类数组对象

所谓的类数组对象:

> 拥有一个 length 属性和若干索引属性的对象

举个例子：

```
var array = ['name', 'age', 'sex'];

var arrayLike = {
    0: 'name',
    1: 'age',
    2: 'sex',
    length: 3
}
```

即便如此，为什么叫做类数组对象呢？

那让我们从读写、获取长度、遍历三个方面看看这两个对象。

读写

```
console.log(array[0]); // name
console.log(arrayLike[0]); // name

array[0] = 'new name';
arrayLike[0] = 'new name';
```

长度

```
console.log(array.length); // 3
console.log(arrayLike.length); // 3
```

遍历

```
for(var i = 0, len = array.length; i < len; i++) {
   ……
}
for(var i = 0, len = arrayLike.length; i < len; i++) {
    ……
}
```

是不是很像？

那类数组对象可以使用数组的方法吗？比如：

```
arrayLike.push('4');
```

然而上述代码会报错: `arrayLike.push is not a function`

所以`arrayList`终归还是类数组，并不存在数组的方法。

调用数组方法

如果类数组就是任性的想用数组的方法怎么办呢？

既然无法直接调用，我们可以用`Function.call`间接调用：

```
var arrayLike = {0: 'name', 1: 'age', 2: 'sex', length: 3 }

Array.prototype.join.call(arrayLike, '&'); // name&age&sex

Array.prototype.slice.call(arrayLike, 0); // ["name", "age", "sex"]

[].slice.call(arrayLike);//["name", "age", "sex"]

// slice可以做到类数组转数组
Array.prototype.map.call(arrayLike, function(item){
    return item.toUpperCase();// ["NAME", "AGE", "SEX"]
});
```

类数组转对象

在上面的例子中已经提到了一种类数组转数组的方法，再补充三个：

```
var arr = ["name","age","sex"];
console.log(Array.prototype.slice.call(arr),arr.slice());

var arrayLike = {0: 'name', 1: 'age', 2: 'sex', length: 3 }
// 1. slice
Array.prototype.slice.call(arrayLike); // ["name", "age", "sex"]
// 2. splice
Array.prototype.splice.call(arrayLike, 0); // ["name", "age", "sex"]
// 3. ES6 Array.from
Array.from(arrayLike); // ["name", "age", "sex"]
// 4. apply
Array.prototype.concat.apply([], arrayLike)
```

那么为什么会讲到类数组对象呢？以及类数组有什么应用吗？

要说到类数组对象，`Arguments`对象就是一个类数组对象。在客户端`JavaScript`中，一些`DOM`方法，例如`document.getElementsByTagName()`，也返回类数组对象。

`Array.prototype.slice.call(arrayLike)` 相当于把 `arrayLike[0]` ，`arrayLike[1]` ，`arrayLike[2]` 的值 `name` ，`age` ，`sex` 放到一个 `[]` ，然后返回。 `Array.prototype.slice.call(arr)` ，`arr.slice()` 效果一样

Arguments对象

接下来重点讲讲 `Arguments` 对象。

`Arguments` 对象只定义在函数体中，包括了函数的参数和其他属性。在函数体中，`arguments` 指代该函数的 `Arguments` 对象。

举个例子：

```
function foo(name, age, sex) {
    console.log(arguments);
}
foo('name', 'age', 'sex')
```

类数组对象01

length属性

`Arguments` 对象的 `length` 属性，表示实参的长度，举个例子：

```
function foo(a, b, c){
    console.log("实参的长度为：" + arguments.length);//1
    console.log(arguments);
}
console.log("形参的长度为：" + foo.length);//3
foo(1);
```

callee属性

`callee` 是 `arguments` 对象的一个属性。它可以用于引用该函数的函数体内当前正在执行的函数。这在函数的名称是未知时很有用，例如在没有名称的函数表达式 (也称为“匿名函数”)内。 （ ES5严格模式废弃的属性 ）

如下，一般在非严格模式下递归调用一般这样使用：

```
function f(num){
    if(num<=1){
        return 1;
    }else {
        return num * arguments.callee(num-1);
    }
}
console.log(f(4)); //24
```

但是如果代码是在严格模式下开发：

```
"use strict";
function f(num){
    if(num<=1){
        return 1;
    }else {
        return num * arguments.callee(num-1);
    }
}
console.log(f(4));
//Uncaught TypeError: 'caller', 'callee', and 'arguments' properties may not be accessed on
strict mode functions or the arguments objects for calls to them
```

在严格模式下不能通过脚本访问`arguments.callee`,访问这个属性会报错，那么可以使用命名函数表达式来达到相同的结果：

```
"use strict";
function f(num){
    if(num<=1){
        return 1;
    }else {
        return num * f(num-1);
    }
}
console.log(f(4)); //24
```