20170802

罚抄

- Array中常用的方法
  - push：作用和意义
  - 参数
  - 返回值
  - 原有数组是否改变
- String
  - substr：作用
  - 参数
  - 返回值
- Math
- DOM
  - 获取元素的方法以及作用
  - 节点之间关系的属性
  - 增删改DOM的方法
- Date
- 数据类型转换
  罚抄完成，再次抽查提问，不会继续罚抄

```
//考试题-计算题六
function Fn(num) {
    this.x = this.y = num;
}
Fn.prototype = {
    x: 20,
    sum: function () {
        console.log(this.x + this.y);
    }
};
var f = new Fn(10);

console.log(f instanceof Fn);//->TRUE
console.log(f instanceof Object);//->TRUE

console.log(f.sum === Fn.prototype.sum);
f.sum();
Fn.prototype.sum();
console.log(f.constructor);
```

在原型题上总结的一些规律

instanceof：检测实例是否属于这个类

- 规律：只要是在当前实例原型链上出现的类，使用instanceof检测的时候，出现的结果都是true
  `f instanceof Fn ->true`
  `f instanceof Object ->true`

  每一个类的原型都应该拥有一个constructor属性，这个属性存储的值就是当前函数本身；但是如果我们把prototype的值重新指向一个新的堆内存（自己手动创建的对象），类的原型就会失去constructor这个属性；所以在以后开发的时候，如果我们想批量给原型设置属性和方法，需要注意constructor，防止覆盖 => `自己开辟创建对象没有constructor`
  构造原型模式：组件封装
  当Fn.prototype指向改变后，当前堆内存不被占用，浏览器会把它销毁

```
~function(){
    var jQuery = function(){
        //...
    };
    //->让jQuery这个类的原型重新指向了新的堆内存，这样jQuery的constructor属性就被覆盖了，这样不好，我们需要自己手动的增加constructor属性
    jQuery.prototype = {

    };
    window.jQuery = window.$=jQuery;
}();
```

isPrototypeOf / propertyIsEnumerable ->枚举

函数的三种角色

- 1、普通函数

> 私有作用域、形参赋值、变量提升、代码执行、返回值、arguments、栈内存的销毁不销毁、作用域链……

- 2、类

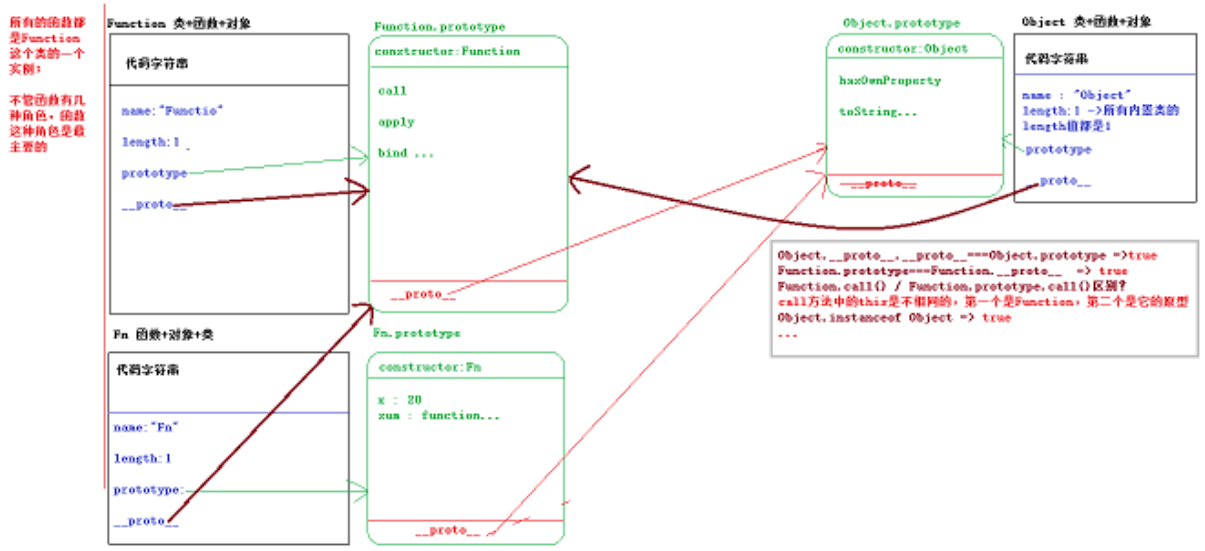
> 实例、类、instanceof、constructor、prototype、__proto__、原型链

- 3、普通对象

> 就把它当作一个普通的obj即可，有自己的属性名和属性值……
> => name：”函数名”
> => length：形参的个数
> => prototype
> => __proto__
> => ……

三种角色之间没有必然的关系

```
function Fn(num) {
    var num = 100;
    this.x = this.y = num;
}
Fn.prototype = {
    constructor:Fn,
    x: 20,
    sum: function () {
        var res = this.x+this.y;
        console.log(res);
    }
};
var f = new Fn(10);
```



call / apply / bind

> 属于Function.prototype上定义的三个方法，所有的函数数据类型值都可以调取这三个方法
>
> 三个方法都是用来改变一个函数中的this关键字指向的（bind不兼容IE6~8，其余两个方法兼容所有的浏览器）

```
var obj = {
    total: 0
};
function sum(num1, num2) {
    //var total = num1 + num2;
    //this.total = total;
    this.total = num1 + num2;
    //console.log(this, total);
}
// sum(10, 20);//->this:window  =>window.total=30
// obj.sum(10, 20);//Uncaught TypeError: obj.sum is not a function
obj不具备sum这个属性，所以不可以通过这种方式把方法中的this修改为obj
//=> sum.call：sum这个实例通过原型链的查找机制，找到Function.prototype的call方法
// sum.call();：基于上一步，把找到的方法执行（执行的是call这个方法）
```

`call方法作用：`

- -> 把需要操作函数中的this变为第一个实参的值

> [非严格模式下]
>
> - 第一个实参为空或者写null或者undefined，this都是window，剩下的第一个实参是谁，this就是谁
>
> [严格模式下]
> - 第一个实参为空，this是undefined，其余的写谁this就是谁