Chapter 3. This

Definitions
定义

this值是执行上下文一属性

```
activeExecutionContext = {
  VO: {...},
  this: thisValue
};
```

this和上下文的执行代码密切相关。它的值在**进入上下文时被定义**，在之后的代码运行阶段是不可改变的。

This value in the global code
全局代码中的this

在全局代码中，this值总是指向**全局对象本身**。因此可以直接引用它(全局变量)。

This value in the function code
方法代码中的this

this并不是**静态绑定到当前方法**，这是它的第一个(也是最核心的)特性。

this在**进入上下文时就被定义**，同时会根据不同的方法代码而被赋予不同的值。

在代码运行时阶段，**this值是不可变的**，也就是说我们不可能给this赋予新值，因为它不是变量。

那是什么影响this值的变化呢？这有很多因素。

首先，在普通的方法调用中，this由触发当前上下文代码的**caller**提供，换句话说，就是**调用当前方法的父级上下文**。
this值由调用表达式的格式定义（换句话说就是**函数被调用的语法**）

什么方式影响了被调用上下文的this值?

确切地讲是调用表达式的方式，即**调用函数的方式，影响了被调用上下文的this值**，仅此而已。

为了完全掌握this值的定义，有必要详细分析一基本类型 -- 引用类型

Reference type
引用类型

用为代码表示引用类型，它具有两种属性：base属性**所属的对象**和base的**属性名**

```
var valueOfReferenceType = {
  base: <base object>,
  propertyName: <property name>
};
```

标示符被**标示符解析过程**引用(处理)，

标示符是**变量名**，**方法名**，**方法参数名**和**全局对象的各种属性名**。例如以下标示符的值

```
var foo = 10;
function bar() {}
```

作为操作的**中间值**，**引用类型的值**类似于：

```
var fooReference = {
  base: global,
  propertyName: 'foo'
};

var barReference = {
  base: global,
  propertyName: 'bar'
};
```

为了从**引用类型中获取对象的真实值**，这里用伪代码GetValue方法表示：

```
function GetValue(value) {

  if (Type(value) != Reference) {
    return value;
  }

  var base = GetBase(value);

  if (base === null) {
    throw new ReferenceError;
  }

  return base.[[Get]](GetPropertyName(value));

}
```

方法上下文中this值定义的基本规则如下：

- 方法上下文的this值由**调用者**提供，由当前**调用方式**(函数被调用的书写格式)定义。
- 如果在调用括号语法左边是**Reference类型**，那么this值将把此**引用类型设为其base对象**。
- 其余情况(即其余任何非引用类型的情况)，this值总是设为**null**，因为this值不能被设置为null，所以间接以全局对象代替；

现在我们可以准确地说，为什么**以不同的代码方式**调用**相同的方法**，得到的this值不同 -- **因为中间的Reference值**

```
function foo() {
  alert(this);
}

foo(); // global, because

var fooReference = {
  base: global,
  propertyName: 'foo'
};

alert(foo === foo.prototype.constructor); // true

// another form of the call expression

foo.prototype.constructor(); // foo.prototype, because

var fooPrototypeConstructorReference = {
  base: foo.prototype,
  propertyName: 'constructor'
};
```

Function call and non-Reference type
non-Reference类型方法调用

我们之前提及过，如果括号调用表达式左边**不是Reference类型**，**而是其他类型**，那么this值将自动设置成null，最终设置成全局对象。

如下示例

```
(function () {
  alert(this); // null => global
})();
```

Reference type and null this value

Reference类型和null this值

下例中，括号表达式左边的调用语句**是Reference类型，但是this值却被设置为null**，即被重置为全局。这是因为Reference的base对象是AO

在内部方法被父级方法调用的情况下。正如第2章所述，本地变量，内部方法和型参都是存储在方法AO中的

```
function foo() {
  function bar() {
    alert(this); // global
  }
  bar(); // the same as AO.bar()
}
```

AO作为this值返回时通常为 -- **null**。因此又回到之前讨论的，this值又被设置为全局对象。

This value in function called as the constructor
构造函数的this值

在方法上下文中，关于this值还有一种情况 -- 当方法作为**构造函数调用**时。

```
function A() {
  alert(this); // newly created object, below - "a" object
  this.x = 10;
}

var a = new A();
alert(a.x); // 10
```

这种情况下，new操作符**调用A的内部构造函数[[Construct]]**,
反过来，在对象创建后**调用内部方法[[Call]]**,
同样是方法**A创建的新对象**做为对象的this值

Manual setting of this value for a function call
方法调用时，手动设置this值

```
var b = 10;

function a(c) {
  alert(this.b);
  alert(c);
}

a(20); // this === global, this.b == 10, c == 20

a.call({b: 20}, 30); // this === {b: 20}, this.b == 20, c == 30
a.apply({b: 30}, [40]) // this === {b: 30}, this.b == 30, c == 40
```