# 跨域问题

## 伪静态实现

### 业务需求

如何让网站更加容易被搜索引擎收录.并且百度时用户可以看到.

### 伪静态概念

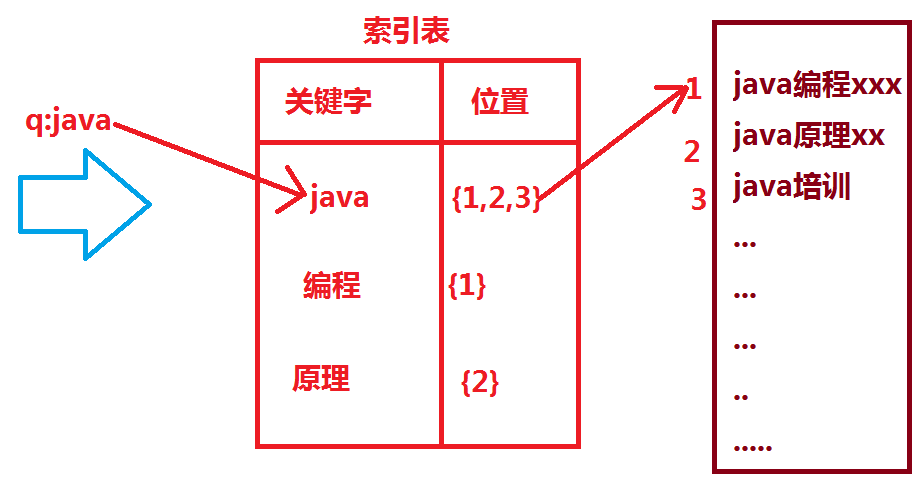
伪静态是相对真实静态来讲的，通常我们**为了增强**[**搜索引擎**](https://baike.baidu.com/item/%E6%90%9C%E7%B4%A2%E5%BC%95%E6%93%8E/104812)**的友好面**，都将文章内容生成静态页面，但是有的朋友为了实时的显示一些信息。或者还想运用动态脚本解决一些问题。不能用静态的方式来展示网站内容。但是这就损失了对搜索引擎的友好面。怎么样在两者之间找个中间方法呢，这就产生了伪静态技术。就是展示出来的是以html一类的静态页面形式，但其实是用ASP一类的动态脚本来处理的。

总结:以html形式展现动态页面效果.

前提:搜索引擎(百度/360)一般只会收录HTML页面信息.

搜索引擎工作原理:

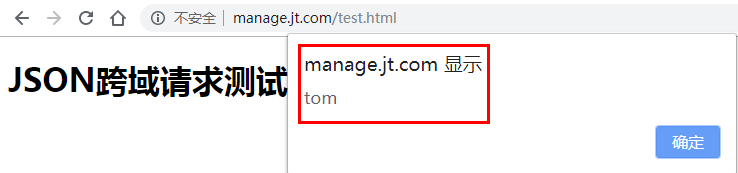
倒排索引规则:根据关键字检索文章位置.



## 跨越访问测试

### Jt-manage测试

说明:在jt-manage中访问test.html测试访问效果.发现数据可以正常访问.



<script type="text/javascript">

$(**function**(){

$.get("http://manage.jt.com/test.json",**function**(data){

alert(data.name);

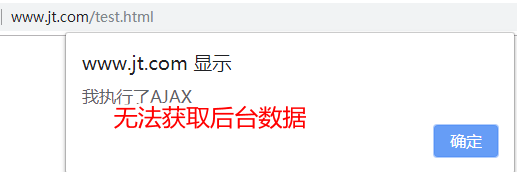
})

})

</script>

### Jt-web测试

说明:在浏览器中执行[www.jt.com/test.html](http://www.jt.com/test.html)中发现请求不能正常执行.



这是为什么呢??

### 关于访问问题说明

当通过不同的url访问相同的html时,发现效果是不同的.

域名与url地址相同时可以正常访问!

域名与url地址不同时 访问的状态码200表示成功.但是实际数据无法获取!!!

### 同源策略

规定:如果 **协议://域名:端口**都相同时,为**同域访问**.浏览器可以正常的解析数据.

如果上述的三项有一项不同,则表示**跨域访问**.浏览器可以正确的请求数据.但是不予解析返回值!!!



<http://manage.jt.com>

<http://localhost:8091> 跨域请求

## 跨域实现

### 跨域原理

1. 利用javascript里边的**src**属性的开放原则实现跨域.
2. 定义回调函数.
3. 封装返回值数据 回调函数名称(json数据).

#### 编辑页面js

1. 利用src的开放策略 进行跨域访问
2. 自定义回调函数 hello

<script type="text/javascript">

/\*JS是解释执行的语言 \*/

/\*2定义回调函数 \*/

**function** hello(data){

alert(data.name);

}

</script>

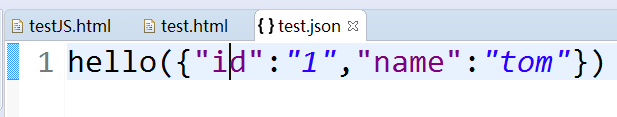
<!--该json一直保存到浏览器中等待调用,但是没有函数名称无法调用 -->

<script type="text/javascript" src="http://manage.jt.com/test.json"></script>

<script type="text/javascript" src="http://manage.jt.com/js/jquery-easyui-1.4.1/jquery.min.js"></script>

#### 编辑返回值数据

3.将返回值进行特殊格式封装.



### JSONP介绍

JSONP(JSON with Padding)是[JSON](https://baike.baidu.com/item/JSON)的一种“使用模式”，可用于解决主流浏览器的**跨域数据访问的问题**。由于同源策略，一般来说位于 server1.example.com 的网页无法与不是 server1.example.com的服务器沟通，而 HTML 的<script> 元素是一个例外。利用 <script> 元素的这个**开放策略**，网页可以得到从其他来源动态产生的 JSON 资料，而这种使用模式就是所谓的 JSONP。用 JSONP 抓到的资料并不是 JSON，而是任意的JavaScript，用 JavaScript 直译器执行而不是用 JSON 解析器解析。

### jQuery-JSONP跨域

$.ajax({

url:"http://manage.jt.com/web/testJSONP",

type:"get", //jsonp只能支持get请求

dataType:"jsonp", //dataType表示返回值类型

//jsonp: "callback", //指定参数名称

//jsonpCallback: "hello", //指定回调函数名称

success:**function** (data){ //data经过jQuery封装返回就是json串

alert(data.id);

alert(data.name);

//转化为字符串使用

//var obj = eval("("+data+")");

//alert(obj.name);

}

});

原理说明:

1. url跨域地址已经人为指定了
2. 回调函数自己没有定义. jQuery底层实现时自定生成了回调函数
3. 将自动生成的函数的名称 通过默认的属性 ?callback=函数名称()进行传递.
4. 后台服务器 获取函数名称之后 拼接json数据 callback(json)

JSONP跨域请求过程:

[**http://manage.jt.com/web/testJSONP**?**callback**=**jQuery111109956610654936453\_1560396296777&\_=1560396296778**](http://manage.jt.com/web/testJSONP?callback=jQuery111109956610654936453_1560396296777&_=1560396296778)**(回调函数名称)**

### jQuery-JSONP后台实现(一)

//在后台编辑方法

@RestController

**public** **class** JSONPController {

//返回值要求: 回调函数(json数据);

@RequestMapping("/web/testJSONP")

**public** String testJSONP(String callback) {

User user = **new** User();

user.setId(100);

user.setName("tomcat");

String json = ObjectMapperUtil.*toJSON*(user);

**return** callback +"("+json+")";

}

}

### jQuery-JSONP后台实现(二)

@RequestMapping("/web/testJSONP")

**public** JSONPObject jsonp(String callback) {

User user = **new** User();

user.setId(100);

user.setName("tomcat");

JSONPObject object =

**new** JSONPObject(callback, user);

**return** object;

}

### JSONP与HttpClient区别

请求方式不同:

HttpClient:请求方式多元化.

JSONP:请求方式 浏览器解析js发起AJAX

接收端不同:

HttpClient:自身Controller接收用户请求之后完成处理

JSONP: 直接进行跨域访问.访问远程服务器.

返回值类型不同;

HttpClient: 可以直接返回JSON数据/xml(报文)

JSONP: 返回时必须经过特殊的格式封装

callback(JSON/String)

目的不同:

HttpClient: 在任何位置都可以发起请求 (业务需要)

JSONP: 只能在浏览器端 发起AJAX请求 (一般查询操作,安全性问题)

调用层级不同:

HttpClient: 调用层级6层

JSONP: 调用4层

# JT-SSO单点登录系统实现

## 跳转登录/注册页面

### 业务分析

说明:当用户点击登录/注册按钮时分别跳转不同的页面

<http://www.jt.com/user/register.html>

<http://www.jt.com/user/login.html>

利用通用跳转技术.

### 编辑Controller

@Controller

@RequestMapping("/user")

**public** **class** UserController {

@RequestMapping("/{moduleName}")

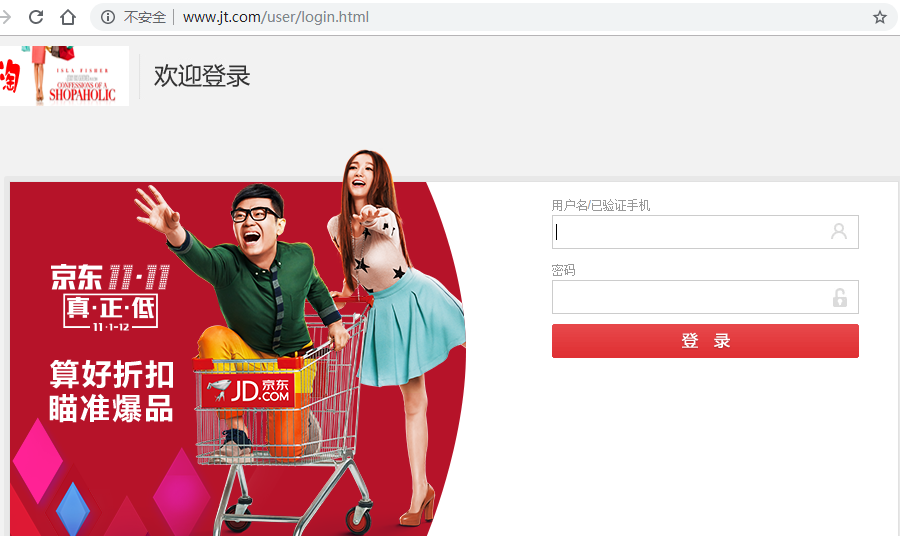
**public** String index(@PathVariable String moduleName) {

**return** moduleName;

}

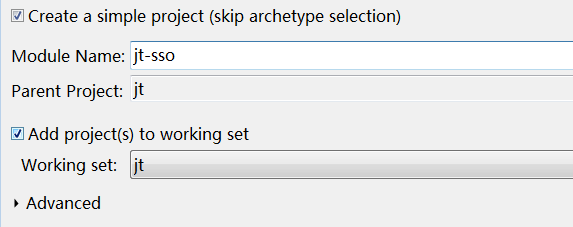
}

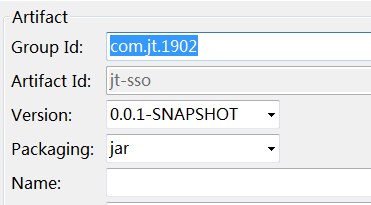
### 页面效果



## 构建单点登录项目

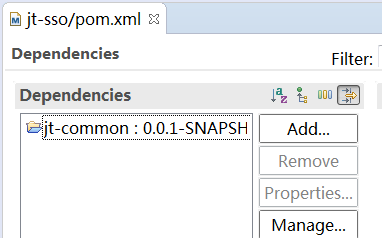
### 创建项目





### 编辑POM.xml文件

#### 添加common依赖



#### 添加maven插件

<build>

<plugins>

<!--跳过测试类打包 -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<configuration>

<skip>true</skip>

</configuration>

</plugin>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

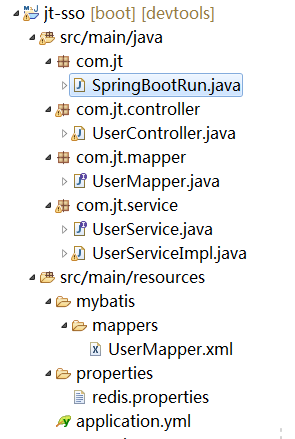
</plugin>

</plugins>

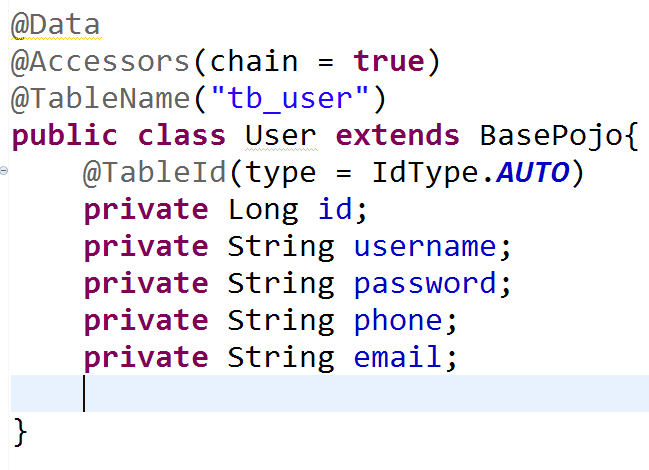
</build>

### 导入src文件

说明:将课前资料中的jt-sso的src文件导入项目.结构如下



### 编辑User POJO

* 

在jt-common中添加userpojo对象.之后编辑UserMapper

**public** **interface** UserMapper **extends** BaseMapper<User>{

}

### 编辑nginx

#单点登录管理系统 hosts

server {

listen 80;

server\_name sso.jt.com;

location / {

#代理路径

proxy\_pass http://127.0.0.1:8093;

}

}

编辑完成后,重启nginx

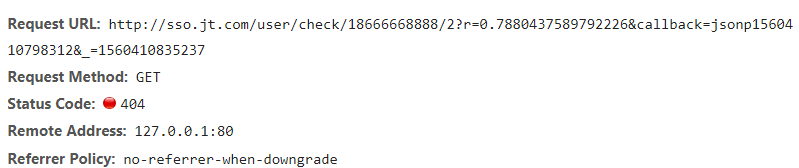
## 用户信息校验实现

### 业务接口文档

|  |  |
| --- | --- |
| 请求方法 | GET |
| URL | http://sso.jt.com/user/check/{param}/{type} |
| 参数 | 格式如：chenchen/1  其中chenchen是校验的数据  Type为类型，可选参数1 username、2 phone、3 email |
| 示例 | http://sso.jt.com/user/check/chenchen/1 |
| 返回值 | {  status: 200 //200 成功，201 没有查到  msg: “OK” //返回信息消息  data: false //返回数据true用户已存在，false用户不存在，可以  } |
|  |  |

### 页面分析

#### 页面URL分析



#### 页面JS分析

##### 根据js请求路径查找js代码

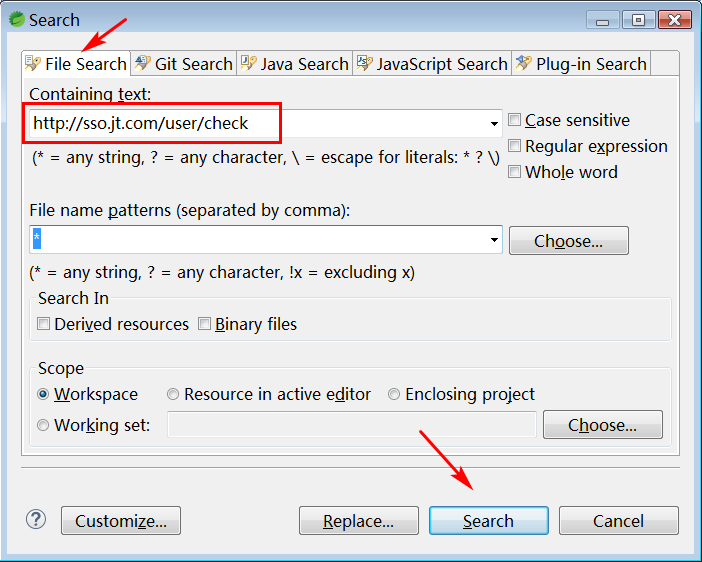
说明:由于大项目开发时有很多人参与.很多的js代码都不是自己完成的.但是为了能够更加了解业务执行过程.需要读懂别人的js.这时需要快速定位js在项目中的位置.

查询规则:

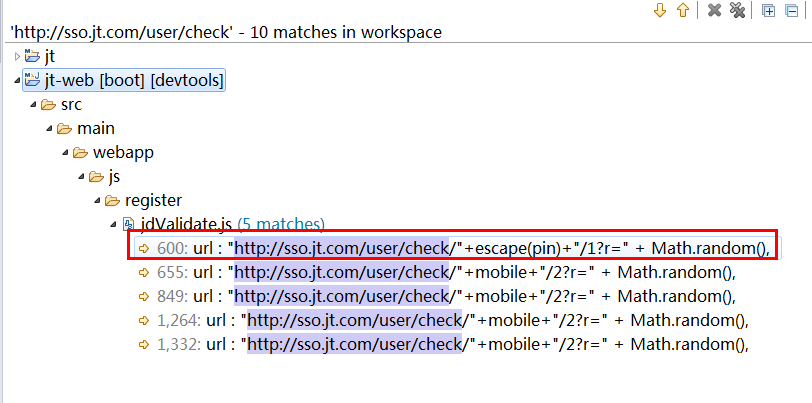
1. 确定js代码中必然写死的代码

<http://sso.jt.com/user/check>

1. 利用搜索工具快速定位. Crtl+h



查找结果:



#### 页面JS分析

$.ajax({

url : "http://sso.jt.com/user/check/"+escape(pin)+"/1?r=" + Math.random(),

dataType : "jsonp",

success : **function**(data) {

checkpin = data.data?"1":"0";

**if** (!data.data) {

validateSettings.succeed.run(option);

namestate = **true**;

}**else** {

validateSettings.error.run(option, "该用户名已占用！");

namestate = **false**;

}

}

});

### 编辑后台Controller

@RestController

@RequestMapping("/user")

**public** **class** UserController {

@Autowired

**private** UserService userService;

/\*\*

\* 业务说明:校验用户是否存在

\*http://sso.jt.com/user/check/{param}/{type}

\* 返回值:SysResult

\* 由于跨域请求 所以返回值必须特殊处理callback(json)

\*/

@RequestMapping("/check/{param}/{type}")

**public** JSONPObject checkUser(@PathVariable String param,

@PathVariable Integer type,

String callback) {

JSONPObject object = **null**;

**try** {

**boolean** flag = userService.checkUser(param,type);

object = **new** JSONPObject(callback, SysResult.*ok*(flag));

} **catch** (Exception e) {

e.printStackTrace();

object = **new** JSONPObject(callback,SysResult.*fail*());

}

**return** object;

}

}

### 编辑Service

@Service

**public** **class** UserServiceImpl **implements** UserService {

@Autowired

**private** UserMapper userMapper;

/\*\*

\* true 表示用户已经存在 false表示用户可以使用

\* 1.param 用户参数

\* 2.type 1 username/2 phone/3 email

\* 将 type转化为具体字段

\*/

@Override

**public** **boolean** checkUser(String param, Integer type) {

String column =

type==1?"username":(type==2?"phone":"email");

QueryWrapper<User> queryWrapper = **new** QueryWrapper<>();

queryWrapper.eq(column, param);

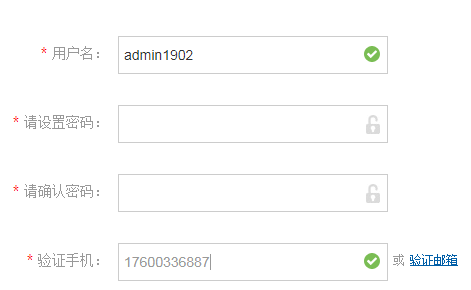
**int** count = userMapper.selectCount(queryWrapper);

**return** count==0?**false**:**true**;

}

}

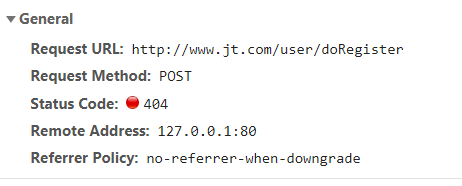
### 页面效果

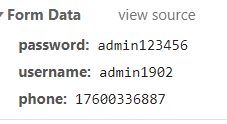


### 作业

作业说明:完成用户注册 使用httpClient技术

1. url: <http://www.jt.com/user/doRegister>
2. 请求方式post请求





1. 编辑前台UserController
2. 拦截用户请求
3. 返回值结果SysResult ok/faile

$.ajax({

type : "POST",

url : "/user/doRegister",

contentType : "application/x-www-form-urlencoded; charset=utf-8",

data : {password:\_password,username:\_username,phone:\_phone},

dataType : 'json',

success : **function**(result) {

**if**(result.status == "200"){

// 注册成功，去登录页

showMessage('注册成功，请登录！');

verc();

$("#registsubmit").removeAttr("disabled").removeClass()

.addClass("btn-img btn-regist");

isSubmit = **false**;

**return**;

}**else**{

showMessage('注册失败,请联系管理员!');

//alert('注册失败，请重新注册！ ' + result.data );

}

4.编辑前台Service

httpClient:

请求路径<http://sso.jt.com/user/register>

请求方式post

要求传输用户提交的数据 username/**password**/phone

不要求判断sso的返回值数据

1. 编辑SSO的Controller

接收jt-web传递的数据 username/password/phone

1. 实现数据入库

要求:使用电话号码暂时代替email

User.setEmail(phone);

### 作业代码实现

#### 编辑前台Controller

/\*

\* 作业代码

\*

\*/

@RequestMapping("/doRegister")

@ResponseBody

**public** SysResult saveUser(User user) {

**try** {

userService.saveUser(user);

**return** SysResult.*ok*();

} **catch** (Exception e) {

e.printStackTrace();

**return** SysResult.*fail*();

}

}

#### 编辑前台Service

@Service

**public** **class** UserServiceImpl **implements** UserService {

@Autowired

**private** HttpClientService httpClient;

@Override

**public** **void** saveUser(User user) {

String url = "http://sso.jt.com/user/register";

//将密码加密

String md5Pass = DigestUtils.*md5DigestAsHex*(user.getPassword().getBytes());

user.setPassword(md5Pass);

String userJSON = ObjectMapperUtil.*toJSON*(user);

Map<String,String> params = **new** HashMap<>();

//将数据封装到map集合中参数key=userJSON

params.put("userJSON", userJSON);

String result = httpClient.doPost(url, params);

//判断返回值是否正确

SysResult sysResult = ObjectMapperUtil.*toObject*(result, SysResult.**class**);

**if**(sysResult.getStatus() == 201) {

//说明后台程序运行出错.业务停止

**throw** **new** RuntimeException();

}

}

}

#### 编辑后台Controller

编辑jt-sso项目的UserController

//http://sso.jt.com/user/register

@RequestMapping("/register")

**public** SysResult saveUser(String userJSON) {

**try** {

User user = ObjectMapperUtil.*toObject*(userJSON, User.**class**);

userService.saveUser(user);

**return** SysResult.*ok*();

} **catch** (Exception e) {

e.printStackTrace();

**return** SysResult.*fail*();

}

}

#### 编辑后台Service

编辑jt-sso的userService

@Transactional //事务控制

@Override

**public** **void** saveUser(User user) {

user.setEmail(user.getPhone())

.setCreated(**new** Date())

.setUpdated(user.getCreated());

userMapper.insert(user);

//int a = 1/0; 测试当用户入库失败后用户展现情况

}