路径。字符%x2F (即"/")用做文件夹分隔符，子文件夹也会被匹配到。如设置Path=/docs，则下面这些地址都将匹配到:

* "/docs",
* "/docs/Web/",
* "/docs/Web/HTTP"

每个Cookie都有与之关联的域（Domain），如果Cookie的域和页面的域是一样的，那么我们称这个Cookie为第一方Cookie（first-party cookie），如果Cookie的域和页面的域不一样，则称之为第三方Cookie（third-party cookie.）。

HTTP-only类型的Cookie不能使用Javascript通过[Document.cookie](https://developer.mozilla.org/zh-CN/docs/Web/API/Document/cookie)属性来访问，从而能够在一定程度上阻止跨域脚本攻击（[XSS](https://developer.mozilla.org/en-US/docs/Glossary/XSS)）。响应头Set-cookie上。

例如：

172.16.100.19 a.localhost.com

172.16.100.19 b.localhost.com

172.16.100.19 localhost.com

程序中：

Cookie cookie = **new** Cookie("sun", "test");

cookie.setDomain(".localhost.com");

cookie.setPath("/");

response.addCookie(cookie);

a.localhost.com

b.localhost.com

localhost.com

c.b.localhost.com

三个连接均能获取到cookie（即二级域名及以下所有域名均能获取到）

程序中：

当我们在进行跨域访问的时候，我们的cookie就不会被保存下来

对应a.localhost.com或b.localhost.com或localhost.com中任何一个的web应用创建cookie时指定了其他的二级域名，则浏览器不会保存该cookie。

例如：cookie.setDomain("localhost1.com");

a.localhost.com或b.localhost.com或localhost.com中创建cookie浏览器不保存。

***后台session跨域：session跨域在父子级域名可通过***

存储在cookie中的sessionid使用默认的domain和path，并且不能更改。

以下几种解决方案：

url重写：在url后面加；jsessionid=

jsonp跨域

cors跨域

将session存储在管理中间件中，例如Redis中，多个域均在此存取session。

**Tomcat中同一个容器不同应用间，**使用servletcontext来共享session：**（特例）**

需要配置容器（对每个context配置路径）:

<Host appBase="webapps" autoDeploy="true" name="localhost" unpackWARs="true" xmlNamespaceAware="false" xmlValidation="false">

　　　　 <Context path="/项目A" reloadable="false" crossContext="true"></Context>

　　　　 <Context path="/项目B" reloadable="false" crossContext="true"></Context>

　 </Host>

不同应用中获取另外应用的servletcontext使用当前应用的servletcontext对象的getContext(uripath)方法。

**url重写：？**

**缺点：url变化了，导致一些服务无法访问**

**要求页面均为动态页面**

**脚本跨域（受浏览器同源策略限制）**

**Jsonp跨域**

**Cors跨域**

Cors跨域：<http://www.ruanyifeng.com/blog/2016/04/cors.html>

https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access\_control\_CORS

主要解决浏览器同源策略，ajax无法跨域请求，ajax跨域传输数据（返回响应setcookie和请求时携带cookie，）