

一、把类数组转换为数组

借用数组原型上的slice方法，让slice方法执行的时候，里面的this指向arguments，这样就相当于在操作arguments这个类数组（原因：arguments是类数组，它的结构和数组太相似了，数组的一些循环遍历操作，也同样适用于类数组）

```
function fn(){
    //arguments: 类数组
    ary=Array.prototype.slice.call(arguments);
    var ary=[];
    ary.slice.call(arguments);
}
fn(12,23,34)
```

在js中，元素集合（HTMLCollection的实例）以及节点集合（NodeList的实例）也都是类数组

这些类数组也可以借用数组原型的方法，实现一些相关的操作，例如：借用slice把类数组转换为数组

```
var allTagList = document.getElementsByTagName("*");//->【*】通配符：获取当前页面中所有的元素标签
var ary = Array.prototype.slice.call(allTagList);
console.log(ary instanceof Array, ary);//=>true,数组
```

★ 重要：call、apply、bind、类数组转换为数组 ★

在IE低版本浏览器（IE6~8）中运行，我们发现arguments这个数组是可以借用数组的方法正常操作的，但是元素集合和节点集合的类数组，无法借用数组原型上的方法，提示：Array.prototype.slice:"this"不是javascript对象的错误

所以在IE低版本中，我们把节点或者元素集合转换为数组的时候，就不能偷懒了

```

var oList=document.getElementsByTagName("li");

//->IE6~8
var ary=[];
for(var i=0; i<oList.length; i++){
    ary[ary.length]=oList[i];
}

```

综上所述，如果我们想实现一个方法 **把类数组转换为数组**，我们需要考虑的事情

- 1、兼容 `[].slice.call([类数组])` 这种模式的，我们使用这种模式，不能兼容的浏览器，我们使用 **循环** 一步步的处理
- 2、如何验证是否兼容：不兼容的情况下，浏览器会抛出异常（报错），换句话说，只要报错了就是不兼容

```

function toArray(likeAry) {
    var ary = [];
    try {
        //->如果不报错代表兼容
        ary = Array.prototype.slice.call(likeAry);
    } catch (e) {
        //->报错进入catch，代表不兼容（IE6~8）
        for (var i = 0, len = likeAry.length; i < len; i++) {
            ary[ary.length] = likeAry[i];
        }
    }
    return ary;
}

```

完善第一个方法库：

```

/**
 * utils: 整个项目的公共方法库(单例模式封装)
 */
var utils = (function () {
    //->toArray: 把类数组转换为数组
    function toArray(likeAry) {
        var ary = [];
        try {
            ary = [].slice.call(likeAry);
        } catch (e) {
            var len = likeAry.length;
            for (var i = 0; i < len; i++) {
                ary[ary.length] = likeAry[i];
            }
        }
        return ary;
    }

    return {
        toArray:toArray//->utils.toArray();
    }
})();

```

二、浏览器异常信息捕获 **try catch finally** => [同后台语法一模一样]

```

try{
    //->js代码
}catch(e){
    //->try中的js代码执行，如果出现错误，会进入到catch中（没出错不进入catch这里）
    //->e (error) 是一个形参（名字可以随便起，但是必须写形参，不写语法报错）
    e.message: 存储的是当前js代码执行的报错信息
}finally{
    //->不管js代码执行是否报错，最后都会执行finally中的内容（项目中不常用）
}

```

使用try catch的作用:

- 可以捕获到错误信息的同时，防止浏览器抛出异常信息，这样即使当前代码报错了，也不会影响下面的执行
- 可以监听到报错，我们可以利用这个机制，做一些兼容处理：把需要执行的代码放在try中，如果不兼容报错了，我们在catch中处理兼容即可（兼容处理兼容检测）

```
try {
    console.log(a);
} catch (e) {
    console.log(e.message); //->报错信息
}
var b = 13;
console.log(b); //->13
```

//->很多公司，会把项目代码最外面包裹一层try catch，以此来收集错误信息，而且还不会在浏览器控制台抛出异常错误

//->还有公司是每个开发者把自己的代码包裹起来，或者经常容易出现问题的代码包裹起来

//=>最后的目的都是收集错误信息

```
try {
    //->项目的所有js代码
} catch (e) {
    //->收集报错信息：统计到服务器上
}
```

手动抛出异常信息：

```
//->手动在浏览器中抛出异常信息
/*
 * 使用try catch捕获异常信息的时候，后面代码还可以继续执行，但是项目中难免会出现
这样的需求：我们上面代码如果不能正常的执行，下面代码也不让它执行了，此时需要我们
手动抛出异常来阻止下面代码的执行
 * */
//throw new Error("您的人品欠费，请充值~~"); //->创建Error类的一个实例： 一
条错误信息 [抛出一个Error类的实例]
/*
 * Error这个类划分了几个常用的小类：
 *   ->ReferenceError： 引用错误
 *   ->TypeError： 类型错误
 *   ->RangeError： 范围错误
 *   ->SyntaxError： 语法错误
 *   ...
 * */
//throw new Error("当前网络繁忙，请稍后重试! ~~");
try{
    console.log(a);
}catch (e){
    //e.message;
    throw new ReferenceError("当前网络繁忙，请稍后重试! ");
}
```


提供两个方法：

方法一：`JSON.parse` 全称 `window.JSON.parse`

在全局对象中存在一个JSON对象，这个对象中有一个parse方法：把JSON格式的字符串转换为JSON格式的对象

```
// -> JSON字符串
var str='[{"id":1,"name":"Junior"}, {"id":2,"name":"Fancy"}]';

var jsonData=JSON.parse(str);
console.log(jsonData); // -> JSON对象
```

// -> 意义：服务器端返回给客户端的一般都是JSON字符串，我们操作里面属性和属性值的时候不好操作，此时我们需要先把这个字符串变为对象才可以

```
> var str='[{"id":1,"name":"Junior"}, {"id":2,"name":"Fancy"}]';
< undefined
> JSON.parse(str)
< ▼ (2) [Object, Object] ⓘ
  0: Object
    id: 1
    name: "Junior"
    __proto__: Object
  1: Object
    id: 2
    name: "Fancy"
    __proto__: Object
  length: 2
  __proto__: Array(0)
```

方法二：`JSON.stringify`

把JSON格式的对象（普通格式的也可以）转换为JSON格式的字符串

//->JSON对象

```
var ary=[{
  id:1,
  name:"Junior"
},{
  id:2,
  name:"Fancy"
}];
```

//把它变成JSON格式字符串

```
var jsonStr=JSON.stringify(ary);//->JSON格式的字符串
```

//->"[{\"id\":1,\"name\":\"Junior\"},{\"id\":2,\"name\":\"Fancy\"}]\""

//->意义：有些时候，客户端不仅仅向服务器发请求，还会给服务器传递点内容（例如：注册的时候，需要客户端把用户输入的内容传递给服务器），客户端传递给服务器的一般都是JSON格式的字符串，此时就需要我们使用stringify这个方法把得到的结果变为字符串，再传递给服务器

```
> var ary=[{
  id:1,
  name:"Junior"
},{
  id:2,
  name:"Fancy"
}];
< undefined
> JSON.stringify(ary)
< "[{\"id\":1,\"name\":\"Junior\"},{\"id\":2,\"name\":\"Fancy\"}]\"
>
```

在IE6~7中，window下没有JSON这个对象，也就是不能使用 `JSON.parse` 和 `JSON.stringify` 这两个方法了

`JSON.parse` 被 `eval` 取代了（兼容用parse，不兼容用eval）

`JSON.stringify` 没有能取代的方法，需要自己编写复杂的处理程序（思考题 -> 同数组和对象的深度克隆相似）

//->JSON字符串

```
var str='[{"id":1,"name":"Junior"}, {"id":2,"name":"Fancy"}]';
```

`eval('(' + str + ')');` //->手动给需要解析的字符串外层包裹一个小括号，防止解析出错：对于'{...}'这种格式的字符串，如果外层不多加一层小括号，解析出来，属于语法不符合

```
> {}.toString()
✖ Uncaught SyntaxError: Unexpected token .
> ({}).toString()
< "[object Object]"
```

综上所述：我们现在有一个需求：编写一个方法，能够实现把JSON字符串转换为JSON对象

- 兼容：JSON.parse
- 不兼容：eval
- 如果window中存在JSON这个属性，属于兼容，反之不兼容

```
// -> toJSON: 把JSON格式字符串转换为JSON对象 (JQ)
function toJSON(str) {
  return 'JSON' in window ? JSON.parse(str) : eval('(' + str + ');');
}
```

```
▼ JSON: JSON
  ▼ parse: function parse()
    arguments: null
    caller: null
    length: 2
    name: "parse"
    ▶ __proto__: function ()
  ▼ stringify: function stringify()
    arguments: null
    caller: null
    length: 3
    name: "stringify"
    ▶ __proto__: function ()
    Symbol(Symbol.toStringTag): "JSON"
  ▼ __proto__: Object
```

四、AJAX

async javascript and xml 作用： 我们可以通过AJAX向服务器发送请求，并且获取到服务器返回的内容


```
//->①、创建一个AJAX对象（相当于xhr一个实例）
var xhr=new XMLHttpRequest;

//->②、打开一个请求的URL地址
xhr.open([请求方式],[请求的URL地址],[同步或者异步]);
//=> xhr.open('GET',"xxx/xxx.json",false);
false代表的是同步编程：当从服务器端把数据获取到之后，才可以做别的事情

//->③、监听AJAX的状态改变，接收服务器返回的结果
xhr.onreadystatechange=function(){
    if(xhr.readyState === 4 && xhr.status === 200){
        //->从服务器获取数据成功，下面的操作就可以接收到服务器端返回的结果
        //var result=xhr.responseText;//->获取服务器端数据返回的结果
        xhr.responseText;//->服务器端返回的结果
    }
}
//->④、向服务器发送请求
xhr.send(null);
```

```
//例题:
//->JSON文件
[
  {
    "id": 1,
    "name": "Junior"
  },
  {
    "id": 2,
    "name": "Fancy"
  },
  {
    "id": 3,
    "name": "樱桃小丸子"
  }
]
//->JS代码 ->ajax数据请求
var xhr = new XMLHttpRequest;
xhr.open('GET', "json/temp.json", false);
xhr.onreadystatechange = function () {
  if (xhr.readyState === 4 && xhr.status === 200) {
    console.log(xhr.responseText);//->获取的结果是一个json字符串
    var res = xhr.responseText;
    console.log(utis.toJSON(res));
  }
}
xhr.send(null);// ->因为get请求是没有请求体的所以为null (http://www.jb51.net/article/86157.htm)
//->返回结果下图截图
```

```
[
  {
    "id": 1,  获取的结果是一个JSON字符串
    "name": "Junior"
  },
  {
    "id": 2,
    "name": "Fancy"
  },
  {
    "id": 3,
    "name": "樱桃小丸子"
  }
]
```

```
▼ (3) [Object, Object, Object] ⓘ
  ▼ 0: Object
    id: 1
    name: "Junior"
    ▶ __proto__: Object
  ▼ 1: Object
    id: 2
    name: "Fancy"
    ▶ __proto__: Object
  ▼ 2: Object  将JSON字符串转换为JSON对象
    id: 3
    name: "樱桃小丸子"
    ▶ __proto__: Object
  length: 3
  ▶ __proto__: Array(0)
```

sort数组排序的方法

```
//->sort原理:
var ary = [12, 23, 14, 25, 23];
ary.sort(function (a, b) {
  // console.log(a, b);
  //->a: 当前项
  //->b: 后一项
  // return a - b;
  return 1; //->return后面不一定非要写a-b, 它的原理是: 如果返回的是一个大于
0的数, 当前项和后一项交换位置, 如果小于等于0的值, 不做任何的处理
});
```

```
//->案例①
var ary = ["Junior", "Fancy", "樱桃小丸子", "曹文博", "小玲子"];
ary.sort(function (a, b) {
    // return a - b;
    return a.localeCompare(b);
    // return 1;
});
console.log(ary);

//-> localeCompare: 可以做字符串的比较, 按照拼音字母在字母表中的顺序排列, 越靠后的字母越大
//"Junior".localeCompare("Fancy") => 1
//"Fancy".localeCompare("Junior") => -1
```

```
//->案例②
var ary = [{
    name: "Junior",
    age: 20,
    score: 66
}, {
    name: "Fancy",
    age: 18,
    score: 88
}, {
    name: "樱桃小丸子",
    age: 22,
    score: 99
}];
/*ary.sort(function (a, b) {
    return b.age - a.age;
});*/
/*ary.sort(function (a, b) {
    return a.score - b.score;
});*/
ary.sort(function (a, b) {
    return a.name.localeCompare(b.name); // 字符串的比较: localeCompare
});
console.log(ary);
```

数据绑定 案例-华为商城-列表排序

```
// -> HTML
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>华为商城-列表排序</title>
  <!--import css-->
  <link rel="stylesheet" href="css/reset.min.css">
  <link rel="stylesheet" href="css/style.css">
</head>
<body>
<div class="container">
  <!--MENU-->
  <div class="clearfix menu">
    <span>排序: </span>
    <a href="javascript:;">
      上架时间
      <i class="up"></i>
      <i class="down"></i>
    </a>
    <a href="javascript:;">
      价格
      <i class="up"></i>
      <i class="down"></i>
    </a>
    <a href="javascript:;">
      热度
      <i class="up"></i>
      <i class="down"></i>
    </a>
  </div>
  <!--list-->
  <ul class="clearfix mallList" id="mallItem">
    <!--<li>
      <a href="#">
        
        <span class="title">华为运动蓝牙耳机</span>
        <span class="price">¥999</span>
      </a>
    </li>-->
  </ul>
</div>
<script type="text/javascript" src="js/utils.js"></script>
<script type="text/javascript" src="js/index.js"></script>
</body>
</html>
```

```

/**
 * 一、数据绑定（JS）
 * 从服务器端使用AJAX获取到需要的数据，然后绑定在页面中
 */
//->1、首先从服务器端获取数据（AJAX）
var productData = null;
var xhr = new XMLHttpRequest;
xhr.open("GET", "json/product.json", false);
xhr.onreadystatechange = function () {
    if (xhr.readyState === 4 && xhr.status === 200) {
        // console.log(xhr.responseText);
        var res = xhr.responseText; //->res是一个JSON字符串，我们需要把它
        转换为JSON对象，这样才方便我们后续操作
        productData = utils.toJSON(res);
    }
};
xhr.send(null); // ->因为get请求是没有请求体的所以为null

```

//->2、把我们得到的数据绑定在页面上即可

// 目前市场上绑定数据常用的办法：

// =>1）字符串拼接：把html标签及需要展示的数据在js中都以字符串的形式拼接起来，最后使用innerHTML插入到页面中

// =>2）ES6中提供的模板字符串（原理也是字符串拼接）

// =>3）数据绑定的模板引擎：kTemplate、EJS、d3、REACT中的虚拟DOM渲染、VUE中的MVVM模式等等

var htmlStr = ``; //tab键上面的点，不是单引号（ES6：兼容这个写法=>webstorm->settings->Languages & Frameworks->JavaScript->右侧选择JavaScript language version`ECMAScript 6`->ok，等待webstorm加载完成后右上侧有个desmis选择完成即可，否则webstorm报错！）

```

for (var i = 0; i < productData.length; i++) {
    var curItem = productData[i];
    // console.log(curItem);
    // ->每当循环一次都应该动态创建一个li，还需要把curItem中存储的内容分别的
    绑定在每一个li的身上
    htmlStr += `<li>
        <a href="#">
            
            <span class="title">${curItem.title}</span>
            <span class="price">${curItem.price}</span>
        </a>
    </li>`;
}

```

//->循环完成后htmlStr存储的东西就是我们最后需要展示内容的全部字符串，我们把这堆字符串最后放到页面的容器中即可

```

var mallItem = document.getElementById("mallItem");
mallItem.innerHTML = htmlStr;

```

视频自学->

① （第二周第一节）：课件10：数据绑定及DOM回流 / 课件11：表格排序及DOM映射

② （第二周第二节）：都看一遍

③ 开始预习正则

复习：作用域链/原型链/`fn.call.call.call.call()/call`、`apply`、`bind`/`utils`：类数组转换为数组、`toJSON`/`sort`表格排序/模板字符串/数据获取/数据绑定