目录

1. JS概述、JS运行方式、JS开发工具 博客系统功能介绍
2. jQuery框架特点、页面中引用，功能
3. Bootstrap框架 Bootstrap功能、Bootstrap样式、Bootstrap可视化设计界面
4. JS基本数据类型、JS运算符与表达式
5. JS String对象、JS RegExp对象
6. JS Number对象、JS Math对象
7. JS Date对象
8. JS Array对象、JS JSON对象
9. JS流程控制语句
10. 函数定义与调用
11. 使用函数创建自定义类
12. 习题课1
13. jQuery框架中插件类型与使用
14. Bootstrap组件
15. CSS基本选择器、H5的选择器使用方法
16. DOM对象创建、查看、删除
17. DOM对象属性修改、样式修改
18. DOM对象遍历
19. DOM对象的事件类型、事件注册、事件触发与事件注销
20. DOM对象的事件参数--事件对象
21. jQuery框架中的插件定义与调用
22. 使用纯JS实现页面动画
23. 使用jQuery实现页面动画
24. 表单及表单域对象的类型、数据获取
25. 表单及表单域对象的事件、数据验证
26. jQuery框架的ajax调用
27. 表单的数据提交
28. BOM -Window对象、Screen对象
29. BOM-Location对象、Navigator对象
30. BOM-弹窗、定时器、Cookie
31. H5中localStorage与sessionStorage
32. 复习

JS概述、JS运行方式、JS开发工具 博客系统功能介绍

1. JS概述

JavaScript 是 Web客户端的编程语言，主要运行在浏览器中。

所有现代的 HTML 页面都使用 JavaScript。HTML 页面是JavaScript的宿主

JavaScript上手容易精通难。

跨平台，基于客户端浏览器，有浏览器负责解释执行

基于对象（DOM对象、BOM对象、String对象、Array对象）

事件驱动（点击事件、拖放事件、键盘事件、定时器事件、装载事件等）

JavaScript的作用：动态修改HTML页面中的内容、样式；产生页面请求、响应页面请求；H5中多媒体操作、页面绘图、本地存储、客户端通信。

JavaScript的地位：HTML是演员，CSS是化妆师，JavaScript是导演。

JS的运行方式

1. 嵌入JS到网页script标签中，该网页装载结束JS就运行。
2. 在浏览器的控制台中，输入语句回车后立即执行。
3. 使用script标签引入JS文件到网页中，该网页装载结束JS就运行。(实现重用)

JS的输出位置

1. 页面中（用于浏览）
2. 对话框中（用于提醒）
3. 浏览器的控制台中（用于调试）
4. JS开发工具

HBuilder或HBuilderX

最大特点是快，通过智能的语法提示和[代码](https://baike.baidu.com/item/%E4%BB%A3%E7%A0%81)输入法、代码块等，大幅提升[HTML](https://baike.baidu.com/item/HTML)、[css](https://baike.baidu.com/item/css/5457)、[js](https://baike.baidu.com/item/js/10687961)的开发效率。

HTML示例：

输入sty或stl等选择后加回车可以迅速产生如下标记：

<style type="text/css">

</style>

输入ul>li\*3加tab可以迅速产生如下标记：

<ul>

<li></li>

<li></li>

<li></li>

</ul>

[css](https://baike.baidu.com/item/css/5457):

输入:pstion选择后可以得到position

JS：输入doc选择.getid选择可以得到document.getElementById()

1. 博客系统功能介绍

如何进入爱课程，学习JavaScript及其框架？

1. 在浏览器中打开[http://www.icourses.cn/home/#](http://www.icourses.cn/home/)
2. 在搜索框中搜索javascript
3. 点击课程图像，进入课程
4. 进入学习
5. 继续学习
6. 选择章、专题
7. 选择资源类型（视屏、富文本、讨论、文档、测验题、向老师提问）

菜鸟教程：<https://www.runoob.com>

示例代码：讲课素材

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title id="title"></title>

<style type="text/css">

h2{

color:green;

}

</style>

</head>

<body>

<h2 id="head1">350181同学们：</h2>

<div></div>

<ul id="list">

<li></li>

<li></li>

<li></li>

</ul>

<p id="info"></p>

<script type="text/javascript">

//document.write("Hello")

document.getElementById("info").innerHTML="<span style='color:red;' >Hello</span>"

document.getElementById("head1").style.color = "red"

document.getElementById("title").innerText = "大家好";

</script>

<div>

</div>

<script type="text/javascript">

document.getElementsByTagName("div")[1].innerHTML = "=========";

document.getElementsByTagName("div")[0].innerHTML = "---------";

alert(document.getElementById("title").innerText)

console.log(document.getElementById("title"))

console.log(document.getElementById("title").innerText)

console.log(document.getElementsByTagName("div"))

</script>

<h2>版权所有</h2>

</body>

</html>

示例代码：视频连续播放

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>视频连续播放</title>

<style type="text/css">

h2{

position: absolute;

color:rgba(0,0,0,0.5);

}

h3{

}

</style>

</head>

<body>

<h2>视频连续播放</h2>

<video width="80%" controls id="myvideo" ></video>

<script>

var vlist = ['1.mp4', '2.mp4'];

var vlen = vlist.length;

var curr = 0;

var vid = document.getElementById("myvideo");

vid.src = vlist[curr];

vid.play();

document.querySelector("h2").innerHTML ="当前播放文件："+ vlist[curr];

vid.onended = function() {

vid.src = vlist[curr + 1];

curr++;

if (curr >= vlen) {

curr = 0;

vid.src = vlist[0]

}

document.querySelector("h2").innerHTML ="当前播放文件："+ vlist[curr];

vid.play();

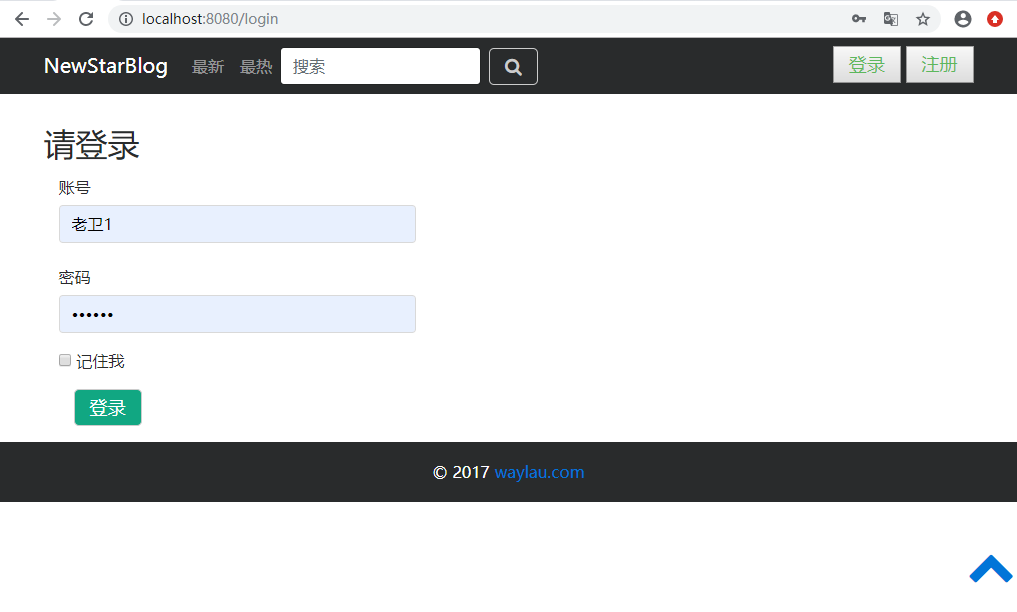
}

</script>

</body>

</html>

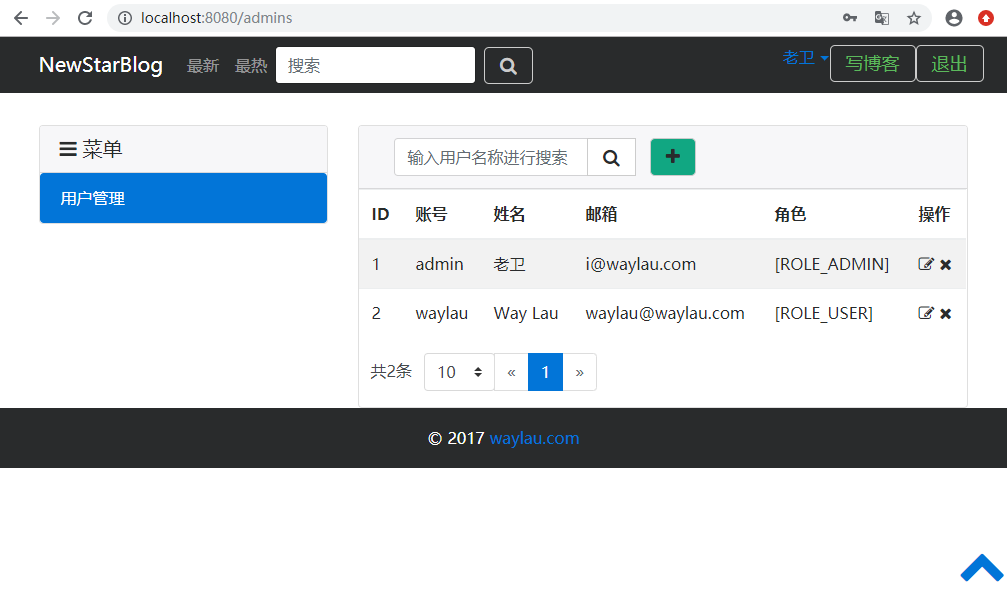
登录：



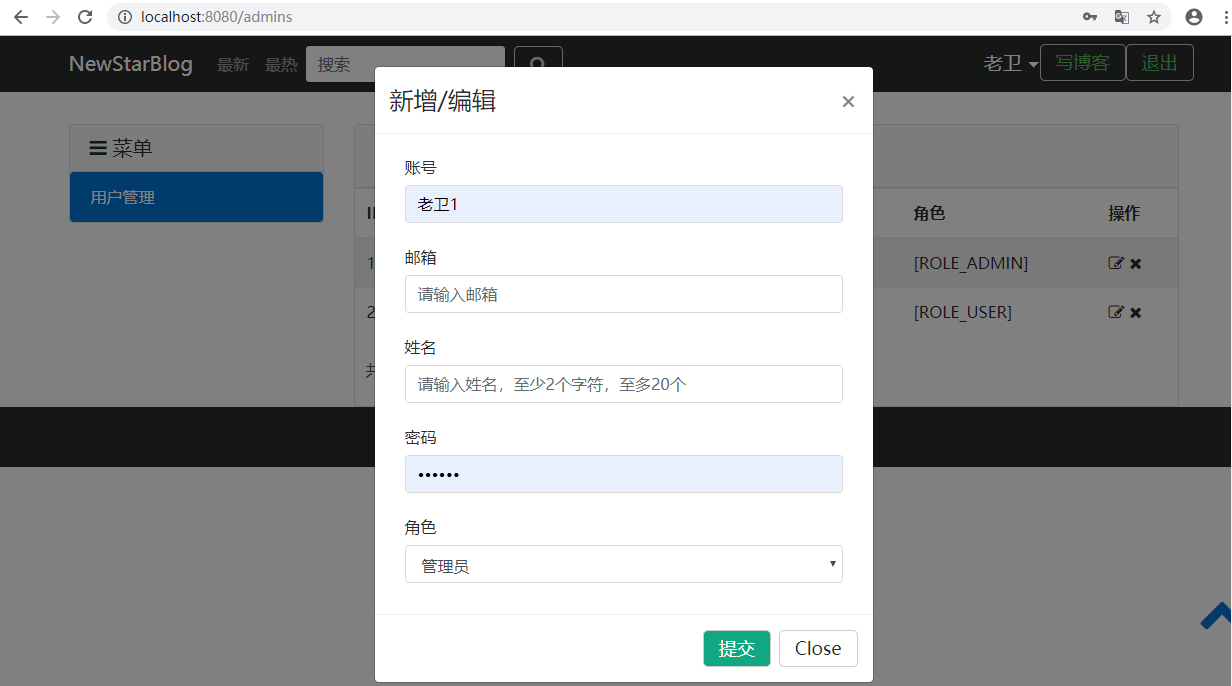
注册：



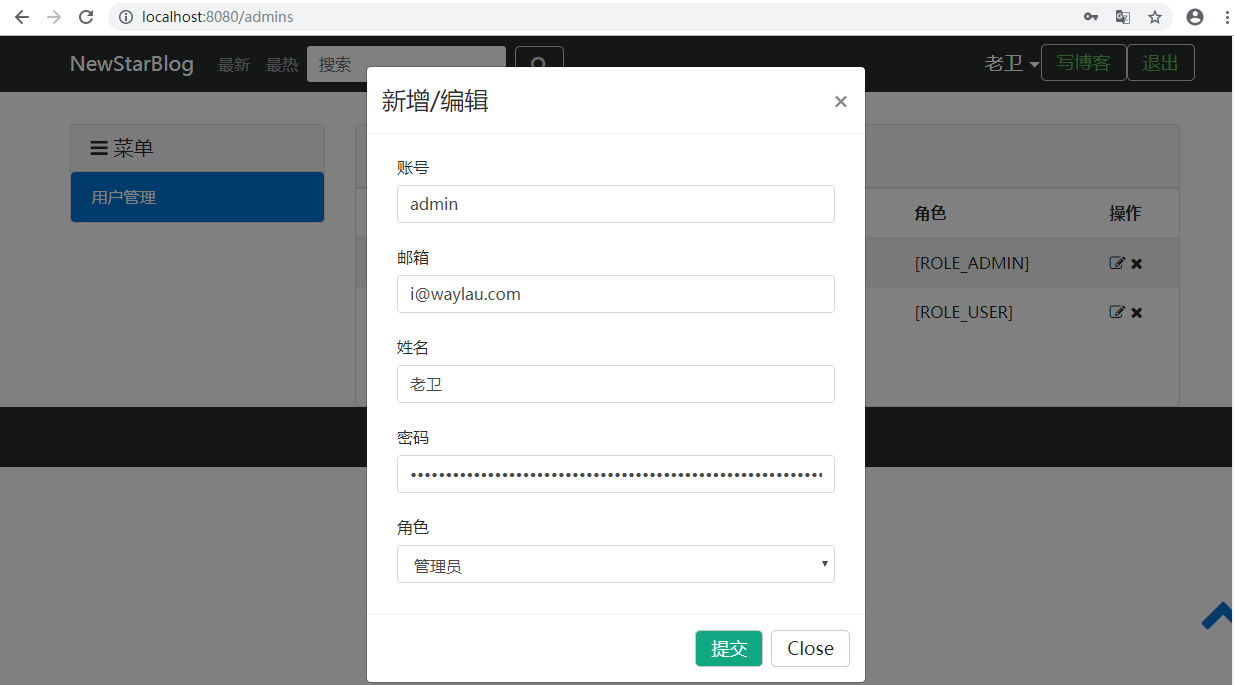
管理员后台：



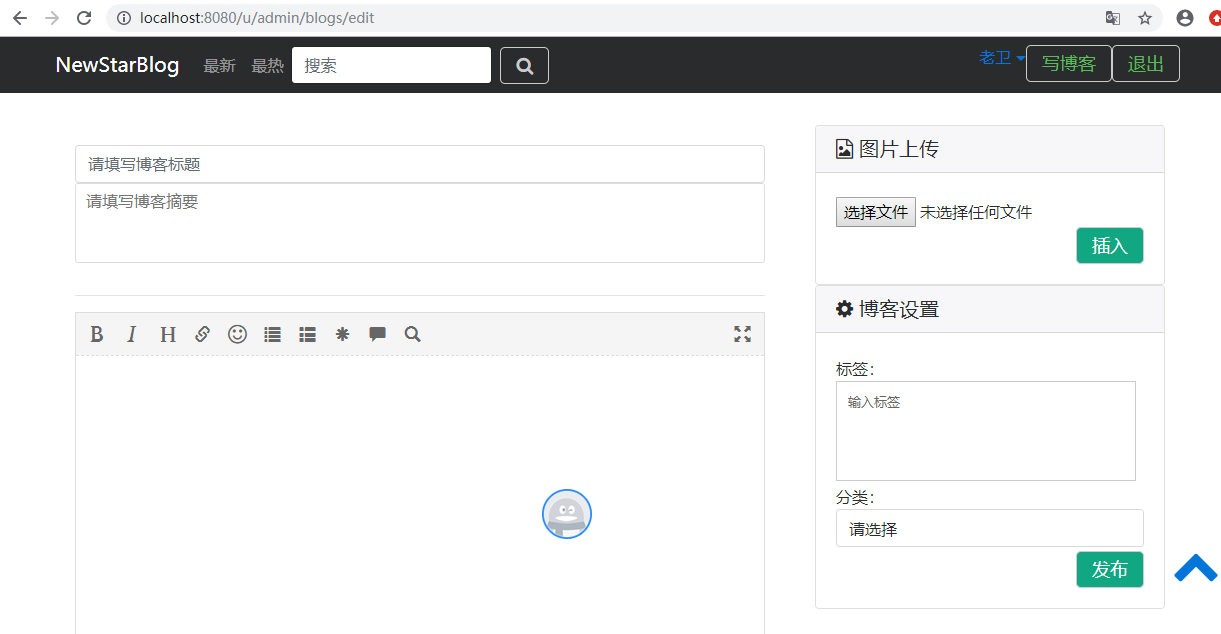
新增用户：



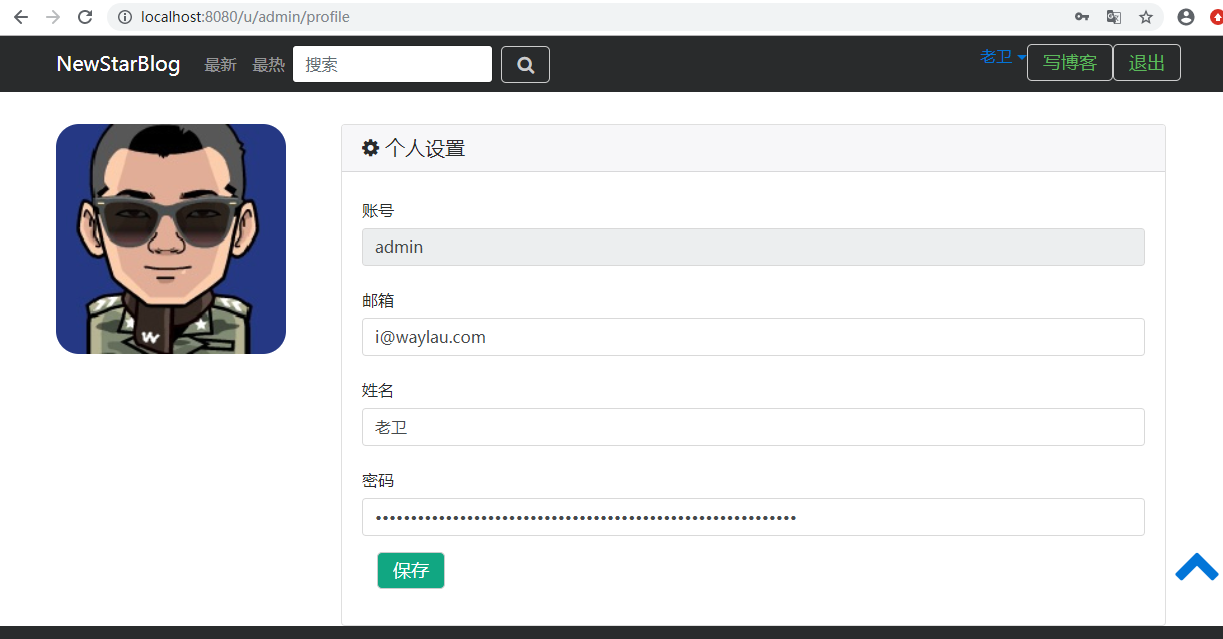
编辑用户：



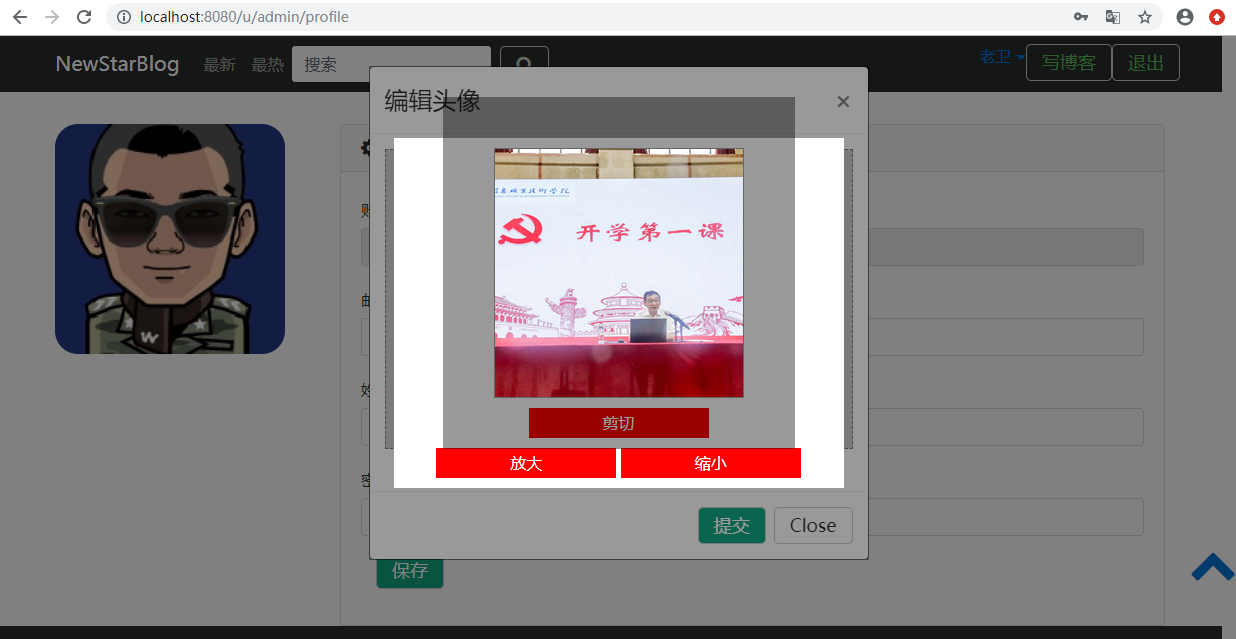
写博客



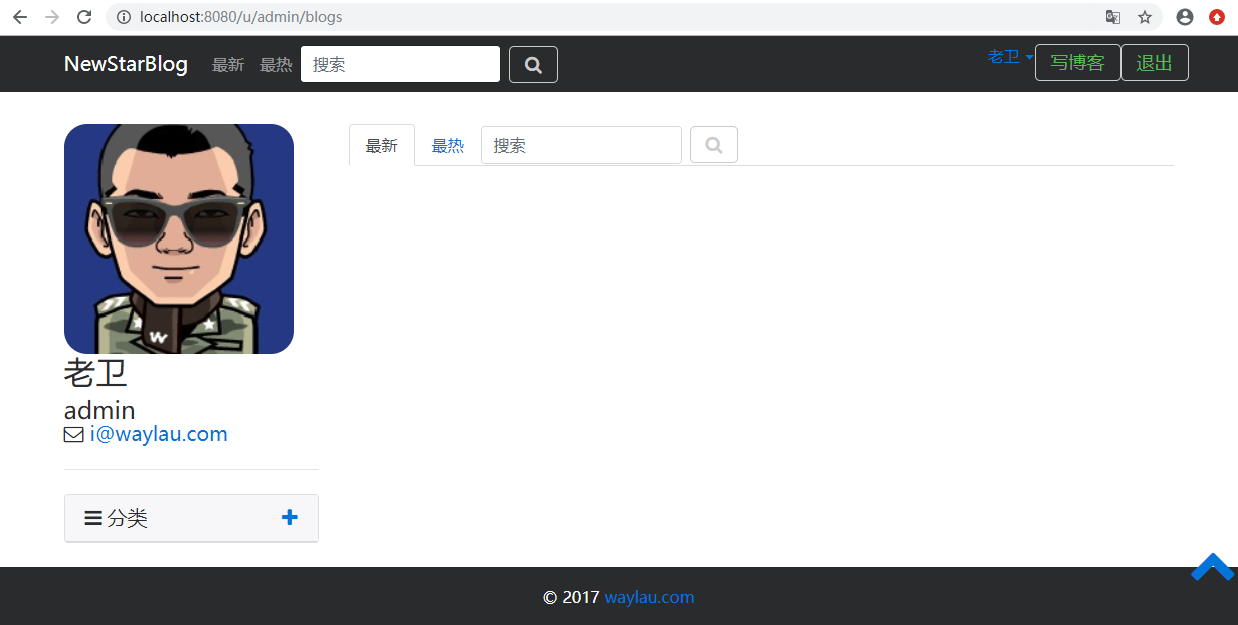
个人设置：



个人头像设置：



个人主页：



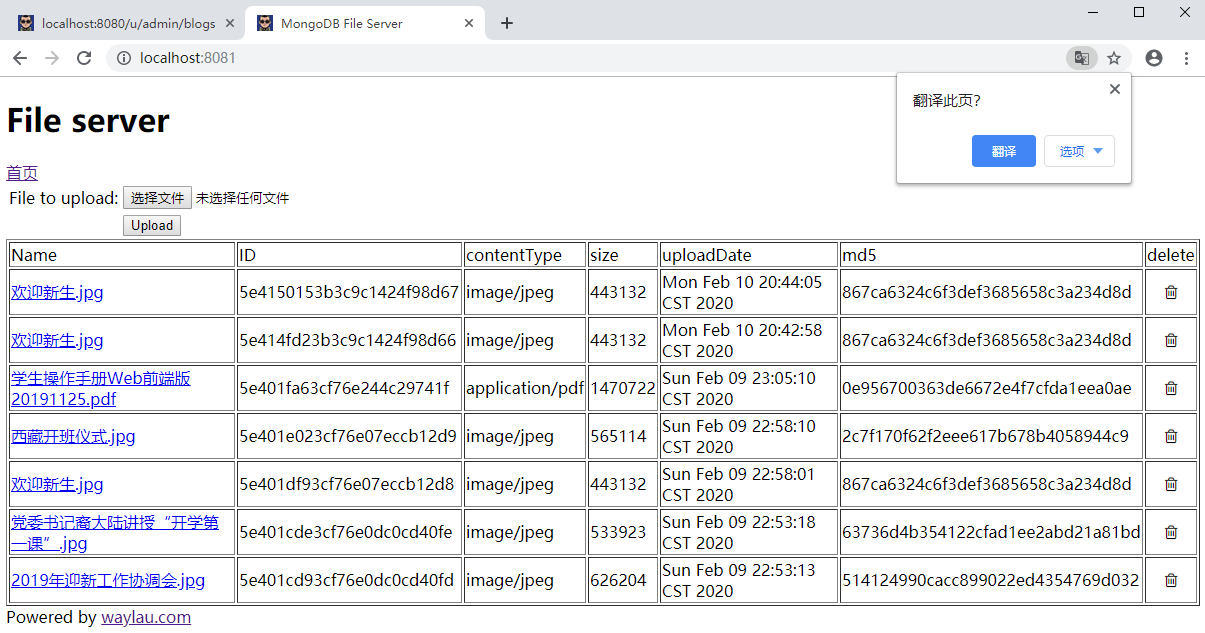
添加分类：

·

博客浏览搜索：



文件服务器页面：



第2次课：jQuery框架特点、页面中引用，功能

复习：

* JavaScript的作用

动态修改HTML页面中的内容、样式；产生页面请求、响应页面请求；

* JavaScript的地位

HTML是演员，CSS是化妆师，JavaScript是导演。

* 基于对象（DOM对象、BOM对象、String对象、Array对象）

对于页面上标签的修改要有两步：1.获得DOM对象；2.操作DOM对象；

* 事件驱动（点击事件、拖放事件、键盘事件、定时器事件、装载事件等）

示例中的连续视频播放使用的是Video对象的ended事件；同学们仿照示例实现按钮单击click事件，单击完成的任务自己确定。

* JS的运行方式

嵌入JS代码到网页script标签中，该网页装载结束JS就运行。

在浏览器的控制台中，输入语句（回车后）或表达式（完整后）立即执行。

使用script标签引入JS文件到网页中，该网页装载结束JS就运行。(实现重用)

* JS的输出位置

页面文档的当前位置或选择位置（用于浏览）

对话框中（用于提醒）

浏览器的控制台中（用于调试）

* JS开发工具

HBuilder或HBuilderX

最大特点是快，通过智能的语法提示和[代码](https://baike.baidu.com/item/%E4%BB%A3%E7%A0%81)输入法、代码块等，大幅提升[HTML](https://baike.baidu.com/item/HTML)、[css](https://baike.baidu.com/item/css/5457)、[js](https://baike.baidu.com/item/js/10687961)的开发效率。

示例：一个移动app首页(只做示例，不做要求)

1. jQuery框架特点

* jQuery 是一个 JavaScript 库。
* jQuery 极大地简化了 JavaScript 编程。
* jQuery 很容易学习。

目标是”写的少,做的多”

1. 页面中引用

jQuery 是一个 JavaScript 库。使用script标签引入JS文件到网页中，该网页装载结束JS就运行。(实现重用)

jQuery库存放位置可以在本地,也可以在远程

示例: 在当前页面中使用远程jQuery库的示例

<script src="https://cdn.staticfile.org/jquery/3.2.1/jquery.min.js"></script>

示例: 在bootstrap中使用本地jQuery库的示例

<script src="../js/jquery/3.2.1/jquery.min.js"></script>

或

<script src="../js/jquery.min.js"></script>

1. jQuery功能
2. 选择页面DOM对象
3. 修改页面显示内容(增加|删除|修改)
4. 修改页面显示样式(增加|删除|修改)
5. 产生事件响应
6. 产生动画
7. 实现与服务端的通信(请求与响应)
8. 构建插件,提高代码集成度与重用性

示例:使用jQuery框架实现代码的简化(//注释部分是纯JS代码)

<script type="text/javascript">

//document.getElementById("title").innerText = "大家好";

$("#title").text("大家好")

//document.getElementById("head1").style.color = "red"

$("#head1").css("color","red")

//document.getElementById("info").innerHTML="<span style='color:red;' >Hello</span>"

//$("#info").html("<span style='color:red;' >Hello</span>")

$("<span>").css("color","red").text("Hello").appendTo("#info")

</script>

<div></div>

<script type="text/javascript">

//document.getElementsByTagName("div")[0].innerHTML = "---------";

$("div").eq(0).html("---------")

//document.getElementsByTagName("div")[1].innerHTML = "=========";

$("div").eq(1).html("=========")

//alert(document.getElementById("title").innerText)

alert($("#title").text())

//console.log(document.getElementById("title"))

console.log($("#title").get(0))

console.log($("#title")[0])

console.log($("#title"))

//console.log(document.getElementById("title").innerText)

console.log($("#title").text())

//console.log(document.getElementsByTagName("div"))

console.log($("div").get())

console.log($("div").get().length)

console.log($("div"))

console.log($("div").length)

</script>

示例: 使用jQuery框架实现事件响应

<script>

$("#head1").click(function(){

console.log("click")

console.log($(this).text())

})

$("#head1").dblclick(function(){

console.log("dlbclick")

animationDemo()

})

function animationDemo(){

console.log("animationDemo")

}

</script>

示例:使用jQuery框架实现动画

在animationDemo函数中追加如下代码,观察动画,说出有哪些动画?使用了哪些函数?

$("#head1")

.hide().show(1000).toggle(400).toggle(400)

.fadeOut(2000).fadeIn(2000).fadeToggle(400).delay(2000).fadeToggle(400)

.slideUp(1000).slideDown(1000).slideToggle(400).slideToggle(400);

以上jQuery函数调用方式是链式调用,前一个函数返回值仍然是一个jQuery对象.

思考:如何让body内所有的h2元素对象都动起来?

示例: 实现与服务端的通信(请求与响应)

1. 创建要访问的服务端资源

<table class="table table-bordered">

<thead>

<tr>

<th>Firstname</th>

<th>Lastname</th>

<th>Email</th>

</tr>

</thead>

<tbody>

<tr>

<td>John</td>

<td>Doe</td>

<td>john@example.com</td>

</tr>

<tr>

<td>Mary</td>

<td>Moe</td>

<td>mary@example.com</td>

</tr>

<tr>

<td>July</td>

<td>Dooley</td>

<td>july@example.com</td>

</tr>

</tbody>

</table>

1. 创建表格显示的容器元素和触发调用的按钮

<div class="container">

</div>

<button id="loadTable">装载表格</button>

1. 编写触发调用的代码

<script type="text/javascript">

$("button#loadTable").click(function(){

$.ajax({

url:"mytable.html",

dataType:"html",

success:function(html){

$("div.container").html(html)

}

})

})

</script>

注意:

1.从服务端获得的一般不是html代码,而是数据,格式如下所示:

[{

"Firstname": "John",

"Lastname": "Doe",

"Email": "john@example.com"

},

{

"Firstname": "Mary",

"Lastname": "Moe",

"Email": "mary@example.com"

},

{

"Firstname": "July",

"Lastname": "Dooley",

"Email": "july@example.com"

}

]

2.要使表格漂亮,可以使用Bootstrap框架

参看菜鸟教程-www.runoob.com/bootstrap4/表格.

Bootstrap功能、Bootstrap样式、Bootstrap可视化设计界面

Bootstrap是最流行和最强大的前端（HTML，CSS和JavaScript）框架，可以更快，更轻松地做响应式（自适应）Web开发。

* Bootstrap简介

Bootstrap最初是由Twitter的设计师和开发人员在2010年中期创建的。在成为开源框架之前，Bootstrap被称为Twitter蓝图。

Bootstrap是一个功能强大的前端框架，可以更快，更轻松地进行Web开发。它包括基于HTML和CSS的设计模板，用于创建常见的用户界面组件，如表单，按钮，导航，下拉菜单，警报，模态，选项卡，手风琴，轮播，工具提示等。

使用Bootstrap可以节省大量的时间和精力。

注意：默认情况下，Bootstrap 4具有响应性，采用移动优先方法。Bootstrap 4.3是Bootstrap最新，最稳定的版本。所有主要的现代浏览器都支持Bootstrap 4，例如Google Chrome，Firefox，Safari，Internet Explorer 10及更高版本等。

* 你可以用Bootstrap做什么

您可以轻松创建响应式网站。

您可以使用预定义的类快速创建多列布局。

您可以快速创建不同类型的表单布局。

您可以快速创建不同的导航栏变体。

您可以轻松创建手风琴，模态等组件，而无需编写任何JS代码。

您可以轻松创建动态选项卡来管理大量内容。

您可以轻松创建工具提示和弹出窗口以显示提示文本。

您可以轻松创建轮播或图像滑块来展示您的内容。

您可以快速创建不同类型的警报框。

其实远远不止这些，你可以用Bootstrap做很多其他有趣的事情。您将在后续章节中详细了解所有这些内容。

* 使用Bootstrap的优点

如果您对任何前端框架有过一些经验，那么您可能想知道是什么让Bootstrap如此特别。以下是为什么应该选择Bootstrap框架的一些优点：

节省大量时间 - 使用Bootstrap预定义设计模板和类可以节省大量时间和精力，并专注于其他开发工作。

响应功能 - 使用Bootstrap，您可以轻松创建在不同设备和屏幕分辨率上更合适的响应式网站，而无需更改标记。

一致的设计 - 所有Bootstrap组件通过中央库共享相同的设计模板和样式，因此网页的设计和布局将保持一致。

易于使用 - Bootstrap非常易于使用。任何具有HTML，CSS和JavaScript基本工作知识的人都可以使用Bootstrap开始开发。

兼容浏览器 - Bootstrap是在考虑现代Web浏览器的情况下创建的，它与所有现代浏览器兼容，如Chrome，Firefox，Safari，Internet Explorer等。

开源 - 最好的部分是，它可以完全免费下载和使用。

* 快速部署

使用 Bootstrap CDN嵌入4行代码就能完成导入！ [点此这里下载离线包及源码](https://code.z01.com/v4/docs/download.html)。

使用Bootstrap框架的模板：

<!doctype html>

<html lang="en">

<head>

<!-- Required meta tags -->

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<!-- Bootstrap CSS -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<title>Hello, world!</title>

</head>

<body>

<h1>Hello, world!</h1>

<!-- Optional JavaScript -->

<!-- jQuery first, then Popper.js, then Bootstrap JS -->

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>

</body>

</html>

思考：为什么将css导入放在前面，将js导入放在后面

全新的Bootstrap v4.4手册见<https://code.z01.com/v4/>

* Bootstrap可视化设计界面

菜鸟教程在Bootstrap3.X.X中提供了Bootstrap可视化设计界面,借助这个工具可以迅速的构成项目使用的界面.当然还有其他的工具软件实现Bootstrap的布局。

* Bootstrap3和Bootstrap4区别

首先引入Bootstrap3和4官网连接； [Bootstrap3](https://v3.bootcss.com/) VS [Bootstrap4.](https://v3.bootcss.com/)

不同点



注：

Bootstrap3的4种栅格：

特小（col-xs-） 适配手机(<768px)

小（col-sm-） 适配平板(≥768px)

中（col-md-） 适配电脑(≥992px)

大（col-lg-） 适配宽屏电脑(≥1200px)

Bootstrap4的5种栅格：

特小（col-）(<576px)

小（col-sm-）(≥576px)

中（col-md-）(≥768px)

大（col-lg-） (≥992px)

特大（col-xl-）（≥1200px）

Bootstrap4特点

新增网格层适配了移动端；

全面引入ES6新特性（重写所有JavaScript插件）；

css文件减少了至少40%；

所有文档都用Markdown编辑器重写；

放弃对IE8的支持

* Bootstrap组件

组件提示

按钮点击状态、勾(复)选框

幻灯片、指标器

折叠面板控制内容是否显示

下拉菜单、显示定位事件(主要[Popper.js](https://popper.js.org/))

显示模态和浏览器侦听

导航条以及响应式适配

工具提示和移动事件 (主要 [Popper.js](https://popper.js.org/))

滚动侦听和导航事件

下面通过浏览一下Bootstrap4的功能

菜鸟教程:

<https://www.runoob.com/bootstrap4/bootstrap4-tutorial.html>

中文文档:

<https://getbootstrap.net/docs>

Bootstrap 4 Flex（弹性）布局

|  |  |
| --- | --- |
| 类(\*:sm, md, lg 或 xl) | 特点 |
| 弹性容器 | |
| .d-\*-flex | 创建弹性盒子容器(行块) |
| .d-\*-inline-flex | 创建行内弹性盒子容器(行内块) |
| 方向 | |
| .flex-\*-row | 在水平方向显示弹性子元素(左到右) |
| .flex-\*-row-reverse | 在水平方向显示弹性子元素(右到左) |
| .flex-\*-column | 在垂直方向显示弹性子元素(上到下) |
| .flex-\*-column-reverse | 在垂直方向显示弹性子元素(下到上) |
| 内容对齐 | |
| .justify-content-\*-start | 在开始位置显示弹性子元素 (左|上) |
| .justify-content-\*-end | 在尾部显示弹性子元素 (右|下) |
| .justify-content-\*-center | 在 flex 容器中居中显示子元素(中) |
| .justify-content-\*-between | 使用 "between" 显示弹性子元素(中间等宽间隔) |
| .justify-content-\*-around | 使用 "around" 显示弹性子元素(周围等宽间隔) |
| 等宽 | |
| .flex-\*-fill | 强制等宽(等宽充满容器) |
| 扩展 |  |
| .flex-\*-grow-0 | 不设置扩展 |
| .flex-\*-grow-1 | 设置扩展(指定部分充满剩余) |
| 收缩 | |
| .flex-\*-shrink-0 | 不设置收缩 |
| .flex-\*-shrink-1 | 设置收缩 |
| 包裹 | |
| .flex-\*-nowrap | 不设置包裹元素(所有元素不换行|列) |
| .flex-\*-wrap | 设置包裹元素(所有元素正向填充,溢出时换行|列) |
| .flex-\*-wrap-reverse | 反转包裹元素(所有元素逆向填充,溢出时换行|列) |
| 内容排列 | |
| .align-content-\*-start | 在起始位置堆叠元素(左|上) |
| .align-content-\*-end | 在结束位置堆叠元素(右|下) |
| .align-content-\*-center | 在中间位置堆叠元素 |
| .align-content-\*-around | 使用 "around" 堆叠元素 |
| .align-content-\*-stretch | 通过伸展元素来堆叠 |
| 元素对齐 | |
| .align-items-\*-start | 让元素在头部显示在同一行。 |
| .align-items-\*-end | 让元素在尾部显示在同一行。 |
| .align-items-\*-center | 让元素在中间位置显示在同一行。 |
| .align-items-\*-baseline | 让元素在基线上显示在同一行。 |
| .align-items-\*-stretch | 让元素延展高度并显示在同一行。 |
| 单独一个子元素的对齐方式 | |
| .align-self-\*-start | 让单独一个子元素显示在头部。 |
| .align-self-\*-end | 让单独一个子元素显示在尾部 |
| .align-self-\*-center | 让单独一个子元素显示在居中位置 |
| .align-self-\*-baseline | 让单独一个子元素显示在基线位置 |
| .align-self-\*-stretch | 延展一个单独子元素 |

JS基本数据类型、JS运算符与表达式

1. JS基本数据类型

JS的数据类型是弱类型,不需要明确指出数据类型,变量也不需要事先定义.

**JS基本数据类型,它们有字符串（String）、数字(Number)、布尔(Boolean)、对空（Null）、未定义（Undefined）。**

**JavaScript 字符串**

**字符串是存储字符（比如 "Bill Gates"）的变量。**

**字符串可以是引号中的任意文本。您可以使用单引号或双引号：**

**示例:**

**var carname="Volvo XC60";**

**var carname='Volvo XC60';**

**您可以在字符串中使用引号，只要不匹配包围字符串的引号即可：**

**示例:**

**var answer="It's alright";**

**var answer="He is called 'Johnny'";**

**var answer='He is called "Johnny"';**

**通过answer.length可以获得字符串的长度**

**x=new String("aaa");console.log(x.toString());console.log(x.substr(1));**

**思考:** **通过answer.length可以获得字符串的长度,为何还称字符串还是值类型?使用示例证明它.**

**x=new String("aaa");console.log(x.toString());y=x;y="bbb";console.log(x);**

**JavaScript 数字**

**JavaScript 只有一种数字类型。数字可以带小数点，也可以不带：**

**示例:**

**var x1=34.00; //使用小数点来写**

**var x2=34; //不使用小数点来写**

**极大或极小的数字可以通过科学（指数）计数法来书写：**

**示例:**

**var y=123e5; // 12300000**

**var z=123e-5; // 0.00123**

**var x= new Number;**

**JavaScript 布尔**

**布尔（逻辑）只能有两个值：true 或 false。**

**var x=true;**

**var y=false;**

**var y= new Boolean;**

**JavaScript 数组**

**下面用三种方法创建名为 cars 的数组：**

**方法一**

**var cars=new Array();**

**cars[0]="Saab";**

**cars[1]="Volvo";**

**cars[2]="BMW";**

**方法二**

**var cars=new Array("Saab","Volvo","BMW");**

**方法三**

**var cars=["Saab","Volvo","BMW"];**

**数组是值类型还是引用类型?**

**var cars=new Array("Saab","Volvo","BMW");**

**x=cars;**

**x.push("aaa");**

**console.log(x,cars);**

**同学测试一下,Array类型数据是值类型还是引用类型**

**//var a=[1,2];b=a;b[0]=23;console.log(a)**

**JavaScript 对象**

**对象由花括号分隔。在括号内部，对象的属性以名称和值对的形式 (name : value) 来定义。属性由逗号分隔：**

**person对象有三个属性：firstname、lastname 以及 id。**

**下面用三种方法创建名为 person对象,**

**方法一**

**var person={};**

**person.firstname="John";**

**person.lastname="Doe";**

**person.id=5566;**

**方法二**

**var person=new Object;**

**person.firstname="John";**

**person.lastname="Doe";**

**person.id=5566;**

**方法三**

**var person={"firstname":"John", "lastname":"Doe", "id":5566};**

**也可以写成如下格式:**

**var person={**

**"firstname" : "John",**

**"lastname" : "Doe",**

**"id" : 5566**

**};**

**对象属性有两种寻址方式：**

**name=person.lastname;**

**name=person["lastname"];//推荐：动态设置键值**

**同学测试一下,Object类型数据是值类型还是引用类型**

**Undefined 和 Null**

**Undefined 这个值表示变量不含有值。**

**可以通过将变量的值设置为 null 来清空变量。**

**Undefined 和 Null的区别**

**Undefined表示未定义,Null表示有值null**

**示例**

**console.log(!undefined,!null)**

**console.log(Number(undefined),Number(null))**

**console.log(5+undefined,5+null)**

**var x**

**console.log(x===undefined)**

**console.log(y===undefined)**

**var y=null**

**console.log(x===undefined)**

**console.log(y===undefined)**

**function f(x){console.log(x)}**

**f();**

**//NaN:是Not a Number缩写**

**===:不仅比较值,同时也比较类型**

**cars=null;//释放cars所引用的空间(如果原来引用具体对象)**

**函数Function**

**函数也是一种类型**

**var newFun = new Function("alert(1)");//或new Function(alert(1));**

**newFun();**

**使用typeof 表达式可以得到表达式的数据类型的字符串,不过是小写的.**

**示例:**

typeof {x:1}//结果: "object"

typeof [1,2] //结果: "object"

**使用instanceof 运算符用来测试一个对象在其原型链中是否存在一个构造函数的**prototype**属性。**

**示例:**

**new String() instanceof String//结果为true**

var obj = { }

var arr = [];

console.log(arr.constructor === Array)

console.log(obj.constructor === Array);

//用instanceof判断是否为数组，用typeof不可以。

console.log(arr instanceof Array);

console.log(obj instanceof Array);

问题：如何判断一种类型的数据是值类型还是引用类型？A=B,修改一方后，查看另一方，如果 两方的值相同，则该类型为引用类型。

JS String对象、JS RegExp对象

1. JS String对象

JavaScript 字符串用于存储和处理文本。

一个JS字符串可以看出一个字符数组，它可以按下标访问（读取）每一个字符元素，也可以获得字符串（字符数组）的长度，也可以对字符串做各种增删改查的操作。

JS字符串是数据显示、数据存储、数据转换、数据传输的重要方式。

JS字符串创建几种方法？

var x = "John";

var y = new String("John");

var z = y;

console.log("x.constructor=",x.constructor)

console.log("y.constructor=",y.constructor)

console.log("z.constructor=",z.constructor)

以上输出是相同的。x.constructor= ƒ String() { [native code] }

数据类型是值类型还是引用类型？

z = new String("Smith");

console.log("y=",y,"\tz=",z)//两者不等,说明String是值类型

console.log("y.toString()=",y.toString(),"\tz.toString()=",z.toString())//两者不等,说明String是值类型

不同构造方法的数据类型相同吗？

console.log("x=",x, "\ty=",y,"\ty.toString()=",y.toString())

console.log("typeof x=",typeof x)//typeof x= string

console.log("typeof y=", typeof y)//typeof y= object

console.log("x==y:",x==y)//x==y: true

console.log("x===y:",x===y)//x===y: false

能否根据下标读写某个字符？

console.log("x[0]=",x[0], "\ty[0]=",y[0])//x[0]= J y[0]= J

结果表明：根据下标能读某个字符

x[0]='j'

y[0]='j'

console.log("x[0]=",x[0], "\ty[0]=",y[0])//x[0]= J y[0]= J，

结果表明：不能根据下标改写某个字符,以上代码执行无效.

根据上面的结果来看，使用创建对象，它会拖慢执行速度，显示字符串的操作也不方便，并可能产生其他副作用。

既然是值类型，就直接将字符串常量付给字符串变量，来创建字符串，既简单有快捷。

字符串操作有自己一套方法

//字符串方法

console.log("abcd".charAt(0)) //a ===console.log("abcd"[0])

//返回指定索引位置的字符

console.log("字abcd".charCodeAt(0), "字abcd".charCodeAt(1)) //23383 97

//返回指定索引位置字符的 Unicode 值

console.log(String.fromCharCode(23383)) //字

console.log("abcbcd".indexOf("bc")) //1,-1表示未找到。

//返回字符串中检索指定字符第一次出现的位置

console.log("abcbcd".lastIndexOf("bc")) //3

//返回字符串中检索指定字符最后一次出现的位置

console.log("0123456".substr(2, 3)) //234,从起始索引号提取字符串中指定数目的字符

console.log("0123456".substring(2, 5)) //234,提取字符串中两个指定的索引号之间的字符

console.log("0123456".slice(2, 5)) //234移除字符串首尾空白

console.log("aa".concat("bb", "cc")) //aabbcc连接两个或多个字符串，返回连接后的字符串

console.log("I am a student.".toLowerCase()) //i am a student.把字符串转换为小写

console.log("I am a student.".toUpperCase()) //I AM A STUDENT.把字符串转换为大写

console.log(" abc\t\t ".trim()) //abc移除字符串首尾空白

console.log("张三,李四,王二".split(",")) //["张三", "李四", "王二"]把字符串分割为子字符串数组

console.log("张三,李四,王二".split(",",2)) //["张三", "李四"]把字符串分割为子字符串数组

//与正则表达式对象有关的方法

console.log("张三,李四,王二".replace(",", ";")) //张三;李四,王二 这里只替换了开始一处。

console.log("张三,李四,王二".replace(/,/g, ";")) //张三;李四;王二 这里全部替换了(g一定要加)。

console.log("cat,bat,fill,sat".match(/at/)) //["cat", "bat", "sat"]找到一个或多个正则表达式的匹配

console.log("cat,bat,fill,sat".match(/.at/)) //["cat", "bat", "sat"]找到一个或多个正则表达式的匹配

console.log("cat,bat,fill,sat".match(/.at/g)) //["cat", "bat", "sat"]找到一个或多个正则表达式的匹配

console.log("cat,bat,fill,sat".search(/at/)) //1 检索与正则表达式相匹配的位置

console.log("cat,bat,fill,sat".search(/.at/)) //0 检索与正则表达式相匹配的位置

1. JS RegExp对象

所有的高级编程语言都有正则表达式。

RegExp：是正则表达式（regular expression）的简写。它是JS中的一个类，正则表达式描述了字符的模式对象。

RegExp模式的作用

可以按模式检索的内容存在及其位置；

可以按模式全部替换其中的内容；

可以按模式检查其中的内容的格式；(yyyy-MM-dd)

可以按模式解析其中的各个部分；

var patt=new RegExp(pattern,modifiers);  
或更简单的方法  
var patt=/pattern/modifiers;

模式描述了一个表达式模型。

修饰符(modifiers)描述了检索是否是全局g，区分大小写i等。

RegExp 修饰符

修饰符用于执行不区分大小写和全文的搜索。

i(iglore) - 修饰符是用来执行不区分大小写的匹配。

g(globel) - 修饰符是用于执行全文的搜索（而不是在找到第一个就停止查找,而是找到所有的匹配）。

示例：看示例，学习知识点、技能点

//使用正则表达式完成简单的模式匹配和模式替换

var str = "Ab,ab,ac,ab,aB,AB";

console.log(str.match(/ab/));//["ab", index: 3, input: "Ab,ab,ac,ab,aB,AB", groups: undefined]

console.log(str.match( /ab/i));//["Ab", index: 0, input: "Ab,ab,ac,ab,aB,AB", groups: undefined]

console.log(str.match(/ab/g));//["ab", "ab"]

console.log(str.match(/ab/gi));//["Ab", "ab", "ab", "aB", "AB"]

console.log(str.match(/ab/ig));//["Ab", "ab", "ab", "aB", "AB"]

console.log(str.replace(/ab/g,"\*\*")) //Ab,\*\*,ac,\*\*,aB,AB

console.log(str.replace(/ab/gi,"\*\*"))//\*\*,\*\*,ac,\*\*,\*\*,\*\*

//正则对象的test()方法

//该方法接受一个字符串参数，判断该字符串是否匹配指定的模式，返回真或假。

var str1="192.168.2.110"

var re1 =/\d+\.\d+\.\d+\.\d+/ （注意：.表示任意字符，\.表示点）

console.log(re1.test(str1)); //true

var str2="192.168.2.110"

var re2 =/(\d+)\.(\d+)\.(\d+)\.(\d+)/

re2.test(str2); //

console.log(RegExp["$1"],RegExp["$2"],RegExp.$3,RegExp.$4);//192 168 2 110

/(\d+)-(\d+)-(\d+)/.test("Date：2020-02-18");

console.log(RegExp["$1"],RegExp["$2"],RegExp.$3);//2020 02 18(注意：语句结尾加分号;)

//exec() 方法检索字符串中的指定值。返回值是被找到的值。如果没有发现匹配，则返回 null。

var patt1=new RegExp("e");

console.log(patt1.exec("The best things in life are free"));

var patt1=new RegExp("el");

console.log(patt1.exec("The best things in life are free"));

//使用正则表达式完成复杂的模式匹配和模式替换(命名捕获分组)

var groups = "04-25-2017".match(/(?<month>\d{2})-(?<day>\d{2})-(?<year>\d{4})/).groups

console.log(groups);//{month: "04", day: "25", year: "2017"}

console.log(groups.year,groups.month,groups.day);//{month: "04", day: "25", year: "2017"}

//"04-25-2017"替换成"2017-04-25"

var str = "04-25-2017".replace(/(?<month>\d{2})-(?<day>\d{2})-(?<year>\d{4})/,"$<year>-$<month>-$<day>");

console.log(str)

//使用正则表达式完成复杂的字符串分割

var str="How are you\tdoing today?";

var n=str.split(" ");// ["How", "are", "you doing", "today?"]

var n=str.split(/\s/);// ["How", "are", "you", "doing", "today?"]

修饰符

修饰符用于执行区分大小写和全局匹配（i、g、m）；

方括号

方括号用于查找某个范围内的字符(如：[abc]、[a-c]、[^abc]等)

元字符

元字符（Metacharacter）是拥有特殊含义的字符（如\d、\w、\s等）

量词

表示数量关系(如?、\*、+、｛x｝、｛x,｝,{x,y})

具体请见：<https://www.runoob.com/jsref/jsref-obj-regexp.html>

JS RegExp 在线测试工具，请见<https://c.runoob.com/front-end/854>  
(它是针对所有编程语言的,如\d+,而不是/d+/)

示例：限定输入

涉及事件：onkeyup、oncontextmenu、onpaste

涉及方法：value=value.replace(re,'')

<input type="text" onkeyup ="value=value.replace(re,'')">

只能输入英文：/[^a-zA-Z]/g

只能输入数字：/\D/g

只能输入数字，小数点：/[^\d\.]/g

只能输入数字，小数点，下划线：/[^\d\.\_]/g

只能输入英文和数字：/[\W]/g

只能输入汉字：/[^\u4E00-\u9FA5]/g  
只能输入中文、英文、数字、@符号和.符号：/[^\a-\z\A-\Z0-9\u4E00-\u9FA5\@\.]/g

示例：表单正则验证（email、tel、url等）

<form method="get">

E-mail: <input type="email" required autofocus name="email" id="email" placeholder="您的电子邮箱">

<br />

<input type="submit" />

</form>

<input type="tel" name="mobile" pattern="^1[3458]{1}[0-9]{9}$" id="mobile" placeholder="您的手机号">

<input type="text" pattern="^[\u4e00-\u9fa5]\*$" required name="username" placeholder="请输入用户名">

[HTML5的表单所有type类型](https://www.cnblogs.com/dadayang/p/5749068.html)：

<https://www.cnblogs.com/dadayang/p/5749068.html>

JS Number对象、JS Math对象

一.Number 对象

Number 对象是原始数值的包装对象。

Number 创建方式 new Number()。

语法

var num = new Number(value);

注意： 如果一个参数值不能转换为一个数字时将返回 NaN (非数字值)。

示例:构造一个Number对象

var num1 = new Number(15);// Number {15}

var num2 = new Number("15");// Number {15}

var num3 = 15; // Number {15} 这种构造最简单

var num4 = new Number("15a");// Number {NaN} NaN=Not a Number

var num5 = new Number("a15");// Number {NaN}

它们都有constructor属性,且都为ƒ Number() { [native code] }

Number 对象属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| [constructor](https://www.runoob.com/jsref/jsref-constructor-number.html) | 返回对创建此对象的 Number 函数的引用。 | |
| [MAX\_VALUE](https://www.runoob.com/jsref/jsref-max-value.html) | 可表示的最大的数。1.7976931348623157e+308 | |
| [MIN\_VALUE](https://www.runoob.com/jsref/jsref-min-value.html) | 可表示的最小的数。5e-324 | |
| [NEGATIVE\_INFINITY](https://www.runoob.com/jsref/jsref-negative-infinity.html) | 负无穷大，溢出时返回该值。-Infinity | |
| [NaN](https://www.runoob.com/jsref/jsref-number-nan.html) | 非数字值。Not a Number | |
| [POSITIVE\_INFINITY](https://www.runoob.com/jsref/jsref-positive-infinity.html) | 正无穷大，溢出时返回该值。Infinity | |
| EPSILON | 最接近 1 且大于 1 的最小 Number 之间的差别。2.220446049250313e-16 | |
| [prototype](https://www.runoob.com/jsref/jsref-prototype-num.html) | 原型:允许您可以向对象添加属性和方法。 | |

Number 对象方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| [isFinite](https://www.runoob.com/jsref/jsref-isfinite-number.html) | 检测指定参数是否为有限的。类方法 | |
|  |  | |
| [toExponential(x)](https://www.runoob.com/jsref/jsref-toexponential.html) | 把对象的值转换为指数计数法。5678->"5.678e+3"  x:精度(小数部分位数)  实例方法:new Number(5678).toExponential(3) | |
| [toFixed(x)](https://www.runoob.com/jsref/jsref-tofixed.html) | 把数字转换为字符串。5.56789-> "5.57" x:精度2 | |
| [toPrecision(x)](https://www.runoob.com/jsref/jsref-toprecision.html) | 把数字格式化为指定的长度。13.3714-> "13.37" x:4 | |
| [toString(x)](https://www.runoob.com/jsref/jsref-tostring-number.html) | 把数字转换为字符串，使用指定的基数。11-> "1101" x:基数2 | |
| [valueOf()](https://www.runoob.com/jsref/jsref-valueof-number.html) | 返回一个 Number 对象的基本数字值。 | |

二.JS Math对象

Math不是类,而是对象, 因此没有构造函数,用于执行数学任务。

Math 对象属性(调用方法:Math. 属性名)

|  |  |
| --- | --- |
| 属性 | 描述 |
| [E](https://www.runoob.com/jsref/jsref-e.html) | 返回算术常量 e，即自然对数的底数（约等于2.718）。 |
| [LN2](https://www.runoob.com/jsref/jsref-ln2.html) | 返回 2 的自然对数（约等于0.693）。 |
| [LN10](https://www.runoob.com/jsref/jsref-ln10.html) | 返回 10 的自然对数（约等于2.302）。 |
| [LOG2E](https://www.runoob.com/jsref/jsref-log2e.html) | 返回以 2 为底的 e 的对数（约等于 1.4426950408889634）。 |
| [LOG10E](https://www.runoob.com/jsref/jsref-log10e.html) | 返回以 10 为底的 e 的对数（约等于0.434）。 |
| [PI](https://www.runoob.com/jsref/jsref-pi.html) | 返回圆周率（约等于3.14159）。 |
| [SQRT1\_2](https://www.runoob.com/jsref/jsref-sqrt1-2.html) | 返回 2 的平方根的倒数（约等于 0.707）。 |
| [SQRT2](https://www.runoob.com/jsref/jsref-sqrt2.html) | 返回 2 的平方根（约等于 1.414）。 |

Math 对象方法(调用方法:Math.方法名(实参表))

|  |  |
| --- | --- |
| 方法 | 描述 |
| [floor(x)](https://www.runoob.com/jsref/jsref-floor.html) | 对 x 进行向下取整。 |
| [round(x)](https://www.runoob.com/jsref/jsref-round.html) | 四舍五入取整。 |
| [ceil(x)](https://www.runoob.com/jsref/jsref-ceil.html) | 对x进行向上取整。 |
| [random()](https://www.runoob.com/jsref/jsref-random.html) | 返回 0 ~ 1 之间的随机数。  Math.floor(a+Math.[random()](https://www.runoob.com/jsref/jsref-random.html)\*b),作用:得到[a,b)的随机整数 |
| [abs(x)](https://www.runoob.com/jsref/jsref-abs.html) | 返回 x 的绝对值。 |
| [max(x,y,z,...,n)](https://www.runoob.com/jsref/jsref-max.html) | 返回 x,y,z,...,n 中的最高值。 |
| [min(x,y,z,...,n)](https://www.runoob.com/jsref/jsref-min.html) | 返回 x,y,z,...,n中的最低值。 |
| [exp(x)](https://www.runoob.com/jsref/jsref-exp.html) | 返回 Ex 的指数。 |
| [pow(x,y)](https://www.runoob.com/jsref/jsref-pow.html) | 返回 x 的 y 次幂。 |
| [log(x)](https://www.runoob.com/jsref/jsref-log.html) | 返回数的自然对数（底为e）。 |
| [sqrt(x)](https://www.runoob.com/jsref/jsref-sqrt.html) | 返回数的平方根。 |
| [acos(x)](https://www.runoob.com/jsref/jsref-acos.html) | 返回 x 的反余弦值。 |
| [asin(x)](https://www.runoob.com/jsref/jsref-asin.html) | 返回 x 的反正弦值。 |
| [atan(x)](https://www.runoob.com/jsref/jsref-atan.html) | 以介于 -PI/2 与 PI/2 弧度之间的数值来返回 x 的反正切值。 |
| [atan2(y,x)](https://www.runoob.com/jsref/jsref-atan2.html) | 返回从 x 轴到点 (x,y) 的角度（介于 -PI/2 与 PI/2 弧度之间）。 |
| [cos(x)](https://www.runoob.com/jsref/jsref-cos.html) | 返回数的余弦。 |
| [sin(x)](https://www.runoob.com/jsref/jsref-sin.html) | 返回数的正弦。 |
| [tan(x)](https://www.runoob.com/jsref/jsref-tan.html) | 返回角的正切。 |

JS Date对象

Date对象用于处理日期和时间。

一.构造Date对象

有四种方式初始化日期:

new Date() // 当前日期和时间

new Date(milliseconds) //返回从 1970 年 1 月 1 日至今的毫秒数

new Date(dateString)

new Date(year, month, day, hours, minutes, seconds, milliseconds)

示例: 用四种方式初始化日期对象

var today = new Date()

var d1 = new Date(1\*24\*60\*60\*1000)// 1970 年 1 月 2 日

var d2 = new Date("October 13, 1975 11:13:00")

var d2 = new Date("2020-2-19")

var d2 = new Date("2020/2/19")

var d3 = new Date(79,5,24)//1979-5-24

var d3 = new Date(2020,5,24,11,33,0)//2020-5-24 11:33:00

二.修改Date对象

示例: 为日期对象设置了一个特定的日期 (2010 年 1 月 14 日)：

var myDate=new Date();

myDate.setFullYear(2010,0,14);//0代表一月份

示例: 将日期对象设置为 5 天后的日期：

var myDate=new Date();

myDate.setDate(myDate.getDate()+5);

三. 比较Date对象

示例:比较两个日期。

var x=new Date();

x.setFullYear(2010,0,14);

var today = new Date();

console.log(x>today)//false

四.Date对象的取值

var x=new Date();

console.log(x.valueOf())//1582033094432(毫秒数)

var x=new Date();

console.log(x.valueOf())//1582033104274

x.setDate(x.getDate()+1)//1582119504274(1582033104274+1天的毫秒数)

从上面的例子看出,日期越后,数值就越大

思考:日期对象的值是什么含义?

var d1 = new Date("2020-2-19 00:00:00")

var d2 = new Date("2020-2-19 00:00:01")

d1.valueOf() //1582041600000

d2.valueOf() //1582041601000

从中可以看出,Date多1秒,值多1000

五.获取或设置Date对象的部分属性

部分属性有Year(2020-1900=120),FullYear(2020),Month(实际月份-1),Day(星期几),Date(日期),Hours(几时),Minutes(几分),Seconds(几秒),Milliseconds(几毫秒)等等

示例:在现有的日期基础上,得到向后延迟40天的日期

var myDate=new Date();

myDate.setDate(myDate.getDate()+40);

六.日期显示

示例:显示当前日期

console.log(new Date().toString())

// Tue Feb 18 2020 22:37:43 GMT+0800 (中国标准时间)

console.log(new Date().toLocaleString())

// 2020/2/18 下午10:35:43

示例:显示Date对象的星期

function myFunction(prefix){

var d = new Date();

var weekday="日一二三四五六";

return prefix+weekday.substr(d.getDay(),1);

}

演变成Date对象的方法

Date.prototype.weekDay = function(prefix){

var weekday="日一二三四五六";

return prefix+weekday.substr(this.getDay(),1);

}

调用: d1.weekDay("星期")

示例:更灵活的日期格式化方法

原来日期输出格式(toString方法,toLocaleString方法,toLocaleDateString方法,toLocaleTimeString方法)显然不符合本地习惯或者不灵活,必须为Date类添加一个新方法format

//  对Date的扩展，将 Date 转化为指定格式的String

//  年y、月M、日d、时h、分m、秒s、毫秒S、季q

//  毫秒(S)只能用 1 个占位符(是 1-3 位的数字)

//  例子：

//  (new Date()).Format("yyyy-MM-dd hh:mm:ss.S")

//  (new Date()).Format("yyyy-M-d h:m:s.S")  
//下面的代码可以复习String,Math,RegExp,Date,JSON,流程控制等

Date.prototype.format = function(fmt) {

var o = {

"M+": this.getMonth() + 1, // 月份,this代表当前Date对象

"d+": this.getDate(), // 日

"h+": this.getHours(), // 小时

"m+": this.getMinutes(), // 分

"s+": this.getSeconds(), // 秒

"q+": Math.floor((this.getMonth() + 3) / 3), // 季度

"S": this.getMilliseconds() // 毫秒

};

if (/(y+)/.test(fmt))//处理y年

fmt = fmt.replace(

RegExp.$1, (this.getFullYear() + "").substr(4 - RegExp.$1.length)

);

for (var k in o)//处理年份之外的

if (new RegExp("(" + k + ")").test(fmt))

fmt = fmt.replace(

RegExp.$1,

(RegExp.$1.length == 1) ?

o[k] :

("00" + o[k]).substr(("" + o[k]).length)

);

return fmt;

}

因为上面的代码通用性很强,可以把它放在工具类文件中(utils.js),在需要的时候可以导入到页面中.

调用：

console.log(new Date().Format("yyyy-MM-dd"));//(y+)-(M+)-(d+)

console.log(new Date().Format("yyyy-MM-dd hh:mm:ss"));

console.log(new Date().Format("yyyy-MM-dd hh:mm:ss.S"));

console.log(new Date().Format("yyyy年第q季度"));

JS Array对象、JS JSON对象

JS中Array对象与JSON对象是构造复杂数据的基础.前面已经提及JS中Array对象与JSON对象都是引用类型数据.（x=y,修改了y，x也会被修改）

一.JS Array对象与JS JSON对象的构造

Array对象与JSON对象最简单的构造方法如下面示例所示:

var arr = [1,2,5];//无数据,如[] 作用: 迅速由下标找值

var json = {x:100,y:200};//无数据,如{} 作用: 迅速由键找值

二. JS Array对象与JS JSON对象的数据访问

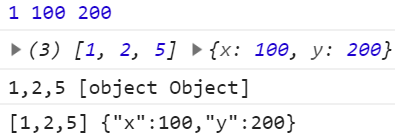
console.log(arr[0],json.x,json["y"])

console.log(arr,json);

console.log(arr.toString(),json.toString());

console.log(JSON.stringify(arr),JSON.stringify(json));

结果如下:



说明:

1. JSON对象成员(键值对)的访问有两种运算符(成员运算符.和下标运算符[]).下标运算符适合在动态设置键名.
2. Array对象成员(元素)的显示与JSON对象成员(键值对)显示都可以采用下标运算符[],只不过JSON对象成员(键值)要用字符串(即要加引号).
3. 用字符串显示整个对象数据最好的方法是JSON.stringify(对象),这也叫序列化操作(由对象变成字符串).
4. 反序列化操作(由字符串变成对象). JSON.parse(字符串).

三. JS Array对象与JS JSON对象的数据添加

JSON对象的数据添加(即键值对的添加),不存在顺序问题.方法只有一种:

json.z=300;或json["z"] = 300;

Array对象数据添加存在顺序问题.方法较多

1.在指定位置添加:

arr[3]=10;//可以在任意位置添加,这是弱类型的特点.

2.在末尾添加

arr.push(20);//压栈或入队

3.在开头添加

arr.unshift(100);

4.在中间添加

arr.splice(3,0,30)

以上方法可以在下标3处插入一个值为33的元素.

四. JS Array对象与JS JSON对象的数据修改

数据的修改可以先访问到成员,然后赋值给它即可.

json.z=301;或json["z"] = 301;

arr[3]=11;

对于数组还可以用如下方法实现修改,只是稍微麻烦一些.

arr.splice(3,1,31)//插入是 arr.splice(3,0,30)

五. JS Array对象与JS JSON对象成员数据的删除

JSON对象中键值对的删除使用delete运算符,例如:

delete json.x;//或

delete json["x"]

Array对象元素的删除与位置有关(不能用delete运算符).

1.删除首元素

arr.shift()//出队

2.删除尾元素

arr.pop()//弹栈

3.删除中间元素

arr.splice(1,1)// 删除下标为1的1个元素

六. JS Array对象与JS JSON对象成员数据的枚举

Array对象与JSON对象都可以用for-in语句实现成员数据的枚举.

|  |  |
| --- | --- |
| 代码 | 结果 |
| for(var i=0;i<arr.length;i++){  console.log(i,arr[i])  }  for(var i in arr){  console.log(i,arr[i])  }  for(var i in json){  console.log(i,json[i])  } |  |

注意:

对于数组的元素的迭代,最好用for(var i=0;i<arr.length;i++){}更为妥当.

arr.forEach(function(item,i){console.log(item,i)})

arr.forEach(item=>{console.log(item\*item)})

七. JS Array对象的其它操作

1.多个Array对象的合并

var parents = ["Jani", "Tove"];

var brothers = ["Stale", "Kai Jim", "Borge"];

var children = ["Cecilie", "Lone"];

var family = parents.concat(brothers, children);

2.Array对象元素连接成字符串

var fruits = ["Banana", "Orange", "Apple", "Mango"];

var str = fruits.join(";")//结果为"Banana;Orange;Apple;Mango"

var str = "<ul><li>"+ fruits.join("</li><li>")+"</li></ul>"

//默认的连接串为逗号.

3.数组排序

升序排列

var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.sort()//结果为["Apple", "Banana", "Mango", "Orange"]

降序排列

fruits.sort(function(a,b){return -(a>b?1:a==b?0:-1);})//或

fruits.sort(function(a,b){return -(a>b?1:-1);})

//结果为["Orange", "Mango", "Banana", "Apple"]

数组逆序

fruits.reverse()//结果为["Apple", "Banana", "Mango", "Orange"]

4.从数组中选择部分区段元素

var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];

var citrus = fruits.slice(1,3);//结果为["Orange", "Lemon"]

5.数组的多功能方法splice (取决于第2,3个参数)

数组对象.splice(start,?deleteCount,...insertItems)

删除: insertItems.length=0,deleteCount>0时,删除start开始的deleteCount个元素;

插入: insertItems.length>0, deleteCount=0时,在start位置插入insertItems.length个新元素;

替换: insertItems.length>0, deleteCount>0时,将start到start+ deleteCount-1的元素用insertItems.length个新元素替换.

示例:在fruits = ["Banana", "Orange", "Lemon","Apple"];情况下分别进行如下操作

fruits.splice(1,2);

//结果为["Orange", "Lemon"]

fruits.splice(1,0,"Mango");

//结果为["Banana", "Mango", "Orange", "Lemon", "Apple"]

fruits.splice(1,1,"Mango");

//结果为["Banana", "Mango", "Lemon", "Apple"]

fruits.splice(1,1,"Mango","Pear");

//结果为["Banana", "Mango", "Pear", "Lemon", "Apple"]

特殊用法

fruits.splice();

//结果为["Banana", "Orange", "Lemon", "Apple"],相当于什么没做

fruits.splice(1);

//结果为["Banana"],删除1开始的所有元素

fruits.splice(-1);

//结果为["Banana", "Orange", "Lemon"],删除从后开始的第1开始的所有元素.

学习素材:

var records = [{

"Firstname": "John",

"Lastname": "Doe",

"Email": "john@example.com"

},

{

"Firstname": "Mary",

"Lastname": "Moe",

"Email": "mary@example.com"

},

{

"Firstname": "July",

"Lastname": "Dooley",

"Email": "july@example.com"

}

]

JS流程控制

与其他语言一样,JS流程控制结构也分为顺序\选择\循环三种结构.

语句与C相似

顺序:输出语句console.log(),document.write(),window.alert()等

DOM结构操作,DOM节点创建中内层节点可以添加到外层节点中,或外层节点中添加内层节点,先有外层节点才有内层节点;删除节点可以直接从待删顶层节点删除.

选择:条件表达式?:适合不同条件得到不同值的情况;if单分支;if-else双分支;if-else if-…else有序多分支;switch-case-break无序多分支.

循环:适用于数组对象\JSON对象的迭代处理.for计数循环;while当循环;do-while直到循环;用数组的迭代方法forEach()、map()、reduce()、filter()对所有元素进行迭代处理.

下面通过示例讲解数组迭代方法.

var arr=[78,90,94,65,50,97,92];

var sum1=0;

//求和

arr.forEach(function(item){sum1+=item;})//每个元素单独操作

console.log(sum1)

sum2 = arr.reduce(function(pre,cur,array,index){return pre+cur});//除首元素外的每个元素单独操作，pre被叠加了，最后返回pre。

console.log(sum2)

sum3 = arr.reduce((pre,cur)=>pre+cur)

console.log(sum3)

var sum4 = 0

var arr2 = arr.map((cur,index,array)=>sum4+=cur)//每个元素单独操作，得到长度相同的数组，返回与元素相关的值替换原来元素。

console.log(sum4,arr2)

//过滤元素

var newArr = arr.filter(item => item>=60)//每个元素单独操作，含元素的表达式为true时该元素被选中。

console.log(arr) // [20,30,50, 96,50]

console.log(newArr) // [50, 96, 50]

//求最小值

min = arr.reduce((p,c)=>p<c?p:c)

console.log(min);

//获得各个元素除10向下取整后的数组

arr2 = arr.map(function(item){return Math.floor(item/10)})

console.log(arr2)

arr3 = arr.map(x=>Math.floor(x/10))

console.log(arr3)

//统计每个姓名出现的次数

var names = ['Alice', 'Bob', 'Tiff', 'Bruce', 'Alice'];

//使用键值对保存每个姓名出现的次数

jsonNames = {};

//使用for语句

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

var name =names[i];

if(jsonNames[name] == undefined){

jsonNames[name]=1;

}else{

jsonNames[name]++;

}

}

console.log(jsonNames);

//使用forEach方法

jsonNames = {};

names.forEach((x)=>{jsonNames[x]=(jsonNames[x]==undefined?1:jsonNames[x]+1)})

console.log(jsonNames);

//使用map方法

jsonNames = {};

names.map((c)=>{jsonNames[c]=(jsonNames[c]==undefined?1:jsonNames[c]+1)})

console.log(jsonNames);

//使用reduce方法

jsonNames = {};jsonNames[names[0]]=1;//处理第一个元素

names.reduce((p,c)=>{jsonNames[c]=(jsonNames[c]==undefined?1:jsonNames[c]+1)})

console.log(jsonNames);

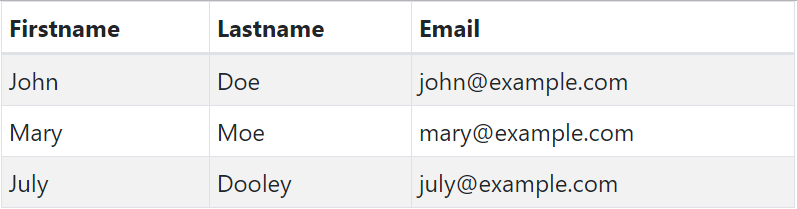
本次要求不适合filte(),但因为每个元素都参与操作，也能实现本题要求。

jQuery中有一个关于数组的操作var arr = $.each(array,function(x){});

数组各种迭代方式的比较

|  |  |  |  |
| --- | --- | --- | --- |
| 迭代方式 | 参与元素 | 特点 | 返回值 |
| for语句 | 可定制，一般为全部元素 | 通用，但不简洁 | 无 |
| forEach方法 | 全部元素 | 通用，比for语句简洁 | 无 |
| map方法 | 全部元素 | 对元素改值 | 返回相同长度的数组 |
| reduce方法 | 除首元素外的全部元素 | 对元素进行迭代操作 | 返回一个值 |
| filter方法 | 全部元素 | 对元素进行过滤 | 返回子数组 |
| $.each | 全部元素 | 通用，array对象作为第1个参数，回调函数作为第2个参数。 | 返回array对象 |

示例：使用上次课的records数据（将它放在js/records.js文件中）,构建一个如图所示的Bootstrap表格.



表格所用样式如下:

class="table table-striped table-bordered table-hover table-sm"

不变标签(初始结构)如下:

<table>

<thead><tr></tr></thead>

<tbody></tbody>

</table>

读取数据：在文档加载结束时加载数据并生成表格

$(function() {

//设置table标签的样式

//读取服务端数据并生成表格

$.ajax({

url: "js/records.js",//服务端数据请求url

dataType: "json",

success: function(data) {

records = data;

//填充表头

//填充表体

}

})

})

使用jQuery产生DOM对象追加到上面初始结构中即可.

代码如下:

//设置table标签的样式

$("table")

.addClass("table table-striped table-bordered table-hover table-sm")

.addClass("mt-1")

//填充表头

for(var k in records[0]){

$("<th>").text(k).appendTo("table.table>thead>tr")

}

//填充表体

for(var i in records){

var $tr = $("<tr>");

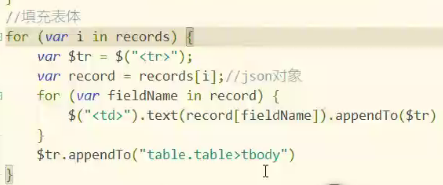
for(var k in records[i]){

$("<td>").text(records[i][k]).appendTo($tr)

}

$tr.appendTo("table.table>tbody")

}



以上for循环语句实现的可否改成其它Array对象方法实现呢？

函数定义与调用

一、函数的意义

函数是代码的模块单位，意义在于代码封装，将一组有意义的代码块封装起来，供其它程序调用或用作事件处理的回调函数，提高其重用性，提高团队项目开发的效率。

前面学习的数组对象的大量方法就使用了函数的封装，只是函数前面加了:

Array.prototype.

二、函数的定义

函数定义的关键字function

函数定义必须使用function，格式如下：

function 函数名(函数参数名列表)

{

//局部变量的定义var

//核心代码

//函数返回return或return 表达式

}

示例：获得数组所有元素的最大值

function max(arr) {//函数名max,参数名列表arr

//局部变量的定义var

var \_max = Number.MIN\_VALUE;

//核心代码

if (arr == undefined || arr.length == 0) {

return \_max;

}

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

if (arr[i] > \_max) \_max = arr[i]

}

//函数返回return或return 表达式

return \_max;

}

函数调用：max([1,4,5,3])

\_max是函数体内用var定义的局部变量，在函数体外是引用不到的。

console.log(\_max);//Uncaught ReferenceError: \_max is not defined

如果参数不是数组对象，如何去处理多个无名参数的最大值？

做法很简单只需要在函数体中添加如下一条语句（其它地方不变）即可:

var arr = arguments;//用arguments去接受调用时提供的所有参数。

函数调用：max(1,4,5,3)

示例：将上次课表格生成代码进行封装

function fillTable(records) {

//初始化

$("table.table>thead>tr").empty();

$("table.table>tbody").empty();

//边界处理

if (records == undefined || records.length == 0) {

$("<th>").text("表格没数据！").appendTo("table.table>thead>tr")

return;

}

//填充表头

for (var k in records[0]) {

$("<th>").text(k).appendTo("table.table>thead>tr")

}

//填充表体

for (var i in records) {

var $tr = $("<tr>");

for (var k in records[i]) {

$("<td>").text(records[i][k]).appendTo($tr)

}

$tr.appendTo("table.table>tbody")

}

}

$(function() {

//设置table标签的样式

$("table")

.addClass("table table-striped table-bordered table-hover table-sm")

.addClass("mt-1")

//读取服务端数据并生成表格

$.ajax({

url: "js/records.js", //服务端数据请求url

dataType: "json",

success: function(data) {

records = data;

fillTable(records);//函数调用

//试试fillTable()或fillTable([])来测试边界

}

})

})

示例：在数据表格中添加分页功能

首先从Bootstrap中获得分页的原型代码

<ul class="pagination">

<li class="page-item disabled"><a class="page-link" href="#">上页</a></li>

<li class="page-item active"><a class="page-link" href="#">1</a></li>

<li class="page-item"><a class="page-link" href="#">2</a></li>

<li class="page-item"><a class="page-link" href="#">3</a></li>

<li class="page-item"><a class="page-link" href="#">下页</a></li>

</ul>

界面如图：



分析：

分页的页码取决于记录总数（取决于总体记录数组records）和分页大小（pageSize）；

页面导航的状态取决于记录总数（页面总数pageCount）和当前页号(pageIndex)。

records、pageSize 、pageCount、pageIndex在页面中是一致的，可以作为全局变量，在页面中随处可以访问。

分页DOM中将<ul class="pagination"></ul>作为初始框架（或占位符）放在初始页码中，使用函数setPage实现li的添加。

提示：阅读程序时，先认真阅读程序注释。

var records;

var pageSize = 2;

var pageIndex = 1;

var pageCount = 0;

//文档加载结束时,设置table标签的样式,读取服务端数据，设置表格组件和分页组件

$(function() {

//设置table标签的样式

$("table")

.addClass("table table-striped table-bordered table-hover table-sm")

.addClass("mt-1")

//读取服务端数据，设置表格组件和分页组件

$.ajax({

url: "js/records.js", //服务端数据请求url

dataType: "json",

success: function(data) {

records = data;

pageCount = Math.ceil(records.length / pageSize);

setPage();//设置分页组件

pageIndex = 1;

$("li.page-item>a").eq(pageIndex).click();//触发首页导航

}

})

})

//设置分页组件

function setPage() {

//初始化分页组件

var $ul = $("ul.pagination").empty();

//添加“上页”导航按钮并实现事件响应

$('<li class="page-item"><a class="page-link" href="#">上页</a></li>')

.appendTo($ul)

.click(() => {

pageIndex = pageIndex > 1 ? pageIndex - 1 : pageIndex;

uiRefresh(); //函数调用

});

//添加页码导航按钮并实现事件响应

for (var i = 0; i < pageCount; i++) {

var $li = $('<li class="page-item"><a class="page-link" href="#"></a></li>')

.appendTo($ul);

$("a", $li).text(i + 1).click(function() {

pageIndex = eval(this.innerText);

uiRefresh(); //函数调用

});

}

//添加“下页”导航按钮并实现事件响应

$('<li class="page-item"><a class="page-link" href="#">下页</a></li>')

.appendTo($ul)

.click(() => {

pageIndex = pageIndex < pageCount ? pageIndex + 1 : pageIndex;

uiRefresh(); //函数调用

});

}

//触发分页导航后,进行表格数据和分页组件状态的更新

function uiRefresh(){

tableRefresh();

pageRefresh();

}

//表格数据的更新

function tableRefresh() {

//初始化

$("table.table>thead>tr").empty();

$("table.table>tbody").empty();

var pageRecords = records.slice((pageIndex - 1) \* pageSize, pageIndex \* pageSize);

//边界处理

console.log("pageRecords", pageRecords)

if (pageRecords == undefined || pageRecords.length == 0) {

$("<th>").text("表格没数据！").appendTo("table.table>thead>tr")

return;

}

//填充表头

for (var k in pageRecords[0]) {

$("<th>").text(k).appendTo("table.table>thead>tr")

}

//填充表体

for (var i in pageRecords) {

var $tr = $("<tr>");

for (var k in pageRecords[i]) {

$("<td>").text(pageRecords[i][k]).appendTo($tr)

}

$tr.appendTo("table.table>tbody")

}

}

//分页组件状态的更新

function pageRefresh(){

//先清除原来的选择

$("ul.pagination>li.active").removeClass("active");

//再设置新的选择

$("ul.pagination>li").eq(pageIndex).addClass("active");

//设置“上页”导航的可用性

if(pageIndex==1)

$("ul.pagination>li").eq(0).addClass("disabled");//再设置新的选择

else

$("ul.pagination>li").eq(0).removeClass("disabled");//再设置新的选择

//设置“下页”导航的可用性

if(pageIndex==pageCount)

$("ul.pagination>li").eq(pageCount+1).addClass("disabled");//再设置新的选择

else

$("ul.pagination>li").eq(pageCount+1).removeClass("disabled");//再设置新的选择

}

使用函数创建自定义类

1. 创建自定义类

在面向对象程序设计中，类包含静态的数据（字段）和动态的代码（方法）

* 示例:定义与引用一个二维点类

//定义二维点类

function Point2D(x,y){

this.x = x;

this.y = y;

this.toString = function(){

return JSON.stringify(this);

//这里的this与this.toString中的this是一致的

}

this.setX = function(x){

this.x = x;

}

this.setY = function(y){

this.y = y;

}

this.setXY = function(x,y){

this.x = x;

this.y = y;

}

}

//引用二维点类

var p = new Point2D(100,200);

console.log(p.toString());

p.setX(101);

console.log(p.toString());

p.setY(201);

console.log(p.toString());

p.setXY(102,202);

console.log(p.toString());

* 示例:定义与引用person类

function person(json){

var fields = {

"Id":"编号",

"Firstname":"姓",

"Lastname":"名",

"Email":"邮箱",

};

for(var field in fields){

this[field] = json[field];

}

this.toString = function(){

return JSON.stringify(this);

}

this.$tbody\_tr = function(){

$tr = $("<tr>");

for(var field in fields){

$("<td>").text(this[field]).appendTo($tr);

}

return $tr;

}

this.$thead\_tr = function(){

$tr = $("<tr>");

for(var field in fields){

$("<th>").text(fields[field]).appendTo($tr);

}

return $tr;

}

}

说明:

1. 函数中的this代表类的实例
2. This.成员 = 非function表达式时,成员为类的字段,表示静态数据
3. This.成员 = function时,成员为类的方法,表示动态代码,可以读写类中的静态数据.
4. 类的function中局部变量是不能在类之外被访问.
5. 当一个函数没有明确的调用对象的时候，也就是单纯作为独立函数调用的时候，将对函数的this使用默认绑定：绑定到全局的window对象.

//使用person类

$(function() {

//设置table标签的样式

$("table")

.addClass("table table-striped table-bordered table-hover table-sm")

.addClass("mt-1")

//读取服务端数据，设置表格组件和分页组件

$.ajax({

url: "js/records.js", //服务端数据请求url

dataType: "json",

success: function(data) {

records = data;

if(records.length>0){

var p = new person(records[0]); console.log(p.toString());//调用person类的方法

p.$thead\_tr().appendTo("table>thead")

}

for(var i in records){

var $tr = new person(records[i]).$tbody\_tr();

$tr.appendTo("table>tbody");

}

}

})

})

习题课1

本次习题课将通过的数据表较为完整操作的实现复习一些知识点:

1. jQuery操作
2. Array对象与Json对象的操作
3. 函数应用
4. Bootstrap应用

图标学习：

<https://www.runoob.com/try/demo_source/bootstrap3-glyph-icons.htm>

<http://www.jsons.cn/bootstrapicons/>

创建fonts文件夹，复制图标文件；创建css文件夹，创建glyphicons.css文件，将图标相关的样式类定义代码复制进去（来自<https://www.cnblogs.com/humi/p/8496152.html>）

.glyphicon .glyphicon-plus

.glyphicon .glyphicon-pencil

.glyphicon .glyphicon-remove