### this

#### 1.this的作用

this提供了一种更优雅的方式来“隐式”传递一个对象引用，因此可以将API设计的更加简洁并且易于复用。

#### 2.this的位置

调用位置就是函数在代码中被调用的位置，它决定了this的指向，调用栈就是为了到达当前执行位置所调用的所有函数。

从下面的代码可以看出，调用位置是在当前正在执行函数的前一个调用中。

     function baz(){

//当前调用位置是全局作用域

       console.log('baz');

       bar()

     }

     function bar(){

//当前调用位置在baz中

       console.log('bar');

       foo()

     }

     function foo(){

//当前调用位置在bar中

       console.log('foo');

     }

     baz()



#### 3.this的绑定

this是在运行时进行绑定的，并不是在编写时进行绑定的，他的上下文取决于函数调用时的各种条件，只取决于函数的调用方式。

    function foo(){

      var a = 2

      this.bar()

    }

    function bar(){

      console.log(this.a);

    }

    foo()    //a is not defind

亮明观点：this在任何情况下都不指向函数的词法作用域，作用域它存在于JavaScript引擎内部，无法通过代码访问它。

所以上面这段代码，试图用this联通foo()和bar()的作用域，让bar()可以访问foo()作用域里的变量a，这是不可能的。

通过以下四条规则来判断this的绑定对象：

（1）默认绑定

（2）由上下文对象调用，绑定到那个上下文对象.

（3）由call或者apply调用，绑定到指定的对象

（4）由new调用，绑定到新创建的对象。

#### 3.1默认绑定

采用独立函数进行调用，this默认绑定到全局对象。

    function foo(){

      console.log(this.a);

    }

    var a = 2

    foo() // 2

foo()直接使用不带任何修饰的函数引用进行调用，因此只能使用默认绑定。

#### 3.1隐式绑定

当函数引用有上下文对象时，隐式绑定规则会把函数调用中的this绑定到这个上下文对象。

  function foo(){

    console.log(this.a);

  }

  var obj = {

    a : 2,

    foo : foo

  }

  obj.foo()  // 2

调用位置会使用obj上下文来引用函数，当调用foo()时，this被绑定到obj，因此this.a和obj.a是一样的。

隐式丢失：有的被隐式绑定的函数会丢失绑定对象，此时会变为默认绑定，this绑定到全局对象或者undefined。

  // 隐式丢失

  function foo(){

    console.log(this.a);

  }

  var obj = {

    a : 2,

    foo : foo

  }

  var bar = obj.foo       // 函数别名

  var a = 'aaps,global'   // a是全局对象属性

  bar()                   // 'oops,global'

此段代码不同于隐式绑定就在于函数别名bar，实际上引用的是foo函数本身，此时bar()就是一个不带任何修饰的函数调用，this指向全局对象。

当进行参数传递时也是一种隐式赋值，相当于传递的foo函数本身，此时就是一个不带任何修饰的函数调用，所以this指向全局对象。

  function foo(){

    console.log(a);

  }

  function doFoo(fn){

    fn()

  }

  var obj = {

    a : 2,

    foo : foo

  }

  var a = 'aaps,global'   // a是全局对象属性

  doFoo(obj.foo)          // 'oops,global'

#### 3.1显式绑定

通过函数的call()和apply()方法，直接指定this的绑定对象。显示绑定仍然无法解决绑定丢失的问题。可以通过硬绑定解决。

  // 显示绑定

  function foo(){

    console.log(this.a);

  }

  var obj = {

    a : 2

  }

  foo.call(obj)      // 2

硬绑定：在函数内部手动进行显示绑定，此时会把this绑定到对象上。

ES5提供了内置的方法Function.prototype.bind来进行硬绑定。

  // 硬绑定

  function foo(something){

    console.log(this.a,something);

    return this.a + something

  }

  var obj = {

    a : 2

  }

  var bar = foo.bind(obj)

  var b = bar(3)              //2  3

  console.log(b)              //5

#### 3.4new绑定

构造函数：使用new操作符时被调用的函数，就是普通函数被new调用。

使用new进行构造函数调用时，会自动执行以下操作。

（1）创建一个全新的对象。

（2）新对象执行[[Prototype]]连接。

（3）新对象绑定到函数调用的this。

（4）如果函数没有返回其他对象，那么new表达式中的函数调用就会自动返回这个新对象

#### 3.5优先级

优先级从高到低：new绑定 > 显示绑定 > 隐式绑定 > 默认绑定

（1）new绑定：this绑定的是新创建的对象。

（2）显示绑定：this绑定的是指定的对象。

（3）隐式绑定：this绑定的是上下文对象。

（4）默认绑定：this绑定的是全局对象。

#### 4.箭头函数

箭头函数this的绑定规则比较特殊，不符合以上规则，他是根据外层（函数或者全局）作用域来决定this。

函数体内的this对象就是定义时所在的对象，而不是使用时所在的对象。并且绑定后无法修改。

  // 箭头函数

  function foo(){

    return(a) => {

      console.log(this.a);

    }

  }

  var obj1 = {

    a : 2

  }

  var obj2 = {

    a : 3

  }

  var bar = foo.call(obj1)

  bar.call(obj2)               // 2

foo()内部创建的箭头函数会捕获调用foo()的this，由于foo()的this绑定到obj1，bar的this也会绑定到obj1，所以输出2.