ajax预习课-基础工具方法

珠峰培训

惰性函数

- **第一次**计算得到的值，可以直接使用。
- 然后用这个函数**重置外部函数**，以后就不用计算，也不用判断分支了。

```
function Max1() {return 1;}
function Max2() {return 2;}
function Max3() {return 3;}

function getMax() {
    var fns = [Max1, Max2, Max3];
    var max = fns[0](), maxFn = fns[0];
    for (var i = 1; i < fns.length; i++) {
        var val = fns[i]();
        if (val > max) {
            max = val;
            maxFn = fns[i];
        }
    }
    getMax = maxFn;
    return max;
}
```

forEach 循环

对数组中的所有元素在指定 **上下文** 中循环逐个调用指定的函数

```
var util = {
    each: (function () {
        if ([].forEach) {
            return function (list, callback, context) {
                list.forEach(callback, context);
            }
        }
        return function (list, callback, context) {
            for (var i = 0, l = list.length; i < l; i++) {
                callback.call(context, list[i], i, list)
            }
        }
    })()
}
```

extends 继承

用config覆盖原有_default对象上的属性，并返回一个全新的对象

```
var util = {
   extends: function (_default, config) {
      var result = {};
      for (var attr in _default) {
         result[attr] = config[attr] || _default[attr];
      }
      return result;
   }
}
```

柯里化 函数

柯里化就是 **预先** 将某些参数传入，得到一个函数。但是预先传入的参数被保存在 **闭包** 中。

```
var isType = function(type){
    return function(obj){
        return Object.prototype.toString.call(obj) == '[object '+type+']';
    }
}
var arr = ['String','Object','Function','Array'];
arr.forEach(function(item){
    util['is'+item] = isType(item);
})
```

验证类

可以对传入的对象链式添加一系列 **验证规则** ，然后开始逐条验证。

```
var util = {
 Verify: (function () {
            var Verify = function (config) {
                if (!(this instanceof Verify)) {
                    return new Verify(config);
                }
                this.config = config;
                this.rules = [];
            }

            Verify.prototype = {
                addRule: function (rule) {
                    this.rules.push(rule);
                    return this;
                },
                start: function () {
                    this.rules.forEach(function (rule) {
                        if (!rule.validate())
                            throw new Error(rule.errMsg);
                    });
                }
            }
            return Verify;
```

对象转成查询字符串

```
var util = {
   toQueryString: function (data) {
               var arr = [];
               for (var attr in data) {
                   if (data.hasOwnProperty(attr)) {
                       arr.push(encodeURIComponent(attr) + '='
                        + encodeURIComponent(data[attr]));
                   }
               }
               return arr.join('&');
           }
}
```

追加url参数

```
var util = {
 append: function(config, data) {
            config.url = config.url+((/\?/.test(config.url) ? '&' : '?') + data);
        }
}
```

bind兼容绑定函数

```
var util = {
    bind: (function () {
        if (Function.prototype.bind) {
            return function(func,context){
                return func.bind(context);
            }
        }
        return function (func,context) {
            return function(){
                func.apply(context,arguments);
            }
        }
    })()
}
```

toJSON(转成JSON)

```
var util = {
        parseJSON: (function () {
           if (global.JSON) {
               return function (data) {
                   return global.JSON.parse(data);
               }
           } else {
               return function (data) {
                   //return eval("("+data+")");
                   return new Function('return ' + data)();
               }
           }
       })()
```

promise

Promise 表示一个**异步**操作的最终结果，与之进行交互的方式主要是`then`方法。

```
function Promise(){
    this.onAlways = this.onFail= this.onDone = function(){}
}
Promise.prototype = {
    done:function(fn){
        this.onDone = fn;
        return this;
    },fail:function(fn){
        this.onFail = fn;
        return this;
    },always:function(fn){
        this.onAlways = fn;
        return this;
    }
```

珠峰培训

❝ Promises/A+规范　深入理解Promise实现细节　JavaScript Promise迷你书

小数取整

```
var num = 1.1;
console.log(parseInt(num));
console.log(Math.ceil(num)); // ceil向上取整
console.log(Math.floor(num)); // floor向下取整:
console.log(Math.round(num)); //round四舍五入
console.log(num.toFixed(0)); //Number 四舍五入为指定小数位数的数字
console.log(~~num);          //~~ 按位两次取反相当于取整
```

encodeURIComponent

- url是ASCII码(只能表示128种字符)格式的，并且特殊字符有特殊含义
- 需要把正常参数里的特殊字符和不能用ascii码表示的字符进行转义成十六进制的转义序列

```
console.log(new Buffer('?'),encodeURIComponent('?'));
// <Buffer 3f> '%3F'
console.log(new Buffer('中'),encodeURIComponent('中'));
// <Buffer e4 b8 ad> '%E4%B8%AD'
```

encodeURIComponent函数 ASCII 码