Javascript中object的使用

写在前面

> 本文涉及到一些javascript语法中的一些概念名词，如不好理解可以忽略，主要理解在工程中的用法

Javascript是ECMAScript规范的一种称呼，js语法即ECMAScript语法，目前版本已经到了es6，但是目前主流浏览器采用的是es5语法标准。Javascript主要有函数和对象两类组成，函数是一种对象，对象由函数构造，彼此依赖，函数中存在闭包，对象中存在原型链，这两类都在我们日常开发中占据重要地位。本次介绍object--对象的常规用法。

object的常规用法

object对象是js所有对象的根对象，所有对象都会继承object对象，跟java中的Object类一样。js中常用的Array对象也是继承自与object对象。

> 以下代码可以在浏览器中f12打开'控制台'，输入测试。

- 创建一个object对象：

```
var obj1={};
var obj2=new Object;
```

  以上两种方式都可以创建object对象，前者为后者的语法糖

- 向object中添加数据

```
obj1.name="zhang3";
obj1.age=23;

obj2["name"]="zhang3";
obj2["age"]=23;
```

  以上两种方式都可以向两个对象中通过key-value的形式添加数据,数据类型可以为string,boolean,number,function,object等类型。

- 也可以将json字符串转化成object

```
var objStr='{"name":"zhang3","age":23}';
var obj3=JSON.parse(objStr);
```

- 对象中key值遍历

```
for(var key in obj1){
    console.log(key+" = "+obj1[key]);
}
```

- object中添加Array

```
var les=["English","Math","Physics"]
obj1.lessons=les;
obj1.lessons.push("Chemistry");//继续推送课程
```

- object中添加object

```
var school={"name":"NO.3","address":"xxx"};
obj1.school=school;
//或者
obj1.school={};
obj1.school["name"]="NO.3";
obj1.school.address="xxx";
//注意引号和[]的配合使用
```

- 对象中对象的key值遍历

```
for(var key in obj1.school){
    console.log(key+" = "+obj1["school"][key]);
    console.log(key+" = "+obj1.school[key]);
}
```

- Object中添加function

```
obj1.getName=function(){
    console.log(this.name);//this指当前调用对象，该name可以理解为该对象环境下的全局变量
}
obj1.getName();//调用测试

obj1.getSchoolName=function(){
    console.log(this["school"].name);//this指当前调用对象,注意[]和引号的配合
}
obj1.getSchoolName();//调用测试
```

采用对象是可以更好按照功能分类组织管理函数，同时解决全局变量满天飘的问题以及因为全局变量重名出现的不应该的bug。

对象的原型链

```
obj1.toString(); //输出"[object Object]"
```

问题：obj1我们并没有创建toString方法，但是却可以输出东西，没有报错，说明该方法是存在于obj1中的。

原因：当obj1调用toString方法的时候，首先在obj1的属性中查找toString属性，当没有找到的时候，就会沿着该对象的原型链prototype进行查找toString方法，因为Object.prototype.toString方法是存在的。

关于js中**原型链**和**闭包**，属于js中的进阶部分，有兴趣的可以参看该篇博客。

对象的应用举例

举例描述：根据每个人的评级以及工资水平计算最后的奖金。

```
    var bonus={
        "S":function(sal){
                return sal*4;
            },
        "A":function(sal){
                return sal*3;
            },
        "B":function(sal){
                return sal*2;
            }
    };
    //定义数据源
    var person={
        "zhang3":{
            "leval":"S",
            "salary":5000
        },
        "li4":{
            "leval":"A",
            "salary":3000
        },
        "Jame":{
            "leval":"B",
            "salary":2000
        }
    };
    //计算
    function calculateBonus(person){
```

```
    for(var key in person){
        var lv=person[key].leval,
            sal=person[key].salary;
        var calSal=bonus[lv]&&bonus[lv](sal);//&&为管道符写法
        person[key].bonus=calSal;
        console.log(key+"'s bonus is "+calSal);
    }
};
//开始计算
calculateBonus(person);//计算每个人的红利，数据存储在person[人名].bunus中;
```

以上为js中代码的策略模式，该结构主要解决了switch-case使用的情况，并且支持扩展，例如扩展"C"级，或者删除"B"级的红利计算策略而不用添加case，代码入侵小,维护方便。这也是object带来的方便与好处。