# HTML5 第一天笔记

## 基础内容

### 1. HTML5 简介

HTML5 是 HTML 标准的下一个重要版本，用来替代 HTML 4.01，XHTML 1.0 以及 XHTML 1.1。HTML5 也是一种在万维网上构建和呈现内容的标准。

HTML5 是万维网联盟（W3C）和网页超文本技术工作小组（WHATWG）合作的产物。

HTML5 是近十年来 Web 开发标准最巨大的飞跃。HTML5 并非仅仅用来表示 Web 内容，它将 Web 带入一个成熟的应用平台，在 HTML5 平台上，视频、音频、图象、动画，以及同电脑的交互都被标准化。

#### 1）HTML5 引入了许多新元素和属性帮助我们构建现代化的网站。下面是 HTML5 引入的主要特性：

* 新的语义化元素： 比如 <header>，<footer> 和 <section>。
* 表单 2.0： 改进了 HTML Web 表单，为 <input> 标签引入了一些新的属性。
* 持久的本地存储： 为了不通过第三方插件实现。
* WebSocket： 用于 Web 应用程序的下一代双向通信技术。
* 服务器推送事件： HTML5 引入了从 Web 服务器到 Web 浏览器的事件，也被称作服务器推送事件（SSE）。
* Canvas： 支持用 JavaScript 以编程的方式进行二维绘图。
* 音频和视频： 在网页中嵌入音频或视频而无需借助第三方插件。
* 地理定位： 用户可以选择与我们的网页共享他们的地理位置。
* 拖放： 把同一网页上的条目从一个位置拖放到另一个位置。

#### 2）HTML5 浏览器支持：

最新版 Apple Safari，Mozilla FireFox 和 Opera 支持大部分 HTML5 新特性，IE9 也支持一些 HTML5 的新功能。

预装在 iPhones，iPads 和 Android 手机上的手机浏览器都对 HTML5 有良好的支持。

**值得注意的是：** IE 8及之前版本浏览器基本不支持 HTML5 新特性。

### 2. HTML5 参考网站

<https://developer.mozilla.org/zh-CN/>

## 表单 2.0

### 1. Input 新类型

| **类型** | **描述** | **说明** |
| --- | --- | --- |
| email | 邮箱类型 | 判断当前字符串中是否包含@符号 |
| search | 搜索类型 |  |
| url | 网址类型 | 判断当前字符串中是否包含http:// |
| number | 数字类型 |  |
| tel | 电话号码类型 | 只在移动端浏览器有效 |
| range | 范围类型 | 适用于应该包含某个范围内数值的输入字段 |
| color | 颜色类型 |  |
| date | 日期类型 | 按照 ISO 8601 编码的日期（包括年，月，日） |
| month | 月份类型 | 由 ISO 8601 编码的年和月组成的日期 |
| week | 星期类型 | 由 ISO 8601 编码的年和星期数组成的日期 |

下述示例代码使用了 HTML5 提供的表单 2.0 中的 input 类型：

<form>

Email:<input type="email"><br>

搜索:<input type="search"><br>

URL地址:<input type="url"><br>

电话号码:<input type="tel"><br>

数字类型:<input type="number" min="18" max="50" step="2"><br>

范围类型:<input type="range" min="0" max="10" step="2" value="10"><br>

颜色类型:<input type="color"><br>

日期类型:<input type="date"><br>

<input type="submit">

</form>

#### 1）email类型

email 类型是专门用于输入 email 地址的文本框。

如果输入内容不是 email 地址，则不允许提交，并提示错误信息。效果如下图所示：



根据上图效果，总结 email 类型的规则是：提交表单时，检查 email 类型的文本框中是否包含 @ 符号。

**值得注意的是：** email 类型的文本框并不能检查 email 地址是否真实存在。

<input type="email" />

#### 2）url类型

url 类型是专门用于输入 url 地址的文本框。

如果输入内容不是 url 地址，则不允许提交，并提示错误信息。效果如下图所示：



根据上图效果，总结 url 类型的规则是：提交表单时，检查 url 类型的文本框中是否包含 http:// 符号。

<input type="url" />

#### 3）search类型

search 类型是专门用于输入搜索关键词的文本框。

在移动设备中，当 search 类型的元素获取焦点时，键盘的“前往”会改为“搜索”字样。效果如下图所示：

* iPhone设备：



* Android设备：



**值得注意的是：**在PC端浏览器 search 类型与 text 类型外观相同。

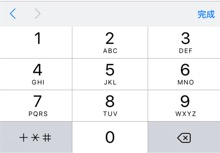
<input type="search" />

#### 4）tel类型

tel 类型是专门用于输入电话号码的文本框。

在移动设备中，当 tel 类型的元素获取焦点时，键盘使用数字面板。效果如下图所示：

* iPhone设备：



* Android设备：



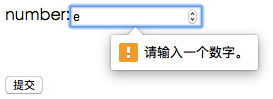
**值得注意的是：**该元素没有特殊的校验规则，不强制输入数字。

<input type="tel" />

#### 5）number类型

number 类型是专门用于输入数字的文本框。

如果输入内容不是数字，则不允许提交，并提示错误信息。效果如下图所示：



根据上图效果，总结 number 类型的规则是：提交表单时，检查 number 类型的文本框中是否为数字。

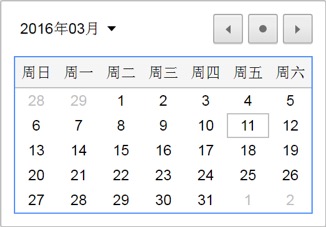
| **属性名称** | **类型** | **描述** |
| --- | --- | --- |
| min | Number | 允许输入的最小数值。 |
| max | Number | 允许输入的最大数值。 |
| step | Number | 设置增加或减少的间隔。（步长） |

<input type="number" min="0" max="100" step="5" />

#### 6）date类型

date 类型是专门用于输入日期，提供日期选择控件。

当用户单击向下箭头按钮时浏览器页面中显示日历。效果如下图所示：



<input type="date" />

#### 7）range类型

range 类型是专门用于输入一定范围内数字值。

range 类型在HTML页面显示为滑动条。效果如下图所示：

http://i2.piimg.com/567571/5db51979471825f8.png

| **属性名称** | **类型** | **描述** |
| --- | --- | --- |
| min | Number | 允许输入的最小数值。 |
| max | Number | 允许输入的最大数值。 |
| step | Number | 设置增加或减少的间隔。（步长） |
| value | Number | 设置滑动条的当前值。 |

<input type="range" min="0" max="100" step="5" value="50" />

#### 8）color类型

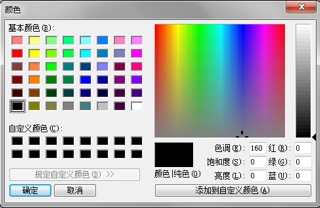
color 类型是专门用于选取颜色。

color 类型提供了一个颜色选取器。效果如下图所示：

* 显示效果：

http://i2.piimg.com/567571/ebbfc8a3555ab547.jpg

* 颜色选取器：



color 类型的显示效果，当用户单击 color 类型弹出显示颜色选取器。

<input type="color" />

#### 练习：自定义调色器

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>自定义调色器</title>

<style>

body {

padding-left: 100px;

padding-top: 100px;

}

.show {

width: 300px;

height: 300px;

border: 1px solid black;

}

</style>

</head>

<body>

<div class="show"></div>

<br><br>

红:<input onchange="mychange()" id="red" min="0" max="255" step="1" value="255" type="range"><br>

绿:<input onchange="mychange()" id="green" min="0" max="255" step="1" value="255" type="range"><br>

蓝:<input onchange="mychange()" id="blue" min="0" max="255" step="1" value="255" type="range">

<script>

function mychange(){

var red = document.getElementById("red").value;

var green = document.getElementById("green").value;

var blue = document.getElementById("blue").value;

var show = document.getElementsByTagName("div")[0];

show.style.backgroundColor = "rgb("+red+","+green+","+blue+")";

}

</script>

</body>

</html>

### 2. 表单新元素

| **元素名称** | **描述** |
| --- | --- |
| <datalist> | <input>标签定义选项列表。请与 input 元素配合使用该元素，来定义 input 可能的值 |
| <progress> | <progress> 标签定义运行中的任务进度（进程） |
| <meter> | <meter> 标签定义度量衡。仅用于已知最大和最小值的度量 |
| <output> | <output> 标签定义不同类型的输出，比如脚本的输出 |

#### 1）datalist元素

<datalist> 元素规定输入域的选项列表。

<datalist> 属性规定 form 或 input 域应该拥有自动完成功能。当用户在自动完成域中开始输入时，浏览器应该在该域中显示填写的选项：

使用 <input> 元素的列表属性与 <datalist> 元素绑定：

<input list="browsers">

<datalist id="browsers">

<option value="Internet Explorer">

<option value="Firefox">

<option value="Chrome">

<option value="Opera">

<option value="Safari">

</datalist>

#### 2）progress元素

<progress> 标签表示进度条，用于显示一个任务的完成进度。

**属性：**

| **属性** | **值** | **描述** |
| --- | --- | --- |
| max | number | 规定需要完成的值 |
| value | number | 规定进程的当前值 |

**提示：**请将 <progress> 标签与 JavaScript 一起使用来显示任务的进度。

**值得注意的是：**<progress> 标签不适合用来表示度量衡（例如，磁盘空间使用情况或相关的查询结果）。表示度量衡，请使用<meter> 标签代替。

<progress id="progress" max="100" value="0"></progress>

<script>

var progress = document.getElementById("progress");

var max = progress.max;

var t = setInterval(function(){

var v = progress.value;

if(v == max){

clearInterval(t);

}else{

v++;

progress.value = v;

}

},100);

</script>

#### 3）meter元素

<meter> 标签定义度量衡。仅用于已知最大和最小值的度量。

比如：磁盘使用情况，查询结果的相关性等。

**属性：**

| **属性** | **值** | **描述** |
| --- | --- | --- |
| high | number | 规定被界定为高的值的范围 |
| low | number | 规定被界定为低的值的范围 |
| max | number | 规定范围的最大值 |
| min | number | 规定范围的最小值 |
| value | number | 必需。规定度量的当前值 |

**值得注意的是：**<meter> 不能作为一个进度条来使用， 进度条 <progress> 标签。

<meter min="0" max="100" value="95" low="10" high="90"></meter>

#### 4）output元素

<output> 元素用于不同类型的输出，比如计算或脚本输出：

<form oninput="x.value=parseInt(a.value)+parseInt(b.value)">0

<input type="range" id="a" value="50">100 +

<input type="number" id="b" value="50">=

<output name="x" for="a b"></output>

</form>

### 3. 表单新属性

#### 1）placeholder属性

placeholder 属性提供一种提示（hint），描述输入域所期待的值。

简短的提示在用户输入值前会显示在输入域上。

**值得注意的是：**placeholder 属性适用于以下类型的 <input> 标签：text, search, url, telephone, email 以及password。

<input type="text" name="fname" placeholder="First name">

#### 2）autofocus属性

autofocus 属性是一个 boolean 属性。

autofocus 属性规定在页面加载时，域自动地获得焦点。

First name:<input type="text" name="fname" autofocus>

#### 3）multiple属性

multiple 属性是一个 boolean 属性。

multiple 属性规定<input> 元素中可选择多个值。

**值得注意的是：**multiple 属性适用于以下类型的 <input> 标签：email 和 file。

Email: <input type="email" multiple>

#### 4）form属性

form 属性规定输入域所属的一个或多个表单。

**值得注意的是：**如需引用一个以上的表单，请使用空格分隔的列表。

位于form表单外的input 字段引用了 HTML form (该 input 表单仍然属于form表单的一部分)：

<form action="demo-form.php" id="form1">

First name: <input type="text" name="fname"><br>

<input type="submit" value="Submit">

</form>

Last name: <input type="text" name="lname" form="form1">

## 表单验证

### 1. 验证属性

| **属性名称** | **描述** |
| --- | --- |
| required | 验证不为空 |
| pattern | 匹配正则表达式 |
| min和max | 验证最小值和最大值 |
| minlength和maxlength | 验证最小长度和最大长度 |
| validity | 获取validityState对象 |

#### 1）required属性

required 属性规定必须在提交之前填写输入域（不能为空）。

**值得注意的是：**required 属性适用于以下类型的 <input> 标签：text, search, url, telephone, email,password, date pickers, number, checkbox, radio 以及 file。

Username: <input type="text" name="usrname" required>

#### 2）pattern属性

pattern 属性的值与指定的正则表达式进行匹配（一般用于验证特定格式）。

**值得注意的是：**pattern 属性适用于以下类型的 <input> 标签: text, search, url, tel, email, 和password。

Country code: <input type="text" name="country\_code" pattern="[A-Za-z]{3}" title="Three letter country code">

#### 3）min和max属性

min、max 和 step 属性用于为包含数字或日期的 input 类型规定限定（约束）。

**值得注意的是：**min、max 和 step 属性适用于以下类型的 <input> 标签：date pickers、number 以及range。

Enter a date before 1980-01-01:

<input type="date" name="bday" max="1979-12-31">

Enter a date after 2000-01-01:

<input type="date" name="bday" min="2000-01-02">

Quantity (between 1 and 5):

<input type="number" name="quantity" min="1" max="5">

#### 4）maxlength属性

maxlength 属性用于设定允许输入的最大字符个数。

**值得注意的是：**maxlength 属性应该叫做限制属性。

### 2. 有效状态

ValidityState 代表了一个元素可以处于的与约束验证相关的有效性状态。

| **名称** | **类型** | **描述** |
| --- | --- | --- |
| valid | Boolean | 其他的约束验证条件都不为 true |
| valueMissing | Boolean | 该元素有 required 属性,但却没有值 |
| typeMismatch | Boolean | 该元素的值不符合元素类型所要求的格式(当 type 是 email 或者 url 时) |
| patternMismatch | Boolean | 该元素的值与指定的 pattern 属性不匹配 |
| stepMismatch | Boolean | 该元素的值不符合由 step 属性指定的规则 |
| rangeUnderflow | Boolean | 该元素的值小于指定的 min 属性 |
| tooLong | Boolean | 该元素的值的长度超过了 HTMLInputElement 或者 HTMLTextAreaElement 对象指定的maxlength 属性中的值 |
| rangeOverflow | Boolean | 该元素的值大于指定的 max 属性 |
| customError | Boolean | 该元素的自定义有效性消息已经通过调用元素的setCustomValidity() 方法设置成为一个非空字符串 |

<form>

用户名:<input id="user" type="text" required><br>

密码:<input id="pwd" type="text" pattern="^[0-9]{6,8}$"><br>

Email:<input id="mail" type="email"><br>

年龄:<input id="age" step="5" type="number" min="20"><br>

<input type="submit">

</form>

<script>

var user = document.getElementById("user");

user.onblur = function(){

if(user.validity.valid){

alert("验证通过.");

}else if(user.validity.valueMissing){

alert("用户名为空.");

}

}

var pwd = document.getElementById("pwd");

pwd.onblur = function(){

if(pwd.validity.valid){

alert("验证通过.");

}else if(pwd.validity.patternMismatch){

alert("密码输入有误.");

}

}

var mail = document.getElementById("mail");

mail.onblur = function(){

if(mail.validity.valid){

alert("验证通过.");

}else if(mail.validity.typeMismatch){

alert("Email输入有误.");

}

}

var age = document.getElementById("age");

age.onblur = function(){

if(age.validity.valid){

alert("验证通过.");

}else if(age.validity.rangeUnderflow){

alert("年龄过小.");

}else if(age.validity.stepMismatch){

alert("年龄有误.");

}

}

</script>

### 3. setCustomValidity() 方法

setCustomValidity() 方法设置自定义验证信息，用于即将实施与验证的约束来覆盖预定义的信息。

<form>

<input id="data" type="text" required>

<input type="submit">

</form>

<script>

var data = document.getElementById("data");

data.onblur = function(){

if(data.validity.valueMissing){

data.setCustomValidity("不能为空.");

}else if(data.validity.customError){

data.setCustomValidity("");

}

}

</script>

**原创声明：**如发布或转载请注明以下作者信息，谢谢！

本文作者 - 金云龙

联系方式 - QQ：2080954535；微信：imlonge；微信公众号：longestory

个人网站：<http://www.longestory.com>

# HTML5 第二天笔记

## 音视频处理

### 1. 视频处理

#### 1）基础内容

##### a. 目前实现网页视频播放的技术 - Flash

* Flash并不是浏览器原生支持(第三方组件)
* Flash的性能并不好
* 移动端不支持Flash技术

##### b. HTML 5 提供的视频处理技术 - <video>

* 提供了相对应的基本处理方式
* 提供了高级编程自定义方式

##### c. <video>元素所支持的视频格式

| **视频格式** | **类型** | **描述** |
| --- | --- | --- |
| MP4格式 | .mp4 | 目前最主流的视频格式。 |
| OGV格式 | .ogv |  |
| WEBM格式 | .webm | 是由Google公司推出的，目前唯一一个支持超高清的视频格式。 |

#### 2）如何使用<video>元素

##### a. 引入单个视频格式

<video> 标签的编写结构如下：

<video src="视频文件的路径" autoplay>

浏览器不支持的提示内容

</video>

例如以下示例代码：

<video src="../DATA/oceans-clip.mp4" autoplay width="640px" height="400px" style="background:black;">

非常抱歉,你的浏览器不支持该视频!

</video>

| **属性名称** | **类型** | **描述** |
| --- | --- | --- |
| width | Number | 设置视频的宽度。 |
| height | Number | 设置视频的高度。 |
| style |  | 设置CSS样式属性。 |
| class |  | 设置CSS样式属性。 |
| autoplay | Boolean | 设置是否自动播放视频。 |

##### b. 引入多个视频格式

<video> 标签和 <source> 标签的编写结构如下：

<video autoplay>

<source src="视频文件的路径" />

<source src="视频文件的路径" />

<source src="视频文件的路径" />

浏览器不支持的提示内容

</video>

例如以下示例代码：

<!-- 解决了浏览器对视频格式的兼容问题 -->

<video autoplay>

<!--

<source>元素

\* 引入视频文件(一个<video>元素允许包含多个<source>)

-->

<source src="../DATA/oceans-clip.mp4" />

<source src="../DATA/oceans-clip.ogv" />

<source src="../DATA/oceans-clip.webm" />

</video>

#### 3）<video>元素的属性

* autoplay属性：自动播放
* controls属性：提供控制面板

<video src="../DATA/oceans-clip.mp4" controls></video>

* loop属性：循环播放

<video src="../DATA/oceans-clip.mp4" autoplay loop></video>

* poster属性：播放之前实现一张图片

<video src="../DATA/oceans-clip.mp4" controls poster="../DATA/oceans-clip.png"></video>

* preload属性：预加载视频
  + none：不预加载
  + auto：默认值,尽快预加载
  + metadata：预加载除视频之外的内容(宽度、高度等)

#### 4）视频高级编程

##### a. 事件

| **事件名称** | **作用** |
| --- | --- |
| play | 视频播放时触发。 |
| pause | 视频暂停时触发。 |
| ended | 视频播放结束时触发。 |
| error | 视频播放错误时触发。 |

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>video元素的高级事件</title>

<style>

img {

position: absolute;

top : 66px;

left : 160px;

display : none;

}

</style>

</head>

<body>

<video id="mmedia" src="../DATA/oceans-clip.mp4" controls></video>

<img id="adv" src="../DATA/oceans-clip.png" width="320">

<script>

var mmedia = document.getElementById("mmedia");

var adv = document.getElementById("adv");

mmedia.addEventListener("pause",function(){

adv.style.display = "block";

});

mmedia.addEventListener("play",function(){

adv.style.display = "none";

});

</script>

</body>

</html>

##### b. 方法

| **方法名称** | **作用** |
| --- | --- |
| play() | 用于播放视频。 |
| pause() | 用于暂停视频。 |
| load() | 用于加载视频。 |
| canPlayType() | 判断当前浏览器是否支持当前视频格式。 |

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>video元素的高级方法</title>

</head>

<body>

<video id="mmedia" src="../DATA/oceans-clip.mp4"></video>

<br>

<input type="button" id="btn" value="播放">

<script>

var btn = document.getElementById("btn");

var mmedia = document.getElementById("mmedia");

btn.onclick = function(){

if(mmedia.paused){// 播放

mmedia.play();

this.value = "暂停";

}else{// 暂停

mmedia.pause();

this.value = "播放";

}

}

</script>

</body>

</html>

##### c. 属性

| **属性名称** | **作用** |
| --- | --- |
| paused | 如果视频为暂停或未播放时,返回true。 |
| ended | 如果视频播放完毕时,返回true。 |
| duration | 返回当前视频的时长。 |
| currentTime | 获取或设置视频的当前位置。 |

**参考内容：**[https://developer.mozilla.org/en-US/docs/Web/HTML/Supportedmediaformats](https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats)

### 2. 音频处理

#### 1）基础内容

##### a. 目前音频处理技术

* Flash技术也可以音频处理
* Media Player播放器允许嵌入在网页中

##### b. HTML5 提供的音频处理 - <audio>

* 浏览器原生支持
* 性能很好
* 移动端支持

##### c. <audio>元素支持的音频格式

| **音频格式** | **类型** | **描述** |
| --- | --- | --- |
| MP3格式 | .mp3 | 目前最主流的音频格式。 |
| OGG格式 | .ogg |  |
| WAV格式 | .wav |  |

#### 2）如何使用<audio>元素

##### a. 引入单个音频格式

<audio src="音频文件的路径" autoplay>

浏览器不支持的提示内容

</audio>

##### b. 引入多个音频格式

<audio autoplay>

<source src="音频文件的路径" />

<source src="音频文件的路径" />

<source src="音频文件的路径" />

浏览器不支持的提示内容

</audio>

#### 3）<audio>元素的特有属性

| **属性名称** | **作用** |
| --- | --- |
| autoplay | 设置是否自动播放。 |
| controls | 提供控制面板。 |
| loop | 设置循环播放。 |
| preload | 预加载视频。 |

#### 4）<audio>元素的高级编程

##### a. 事件

| **事件名称** | **作用** |
| --- | --- |
| play | 视频播放时触发。 |
| pause | 视频暂停时触发。 |
| ended | 视频播放结束时触发。 |
| error | 视频播放错误时触发。 |

##### b. 方法

| **方法名称** | **作用** |
| --- | --- |
| play() | 用于播放视频。 |
| pause() | 用于暂停视频。 |
| load() | 用于加载视频。 |
| canPlayType() | 判断当前浏览器是否支持当前视频格式。 |

##### c. 属性

| **属性名称** | **作用** |
| --- | --- |
| paused | 如果视频为暂停或未播放时,返回true。 |
| ended | 如果视频播放完毕时,返回true。 |
| duration | 返回当前视频的时长。 |
| currentTime | 获取或设置视频的当前位置。 |

## Canvas 画布

### 1. 基础内容

#### 1）Canvas技术允许在HTML页面直接绘制图形

* 不再需要引入外部图片(图形),HTML页面性能有所提高。
* 可以实现一些比较复杂的图形绘制工作。

#### 2）Canvas技术主要应用方向

* Web应用方面主要实现图表类。
* 网页游戏方面 - 主要实现2D效果。

#### 3）HTML5 提供有关图形方面的技术

* Canvas - 主要以2D为主
* WebGL - 主要以3D为主
* SVG - 矢量图

#### 4）Canvas的特点

* 绘制的图形与HTML页面之间是无关系的
* 通过Canvas绘制的图形不能使用DOM API
* 通过Canvas绘制的图形不能绑定事件
* Canvas画布最终是以图片(png|jpg)形式出现
* 绘制图形默认的颜色为黑色

### 2. 如何使用Canvas画布

#### 1）在HTML页面中定义<canvas>元素

* 设置<canvas>元素宽度和高度使用属性方式。
* 使用CSS样式方式或属性方式设置<canvas>元素的宽度和高度。

<!--

在HTML页面中,定义<canvas>元素

\* 默认只定义<canvas>元素时

\* 效果非常类似于<div>元素,但不一样的地方:

\* <div>元素在默认情况下,不具有高度和宽度的

\* <canvas>元素在默认情况下,具有高度和宽度的

\* 宽度 - 300px

\* 高度 - 150px

\* 设置<canvas>元素的高度和宽度

\* (建议)使用属性width和height

\* 使用CSS中的属性width和height

\* 绘制的图形会被拉伸

-->

<canvas id="canvas" width="500px" height="500px" style="background:pink;"></canvas>

<!--

<canvas id="canvas" style="width:500px;height:500px;"></canvas>

-->

#### 2）在JS代码中实现

* 获取<canvas>元素
* 通过<canvas>元素,创建画布对象
  + getContext('2d')函数
  + 返回画布对象
* 利用画布对象进行图形的绘制

// 获取HTML页面中的<canvas>元素

var canvas = document.getElementById("canvas");

/\*

通过<canvas>元素,创建画布对象

使用getContext(type)函数,创建画布对象

\* 该函数返回画布对象

\* type参数

\* 设置当前创建的画布是2d还是3d

\* 注意

\* 参数选项是2d(3d效果使用WebGL)

\* 必须写成"2d"

\*/

var context = canvas.getContext("2d");

// 利用画布对象,进行绘制图形

context.fillRect(10,10,100,100);

### 3. 绘制图形

#### 1）绘制矩形

##### a. 绘制实心(填充)矩形

fillRect(x,y,width,height)

| **参数名称** | **作用** |
| --- | --- |
| x和y | 绘制矩形的左上角的坐标值 |
| width | 设置绘制矩形的宽度(单位为px) |
| height | 设置绘制矩形的高度(单位为px) |

##### b. 绘制空心(描边)矩形

strokeRect(x,y,width,height)

| **参数名称** | **作用** |
| --- | --- |
| x和y | 绘制矩形的左上角的坐标值 |
| width | 设置绘制矩形的宽度(单位为px) |
| height | 设置绘制矩形的高度(单位为px) |

##### c. 清除指定区域的矩形

clearRect(x,y,width,height)

| **参数名称** | **作用** |
| --- | --- |
| x和y | 绘制矩形的左上角的坐标值 |
| width | 设置绘制矩形的宽度(单位为px) |
| height | 设置绘制矩形的高度(单位为px) |

<canvas id="canvas" width="500px" height="500px"></canvas>

<script>

// 1. 获取<canvas>元素

var canvas = document.getElementById("canvas");

// 2. 创建画布对象

var context = canvas.getContext('2d');

// 3. 绘制图形

// a. 绘制实心矩形

context.fillRect(10,10,100,100);

// b. 绘制空心矩形

context.strokeRect(120,10,100,100);

// c. 清除指定区域的矩形

context.fillRect(230,10,100,100);

context.clearRect(240,20,80,80);

</script>

#### 2）设置样式

| **属性名称** | **作用** |
| --- | --- |
| fillStyle | 设置填充样式 |
| strokeStyle | 设置描边样式 |
| globalAlpha | 设置透明度(0-1) |

**值得注意的是：**

* 一定要先设置样式(颜色),再绘制图形。
* 每次改变样式(颜色),重新设置。

<canvas id="canvas" width="500px" height="500px"></canvas>

<script>

var canvas = document.getElementById("canvas");

var context = canvas.getContext('2d');

// a. 设置填充样式

context.fillStyle = "pink";

// b. 绘制实心矩形

context.fillRect(10,10,100,100);

context.fillStyle = "blue";

context.fillRect(10,120,100,100);

// 设置描边样式

context.strokeStyle = "red";

context.strokeRect(120,10,100,100);

context.strokeStyle = "green";

context.strokeRect(120,120,100,100);

// 设置透明度

context.globalAlpha = 0.5;

context.fillRect(230,10,100,100);

context.fillStyle = "black";

context.globalAlpha = 0.1;

context.fillRect(230,120,100,100);

</script>

#### 3）设置渐变

##### a. 线性渐变

createLinearGradient(x1,y1,x2,y2)

线性渐变主要依靠基准线概念，所谓基准线就是设置线性渐变的标准。

| **参数名称** | **作用** |
| --- | --- |
| x1和y1 | 基准线的起点坐标值 |
| x2和y2 | 基准线的终点坐标值 |

<canvas id="canvas" width="500px" height="500px"></canvas>

<script>

var canvas = document.getElementById("canvas");

var context = canvas.getContext('2d');

/\*

设置线性渐变

createLinearGradient(x1,y1,x2,y2)方法

\* 该方法具有返回值,是渐变对象

\*/

var grd = context.createLinearGradient(0,0,100,100);

/\*

设置线性渐变的颜色和位置

addColorStop(position,color)

\* position - 设置颜色的位置

\* 值的范围为 0 - 1

\* color - 设置颜色

\*/

grd.addColorStop(0,"red");

grd.addColorStop(1,"blue");

grd.addColorStop(0.5,"yellow");

// 将设置的线性渐变,赋值给样式(fillStyle和strokeStyle)

context.fillStyle = grd;

// 绘制矩形

context.fillRect(0,0,100,100);

</script>

##### b. 射线(扇形)渐变

createRadialGradient(x1,y1,r1,x2,y2,r2)

射线渐变主要依靠基准圆概念，所谓基准圆就是设置射线渐变的标准。

| **参数名称** | **作用** |
| --- | --- |
| x1和y1 | 第一个基准圆的圆心 |
| r1 | 第一个基准圆的半径 |
| x2和y2 | 第二个基准圆的圆心 |
| r2 | 第二个基准圆的半径 |

<canvas id="canvas" width="500px" height="500px"></canvas>

<script>

var canvas = document.getElementById("canvas");

var context = canvas.getContext("2d");

/\*

设置射线渐变

createRadialGradient(x1,y1,r1,x2,y2,r2)

\* 该方法返回渐变对象

\*/

var grd = context.createRadialGradient(100,100,100,canvas.width,canvas.height,200);

/\*

设置渐变颜色

\*/

grd.addColorStop(0,"red");

grd.addColorStop(1,"blue");

// 将渐变对象赋值给样式

context.fillStyle = grd;

// 绘制矩形

context.fillRect(0,0,canvas.width,canvas.height);

</script>

##### c. 设置渐变颜色

addColorStop(position,color)

| **参数名称** | **作用** |
| --- | --- |
| position | 设置渐变颜色的位置，值的范围必须是0-1。 |
| color | 设置渐变的颜色 |

#### 4）绘制文字

| **属性名称** | **作用** |
| --- | --- |
| font | 设置文字的属性（用法同 CSS 属性font）。 |
| textAlign | 水平对齐方式。（left：基准线在左边；center：基准线在中间；right：基准线在右边） |
| textBaseline | 垂直对齐方式。（top：基准线在上边；middle：基准线在中间；bottom：基准线在下边；hanging：悬挂基线；alphabetic：字母基线） |

##### a. 绘制实心文字

fillText(text,x,y)

| **参数名称** | **作用** |
| --- | --- |
| text | 绘制的文字内容。 |
| x和y | 绘制文字的坐标值。 |

##### b. 绘制空心文字

strokeText(text,x,y)

| **参数名称** | **作用** |
| --- | --- |
| text | 绘制的文字内容。 |
| x和y | 绘制文字的坐标值。 |

<canvas id="canvas" width="500px" height="500px"></canvas>

<script>

var canvas = document.getElementById("canvas");

var context = canvas.getContext('2d');

// 设置文字样式

context.font = "bold 48px 宋体";

// 基准线

context.beginPath();

context.moveTo(100,0);

context.lineTo(100,400);

context.stroke();

// 设置水平对齐

context.textAlign = "right";

// 绘制文字

context.fillText("text",100,50);

// 设置水平对齐

context.textAlign = "center";

// 绘制文字

context.fillText("text",100,100);

// 设置水平对齐

context.textAlign = "left";

// 绘制文字

context.fillText("text",100,150);

// 基准线

context.beginPath();

context.moveTo(0,300);

context.lineTo(500,300);

context.stroke();

// 设置垂直对齐

context.textBaseline = "top";

context.strokeText("text",0,300);

context.textBaseline = "middle";

context.strokeText("text",100,300);

context.textBaseline = "bottom";

context.strokeText("text",200,300);

context.textBaseline = "hanging";

context.strokeText("text",300,300);

context.textBaseline = "alphabetic";

context.strokeText("text",400,300);

</script>

#### 5）设置阴影

| **属性名称** | **作用** |
| --- | --- |
| shadowColor | 设置阴影颜色。 |
| shadowOffsetX | 设置水平方向阴影。 |
| shadowOffsetY | 设置垂直方向阴影。 |
| shadowBlur | 设置阴影程度。 |

**原创声明：**如发布或转载请注明以下作者信息，谢谢！

本文作者 - 金云龙

联系方式 - QQ：2080954535；微信：imlonge；微信公众号：longestory

个人网站：<http://www.longestory.com>

# HTML5 第三天笔记

## Canvas绘图

### 1. 创建路径

图形的基本元素是路径。路径是通过不同颜色和宽度的线段或曲线相连形成的不同形状的点的集合。一个路径，甚至一个子路径，都是闭合的。使用路径绘制图形需要一些额外的步骤。

1. 首先，需要创建路径起始点。
2. 然后，使用画图命令去画出路径。
3. 之后，把路径封闭。
4. 一旦路径生成，你就能通过描边或填充路径区域来渲染图形。

#### 1）绘制矩形

##### 绘制矩形的实现步骤

1. 调用beginPath()方法，创建新建一条路径。
2. 使用rect(x, y, width, height)方法，设置矩形的坐标值及宽度和高度。
3. 通过fill()或stroke()方法进行绘制。

##### 使用方法说明

rect(x, y, width, height)

| **参数名称** | **作用** |
| --- | --- |
| x和y | 表示矩形的左上角坐标值。 |
| width和height | 表示矩形的宽度和高度。 |

#### 2）绘制圆形

##### 绘制圆形的实现步骤

1. 调用beginPath()方法，创建新建一条路径。
2. 使用arc(x, y, radius, startAngle, endAngle, anticlockwise)方法，设置矩形的坐标值及宽度和高度。
3. 通过fill()或stroke()方法进行绘制。

##### 使用方法说明

arc(x, y, radius, startAngle, endAngle, anticlockwise)

| **参数名称** | **作用** |
| --- | --- |
| x和y | 表示圆形的圆心坐标值。 |
| radius | 表示圆形的半径。 |
| startAngle | 表示绘制圆形的开始点，值为 0。 |
| endAngle | 表示绘制圆形的结束点，值为 Math.PI\*2。 |
| anticlockwise | 表示是以顺时针还是以逆时针方式绘制圆形，Boolean值。（默认值为false，表示顺时针。） |

#### 3）绘制弧形

##### 绘制圆形的实现步骤

1. 调用beginPath()方法，创建新建一条路径。
2. 使用arc(x, y, radius, startAngle, endAngle, anticlockwise)方法，设置矩形的坐标值及宽度和高度。
3. 通过fill()或stroke()方法进行绘制。

##### 使用方法说明

arc(x, y, radius, startAngle, endAngle, anticlockwise)

| **参数名称** | **作用** |
| --- | --- |
| x和y | 表示圆形的圆心坐标值。 |
| radius | 表示圆形的半径。 |
| startAngle | 表示绘制圆形的开始点（取值范围：0 至 Math.PI\*2。）。 |
| endAngle | 表示绘制圆形的结束点（取值范围：0 至 Math.PI\*2。）。 |
| anticlockwise | 表示是以顺时针还是以逆时针方式绘制圆形，Boolean值。（默认值为false，表示顺时针。） |

**值得注意的是：**

如果绘制的是空心弧形的话，在arc()方法调用后：

* 如果使用closePath()方法的话，绘制的图形会自动将终点和起点连接成线。
* 如果不用closePath()方法的话，绘制的图形会呈现开口状。

#### 4）绘制直线

1. 调用beginPath()方法，创建新建一条路径。
2. 使用moveTo(x, y)方法，设置直线的起点坐标值。
3. 使用lineTo(x, y)方法，设置直线的终点坐标值。
4. 通过stroke()方法进行绘制。

#### 5）绘制折线

1. 调用beginPath()方法，创建新建一条路径。
2. 使用moveTo(x, y)方法，设置直线的起点坐标值。
3. 使用lineTo(x, y)方法，设置直线的终点坐标值。
4. 通过stroke()方法进行绘制。

**值得注意的是：**在绘制折线的时候，lineTo()方法既可以绘制折点，也可以绘制终点。

#### 6）绘制多边形

1. 调用beginPath()方法，创建新建一条路径。
2. 使用moveTo(x, y)方法，设置直线的起点坐标值。
3. 使用lineTo(x, y)方法，设置直线的终点坐标值。
4. 调用closePath()方法，闭合当前绘制的路径。
5. 通过fill()或stroke()方法进行绘制。

### 2. 设置线型

所有画布操作都使用相同的线型，即默认线型。实际上线条的宽度、端点都可以根据实际绘图需要进行调整。

#### 1）设置线宽

lineWidth：指定线条粗细，默认值是1.0。

<canvas id="myCanvas" width="578" height="200"></canvas>

<script>

var canvas = document.getElementById('myCanvas');

var context = canvas.getContext('2d');

context.beginPath();

context.moveTo(100, 150);

context.lineTo(450, 50);

context.lineWidth = 15;

context.stroke();

</script>

#### 2）设置端点形状

lineCap：指定线条端点形状。

* butt：默认，向线条的每个末端添加平直的边缘。
* round：向线条的每个末端添加圆形线帽。
* square：向线条的每个末端添加正方向线帽。

**值得注意的是：**round和square会使线条略变微长。

<canvas id="myCanvas" width="578" height="200"></canvas>

<script>

var canvas = document.getElementById('myCanvas');

var context = canvas.getContext('2d');

// butt line cap (top line)

context.beginPath();

context.moveTo(200, canvas.height / 2 - 50);

context.lineTo(canvas.width - 200, canvas.height / 2 - 50);

context.lineWidth = 20;

context.strokeStyle = '#0000ff';

context.lineCap = 'butt';

context.stroke();

// round line cap (middle line)

context.beginPath();

context.moveTo(200, canvas.height / 2);

context.lineTo(canvas.width - 200, canvas.height / 2);

context.lineWidth = 20;

context.strokeStyle = '#0000ff';

context.lineCap = 'round';

context.stroke();

// square line cap (bottom line)

context.beginPath();

context.moveTo(200, canvas.height / 2 + 50);

context.lineTo(canvas.width - 200, canvas.height / 2 + 50);

context.lineWidth = 20;

context.strokeStyle = '#0000ff';

context.lineCap = 'square';

context.stroke();

</script>

#### 3）设置交点形状

lineJoin：指定两条线之间的连接点形状。

* round：创建圆角。
* bevel：创建斜角。
* miter：默认，创建尖角。

miterLimit：与lineJoin一起使用，当lineJoin设置为miter时，可用于确定线条交接点的延伸范围。

<canvas id="myCanvas" width="578" height="200"></canvas>

<script>

var canvas = document.getElementById('myCanvas');

var context = canvas.getContext('2d');

// set line width for all lines

context.lineWidth = 25;

// miter line join (left)

context.beginPath();

context.moveTo(99, 150);

context.lineTo(149, 50);

context.lineTo(199, 150);

context.lineJoin = 'miter';

context.stroke();

// round line join (middle)

context.beginPath();

context.moveTo(239, 150);

context.lineTo(289, 50);

context.lineTo(339, 150);

context.lineJoin = 'round';

context.stroke();

// bevel line join (right)

context.beginPath();

context.moveTo(379, 150);

context.lineTo(429, 50);

context.lineTo(479, 150);

context.lineJoin = 'bevel';

context.stroke();

</script>

## 处理图像

在HTML5中，不仅可以使用Canvas API来绘制图形，还可以读取磁盘或网络中的图像文件，然后使用Canvas API将该图像绘制在画布中。

### 1. 绘制图像

#### 1）加载图像

* 使用相同页面中的图片。
* 使用相同页面中的其他Canvas元素。
* 可以脚本通过Image( )构造函数创建图像。

#### 2）绘制图像

使用 drawImage() 方法绘制图像，该方法具体用法如下：

drawImage(img, x, y)

| **参数名称** | **作用** |
| --- | --- |
| img | 需要绘制的图像。 |
| x和y | 绘制图像的坐标值。 |

drawImage(img, x, y, width, height)

| **参数名称** | **作用** |
| --- | --- |
| img | 需要绘制的图像。 |
| x和y | 绘制图像的坐标值。 |
| width和height | 设置绘制图像的宽度和高度。 |

<canvas id="myCanvas" width="578" height="400"></canvas>

<script>

var canvas = document.getElementById('myCanvas');

var context = canvas.getContext('2d');

var imageObj = new Image();

imageObj.src = 'darth-vader.jpg';

imageObj.onload = function() {

context.drawImage(imageObj, 69, 50);

};

</script>

**值得注意的是：**若调用 drawImage 时，图片没装载完，那什么都不会发生（在一些旧的浏览器中可能会抛出异常）。因此你应该用load时间来保证不会在加载完毕之前使用这个图片：

var img = new Image(); // 创建img元素

img.src = 'myImage.png'; // 设置图片源地址

img.onload = function(){

// 执行drawImage语句

}

### 2. 平铺图像

所谓图像平铺就是用按一定比例缩小后的图像将画布填满。

1. 加载图像。
2. 通过 createPattern(img, type) 方法设置平铺方式。
3. 将平铺对象赋值给filleStyle或strokeStyle属性。
4. 将平铺的图像进行绘制。

createPattern(img, type)

| **参数名称** | **作用** |
| --- | --- |
| img | 需要平铺的图像。 |
| type | 平铺方式。（no-repeat：不平铺；repeat-x：水平方向平铺；repeat-y：垂直方向平铺；repeat：全方向平铺） |

该方法返回平铺对象。

<canvas id="myCanvas" width="578" height="200"></canvas>

<script>

var canvas = document.getElementById('myCanvas');

var context = canvas.getContext('2d');

var imageObj = new Image();

imageObj.src = 'wood-pattern.png';

imageObj.onload = function() {

var pattern = context.createPattern(imageObj, 'repeat');

context.rect(0, 0, canvas.width, canvas.height);

context.fillStyle = pattern;

context.fill();

};

</script>

**值得注意的是：**若调用 createPattern 时，图片没装载完，那什么都不会发生（在一些旧的浏览器中可能会抛出异常）。因此你应该用load时间来保证不会在加载完毕之前使用这个图片：

var img = new Image(); // 创建img元素

img.src = 'myImage.png'; // 设置图片源地址

img.onload = function(){

// 执行createPattern语句

}

### 3. 切割图像

1. 调用beginPath()方法，创建新建一条路径。
2. 使用rect()或arc()方法
3. 通过clip()方法进行切割。

<canvas id="canvas" width="578" height="200"></canvas>

<script>

var canvas = document.getElementById('canvas');

var context = elem.getContext('2d');

var image=new Image();

image.src="img/flower.jpg";

image.onload=function(){

context.drawImage(image,0,0,280,190);

}

context.beginPath();

context.arc(140,95,60,0,Math.PI\*2,true);

context.closePath();

context.clip();

</script>

### 4. 画布方法

#### 1）状态方法

* save()，保存当前画布属性、状态。
* restore()，恢复画布属性、状态。

<canvas id="canvas" width="578" height="200"></canvas>

<script>

var ctx = document.getElementById('canvas').getContext('2d');

ctx.fillRect(0,0,150,150); // Draw a rectangle with default settings

ctx.save(); // Save the default state

ctx.fillStyle = '#09F' // Make changes to the settings

ctx.fillRect(15,15,120,120); // Draw a rectangle with new settings

ctx.save(); // Save the current state

ctx.fillStyle = '#FFF' // Make changes to the settings

ctx.globalAlpha = 0.5;

ctx.fillRect(30,30,90,90); // Draw a rectangle with new settings

ctx.restore(); // Restore previous state

ctx.fillRect(45,45,60,60); // Draw a rectangle with restored settings

ctx.restore(); // Restore original state

ctx.fillRect(60,60,30,30); // Draw a rectangle with restored settings

</script>

#### 2）转换方法

translate(x, y)：用来移动 canvas 和它的原点到一个不同的位置。

* x 是左右偏移量。
* y 是上下偏移量。

scale(x, y)：用它来增减图形在 canvas 中的像素数目，对形状，位图进行缩小或者放大。

* x,y 分别是横轴和纵轴的缩放因子，它们都必须是正值。
* 值分类：
  + 值比 1.0 小表示缩小。
  + 比 1.0 大则表示放大。
  + 值为 1.0 时什么效果都没有。

rotate(angle)：用于以原点为中心旋转 canvas。

* 旋转的角度(angle)，它是顺时针方向的，以弧度为单位的值。

## Chart.js库

### 1. 基础内容

Chart.js是一个简单、面向对象、为设计者和开发者准备的图表绘制工具库。

官方网址：<http://www.chartjs.org/>

Chart.js的特点：

* 基于HTML 5：Chart.js基于HTML5 canvas技术，支持所有现代浏览器，并且针对IE7/8提供了降级替代方案。
* 简单、灵活：Chart.js不依赖任何外部工具库，轻量级（压缩之后仅有4.5k），并且提供了加载外部参数的方法。

### 2. 使用Chart.js

#### 1）使用Chart.js的步骤

* 在HTML页面中引入Chart.js文件。

<script src="Chart.js"></script>

* 创建<canvas>元素：用于显示Chart图表的容器。

<canvas id="myChart" width="400" height="400"></canvas>

* 获取Canvas对象。

document.getElementById("myChart").getContext("2d");`

* 创建Chart图表对象。

new Chart(ctx);

* 通过Chart图表对象进行绘制。

chart.PolarArea(data);

#### 2）Chart.js全局配置

Chart.js 全局配置是在chart.js 第一个正式版本中引入。Chart.js 全局配置用于改变所有图表的类型，避免了需要在每一个图表中单独进行设置。当然，Chart.js 全局配置也可以专门为某一个特定的图表进行配置。

语法如下：

Chart.defaults.global.参数名 = 参数值;

例如如下方式设定：

Chart.defaults.global.responsive = true;

#### 3）绘制图表

##### a. 曲线图

曲线图就是将多个数据点绘制在一条线上，通常被用于展示趋势的数据或两组数据之间的对比。

new Chart(ctx).Line(data, options)

* data：用于设置曲线上的数据、样式及名称。
* options：选项，用于配置曲线图。

##### b. 柱状图

柱状图就是使用柱状方式显示数据的一种方式，通常被用于展示趋势的数据或多组数据之间的比较。

new Chart(ctx).Bar(data, options)

* data：用于设置柱状图上的数据、样式及名称。
* options：选项，用于配置柱状图。

##### c. 饼状图

饼状图可能是所有图表中最为常用的一种，就是将一个圆划分成若干个部分，每个弧形展示每个数据的比例值。通常被用于展示多组数据之间的比例。

new Chart(ctx).Pie(data,options)

* data：用于设置饼图的数据、样式及名称。
* options：选项，用于配置饼图。

##### d. 雷达图

雷达图就是一种展示多个数据点以及它们之间变化的方式，通常被用于比较点的两个或多个不同的数据集。

new Chart(ctx).Radar(data, options)

* data：用于设置雷达图的数据、样式及名称。
* options：选项，用于配置雷达图。

##### e. 环形图

环形图类似于饼状图，但环形图是一个空心的环形形状，通常被用于展示多组数据之间关系的比例。

new Chart(ctx).Doughnut(data,options)

* data：用于设置环形图的数据、样式及名称。
* options：选项，用于配置环形图。

##### f. 极地区域图

极地区域图类似于饼状图，但每一个扇形的角度和半径取决于不同的值，通常被用于需要展示类似于饼状图的比较数据的基础上，还需要展示范围值的比较。

new Chart(ctx).PolarArea(data, options)

* data：用于设置极地区域图的数据、样式及名称。
* options：选项，用于配置极地区域图。

<canvas id="canvas" width="300px" height="300px"></canvas>

<script>

var canvas = document.getElementById("canvas");

var context = canvas.getContext("2d");

// 初始化饼状图的数据 - 以数组格式

var data = [

{

// 当前数据占饼状图的比例

value: 30,

// 当前数据块的颜色

color:"blue",

// 当鼠标悬停在当前数据块时的颜色

highlight: "yellow",

// 当前数据块的提示信息

label: "Blue"

},

{

value: 50,

color: "#46BFBD",

highlight: "#5AD3D1",

label: "Green"

},

{

value: 100,

color: "#FDB45C",

highlight: "#FFC870",

label: "Yellow"

},

{

value: 40,

color: "#949FB1",

highlight: "#A8B3C5",

label: "Grey"

}

];

//Chart.defaults.global.responsive = true;

// 通过画布对象,创建Chart对象

var chart = new Chart(context);

// 利用Chart对象调用图表方法

chart.Pie(data);

</script>

**原创声明：**如发布或转载请注明以下作者信息，谢谢！

本文作者 - 金云龙

联系方式 - QQ：2080954535；微信：imlonge；微信公众号：longestory

个人网站：<http://www.longestory.com>

# HTML5 第四天笔记

## 基础内容

### 1. 什么是 SVG

SVG 全称为 Scalable Vector Graphics，译为可缩放矢量图形,简称矢量图。是一种用来描述二维矢量图形的 XML 标记语言。

### 2. SVG与Flash的区别

|  | **SVG** | **Flash** |
| --- | --- | --- |
| 相同点 | 用于二维矢量图形 | 用于二维矢量图形 |
| 不同点 | 是一个开放的W3C标准，基于XML | 封闭的基于二进制格式的 |

### 3. SVG与Canvas的区别

| **SVG** | **Canvas** |
| --- | --- |
| 不依赖于分辨率 | 依赖于分辨率 |
| 使用DOM及事件处理器（DOM专门为SVG开放接口） | 不能使用DOM及事件处理器 |
| 不能实现游戏开发 | 可以实现游戏开发 |
| 实现大型渲染区域的应用(例如百度地图) | 是以图片(png |

### 4. SVG的优势

* 不需要专门的编辑器,文本编辑器都可以
* 可被搜索、索引、脚本化及压缩
* 图像不失真(和分辨率无关)

**值得注意的是：**

* SVG技术并不是专属于HTML5的
* SVG技术本身是一套独立的用于描述二维图形
* HTML5版本之前,以图片形式进行引入
* HTML5版本之后,允许在HTML页面直接使用SVG技术

**可参考的资源：**

* SVG 标准：<http://www.w3.org/Graphics/SVG/>
* SVG 教程：<https://developer.mozilla.org/zh-CN/docs/Web/SVG/Tutorial>

### 5. 如何使用 SVG

#### 1）SVG 文件（了解）

SVG 文件的扩展名为 “.svg”，使用的是 XML 技术的语法内容，并且可以在浏览器中直接运行。

<?xml version="1.0" encoding="utf-8" ?>

<!-- SVG的语法标准(必要的) -->

<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<!-- xmlns="http://www.w3.org/2000/svg" SVG的命名空间 -->

<svg version="1.1" xmlns="http://www.w3.org/2000/svg">

<rect x="100" y="100" width="300" height="100" fill="blue" stroke="black" stroke-width="4"/>

</svg>

也可以在 HTML 页面引入 SVG 文件。

<img src="01\_SvgFile.svg" width="800" height="500">

#### 2）HTML 直接定义 SVG

HTML5 的新特性允许在 HTML 页面中直接定义 SVG 元素（需要学习有关SVG的一些HTML新元素）。

**<svg>元素：**

|  | **<svg>元素** |
| --- | --- |
| 作用 | 类似于canvas元素 |
| 特点 | 默认的宽度和高度分别为300px和150px，不具有任何的样式(默认情况下不显示) |
| 目的 | 表示当前使用SVG语法内容 |

**<svg>元素的属性：**

| **属性名称** | **描述** |
| --- | --- |
| width | 设置 <svg> 元素的宽度 |
| height | 设置 <svg> 元素的高度 |
| style | 设置 <svg> 元素的样式 |

<svg width="500" height="300" style="background:pink;">

<rect x="100" y="100" width="300" height="100" fill="blue" stroke="black" stroke-width="4"/>

</svg>

## 图像元素

### 1. 样式

| **属性名称** | **描述** |
| --- | --- |
| fill | 设置填充样式 |
| stroke | 设置描边样式 |
| stroke-width | 设置描边宽度 |

### 2. 矩形

<rect x="" y="" rx="" ry="" width="" height="" />

| **属性名称** | **描述** |
| --- | --- |
| x和y | 表示绘制矩形的左上角坐标值 |
| width | 表示绘制矩形的宽度 |
| height | 表示绘制矩形的高度 |
| rx和ry | 表示绘制矩形的四个角的水平圆角和垂直圆角（如果rx和ry的值分别为width/2和height/2,绘制圆形） |

**值得注意的是：**

* 默认绘制出来的效果：实心矩形效果。
* 可以实现"空心"的效果：不能实现空心效果。

<svg width="1000" height="600" style="background:pink;">

<!-- 绘制矩形 -->

<rect x="10" y="10" width="100" height="100" />

<rect x="10" y="120" width="100" height="100" fill="white" stroke="black" />

<rect x="10" y="230" width="100" height="100" rx="10" ry="10" />

<rect x="10" y="340" width="100" height="100" rx="50" ry="50" style="fill:white;stroke:black;" />

</svg>

### 3. 圆形

<cirlcle cx="" cy="" r="" />

| **属性名称** | **描述** |
| --- | --- |
| cx和cy | 表示绘制圆形的圆心坐标值 |
| r | 表示绘制圆形的半径 |

<svg width="1000" height="600" style="background:pink;">

<!-- 绘制圆形 -->

<circle cx="170" cy="60" r="50" />

<circle cx="170" cy="170" r="50" fill="white" stroke="black" />

</svg>

### 4. 椭圆

<ellipse cx="" cy="" rx="" ry="" />

| **属性名称** | **描述** |
| --- | --- |
| cx和cy | 表示绘制椭圆的圆心坐标值 |
| rx | 表示绘制椭圆的水平方向半径 |
| ry | 表示绘制椭圆的垂直方向半径 |

<svg width="1000" height="600" style="background:pink;">

<!-- 绘制椭圆 -->

<ellipse cx="280" cy="70" rx="50" ry="60" />

<ellipse cx="280" cy="190" rx="50" ry="50" />

<ellipse cx="280" cy="300" rx="50" ry="30" />

</svg>

### 5. 线条

<line x1="" y1="" x2="" y2="" stroke="" />

| **属性名称** | **描述** |
| --- | --- |
| x1和y1 | 表示绘制直线的起点坐标值 |
| x2和y2 | 表示绘制直线的终点坐标值 |
| stroke | 设置绘制直线的样式(颜色) |
| stroke-width | 设置绘制直线的宽度 |

<svg width="1000" height="600" style="background:pink;">

<!-- 绘制直线 -->

<line x1="350" y1="10" x2="500" y2="200" stroke="black" stroke-width="10" />

</svg>

### 6. 折线

<polyline points="" />

| **属性名称** | **描述** |
| --- | --- |
| points | 表示绘制折线的起点、折点及终点坐标值（格式：x1,y1 x2,y2 x3,y3 xn,yn） |
| stroke | 设置折线的颜色 |
| stroke-width | 设置折线的线宽 |
| fill | 设置与<svg>元素的背景色相同 |

<svg width="1000" height="600" style="background:pink;">

<!-- 绘制折线 -->

<polyline points="520,20 520,200" stroke="black" stroke-width="10" />

<polyline points="550,20 550,200 700,200 700,20 540,20" stroke="black" stroke-width="20" fill="pink" />

</svg>

### 7. 多边形

<polygon points="" />

| **属性名称** | **描述** |
| --- | --- |
| points | 表示绘制多边形的所有点坐标值 |

<svg width="1000" height="600" style="background:pink;">

<!-- 绘制多边形 -->

<polygon points="550,300 550,500 800,500" stroke="black" fill="pink" stroke-width="10" />

</svg>

**值得注意的是：**

* 一个HTML页面允许包含多个<svg>元素
* 一个<svg>元素允许包含多个图形元素
* SVG的图形元素基本都是起始元素
* 定义图形元素时,只定义开始元素,没有结束元素
  + 浏览器在运行页面时,并不报错
  + 浏览器解析这段元素代码时,自动补全结束元素
  + 自动补全的结束元素是不正确的

### 8. 渐变

#### 1）线性渐变

<linearGradient id="" x1="" y1="" x2="" y2=""></linearGradient>

| **属性名称** | **描述** |
| --- | --- |
| id | 标识，便于其他元素进行引用 |
| x1和y1 | 表示基准线的起点坐标值（值范围为 0-1, 是百分值 0%-100%） |
| x2和y2 | 表示基准线的终点坐标值（值范围为 0-1, 是百分值 0%-100%） |

<stop offset="" stop-color=""/>

| **属性名称** | **描述** |
| --- | --- |
| offset | 表示设置渐变颜色的位置 |
| stop-color | 表示设置的渐变颜色 |
| stop-opacity | 表示设置渐变颜色的透明度 |

<svg width="800" height="500">

<defs>

<linearGradient id="grd" x1="0" y1="0" x2="1" y2="1">

<stop offset="0" stop-color="red" stop-opacity="0.5" />

<stop offset="0.5" stop-color="green" stop-opacity="0.5" />

<stop offset="1" stop-color="blue" stop-opacity="0.5" />

</linearGradient>

</defs>

<rect x="0" y="0" width="200" height="200" fill="url(#grd)" />

<circle cx="400" cy="400" r="100" fill="url(#grd)" />

</svg>

#### 2）射线渐变

<radialGradient id="" cx="" cy="" r="" fx="" fy=""></radialGradient>

| **属性名称** | **描述** |
| --- | --- |
| id | 标识，便于其他元素进行引用 |
| cx和cy | 表示基准线中的中心点坐标值（值范围为 0-1, 是百分值 0%-100%） |
| fx和fy | 表示基准线中的焦点坐标值（值范围为 0-1, 是百分值 0%-100%） |
| r | 设置其边缘位置（值范围为 0-1, 是百分值 0%-100%） |

<stop offset="" stop-color=""/>

| **属性名称** | **描述** |
| --- | --- |
| offset | 表示设置渐变颜色的位置 |
| stop-color | 表示设置的渐变颜色 |
| stop-opacity | 表示设置渐变颜色的透明度 |

<svg width="800" height="500">

<defs>

<radialGradient id="grd1">

<stop offset="0" stop-color="red" />

<stop offset="1" stop-color="blue" />

</radialGradient>

<radialGradient id="grd2" cx="0" cy="0">

<stop offset="0" stop-color="red" />

<stop offset="1" stop-color="blue" />

</radialGradient>

<radialGradient id="grd3" r="1">

<stop offset="0" stop-color="red" />

<stop offset="1" stop-color="blue" />

</radialGradient>

<radialGradient id="grd4" fx="1" fy="1">

<stop offset="0" stop-color="red" />

<stop offset="1" stop-color="blue" />

</radialGradient>

<radialGradient id="grd5">

<stop offset="0" stop-color="red" />

<stop offset="0.5" stop-color="yellow" />

<stop offset="1" stop-color="blue" />

</radialGradient>

<radialGradient id="grd6" cx="0" cy="0">

<stop offset="0" stop-color="red" />

<stop offset="0.5" stop-color="yellow" />

<stop offset="1" stop-color="blue" />

</radialGradient>

<radialGradient id="grd7" r="1">

<stop offset="0" stop-color="red" />

<stop offset="0.5" stop-color="yellow" />

<stop offset="1" stop-color="blue" />

</radialGradient>

<radialGradient id="grd8" fx="1" fy="1">

<stop offset="0" stop-color="red" />

<stop offset="0.5" stop-color="yellow" />

<stop offset="1" stop-color="blue" />

</radialGradient>

</defs>

<rect x="0" y="0" width="200" height="200" fill="url(#grd1)" />

<rect x="210" y="0" width="200" height="200" fill="url(#grd2)" />

<rect x="420" y="0" width="200" height="200" fill="url(#grd3)" />

<rect x="630" y="0" width="200" height="200" fill="url(#grd4)" />

<rect x="0" y="210" width="200" height="200" fill="url(#grd5)" />

<rect x="210" y="210" width="200" height="200" fill="url(#grd6)" />

<rect x="420" y="210" width="200" height="200" fill="url(#grd7)" />

<rect x="630" y="210" width="200" height="200" fill="url(#grd8)" />

</svg>

**值得注意的是：**

* 渐变元素需要定义id属性,便于其他元素进行引用
* 渐变元素定义在元素内
  + 元素内定义：表示该元素允许重复使用

### 9. 滤镜

<filter id="myfilter"></filter>

**值得注意的是：**该元素只是滤镜的容器。

#### 高斯模糊

<feGaussianBlur in="" stdDeviation="" />

| **属性名称** | **描述** |
| --- | --- |
| in | 设置高斯模糊的样式 |
| stdDeviation | 设置模糊的程度 |

<svg width="800" height="500">

<defs>

<filter id="myfilter">

<feGaussianBlur in="SourceGraphic" stdDeviation="5" />

</filter>

</defs>

<rect x="100" y="100" width="100" height="100" filter="url(#myfilter)" />

</svg>

## Two.js库

### 1. 基础内容

Two.js支持不同的上下文环境：

* SVG（默认）
* Canvas
* WebGL

官网地址：<http://jonobr1.github.io/two.js/>

#### 如何使用 Two.js

##### 1）HTML 页面

* 引入Two.js库文件
* 定义容器元素 <div></div>

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>如何使用Two.js库</title>

<!-- 1. 引入Two.js库文件 -->

<script src="two.js"></script>

</head>

<body>

<!--

2. 定义用于显示矢量图的容器元素

\* <div></div>元素 - 建议使用

-->

<div id="d1"></div>

</body>

</html>

##### 2）JavaScript逻辑

* 获取HTML页面的容器元素
* 通过 Two.js 库提供的 Two() 构造函数创建 Two 对象

var params = {// 创建svg时初始化的数据

width : 宽度,

height : 高度

}

var two = new Two(params);

* 将创建的 Two 对象添加到 HTML 页面容器元素内

two.appendTo(elem);

* 使用 Two.js 库提供的 API 方法绘制图形
* 调用 update() 方法进行绘制

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>如何使用Two.js库</title>

<!-- 1. 引入Two.js库文件 -->

<script src="two.js"></script>

</head>

<body>

<!--

2. 定义用于显示矢量图的容器元素

\* <div></div>元素 - 建议使用

-->

<div id="d1"></div>

<script>

// 3. 获取HTML页面的容器元素

var elem = document.getElementById("d1");

/\*

4. 通过Two(params)构造函数来创建Two对象

\* params - 创建Two对象的初始化数据

\* Object类型,{ key : value }格式

\*/

var params = {

type : Two.Types.canvas,

width : 285, // 默认为640px

height : 200 // 默认为480px

};

var two = new Two(params);

/\*

5. 将创建的Two对象添加到页面容器元素中

\* Two.js库提供了appendTo()方法向容器添加Two对象

\* 注意 - appendTo()方法并不是jQuery的

\*/

two.appendTo(elem);

// 6. 通过调用Two对象提供的API方法进行绘制

var circle = two.makeCircle(72, 100, 50);

var rect = two.makeRectangle(213, 100, 100, 100);

// 7. 调用update()方法进行绘制

two.update();

</script>

</body>

</html>

### 2. 绘制图像

#### 1）绘制图像的方法

| **图像形状** | **方法** |
| --- | --- |
| 圆形 | makeCircle(x, y, radius) |
| 直线 | makeLine(x1, y1, x2, y2) |
| 矩形 | makeRectangle(x, y, width, height) |
| 圆角矩形 | makeRoundedRectangle(x, y, width, height, radius) |
| 椭圆 | makeEllipse(x, y, width, height) |
| 多边形 | makePolygon(ox, oy, r, sides) |
| 路径 | makePath(x1, y1, x2, y2, xN, yN, open) |
| 星形 | makeStar(ox, oy, or, ir, sides) |

#### 2）绘制图像的属性

| **属性名称** | **描述** |
| --- | --- |
| fill | 设置填充样式 |
| stroke | 设置描边样式 |
| linewidth | 设置线条宽度 |
| opacity | 设置透明度 |
| cap | 设置线条端点形状，默认为round |
| join | 设置线条交点形状，默认为round |

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>如何使用Two.js库</title>

<script src="two.js"></script>

</head>

<body>

<div id="d1"></div>

<script>

var elem = document.getElementById("d1");

var params = {

type : Two.Types.canvas,

width : 285, // 默认为640px

height : 200 // 默认为480px

};

var two = new Two(params);

two.appendTo(elem);

var circle = two.makeCircle(72, 100, 50);

var rect = two.makeRectangle(213, 100, 100, 100);

// 设置绘制的圆形和矩形的样式

circle.fill = '#FF8000';

circle.stroke = 'orangered';

circle.linewidth = 5;

rect.fill = 'rgb(0, 200, 255)';

rect.opacity = 0.75;

rect.noStroke();// 设置矩形没有描边

two.update();

</script>

</body>

</html>

### 3. 图像分组

调用 Two 对象的 makeGroup() 方法对图像进行分组操作。

* makeGroup() 方法任意图形对象作为参数
* makeGroup() 方法返回分组对象

**值得注意的是：**在统一对一组图像设置后,针对不同图像进行个性化设置。

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Two.js库进行图形分组</title>

<script src="two.js"></script>

</head>

<body>

<div id="d1"></div>

<script>

var elem = document.getElementById("d1");

var two = new Two({

width : 285,

height : 200

}).appendTo(elem);

var circle = two.makeCircle(72, 100, 50);

var rect = two.makeRectangle(213, 100, 100, 100);

var group = two.makeGroup(circle,rect);

group.fill = 'rgb(0, 200, 255)';

group.opacity = 0.75;

group.noStroke();// 设置矩形没有描边

// 针对不同的图形进行个性化样式设置

circle.stroke = 'orangered';

circle.linewidth = 5;

two.update();

</script>

</body>

</html>

### 4. 动画效果（了解）

| **方法名称** | **描述** |
| --- | --- |
| play() | 提供一组循环动画 |
| pause() | 提供中止动画效果 |
| update() | 提供更新当前绘制或设置 |

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Two.js库实现动画效果</title>

<script src="two.js"></script>

</head>

<body>

<div id="d1"></div>

<script>

var elem = document.getElementById("d1");

var two = new Two({

width : 285,

height : 200

}).appendTo(elem);

// 设置圆形和矩形

var circle = two.makeCircle(-70, 0, 50);

var rect = two.makeRectangle(70, 0, 100, 100);

// 设置圆形和矩形的样式

circle.fill = '#FF8000';

rect.fill = 'rgba(0, 200, 255, 0.75)';

// 将圆形和矩形分为一组

var group = two.makeGroup(circle, rect);

// 针对这一组进行样式设置

group.translation.set(two.width / 2, two.height / 2);// 将"画布"平移水平和垂直一半距离

group.scale = 0;// 将圆形和矩形缩放为 0(不显示)

group.noStroke();// 设置圆形和矩形没有边框

// Two对象绑定update事件(方法)

two.bind('update', function(frameCount) {

// 判断如果缩放为原大小,将缩放和旋转设置为 0

if (group.scale > 0.9999) {

group.scale = group.rotation = 0;

}

// 每次执行的增量

var t = (1 - group.scale) \* 0.125;

// 每次执行后缩放值进行累加

group.scale += t;

// 每次执行后旋转至进行累加

group.rotation += t \* 4 \* Math.PI;

}).play();// 开始无限循环动画

</script>

</body>

</html>

**原创声明：**如发布或转载请注明以下作者信息，谢谢！

本文作者 - 金云龙

联系方式 - QQ：2080954535；微信：imlonge；微信公众号：longestory

个人网站：<http://www.longestory.com>

# HTML5 第五天笔记

## 百度地图

### 1. 如何使用

#### 1）HTML 页面

* 引入百度地图提供的JS文件

<script src="http://api.map.baidu.com/api?v=2.0&ak=HbUVYMUg6PwbOnXkztdgSQlQ"></script>

* 定义容器元素<div></div>

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>百度地图如何使用</title>

<style type="text/css">

body, html,#allmap {

width: 100%;

height: 100%;

overflow: hidden;

margin:0;

font-family:"微软雅黑";

}

</style>

<!--

引入百度地图提供的JS文件

\* 是在线提供的JS文件 - 必须可以联网

http://api.map.baidu.com/api?v=2.0&ak=您的密钥

\* v=2.0 - 使用的百度地图的版本

\* ak=您的密钥 - 传递当前应用的秘钥

\* 只有秘钥正确,正常地使用百度地图的功能

-->

<script src="http://api.map.baidu.com/api?v=2.0&ak=HbUVYMUg6PwbOnXkztdgSQlQ"></script>

</head>

<body>

<!-- 定义一个容器元素 - 用于显示百度地图 -->

<div id="allmap"></div>

</body>

</html>

#### 2）JavaScript 逻辑

* 创建百度地图对象

var map = new BMap.Map(容器元素id);

* 初始化百度地图

map.centerAndZoom(中心点坐标,显示级别)

如下述示例代码：

/\*

1. 通过Map()构造函数,创建百度地图对象

var map = new BMap.Map(容器元素id属性值);

\*/

var map = new BMap.Map("allmap");

/\*

2. (必要)初始化百度地图 - 设置中心点和显示级别

map.centerAndZoom(center,zoom)

\* center - 设置百度地图当前的中心点坐标

\* Point类型,zoom参数必须设置

\* String类型,zoom参数可有可无

\* 如果不设置zoom,自动匹配最佳显示级别

\* zoom - 设置百度地图当前的显示级别

\* 一般情况下,值范围为 3-19

\* 高清地图(移动端),值范围为 3-18

\*/

map.centerAndZoom("北京",12);

### 2. 核心类

#### Map类

**构造器**

| **构造函数** | **参数** | **说明** |
| --- | --- | --- |
| Map(container) | container：页面容器元素id | 创建地图实例对象 |

**配置方法**

| **方法名称** | **参数** | **说明** |
| --- | --- | --- |
| enableScrollWheelZoom() |  | 开启鼠标滚轮放大或缩小地图的显示级别 |
| disableScrollWheelZoom() |  | 禁用鼠标滚轮功能 |
| enableDragging() |  | 启用鼠标拖拽功能 |
| disableDragging() |  | 禁用鼠标拖拽功能 |

**地图状态方法**

| **方法名称** | **参数** | **说明** |
| --- | --- | --- |
| getCenter() |  | 返回地图当前中心点(Point类型) |
| getZoom() |  | 返回地图当前缩放级别(Number类型) |

**修改地图状态方法**

| **方法名称** | **参数** | **说明** |
| --- | --- | --- |
| centerAndZoom(center,zoom) | center：设置地图的中心点坐标；zoom：设置地图的显示级别 | 初始化地图中心点和级别 |
| setCenter(center) | center：设置地图的中心点坐标 | 设置地图中心点(Point |
| setCurrentCity(city) | city：设置地图城市 | 设置地图城市 |
| setZoom(zoom) | zoom：设置地图的显示级别 | 设置显示级别 |

**控件方法**

| **方法名称** | **参数** | **说明** |
| --- | --- | --- |
| addControl(control) |  | 将控件添加到地图 |
| removeControl(control) |  | 从地图中移除控件 |

**覆盖物方法**

| **方法名称** | **参数** | **说明** |
| --- | --- | --- |
| addOverlay(overlay) |  | 将覆盖物添加到地图中 |
| removeOverlay(overlay) |  | 从地图中移除覆盖物 |
| openInfoWindow(InfoWindow, Point) | InfoWindow：设置信息窗口；Point：在指定点打开 | 设置信息窗口 |

### 3. 控件类

#### ScaleControl类

表示比例尺控件。

**构造器**

| **构造函数** | **参数** | **说明** |
| --- | --- | --- |
| ScaleControl(ScaleControlOptions) | - | 创建地图比例尺对象 |

#### ScaleControlOptions类

表示ScaleControl构造函数的可选参数。

**选项**

* anchor：设置当前控件显示的位置。
  + BMAP\_ANCHOR\_TOP\_LEFT：左上角
  + BMAP\_ANCHOR\_BOTTOM\_LEFT：左下角
  + BMAP\_ANCHOR\_TOP\_RIGHT：右上角
  + BMAP\_ANCHOR\_BOTTOM\_RIGHT：右下角

#### NavigationControl类

表示地图的平移缩放控件。

**构造器**

| **构造函数** | **参数** | **说明** |
| --- | --- | --- |
| NavigationControl(NavigationControlOptions) | - | 创建地图平移缩放对象 |

#### MapTypeControl类

负责切换地图类型的控件。

**构造器**

| **构造函数** | **参数** | **说明** |
| --- | --- | --- |
| MapTypeControl(MapTypeControlOptions) | - | 创建地图类型对象 |

#### OverviewMapControl类

表示缩略地图控件。

**构造器**

| **构造函数** | **参数** | **说明** |
| --- | --- | --- |
| OverviewMapControl(OverviewMapControlOptions) | - | 创建缩略地图对象 |

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>百度地图的控件类</title>

<style type="text/css">

body, html,#allmap {

width: 100%;

height: 100%;

overflow: hidden;

margin:0;

font-family:"微软雅黑";

}

</style>

<script src="http://api.map.baidu.com/api?v=2.0&ak=HbUVYMUg6PwbOnXkztdgSQlQ"></script>

</head>

<body>

<div id="allmap"></div>

<script>

// 1. 创建地图对象

var map = new BMap.Map("allmap");

// 2. 初始化地图

map.centerAndZoom("北京",13);

map.enableScrollWheelZoom();

/\*

3. 添加比例尺控件

\* 创建比例尺控件对象

\* 将比例尺控件添加到地图中

\*/

var scale = new BMap.ScaleControl({

anchor: BMAP\_ANCHOR\_TOP\_RIGHT

});

map.addControl(scale);

/\*

4. 添加平移缩放控件

\* 创建平移缩放控件对象

\* 将平移缩放控件添加到地图中

\*/

var nav = new BMap.NavigationControl({

anchor: BMAP\_ANCHOR\_TOP\_RIGHT

});

map.addControl(nav);

// 5. 添加地图类型切换控件

var type = new BMap.MapTypeControl({

anchor: BMAP\_ANCHOR\_TOP\_LEFT

});

map.addControl(type);

// 6. 添加缩略地图控件

var overview = new BMap.OverviewMapControl({

isOpen : true//表示是否自动打开

});

map.addControl(overview);

</script>

</body>

</html>

### 4. 覆盖物类

#### Marker类

表示地图上一个图像标注。

**构造器**

| **构造函数** | **参数** | **说明** |
| --- | --- | --- |
| Marker(point) | - | 创建标注对象 |

#### InfoWindow类

表示地图上包含信息的窗口。

**构造器**

| **构造函数** | **参数** | **说明** |
| --- | --- | --- |
| InfoWindow(content) | content：设置当前点的地址信息内容 | 创建信息的窗口对象 |

**方法**

| **方法名称** | **参数** | **说明** |
| --- | --- | --- |
| setWidth(width) |  | 设置信息窗口的宽度 |
| setHeight(height) |  | 设置信息窗口的高度 |
| setTitle(title) |  | 设置信息窗口的标题 |

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>为百度地图进行标注</title>

<style type="text/css">

body, html,#allmap {

width: 100%;

height: 100%;

overflow: hidden;

margin:0;

font-family:"微软雅黑";

}

</style>

<script src="http://api.map.baidu.com/api?v=2.0&ak=HbUVYMUg6PwbOnXkztdgSQlQ"></script>

</head>

<body>

<div id="allmap"></div>

<script>

var map = new BMap.Map("allmap");

var point = new BMap.Point(116.404, 39.915);

map.centerAndZoom(point, 15);

/\*

如何进行标注

\* 创建标注对象

\* 将标注添加到地图中

\*/

var marker = new BMap.Marker(point);

map.addOverlay(marker);

marker.setAnimation(BMAP\_ANIMATION\_BOUNCE);

</script>

</body>

</html>

### 5. 服务类

#### Geocoder类

用于获取用户的地址解析。

**构造器**

| **构造函数** | **参数** | **说明** |
| --- | --- | --- |
| Geocoder() | content：设置当前点的地址信息内容 | 创建地址解析对象 |

**方法**

| **方法名称** | **参数** | **说明** |
| --- | --- | --- |
| getPoint(address,callback,city) | address：要解析的地址内容；callback：回调函数（解析address地址成功的话,回调函数具有一个形参）；city：当前地址所在的城市名称 | 对指定的地址进行解析 |

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>百度地图案例</title>

<style type="text/css">

body, html,#allmap {

width: 100%;

height: 100%;

overflow: hidden;

margin:0;

font-family:"微软雅黑";

}

</style>

<script src="http://api.map.baidu.com/api?v=2.0&ak=HbUVYMUg6PwbOnXkztdgSQlQ"></script>

</head>

<body>

<div id="allmap"></div>

<script>

// 1. 创建地图对象

var map = new BMap.Map("allmap");

// 2. 初始化地图

//map.centerAndZoom("北京",13);

// 3. 启用鼠标的滚轮功能

map.enableScrollWheelZoom();

/\*

4. 根据给定的地址获取Point的经度和纬度

var geocoder = new BMap.Geocoder();

geocoder.getPoint(address,function(point){},city);

\*/

var geocoder = new BMap.Geocoder();

geocoder.getPoint("北京市海淀区万寿路西街2号",function(point){

// 5. 根据Point进行标注

var marker = new BMap.Marker(point);

map.addOverlay(marker);

// 6. 重新设置中心点位置

map.centerAndZoom(point,16);

// 7. 设置信息窗口

var opts = {

width : 200,

height: 100,

title : "文博大厦"

}

var info = new BMap.InfoWindow("地址:北京市海淀区万寿路西街2号",opts);

marker.addEventListener("click",function(){

map.openInfoWindow(info,point);

});

},"北京");

</script>

</body>

</html>

## 拖放 API

允许用户在一个元素上点击并按住鼠标按钮，拖动它到别的位置，然后松开鼠标按钮将元素放到那儿。

在拖动操作过程中，被拖动元素会以半透明形式展现，并跟随鼠标指针移动。放置元素的位置可能会在不同的应用内。

### 1. 源元素事件

**所谓源元素**就是被拖动的元素。

| **事件名称** | **作用** | **触发次数** |
| --- | --- | --- |
| dragstart事件 | 开始拖动源元素时被触发 | 只被触发一次 |
| drag事件 | 拖动的过程中，实时被触发 | 被触发多次 |
| dragend事件 | 结束拖动源元素时被触发 | 只被触发一次 |

<div id="d1">

<img id="img" src="Penguins.jpg" width="256">

</div>

<script>

// 获取HTML页面中的源元素<img>

var img = document.getElementById("img");

// 为源元素绑定拖动事件

img.addEventListener("dragstart",myDragstart);

img.addEventListener("drag",myDrag);

img.addEventListener("dragend",myDragend);

// 定义事件的处理函数

function myDragstart(event){

console.log("开始拖动啦...");

}

function myDrag(event){

console.log("拖动过程中...");

}

function myDragend(event){

console.log("结束拖动啦...");

}

</script>

### 2. 目标元素事件

**所谓目标元素**就是投放到的元素。

| **事件名称** | **作用** | **触发次数** |
| --- | --- | --- |
| dragenter事件 | 当源元素到达目标元素时被触发 | 只被触发一次 |
| dragover事件 | 当源元素到达目标元素时被触发 | 被触发多次。阻止默认行为，触发drop事件 |
| drop事件 | 当源元素投放在目标元素时被触发 | 默认该事件不会被触发 |
| dragleave事件 | 当源元素离开目标元素时被触发 | 只被触发一次 |

<div id="d1">

<img id="img" src="Penguins.jpg" width="256">

</div>

<div id="d2"></div>

<script>

// 获取目标元素

var d2 = document.getElementById("d2");

// 为目标元素绑定事件

d2.addEventListener("dragenter",myDragenter);

d2.addEventListener("dragover",myDragover);

d2.addEventListener("drop",myDrop);

d2.addEventListener("dragleave",myDragleave);

// 定义事件的处理函数

function myDragenter(event){

//event.preventDefault();

console.log("大爷,你来啦...");

}

function myDragover(event){

event.preventDefault();

console.log("大爷,又来啦...");

}

function myDrop(event){

//event.preventDefault();

console.log("大爷,别走啦...");

}

function myDragleave(event){

//event.preventDefault();

console.log("大爷,要走啦...");

}

</script>

### 3. dataTransfer对象

该对象已被集成在 event 对象中，起到剪切板的功能。

| **方法名称** | **作用** | **场景** |
| --- | --- | --- |
| setData(type,data) | 存储数据 | 在源元素事件中 |
| getData(type) | 取出数据 | 在目标元素事件中 |
| clearData() | 清除剪切板中所有数据 |  |

参数：

* type：指定当前存储数据的类型(标识)。
* data：需要中转的数据内容。

<div id="d1">

<img id="img" src="Penguins.jpg" width="256">

</div>

<div id="d2"></div>

<script>

// 1. 获取源元素和目标元素

var img = document.getElementById("img");

var d2 = document.getElementById("d2");

/\*

2. 为源元素和目标元素绑定事件

\* 绑定源元素事件的目的:

\* 通过dataTransfer对象获取源元素的关键数据

\* 只需绑定dragstart事件即可

\* 绑定目标元素事件的目的:

\* 通过dataTransfer对象使用源元素的关键数据

\* 需要绑定dragover和drop事件

\*/

img.addEventListener("dragstart",function(event){

// 通过dataTransfer对象,存储源元素的关键数据

event.dataTransfer.setData("text",img.src);

});

d2.addEventListener("dragover",function(event){

event.preventDefault();

});

d2.addEventListener("drop",function(event){

// 通过dataTransfer对象,使用源元素的关键数据

var src = event.dataTransfer.getData("text");

//d2.innerHTML = "<img src='"+src+"' width='256'>";

var newImg = document.createElement("img");

newImg.src = src;

newImg.width = "256";

newImg.id = "newImg";

d2.appendChild(newImg);

img.parentNode.removeChild(img);

});

</script>

**原创声明：**如发布或转载请注明以下作者信息，谢谢！

本文作者 - 金云龙

联系方式 - QQ：2080954535；微信：imlonge；微信公众号：longestory

个人网站：<http://www.longestory.com>

# HTML5 第六天笔记

## Web Workers

### 1. 基础内容

#### 1）什么是 Web Workers

能够把 JavaScript 计算委托给后台线程，通过允许这些活动以防止使交互型事件变得缓慢。

上句话是 MDN 对象 Web Workers 的简单描述。

Web Workers 是可以在后台（页面端）运行的任务，它能够被轻松的创建，还能向它的创建者发送消息。

Web Workers 的三大主要特征：能够长时间运行（响应），理想的启动性能以及理想的内存消耗。

#### 2）两种 Web Workers

Web Workers 可以分为两种类型：Dedicated Web Worker（专用线程）和 Shared Web Worker（共享线程）。

##### a. Dedicated Web Worker

目前绝大多数的应用场景所使用的是 Dedicated Web Worker（专用线程）。Dedicated Web Worker随着 HTML 页面关闭而被终止，这就意味着 Dedicated Web Worker 只能被创建它的 HTML 页面所访问。

在 JavaScript 代码中，Work 类型代表 Dedicated Web Worker。

##### b. Shared Web Worker

与 Dedicated Web Worker（专用线程）相比，Shared Web Worker 一般用于一些特定情况，可以为多个 HTML 页面服务，即可以被多个 HTML 页面所访问。与之对应的，只有将与之相关的所有 HTML 页面都关闭才能被终止。

在 JavaScript 代码中，SharedWorker 类型代表 Shared Web Worker。

#### 3）Web Workers 的限制

Web Workers 在具体使用中，具有以下几种限制：

* Web Workers 无法访问 DOM 节点；
* Web Workers 无法访问全局变量或是全局函数；
* Web Workers 无法调用 alert() 或者 confirm() 之类的函数；
* Web Workers 无法访问 window、document 之类的浏览器全局变量；

Web Workers 可以在它的作用域内访问 navigator 对象。它含有如下能够识别浏览器的字符串，就像在普通脚本中做的那样：

* appName
* appVersion
* platform
* userAgent

#### 4）Web Workers 的使用场景

* Web Workers 线程能够在不干扰 UI 的情况下执行一些计算的任务。
* Web Workers 线程能够使用 XMLHttpRequest 对象与服务器端进行异步交互。

#### 5）参考资料

* [Web Workers - W3C](http://www.w3.org/TR/workers/)
* [使用 Web Workers](https://developer.mozilla.org/zh-CN/docs/Web/API/Web_Workers_API/Using_web_workers)
* [Microsoft Web Workers](https://msdn.microsoft.com/library/hh673568(v=vs.85).aspx)

### 2. Worker API

#### 1）Worker 构造函数

Worker(DOMString scriptURL)

通过该构造函数可以直接创建 Worker 对象，并传递 Worker 文件的路径（scriptURL）。具体用法如下：

var worker = new Worker(my\_task.js);

通过 Worker 的构造函数得到 Worker 对象，在 HTML 页面与 Worker 文件之间创建通信。

**值得注意的是：**指定的 Worker 文件路径必须遵循同源策略。

#### 2）监听通信消息机制

worker.onmessage = function(event){

逻辑编写在这里...

}

当 Worker 传递消息给 HTML 页面（或 HTML 页面传递消息给 Worker）时，可以通过 onmessage 事件进行监听并且捕获。

传递的消息存储在 onmessage 事件处理函数的参数 data 中。

worker.onmessage = function(event) {

console.log("Worker's message: " + event.data);

};

#### 3）发送通信消息方法

postMessage(JSObject message)

HTML 页面通过该方法向 Worker 的内部作用域内传递消息。该方法接收一个单独的参数，即要传递给 Worker 的数据。

message 参数用于传递给 Worker 的数据，该数据将包含于传递给 onmessage 处理函数的事件对象中的 data 字段内。

// 普通数据

worker.postMessage("ali");

// JSON格式的数据

worker.postMessage({"cmd": "init", "timestamp": Date.now()});

#### 4）终止 Worker 通信

terminate()

该方法用于立即终止 Worker。该方法不会给 Worker 留下任何完成操作的机会；就是简单的立即停止。

#### 5）监听通信错误机制

worker.onerror = function(event){

错误处理编写在这里...

}

错误信息对象包含三个属性：

* message：一个可读性良好的错误信息。
* filename：产生错误的脚本文件名。
* lineno：发生错误时所在的脚本文件行号。

### 3. 双向通信

所谓“单向通信”就是指 HTML 页面与 Worker 建立通信后，只是 Worker 向 HTML 页面传递消息。

所谓“双向通信”就是指 HTML 页面与 Worker 之间可以相互传递消息的一种方式。

要建立双向通信，HTML 页面和 Worker 线程都要侦听 onmessage 事件。

* 首先，该脚本创建 worker 线程。

var echo = new Worker('echo.js');

echo.onmessage = function(e) {

alert(e.data);

}

* 当用户单击提交按钮时，脚本会将两条信息以 JavaScript 对象文本的形式传递给 Worker。

<script>

window.onload = function() {

var echoForm = document.getElementById('echoForm');

echoForm.addEventListener('submit', function(e) {

echo.postMessage({

message : e.target.message.value,

timeout : e.target.timeout.value

});

e.preventDefault();

}, false);

}

</script>

<form id="echoForm">

<p>Echo the following message after a delay.</p>

<input type="text" name="message" value="Input message here."/><br/>

<input type="number" name="timeout" max="10" value="2"/> seconds.<br/>

<button type="submit">Send Message</button>

</form>

* 最后，Worker 侦听消息，并在指定的超时间隔之后将其返回。

onmessage = function(e){

setTimeout(function(){

postMessage(e.data.message);

},

e.data.timeout \* 1000);

}

## Web 存储

在开发 Web 应用时，开发人员有时需要在本地存储数据。当前浏览器支持 cookie 存储，但其大小有 4KB 的限制。

HTML5 中新引入了 Web Storage 机制，通过使用键值对在客户端保存数据，并且提供了更大容量的存储空间。

HTML5 中的 Web 存储对象有两种类型：

* sessionStorage 对象负责存储一个会话的数据。如果用户关闭了页面或浏览器，则会销毁数据。
* localStorage 对象负责存储没有到期的数据。当 Web 页面或浏览器关闭时，仍会保持数据的存储，当然这还取决于为此用户的浏览器设置的存储量。

这两种存储对象具有相同的方法和属性。

**值得注意的是：**Web 存储并不比 cookies 安全。所以不要在客户端存储敏感信息，比如密码或信用卡信息。

官方规范资料地址：<https://html.spec.whatwg.org/multipage/webstorage.html>

### 1. sessionStorge

#### 1）浏览器支持性检查

//sessionStorage

if(window.sessionStorage){

alert(“support sessionStorage”);

}else{

alert(“not support sessionStorage”);

// 不支持 sessionStorage

}

#### 2）sessionStorage的方法

| **方法名称** | **描述** |
| --- | --- |
| setItem(key, value) | 为 Web 存储对象添加一个键/值对，供以后使用。该值可以是任何的数据类型：字符串、数值、数组等。 |
| getItem(key) | 基于起初用来存储它的这个键检索值。 |
| clear() | 从此 Web 存储对象清除所有的键/值对。 |
| removeItem(key) | 基于某个键从此 Web 存储对象清除特定的键/值对。 |
| key(n) | 检索 key[n] 的值。 |

##### a. 向 Web 存储对象添加键/值对

* 使用 setItem() 方法

sessionStorage.setItem('myKey', 'myValue');

* 使用 Web 存储对象来直接设置此键的值

sessionStorage.myKey = 'myValue';

##### b. 从 Web 存储对象中检索键/值对

* 使用 getItem() 方法

sessionStorage.getItem('myKey');

* 使用 Web 存储对象来直接获取此键的值

sessionStorage.myKey;

##### c. 删除所有键/值对

sessionStorage.clear();

##### d. 删除单个键/值对

sessionStorage.removeItem('myKey');

#### 示例代码

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>sessionStorage</title>

</head>

<body>

<input type="text" id="data">

<input type="button" id="save" value="保存">

<input type="button" id="read" value="读取">

<input type="button" id="dele" value="删除">

<script>

var save = document.getElementById("save");

save.onclick = function(){

// 将用户输入的数据,保存到sessionStorage中

// 1. 获取用户输入的数据

var value = document.getElementById("data").value;

// 2. 如何生成存储数据所使用的KEY

var key = new Date().getTime();// 时间戳

// 3. 存储数据

window.sessionStorage.setItem(key,value);

}

var read = document.getElementById("read");

read.onclick = function(){

// 读取sessionSotorage中所有的数据

// 1. 获取当前sessionStorage中所有数据的个数

var count = window.sessionStorage.length;

// 2. 进行遍历

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

// 3. 根据索引值得到key

var key = window.sessionStorage.key(i);

// 4. 根据key值得到value

var value = window.sessionStorage.getItem(key);

// 5. 进行测试

console.log(key + " : " + value);

}

}

var dele = document.getElementById("dele");

dele.onclick = function(){

// removeItem()

//window.sessionStorage.removeItem(1452666425707);

window.sessionStorage.clear();

}

</script>

</body>

</html>

### 2. localStorage

#### 1）浏览器支持性检查

//localStorage

if(window.localStorage){

alert(“support localStorage”);

}else{

alert(“not support localStorage”);

// 不支持 localStorage

}

#### 2）localStorage的方法

| **方法名称** | **描述** |
| --- | --- |
| setItem(key, value) | 为 Web 存储对象添加一个键/值对，供以后使用。该值可以是任何的数据类型：字符串、数值、数组等。 |
| getItem(key) | 基于起初用来存储它的这个键检索值。 |
| clear() | 从此 Web 存储对象清除所有的键/值对。 |
| removeItem(key) | 基于某个键从此 Web 存储对象清除特定的键/值对。 |
| key(n) | 检索 key[n] 的值。 |

##### a. 向 Web 存储对象添加键/值对

* 使用 setItem() 方法

localStorage.setItem('myKey', 'myValue');

* 使用 Web 存储对象来直接设置此键的值

localStorage.myKey = 'myValue';

##### b. 从 Web 存储对象中检索键/值对

* 使用 getItem() 方法

localStorage.getItem('myKey');

* 使用 Web 存储对象来直接获取此键的值

localStorage.myKey;

##### c. 删除所有键/值对

localStorage.clear();

##### d. 删除单个键/值对

localStorage.removeItem('myKey');

#### 3）localStorage的事件

如果想在存储成功或修改存储的值时执行一些操作，可以用 Web Storage 接口提供的事件。

// 显示存储事件的相关内容

function handleStorageEvent(e) {

document.write(“key” + e.key + “oldValue” + e.oldValue + “newValue” + e.newValue);

}

// 添加存储事件监听

window.addEventListener(“storage”, handleStorageEvent, false);

| **事件参数** | **描述** |
| --- | --- |
| key | 发生改变的键 |
| oldValue | 键改变之前的值 |
| newValue | 键改变之后的值 |
| url | 触发存储事件的页面 url |

## Web Sockets

### 1. 什么是 Web Sockets

#### 1）什么是 Web Sockets

Web Sockets 技术使得浏览器直接与服务器端的程序通过 socket 可以实时的推送或者获取信息的通讯方式成为可能。

在 HTML5 之前实现浏览器与服务器端实时通信的技术如下：

* 轮询：原理简单易懂，就是客户端通过一定的时间间隔以频繁请求的方式向服务器发送请求，来保持客户端和服务器端的数据同步。
* Flash：Flash 通过自己的 Socket 实现完成数据交换，再利用 Flash 暴露出相应的接口为 JavaScript 调用，从而达到实时传输目的。

#### 2）Web Sockets 的优势与劣势

##### a. Web Sockets 的优势

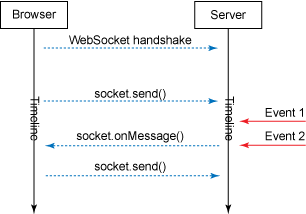
* Web Sockets 提供强大的、双向、低延迟和易于处理的错误。
* 它没有很多连接，比如：Comet 长轮询。

##### b. Web Sockets 的劣势

* Web Sockets 是 HTML5 的新规范，并不是所有浏览器都支持。
* 无请求作用域。由于 Web Sockets 是一个 TCP 套接字，而不是一个 HTTP 请求，因此无法轻松使用请求作用域服务。

#### 3）Web Sockets 的实现原理

下图展示了如何使用 WebSockets 进行通信：



1. HTTP 握手被发送到带有特定标头的服务器。
2. 在 JavaScript 的服务器或客户端上提供某种类型的套接字。
3. 使用该套接字来通过事件处理器异步接收数据。

### 2. Web Sockets API

#### 1）Web Sockets 的构造函数

var websocket = new WebSocket(url, protocols);

| **参数名称** | **描述** |
| --- | --- |
| url | 表示要连接的URL,这个URL应该是响应WebSocket的地址。 |
| protocols | 单个协议名称或字符串数组,默认为空字符串。 |

****值得注意的是：**Web Sockets 的请求地址（url）必须是 ws:// 或 wss://。**

var exampleSocket = new WebSocket("ws://www.example.com/socketserver");

#### 2）Web Sockets 的方法

| **方法名称** | **参数** | **描述** |
| --- | --- | --- |
| send(data) | data表示发送的请求数据。 | 通过 WebSocket 连接向服务器发送数据。 |
| close() |  | 关闭WebSocket连接或停止正在进行的连接请求。 |

exampleSocket.send("Here's some text that the server is urgently awaiting!");

#### 3）Web Sockets 的属性

Web Sockets 的属性主要使用的就是 readyState，该属性表示连接的当前状态。

| **常量名** | **值** | **描述** |
| --- | --- | --- |
| CONNECTING | 0 | 连接还没开启。 |
| OPEN | 1 | 连接已开启并准备好进行通信。 |
| CLOSING | 2 | 连接正在关闭的过程中。 |
| CLOSED | 3 | 连接已经关闭，或者连接无法建立。 |

#### 4）Web Sockets 的事件

| **事件名称** | **描述** |
| --- | --- |
| onopen | 用于监听 Web Sockets 打开的事件。 |
| onmessage | 用于监听 Web Sockets 服务器端传递消息的事件。 |
| onerror | 用于监听 Web Sockets 发生错误的事件。 |
| onclose | 用于监听 Web Sockets 通信关闭的事件。 |

#### 5）Web Sockets 使用步骤

var ws = new WebSocket('ws://127.0.0.1:8080/async');

ws.onopen = function() {

// called when connection is opened

};

ws.onerror = function(e) {

// called in case of error, when connection is broken in example

};

ws.onclose = function() {

// called when connexion is closed

};

ws.onmessage = function(msg) {

// called when the server sends a message to the client.

// msg.data contains the message.

};

// Here is how to send some data to the server

ws.send('some data');

// To close the socket:

ws.close();

### 3. Web Sockets 案例

* 客户端代码示例：

// 假设服务端ip为127.0.0.1

ws = new WebSocket("ws://127.0.0.1:2346");

ws.onopen = function() {

alert("连接成功");

ws.send('tom');

alert("给服务端发送一个字符串：tom");

};

ws.onmessage = function(e) {

alert("收到服务端的消息：" + e.data);

};

* 服务器端代码示例：

这里利用了 PHP 的第三方框架 Workerman 实现 Web Sockets 的服务器端逻辑。

<?php

use Workerman\Worker;

require\_once '/Workerman/Autoloader.php';

// 创建一个Worker监听2346端口，使用websocket协议通讯

$ws\_worker = new Worker("websocket://0.0.0.0:2346");

// 启动4个进程对外提供服务

$ws\_worker->count = 4;

// 当收到客户端发来的数据后返回hello $data给客户端

$ws\_worker->onMessage = function($connection, $data)

{

// 向客户端发送hello $data

$connection->send('hello ' . $data);

};

// 运行worker

Worker::runAll();

?>

**原创声明：**如发布或转载请注明以下作者信息，谢谢！

本文作者 - 金云龙

联系方式 - QQ：2080954535；微信：imlonge；微信公众号：longestory

个人网站：<http://www.longestory.com>