回调函数原理：

我现在出发，到了通知你”  
这是一个异步的流程，“我出发”这个过程中（函数执行），“你”可以去做任何事，“到了”（函数执行完毕）“通知你”（回调）进行之后的流程

1.基本方法

<script language="javascript" type="text/javascript">

function doSomething(callback) {

// …

// Call the callback

callback('stuff', 'goes', 'here');

}

function foo(a, b, c) {

// I'm the callback

alert(a + " " + b + " " + c);

}

doSomething(foo);

</script>

1.匿名函数方法

<script language="javascript" type="text/javascript">

 function dosomething(damsg, callback){

  alert(damsg);

  if(typeof callback == "function")

  callback(); //传入一个参数，这个时候才开始调用

 }

dosomething("回调函数", function(){

  alert("和 jQuery 的 callbacks 形式一样!");

 });

</script>

2.高级方法

使用javascript的call方法

<script language="javascript" type="text/javascript">

function Thing(name) {

this.name = name;

}

Thing.prototype.doSomething = function(callback) {

// Call our callback, but using our own instance as the context

callback.call(this);

}

function foo() {

alert(this.name);

}

var t = new Thing('Joe');

t.doSomething(foo); // Alerts "Joe" via `foo`

</script>

传参数

<script language="javascript" type="text/javascript">

function Thing(name) {

this.name = name;

}

Thing.prototype.doSomething = function(callback, salutation) {

// Call our callback, but using our own instance as the context

callback.call(this, salutation);

}

function foo(salutation) {

alert(salutation + " " + this.name);

}

var t = new Thing('Joe');

t.doSomething(foo, 'Hi'); // Alerts "Hi Joe" via `foo`

</script>

使用 javascript 的 apply 传参数

<script language="javascript" type="text/javascript">

function Thing(name) {

this.name = name;

}

Thing.prototype.doSomething = function(callback) {

// Call our callback, but using our own instance as the context

callback.apply(this, ['Hi', 3, 2, 1]);

}

function foo(salutation, three, two, one) {

alert(salutation + " " + this.name + " – " + three + " " + two + " " + one);

}

var t = new Thing('Joe');

t.doSomething(foo); // Alerts "Hi Joe – 3 2 1" via `foo`

</script>

//模拟查找页面中的dom节点，将查找到的节点存在数组里面统一返回

  //此函数只用于查找不对dom节点做任何的逻辑处理

  var findNodes = function(){

   var i = 100000;//大量的循环，

   var nodes = [];//用于存储找到的dom节点

   var found;

   while(i){

    i -=1;

    nodes.push(found);

   }

   return nodes;

  }

  //将查找找到的dom节点全部隐藏

  var hide = function(nodes){

   var i = 0,

    max = nodes.length;

   for(;i<max;i++){

//findNodes后面有括号代表立即执行，先执行findNodes()然后执行hide()< hide(findNodes()); 执行函数 } ;

nodes[i].style.display="none"

}

上面的方法是低效的，以为hide()必须再次遍历有findNodes()返回的数组节点，如何避免这种多余的循环呢。

  我们不能直接在findNodes中对查询到的节点进行隐藏（这样检索就可修改逻辑耦合了），那么他就不再是一个通用函数了。

  解决方法是用回调模式，可以将节点隐藏逻辑以回调函数方式传递给findNodes()并委托其执行

//重构findNodes以接受一个回调函数

   var findNodes = fucntion(callback){

    var i = 100000,

     nodes = [],

     found;

    //检查回调函数是否可用调用的

    if(typeof callback !== 'function'){

     callback = false;

    }

    while(i){

     i -= 1;

     if(callback){

      callback(found);

     }

     nodes.push(found);

    }

    return nodes;

   }

   //回调函数

   var hide = function(node){

    node.style.display = 'none ';

   }

   //找到后续节点并在后续执行中对其进行隐藏

 findNodes(hide);//先执行findNodes然后执行hide，当然回调函数也可以在调用主函数时创建：findNodes(function(node){node.style.display = 'none';});