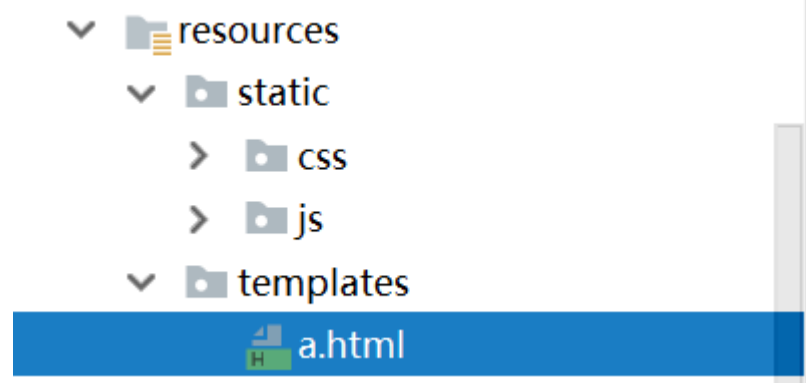


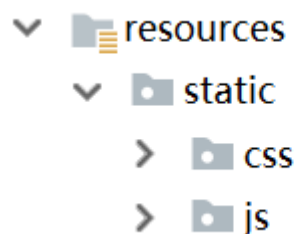
我们在 springboot 中也可以不使用模板引擎，直接创建 html 文件进行访问，但是需要进行相关视图设置

```
spring:
  mvc:
    view:
      # 视图文件前缀配置
      prefix: classpath:/templates/
      # 视图文件后缀
      suffix: .html
      # 使用的静态资源文件的位置
  resources:
    static-locations: classpath:/static/
```

配置后 在对应的文件夹下创建对应的 html 文件



html 可能用到的静态资源文件在 static 文件夹下



Controller 控制器中对应的信息

```

@Controller
public class BTest {
    @RequestMapping("/a")
    public String a(){
        return "a.html";
    }
}

```

在 a.html 文件中 与我们之前写的 html 相同 静态文件有默认的存放路径

```

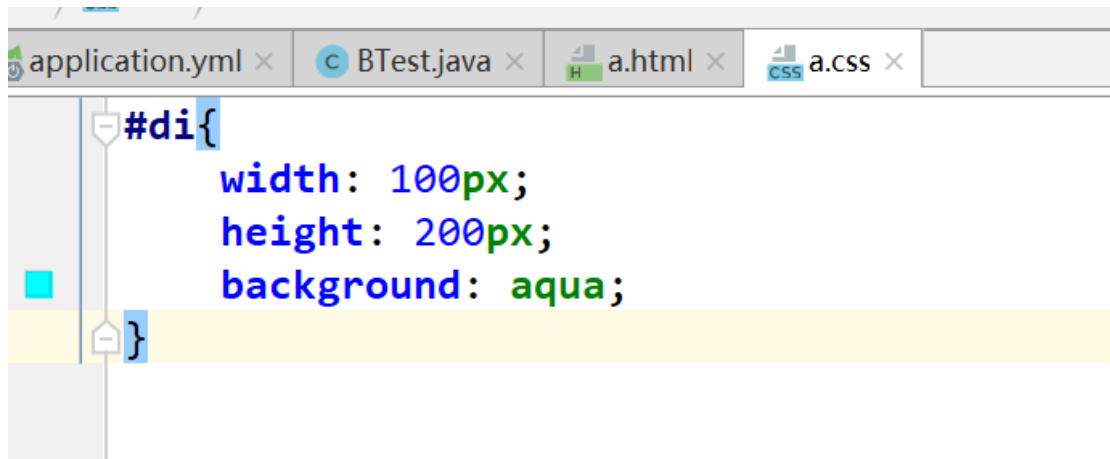
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link href="css/a.css" type="text/css" rel="stylesheet">
</head>
<body>
    <div id="di">a page</div>
    <script src="js/jquery-3.4.1.min.js"></script>
    <script>
        $(function () {
            alert(1)
        })
    </script>
</body>
</html>

```

访问后



js 中的弹窗正常执行 说明 jquery 正确引入



css 中的样式也被正常引入

因此 我们可以在 html 中通过 ajax 请求对应后台的数据信息，并在页面中通过 jquery 进行数据渲染

（ajax 获取数据与数据渲染 不在赘述）

现在我们换一种在后台获取数据的方式 axios 使用方式比 ajax 更简单

Axios 特征

- 支持浏览器和 node.js
- 支持 promise
- 能拦截请求和响应
- 能转换请求和响应数据
- 能取消请求
- 自动转换 JSON 数据
- 浏览器端支持防止 CSRF(跨站请求伪造)

Axios 安装方式

1) 可以下载到本地进行 js 的引入

<script src="axios.js"></script>

2) 可以参考 axios 官网进行引入 这种方式不用将 js 下载到本地

<http://www.axios-js.com/docs/>

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

html 文件中对应操作

引包 （两种方式都可以）

```
<!-- <script src="https://unpkg.com/axios/dist/axios.min.js"></script>-->
<script src="js/axios.min.js"></script>
```

通过 axios 发送请求

```
<script>
    axios.get("/a/test").then(function (value) {
        console.log(value)
    })

```

Controller 中对应的请求处理（RestController 注解）

```
@RestController
@RequestMapping("/a")
public class TestController {
    @RequestMapping("/test")
    public String a(){
        return "abcd";
    }
}
```

启动服务器后响应 浏览器控制台输出的内容

```
▶ {data: "abcd", status: 200, statusText: "", headers: {...}, config: {...}, ...} a:16
```

data 为后台响应的数据

如果返回的是对象

```
@RequestMapping("/a")
public class TestController {
    @RequestMapping("/b")
    public Student b(){
        Student student = new Student();
        student.setName("abc");
        student.setAge(11);
        return student;
    }
}
```

页面中请求的地址为

```
<script>
  axios.get("/a/b").then(function (value) {
    console.log(value)
  })

```

启动服务器后响应 浏览器控制台输出的内容

```

a:16
▼ {data: {...}, status: 200, statusText: "", headers: {...}, config: {...}, ...} ⓘ
  ► data: {name: "abc", age: 11}
    status: 200
    statusText: ""
  ► headers: {content-type: "application/json;charset=UTF-8", date: "Mon, 16 ...
  ► config: {transformRequest: {...}, transformResponse: {...}, timeout: 0, xsrfC...
  ► request: XMLHttpRequest {readyState: 4, timeout: 0, withCredentials: fals...
  ► __proto__: Object

```

可以看到 在 data 中的数据为响应数据，在 axios 中，成功返回的数据，通过 then() 方法进行返回，并且把数据封装为一个对象

在响应的对象组成的分析

data: 后台响应的数据

headers: 响应头等信息

config: 对应的一些配置文件

request: 在 request 对象中可以看到 axios 仍然是以 XMLHttpRequest 对象为核心，请求发送

另外，还可以发送请求时携带参数 例如

```

axios.get('/user?ID=12345')
  .then(function (response) {
    // handle success
    console.log(response);
  })
  .catch(function (error) {

```

或者

```
axios.get('/user', {  
  params: {  
    ID: 12345  
  }  
})  
.then(function (response) {  
  console.log(response);  
})
```

也可以调用 post 方法

```
axios.post('/user', {  
  firstName: 'Fred',  
  lastName: 'Flintstone'  
})  
.then(function (response) {  
  console.log(response);  
})
```

在获取到数据后 我们要将这些在页面中进行显示

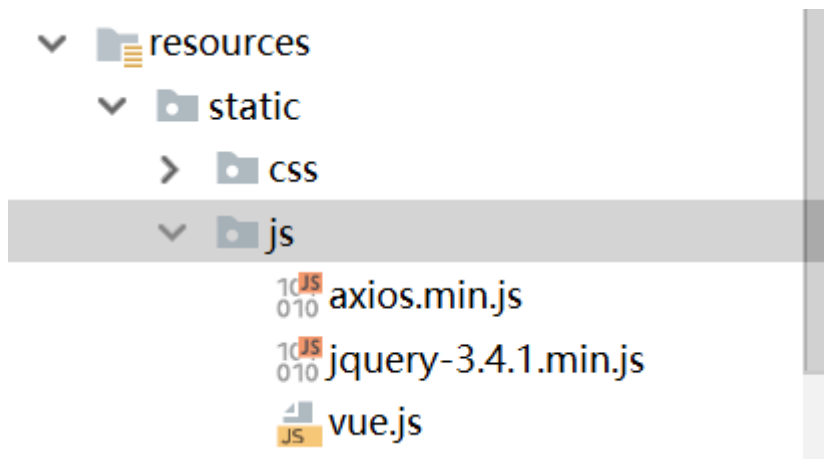
现在我们通过前端框架 vue 进行数据的渲染

Vue (读音 /vju:/, 类似于 view) 是一套用于构建用户界面的渐进式框架。

与其它大型框架不同的是, Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层, 不仅易于上手, 还便于与第三方库或既有项目整合。

使用方式:

引入对应的 vue.js 的文件



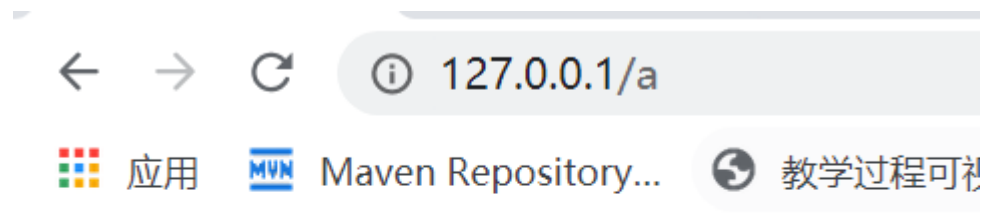
页面中引入

```
<body>
  <div id="di">a page{{msg}}</div>
  <script src="js/vue.js"></script>
  </body>
```

如果使用 vue 则要创建 vue 对象，并且指定对应的参数配置

```
new Vue({
  //el 数据挂载点 数据的作用范围
  el: "#di",
  //要显示在页面中的数据
  data: function () {
    return {
      msg: "mess"
    }
  }
})
```

在 data 中的数据 key 为 a 的数据对应的值为 mess，将数据渲染到页面中的方式为 {{}} 双大括号包裹的形式
启动服务器后，页面的响应为



a pagemess

在页面中的绑定的数据进行显示

vue 中的常用指令

v-text 文本绑定

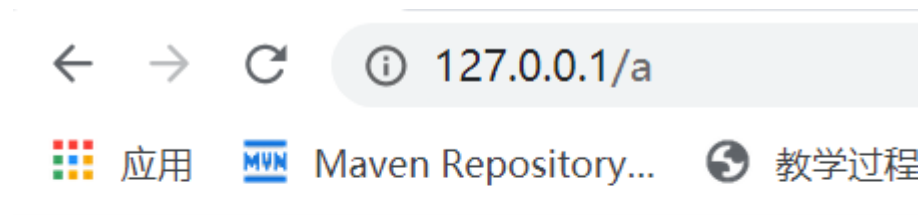
```
<span v-text="msg"></span>
<!-- 和下面的一样 -->
<span>{{msg}}</span>
```

v-if 根据条件是否成立进行绑定

v-else 与 if 形成一组条件

```
data:function () {
  return{
    msg:"mess",
    show:true
  }
}
```

```
<div v-if="show">show</div>
<div v-else>not show</div>
```

a pagemess
show

只显示了 show

v-for 遍历循环
遍历数组

```
<ul>  
  <li v-for="item in lists">{{item}}</li>  
</ul>
```

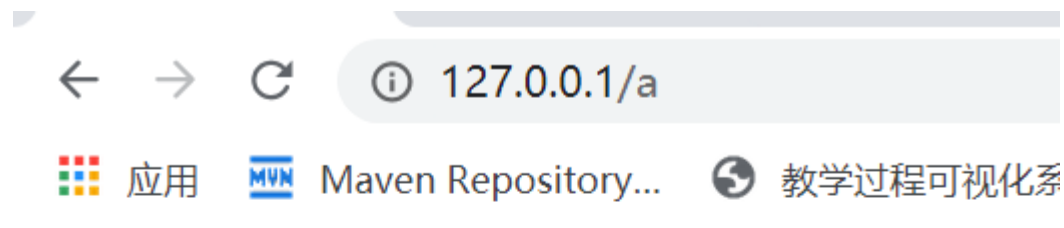
```
data:function () {  
  return{  
    msg:"mess",  
    show:true,  
    lists:[1,2,3,4,5,6]  
  }  
}
```

a pagemess
show

- 1
- 2
- 3
- 4
- 5
- 6

遍历对象 objs

```
data: function () {  
    return {  
        msg: "mess",  
        show: true,  
        lists: [1, 2, 3, 4, 5, 6],  
        objs: [  
            {id: 1, name: "apple"},  
            {id: 2, name: "apple"},  
            {id: 3, name: "apple"},  
            {id: 4, name: "apple"},  
            {id: 5, name: "apple"}  
        ]  
    }  
}
```



a pagemess

- { "id": 1, "name": "apple" }
- { "id": 2, "name": "apple" }
- { "id": 3, "name": "apple" }
- { "id": 4, "name": "apple" }
- { "id": 5, "name": "apple" }

对象可以遍历，也可以取出某个属性对应的值

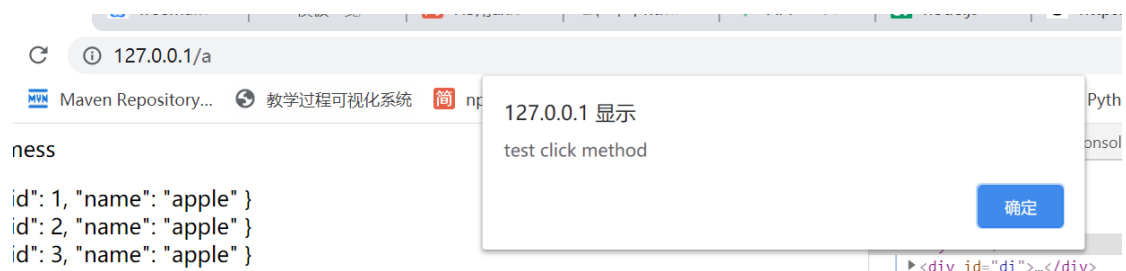
还可以使用 v-on 绑定事件，渐变写法为@事件类型 例如单击事件可以写成 @click

```
<button @click="test()">test button</button>
```

单机时触发 test 事件，事件为一个方法，可以将其定义在 vue 对象中 methods 属性中

```
,
methods: {
  test: function () {
    alert("test click method")
  }
}
```

点击时触发事件



在后台响应的数据我们可以通过 vue 进行渲染

当页面加载完成时，执行的函数为 created

```
},
created: function () {
  // 将当前的vue对象进行保存 否则在then()方法中指的是封装好的对象
  var _this=this;
  axios.get("/a/b").then(function (value) {
    _this.dat=value.data
    console.log( value)
  })
}
```

只有在 data 中的数据才可以在页面进行渲染，所以预先定义

```
data: function () {
  return {
    dat: "",
  }
}
```

html 部分

```
<div v-text="dat"></div>
```

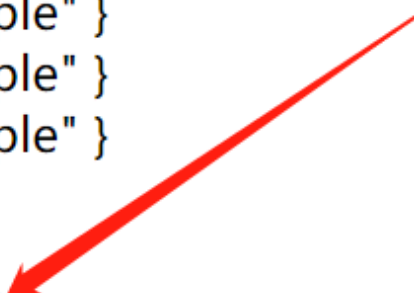
访问后发现

a pagemess

- { "id": 1, "name": "apple" }
- { "id": 2, "name": "apple" }
- { "id": 3, "name": "apple" }
- { "id": 4, "name": "apple" }
- { "id": 5, "name": "apple" }

test button

{ "name": "abc", "age": 11 }



Controller 响应的数据被在页面中获取到