学习方法

1. 建立逻辑思维
   1. 多分析
   2. 理解逻辑算法
   3. 多练、多敲、多读
2. 编程习惯
3. 大胆动手
4. 保持好奇好
5. 乐观面对错误

就业能力

1. 分析能力
2. 独立思考能力
3. 程序阅读能力
4. 程序排错能力
5. 独立编写程序能力
6. 团队协作能力
7. 学习能力
8. 善于总结的能力

专业技能

Ajax下/promise

1. 回调函数的嵌套
   1. 假设你的业务出现了三个定时器嵌套的逻辑：  
      setTimeout(function(){  
       setTimeout(function(){  
       setTimeout(function(){  
         
       })  
       })  
       })
   2. 在JavaScript语言中，无论是写浏览器端的各种事件处理回调、还是ajax回调，不得不面对的问题就是各种回调函数。回调函数少了还好，一旦多了起来而且必须讲究顺序的话，回调函数开始嵌套，那代码的恶心程度是相当不符合常人的线性思维的。
      1. 就像下面这样：  
         你不在乎下面这三个ajax的执行顺序还好  
         如果你在乎顺序呢？  
         ajax(“url”,function(){});  
         ajax(“url1”,function(){});  
         ajax(“url2”,function(){});  
         就像这样？  
         ajax(“url”,function(){  
          ajax(“url1”,function(){  
          ajax(“url2”,function(){  
          });  
          });  
         });
2. Ajax的依赖调用
   1. 在使用ajax异步调用的时候，可能碰到同时调用多个ajax这种情况。而且多个ajax之间还存在依赖关系。这种情况怎么处理呢？
      1. 多个ajax嵌套调用，这时需要设置open(“GET/POST”,url,false),也就是同步的了，但是这种方法存在一个弊端，如果后台出现问题，一直忙，不能返回数据，浏览器将出现卡死的情况。
3. Promise介绍
   1. ES6中新推出的promise函数，是一个构造函数，它实际上是对回调函数的一种封装，对异步编程的一种改进。  
      下面我们看看它能解决哪些以前难以解决的问题：
4. Promise改造ajax依赖调用
   * 1. 问题1：当有两个或以上的ajax请求需要按顺序执行，该如何编写？  
        //以下是伪代码  
        ajax1({  
         url:”xxxx”,  
         success:function(data1){  
         //在这里，当ajax1完成后，才能开启ajax2  
         ajax2({  
         url:”xxxx”,  
         success:function(data2){  
         //第二次完成的回调，可能还有第三个/第四个无止境的嵌套…  
         }  
         });  
         },  
         error:function(msg){  
         console.log(“请求出现错误。”,msg);  
         }  
         });  
        而使用promise的写法如下：  
        new Promise(function(resolve,reject){  
         ajax1({  
         url:”xxxx”,  
         success:function(data){  
           
         resolve();  
         },  
         error:function(msg){  
         reject();  
         }  
         });  
        }).then(function(){  
         ajax2({  
         url:”xxxx”,  
         success:function(data){  
           
         }  
         });  
        }).catch(function(){  
         console.log(“请求出现错误：”,msg);  
        });  
        也就是代码被改进成了new Promise(ajax1).then(ajax2).catch()  
        使用了链式调用代替了嵌套结构。  
        一个 Promise 对象可以理解为一次将要执行的操作（常常被用于异步操作），使用了 Promise 对象之后可以用一种链式调用的方式来组织代码，让代码更加直观。而且由于 Promise.all 这样的方法存在，可以让同时执行多个操作变得简单。  
        resolve()和reject()则意味着成功回调和失败回调的执行。  
        看一段使用Promise对象的代码  
        function helloWorld(ready){  
         return new Promise(function(resolve,reject){  
         if(ready){  
         resolve(“Hello World!”);  
        //除了标志成功外，还具体给下一次调用返回的值  
         } else{  
         reject(“Good bye!”);  
         }  
         });  
        }  
        helloWorld(true).then(function(message){  
         alert(message);  
        },function(error){  
         alert(error);  
        });  
        上面的代码，helloWord函数接受一个参数，如果为true就打印”Hello World!”,如果为false就打印错误的信息。helloWorld函数返回的是一个Promise对象。  
        resolve 方法可以使 Promise 对象的状态改变成成功，同时传递一个参数用于后续成功后的操作，在这个例子当中就是 *Hello World!* 字符串。  
        reject 方法则是将 Promise 对象的状态改变为失败，同时将错误的信息传递到后续错误处理的操作。  
        function printHello(ready){

return new Promise(function (resolve,reject){

if(ready){

resolve("Hello");

}else{

reject("Good Bye!");

}

});

}

printHello(true).then(function(message){

return message;

}).then(function(message){

return message + ' World';

}).then(function(message){

return message + ' !';

}).then(function (message){

alert(message);

});  
如果有三个或更多的ajax请求呢？  
new Promise(function(resolve,reject){  
 ajax1({  
 url:”xxxx”,  
 success:function(data){  
 resolve();  
 //use data…….  
 },  
 error:function(msg){  
 reject();  
 }  
 });  
 }).then(function(){  
 return new Promise(function(resolve,reject){  
 ajax2({  
 url:”xxx”,  
 success:function(data){  
 resolve();  
 //use data……  
 },  
 error:function(msg){  
 reject();  
 }  
 })  
 })  
 }).then(function(){  
 ajax3({  
 url:”xxxx”,  
 success:function(data){  
 //use data……  
 },  
 error:function(msg){  
 //print msg  
 }  
 })  
 });  
为了简化代码，我这里将catch方法省略了  
代码结构大概是这样的：new Promise(ajax1).then(function(){return new Promise(ajax2)}).then(ajax3);  
如果有第四个请求要顺序执行，则大概写成这样：  
new Promise(ajax1).then(function(){ return new Promise(ajax2)}),then(function(){ return new Promise(ajax3)}),then(ajax4);  
从这里仔细观察，可以看出，只有被Promise封装过了，回调才能保证顺序。  
也就是Promise（意为承诺）设计的初衷  
但前一个方法必须在它最后，执行resolve(),后一个方法才可以开始。  
如果执行了reject(),则进入catch()方法。  
其实这里不能单纯的理解为resolve就是success，reject就是error  
现就职于阿里的大名鼎鼎的阮一峰老师，喜欢管它叫状态机，这是非常恰当的叫法。  
只有理解了它的设计思路，才能明白为什么叫状态机。  
我们把代码写的再实际一些  
new Promise(function(resolve,reject){ //p1  
 ajax1({  
 ……  
 resolve()  
 });  
}).then(function(){  
 return new Promise(resolve,reject){ //p2  
 ajax2({  
 ……  
 resolve();  
 });  
 }  
}).then(function(){  
 return new Promise(resolve,reject){ //p3  
 ajax3({  
 ……  
 resovle()  
 });  
 }  
 })  
如果，我们把每一个promise对象都看做一个状态机的话，它实际上只有三种状态 进行中，已完成，已失败，它分别代表了所承诺的回调函数的执行状态。  
而resolve方法，只是把这个状态机从Pending(进行中)，改成了Resolved(已完成)  
由于p2现在依赖了p1，因此，p2会观察p1的状态，直到p1变为Resolved，p2才会开始，但这里then方法返回的不再是p1,而是p2,因此p3会依赖p2的状态

* 1. 问题2：如果有三个ajax请求，调用顺序无关，但必须保证三个都成功，才能开始第四个？  
     Promise.all(function(){ //p123  
      return [p1,p2,p3];  
     }).then(function(){  
      return p4;  
     });  
     p4依赖p123的状态，那么理解了状态机，promise是不是如此简单呢？
  2. 问题3：如果有三个ajax请求，只要任意一个成功，则开始第四个?  
     Promise.race(function(){  
      return [p1,p2,p3];  
     }).then(function(){  
      return p4;  
     });

1. 扩展案例  
   var val = 1;

// 我们假设step1, step2, step3都是ajax调用后端或者是在Node.js上查询数据库的异步操作

// 每个步骤都有对应的失败和成功处理回调

// 需求是这样，step1、step2、step3必须按顺序执行

function step1(resolve, reject) {

console.log('步骤一：执行');

if (val >= 1) {

resolve('Hello I am No.1');

} else if (val === 0) {

reject(val);

}

}

function step2(resolve, reject) {

console.log('步骤二：执行');

if (val === 1) {

resolve('Hello I am No.2');

} else if (val === 0) {

reject(val);

}

}

function step3(resolve, reject) {

console.log('步骤三：执行');

if (val === 1) {

resolve('Hello I am No.3');

} else if (val === 0) {

reject(val);

}

}

new Promise(step1).then(function(val){

console.log(val);

return new Promise(step2);

}).then(function(val){

console.log(val);

return new Promise(step3);

}).then(function(val){

console.log(val);

return val;

}).then(function(val){

console.log(val);

return val;

});

// 执行之后将会打印

步骤一：执行

Hello I am No.1

步骤二：执行

Hello I am No.2

步骤三：执行

Hello I am No.3

Hello I am No.3

第二节：应用

1. 掌握ajax依赖调用
2. 掌握回调函数的应用
3. 掌握promise规范

第三节

1. Ajax的同源策略
   1. 概念：同源策略是客户端脚本（尤其是Javascript）的重要的安全度量标准。  
      同源策略阻止从一个域上加载的脚本获取或操作另一个域上的文档属性。也就是说，受到请求的 URL 的域必须与当前 Web 页面的域相同。这意味着浏览器隔离来自不同源的内容，以防止它们之间的操作。  
      这里的同源指的是：同协议，同域名和同端口。
   2. 精髓：它的精髓很简单：它认为自任何站点装载的信赖内容是不安全的。当被浏览器半信半疑的脚本运行在沙箱时，它们应该只被允许访问来自同一站点的资源，而不是那些来自其它站点可能怀有恶意的资源。
   3. 为什么要有同源限制？  
      我们举例说明：比如一个黑客程序，他利用IFrame把真正的银行登录页面嵌到他的页面上，当你使用真实的用户名，密码登录时，他的页面就可以通过Javascript读取到你的表单中input中的内容，这样用户名，密码就轻松到手了。
2. Jsonp跨域
   1. 什么是Jsonp？
      1. JSONP(JSON with Padding)是一个非官方的协议，它允许在服务器端集成Script tags返回至客户端，通过javascript callback的形式实现跨域访问（这仅仅是JSONP简单的实现形式）。
   2. 作用：
      1. 由于同源策略的限制，XmlHttpRequest只允许请求当前源（域名、协议、端口）的资源，为了实现跨域请求，可以通过script标签实现跨域请求，然后在服务端输出JSON数据并执行回调函数，从而解决了跨域的数据请求。
      2. 百度接口：https://sp0.baidu.com/5a1Fazu8AA54nxGko9WTAnF6hhy/su?wd=a&cb=
3. 服务器端跨域
   1. Php文件  
      <?php  
      //header('Content-type: application/json');  
      //获取回调函数名  
      $jsoncallback = htmlspecialchars($\_REQUEST ['callback']);  
      //json数据  
      $arr=array('a'=>1,'b'=>2,'c'=>3,'d'=>4,'e'=>5);  
      $result=json\_encode($arr); //转换成Json数据的存储格式{"a":1,"b":2,"c":3,"d":4,"e":5}  
      /\*  
      $json\_data = '[{name:"商品1",tip:"好用1",price:"100",url:"http://10.17.160.154/javascriptAJAX/images/1.jpg"}, {name:"商品2",tip:"好用2",price:"102",url:"http://10.17.160.154/javascriptAJAX/images/2.jpg"},{name:"商品3",tip:"好用3",price:"103",url:"http://10.17.160.154/javascriptAJAX/images/3.jpg"},{name:"商品4",tip:"好用4",price:"104",url:"http://10.17.160.154/javascriptAJAX/images/4.jpg"},{name:"商品5",tip:"好用5",price:"105",url:"http://10.17.160.154/javascriptAJAX/images/5.jpg"},{name:"商品6",tip:"好用6",price:"106",url:"http://10.17.160.154/javascriptAJAX/images/6.jpg"}]';  
      \*/  
      //输出jsonp格式的数据  
      echo $jsoncallback . "(" . $result . ")";  
      ?>
   2. Html文件  
      <script type="text/javascript">   
      function jsonpCallback(result) {   
       alert(result);   
       for(var i in result) {   
       alert(i+":"+result[i]);//循环输出a:1,b:2,etc.   
       }   
       }   
       var JSONP=document.createElement("script");   
       JSONP.type="text/javascript";  
      JSONP.src="http://localhost/jsonp.php?callback=jsonpCallback";  
      document.getElementsByTagName("head")[0].appendChild(JSONP);   
      </script>
4. Get和post提交
   1. Get方式：用get方式可传送简单数据，但大小一般限制在1KB下，数据追加到url中发送（http的header传送），也就是说，浏览器将各个表单字段元素及其数据按照URL参数的格式附加在请求行中的资源路径后面。另外最重要的一点是，它会被客户端的浏览器缓存起来，那么，别人就可以从浏览器的历史记录中，读取到此客户的数据，比如帐号和密码等。因此，在某些情况下，get方法会带来严重的安全性问题。
   2. Post方式：当使用POST方式时，浏览器把各表单字段元素及其数据作为HTTP消息的实体内容发送给Web服务器，而不是作为URL地址的参数进行传递，使用POST方式传递的数据量要比使用GET方式传送的数据量大的多。  
      总之：get方式传送数据量小，处理效率高，安全性低，会被缓存，而POST反之。
   3. 区别：
      1. Post传输数据时，不需要在URL中显示出来，而Get方法要在URL中显示。
      2. Post传输的数据量大，可以达到2M，而Get方法由于受到URL长度的限制,只能传递大约1024字节.
      3. Post顾名思义,就是为了将数据传送到服务器段,Get就是为了从服务器段取得数据.而Get之所以也能传送数据,只是用来设计告诉服务器,你到底需要什么样的数据.Post的信息作为http请求的内容，而Get是在Http头部传输的。
      4. get 方法用Request.QueryString["strName"]接收  
         post 方法用Request.Form["strName"] 接收
5. 兼容问题
   1. 虽然两种提交方式可以统一用Request("strName")来获取提交数据，但是这样对程序效率有影响，不推荐使用。

第四节：应用

1. 请求天气预报
2. 请求淘宝搜索下拉提示  
   https://suggest.taobao.com/sug?&q=b&callback=jsonp267

第五节：综合应用

1. 京东商品评论列表