自执行函数

一、 基础概念：定义函数的方式

一般定义函数有两种方式：

1、 函数的声明

下面的代码就是函数声明的机构：

```
function sum(a,b){

alert(a+b);   }

sum(1,2)
```

关于函数声明，它最重要的一个特征就是函数声明提前，意思是执行代码之前先读取函数声明。这意味着可以把函数声明放在调用它的语句之后。如下代码可以正确执行：

```
sum(1, 2)

function sum(a, b) {

 alert(a + b); }
```

2、 函数表达式

函数表达式中有几种不同的语法。最常见和最具代表性的一种如下代码所示：

```
var sum = function(a,b) {

alert(a+b);

sum(1, 2);
```

这种形式看起来好像是常规的变量赋值语句。但是函数表达式和函数声明的区别在于，函数表达式在使用前必须先赋值。

可以看下面的例子，如果顺序调换了，代码执行的时候就会出错。

```
sum(1, 2);//打印结果是 undefined

    var sum = function(a,b) {

        alert(a+b);

    };
```

造成这种现象是因为在解析代码时，浏览器会解析函数的声明，并把起放在执行任何代码的前面；至于函数表达式，则必须等到执行它的所在的代码时，才会真正的被解析。（涉及到变量的预解析问题，后面的进阶路径或者 JS 进阶课程会有讲解，感兴趣的也可以网上查阅资料学习下）

函数表达式中，创建的函数叫做匿名函数，因为 function 关键字后面没有标识符。

既然说到了匿名函数，我们就来简单谈谈匿名函数

匿名函数，顾名思义就是没有名字的函数。上面的函数表达式中的创建，即创建一个匿名函数，并将匿名函数赋值给变量 sum，用 sum 来进行函数的调用，调用的方式就是在变量 sum 后面加上一对括号()，如果有参数传入的话就是 sum(1,2)，这就是匿名函数的一种调用方式。

还有一种匿名函数的调用方式是：使用()将匿名函数括起来，然后后面再加一对小括号

```
alert((function(a, b) { return a + b; })(2, 3)); //5
```

二、 自执行函数

有上面对于函数和匿名函数的了解，我们引出来了一个概念——自执行函数。

创建了一个匿名的函数 ,并立即执行它 ,由于外部无法引用它内部的变量，因此在执行完后很快就会被释放 ,关键是这种机制不会污染全局对象。(这个知识点涉及到了 JS 变量的作用域 ,会在后面的进阶路径或者 JS 进阶课程中有讲到，这里大家可以先做了解)

自执行函数 , 即定义和调用合为一体，下面我们来看下一下自执行函数的一些表达方式

```
 (function() {alert("123")}()); //  推荐使用这个
(function() {alert("123")})(); //  但是这个也是可以用的
```