1. 在JS中，类的实现是基于原型继承机制的。一个很重要的特性是“动态可继承”
2. 原型对象才是类的唯一标识：当且仅当两个对象都继承自同一个原型对象时，它们才属于同一个类的实例。初始化的构造函数不能作为类的标识，因为有可能两个构造函数的prototype都指向同一个对象，那么这两个构造函数的实例都属于同一个类。
3. 每个JS函数，除了bind（）返回的函数外，其自动都有一个prototype属性，这个属性指向一个对象，这个对象包含一个不可枚举的属性constructor，constructor的值是该函数对象。所以这一切都是自动的。根本没必要将函数的prototype重写为一个对象，这是有点浪费。  
   new f();  
   function f(){}  
   f.prototype={“x”：1，constructor:f}//此时必须手动写上，其实没必要，函数本身就有prototype属性，且属性中constructor已自动设置好了，这样是在重新定义prototype这个属性。不如对这个prototype添加属性。如：  
   f.prototype.x=1;

当改写这个prototype整个对象时，即使构造函数不同，也可能是同一个类。

1. 类的扩充  
   动态可继承中的“动态”的意思是：对象从原型继承属性，如果创建对象之后原型发生了改变，也会影响到继承这个原型的所有实例对象。这就意味着我们可以通过给原型添加新方法来扩充类。
2. 类的检查
3. instanceOf  
   instanceOf左操作数是一个对象，右操作数实际是一个函数（函数也是个对象嘛），实际上就是在检查原型链。每一个函数除了bind（）方法返回的函数，都有一个prototype属性，而每一个对象都有\_proto\_(浏览器实现)属性，内置为[[prototype]]，指向其原型。其功能就相当于isPrototypeOf（）方法。缺点是：只能检查对象是否属于某个类，无法得到对象的类名。在客户端JS中，还有一个缺点，在多窗口的Web应用中表现不佳。因为多窗口下，每个窗口都包含独立的执行上下文，每个上下文都有独立的全局变量和一组构造函数。因此一个窗口中的数组实例是该Array类，在另一个窗口中就不是了。
4. constructor属性  
   构造函数是类的公共标识。缺点是：在多窗口的web应用中表现不佳。同时，并不是所有的对象都有constructor属性，因为存在一种情况你忘了在构造函数的prototype中写上。但是最后一定会有constructor属性，因为最终沿着原型链查找最后肯定会到Object。
5. 构造函数的名字  
   上述方法都会在多窗口中产生问题。解决方案是使用构造函数的名字，而不是构造函数本身。一个窗口中的Array和另一个窗口的Array构造函数是不一样的，但是构造函数的名称是一样的。函数对象有个name属性可以用。