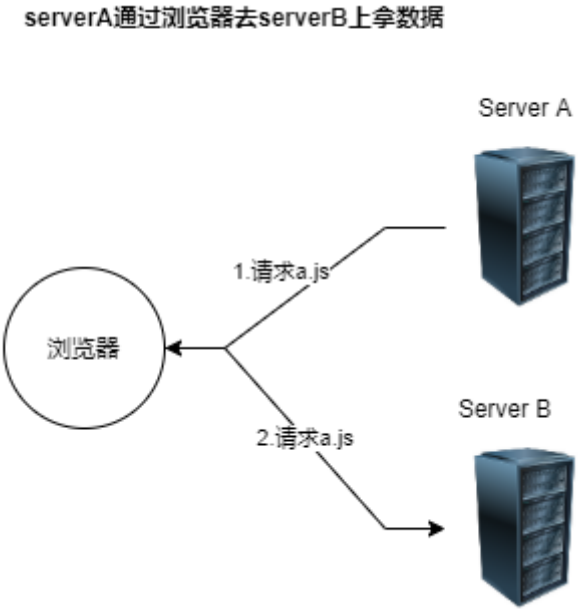
什么是跨域问题

> 跨域问题的出现是因为浏览器的同源策略问题，所谓同源:就是两个页面具有相同的协议（protocol），主机（host）和端口号（port），即指协议，端口，域名。只要这个3个中有一个不同就是跨域。它是浏览器最核心也是最基本的功能，如果没有同源策略我们的浏览器将会十分的不安全，随时都可能受到攻击。
>
> 当我们请求一个接口的时候，出现如：Access-Control-Allow-Origin 字眼的时候说明请求跨域了



展示跨域问题

前端请求后端8889端口开放的端口

```
this.$axios({
    method:'get',
    url:'http://localhost:8889/'
}).then((response) =>{          //这里使用了ES6的语法
    console.log(response)       //请求成功返回的数据
}).catch((error) =>{
    console.log(error)       //请求失败返回的数据
})
```

会出现因为跨域，所以请求失败的消息

```
Access to XMLHttpRequest at 'http://localhost:8889/' from origin
'http://localhost:8080' has been blocked by CORS policy: No 'Access-Control-
Allow-Origin' header is present on the requested resource.
```

处理跨域

前端

jsonp

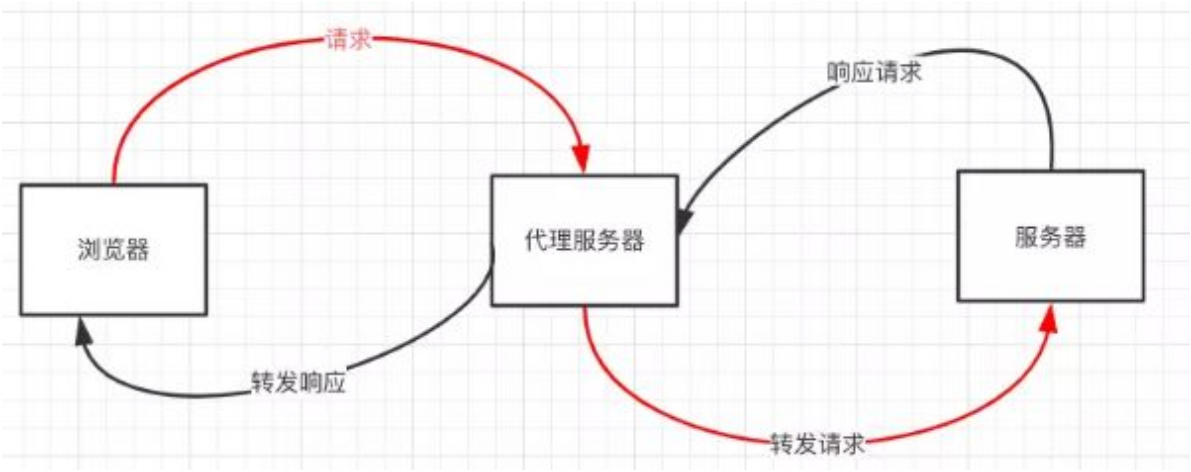
```
// 后端接口
app.get("/", function(req, res){
    var func = req.query.callback        // 拿到请求参数query中的callback数据
    res.send(func + "('你好')")           // 拼接成函数调用的形式==》 f("你好")
})
// 前端请求
<script>
    function f(msg){
        alter(msg)
    }
</script>
<script src="http://localhost:8999?callback=f"></script>     // 请求这个接口的时候会得到f("你好")的返回结果，从而触发调用函数
```

Proxy

> 通过网络代理，通过代理服务与另一个网络终端进行非直接的连接

通过webpack搭建一个本地服务器，作为请求的代理对象，通过该服务器转发请求至目标服务器。

但存在的问题是，如果webpack服务器与web接口服务器不在一起，仍然存在跨域问题。===>由于生产环境基本上后端有后端部署的服务器、前端有前端部署的服务器，因此这个无法在生产上使用



Vue

- 开发阶段: 配置代理
  - 如果你是用 Vue CLI 创建的项目，也可以开发阶段，配置一下 webpack 的 devServer，它有自带一个 proxy 代理服务器。
  - ```
    // Vue3中是在vue.config.js配置
    module.exports = {
        devServer:{
            host: 'localhost',
            open: true,                 // vue项目启动时自动打开浏览器
            port: 8080,                 // 本地服务器使用的端口
            proxy: {                    //配置跨域
              '/api': {                 // `/api`是代理标识，用于告诉node，url前面是`/api`的就是使用代理的
    ```
    ```

```
            target: 'http://127.0.0.1:5000/',  // 目标地址，应该写提供接口的后台服务器的真实地址
            changOrigin: true,      // 是否跨域
            pathRewrite: {
              '^/api': ''           // 把实际request URL中的`/api`用``来代替
              /* 重写路径，当我们在浏览器中看到请求的地址为：http://localhost:8080/api/core/getData/userInfo 时，实际上访问的地址是：http://121.121.67.254:8185/core/getData/userInfo */
            }
          },
        }
    }
}
```

- 后端配置

```
from flask import Flask
from flask_cors import cross_origin

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

@app.route('/api')
def apix():
    return 'Hello api!'

@app.route('/do')
@cross_origin()
def api():
    return 'Hello do!'

if __name__ == '__main__':
    app.run()
```

- 访问`http://localhost:8080/`时会出现跨域问题，`http://localhost:8080/api`时浏览器直接跳转到后端内容
- 网络请求`http://localhost:8080/do`由于do接口允许跨域所以成功拿到数据，`/api`成功拿到Hello World实现跨域
  - 为什么get("/api")拿到的是`http://localhost:8080/`的数据呢？首先拼接成`http://localhost:8080/`，然后符合代理规则，所以根据pathRewrite吧`/api`替换成了`""`，于是请求变成了访问`http://localhost:8080/`。再因为vue.config.vue中target指定了axios的目标url，会把`http://localhost:8080`代理掉，所以就代理转向访问了`http://localhost:5000/`
    注： 没有指定axios的baseURL则为本身，会自动加上http://localhost
  - 可以测试是不是这个逻辑，axios.get("/do"), 提示`http://localhost:8080/do`不存在，也不符合代理规则
    - axios.get("/"), 虽然也不符合代理规则，但其本身`http://localhost:8080/`是有的，所以也能请求到数据，只不过这个是index.html的数据

    - 如果指定axios的baseURL="http://baidu.com"，则xxx.get("/api")会变成 `http://baidu.com/api`
- 生产环境: 配置nginx转发

后端

nginx

```
# 如下的是青龙的转发配置
upstream api {
  server 0.0.0.0:5600;
}

map $http_upgrade $connection_upgrade {
  default keep-alive;
  'websocket' upgrade;
}

server {
  listen 5700;
  root /ql/dist;
  ssl_session_timeout 5m;

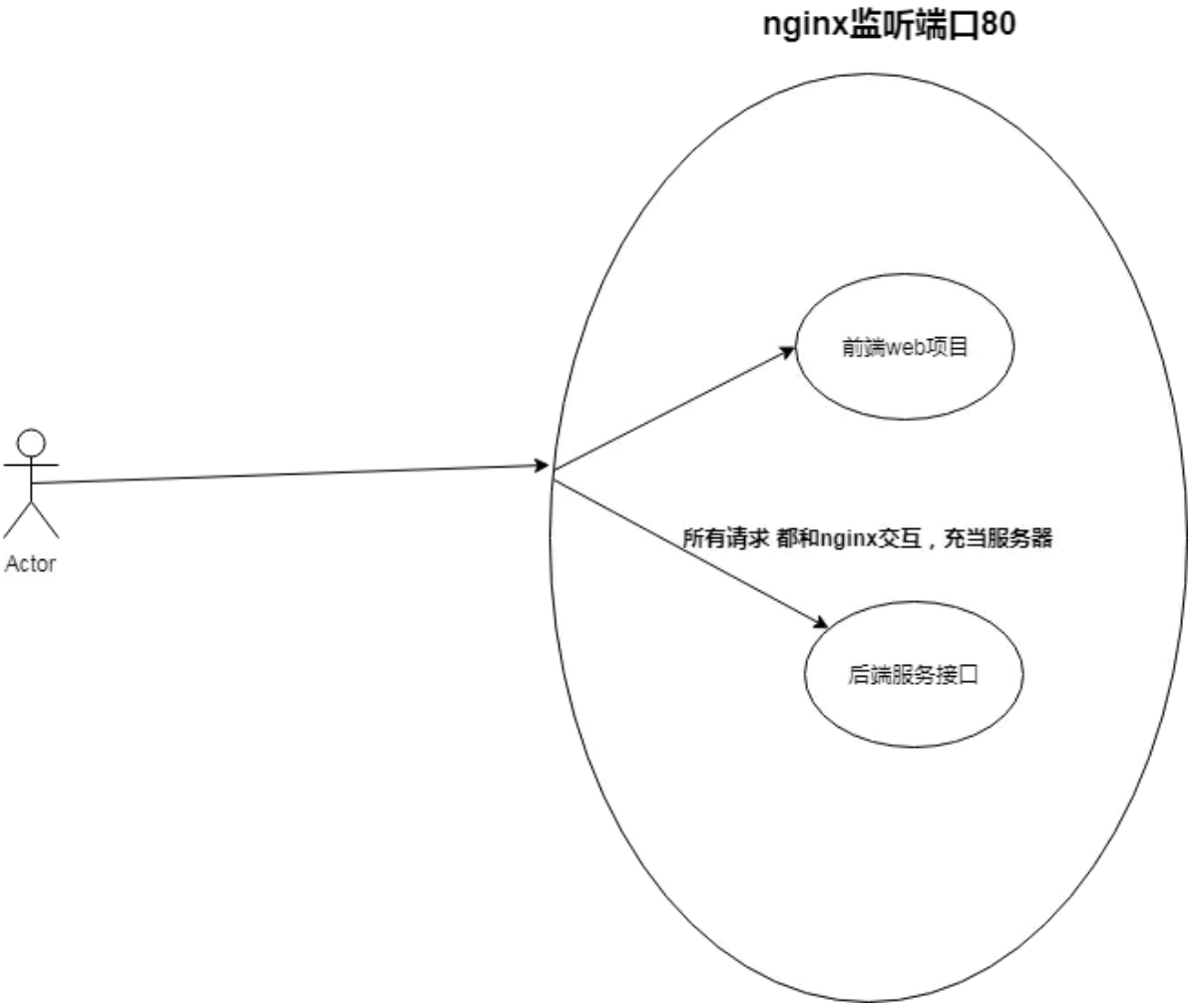
  location /api {
    proxy_set_header Host $http_host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_pass http://api;

    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection $connection_upgrade;
  }

  location /open {
    proxy_set_header Host $http_host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_pass http://api;
  }

  gzip on;
  gzip_static on;
  gzip_types text/plain application/json application/javascript application/x-
javascript text/css application/xml text/javascript;
  gzip_proxied  any;
  gzip_vary on;
  gzip_comp_level 6;
  gzip_buffers 16 8k;
  gzip_http_version 1.0;

  location / {
    index  index.html index.htm;
    try_files $uri $uri/ /index.html;
  }
}
```



修改响应头

> 根据上述CORS反馈的消息`No 'Access-Control-Allow-Origin' header is present on the requested resource.`来看，是由于没有设置响应头`Access-Control-Allow-Origin`，因此其中一个做法是可以修改响应头中的Access-Control-Allow-Origin

```
app.get("/", function(req, res){
    res.header("Access-Control-Allow-Origin", "*")  // 允许任何域的请求
    res.send("你好")
})
```

还在相应的接口上添加`@CrossOrigin`表示允许跨域请求

> 可以看做是修改响应头的变种

```java
@RestController
public class InfoController {
    @CrossOrigin
    @GetMapping("/")
    public HashMap<Object, Object> getInfo() {
        HashMap<Object, Object> hashMap = new HashMap<>();
        hashMap.put("cl", 19);
        return hashMap;
    }
}
```

附录

- B站视频：【跨域请求】【前端】什么是CORS，教你解决跨域问题

采坑说明：

- CORS是浏览器的限制，而HbuilderX中内置的浏览器是没有跨域问题的，需要打开外部浏览器才能看到。