

# AJAX和Promise

## 回调函数

概念：将一个函数当做参数传入另一个函数的时候，这个函数就叫回调函数。

我们之前用过很多次回调函数，比如：数组方法map、filter等；运动函数中处理运动结束传入的函数；分页插件中使用插件的时候执行的函数。。。包括封装的ajax中，请求成功以后执行的success函数。都是回调函数。

为什么要使用回调函数？

当我们执行一些异步操作的时候，需要在操作完成以后，做另外的一些事情，但是我们又没有办法预知这个异步操作什么时候结束，此时只能使用回调函数的形式来解决这个问题。

## 回调地狱

我们在封装ajax的时候，发现：请求成功后的值不能直接返回给调用者，而需要在其内部执行一个回调函数。如果在请求一次后需要再次请求，那么，也就是在回调函数中需要再次调用ajax，再次传入回调函数，次数多了以后，代码是下面这个样子：

### 回调地狱



```
function register()
{
  if (!empty($_POST)) {
    $msg = '';
    if ($_POST['user_name']) {
      if ($_POST['user_password_new']) {
        if ($_POST['user_password_new'] === $_POST['user_password_confirm']) {
          if (strlen($_POST['user_password_new']) >= 6) {
            if (strlen($_POST['user_name']) <= 64 && strlen($_POST['user_name']) >= 3) {
              if (preg_match('/^[a-z\d]{3,64}$/i', $_POST['user_name'])) {
                $user = read_user($_POST['user_name']);
                if (!isset($user['user_name'])) {
                  if ($_POST['user_email']) {
                    if (strlen($_POST['user_email']) <= 65) {
                      if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                        create_user();
                        $_SESSION['msg'] = 'You are now registered so please login';
                        header('location: ' . $_SERVER['PHP_SELF']);
                        exit();
                      } else $msg = 'You must provide a valid email address';
                    } else $msg = 'Email must be less than 64 characters';
                  } else $msg = 'Email cannot be empty';
                } else $msg = 'Username already exists';
              } else $msg = 'Username must be only a-z, A-Z, 0-9';
            } else $msg = 'Username must be between 3 and 64 characters';
          } else $msg = 'Password must be at least 6 characters';
        } else $msg = 'Passwords do not match';
      } else $msg = 'Empty Password';
    } else $msg = 'Empty Username';
    $_SESSION['msg'] = $msg;
  }
  return register_form();
}
```

这样的代码难以阅读和维护，这是著名的“回调地狱”。

例：

```
// 最后结果：安徽 - 合肥 - 包河区
var str = '';
var pid = 1;
sendAjax({
  url:"10-demo.php?pid="+pid,
  success:function(res){
    for(var i=0;i<res.length;i++){
      if(res[i].REGION_NAME == "安徽省"){
        str += res[i].REGION_NAME;
```

```

        pid = res[i].REGION_ID;
    }
}
// console.log(res);
// console.log(str);
str += '-';
// 发送ajax得到所有的市
sendAjax({
    url: "10-demo.php?pid="+pid,
    success: function(res){
        // console.log(res);
        for(var i=0; i<res.length; i++){
            if(res[i].REGION_NAME == "合肥市"){
                str += res[i].REGION_NAME;
                pid = res[i].REGION_ID;
            }
        }
        // console.log(str);
        str += '-';
        sendAjax({
            url: "10-demo.php?pid="+pid,
            success: function(res){
                // console.log(res);
                for(var i=0; i<res.length; i++){
                    if(res[i].REGION_NAME == "包河区"){
                        str += res[i].REGION_NAME;
                        pid = res[i].REGION_ID;
                    }
                }
                console.log(str);
            }
        });
    }
});
}
});
}
});

```

这种嵌套多了以后，就会形成回调地狱。

这个问题必须解决，所以es6中提供了一个构造函数promise，可以解决这个问题。

## Promise

promise是承诺的意思，表示他承诺帮你做这件事情，然后将结果给你。

语法：

```

new Promise(function (resolve, reject) {
  // resolve 表示成功的回调
  // reject 表示失败的回调
}).then(function (res) {
  // 成功的函数
}).catch(function (err) {
  // 失败的函数
})

```

在promise构造函数中，提供了两个参数，分别表示执行成功和失败的回调函数，执行成功调用resolve，失败调用reject即可，具体resolve和reject的执行，分别在then和catch中。

这样可以将回调函数变成链式结构，从而解决了回调地狱的问题。

例：

```

var str = '';
var pid = 1;
new Promise(function(resolve, reject){
  sendAjax({
    url: "10-demo.php?pid="+pid,
    success: function(res){
      resolve(res);
    }
  });
}).then(function(res){
  // console.log(res);
  for(var i=0; i<res.length; i++){
    if(res[i].REGION_NAME == "安徽省"){
      str += res[i].REGION_NAME;
      pid = res[i].REGION_ID;
    }
  }
  // console.log(str);
  str += '-';
  return new Promise(function(resolve, reject){
    sendAjax({
      url: "10-demo.php?pid="+pid,
      success: function(res){
        resolve(res);
      }
    });
  });
}).then(function(res){
  for(var i=0; i<res.length; i++){
    if(res[i].REGION_NAME == "合肥市"){
      str += res[i].REGION_NAME;
      pid = res[i].REGION_ID;
    }
  }
  // console.log(str);
  str += '-';
  return new Promise(function(resolve, reject){

```

```

        sendAjax({
            url: "10-demo.php?pid="+pid,
            success: function(res){
                resolve(res);
            }
        });
    })
}).then(function(res){
    for(var i=0; i<res.length; i++){
        if(res[i].REGION_NAME == "包河区"){
            str += res[i].REGION_NAME;
            pid = res[i].REGION_ID;
        }
    }

    console.log(str);
}).catch(function(){
    console.log("出错了");
});

```

使用说明：

1. promise的then可以直接写在对象后面
2. 如果在promise的then里面返回一个promise对象，那么里面的promise的then可以跟在外面的promise的then后面

补充：then和catch不会同时触发，也就是说，只要一个then出错了，执行最底下的catch就行，所以也就可以连续写多个then，一个catch就行。

promise成功解决了回调地狱的问题，但是这对于我们一个追求完美的攻城狮来说，远远不够，我们希望可以再次优化。

将ajax封装到promise中：

```

function PAjax(obj){
    return new Promise(function(resolve, reject){
        sendAjax({
            url: obj.url,
            data: obj.data,
            method: obj.method,
            dataType: obj.dataType,
            async: obj.async,
            success: function(res){
                resolve(res);
            },
            error: function(){
                reject();
            }
        })
    });
}

```

调用方式：

```

var str = '';
var pid = 1;
PAjax({
    url: "10-demo.php?pid="+pid
}).then(function(res){
    // console.log(res);
    for(var i=0;i<res.length;i++){
        if(res[i].REGION_NAME == "安徽省"){
            str += res[i].REGION_NAME;
            pid = res[i].REGION_ID;
        }
    }
    // console.log(str);
    str += '-';
    PAjax({
        url: "10-demo.php?pid="+pid
    });
}).then(function(res){
    for(var i=0;i<res.length;i++){
        if(res[i].REGION_NAME == "合肥市"){
            str += res[i].REGION_NAME;
            pid = res[i].REGION_ID;
        }
    }
    // console.log(str);
    str += '-';
    PAjax({
        url: "10-demo.php?pid="+pid
    });
}).then(function(res){
    for(var i=0;i<res.length;i++){
        if(res[i].REGION_NAME == "包河区"){
            str += res[i].REGION_NAME;
            pid = res[i].REGION_ID;
        }
    }
    console.log(str);
}).catch(function(){
    console.log("请求错误");
});

```

## ASYNCAWAIT

es7提供了async/await来编写异步代码，是回调地狱的终极解决方案。

他可以将异步代码写的和同步代码一样。

语法：

```

async function fn() {
    const res = await promise对象
}

```

只要是一个 promise 对象，那么我们就可以使用 `async/await` 来书写

例：

```
async function fn(){
  var res = await new Promise(function(resolve,reject){
    sendAjax({
      url:"2-demo.php?pid=1",
      success:function(res){
        resolve(res);
      }
    });
  });
  var str = '';
  var pid;
  for(var i=0;i<res.length;i++){
    if(res[i].name == "安徽省"){
      str += res[i].en;
      pid = res[i].id;
    }
  }
  str += "-";

  var res = await new Promise(function(resolve,reject){
    sendAjax({
      url:"2-demo.php?pid="+pid,
      success:function(res){
        resolve(res);
      }
    });
  });

  for(var i=0;i<res.length;i++){
    if(res[i].name == "合肥市"){
      str += res[i].en;
      pid = res[i].id;
    }
  }
  str+="-"
  var res = await new Promise(function(resolve,reject){
    sendAjax({
      url:"2-demo.php?pid="+pid,
      success:function(res){
        resolve(res);
      }
    });
  });

  for(var i=0;i<res.length;i++){
    if(res[i].name == "包河区"){
      str += res[i].en;
    }
  }
}
```

```
    console.log(str);
}
fn();
```

使用说明：

1. async修饰的函数，需要调用的话，就正常调用
2. await必须包在async修饰的函数中

将async和封装的promise结合起来：

```
async function fn(){
    var res = await PAjax({
        url:"2-demo.php?pid=1"
    });
    var str = '';
    var pid;
    for(var i=0;i<res.length;i++){
        if(res[i].name == "安徽省"){
            str += res[i].en;
            pid = res[i].id;
        }
    }
    str += "-";
    var res = await PAjax({
        url:"2-demo.php?pid="+pid
    });
    for(var i=0;i<res.length;i++){
        if(res[i].name == "合肥市"){
            str += res[i].en;
            pid = res[i].id;
        }
    }
    str+="-"
    var res = await PAjax({
        url:"2-demo.php?pid="+pid
    });
    for(var i=0;i<res.length;i++){
        if(res[i].name == "包河区"){
            str += res[i].en;
        }
    }
    console.log(str);
}
fn();
```

## 跨域

正常情况下，我们使用ajax请求的数据都在自己的服务器上。但在一些特定的场景中，我们需要获取到别人的服务器上的数据，也就是在自己的服务器中的ajax要请求到别人的服务器的网址，这就是跨域。但是浏览器是不允许这样操作的，因为浏览器有同源策略。

同源策略：所谓同源，就是指域名、协议、端口都相同。比如说：在自己的localhost域名下请求[www.baidu.com](http://www.baidu.com)下的内容，这样的协议首先就不同，自己的是http，百度的是https，所以会被同源策略限制。

如果解决跨域？

#### 1. 使用php做代理

也就是说跨域请求只是限制客户端向服务端，如果是服务端向服务端请求的话就不存在这个问题，也就是说需要跨域的请求交给php服务端来做，有了结果再响应给ajax即可。

原理上利用的php的爬虫技术。有 `file_get_contents()`、`curl`、`ob_get_contents()`

#### 2. 在服务端设置响应头，允许跨域请求

如果请求的服务端是自己可操作的话，可以在php端设置允许跨域的响应头。代码如下：

```
header("Access-Control-Allow-Origin:");
```

#### 3. 使用服务器代理(nginx)

在Nginx配置文件中配置代理，具体配置如下：

```
location = 自定义url {  
    proxy_pass 待跨域请求的地址  
}
```

#### 4. 通过jsonp来实现

利用标签可以跨域（当前网页的图片链接可以是别的网站上的图片）的特性，制作标签进行跨域js代码：

```
var script=document.createElement('script');  
script.setAttribute('src',"http://www.php.com/test.php?callback=response");  
$('head').append(script);  
function response(res){  
    $('#result').text(res);  
    script.parentNode.removeChild(script); // 执行完成后将这个标签删掉  
}
```

php代码：

```
$fun=$_GET['callback'];  
echo "$fun($.$.str)";
```