JavaScript 中 this 指向

对于 JavaScript 初学者来说，this 指针的指向问题一直是很混乱的问题。在不同的场景下，this 会化身不同的对象。有一种观点认为，只有正确掌握了 JavaScript 中的 this 关键字，才算是迈入了 JavaScript 这门语言的门槛。在主流的面向对象的语言中（例如 Java,C#等），this 含义是明确且具体的，即指向当前对象。一般在编译期绑定。而 JavaScript 中 this 在运行期进行绑定的，这是 JavaScript 中 this 关键字具备多重含义的本质原因。

随着函数使用场合的不同，this 的值会发生变化。但是有一个总的原则，那就是 **this 指的是，调用函数的那个对象**。

1、指向 window 的隐式指向

```
function sayHello(){
    this.name = "tsrot";
    console.log("hello " + this.name);
}
sayHello();       // hello tsrot
```

此时的变量 name 相当于 window.name，因为调用 sayHello 函数的对象为 window，相当于 window.sayHello()。和下面两种情况是一样的：

```
var name = "tsrot";
function sayHello(){
    console.log("hello " + this.name);
}
sayHello();       // hello tsrot
```

下面的 name 相当于在 window 下赋值了两次，谁后面执行就取谁。如果把 var name = “xieliqun” 放到 sayHello（）后面，此时输出的就是 hello xieliqun。

```
var name = "xieliqun";
function sayHello(){
    this.name = "tsrot";
}
sayHello();
console.log("hello " + this.name);    // hello tsrot
```

注：当函数在 DOM 上调用时，未传入 this，此时，this 也指向 window。当传入 this 时，this 指向当前 DOM input，例如下面情况：

```
<input type="button" value="click me" onclick="sayHello()">
<script>
    function sayHello(tagName){
        console.log("hello " + this.tagName);   //hello undefined
    }
</script>
```

```
<input type="button" value="click me" onclick="sayHello(this.tagName)">
<script>
    function sayHello(tagName){
        console.log("hello " + tagName);   //hello INPUT
    }
</script>
```

2、指向当前对象的显式调用

（1）、作为对象方法的调用

函数还可以作为某个对象的方法调用，这时 this 就指这个上级对象。

```
function sayHello(){
    console.log("hello " + this.name);
}
var o = {};
o.name = "tsrot";
o.hello = sayHello;
o.hello();          //hello tsrot
```

（2）、作为构造函数调用

所谓构造函数，就是通过这个函数生成一个新对象（object）。这时，this 就指这个新对象。

```
function sayHello(){
    this.name = "tsrot";
}
var o = new sayHello();
console.log("hello " + o.name); //hello tsrot
```

3、指向当前对象的隐式调用

（1） call、apply 调用

call（）、apply()都是函数对象的一个方法，它们的作用是改变函数的调用对象，它们的第一个参数就表示改变后的调用这个函数的对象。因此，this 指的就是这第一个参数。

```
function sayHello(){
    console.log("hello " + this.name);
}
var o = {};
o.name = "tsrot";
o.hello = sayHello;
o.hello.apply();          //hello
```

apply()的参数为空时，默认调用全局对象。因此，这时的运行结果为 0，证明 this 指的是全局对象。
如果把最后一行代码修改为

```
// o.hello.apply();          //hello
o.hello.apply(o);            //hello tsrot
```

此时 this 就指向对象 o 了。

（2） 原生 Function 方法 bind（）调用

很多人不知道原生 js 也有 bind（）方法，一直以为 bind（）的方法只有 jQuery 有，我也是最近看别人博客知道的。其实原生的 bind 和 jQuery 的 bind 是不同的，原生的 bind 相当于 apply 和 call。

```
var person = {
    name:"tsrot",
    sayHello:function(){
        console.log("你好，我是"+this.name);
    }
}
var boundFunc = person.sayHello.bind(person,person.sayHello);
setTimeout(boundFunc,5000);    //5秒后输出 你好，我是tsrot
```

下图代码中 person.sayHello,相当于在 window.person.sayHello，所以 this 指向 window。

```
var person = {
    name:"tsrot",
    sayHello:function(){
        console.log("你好，我是"+this.name);
    }
}
// var boundFunc = person.sayHello.bind(person,person.sayHello);
setTimeout(person.sayHello,5000);    //5秒后输出 你好，我是
```

用 apply 和 call 调用时，函数将立即执行

```
var person = {
    name:"tsrot",
    sayHello:function(){
        console.log("你好，我是"+this.name);
    }
}
var boundFunc = person.sayHello.apply(person,person.sayHello);
setTimeout(boundFunc,5000);    //立即输出 你好，我是tsrot
```

4、当 this 在构造函数有 return 时

如果返回值是一个对象，那么 this 指向的就是那个返回的对象，如果返回值不是一个对象那么 this 还是指向函数的实例。

```
function fn(){
    this.name = "tsrot";
    return {};
}
var o = new fn;
console.log(o.name); //undefined
```

```
function fn(){
    this.name = "tsrot";
    return function(){};
}
var o = new fn;
console.log(o.name); //undefined
```

当 return null 和 undefined 时

```
function fn(){
    this.name = "tsrot";
    return null;
}
var o = new fn;
console.log(o.name); //tsrot
```

```
function fn(){
    this.name = "tsrot";
    return undefined;
}
var o = new fn;
console.log(o.name); //tsrot
```