20170806

DOM映射机制

> 在js中获取的DOM元素或者元素集合，和HTML页面上的元素标签是存在“映射关系”的
>
> - js中把DOM元素进行修改，页面中的元素样式也会跟着改变
> - 页面中元素的结构发生改变，js中DOM元素也会跟着改变

```
var stuList = document.getElementById("stuList"),
    stuBody = stuList.tBodies[0],
    stuRows = stuBody.rows;
//->开始的时候也没中没有tr，所以stuRows是一个空的类数组
//->ajax获取数据然后做数据绑定
~function () {
    //get data
    var stuData = null;
    var xhr = new XMLHttpRequest;
    xhr.open("GET", "json/data.json", false);
    xhr.onreadystatechange = function () {
        if (xhr.readyState === 4 && xhr.status === 200) {
            stuData = utils.toJSON(xhr.responseText);
        }
    };
    xhr.send(null);

    //bind data
    var str = ``;
    for (var i = 0; i < stuData.length; i++) {
        var curItem = stuData[i];
        str += `<tr>
        <td>${curItem.id}</td>
        <td>${curItem.name}</td>
        <td>${curItem.age}</td>
    </tr>`;
    }
    stuBody.innerHTML = str;//->向页面中增加了20个tr
    console.log(stuRows);//->由于页面中tbody结构中的内容改变了，根据DOM映射机制此处不需要重新的获取，stuRows中存储的就是最新的20条数据
}();
```

正则

> 所谓的正则：
> 用来处理 `字符串` 的规则
>
> - `把验证当前的字符串是否符合规则` – `匹配`
> - `把字符串中符合规则的字符捕获到` – `捕获`
>
> 正则匹配：`[正则].test([字符串])`
> 正则捕获：`[正则].exec([字符串])` 或者 `[字符串].match([正则])` 或者 `[字符串].replace([正则],function...)` 或者 `[字符串].split([正则])` ...

元字符和修饰符

> 一个正则就是由元字符和修饰符组成的，想要学会编写自己所需的规则，需要牢牢掌握元字符和修饰符

`修饰符 img`

- i(ignoreCase)：忽略单词大小写匹配
- m(multiline)：多行匹配
- g(global)：全局匹配

```
//->修饰符放在最后一个斜杠的后面（字面量创建方式）
var reg=/^\d+$/img;

//->修饰符放在第二个实参字符串中（实例创建方式）
var reg=new RegExp("","img");
```

`特殊元字符`

- \：转义字符，把普通元字符转换为特殊的意义，或者把特殊元字符转换为普通的意义，例如：`/\d/ d本身是一个字母，前面加一个转义字符,代表0~9之间的一个数字` 或者 `/\./ 点在正则中代表任意字符（特殊含义），此处加上转义字符，代表的就是本身意思点了`
- ^：以某一个元字符开始，例如：`/^1/ 代表当前的字符串应该是以1开始的`
- $：以某一个元字符结束，例如：`/2$/ 代表当前字符串最后一个字符应该是以2结尾`
- \d：代表一个0~9之间的数字
- \D：和\d正好相反，代表一个非0~9之间的任意字符（所有大写字母都和小写字母的是相反的）
- \w：数字、字母、下划线三者中的任意一个
- \n：匹配一个换行符
- \b：匹配一个边界
- \s：匹配一个空白字符
- .：除了\n以外的任意一个字符

- x|y：x或者y中的一个字符
- ()：分组
- [a-z]：匹配一个a-z中的任意字符`/[0-9]/ 0~9之间的任何一个数字，等价于\d`
- [^a-z]：除了a-z以外的任意一个字符，这里^是取反的意思
- [xyz]：x或者y或者z，三者中的一个
- [^xyz]：除了三者以外的任意一个字符
- ?=：正向预查
- ?!：负向预查
- ?:：只匹配不捕获

量词元字符

- *：前面的元字符出现0次到多次
- +：前面的元字符出现1次到多次
- ?：前面的元字符出现0次或者1次
- {n}：出现n次
- {n,}：出现n到多次
- {n,m}：出现n到m次

```
//->两个斜杠中间包起来的都是正则的元字符
//1、特殊元字符：有特殊含义的
//2、量词元字符：代表出现多少次的元字符
//3、普通元字符：代表本身含义的
var reg=/^\d+$/

var reg=new RegExp("[元字符]","[修饰符]");
```