

# 为什么要引入对象和类？

- 代码组织方式更加符合人的思考习惯。
- 对象是属性的集合。
  - 属性指：属性名+属性值。
  - 属性名为字符串类型，属性值为非undefined的简单数据类型或者对象类型。

# 对象的创建与使用

- 使用对象的属性
- 使用对象的方法
- 对象的创建

# 对象属性的引用

- 方式1：使用属性操作符 (.) 后面加上属性名称
  - `university.Name=“广西”`
- 方式2：通过中括号内置属性名称字符串表达式，如果表达式结果非字符串类型，则执行自动类型转换，转换为字符串类型，这是因为对象的属性名必须是字符串类型：
  - `university["Name"]="广西"`
- 如果对象的属性值为`undefined`，则表示在对象这个属性集合中，该属性名不存在。

# 用哪个方式？

- 当属性名为一个变量或者是一个表达式，或者一个中文字符串时，用中括号方式来引用对象属性：

```
var obj = {  
    name: 42  
}
```

// 等价于

```
obj.name = 42 ;
```

// 等价于

```
var propertyName = "name" ;  
obj[propertyName] = 42
```

- 动态属性名的取值方法

```
> var obj= {name: "zhangsan",age:123}
< undefined
> obj
< Object {name: "zhangsan", age: 123}
> var prop = "name"
< undefined
> obj[prop]
< "zhangsan"
> obj.prop
< undefined
> obj[prop]
< "zhangsan"
> prop
< "name"
> obj["name"]
< "zhangsan"
> var f = function(){return "name";}
< undefined
> f
< function () {return "name";}
> f()
< "name"
> obj[f()]
< "zhangsan"
> obj["nam"+"e"]
< "zhangsan"
> var t = {prop: "name"}
< undefined
> t.prop
< "name"
> obj[t.prop]
< "zhangsan"
```

# 对象方法的引用

- 在JavaScript中对象方法的引用是非常简单的。  
    ObjectName.methods()  
    实际上methods方法实质上是一个函数。
- 如引用university对象中的showmy()方法，则可使用：  
    university.showmy()

# 有关对象操作语句

- for...in语句
- with语句
- this关键字
- new运算符

# 对象属性的枚举（遍历）

- 格式如下：

For（对象属性名 in 已知对象名）

说明：

- 1.该语句的功能是用于对已知对象的所有属性进行操作的控制循环。它是将一个已知对象的所有属性反复置给一个变量；而不是使用计数器来实现的。
- 2.该语句的优点就是无需知道对象中属性的个数即可进行操作。

- 例：下列函数是显示数组中的内容：

```
Function showData(object) {  
    for (var X=0; X<30;X++)  
        document.write(object[i]);  
}
```

该函数是通过数组下标顺序值，来访问每个对象的属性，使用这种方式首先必须知道数组的下标值，否则若超出范围，则就会发生错误。

而使for...in语句，则根本不需要知道对象属性的个数，见下：

```
Function showData(object) {  
    for (var prop in object) document.write(object[prop]);  
}
```

- 使用该函数时，在循环体中，For自动将属性取出来，直到最后为此。



# 遍历

```
obj = {}  
for(var i=0;i<100;i++){  
    obj["name"+i] = "我的ID是: "+i;  
}  
  
for(var propName in obj){  
    console.log("属性名: "+propName+" 属性值: "+obj[propName]);  
}
```

# with语句

- 使用该语句的意思是：在该语句体内，任何对变量的引用被认为是这个对象的属性，以节省一些代码。

```
with object{  
...}
```

所有在with语句后的花括号中的语句，都是在后面object对象的作用域的。

```
with(Math) {  
  document.write(cos(35));  
  document.write(cos(80));  
}
```

=  
等效

```
document.write(Math.cos(35));  
document.write(Math.cos(80));
```

# with语句

```
.  
> with(obj){  
    name1 = "ddd";  
    console.log(name1);  
}  
ddd  
⏏ undefined  
> obj.name1  
⏏ "ddd"  
> name1  
✖ ▶ ReferenceError: name1 is not defined  
.
```

# this关键字

```
var book = {  
  price: somePrice * discount,  
  pages: 500,  
  pricePerPage: this.price / this.pages  
};
```

不等于

```
var book = {  
  price: somePrice * discount,  
  pages: 500  
};  
book.pricePerPage = book.price / book.pages;  
// or book['pricePerPage'] = book.price / book.pages;
```

# this 和 arguments

- **this**是函数体中的缺省局部变量之一。另一个缺省局部变量是**arguments**。
- **this**根据函数不同的调用方式有不同的值（含义）。换句话说，**this**的值只在函数的被调用的时刻确定。
- **arguments**是指函数调用时传入的实参表，它也是只能在函数调用时刻才能确定。实参表是一个对象，可以像数组一样使用下标来访问各个实参，有**length**属性。但不是**Array**类型。

# 函数直接使用 vs new操作符一起使用

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

```
foo() // window  
new foo() // foo
```

# 函数作为方法使用

```
var object = {  
  foo: function(){  
    alert(this === object);  
  }  
};  
  
object.foo(); // true
```

# 另类的函数调用：call & apply

- “借尸还魂术”





```
function user(first, last, age){  
    // do something  
}  
user.call(window, 'John', 'Doe', 30);  
user.apply(window, ['John', 'Doe', 30]);
```



# 对象方法的重载（overload）

- 一个方法名，根据输入参数的数据类型不同，可有不同的功能。

jQuery对象的  
filter方法

 <b>.filter( selector )</b>	version added: 1.0
<b>selector</b> Type: <a href="#">Selector</a> A string containing a selector expression to match the current set of elements against.	
 <b>.filter( function(index) )</b>	version added: 1.0
<b>function(index)</b> Type: <a href="#">Function()</a> A function used as a test for each element in the set. <code>this</code> is the current DOM element.	
 <b>.filter( element )</b>	version added: 1.4
<b>element</b> Type: <a href="#">Element</a> An element to match the current set of elements against.	
 <b>.filter( jQuery object )</b>	version added: 1.4
<b>jQuery object</b> Type: <a href="#">Object</a> An existing jQuery object to match the current set of elements against.	

# 对象的创建

- 字面量方式
- 后定义方式
- 构造函数方式
- 类继承方式
- `Object.create()`方式

# 方式1

- 字面量方式:

```
var obj = {  
  sex : '男',  
  name : '张三',  
  speak : function() {  
    alert('我是'+this.name) ;  
  }  
}  
  
obj.speak() ;
```

## 方式2

- 后定义方式:

```
var zhangsan = {};  
zhangsan['sex'] = '男';  
zhangsan['name'] = '张三';  
zhangsan['speak'] = function() {  
    |   alert('我是'+this.name) ;  
    |  
    }  
  
zhangsan.speak() ;
```

# 方式3

- 构造器方式: new

```
function Person(name, sex) {  
    this.name = name;  
    this.sex = sex ;  
    this.speak = function() {  
        alert('我是'+this.name) ;  
    }  
}  
  
var mySon = new Person("张三", "男");  
mySon.speak() ;
```

# 方式4

- 继承方式：（好处：实现父亲共享）

```
function Circle(radius) {  
  |   this.radius = radius;  
  |  
  |}  
  
Circle.prototype.getCircumference = function() {  
  |   return Math.PI * 2 * this.radius;  
  |}  
};  
  
Circle.prototype.getArea = function() {  
  |   return Math.PI * this.radius * this.radius;  
  |}  
};  
  
var myCircle = new Circle(10) ;  
console.log(myCircle.getArea()) ;
```

前例中的Person仅仅是构造函数而已，并不是父亲！父亲是Person.prototype!

# 方式5

- Object.create方式:

语法: Object.create(proto [, propertiesObject ])

```
function Car (desc) {  
  this.desc = desc;  
  this.color = "red";  
}  
  
Car.prototype = {  
  getInfo: function() {  
    return 'A ' + this.color + ' ' + this.desc + ' .';  
  }  
};  
  
//instantiate object using the constructor function  
var car = Object.create(Car.prototype);  
car.color = "blue";  
alert(car.getInfo()); //displays 'A blue undefined.'
```

# 方式5

- `Object.create = function (o) {  
 var F = function () {};  
 F.prototype = o;  
 return new F();  
};`



# Call方法举例

- `[].forEach.call($$("*"),function(a){var b = "1px solid #" + ((Math.random()*(1<<24))-1).toString(16) ; b= b.substr(0,b.length-3); a.style.outline = b; })`