回调函数的基本实现：

// 声明一个函数，它的参数是一个函数fn

function example(fn) {

// 回调前，可以do something

alert('我是回调前执行的代码');

// 存在fn则直接用，不存fn在则用函数匿名表达式的方式声明一个

fn = fn || function () {};

fn(); // 调用传进来的fn

}

// 调用传进来的fn

function callback () {

alert ('I am callback!')

}

example(callback); // 调用函数a

这个例子展示回调函数基本实现方法，实现回调的关键是把***一个函数当成另一个参数传入函数当中***。当然实践中很少用上面显示定义一个函数，再作为参数传入方法的方式去使用回调函数。为了方便，***一般直接把一个匿名函数传入当成回调函数***，这种方法在Javascript中使用非常广泛，下面是一个基本的例子。

// 声明一个函数，它的参数是一个函数fn

function example(fn) {

// 回调前，可以do something

alert('我是回调前执行的代码');

// 存在fn则直接用，不存fn在则用函数表达式的方式声明一个

fn = fn || function () {};

fn(); // 调用传进来的fn

}

// 把匿名函数当做参数

example (function () {

alert ('I am callback!')

});

使用匿名函数作为参数，不仅可以极少代码，也让代码更好管理，更加灵活。

另外回调函数经常使用带有参数的匿名函数：

例如Javascript中的数组遍历forEach方法

var friends = ["Mike", "Stacy", "Andy", "Rick"];

friends.forEach(function (eachElement, index){

console.log (index + 1 + ". " + eachElement); // 1. Mike, 2. Stacy, 3. Andy, 4. Rick

});

这里我们传入匿名函数并且此匿名函数带有参数。我们根据文档规定知道第一个参数将会是数组中元素，第二个参数将会是元素下标，根据这样的参数排列我们定义匿名函数。那为什么参数会如此的排列？ 原因是我们定义的匿名函数将作为参数传入Javascript的forEach API中。forEach API将执行自己代码逻辑后，调用我们的匿名参数。并且在Javascript的forEach API中调用我们传入匿名函数时，传入给匿名函数的参数排列便是第一个参数是数组中元素，第二个参数将会是元素下标。

所以forEach的Javascript文档将告诉我们，定义传入的匿名函数第一，第二个参数将代表什么含义。真正参数的传入顺序，是在forEach API函数中，调用我们传入的匿名函数时候规定的。