13. promise

回调函数的嵌套使用：

function fun1(f){

        console.log("fun1");

        f();

    }

    function fun2(f){

        console.log("fun2");

        f();

    }

    function fun3(f){

        console.log("fun3");

        f();

    }

    function fun4(){

        console.log("fun4");

    }

    //多个回调函数 拼接起来 可读性差

    fun1(function(){

        fun2(function(){

            fun3(fun4);

        })

    });

    //加上异步操作

    function fun1(f){

        console.log("fun1");

        setTimeout(function(){

            console.log("fun1\_异步");

            f();

        },1000);

    }

    function fun2(f){

        console.log("fun2");

        setTimeout(function(){

            console.log("fun2\_异步");

            f();

        },1000);

    }

    function fun3(f){

        console.log("fun3");

        setTimeout(function(){

            console.log("fun3\_异步");

            f();

        },1000);

    }

    function fun4(){

        console.log("fun4");

        setTimeout(function(){

            console.log("fun4\_异步");

        },1000);

    }

    fun1(function(){

        fun2(function(){

            fun3(fun4);

        })

    });

\* Promise解决的问题：回调

//如果代码可以这么实现  这样书写的代码更加容易理解

//fun1().then(fun2).then(fun3).then(fun4);

即Promise将回调模式的主从关系调换了一个位置，变成了同等的只是顺序的关系。

Promise的概念：

所谓promise，简单说是一个容器，里面保存着某个未来才会结束的事件（通常是一个异步操作）的结果，从语法上说，promise是一个对象，从它可以获取异步操作的消息，promise提供了统一的API，各种异步操作都可以用同样的方法进行处理。

promise对象的特点

（1）对象的状态不受外界影响，promise对象代表一个异步操作，有三种状态，pending（进行中）、fulfilled（已成功）、rejected（已失败）。只有异步操作的结果，可以决定当前是哪一种状态，任何其他操作都无法改变这个状态，这也是promise这个名字的由来“承若”；

（2）一旦状态改变就不会再变，任何时候都可以得到这个结果，promise对象的状态改变，只有两种可能：从pending变为fulfilled，从pending变为rejected。这时就称为resolved（已定型）。如果改变已经发生了，你再对promise对象添加回调函数，也会立即得到这个结果，这与事件（event）完全不同，事件的特点是：如果你错过了它，再去监听是得不到结果的。

有了Promise对象，就可以将异步操作以同步操作的流程表达出来，避免了层层嵌套的回调函数。此外，Promise对象提供统一的接口，使得控制异步操作更加容易。

Promise也有一些缺点。

首先，无法取消Promise，一旦新建它就会立即执行，无法中途取消。

其次，如果不设置回调函数，Promise内部抛出的错误，不会反应到外部。

第三，当处于pending状态时，无法得知目前进展到哪一个阶段（刚刚开始还是即将完成）。

强调:promise对象必须作为一个函数的返回值返回

promise语法:

    let p = new Promise(function(resolve,reject){

        if(条件){

           resolve();

        }else{

           reject();

        }

    });

    //切记return promise对象

    return p;

    关键的then方法:

promise对象的简单例子：

function f(param){

        let p = new Promise(function(resolve,reject){

            if(param>=3000){

                resolve(param);

            }else{

                reject(param);

            }

        });

        return p;

    }

    function f1(a){

        console.log("成功",a);

    }

    function f2(a){

        console.log("失败",a);

    }

    f(100).then(f1,f2);

f函数根据传参判断异步条件，若参数param>=3000则我们认为成功，执行reslove，反之执行reject。

then方法的参数为回调函数，根据异步的条件，决定调用哪个回调函数，参数一为reslove,参数二为reject。

通过promise修改代码：

<script>

    function fun1(){

        console.log("fun1");

        let p = new Promise(function(resolve,reject){

            setTimeout(function(){

                console.log("fun1异步");

                resolve();

            },1000);

        });

        return p;

    }

    function fun2(){

        console.log("fun2");

        let p = new Promise(function(resolve,reject){

            setTimeout(function(){

                console.log("fun2异步");

                resolve();

            },1000);

        });

        return p;

    }

    function fun3(){

        console.log("fun3");

        let p = new Promise(function(resolve,reject){

            setTimeout(function(){

                console.log("fun3异步");

                resolve();

            },1000);

        });

        return p;

    }

    function fun4(){

        console.log("fun4");

        let p = new Promise(function(resolve,reject){

            setTimeout(function(){

                console.log("fun4异步");

                resolve();

            },1000);

        });

        return p;

    }

    fun1().then(fun2).then(fun3).then(fun4);

</script>

使用Promise对象时，

1、找到异步操作的代码，放在Prmoise构造函数的参数（函数）里

2、参数（函数）的第一个参数resolve是成功时调用的函数，对应then方法（函数）的第一个参数

3、参数（函数）的第二个参数reject是失败时调用的函数，对应then方法（函数）的第二个参数

promise:修改ajax封装

<script>

        function myAjaxPromise(type,url,data,isAsyn){

        let xhr;

        if (window.ActiveXObject) {

            //ie

            xhr = new ActiveXObject("Microsoft.XMLHttp");

        } else {

            //非ie

            xhr =  new XMLHttpRequest();

        }

        type = type.toLowerCase();

        if(type=="get"){

            let urlParam = url;

            urlParam += "?"+data;

            xhr.open(type,urlParam,isAsyn);

            xhr.send();

        }else if(type=="post"){

            xhr.open(type,url,isAsyn);

            xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded");

            xhr.send(data);

        }

        let p = new Promise(function(resolve,reject){

            xhr.onreadystatechange = function(){

                if(xhr.status == 200 && xhr.readyState == 4){

                    resolve(xhr.responseText+"成功");

                }else if(xhr.readyState == 4){

                    reject(xhr.responseText+"失败");

                }

            }

        });

        return p;

    }

    function showValue1(str){

        console.log(str);

    }

    function showValue2(str){

        console.log("heihei");

    }

    myAjaxPromise("post","tes1t.php","name=laowangwang",true).then(showValue1,showValue2);

</script>

<?php

    header("Content-type:text/htm;charset=utf-8");

    $name = $\_POST["name"];

    echo $name;

?>

14.跨域访问

\* 什么是跨域访问

    \* 跨域访问就是跨域名访问，即A网站的网页在代码上访问了B网站的页面

    \* 由于同源策略（浏览器的安全机制），所以，AJAX不能实现跨域访问。

    同源策略：这是一种浏览器策略，浏览器的安全机制：浏览器只允许来自同一个来源的文件相互访问。（前段与后端）

    同源：同样的协议，同样的地址，同样的端口

    JavaScript或Cookie只能访问同域名下的内容同样的协议，同样的地址，同样的端口。

\* AJAX不支持跨域访问，为了达到跨域访问的目的，出现了很多的解决方案 ：JSONP,iframe,flash,xhr2等。但是比较常用的是JSONP。

\*  JSONP（JSON with Padding）可用于解决主流浏览器的跨域数据访问的问题。跟JSON没有关系。 这是一种跨域的技巧。

\*  JSONP是如何实现跨域访问的？本质上是利用HTML元素的src属性都可以跨域的思路来解决的。 因为src是可以跨域的。

如：img，script，iframe等标记的src属性的值都可以赋成其它域名的合法地址。

<!DOCTYPE html>

<html>

    <head>

        <meta charset="utf-8">

        <title></title>

    </head>

    <body>

        <!-- 但是能通过src跨域访问 -->

        <img src= "<http://192.168.0.102/myguayu/img/1.jpg>">

    </body>

</html>

<script src="js/ajaxjs.js" type="text/javascript" charset="utf-8"></script>

<script>

    function test(value){

        console.log(value);

    }

</script>

<script src="[http://192.168.0.102/myguayu/test.php"/](http://192.168.0.102/myguayu/test.php)></script>

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

<script src="text.txt"></script>

<script src="my.php?cb=test"/></script>

<?php

    $cb = $\_GET["cb"];

    echo "$cb(123)";

?>

\* 面试题：请问jsonp是不是ajax中实现跨域访问的技术

\* jsonp不是AJAX中实现跨域访问的技术

1、jsonp没有使用XMLHttpRequest对象。

2、jsonp只是在一种跨域的技巧。

3、jsonp只支持Get方式

由于按照jsonp的这种方式跨域访问时，好像可以利用javascript和服务器端交互，能达到AJAX中XMLHttpRequest对象同样的效果。所以，人们总是把

jsonp和AJAX联系在一起。

\* 面试题：jsonp和json的区别？

1、jsonp和json根本就没有关系

2、jsonp是跨域访问的技巧

3、json是描述数据的格式

15.jsonp访问百度搜索引擎

各大搜索引擎接:https://www.cnblogs.com/woider/p/5805248.html

<!DOCTYPE html>

<html>

    <head>

        <meta charset="utf-8">

        <title></title>

    </head>

    <body>

        <input type="text" name="" id="wd" value="" />

        <ul id="list">

        </ul>

    </body>

</html>

<script>

    let oWd = document.querySelector("#wd");

    let oUl = document.querySelector("#list");

    oWd.oninput = function(){

        let scrpit = document.createElement("script");

        scrpit.src = "http://suggestion.baidu.com/su?wd="+this.value+"&cb=fun";

        document.body.appendChild(scrpit);

    }

    function fun(str){

        let strHtml = "";

        for(let i=0; i<str.s.length; i++){

            strHtml += "<li>" +str.s[i]+ "</li>";

        }

        oUl.innerHTML = strHtml;

    }

</script>

0.整理笔记,课堂案例

1. 掌握回调函数的应用

2. 掌握Promise规范

3. 优化ajax封装

4. 跨域访问百度，实现下拉菜单