/**
* each是一个集合迭代函数，它接受一个函数作为参数和一组可选的参数
# * 这个迭代函数依次将集合的每一个元素和可选参数用函数进行计算，并将计算得的结果集返回
{%example
<script>
var a = [1,2,3,4].each(function(x){return x > 2 ? x : null});
var b = [1,2,3,4].each(function(x){return x < 0 ? x : null});
alert(a);
alert(b);
</script>
%}
* @param {Function} fn 进行迭代判定的函数
* @param more ... 零个或多个可选的用户自定义参数
* @returns {Array} 结果集，如果没有结果，返回空集
*/
Array.prototype.each = function(fn){
fn = fn || Function.K;
var a = [];
var args = Array.prototype.slice.call(arguments, 1);
for(var i = 0; i < this.length; i++){
var res = fn.apply(this,[this[i],i].concat(args));
if(res != null) a.push(res);
}
return a;
};

/**
* 得到一个数组不重复的元素集合
# * 唯一化一个数组

```
* @returns {Array} 由不重复元素构成的数组
*/
Array.prototype.uniquelize = function(){
var ra = new Array();
for(var i = 0; i < this.length; i ++){
if(!ra.contains(this[i])){
ra.push(this[i]);
}
}
return ra;
};

/**
```

* 求两个集合的补集

```
{%example
<script>
var a = [1,2,3,4];
var b = [3,4,5,6];
alert(Array.complement(a,b));
</script>
%}
* @param {Array} a 集合A
* @param {Array} b 集合B
* @returns {Array} 两个集合的补集
*/
Array.complement = function(a, b){
return Array.minus(Array.union(a, b),Array.intersect(a, b));
};

/**
```

* 求两个集合的交集

```
{%example
<script>
```

```
var a = [1,2,3,4];
var b = [3,4,5,6];
alert(Array.intersect(a,b));
</script>
%}
* @param {Array} a 集合A
* @param {Array} b 集合B
* @returns {Array} 两个集合的交集
*/
Array.intersect = function(a, b){
return a.uniquelize().each(function(o){return b.contains(o) ? o : null});
};

/**
```

* 求两个集合的差集

```
{%example
<script>
var a = [1,2,3,4];
var b = [3,4,5,6];
alert(Array.minus(a,b));
</script>
%}
* @param {Array} a 集合A
* @param {Array} b 集合B
* @returns {Array} 两个集合的差集
*/
Array.minus = function(a, b){
return a.uniquelize().each(function(o){return b.contains(o) ? null : o});
};

/**
```

* 求两个集合的并集

```
{%example
```

```
<script>
var a = [1,2,3,4];
var b = [3,4,5,6];
alert(Array.union(a,b));
</script>
%}
* @param {Array} a 集合A
* @param {Array} b 集合B
* @returns {Array} 两个集合的并集
*/
Array.union = function(a, b){
return a.concat(b).uniquelize();
};
```

巧妙的两个数组比较去重

```
var arr1 = ["i", "b", "c", "d", "e", "f","x"]; //数组A

var arr2 = ["a", "b", "c", "d", "e", "f", "g"];//数组B

var temp = []; //临时数组1

var temparray = [];//临时数组2

for (var i = 0; i < arr2.length; i++) {

temp[arr2[i]] = true;//
```

巧妙地方： 把数组B的值当成临时数组1的键并赋值为真

```
};

for (var i = 0; i < arr1.length; i++) {
```

```
if (!temp[arr1[i]]) {

temparray.push(arr1[i]);//
```
巧妙地方： 同时把数组A的值当成临时数组1的键并判断是否为真，如果不为真说明没重复，就合并到一个新数组里，这样就可以得到一个全新并无重复的数组

```
} ;

};

document.write(temparray.join(",") + "");
```

封装去重

```
// 求数组差值（去重  arr1有arr2没有的）
function minus(arr1, arr2) {
        var newArr1 = [];
        var newArr2 = [];
        for(var i in arr1) {
                newArr1.push(arr1[i].name);
        }
        for(var i in arr2) {
                newArr2.push(arr2[i].name)
        }
        console.log(newArr1)
        console.log(newArr2)
        var temp = []; //临时数组1
        var temparray = []; //临时数组2
        for(var i = 0; i < newArr2.length; i++) {
                temp[newArr2[i]] = true;
        };
        for(var i = 0; i < newArr1.length; i++) {
                if(!temp[newArr1[i]]) {
                        temparray.push(newArr1[i]);
```

```
                };
        };
        return temparray;
}
var noPowers = minus(list_control,list);
```