ajax：技术要求：javascript+css+dom+XMLHttpRequest
特点：
1、减少后台访问量，减少http向服务器发送请求
2、能生成异步传输，不影响整个DOM渲染，可改变局部数据
3、ajax起中间介质作用
4、ajax是异步+url传值
创建ajax步骤
判断浏览器创建ajax对象的方式

```
varxmlHttp;
if(window..XMLHttpRequest){
  //(包括IE7/8/9)firefox chrome浏览器支持，IE6不支持
  alert("标准写法(仅IE6不支持)";
}
if(window.ActiveXObject){
  //IE所有浏览器支持
  alert("所有IE支持，IE6只支持之中写法");
}
```

创建Ajax对象—支持IE6/7/8/9/10 firefox chrome

```
var xmlHttp;
if(window.XMLHttpRequest){//标准创建方式
   xmlHttp=new XMLHttpRequest();
}else if(window.ActiveXObject){
   xmlHttp=newActiveXObject("Microsoft XMLHttp");
}
```

ajax对象的方法
open方法：用于设置进行一步请求的目标URL，请求方法及其他参数信息
语法：

```
open("method",url"[,asyncFlag]");
```

参数说明
method--方法get/post
url-请求地址，并可以传递查询字符串
asyncFlag-可选参数，用于指定请求方式，默认为true
当该boolean值为true时，服务器请求是异步进行的，也就会是脚本执行
send()方法后不等待服务器得执行结果，而是继续执行脚本代码
send()方法用于向服务器发送请求。如果请求声明为异步，该方法将立即返回，否则将直到接受到享受为止
当该boolean值为true时，服务器请求是异步进行的，也就是脚本执行send()方法后不等待服务器的执行结果，而是继续执行脚本代码
当该boolean值为false时服务器请求是同步进行的，也就是脚本执行send()方法后等待服务器的执行结果的返回，若在等待过程中超时，则不再等待，继续执行后面的脚本代码
send(参数)：参数用于指定发送的数据，可以是DOM对象的实例，输入流式字符串
eg：open("POST","index.php",true);

setRequestHeader方法：为请求的HTTP头设置值；
语法：
setRequestHeader("label","value");
实例：setRequestHeader("Content-type","application/x-www-form-urlencoded");
注意：setRequestHeader()方法必须在调用open方法会后才能调用
ajax对象事件onreadystatechange
例如：xmlHttp.onreadystatechange=函数名/匿名函数;
说明：事件处理函数中负责接收响应数据，并进行响应处理
当请求被发送到服务器时，我们需要执行一些基于响应的任务。
每当readyState属性改变时，就会触发onreadystatechange事件
readyState属性存有XMLHttpRequest的状态信息
readyState存储XMLHttpRequest的状态，从0到4发生变化
0：请求为初始化
1：服务器连接以建立，但是还没发送
2：请求以接受，正在处理中
3：请求处理中，通常响应中已有部分数据可用了，但服务器还没完成响应的完成
4：请求已完成，且响应已就绪，可以获取并使用服务器的响应了
status属性
200：“ok”
202：请求被接收，但未成功
400：错误的请求
404：未找到页面
500：服务器端错误
status属性详解：
1：信息响应类，表示接收到请求并且继续处理。（所有拉去的货，工厂还没有加工完毕）
2：处理成功响应类，表示动作被成功接收、理解和接收。（所有拉去的货，工厂设备不够，让其他工厂帮忙加工）
3：重定向响应类，伪类完成指定的动作，必须接收进一步处理（所有拉去的货，工厂设备不够，让其他工厂帮忙加工）
4：客户端错误，客户请求包含语法错误活接是不能正确执行（这十车货有质量问题，工厂不能正常加工）
5：服务端错误，服务器不能正确执行一个正确的请求（工厂在加工到一半过程中断电，不能继续加工）

ajax对象属性
responseText获取服务器端字符串形式数据（包括字符串形式json数据）
responseXML获取服务器端XML形式数据

ajax组成和工作原理
工作原理：异步和同步
ajax工作顺序

1、创建对象（用XMLHttpRequest，但是这个只兼容IE6）
2、打开连接（用open方法，必须设置方式/地址/同步异步）
3、设置头信息（用setRequestHeader）
4、设置响应事件（用onreadyStateChange）
5、send

————————————分割线————————————

作用域
通常来说有一段程序代码中使用的变量和函数
全局作用域有以下几种
最外层函数和在最外层函数外面定义的变量拥有全局作用于
未定义直接复制的变量自动声明为拥有全局作用域
所有window对象的属性拥有全局作用域
局部作用域

变量的作用域
前提：这里只全部都通过var创建的变量或对象
1.全局变量：函数外创建的变量

```
var x=10;
function test(){
    alert("全局变量在test函数中"+x);
    alert("局部变量在test函数中"+y);
    alert("局部变量在test函数中"+z);//报错z未定义
    function a(){
        var z=30;
        alert("局部变量在函数a中"+y);
    }
    a();
}
```

注：x是全局变量，整个作用域有效
y是局部变量在test函数中有效，在a函数中有效
z是局部变量只在a函数中有效

闭包：
指有权访问另一个函数作用域中的变量的函数。
创建闭包常见的方式，就是在一个函数内部创建另一个内部（私有）函数
闭包实例：
在函数外部直接调用局部变量x，报错

```
function test(){
    var x=10;
    return function(){
```

```
        return x;
      }
    }//调用局部变量x，报错未定义
    alert(x);
    //调用
    var a=test();
    alert(a());
```
说明：a实际上就是闭包匿名函数，函数test中的局部变量x一直保存在内存中，并没有在test调用后被自动清除

闭包2：

```
    var y;
    function test(){
      var x=10;
      y=function(){
        return x;
      }
    }//调用函数
    test();
    alert(y());
```
说明：y实际上就是闭包y函数。函数test中 的局部变量，
并没有在test函数调用后删除掉，x一直保留在内存中。
y函数依赖于test函数都一直保存在内存中，
不会被垃圾回收机制（garbage collection）回收

闭包3：

```
    function test(arg){
      var y=function(){
        return arg;
      }
      arg++;
      return y;
    }
    //调用
    var a=test(123);
    alert(a());
```
迭代器实例

```
  function setup(x){
    var i=0;
    return function(){
      return x[i++];//其中i不是一个活动对象，0是一直存在的，直到next方法调用完，
    }
  }
  var next=setup(["a","b","c"]);
  alert(next());
  alert(next());
  alert(next());
```

——————————分割线——————————————

面向对象

对象：

把对象定义为“属性的无序集合，每个属性存放一个原始值、对象或函数”。严格来说，意味着对象是无特定顺序的值的数组。

尽管ECMA如此定义对象，但它更通用的定义是基于代码的名词的表示

类：相当于对象的基本结构

每个对象都有类定义，可以把类看做对象的配方。类不仅要定义对象的接口（interface）（开发者访问的属性和方法），还要定义对象的内部工作（使属性和方法发挥作用的代码）。编译器和解释程序都根据类的说明构建对象

实例

程序使用类创建对象时，生成的对象叫做类的实例（instance）。对类生成的对象的个数的唯一限制来自于运行diamante的机器的的物理内存。每个实例的行为相同，但实例处理一组独立的数据。由类创建对象实例的过程叫做实例化（instantiation）

例：

```
var a=new Array(1,2,3,4);
```

其中a相对于数组来说就是实例化对象

面向对象特点

封装：把相关的信息（无论数据或方法）存储在对象中的能力

聚集：把一个对象存储在另一个对象内的能力

继承：由另一个类（或多个类）得来类的属性和方法的能力

多态：编写能以多种方法运行的函数或方法的能力

对象类型:

可以创建并使用的对象有三种：本地对象、内置对象和宿主对象

ECMA-262把本地对象（native object）定义为“独立于宿主环境的ECMAScript实现提供的对象”。简单来说，本地对象就是ECMA-262定义的类（引用类型）

Object、Function、Array、String、Boolean、Number、Date、RegExp、Errorr、EvalError、RangeError、ReferenceError、SyntaxError、TypeError、URLError

作用域

作用域值得是变量的适用范围

公用、私有和受保护作用域

传统定义：

共有作用域中的对象属性可以从对象外部访问，既开发者创建对象的

私有作用域中的属性只能在对象内部访问，既对于外部世界来说，这些属性不存在

受保护作用域也是用于定义私有的属性和方法，只是这些属性和方法还能被其子类访问

+

this的作用：代表的是当前方法使用的对象

定义类和对象

原始方式:没有在格式上和语法上实现完整的封装，未实现多态，占用缓存较多

因为对象的属性可以在对象创建后动态定义，所有许多开发者都在javascript最初引入时编写类似下面的代码

例：

```
var oCar=new Object;
oCar.color="blue";
oCar.doors=4;
oCar.mpg=25;
oCar.showColor=function(){
        alert(this.color);
}
```

工厂方式：实现完整封装，但要使用闭包，返回必须是整个对象，节省缓存

要解决该问题，开发者创造了能创建并返回特定类型的对象的工厂函数

例：

```
function createCar(){
  var oCar=new Object;
  oCar.color="blue";
  oCar.doors=4;
  oCar.mpg=25;
  oCar.showColor=function(){
        alert(this.color);
  }
  return oCar;
}
var newcar=createCar();
newcar.showColor;
```

构造函数方式：实现了封装、聚集

创建构造函数就像创建工厂函数一样

例：

```
function Car(sColor,iDoors,iMpg){
   this.color=sColor;
   this.doors=iDoors;
   this.mpg=iMpg;
   this.showColor=function(){
      alert(this.color);
   }
}
var cra1=new createCar("red","4");
car1.showColor();
var cra2=new createCar("blue","2");
car2.showColor();
alert(car1.color);
alert(car2.door);
```

补充：使用原始方式和工厂方式时，要想改变属性，就得改变原始值，不能存储，未实现多态

```
var oCar=new Object;
    oCar.color="red";
    oCar.door="4";
    oCar.info=function(){
        alert("颜色："+oCar.color+",车门："+oCar.door);
    }
    oCar.setInfo=function(c,d){
        oCar.door=c;
            oCar.door=d;
    }
    oCar.info();
    oCar.setInfo("green","2");
    oCar.info();//输出，[绿色，2]；会替换原来的属性值
```

原型对象与原型链

我们创建的每个函数都有一个原型属性（prototype），这个属性是一个指针指向一个对象，这个对象有自己的特定类型的共所有实例是用的属性和方法，那么prototype就是通过调用构造函数而创建的那个原型对象

理解：

（1）只要创建了一个新函数就会为该函数创建一个prototype属性，这个属性指向原型对象

（2）所有的原型对象都会自动获得一个constructor（构造函数）属性，这个属性包含指向prototype属性的指针

（3）构造函数方式就会有一个prototype，数组去重时，也是用到原型对象

定义类和对象的第四种方式：原型方式构造方法

```
function oCar(){};
    oCar.prototype.color="red";
    oCar.prototype.door="4";
    oCar.prototype.info=function(){
        alert(this.color+","this.door);
    }
    var car1=new oCar();
    var car2=new oCar();
    car1.info();
    car2.info();
```

原型方式构造方法出现的问题

能通过给构造函数传递参数来初始化属性的值

hasOwnProperty()判断对象是否含有其属性，通过原型链继承不算

in 判断对象是否有其属性，包括通过原型链继承的

原型与in的操作

例：

```
function oCar(){};
        oCar.prototype.color="red";
        oCar.prototype.door="4";
        oCar.prototype.info=function(){
            alert("颜色："+this.color+"，车门："+this.door);
```

```
        }
    var cra1=new oCar();
    var cra2=new oCar();
    alert(car1.hasOwnProperty("color"));//提示false，car1本身没有colo，是通过原型链得到的
        alert("color" in car1);//提示true
        car1.color="green";
        alert(car1.hasOwnProperty("color"));//提示true，因单独给car1复制了color设置了属性值
        alert(car2.color);//提示red，有通过原型链得到的属性
```

原型链

1、因为每个对象和原型都有原型，所以对象的原型指向原型对象 而父的原型又指向父的父，这种原型层层链接起来的就构成了原型链

2、_proto_是所有对象（包括函数）都有的，它才叫做对象的原型，原型链就是靠它形成的。用于指向创建它的函数对象的原型对象prototype

3、prototype只有函数（准确的说是构造函数）才有的，它跟原型链无关，它的作用为：构造函数new对象的时候，告诉构造函数新创建的对象原型是谁。是的，只在new一个对象的时候才起作用，当new得到这个对象后，无论怎么改构造函数的pototype属性，都不会影响已创建的对象原型链

例：

```
var Person=function(){};
var p=new Person();
alert(p.__proto__===Person.prototype);//返回值为true
```

例子解释：

每个对象都会在内部初始化一个__proto__属性，当我们访问一个对象的属性时，如果这个对象内部不存在这个属性，那么他就会去__proto__里找到这个属性，这个__proto__又会有自己的__proto__，于是就这样一直找下去，也就是我们平时所说的原型链概念。

按照标准，__proto__是不对外公开的，也就是说是个私有属性，（ie中对象的__ptoto__属性为undefined）但是firefox将它暴漏出为一个共有的属性，我们可以对外访问和设置

原型链实例

```
var Person=function(){};
    person.prototype.Say=function(){
        alert("Person say");
    }
    var p=new Person();
p.Say();
```

对象的继承

（1）所有开发者定义的类都可作为基类

（2）出于安全原因，本地类和宿主类不能作为基类，这样可以防止公用访问变异过得浏览器及的代码，因为这些代码可以被用于恶意攻击

（3）创建的子类将继承超累的所有属性和方法，包括构造函数及方法的实现

（4）所有属性和方法都是公用的，因此子类可直接访问这些方法。子类还可以添加超累中没有新属性和方法，也可以覆盖超累的属性和方法

继承是实现

继承方法

apply方法

语法：子类.apply(父类.new Array(子类参数))

call方法

语法：子类.call(父类,"参数","参数");

例：原型对象类型不能使用call继承参数

```
function ocar(colors,doors){
    this.color=colors;
    this.door=doors;
}
function newcar(a){
   alert(a+this.color+","+this.door)
}
newcar.call(oCar("red","3"),"车的信息：")
```

继承方式有三种

一：构造函数类型

二：call

三：apply

构造函数和原始类都可以用

原型链不能继承

————————————————分割线————————————————

跨域传值

同源策略：它是由Netscape提出的一个著名的安全策略：同域（或同源）指同一个协议、域名、端口

跨域：指在不同域里获得对应的数据

实现跨域

凡是有src属性的都可以实现跨域，例如：<script> <img> <iframe>

实现跨域有三种方式

第一种利用iframe实现

主要利用的是domain方式（很多浏览器都不支持）
location.search实现，也可以是ifram，例如面包屑导航
第二种是jsonp
第三种是利用后台代理（例如：案例跨域传值-服务器处理，php文件处理）

sonp
jsonp是一种非官方跨域数据交互协议
jsonp即json with padding。由于同源策略的限制，XmlHttpRequest只允许请求当前源（域名、协议、端口）的资源。如果要进行跨域请求，我们可以通过使用html的script标记来进行跨域请求，并在响应中返回要执行的script代码，其中可以直接使用javascirpt对象。这种跨域的通讯方式称为jsonp
jsonp原理
script标签src属性中的链接可以访问跨域的js脚本，利用这个特性，服务端不在返回json格式的数据，而是返回一段调用某个函数的js代码，在src中进行了调用，这样实现了跨域

<script src="jsonp.php?callback=jsonpCallback"></script>
其中：callback参数，为回掉函数
jsonpCallback为一个函数名称

jsonp执行过程
首先在客户端注册一个callback，然后把callback的名字穿个服务器
此时，服务器思安生成json数据。然后以javascript语法的方式，生成一个function，function名字就是传递上来的参数jsonp
最后将json数据直接以入参的方式，防止到function中，这样就生成了一段js文档，返回给客户端

json和jsonp的区别
jsonp和json没有任何关系
json是数据格式，用在同源异步请求的返回结果
jsonp是一种跨域请求方式，其原理就是动态生成script标签，设置src为远端地址，内容为一个js调用

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。它基于ECMAScript的一个子集。 JSON采用完全独立于语言的文本格式，但是也使用了类似于C语言家族的习惯（包括C、C++、C#、Java、JavaScript、Perl、Python等）。这些特性使JSON成为理想的数据交换语言。 易于人阅读和编写，同时也易于机器解析和生成(一般用于提升网络传输速率)。

JSONP(JSON with Padding)是JSON的一种“使用模式”，可用于解决主流浏览器的

跨域数据访问的问题。由于同源策略，一般来说位于 server1.example.com 的网页无法与不是 server1.example.com的服务器沟通

如果需要设置允许所有域名发起的跨域请求，可以使用通配符
header("Access-Control-Allow-Origin:*");