Javascript 高级 ajax、jsonp

在前面学习 javascript 时我们就说过，javascript 在 2003 年时面临着被程序界淘汰的局面，但是 javascript 除了在页面上做些广告特效外，没变法与后台数据交互，这样就显得鸡肋，在加上这个广告使得用户反感，所以就被程序界有所放弃。但是在 2003 年微软的工程师在 com+控件中封装得到了一个 XMLHttpRequest 对象在后，被 Google 工程师发扬光大发明了 Ajax 技术，使得 javascript 焕发了新的生机，可以说 Ajax 就是 javascript 得以生存到现在的直接原因。随着 ajax 技术的不断被使用，javascript 在程序界就显得更加重要了，特别是在 10 年左右的 node.js 的出现，javascript 已经不仅仅是在客户端编程了，在服务器端也有了一席之地。移动互联网的发展也为 javascript 的越来越火提供了强大的动力。

Form 表单的回顾

在前面学习 HTML 时，我们学习过 form 表单，但是因为没有服务器，所以大家可能不清楚 form 标签中的一些属性的作用，这里就稍微回顾下：

```
<!--
    action表示服务器请求的路径，method表示请求的方法类型
    get请求会将数据显示在地址栏上，而且get请求在服务器端编码需求特殊处理
    所以如果不是要求使用get请求，尽量不要使用get请求
    post不会显示，但是这两种都不太安全，使用一些抓包工具就可以将我们
    输入页面的信息得到，建议大家使用安全的网络，不要随意连接陌生的网络
 -->
<form action="ajax.do" method="post">

    用户名称：<input type="text" name="username" id="username"
class="username" /><br />
    用户密码：<input type="password" name="password" id="password"
class="password" /><br />

    <input type="submit"  id="submit" value="注册用户" />
    <input type="reset"  id="reset" value="重置数据" />
</form>
```

服务器端使用 java 搭建，当我们使用 get 和 post 做实验时发现，get 请求的编码需要做特殊处理，而 post 就简单点：

```
ajax Hello
胜多负少------------sdfsa
```

原生 Ajax 的使用

下面我们使用 ajax 来从服务器端获取数据：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>form测试</title>
        <script type="text/javascript">

            /**
             * ajax的使用分四步走
             */
            function getDate() {
                //1、创建XMLHttpRequest对象，注意，IE6的兼容问题
                var xhr = new XMLHttpRequest();
                //打开连接，open方法有三个参数
                /**
                 * 第一个是访问的请求方式
                 * 第二个是访问服务器的URL
                 * 第三个是一个Boolean值，true表示异步，
                 * false表示同步，一般为异步，默认是true
                 *
                 * 当请求是GET时，参数可以直接在URL后面拼接，如：
                 * ajax.do?name=liujianhong&password=123
                 * 当参数拼接时，一定要加?表示携带参数，当存在多个参数时，
                 * 每个参数间使用&拼接，注意参数时键值对的方式，以等号连接
                 */
                xhr.open("GET","ajax.do",true);

                //3、ajax的回调函数
                xhr.onreadystatechange = function() {
                    /**
                     * 注意：回调函数会触发好几次，但是我们一般只会使用最后一次
                     * 就是请求完成的触发,readState 等于4的时候
                     */
                    /**
                     *  0    请求未初始化（在调用 open() 之前）
```

```
                     *   1    请求已提出（调用 send() 之前）
                     *   2    请求已发送（这里通常可以从响应得到内容头部）
                     *   3    请求处理中（响应中通常有部分数据可用，但是服务器还没有完
成响应）
                     *   4     请求已完成（可以访问服务器响应并使用它）
                     */
//                   alert(xhr.readyState);
                     /**
                      * readyState等于4表示请求已经结束，服务器响应完成
                      * status表示http请求的状态，200表示正常响应
                      * 404表示资源不可达（找不到）
                      * 500表示服务器端错误
                      */
                     if (xhr.readyState == 4 && xhr.status == 200) {
                         //说明请求完成，并且成功
                         alert(xhr.responseText);//得到服务器端的文本数据
                         alert(xhr.responseXML);//得到服务器端的XML数据
                     }
                 };
                 /**
                  * 发送数据，如果没有数据，可以不传递数据或者传递null,
如:xhr.send(null);
                  * 如果post请求传递数据，则这样写
                  * 首先设置一下xhr的请求头信息
                  * xhr.setRequestHeader("Content-type","application/x-www-form-
urlencoded");
                  * 再这样传递参数
                  * xhr.send(name=liujianhong&password=123);
                  */
                 xhr.send();
             }
             //下面代码是为了兼容IE6及其以下版本，现在可以不做考虑
/*           function ajaxFunction() {
                 var xmlHttp;
                 try {
                     // Firefox, Opera 8.0+, Safari
                     xmlHttp = new XMLHttpRequest();
                 } catch(e) {
                     // Internet Explorer
                     try {
                         xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
                     } catch(e) {
                         try {
                             xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
```

```
                        } catch(e) {
                            alert("您的浏览器不支持AJAX! ");
                            return false;
                        }
                    }
                }
                return xmlHttp;
            }*/
        </script>
    </head>
    <body>
        <button onclick="getDate()">请求数据</button>
        <div id="content"></div>
    </body>
</html>
```

Ajax 的使用包括了四步：

1、 得到 XMLHttpRequest 对象，注意如果要兼容 IE6 则使用我注释的代码

2、 使用 XMLHttpRequest 对象的 open 方法，注意参数问题

3、 Ajax 的回调函数，服务器响应后，得到数据的处理

4、 发送请求 send 方法。

下面我们使用 ajax 来做一个案例：

因为 ajax 请求服务器可以响应两种数据过来：XML 和 json，所以我们针对两种情况来处理：

XML 数据的处理：

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Insert title here</title>
        <script type="text/javascript">
            window.onload = init;
            function init() {
                //1、获取部门节点
                var dn = document.getElementById("dep");
                //2、为该节点创建onchange
                dn.onchange = getPerson
                    //3、创建一个getPerson的方法来处理事件
            }
            function getPerson() {
```

```
                var did = this.value;
                //1、获取XMLHttpRequest;
                var xhr = new XMLHttpRequest();
                //2、通过xhr来打开页面，使用POST
                xhr.open("POST", "person.do", true);
                xhr.onreadystatechange = function() {
                    //3、处理请求
                    if(xhr.readyState == 4 && xhr.status == 200) {
                        //3.1、获取xml节点
                        var xmlDoc = xhr.responseXML;
                        //3.2、获取所有的person节点
                        var pns = xmlDoc.getElementsByTagName("person");
                        //3.3、遍历所以节点，获取id，name等信息
                        var node = "";
                        for(var i = 0; i < pns.length; i++) {
                            node += getValueByProp(pns[i], "id") + "----------" +
                                getValueByProp(pns[i], "name") + "----------" +
                                getValueByProp(pns[i], "age") + "<br/>"
                        }
                        //3.4、写入到persons
                        document.getElementById("persons").innerHTML = node;
                    }
                }
                xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
                xhr.send("did=" + did);
                //4、发送信息,需要传入did
            }
            //根据节点获取值
            function getValueByProp(node, prop) {
                return(node.getElementsByTagName(prop))[0].firstChild.nodeValue;
            }
        </script>
    </head>
    <body>
        <select id="dep">
            <option value="1">普通组</option>
            <option value="2">明星组</option>
            <option value="3">西游组</option>
        </select>
        <div id="persons"></div>
    </body>
</html>
```

在 XML 数据在接受是使用 responseXML 来接受，之后就是解析数据了，这些事 DOM 中

知识，相信大家都没有问题的。

Json 数据的处理：

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Insert title here</title>
        <script type="text/javascript">
            window.onload = init;
            function init() {
                //1、获取部门节点
                var dn = document.getElementById("dep");
                //2、为该节点创建onchange
                dn.onchange = getPerson
                    //3、创建一个getPerson的方法来处理事件
            }
            function getPerson() {
                var did = this.value;
                //1、获取XMLHttpRequest;
                var xhr = new XMLHttpRequest();
                //2、通过xhr来打开页面，使用POST
                xhr.open("POST", "personJson.do", true);
                xhr.onreadystatechange = function() {
                    //3、处理请求
                    if(xhr.readyState == 4 && xhr.status == 200) {
                        //3.1、获取json
                        var json = xhr.responseText;
                        //如果传递的是json可以直接通过xhr.responseText获取。
                        //3.2、此时json是一个字符串，如果要转换为对象需要使用eval
                        var ps = eval(json);
                        var node = "";
                        for(var i = 0; i < ps.length; i++) {
                            //json就是已经是个javascirpt的对象了，可以直接使用
                            node += ps[i].id + "--------" + ps[i].name + "--------" +
                                ps[i].age + "<br/>";
                        }
                        //3.4、写入到persons
                        document.getElementById("persons").innerHTML = node;
                    }
                }
                xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
                xhr.send("did=" + did); //4、发送信息,需要传入did
            }
```

```
        </script>
    </head>
    <body>
        <select id="dep">
            <option value="1">普通组</option>
            <option value="2">明星组</option>
            <option value="3">西游组</option>
        </select>
        <div id="persons"></div>
    </body>
</html>
```

Json 接受数据是使用 responseText 来接受，默认是字符串，所以要将字符串转换为 json 数据，使用 eval 这个全局函数，将字符串转换为 javascript 脚本，javascript 就能识别为 json 对象了。

服务器端的项目是 java 项目，我压缩了，大家导入 eclipse 中就可以使用了

hellAjax.zip

注意：XML 和 json 在项目中可能都会使用，虽然现在越来越多的公司使用 json 来传输数据，但是 XML 依旧可能使用，要求大家两种都掌握。

jQuery 中的 Ajax

ajax 虽然很好用，但是原生的写法有点复杂，所以一些 js 库对它的四步进行了封装，让 ajax 更加的好用，那么下面我们以 jQuery 为例，看看 jQuery 的 ajax 如何使用。

jQuery 提供了好多 ajax 的方法，最原始的一个是$.ajax()方法：

jQuery 原生 ajax 方法：

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Insert title here</title>
        <script type="text/javascript" src="js/jquery-1.11.1.js"></script>
        <script type="text/javascript">
            $(function() {
```

```
                $("#dep").on("change", getPerson);
            });

            function getPerson() {
                var did = this.value;
                $.ajax({
                    type: "post"//表示请求类型,
                    url: "personJson.do",//请求的URL
                    data: "did=" + did, //请求的参数
                    async: true, //是否异步
                    success: function(data) {
                        //成功后的回调函数传递的参数就是服务器端传递的参数
                        var ps = eval(data);
                        var node = "";
                        for(var i = 0; i < ps.length; i++) {
                            //json就是已经是个javascirpt的对象了，可以直接使用
                            node += ps[i].id + "--------" + ps[i].name +
                                "--------" +
                                ps[i].age + "<br/>";
                        }
                        //3.4、写入到persons
                        document.getElementById("persons").innerHTML = node;
                    }
                });
            }
        </script>
    </head>
    <body>
        <select id="dep">
            <option value="1">普通组</option>
            <option value="2">明星组</option>
            <option value="3">西游组</option>
        </select>
        <div id="persons"></div>
    </body>
</html>
```

Load 方法：

加载一个路径下的静态页面：

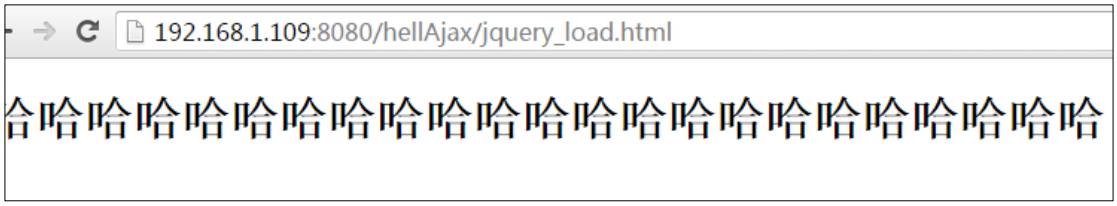
```
<!DOCTYPE html>
<html>
```

```
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Insert title here</title>
        <script type="text/javascript" src="js/jquery-
1.11.1.js" ></script>
        <script type="text/javascript">

            $(function(){
                $("#content").load("a.html");
            });

        </script>
    </head>
        <div id="content"></div>
    <body>
    </body>
</html>
```

此时 a.html 页面：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <h1>哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈</h1>
    </body>
</html>
```

运行结果：



Load 方法不常用，知道就行了。

jQuery 中的 ajax 原生方法使用的比较多，除了这个以外，最常使用就是如下写方法：

jQuery.get(url, *[data], [callback], [type]*)

jQuery.post(url, *[data], [callback], [type]*)

get 方法和 post 方法用法一样，就是 jQuery 认为 method 只有两个值，如果我们确实想使用其中一个值，就可以对应的使用其中一个方法。

```
<!DOCTYPE html>
<html>

    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Insert title here</title>
        <script type="text/javascript" src="js/jquery-1.11.1.js"></script>
        <script type="text/javascript">
            $(function() {
                $("#dep").on("change", getPerson);
            });

            function getPerson() {
                var did = this.value;
                $.get("personJson.do", {"did=": did },
                    function(data) {
                        var ps = eval(data);
                        var node = "";
                        for(var i = 0; i < ps.length; i++) {
                            node += ps[i].id + "--------" + ps[i].name + "--------" +
                                ps[i].age + "<br/>";
                        }
                        document.getElementById("persons").innerHTML = node;
                    });
            }
        </script>
    </head>

    <body>
        <select id="dep">
            <option value="1">普通组</option>
            <option value="2">明星组</option>
            <option value="3">西游组</option>
        </select>
        <div id="persons"></div>
    </body>

</html>
```

第一个参数是 URL，第二个参数可选，如果有数据可以以 json 的形式传递数据，但是 get 方法中也可以在 URL 后面拼接，post 就只能第二个参数传递了，第三个参数是回调函数，

第四个参数表示以什么形式的数据接收数据，数据类型。

jQuery 中 ajax 的方法最常用的就是 ajax、get、post 这三个方法，希望大家好好的使用，熟练掌握。

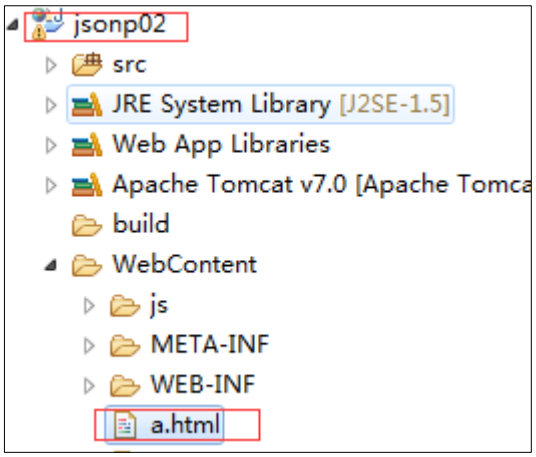
Jsonp

前面我们学习了 ajax 的相关知识，大家都感觉 ajax 相当好用，但是 ajax 存在了一个很大的问题，就是 ajax 无法跨域访问。

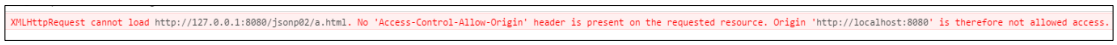
如我们存在两个项目：jsonp01 和 jsonp02，我们在 jsonp01 的页面中调用如下代码：

```
$(function(){
    $("#content").load("http://127.0.0.1:8080/jsonp02/a.html");
});
```

当然 jsonp02 中是存在 a.html 页面的：



但是当我们运行页面的时候，发现无法使用 ajax 的 load 方法加载：



提示我们是 ajax 无法跨域取数据，因为我们在这里使用了 IP 地址访问，此时默认是跨域访问的（注意：这种方式在程序界很常见的，我们要经常跨域取访问数据的）。

Jsonp 的原理

虽然 ajax 无法跨域，但是 javascript 却是可以跨域访问的，所以一些天才程序员利用 javascript 跨域访问的方式将要调用的函数传递到了服务器端，由服务器端拼接成 javascript 代码，再写回到客户端完成调用，这种方式就叫做 jsonp。

做个案例看看：

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Insert title here</title>
        <script type="text/javascript" src="js/jquery-1.11.1.js" ></script>
        <script type="text/javascript">
            //在jsonp01中定义一个函数show
            function show(msg) {
                alert(msg);
            }
        </script>

        <!-- javascript支持跨域，所以这样写是没问题的 -->
        <script type="text/javascript" src="http://127.0.0.1:8080/jsonp02/a.js"></script>
    </head>
    <body>
        <div id="content"></div>
        <h1>你好</h1>
    </body>
</html>
```

我们在 jsonp01 的页面上写了一个函数，但是没有调用它，之后我们跨域导入了 jsonp02 的一个脚本，脚本这样写：

```
show("hello jsonp");
```

运行页面：

我们发现 a.js 脚本的代码运行了，说明 javascript 的确支持跨域，所以一些天才程序员利用 javascript 跨域访问的方式将要调用的函数传递到了服务器端，由服务器端拼接成 javascript 代码，再写回到客户端完成调用，这种方式就叫做 jsonp。

下面我们就以一个案例来说明：

```
<html>
```

```
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Insert title here</title>
        <script type="text/javascript" src="js/jquery-1.11.1.js" ></script>
        <script type="text/javascript">
            //在jsonp01中定义一个函数show
            function show(msg) {
                alert(msg.name);
            }
        </script>

        <!--
            利用javascript跨域的特性，将要调用函数传递到服务器端，
            有服务器端拼接为调用的函数，再返回客户端执行
         -->
        <script type="text/javascript"
src="http://127.0.0.1:8080/jsonp02/json.do?callback=show&did=1"></script>
    </head>
    <body>
        <div id="content"></div>
        <h1>你好</h1>
    </body>
</html>
```

在客户端使用参数传递的方式，将方法名称传递过去，在服务器端完成拼接：

```
        call   request              "callback"
        out          call  //得到页面传递过来的方法名称
        str   call "(" px            ")" //拼接为方法调用
```

再讲拼接完成的字符串写回到客户端，这样客户端就可以得到执行这段脚本：

jQuery 的 jsonp 的实现

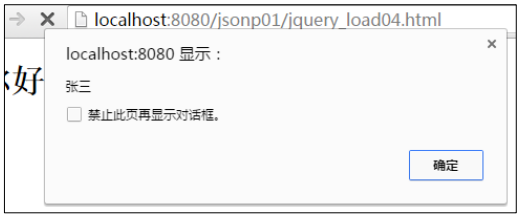
这样就完成了跨域的访问了，基于这种情况，jQuery 为我们提供了使用 getJSON 方法或者使用其他 jQuery 中 ajax 方法（其他的就需要说明类型为 jsonp）来完成 ajax 的跨域：

我们就完成不用管这个函数名称了，jQuery 会自动的生成一个随机不重复的方法名称，

我们只要关系调用得到的数据问题就行了，如：

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Insert title here</title>
        <script type="text/javascript" src="js/jquery-1.11.1.js" ></script>
        <script type="text/javascript">
            //我们不用关心callback所对应的方法名称了
            //jQuery会自动生成一个不会重复的随机函数名称
            //这个函数在服务器端拼接后，会在getJSON的第二个参数
            //就是这个回调函数，如下面：
            $.getJSON("http://127.0.0.1:8080/jsonp02/json.do?callback=?&did=1",function(date){
                alert(date.name);
            });
        </script>
    </head>
    <body>
        <div id="content"></div>
        <h1>你好</h1>
    </body>
</html>
```

调用得到结果：



在 jQuery 中，除了 getJSON 方法外，其他的 ajax 方法也可以实现 jsonp，在数据类型中加入 jsonp 就行了，如：

```
$.ajax({
    type:"POST",
    url:"http://127.0.0.1:8080/jsonp02/json.do",
    data:{"did":1},
    dataType:"jsonp",//类型必须写错jsonp
    jsonp:"callback",//注意，这个可以不写，默认就是callback，如果写了服务器端必须一致
    success: function(data) {
        alert(data.name)
    }
});
```

注意：ajax 和 jsonp 和 json 的区别：

Ajax 是就要 XMLHttpRequest 对象实现的一种可以异步访问服务器的技术

Jsonp 是一种使用 JSON 数据的方式，返回的不是 JSON 对象，是包含 JSON 对象的 javaScript 脚本，辅助 ajax 实现跨越的技术（ajax 本身无法跨越取数据）。

Json 则是一种轻量级的数据交换格式，像 xml 一样，是用来描述数据间的关系，javascript 中默认是对象。

附录

http 请求状态码：

status 返回当前请求的 http 状态码

status 属性返回当前请求的 http 状态码,此属性仅当数据发送并接收完毕后才可获取。完整的 HTTP 状态码如下:

100 Continue 初始的请求已经接受，客户应当继续发送请求的其余部分

101 Switching Protocols 服务器将遵从客户的请求转换到另外一种协议

200 OK 一切正常，对 GET 和 POST 请求的应答文档跟在后面。

201 Created 服务器已经创建了文档，Location 头给出了它的 URL。

202 Accepted 已经接受请求，但处理尚未完成。

203 Non-Authoritative Information 文档已经正常地返回，但一些应答头可能不正确，因为使用的是文档的拷贝

204 No Content 没有新文档，浏览器应该继续显示原来的文档。如果用户定期地刷新页面，而 Servlet 可以确定用户文档足够新，这个状态代码是很有用的

205 Reset Content 没有新的内容，但浏览器应该重置它所显示的内容。用来强制浏览器清除表单输入内容

206 Partial Content 客户发送了一个带有 Range 头的 GET 请求，服务器完成了它

300 Multiple Choices 客户请求的文档可以在多个位置找到，这些位置已经在返回的文档内列出。如果服务器要提出优先选择，则应该在 Location 应答头指明。

301 Moved Permanently 客户请求的文档在其他地方，新的 URL 在 Location 头中给出，浏览器应该自动地访问新的 URL。

302 Found 类似于 301，但新的 URL 应该被视为临时性的替代，而不是永久性的。

303 See Other 类似于 301/302，不同之处在于，如果原来的请求是 POST，Location 头指定的重定向目标文档应该通过 GET 提取

304 Not Modified 客户端有缓冲的文档并发出了一个条件性的请求（一般是提供 If-Modified-Since 头表示客户只想比指定日期更新的文档）。服务器告诉客户，原来缓冲的文档还可以继续使用。

305 Use Proxy 客户请求的文档应该通过 Location 头所指明的代理服务器提取

307 Temporary Redirect 和 302（Found）相同。许多浏览器会错误地响应 302 应答进行重定向，即使原来的请求是 POST，即使它实际上只能在 POST 请求的应答是 303 时才能重定向。由于这个原因，HTTP 1.1 新增了 307，以便更加清除地区分几个状态代码：当出现 303 应答时，浏览器可以跟随重定向的 GET 和 POST 请求；如果是 307 应答，则浏览器只能跟随对 GET 请求的重定向。

400 Bad Request 请求出现语法错误。

401 Unauthorized 客户试图未经授权访问受密码保护的页面。应答中会包含一个 WWW-Authenticate 头，浏览器据此显示用户名字/密码对话框，然后在填写合适的 Authorization 头后再次发出请求。

403 Forbidden 资源不可用。

404 Not Found 无法找到指定位置的资源

405 Method Not Allowed 请求方法（GET、POST、HEAD、Delete、PUT、TRACE 等）对指定的资源不适用。

406 Not Acceptable 指定的资源已经找到，但它的 MIME 类型和客户在 Accpet 头中所指定的不兼容

407 Proxy Authentication Required 类似于 401，表示客户必须先经过代理服务器的授权。

408 Request Timeout 在服务器许可的等待时间内，客户一直没有发出任何请求。客户可以在以后重复同一请求。

409 Conflict 通常和 PUT 请求有关。由于请求和资源的当前状态相冲突，因此请求不能成功。

410 Gone 所请求的文档已经不再可用，而且服务器不知道应该重定向到哪一个地址。它和 404 的不同在于，返回 407 表示文档永久地离开了指定的位置，而 404 表示由于未知

的原因文档不可用。

411 Length Required 服务器不能处理请求，除非客户发送一个 Content-Length 头

412 Precondition Failed 请求头中指定的一些前提条件失败

413 Request Entity Too Large 目标文档的大小超过服务器当前愿意处理的大小。如果服务器认为自己能够稍后再处理该请求，则应该提供一个 Retry-After 头

414 Request URI Too Long URI 太长

416 Requested Range Not Satisfiable 服务器不能满足客户在请求中指定的 Range 头

500 Internal Server Error 服务器遇到了意料不到的情况，不能完成客户的请求

501 Not Implemented 服务器不支持实现请求所需要的功能。例如，客户发出了一个服务器不支持的 PUT 请求

502 Bad Gateway 服务器作为网关或者代理时，为了完成请求访问下一个服务器，但该服务器返回了非法的应答

503 Service Unavailable 服务器由于维护或者负载过重未能应答。例如，Servlet 可能在数据库连接池已满的情况下返回 503。服务器返回 503 时可以提供一个 Retry-After 头

504 Gateway Timeout 由作为代理或网关的服务器使用，表示不能及时地从远程服务器获得应答

505 HTTP Version Not Supported 服务器不支持请求中所指明的 HTTP 版本

事实上,我们只需要知道状态为 200 的时候(OK)才读取 response 就行了!