WEB前端

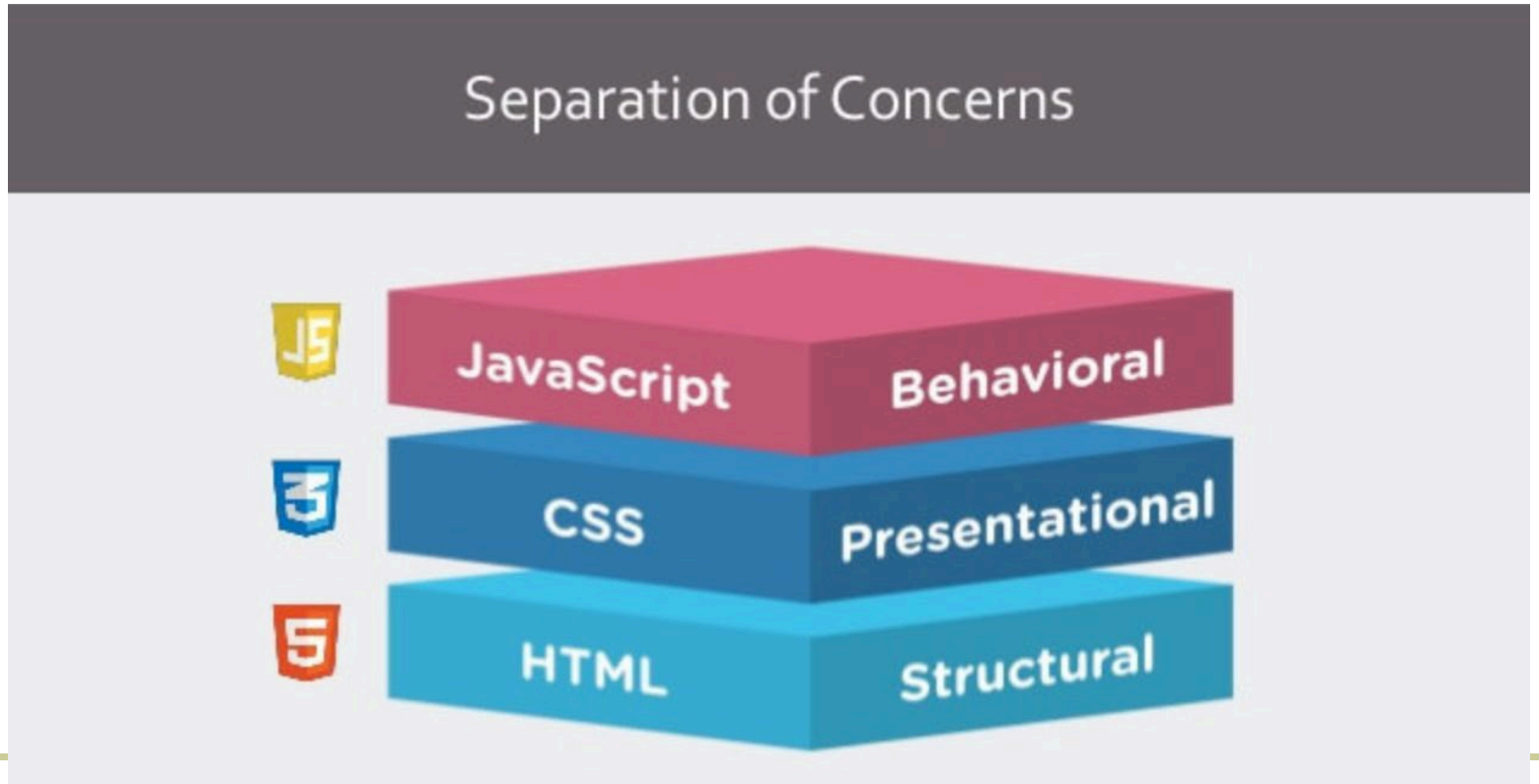
南京大学软件学院

互联网计算

网页开发的原则

关注点分离：各种技术只负责自己的领域，不要混合在一起，形成耦合。

对于网页开发来说，主要是三种技术分离：



网页开发的原则

- **HTML 语言**

  负责网页的结构，又称语义层。

- **CSS 语言**

  负责网页的样式，又称视觉层。

- **JavaScript 语言**

  负责网页的逻辑和交互，又称逻辑层或交互层。

HTML

- HTML简介
- HTML标签
- 剖析一个HTML元素
- 元素的属性
- 分析HTML文档
- 学习网址
- 特殊字符
- HTML语法：注释、块级元素和内联元素等
- HTML的CSS样式

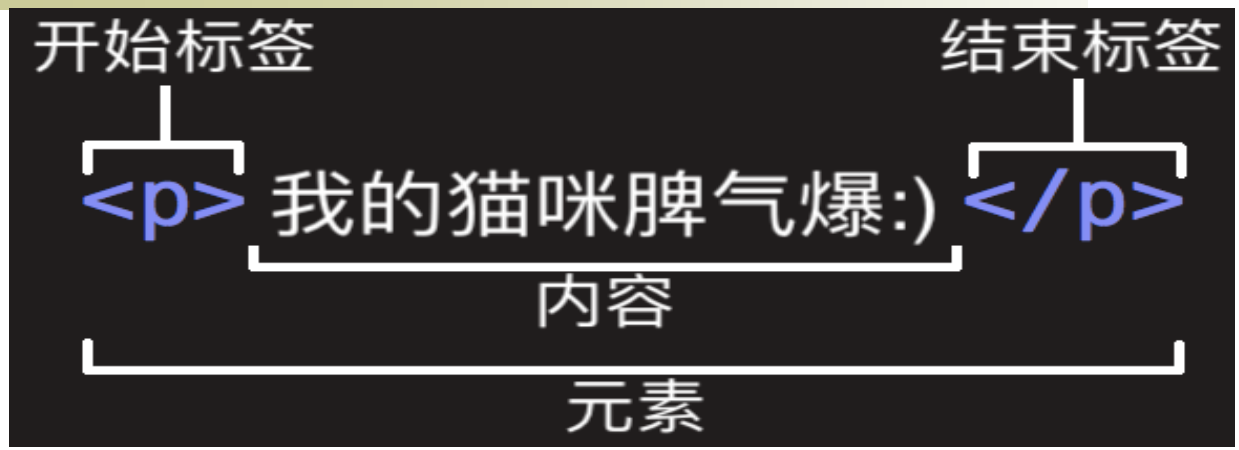
HTML简介

- HTML (Hyper Text Markup Language),是一种超文本标记语言，用来描述网页。
- HTML由一系列的元素（elements）组成，这些元素可以用来包围不同部分的内容，使其以某种方式呈现。
- Web 浏览器的作用是读取 HTML 文档，并以网页的形式显示出它们。浏览器不会显示 HTML 标签，而是使用标签来解释页面的内容。
- 一个HTML文件的后缀名是.htm 或者是.html。
- 用文本编辑器就可以编写HTML文件。

HTML标签

- HTML使用标记标签来描述网页。
  - HTML标签是由尖括号包围的关键词，比如 <html>。
  - HTML 标签通常是成对出现的，比如 和 。标签对中的第一个标签是开始标签，第二个标签是结束标签。
  - HTML 标签不区分大小写。也就是说，输入标签时既可以使用大写字母也可以使用小写字母。例如，标签 <title> 写作<title>、<TITLE>、<Title>、<TiTlE>，等等都可以正常工作。不过，从一致性、可读性等各方面来说，最好仅使用小写字母。

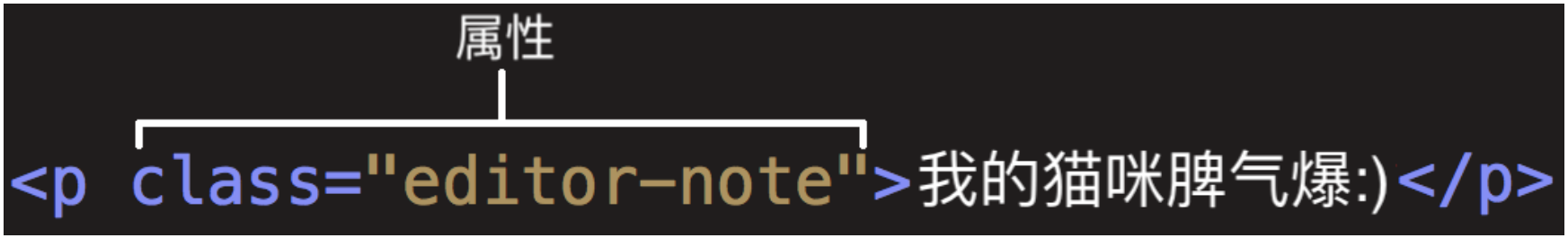
剖析一个HTML元素



这个元素的主要部分有：

- **开始标签**：包含元素的名称（上图中为 p），被<、>所包围。
  表示元素从这里开始或者开始起。
- **结束标签**：与开始标签相似，只是其在元素名之前包含了一个斜杠。
  表示着元素的结尾 。
- **内容**：元素的内容，本例中就是所输入的文本本身。
- **元素**：开始标签、结束标签与内容相结合，便是一个完整的元素。

元素的属性



属性包含元素的额外信息。在上述例子中，这个class属性给元素赋了一个识别的名字，这个名字此后可以被用来识别此元素的样式信息和其他信息。

一个属性必须包含如下内容：

1.在元素和属性之间有个空格space (如果有一个或多个已存在的属性，就与前一个属性之间有一个空格.)

2.属性后面紧跟着一个“=”符号.

3.有一个属性值,由一对引号“ ”引起来.

元素的属性

```
<a href=“https://www.mozilla.org/” title=“链接元素”></a>
```

元素<a>是锚，它使被标签包裹的内容成为一个超链接。

此元素也可以添加大量的属性，其中几个如下：

- **href:** 这个属性声明超链接的web地址，当这个链接被点击，浏览器会跳转至href声明的web地址。例如：href=“https://www.mozilla.org/”。
- **title:** 标题title属性为超链接声明额外的信息，比如你将链接至那个页面。例如：title=“The Mozilla homepage”。当鼠标悬浮时，将出现一个工具提示。
- **target:** 目标target属性指定将用于显示链接的浏览上下文。例如，target=“_blank”将在新标签页中显示链接。如果你希望在目前标签页显示链接，只需忽略这个属性。

分析HTML文档

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>我的测试站点</title>
6    </head>
7    <body>
8      <p>这是我的页面</p>
9    </body>
10 </html>
```

<!DOCTYPE html>：声明文档类型。
类型声明类似于链接，规定了HTML页面必须遵从的良好规则，能自动检测错误和其他有用的东西。你只需要知道<!DOCTYPE html>是最短的有效的文档声明。

分析HTML文档

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>我的测试站点</title>
6    </head>
7    <body>
8      <p>这是我的页面</p>
9    </body>
10 </html>
```

<html></html>: <html>元素。
这个元素包裹了整个完整的页面，是一个根元素。

分析HTML文档

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>我的测试站点</title>
6    </head>
7    <body>
8      <p>这是我的页面</p>
9    </body>
10 </html>
```

<head></head>: <head>元素。
这个元素是一个容器，它包含了所有你想包含在HTML页面中，但不想在HTML页面中显示的内容。这些内容包括CSS样式，字符集声明等等。

分析HTML文档

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>我的测试站点</title>
6    </head>
7    <body>
8      <p>这是我的页面</p>
9    </body>
10 </html>
```

<meta charset="utf-8">: 这个元素设置文档使用utf-8字符集编码。
utf-8字符集包含了人类大部分的文字。基本上能识别你放上去的所有文本内容。
毫无疑问要使用它，并且它能在以后避免很多其他问题。

分析HTML文档

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>我的测试站点</title>
6    </head>
7    <body>
8      <p>这是我的页面</p>
9    </body>
10 </html>
```

<title></title>: 页面标题。
出现在浏览器标签上，当你标记/收藏页面时它可用来描述页面。

分析HTML文档

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>我的测试站点</title>
6    </head>
7    <body>
8      <p>这是我的页面</p>
9    </body>
10 </html>
```

<body></body>: <body>元素。
包含了你访问页面时所有显示在页面上的内容，文本，图片，音频，游戏等等。

特殊字符

在HTML中，字符 < > “ ‘ & 是特殊字符，它们是HTML语法自身的一部分,。
如何将这些字符包含进你的文本中呢？

——使用字符引用, 每个字符引用以符号&开始, 以分号(;)结束。

原义字符	等价字符引用
<	<
>	>
"	"
'	'
&	&

特殊字符

```
1 <p>HTML 中用 <p> 来定义段落元素。</p>
2
3 <p>HTML 中用 &lt;p&gt; 来定义段落元素</p>
```

HTML 中用

来定义段落元素。

HTML 中用 <p> 来定义段落元素

上面的代码实例和结果展示中，我们可以看出：
第一行是错误的，因为浏览器会认为第二个<p>是开始一个新的段落。
第三行是正确的，因为我们用字符引用来代替了角括号（‘<’和‘>’符号）。

HTML注释

如同大部分的编程语言一样，在HTML中有一种可用的机制来在代码中书写注释。

为了将一段HTML中的内容置为注释，你需要将其用特殊的记号<!--和-->包括起来：比如：

```
<!-- <p>我在注释内！</p> -->
```

块级元素和内联元素

在HTML中有两种元素类别，块级元素和内联元素：

•块级元素

- 在页面中以块的形式展现。相对与其前面的内容它会出现在新的一行，其后的内容也会被挤到下一行展现。
- 通常用于展示页面上结构化的内容，例如段落、列表、导航菜单。
- 一个以block形式展现的块级元素不会被嵌套进内联元素中，但可以嵌套在其它块级元素中。

•内联元素

- 通常出现在块级元素中并包裹文档内容的一小部分，而不是一整个段落或者一组内容。
- 内联元素不会导致文本换行。
- 通常出现在一堆文字之间，例如超链接元素<a>。

块级元素和内联元素

```
1 <em>第一</em><em>第二</em><em>第三</em>
2
3 <p>第四</p><p>第五</p><p>第六</p>
```

HTML语句

 是一个内联元素，所以第一行代码中的三个元素都没有间隙的展示在了同一行。<p>是一个块级元素，所以第二行代码中的每个元素分别都另起了新的一行展现，并且每个段落间都有一些间隔。（这是因为默认的浏览器有着默认的展示<p>元素的CSS styling）。

展示结果

第一第二第三

第四

第五

第六

块级元素和内联元素

常见的块级元素：

div , dl , form , h1 , h2 , h3 , h4 , h5 , h6 , menu , ol , p , table , ul , li

块级元素的宽度始终是与浏览器宽度一样，与内容无关；

常用的内联元素：

a , em , strong , font , img , input , label , select , span , textarea , cite , dfn

内联元素的宽度随着内容增加，高度随字体大小而改变。
内联元素可以设置外边界，但是外边界不对上下起作用，只能对左右起作用。

HTML的CSS样式

当浏览器读到一个样式表，它就会按照这个样式表来对文档进行格式化。
有以下三种方式来插入样式表：

1. 外部样式表

当样式需要被应用到很多页面的时候，外部样式表将是理想的选择。
使用外部样式表，你就可以通过更改一个文件来改变整个站点的外观。

```
<head>
    <link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

HTML的CSS样式

2.内部样式表

当单个文件需要特别样式时，就可以使用内部样式表。
可以在 head 部分通过 <style> 标签定义内部样式表。

```
<head>
    <style type="text/css">
        body {background-color: red}
        p {margin-left: 20px}
    </style>
</head>
```

3.内联样式

当特殊的样式需要应用到个别元素时，就可以使用内联样式。
使用内联样式的方法是在相关的标签中使用样式属性，可以包含任何 CSS 属性。

```
<p style="color: red; margin-left: 20px">
    This is a paragraph
</p>
```

学习网址

- http://www.w3school.com.cn/html/html_getstarted.asp
- https://developer.mozilla.org/zh-CN/docs/Learn/HTML/Introduction_to_HTML/Getting_started
- http://www.ruanyifeng.com/blog/2009/05/guide_to_semantic_html_elements.html

CSS

- CSS简介
- 层叠次序
- CSS语法
- CSS的选择器
- 学习网址

CSS简介

- CSS 指层叠样式表 (Cascading Style Sheets)，定义如何显示 HTML 元素。
- 样式表允许以多种方式规定样式信息。样式可以规定在单个HTML元素中，在HTML页头元素中，在一个外部CSS文件中，甚至可以在同一个HTML文档内部引用多个外部样式表。
- 更多情况下，样式保存在外部的 .css 文件中。通过仅仅编辑一个简单的 CSS 文档，外部样式表使你有能力同时改变站点中所有页面的布局和外观，极大提高工作效率。
- 多个样式定义可层叠为一。

层叠次序

当同一个 HTML 元素被不止一个样式定义时，会使用哪个样式呢？

一般而言，所有的样式会根据下面的规则层叠于一个新的虚拟样式表中，
其中数字 4 拥有最高的优先权。

1.浏览器缺省设置。

2.外部样式表。

3.内部样式表（位于 <head> 标签内部）。

4.内联样式（在 HTML 元素内部）。

因此，内联样式（在 HTML 元素内部）拥有最高的优先权，
这意味着它将优先于以下的样式声明：

•<head> 标签中的样式声明

•外部样式表中的样式声明

•或者浏览器中的样式声明（缺省值）。

CSS语法

CSS 规则由两个主要的部分构成：选择器，以及一条或多条声明。

```
selector {declaration1; declaration2; ... declarationN }
```

选择器通常是您需要改变样式的 HTML 元素。

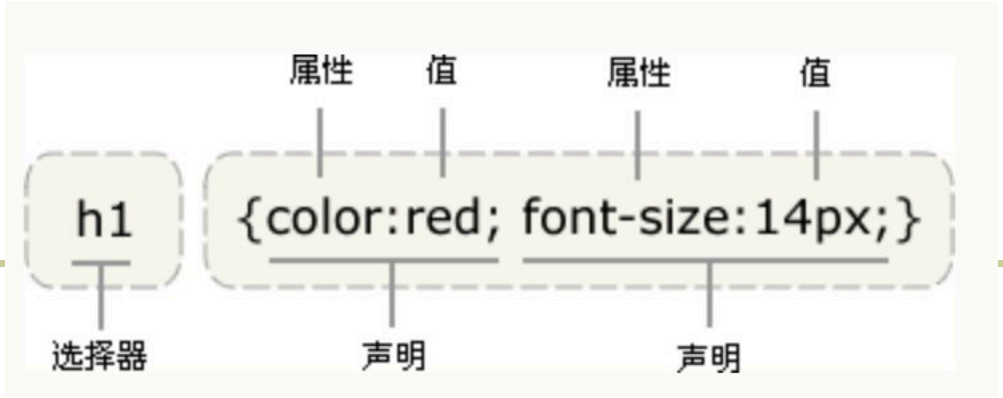
每条声明由一个属性和一个值组成。属性和值被冒号分开。

属性（property）是希望设置的样式属性（style attribute）。

```
selector {property: value}
```

下面代码是将 h1 元素内的文字颜色定义为红色，同时将字体大小设置为 14 像素。

```
h1 {color:red; font-size:14px;}
```



CSS的选择器

- **派生选择器（上下文选择器）**
  - 依据元素在其位置的上下文关系来定义样式。
  - 通过合理地使用派生选择器，我们可以使 HTML 代码变得更加整洁。

比如，希望列表中的 strong 元素变为斜体字，而不是通常的粗体字，
可以这样定义一个派生选择器：

```
li strong {
    font-style: italic;
    font-weight: normal;
}
```

CSS的选择器

- **id 选择器**
  - 可以为标有特定 id 的 HTML 元素指定特定的样式。
  - 以 "#" 来定义。

可以这样定义一个id选择器：下面的两个 id 选择器，
第一个可以定义元素的颜色为红色，第二个定义元素的颜色为绿色：

```
#red {color:red;}
#green {color:green;}
```

CSS的选择器

- **属性选择器**
  - 对带有指定属性的 HTML 元素设置样式。
  - 只有在规定了 !DOCTYPE 时，IE7 和 IE8 才支持属性选择器。在 IE6 及更低的版本中，不支持属性选择。

可以这样定义一个属性选择器：
下面的例子为带有 title 属性的所有元素设置样式：

```
[title]
{
color:red;
}
```

学习网址

- http://www.w3school.com.cn/css/css_jianjie.asp
- https://developer.mozilla.org/zh-CN/docs/Learn/CSS/Introduction_to_CSS
- http://www.ruanyifeng.com/blog/2010/03/css_cookbook.html

Bootstrap

- Bootstrap简介
- Bootstrap结构
- Bootstrap的简单使用
- Bootstrap CSS
- Bootstrap布局组件
- Bootstrap插件
- 学习网址

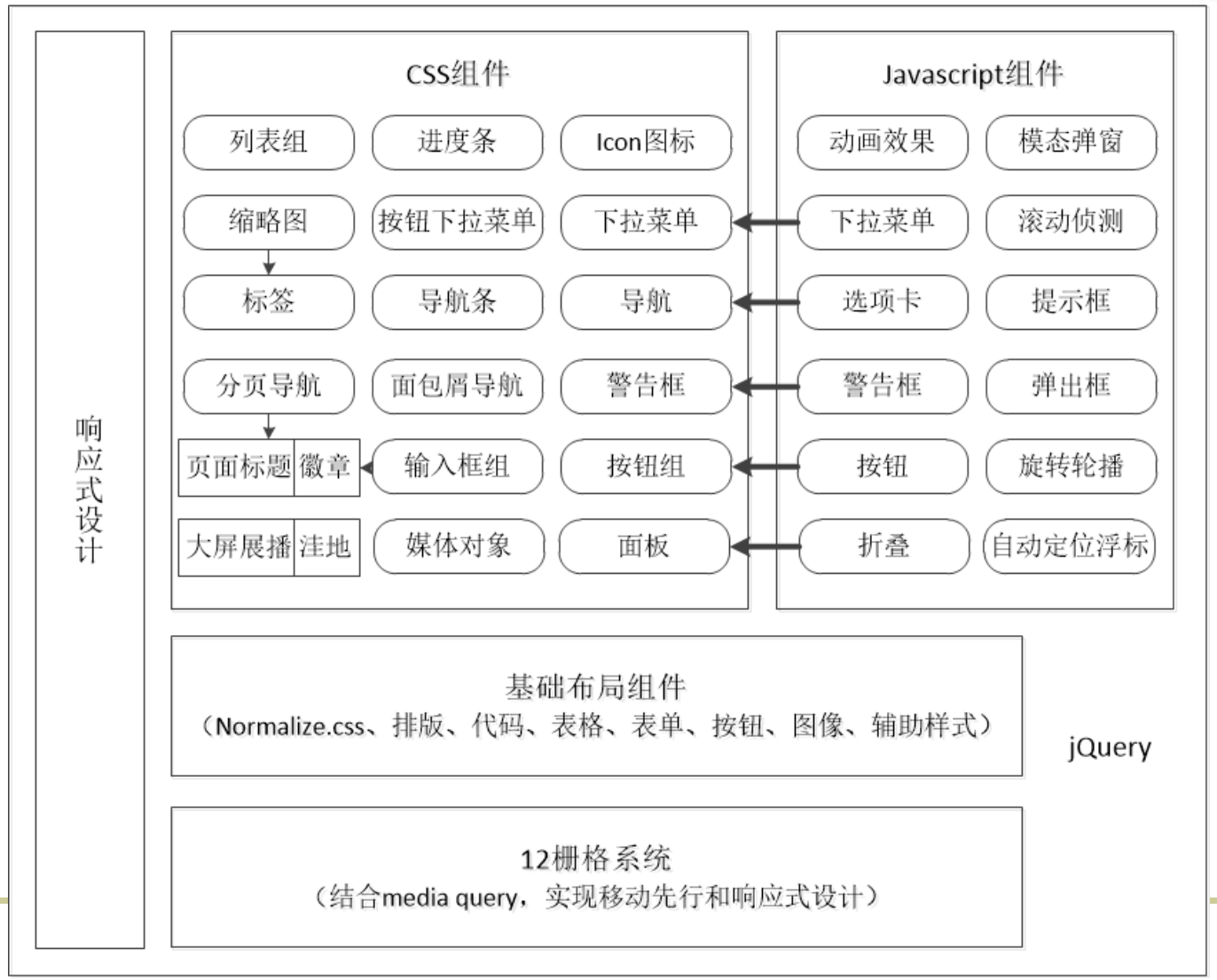
Bootstrap简介

- Bootstrap 是由 Twitter 的 Mark Otto 和 Jacob Thornton 开发的，用于快速开发 Web 应用程序和网站的前端框架。
- Bootstrap是基于 HTML、CSS、JAVASCRIPT 的前端框架，实际上是CSS样式的合集。

Bootstrap特点

- 移动设备优先：自 Bootstrap 3 起，框架包含了贯穿于整个库的移动设备优先的样式。
- 浏览器支持：所有的主流浏览器都支持 Bootstrap。
- 响应式设计：Bootstrap 的响应式 CSS 能够自适应于台式机、平板电脑和手机。
- 容易上手：只要具备 HTML 和 CSS 的基础知识，就可以开始学习 Bootstrap。

Bootstrap结构



Bootstrap使用

- 下载bootstrap:http://getbootstrap.com/

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   ├── bootstrap-theme.css
│   ├── bootstrap-theme.css.map
│   └── bootstrap-theme.min.css
├── js/
│   ├── bootstrap.js
│   └── bootstrap.min.js
└── fonts/
    ├── glyphicons-halflings-regular.eot
    ├── glyphicons-halflings-regular.svg
    ├── glyphicons-halflings-regular.ttf
    ├── glyphicons-halflings-regular.woff
    └── glyphicons-halflings-regular.woff2
```

Bootstrap使用

- 在html文档中加载bootstrap相关的文件（jquery.js、bootstrap.min.js 和 bootstrap.min.css 文件）

  Bootstrap 所有 JavaScript 插件都依赖 jQuery，因此 jQuery 必须在 Bootstrap 之前引入， jQuery 也必须使用最新版。

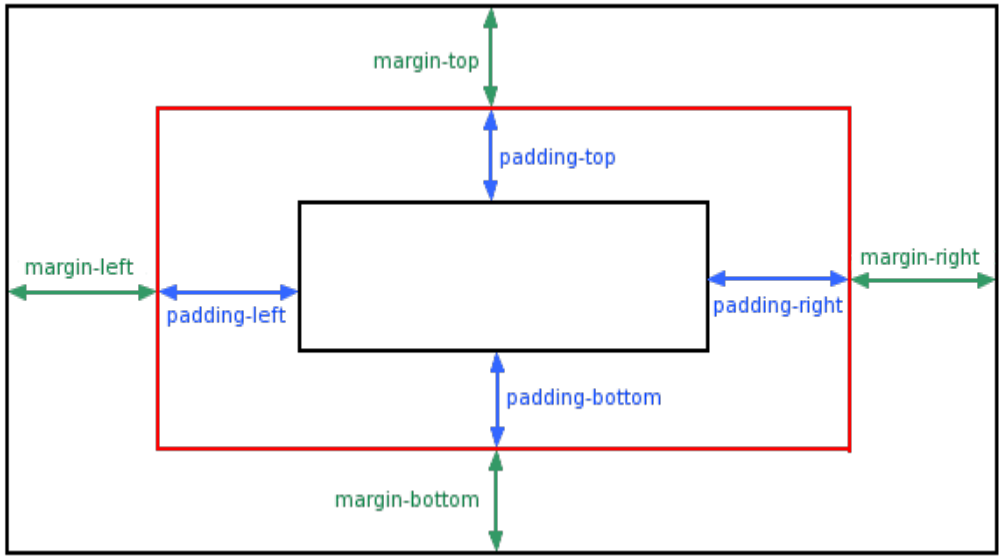
- 为了Bootstrap开发的网站对移动设备友好，确保适当的显示和触屏缩放，需要在网页的head中增加viewport meta 标签

Bootstrap使用

```
<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Bootstrap example</title>
<link rel="stylesheet" href="http://js.biocloud.cn/bootstrap/3.3.4/css/bootstrap.min.css">
    <!--[if lt IE 9]>
    <script src="http://js.biocloud.cn/html5shiv/3.7.2/html5shiv.min.js"></script>
    <script src="http://js.biocloud.cn/respond.js/1.4.2/respond.min.js"></script>
    <![endif]-->
  </head>
  <body>
    <h1>Hello World! </h1>
    <script src="http://js.biocloud.cn/jquery/1.11.3/jquery.min.js"></script>
    <script src="http://js.biocloud.cn/bootstrap/3.3.4/js/bootstrap.min.js"></script>
  </body>
</html>
```

user-scalable=no
maximum-scale=1.0

CSS 简介—盒子模型



总元素的宽度=宽度+左填充+右填充+左边框+右边框+左边距+右边距

总元素的高度=高度+顶部填充+底部填充+上边框+下边框+上边距+下边距

Bootstrap CSS

- Bootstrap网格系统
- Bootstrap排版
- Bootstrap表格
- Bootstrap表单
- Bootstrap按钮

Bootstrap网格系统

- Bootstrap包含了一个响应式的、移动设备优先的、不固定的网格系统，可以随着设备和视口大小的增加而适当的增加到最多12列
- 移动设备优先策略

  内容：决定什么是最重要的

  布局：优先设计更小的宽度

  渐进增强：随着屏幕大小的增加而添加元素

Bootstrap网格系统

- 行必须放置在 .container class 内
- 使用行来创建列的水平组
- 内容应该放置在列内，且唯有列可以是行的直接子元素。
- 预定义的网格类，比如 .row 和 .col-xs-4，可用于快速创建网格布局。
- 列通过padding来创建列内容之间的间隙。该内边距是通过 .rows 上的margin取负，表示第一列和最后一列的行偏移
- 网格系统是通过指定想要横跨的12个可用的列来创建的

Bootstrap网格系统

1	1	1	1	1	1	1	1	1	1	1	1
.col-*-4				.col-*-4				.col-*-4			
.col-*-4				.col-*-8							
.col-*-6								.col-*-4			
.col-*-12											

Bootstrap网格系统

	额外的小设备收集（<768px）	小型设备平板电脑（≥768px）	中型设备台式电脑（≥992px）	大型设备台式电脑（≥1200px）
网格行为	一直是水平的	以折叠开始，断点以上是水平的	以折叠开始，断点以上是水平的	以折叠开始，断点以上是水平的
最大容器宽度	None (auto)	750px	970px	1170px
Class 前缀	.col-xs-	.col-sm-	.col-md-	.col-lg-
列 #	12	12	12	12
最大列宽	Auto	60px	78px	95px

Bootstrap网格列偏移

```
<div class="container">
    <div class="row" >
        <div class="col-md-6 col-md-offset-3"
        style="…">
            <p>This is an example    </p>
        </div>
    </div>
</div>
```

.col-md-offset-* 1-11

Bootstrap网格列嵌套

```
<div class="container">
    <div class="row">
        <div class="col-md-12" style="…">
            <h4>分为四个盒子</h4>
            <div class="row">
                <div class="col-md-6" style="…"></div>
                <div class="col-md-6" style="…"></div>
            </div>
            <div class="row">
                <div class="col-md-6" style="…"></div>
                <div class="col-md-6" style="…"></div>
            </div>
        </div>
    </div>
</div>
```

Bootstrap排版

- 内联子标题

  <h3>我是标题<small>我是副标题</small></h3>

- 列表

  有序列表

  无序列表

  未定义样式列表 <ul class="list-unstyled">

  内联列表 <ul class="list-inline">

  定义列表 <dl></dl>

  内联列表 <dl class="dl-horizontal"></ dl>

Bootstrap排版

- 强调 <small>

```
<small>本行内容是在标签内</small><br>
<strong>本行内容是在标签内</strong><br>
<em>本行内容是在标签内，并呈现为斜体</em><br>
<p class="text-left">向左对齐文本</p>
<p class="text-center">居中对齐文本</p>
<p class="text-right">向右对齐文本</p>
<p class="text-muted">本行内容是减弱的</p>
<p class="text-primary">primary class</p>
<p class="text-success">success class</p>
<p class="text-info">info class</p>
<p class="text-warning">warning class</p>
<p class="text-danger">danger class</p>
```

primary class

success class

info class

warning class

danger class

Bootstrap表格

标签	描述
<table>	为表格添加基础样式。
<thead>	表格标题行的容器元素（<tr>），用来标识表格列。
<tbody>	表格主体中的表格行的容器元素（<tr>）。
<tr>	一组出现在单行上的表格单元格的容器元素（<td> 或 <th>）。
<td>	默认的表格单元格。
<th>	特殊的表格单元格，用来标识列或行（取决于范围和位置）。必须在 <thead> 内使用。
<caption>	关于表格存储内容的描述或总结。

Bootstrap表单

- 垂直表单 <form role="form">
- 内联表单 <form class="form-inline" role="form">
- 水平表单 <form class="form-horizontal" role="form">

◆ …

◆向所有的文本元素 <input>、<textarea> 和 <select> 添加class ="form-control"

Bootstrap表单控件

- 输入框input

Type： text、password、datetime、datetime-local、date、month、time、week、number、email、url、search、tel 、 color

- 文本框textarea
- 复选框Checkbox
- 单选框Radio
- 选择框Select

Bootstrap按钮

- 任何带有 class .btn 的元素都会继承圆角灰色按钮的默认外观
- 提供一些选项来定义按钮样式，可用于<a>，<button>，或<input>元素上
- 建议您在<button>元素上使用按钮 class，避免跨浏览器的不一致性问题
- 控制按钮大小 .btn-lg/sm/xs/block

Bootstrap按钮

```
<button type="button" class="btn btn-default">默认按钮</button>
<button type="button" class="btn btn-primary">原始按钮</button>
<button type="button" class="btn btn-success">成功按钮</button>
<button type="button" class="btn btn-info">信息按钮</button>
<button type="button" class="btn btn-warning">警告按钮</button>
<button type="button" class="btn btn-danger">危险按钮</button>
<button type="button" class="btn btn-link">链接按钮</button>
```

默认按钮 原始按钮 成功按钮 信息按钮 警告按钮 危险按钮 链接按钮

Bootstrap布局组件

- 字体图标
- 下拉菜单
- 按钮组
- 按钮下拉菜单
- 输入框组
- 导航元素
- 导航栏
- 面包屑导航
- 分页
- 标签
- 徽章
- 超大屏幕
- 页面标题
- 缩略图
- 警告
- 进度条
- 多媒体对象
- 列表组
- 面板

Bootstrap插件

- Bootstrap 自带 12 种 jQuery 插件，扩展了功能，可以给站点添加更多的互动
- 单独引用插件。使用 Bootstrap 的个别的 *.js 文件。一些插件和 CSS 组件依赖于其他插件。
- 编译（同时）引用插件。使用 bootstrap.js 或压缩版的 bootstrap.min.js。

学习网址

- Bootstrap学习
  - http://www.runoob.com/bootstrap/bootstrap-tutorial.html
  - https://v3.bootcss.com/
  - http://how2j.cn/k/boostrap/boostrap-jumbotron/495.html#nowhere
- CSS学习
  - https://www.runoob.com/css/css-tutorial.html
  - http://zh.learnlayout.com
  - http://css.doyoe.com/

JavaScript

- JavaScript简介
- JavaScript使用
- JavaScript基本语法
- JavaScript异常处理
- 学习网址

JavaScript简介

JavaScript 是一门跨平台、面向对象的脚本语言，它能够让网页具有交互性（例如具有复杂的动画，可点击的按钮，通俗的菜单等）。另外还有高级的服务端JavaScript版本，例如Node.js，它可以让你在网页上添加更多功能，不仅仅是下载文件（例如在多台电脑之间的协同合作）。

JavaScript 内置了一些对象的标准库，比如数组（Array），日期（Date），数学（Math）和一套核心语句，包括运算符，流程控制符以及申明方式等。JavaScript 的核心部分可以通过添加对象来扩展语言以适应不同用途。

JavaScript简介

- 客户端的 JavaScript 通过提供控制浏览器及其 DOM 对象来扩展语言核心。例如：客户端版本直接支持应用将元素放在HTML表单中并且支持响应用户事件，比如鼠标点击、表单提交和页面导航。

- 服务端的 JavaScript 则通过提供有关在服务器上运行 JavaScript 的对象来可扩展语言核心。例如：服务端版本直接支持应用和数据库通信，提供应用不同调用间的信息连续性，或者在服务器上执行文件操作。

JavaScript使用

如需在 HTML 页面中插入 JavaScript，请使用 <script> 标签。

<script> 和 </script> 之间的代码行包含了 JavaScript：

```
<script>
    alert("My First JavaScript");
</script>
```

HTML 中的脚本必须位于 <script> 与 </script> 标签之间。脚本可被放置在 HTML 页面的 <body> 和 <head> 部分中。也可以把脚本保存到外部文件中。外部文件通常包含被多个网页使用的代码。

外部 JavaScript 文件的文件扩展名是 .js。

如需使用外部文件，请在 <script> 标签的 "src" 属性中设置该 .js 文件：

```
<!DOCTYPE html>
<html>
<body>
    <script src="myScript.js"></script>
</body>
</html>
```

JavaScript基本语法

JavaScript 是区分大小写的，并使用 Unicode 字符集。
举个例子，可以将单词 Früh （在德语中意思是“早”）用作变量名。

```
var Früh = "foobar";
```

但是，由于 JavaScript 是大小写敏感的，因此变量 früh 和 Früh 则是两个不同的变量。

在 JavaScript 中，指令被称为语句 Statement，并用分号（;）进行分隔。
如果一条语句独占一行的话，那么分号是可以省略的。
但如果一行中有多条语句，那么这些语句必须以分号分开。
虽然不是必需的，但是在一条语句的末尾加上分号可以大大减少代码中产生 bug 的可能性。

JavaScript基本语法

注释

Javascript 注释的语法和 C++ 或许多其他语言类似：

```
1 // 单行注释
2
3 /* 这是一个更长的，
4    多行注释
5 */
6
7 /* 然而，你不能，/* 嵌套注释 */ 语法错误 */
```

在代码执行过程中，注释将被自动跳过（不执行）。

JavaScript基本语法

变量

在应用程序中，使用变量来作为值的符号名。变量的名字又叫做标识符，其需要遵守一定的规则。一个 JavaScript 标识符必须以字母、下划线（_）或者美元符号（$）开头，后续的字符也可以是数字（0-9）。因为 JavaScript 语言是区分大小写的，所以字母可以是从“A”到“Z”的大写字母和从“a”到“z”的小写字母。

变量声明

Javascript 有三种声明方式：

- var：声明一个变量，可选初始化一个值。
- let：声明一个块作用域的局部变量，可选初始化一个值。
- const:声明一个块作用域的只读常量。

JavaScript基本语法

数据类型

- 数字
- 字符串
- 布尔类型
- 数组
- 对象
- Undefined 和 Null

JavaScript基本语法

声明变量类型

当声明新变量时，可以使用关键词 "new" 来声明其类型：

```
var carname = new String;
var x = new Number;
var y = new Boolean;
var cars = new Array;
var person = new Object;
```

JavaScript 变量均为对象。当声明一个变量时，就创建了一个新的对象。

JavaScript基本语法

变量的作用域

在函数之外声明的变量，叫做全局变量，因为它可被当前文档中的任何其他代码所访问。

在函数内部声明的变量，叫做局部变量，因为它只能在当前函数的内部访问。

变量提升

JavaScript 变量的另一个不同寻常的地方是，你可以先使用变量稍后再声明变量而不会引发异常。这一概念称为变量提升；JavaScript 变量感觉上是被“提升”或移到了函数或语句的最前面。但是，提升后的变量将返回 undefined 值。因此在使用或引用某个变量之后进行声明和初始化操作，这个被提升的变量仍将返回 undefined 值。

JavaScript基本语法

变量提升

左边的例子也可写作右边：

```
/**
 * 例子1
 */
console.log(x === undefined); // true
var x = 3;


/**
 * 例子2
 */
// will return a value of undefined
var myvar = "my value";

(function() {
  console.log(myvar); // undefined
  var myvar = "local value";
})();
```

```
/**
 * 例子1
 */
var x;
console.log(x === undefined); // true
x = 3;

/**
 * 例子2
 */
var myvar = "my value";

(function() {
  var myvar;
  console.log(myvar); // undefined
  myvar = "local value";
})();
```

JavaScript异常处理

可以用 throw 语句抛出一个异常并且用 try...catch 语句捕获处理它。

throw语句

使用throw语句抛出一个异常。当你抛出异常，你规定一个含有值的表达式要被抛出。

你可以抛出任意表达式而不是特定一种类型的表达式。

下面的代码抛出了几个不同类型的表达式：

```
throw "Error2";   // String type
throw 42;         // Number type
throw true;       // Boolean type
throw {toString: function() { return "I'm an object!"; } };
```

JavaScript异常处理

try...catch语句

下面的例子使用了try...catch语句。

示例调用了一个函数用于从一个数组中根据传递值来获取一个月份名称。

如果该值与月份数值不相符，会抛出一个带有“InvalidMonthNo”值的异常，

然后在捕捉块语句中设置monthName变量为unknown。

```
function getMonthName(mo) {
  mo = mo - 1; // Adjust month number for array index (1 = Jan, 12 = Dec)
  var months = ["Jan","Feb","Mar","Apr","May","Jun","Jul",
                "Aug","Sep","Oct","Nov","Dec"];
  if (months[mo]) {
    return months[mo];
  } else {
    throw "InvalidMonthNo"; //throw keyword is used here
  }
}

try { // statements to try
  monthName = getMonthName(myMonth); // function could throw exception
}
catch (e) {
  monthName = "unknown";
  logMyErrors(e); // pass exception object to error handler -> your own function
}
```

JavaScript异常处理

finally块

finally块包含了在try和catch块完成后、下面接着try...catch的语句之前执行的语句。finally块无论是否抛出异常都会执行。

下面的例子中，如果在文件打开时有异常抛出，finally块会在脚本错误之前关闭文件。

```
openMyFile();
try {
    writeMyFile(theData); //This may throw a error
}catch(e){
    handleError(e); // If we got a error we handle it
}finally {
    closeMyFile(); // always close the resource
}
```

学习网址

- http://www.w3school.com.cn/js/js_intro.asp
- https://developer.mozilla.org/zh-CN/docs/Learn/JavaScript
- http://javascript.ruanyifeng.com/introduction/intro.html

JQuery

- JQuery简介
- JQuery语法
- JQuery事件
- 学习网址

JQuery简介

JQuery是目前使用最广泛的JavaScript函数库，能够简化 DOM 操作，AJAX 调用和 Event 处理。据统计，全世界57.5%的网站使用jQuery，在使用JavaScript函数库的网站中，93.0%使用jQuery。它已经成了开发者必须学会的技能。

JQuery的最大优势有两个。首先，它基本是一个DOM操作工具，可以使操作DOM对象变得异常容易。其次，它统一了不同浏览器的API接口，使得代码在所有现代浏览器均能运行，开发者不用担心浏览器之间的差异。

JQuery语法

jQuery 语法是通过选取 HTML 元素，并对选取的元素执行某些操作。

基础语法： $(selector).action()

•美元符号定义 JQuery。

•选择符（selector）查找 HTML 元素

•jQuery 的 action() 执行对元素的操作

实例：

•$(this).hide() - 隐藏当前元素

•$(“p”).hide() - 隐藏所有 <p> 元素

•$(“p.test”).hide() - 隐藏所有 class=“test” 的 <p> 元素

•$("#test").hide() - 隐藏所有 id="test" 的元素

JQuery语法

所有 jQuery 函数位于一个 document ready 函数中，这是为了防止文档在完全加载（就绪）之前运行 jQuery 代码，即在 DOM 加载完成后才可以对 DOM 进行操作。

如果在文档没有完全加载之前就运行函数，操作可能失败。下面是两个具体的例子：

•试图隐藏一个不存在的元素。

•获得未完全加载的图像的大小。

```
$(document).ready(function(){

    // 开始写 jQuery 代码...

});
```

JQuery事件

页面对不同访问者的响应叫做事件。

事件处理程序指的是当 HTML 中发生某些事件时所调用的方法。

实例：

- 在元素上移动鼠标。
- 选取单选按钮。
- 点击元素。

常见 DOM 事件：

鼠标事件	键盘事件	表单事件	文档/窗口事件
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave	hover	blur	unload

JQuery事件

$(document).ready()：

当鼠标指针移动到元素上方，并按下鼠标按键时，会发生 mousedown 事件。

click()

click() 方法是当按钮点击事件被触发时会调用一个函数。

mousedown()

$(document).ready() 方法允许我们在文档完全加载完后执行函数。

mouseup()

当在元素上松开鼠标按钮时，会发生 mouseup 事件。

JQuery事件

hover()

hover()方法用于模拟光标悬停事件。

focus()

当元素获得焦点时，发生 focus 事件。

当通过鼠标点击选中元素或通过 tab 键定位到元素时，该元素就会获得焦点。

blur()

当元素失去焦点时，发生 blur 事件。

JQuery事件

事件绑定

$().事件名(事件处理函数)　　如： $(“#d”).click(function(){})

$().on(“事件名”,事件处理函数)　　如：$(“#d”).on(“click”,function(){})

$().bind(“事件名”,事件处理函数)　　如：$("#d").bind("click",function(){})

解除事件绑定

$().unbind("事件名")

学习网址

- http://www.w3school.com.cn/jquery/jquery_intro.asp
- https://developer.mozilla.org/zh-CN/docs/Glossary/jQuery
- https://javascript.ruanyifeng.com/jquery/basic.html
- http://www.runoob.com/jquery/jquery-tutorial.html

AJAX

- AJAX简介
- XMLHttpRequest对象
- XMLHttpRequest的实例方法
- XMLHttpRequest实例的事件
- AJAX代码示例
- 学习网址

AJAX简介

浏览器与服务器之间，采用 HTTP 协议通信。用户在浏览器地址栏键入一个网址，或者通过网页表单向服务器提交内容，这时浏览器就会向服务器发出 HTTP 请求。

AJAX 是 Asynchronous JavaScript and XML 的缩写，指的是通过 JavaScript 的异步通信，从服务器获取 XML 文档，从中提取数据，再更新当前网页的对应部分，而不用刷新整个网页。后来，AJAX 这个词就成为 JavaScript 脚本发起 HTTP 通信的代名词，也就是说，只要用脚本发起通信，就可以叫做 AJAX 通信。

AJAX简介

具体来说，AJAX 包括以下几个步骤。

1. 创建 XMLHttpRequest 实例。
2. 发出 HTTP 请求。
3. 接收服务器传回的数据。
4. 更新网页数据

概括起来，就是一句话，AJAX 通过原生的XMLHttpRequest对象发出 HTTP 请求，得到服务器返回的数据后，再进行处理。现在，服务器返回的都是 JSON 格式的数据，XML 格式已经过时了，但是 AJAX 这个名字已经成了一个通用名词，字面含义已经消失了。

XMLHttpRequest对象

XMLHttpRequest对象是 AJAX 的主要接口，用于浏览器与服务器之间的通信。尽管名字里面有XML和Http，它实际上可以使用多种协议（比如file或ftp），发送任何格式的数据（包括字符串和二进制）。

XMLHttpRequest本身是一个构造函数，可以使用new命令生成实例。它没有任何参数。

```
var xhr = new XMLHttpRequest();
```

一旦新建实例，就可以使用open()方法发出 HTTP 请求。

```
xhr.open('GET', 'http://www.example.com/page.php', true);
```

上面代码向指定的服务器网址，发出 GET 请求。

XMLHttpRequest对象

然后，指定回调函数，监听通信状态（readyState属性）的变化。

```
ajax.onreadystatechange = handleStateChange;

function handleStateChange() {
  // ...
}
```

上面代码中，一旦XMLHttpRequest实例的状态发生变化，就会调用监听函数handleStateChange。一旦拿到服务器返回的数据，AJAX 不会刷新整个网页，而是只更新网页里面的相关部分，从而不打断用户正在做的事情。注意，AJAX 只能向同源网址（协议、域名、端口都相同）发出 HTTP 请求，如果发出跨域请求，就会报错。

XMLHttpRequest对象

下面是XMLHttpRequest对象简单用法的完整例子。

```
var xhr = new XMLHttpRequest();

xhr.onreadystatechange = function(){
  // 通信成功时，状态值为4
  if (xhr.readyState === 4){
    if (xhr.status === 200){
      console.log(xhr.responseText);
    } else {
      console.error(xhr.statusText);
    }
  }
};

xhr.onerror = function (e) {
  console.error(xhr.statusText);
};

xhr.open('GET', '/endpoint', true);
xhr.send(null);
```

XMLHttpRequest对象

XMLHttpRequest.readyState返回一个整数，表示实例对象的当前状态。该属性只读。它可能返回以下值。

- 0，表示 XMLHttpRequest 实例已经生成，但是实例的open()方法还没有被调用。
- 1，表示open()方法已经调用，但是实例的send()方法还没有调用，仍然可以使用实例的setRequestHeader()方法，设定 HTTP 请求的头信息。
- 2，表示实例的send()方法已经调用，并且服务器返回的头信息和状态码已经收到。
- 3，表示正在接收服务器传来的数据体（body 部分）。这时，如果实例的responseType属性等于text或者空字符串，responseText属性就会包含已经收到的部分信息。
- 4，表示服务器返回的数据已经完全接收，或者本次接收已经失败。

XMLHttpRequest对象

XMLHttpRequest.responseType属性是一个字符串，表示服务器返回数据的类型。这个属性是可写的，可以在调用open()方法之后、调用send()方法之前，设置这个属性的值，告诉服务器返回指定类型的数据。

XMLHttpRequest.responseType属性可以等于以下值。

- “”（空字符串）：等同于text，表示服务器返回文本数据。
- “arraybuffer”：ArrayBuffer 对象，表示服务器返回二进制数组。
- “blob”：Blob 对象，表示服务器返回二进制对象。
- “document”：Document 对象，表示服务器返回一个文档对象。
- “json”：JSON 对象。
- “text”：字符串。

XMLHttpRequest对象

XMLHttpRequest.status属性返回一个整数，表示服务器回应的 HTTP 状态码。一般来说，如果通信成功的话，这个状态码是200；如果服务器没有返回状态码，那么这个属性默认是200。请求发出之前，该属性为0。

- 200, OK，访问正常
- 301, Moved Permanently，永久移动
- 302, Move temporarily，暂时移动
- 304, Not Modified，未修改
- 307, Temporary Redirect，暂时重定向
- 401, Unauthorized，未授权
- 403, Forbidden，禁止访问
- 404, Not Found，未发现指定网址
- 500, Internal Server Error，服务器发生错误

基本上，只有2xx和304的状态码，表示服务器返回是正常状态。

XMLHttpRequest的实例方法

XMLHttpRequest.open()方法用于指定 HTTP 请求的参数，或者说初始化 XMLHttpRequest 实例对象。它一共可以接受五个参数。

- method：表示 HTTP 动词方法，比如GET、POST、PUT、DELETE、HEAD等
- url：表示请求发送目标 URL。
- async：布尔值，表示请求是否为异步，默认为true。如果设为false，则send()方法只有等到收到服务器返回了结果，才会进行下一步操作。该参数可选。由于同步 AJAX 请求会造成浏览器失去响应，许多浏览器已经禁止在主线程使用，只允许 Worker 里面使用。所以，这个参数轻易不应该设为false。
- user：表示用于认证的用户名，默认为空字符串。该参数可选。
- password：表示用于认证的密码，默认为空字符串。该参数可选。

下面是发送 POST 请求的例子。

```
var xhr = new XMLHttpRequest();
xhr.open('POST', encodeURI('someURL'));
```

XMLHttpRequest的实例方法

XMLHttpRequest.send()方法用于实际发出 HTTP 请求。它的参数是可选的，如果不带参数，就表示 HTTP 请求只包含头信息，也就是只有一个 URL，典型例子就是 GET 请求；如果带有参数，就表示除了头信息，还带有包含具体数据的信息体，典型例子就是 POST 请求。

下面是发送 GET 请求的例子。

```
var xhr = new XMLHttpRequest();
xhr.open('GET',
  'http://www.example.com/?id=' + encodeURIComponent(id),
  true
);
xhr.send(null);

// 等同于
var data = 'id=' + encodeURIComponent(id);
xhr.open('GET', 'http://www.example.com', true);
xhr.send(data);
```

XMLHttpRequest实例的事件

readyStateChange 事件

readyState属性的值发生改变，就会触发 readyStateChange 事件。

我们可以通过onReadyStateChange属性，指定这个事件的监听函数，对不同状态进行不同处理。尤其是当状态变为 4 的时候，表示通信成功，这时回调函数就可以处理服务器传送回来的数据。

```
xmlhttp.onreadystatechange = state_Change;

function state_Change() {
    if (xmlhttp.readyState==4){
    // 4 = "loaded"
        if (xmlhttp.status==200){
        // 200 = "OK"
        }
    }
}
```

XMLHttpRequest实例的事件

progress 事件

上传文件时，XMLHTTPRequest 实例对象本身和实例的upload属性，都有一个progress事件，会不断返回上传的进度。

```
var xhr = new XMLHttpRequest();

function updateProgress (oEvent) {
  if (oEvent.lengthComputable) {
    var percentComplete = oEvent.loaded / oEvent.total;
  } else {
    console.log('无法计算进展');
  }
}

xhr.addEventListener('progress', updateProgress);

xhr.open();
```

XMLHttpRequest实例的事件

load 事件、error 事件、abort 事件

load 事件表示服务器传来的数据接收完毕，error 事件表示请求出错，abort 事件表示请求被中断（比如用户取消请求）。

```
var xhr = new XMLHttpRequest();

xhr.addEventListener('load', transferComplete);
xhr.addEventListener('error', transferFailed);
xhr.addEventListener('abort', transferCanceled);

xhr.open();

function transferComplete() {
  console.log('数据接收完毕');
}

function transferFailed() {
  console.log('数据接收出错');
}

function transferCanceled() {
  console.log('用户取消接收');
}
```

AJAX代码示例：JavaScript

```
var xmlhttp;
function loadURLDoc(url) {
  xmlhttp=null;
  if (window.XMLHttpRequest) {// code for Firefox, Mozilla, IE7, etc.
    xmlhttp=new XMLHttpRequest();
  }
  else if (window.ActiveXObject) {// code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  if (xmlhttp!=null) {
    xmlhttp.onreadystatechange=state_Change;
    xmlhttp.open("GET",url,true);
    xmlhttp.send(null);
  }
  else {
    alert("Your browser does not support XMLHTTP.");
  }
}

function state_Change() {
  if (xmlhttp.readyState==4 && xmlhttp.status==200) {// 4 = "loaded" and 200 = "OK"
    // doSomething
  } else {
    alert("Problem retrieving data:" + xmlhttp.statusText);
  }
}
```

AJAX代码示例：JQuery

GET请求

```
$.get("demo_test.asp",function(data,status){
  alert("Data: " + data + "\n Status: " + status);
});
```

POST请求

```
$.post("demo_test_post.asp",
{
  name:"Donald Duck",
  city:"Duckburg"
},
function(data,status){
  alert("Data: " + data + "\nStatus: " + status);
});
```

学习网址

- http://www.w3school.com.cn/jquery/jquery_ajax_intro.asp
- https://developer.mozilla.org/zh-CN/docs/Web/Guide/AJAX
- http://javascript.ruanyifeng.com/bom/ajax.html
- http://www.w3school.com.cn/ajax/index.asp

REACT

- REACT简介
- REACT特点
- JSX简介
- REACT Hello World
- REACT元素渲染
- REACT事件处理
- REACT条件渲染
- 学习网址

REACT简介

React 是一个声明式，高效且灵活的用于构建用户界面的 JavaScript 库。使用 React 可以将一些简短、独立的代码片段组合成复杂的 UI 界面，这些代码片段被称作“组件”。

REACT起源于 Facebook 的内部项目，用来架设Instagram 的网站，并在2013年5月开源。由于 React的设计思想极其独特，属于革命性创新，性能出众，代码逻辑却非常简单。所以，越来越多的人开始关注和使用，认为它可能是将来 Web 开发的主流工具。你可以在React里传递多种类型的参数，如声明代码，帮助你渲染出UI、也可以是静态的HTML DOM元素、也可以传递动态变量、甚至是可交互的应用组件

REACT特点

- 声明式设计 -React采用声明范式，可以轻松描述应用。
- 高效 -React通过对DOM的模拟，最大限度地减少与DOM的交互。
- 灵活 -React可以与已知的库或框架很好地配合。
- JSX - JSX 是 JavaScript 语法的扩展。React 开发不一定使用 JSX ，但我们建议使用它。
- 组件 - 通过 React 构建组件，使得代码更加容易得到复用，能够很好的应用在大项目的开发中。
- 单向响应的数据流 - React 实现了单向响应的数据流，从而减少了重复代码，这也是它为什么比传统数据绑定更简单。

JSX简介

```
const element = <h1>Hello, world!</h1>;
```

- 这个标签语法既不是字符串也不是 HTML。
- 它被称为 JSX，是一个 JavaScript 的语法扩展。建议在 React 中配合使用 JSX，JSX 可以很好地描述 UI 应该呈现出它应有交互的本质形式。JSX 可能会使人联想到模版语言，但它具有 JavaScript 的全部功能。JSX 可以生成 React “元素”。

JSX简介

在下面的例子中，声明了一个名为 name 的变量，然后在 JSX 中使用它，并将它包裹在大括号中：

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

在 JSX 语法中，你可以在大括号内放置任何有效的 JavaScript 表达式。例如 2 + 2，user.firstName 或 formatName(user) 都是有效的 JavaScript 表达式。

JSX简介

在下面的示例中，调用 了JavaScript 函数 formatName(user) 的结果，并将结果嵌入到 <h1> 元素中。

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

REACT Hello World

最简易的REACT示例如下：

```
ReactDOM.render(
  <h1>Hello, world!</h1>,
  document.getElementById('root')
);
```

它将在页面上展示一个“Hello,world!”的标题

REACT元素渲染

- 元素：
  - 元素是构成 React 应用的最小砖块。元素描述了你在屏幕上想看到的内容。
  - 与浏览器的 DOM 元素不同，React 元素是创建开销极小的普通对象。React DOM 会负责更新 DOM 来与 React 元素保持一致。

```
const element = <h1>Hello, world</h1>;
```

REACT元素渲染

- 将一个元素渲染为DOM：
  - 假设你的 HTML 文件某处有一个 ：

```
<div id="root"></div>
```

  - 我们将其称为“根” DOM 节点，因为该节点内的所有内容都将由 React DOM 管理
  - 仅使用 React 构建的应用通常只有单一的根 DOM 节点。如果你在将 React 集成进一个已有应用，那么你可以在应用中包含任意多的独立根 DOM 节点。
  - 想要将一个 React 元素渲染到根 DOM 节点中，只需把它们一起传入 ReactDOM.render()：

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

REACT元素渲染

- 更新已渲染的元素：
  - React 元素是不可变对象。一旦被创建，你就无法更改它的子元素或者属性。一个元素就像电影的单帧：它代表了某个特定时刻的 UI。更新 UI 唯一的方式是创建一个全新的元素，并将其传入 ReactDOM.render()。
  - 这个计时器例子会在 setInterval() 回调函数，每秒都调用 ReactDOM.render()

```
function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  ReactDOM.render(element, document.getElementById('root'));
}

setInterval(tick, 1000);
```

REACT组件

- 组件：组件允许你将 UI 拆分为独立可复用的代码片段，并对每个片段进行独立构思。组件，从概念上类似于 JavaScript 函数。它接受任意的入参（即 “props”），并返回用于描述页面展示内容的 React 元素。

- 注意： 组件名称必须以大写字母开头。
- React 会将以小写字母开头的组件视为原生 DOM 标签。
- 例如， 代表 HTML 的 div 标签，而 <Welcome /> 则代表一个组件，并且需在作用域内使用 Welcome。

REACT组件

- 函数组件：定义组件最简单的方式就是编写 JavaScript 函数：

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

- class组件：还可以使用 ES6 的 class 来定义组件：

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

- 上述两个组件在REACT中等效。

REACT组件

- 将函数组件转换成class组件：
  1. 创建一个同名的 ES6 class，并且继承于 React.Component。
  2. 添加一个空的 render() 方法。
  3. 将函数体移动到 render() 方法之中。
  4. 在 render() 方法中使用 this.props 替换 props。
  5. 删除剩余的空函数声明。

```
class Clock extends React.Component {
  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.props.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

REACT组件

- 渲染组件：
- React 元素也可以是用户自定义的组件，当 React 元素为用户自定义组件时，它会将 JSX 所接收的属性（attributes）以及子组件（children）转换为单个对象传递给组件，这个对象被称之为 “props”。
- 例如，这段代码会在页面上渲染 “Hello, Sara”：

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

REACT事件处理

- React 元素的事件处理和 DOM 元素的很相似，但是有一点语法上的不同：
  - React 事件的命名采用小驼峰式（camelCase），而不是纯小写。
  - 使用 JSX 语法时你需要传入一个函数作为事件处理函数，而不是一个字符串。

- 例如，传统的 HTML：

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```

- 在REACT中略微不同：

```
<button onClick={activateLasers}>
  Activate Lasers
</button>
```

REACT事件处理

- 在 React 中另一个不同点是你不能通过返回 false 的方式阻止默认行为。你必须显式的使用 preventDefault 。

```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('The link was clicked.');
  }

  return (
    <a href="#" onClick={handleClick}>
      Click me
    </a>
  );
}
```

- 在这里，e 是一个合成事件。React 根据 W3C 规范来定义这些合成事件，所以你不需要担心跨浏览器的兼容性问题。

REACT事件处理

- 向事件处理程序传递参数：
- 在循环中，通常我们会为事件处理函数传递额外的参数。例如，若 id 是你要删除那一行的 ID，以下两种方式都可以向事件处理函数传递参数：

```
<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>
<button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>
```

- 上述两种方式是等价的，分别通过箭头函数和 Function.prototype.bind 来实现
- 在这两种情况下，React 的事件对象 e 会被作为第二个参数传递。如果通过箭头函数的方式，事件对象必须显式的进行传递，而通过 bind 的方式，事件对象以及更多的参数将会被隐式的进行传递。

REACT条件渲染

- React 中的条件渲染和 JavaScript 中的一样，使用 JavaScript 运算符 if 或者条件运算符去创建元素来表现当前的状态，然后让 React 根据它们来更新 UI。

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
```

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}

ReactDOM.render(
  // Try changing to isLoggedIn={true}:
  <Greeting isLoggedIn={false} />,
  document.getElementById('root')
);
```

REACT列表

- 渲染多个组件：可以通过使用 {} 在 JSX 内构建一个元素集合。
- 下面，我们使用 Javascript 中的 map() 方法来遍历 numbers 数组。将数组中的每个元素变成 <li> 标签，最后我们将得到的数组赋值给 listItems：

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li>{number}</li>
);
```

- 我们把整个 listItems 插入到 <ul> 元素中，然后渲染进 DOM：

```
ReactDOM.render(
  <ul>{listItems}</ul>,
  document.getElementById('root')
);
```

REACT列表

- 基础列表组件：通常需要在一个组件中渲染列表。
- 我们可以把前面的例子重构成一个组件，这个组件接收 numbers 数组作为参数并输出一个元素列表。

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

REACT列表

- key：key 帮助 React 识别哪些元素改变了，比如被添加或删除。因此我们应当给数组中的每一个元素赋予一个确定的标识。
- 一个元素的 key 最好是这个元素在列表中拥有的一个独一无二的字符串。通常，我们使用数据中的 id 来作为元素的 key。当元素没有确定 id 的时候，万不得已可以使用元素索引 index 作为 key。
- 如果列表项目的顺序可能会变化，我们不建议使用索引来用作 key 值，因为这样做会导致性能变差，还可能引起组件状态的问题。如果你选择不指定显式的 key 值，那么 React 将默认使用索引用作为列表项目的 key 值。

REACT列表

- 用key提取组件：

```
function ListItem(props) {
  // 正确！这里不需要指定 key:
  return <li>{props.value}</li>;
}

function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    // 正确！key 应该在数组的上下文中被指定
    <ListItem key={number.toString()}                value={number} />

  );
  return (
    <ul>
      {listItems}
    </ul>
  );
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

学习网址

- https://reactjs.org/
- https://react.docschina.org/
- http://caibaojian.com/react/
- http://react-china.org/
- https://www.runoob.com/react/react-tutorial.html

Vue

- Vue简介
- Vue特点
- Vue数据绑定
- Vue Virtual DOM
- Vue组件
- Vue Hello World
- Vue模板语法
- Vue事件处理
- 学习网址

Vue简介

vue是一套构建用户界面的渐进式框架。

只关注视图层，采用自底向上增量开发的设计。通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。它使用了基于 HTML 的模板语法，允许开发者声明式地将 DOM 绑定至底层 Vue 实例的数据。所有 Vue.js 的模板都是合法的 HTML ，所以能被遵循规范的浏览器和 HTML 解析器解析。在底层的实现上，Vue 将模板编译成虚拟 DOM 渲染函数。结合响应系统，Vue 能够智能地计算出最少需要重新渲染多少组件，并把 DOM 操作次数减到最少。

Vue特点

1）轻量级的框架

Vue.js 能够自动追踪依赖的模板表达式和计算属性，提供 MVVM 数据绑定和一个可组合的组件系统，具有简单、灵活的 API，使读者更加容易理解，能够更快上手。

2）双向数据绑定

声明式渲染是数据双向绑定的主要体现，同样也是 Vue.js 的核心，它允许采用简洁的模板语法将数据声明式渲染整合进 DOM。

3）指令

Vue.js 与页面进行交互，主要就是通过内置指令来完成的，指令的作用是当其表达式的值改变时相应地将某些行为应用到 DOM 上。

Vue特点

4）组件化

组件（Component）是 Vue.js 最强大的功能之一。组件可以扩展 HTML 元素，封装可重用的代码。

在 Vue 中，父子组件通过 props 传递通信，从父向子单向传递。子组件与父组件通信，通过触发事件通知父组件改变数据。这样就形成了一个基本的父子通信模式。

在开发中组件和 HTML、JavaScript 等有非常紧密的关系时，可以根据实际的需要自定义组件，使开发变得更加便利，可大量减少代码编写量。

组件还支持热重载（hotreload）。当我们做了修改时，不会刷新页面，只是对组件本身进行立刻重载，不会影响整个应用当前的状态。CSS 也支持热重载。

Vue特点

5）客户端路由

Vue-router 是 Vue.js 官方的路由插件，与 Vue.js 深度集成，用于构建单页面应用。Vue 单页面应用是基于路由和组件的，路由用于设定访问路径，并将路径和组件映射起来，传统的页面是通过超链接实现页面的切换和跳转的。

6）状态管理

状态管理实际就是一个单向的数据流，State 驱动 View 的渲染，而用户对 View 进行操作产生 Action，使 State 产生变化，从而使 View 重新渲染，形成一个单独的组件。

Vue特点

Vue优势：

1. 轻量化
2. 低复杂性
3. Virtual DOM
4. 低学习曲线

缺点：

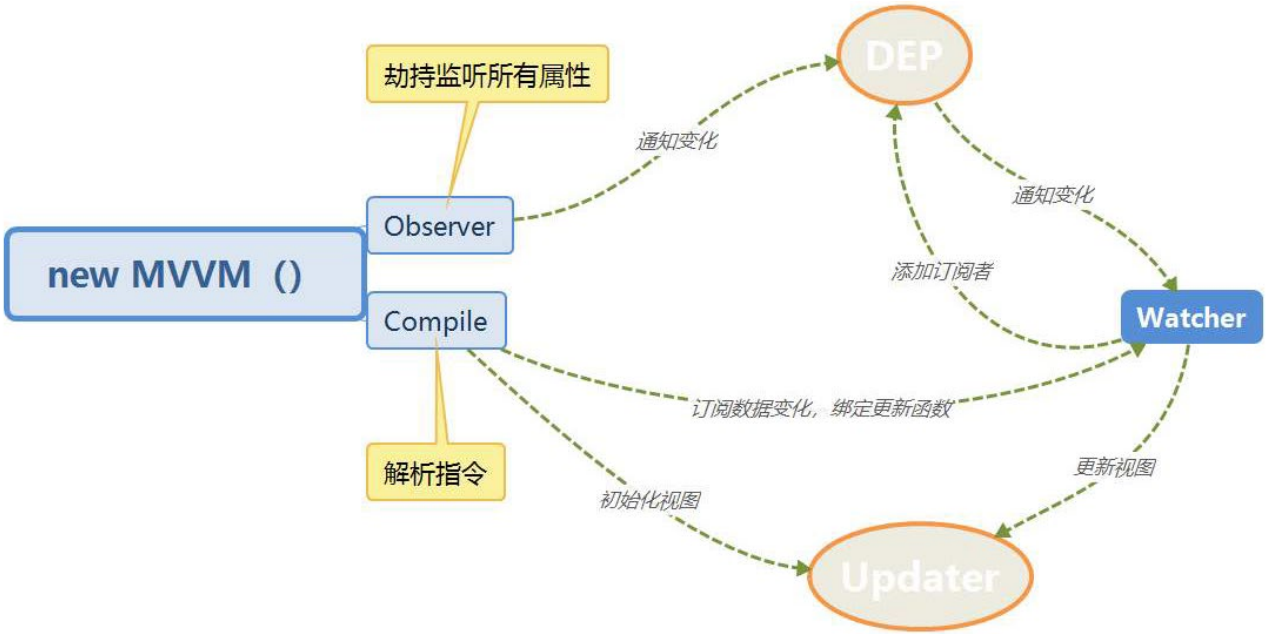
1. 相比React、Angular发展时间短，社区规模小
2. 目前使用Vue公司的不多

Name	Size
Ember 2.2.0	435K
Ember 1.13.8	486K
Angular 2	566K
Angular 2 + Rx	**766K**
Angular 1.4.5	143K
Vue 2.4.2	58.8K
Inferno 1.2.2	48K
Preact 7.2.0	**16K**
React 0.14.5 + React DOM	**133K**
React 0.14.5 + React DOM + Redux	139K
React 16.2.0 + React DOM	97.5K

不同前端框架/库的下载空间

Vue数据绑定

目前主流的数据绑定方式有三种，分别是发布者-订阅者模式、脏值检查、数据劫持。vue.js 则是采用数据劫持结合发布者-订阅者模式的方式，通过Object.defineProperty()来劫持各个属性的setter，getter，在数据变动时发布消息给订阅者，触发相应的监听回调。



Vue数据绑定

Vue数据绑定

vue实现双向数据绑定，它会自动响应数据的变化情况，并且根据用户在代码中预先写好的绑定关系，对所有绑定在一起的数据和视图内容都进行修改。这需要实现三个模块：Observer、Compile与Watcher。

Observer：能够对数据对象的所有属性进行监听，如有变动可拿到最新值并通知订阅者。

Compile：对每个元素节点的指令进行扫描和解析，根据指令模板替换数据，以及绑定相应的更新函数。

Watcher：作为连接Observer和Compile的桥梁，能够订阅并收到每个属性变动的通知，执行指令绑定的相应回调函数，从而更新视图。

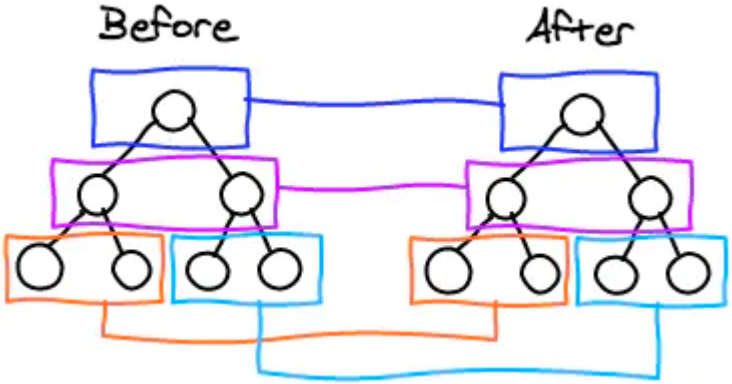
Vue数据绑定

具体来说vue会遍历此data中对象所有的属性，并使用Object.defineProperty把这些属性全部转为getter/setter，而每个组件实例都有watcher对象，它会在组件渲染的过程中把属性记录为依赖，之后当依赖项的 setter被调用时，会通知watcher重新计算，从而致使它关联的组件得以更新。

Vue Virtual DOM

Vue和React类似，也是使用了Virtual DOM

为了实现高效的DOM操作，一套高效的虚拟DOM diff算法显得很有必要。我们通过patch 的核心----diff 算法，找出本次DOM需要更新的节点来更新，其他的不更新。



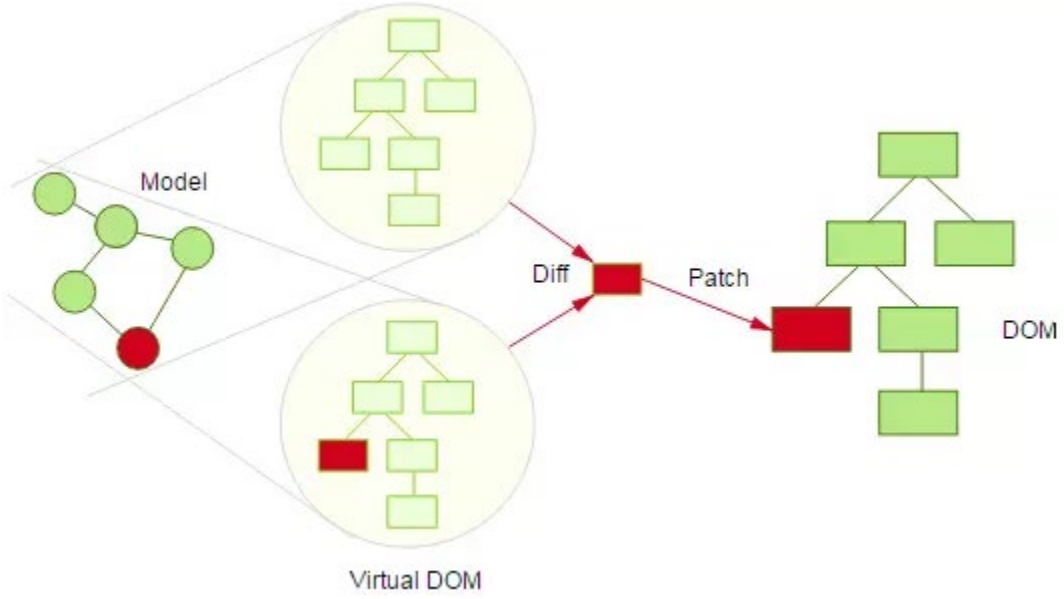
diff算法

仅在同级的vnode间做diff，递归地进行同级vnode的diff，最终实现整个DOM树的更新。因为跨层级的操作是非常少的，忽略不计，这样时间复杂度就是O(n)。

Vue Virtual DOM

diff 算法包括几个步骤：

1.用 JavaScript 对象结构表示 DOM 树的结构；然后用这个树构建一个真正的 DOM 树，插到文档当中。

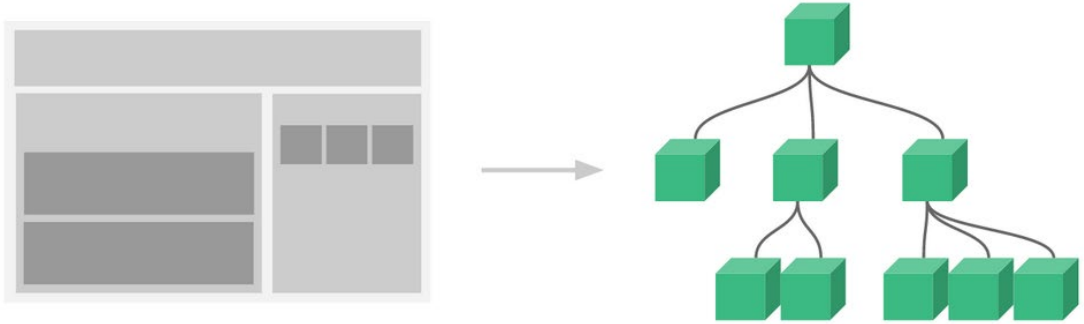


2.当状态变更的时候，重新构造一棵新的对象树。然后用新的树和旧的树进行比较，记录两棵树差异。

3.把所记录的差异应用到所构建的真正的DOM树上，视图就更新了。

Vue组件

组件（Component）是 Vue.js 最强大的功能之一。组件可以扩展 HTML 元素，封装可重用的代码。组件系统让我们可以用独立可复用的小组件来构建大型应用，几乎任意类型的应用的界面都可以抽象为一个组件树，如图所示，例如，可能会有页头、侧边栏、内容区等组件，每个组件又包含了其它的像导航链接、博文之类的组件。



组件树

Vue Hello World

一个简单的Hello World示例

```
<div id="app">
    {{ message }}
</div>
</body>
<script src="js/vue.js"></script>
<script>
    var exampleData = {
        message: 'Hello World!'
    }

    new Vue({
        el: '#app',
        data: exampleData
    })
</script>
```

Vue模板语法

Vue.js 使用了基于 HTML 的模版语法，允许开发者声明式地将 DOM 绑定至底层 Vue 实例的数据。

Vue.js 的核心是一个允许你采用简洁的模板语法来声明式的将数据渲染进 DOM 的系统。

结合响应系统，在应用状态改变时， Vue 能够智能地计算出重新渲染组件的最小代价并应用到 DOM 操作上。

Vue模板语法

文本：使用双大括号：

```
<div id="app">
    {{ message }}
</div>
```

Html：使用 v-html 指令用于输出 html 代码：

```
<div id="app">
    <div v-html="message"></div>
</div>

<script>
    new Vue({
        el: '#app',
        data: {
            message: '<h1>hello world</h1>'
        }
    })
</script>
```

Vue模板语法

属性：HTML 属性中的值应使用 v-bind 指令。

以下实例判断 use 的值，如果为 true 使用 class1 类的样式，否则不使用该类：

```
<div id="app">
    <label for="r1">修改颜色</label><input type="checkbox" v-model="use" id="r1">
    <br><br>
    <div v-bind:class="{'class1': use}">
        v-bind:class 指令
    </div>
</div>

<script>
    new Vue({
        el: '#app',
        data:{
            use: false
        }
    });
</script>
```

Vue模板语法

条件判断：条件判断使用 v-if 、v-else、v-else-if指令：

```
<div id="app">
    <div v-if="type === 'A'">A</div>
    <div v-else-if="type === 'B'">B</div>
    <div v-else-if="type === 'C'">C</div>
    <div v-else>Not A/B/C</div>
</div>
<script>
    new Vue({
        el: '#app',
        data: {
            type: 'C'
        }
    })
</script>
```

判断type变量值

Vue事件处理

可以用 v-on 指令监听 DOM 事件，并在触发时运行一些 JavaScript 代码。使用 v-on 有几个好处：

能轻松定位在 JavaScript 代码里对应的方法。因为无须在 JavaScript 里手动绑定事件，你的 ViewModel 代码可以是非常纯粹的逻辑，和 DOM 完全解耦，更易于测试。

```
<div id="app">
    <!-- `greet` 是在下面定义的方法名 -->
    <button v-on:click="greet">Greet</button>
</div>

<script>
    var app = new Vue({
        el: '#app',
        data: {
            name: 'Vue.js'
        },
        // 在 `methods` 对象中定义方法
        methods: {
            greet: function (event) {
                // `this` 在方法里指当前 Vue 实例
                alert('Hello ' + this.name + '!')
                // `event` 是原生 DOM 事件
                if (event) {
                    alert(event.target.tagName)
                }
            }
        }
    })
    // 也可以用 JavaScript 直接调用方法
    app.greet() // -> 'Hello Vue.js!'
</script>
```

学习网址

- https://www.runoob.com/vue2/vue-tutorial.html
- https://www.vue-js.com/
- https://vuejs.bootcss.com/guide/

Angular

- Angular简介
- TypeScript
- Angular Hello World
- Angular特点
- 学习网址

Angular简介

Angular是由Google维护的一款开源JavaScript，使用TypeScript编写。Anguar1.5叫做AngularJs，Angular4.0称为Angular，Angular1.5到Angular4.0是完全重写。Angular是一种单页应用，组件应用。

首先Angular是一个MVC框架，它与JQuery不同之处在于，前者致力于MVC代码解耦，采用Model,Controller以及View方式去组织代码，而后者提供给你了很多API函数，你可以不用写很多原生JS去实现比较复杂的效果。

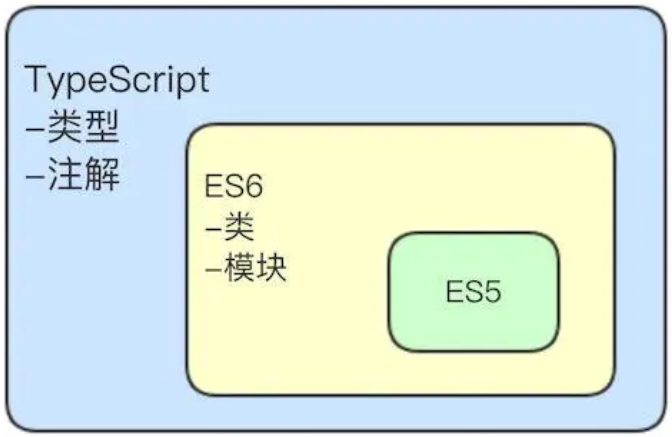
此外，Angular和Vue/React的一个主要不同点，Vue/React本质上是一个前端View层的Library，而Angular是一个大而全的框架。

TypeScript

TypeScript 是 JavaScript 的一个超集，支持 ECMAScript 6 标准。

TypeScript 由微软开发的自由和开源的编程语言。

TypeScript 设计目标是开发大型应用，它可以编译成纯 JavaScript，编译出来的 JavaScript 可以运行在任何浏览器上。



TypeScript相对ES5有许多改善，比如：类型、类、注解、模块导入等。

TypeScript

一、类型

相对于ES6，TypeScript最大的改善是增加了类型系统。增加类型检测一方面可以在编译期预防bug，有助于代码的编写；另一方面，它能清晰地表明你的意图，有助于代码阅读。

TypeScript的基本类型与平时所写的JavaScript代码中的隐式类型一样，包括字符串、数字、布尔值等。TypeScript语法从ES5演化而来，仍然延用var来定义变量，但其可以为变量提供类型。例如：

```
var name: string;

var isPass: boolean = true;
```

TypeScript

二、类

在ES6中有了“类”的概念，使用class关键字来定义一个类，紧随其后的是类名和类的代码块。类包含属性、方法和构造函数。TypeScript同样有类的概念。

```
class Person  {
    name: string;
    age: number;
    constructor(name: string, age: number) {……}
}
```

TypeScript

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app works!';
}
```

三、注解

注解为程序的元素(类、方法、变量)加上更直观更明了的说明,这些说明信息与程序的业务逻辑无关,而是供指定的工具或框架使用的。

我们看到AppComponent上面有个@Compoent注解，这个@Compoent注解本身是由Angular框架提供的，在这个注解里面有一些属性。这些属性告诉Angular怎么处理AppComponent这样一个TypeScript类，这个意思就是说在Angular框架里面我们实例话应该AppComponent类，Angular框架应该去加载app.component.html页面和app.component.css，然后展示在页面上

TypeScript

四、模块导入

TypeScript 模块的设计理念是可以更换的组织代码。

模块是在其自身的作用域里执行，并不是在全局作用域，这意味着定义在模块里面的变量、函数和类等在模块外部是不可见的，除非明确地使用 export 导出它们。类似地，我们必须通过 import 导入其他模块导出的变量、函数、类等。

两个模块之间的关系是通过在文件级别上使用 import 和 export 建立的。

Angular hello world

```
import { Component } from '@angular/core';
@Component({
    selector: 'my-app',
    template: `
    <h1>{{title}}</h1>
    `
})
export class AppComponent {
    title = 'hello world!';
}
```

一个简单的hello world示例

Angular特性

1.MVC。Model模型、View视图、Controller控制器，互联网的三层架构：视图负责客户端与用户交互，控制器负责中间业务逻辑的处理，互通了视图和数据，模型用于对数据的增删改查。

```
<div ng-controller="HelloAngular">

    <p>{{greeting.text}},Angular</p>

</div>
```

```
function HelloAngular($scope) {
    $scope.greeting = {
        text: 'Hello'
    };
}
```

View很好理解，ng-controller="HelloAngular"指定了Controller，{{greeting.text}}指定了Model，在js代码中可以了解Controller是如何控制Model的。$scope指明了作用域，这里的作用域是div。

Angular特性

2.模块化设计（体现 “高聚合低耦合” 设计原则）

模块由一块代码组成，可用于执行一个简单的任务。

Angular 应用是由模块化的，它有自己的模块系统：NgModules。

每个 Angular 应该至少要有一个模块(根模块)，一般可以命名为：AppModule。

Angular 模块是一个带有 @NgModule 装饰器的类，它接收一个用来描述模块属性的元数据对象。

Angular特性

几个重要的属性如下：

declarations （声明） - 视图类属于这个模块。 Angular 有三种类型的视图类： 组件 、 指令 和 管道 。

exports - 声明（declaration）的子集，可用于其它模块中的组件模板 。

imports - 本模块组件模板中需要由其它导出类的模块。

providers - 服务的创建者。本模块把它们加入全局的服务表中，让它们在应用中的任何部分都可被访问到。

bootstrap - 应用的主视图，称为根组件，它是所有其它应用视图的宿主。只有根模块需要设置 bootstrap 属性中。

Angular特性

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
    imports:      [ BrowserModule ],
    providers:    [ Logger ],
    declarations: [ AppComponent ],
    exports:      [ AppComponent ],
    bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

一个简单的根模块

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app.module';

platformBrowserDynamic().bootstrapModule(AppModule);
```

通过引导根模块来启动应用

Angular特性

3.数据双向绑定

Angular中的双向数据绑定中数据到界面的呈现是使用的脏数据检查方法。

什么是脏数据检查（Dirty checking）：原理就是，Angular 在 scope 模型上设置了一个 监听队列，用来监听数据变化并更新 view 。每次绑定一个东西到 view 上时 AngularJS 就会往 $watch 队列里插入一条 $watch，用来检测它监视的 model 里是否有变化的东西。当浏览器接收到可以被 angular context 处理的事件时，$digest 循环就会触发，遍历所有的 $watch，最后更新 dom。

Angular特点

4.指令

Angular模板是动态的 。当 Angular 渲染它们时，它会根据指令对 DOM 进行修改。

指令是一个带有"指令元数据"的类。在 TypeScript 中，要通过@Directive 装饰器把元数据附加到类上。

在Angular中包含以下三种类型的指令：

属性指令：以元素的属性形式来使用的指令。

结构指令：用来改变DOM树的结构

组件：作为指令的一个重要子类，组件本质上可以看作是一个带有模板的指令。

Angular特性

*ngFor 告诉 Angular 为 sites 列表中的每个项生成一个 <li> 标签。

*ngIf 表示只有在选择的项存在时，才会包含 SiteDetail 组件。

```
<li *ngFor="let site of sites"></li>
<site-detail *ngIf="selectedSite"></site-detail>
```

Angular特性

5.服务

Angular2中的服务是封装了某一特定功能，并且可以通过注入的方式供他人使用的独立模块。

服务分为很多种，包括：值、函数，以及应用所需的特性。

例如，多个组件中出现了重复代码时，把重复代码提取到服务中实现代码复用。

以下是几种常见的服务：

日志服务、数据服务、消息总线、税款计算器、应用程序配置

Angular特性

以下实例是一个日志服务，用于把日志记录到浏览器的控制台：

```
export class Logger {
    log(msg: any)   { console.log(msg); }
    error(msg: any) { console.error(msg); }
    warn(msg: any)  { console.warn(msg); }
}
```

Angular特性

6.依赖注入

控制反转（Inversion of Control，缩写为IoC），是面向对象编程中的一种设计原则，可以用来减低计算机代码之间的耦合度。其中最常见的方式叫做依赖注入（Dependency Injection，简称DI），还有一种方式叫"依赖查找"（Dependency Lookup）。

通过控制反转，对象在被创建的时候，由一个调控系统内所有对象的外界实体，将其所依赖的对象的引用传递给它。也可以说，依赖被注入到对象中。

在传统的开发模式中，调用者负责管理所有对象的依赖，循环依赖一直是梦魇，而在依赖注入模式中，这个管理权交给了注入器(Injector)，它在软件运行时负责依赖对象的替换，而不是在编译时。这种控制反转，运行注入的特点即是依赖注入的精华所在。

Angular特性

Angular 能通过查看构造函数的参数类型，来得知组件需要哪些服务。 例如，SiteListComponent 组件的构造函数需要一个 SiteService:

```
constructor(private service: SiteService) { }
```

当 Angular 创建组件时，会首先为组件所需的服务找一个注入器（ Injector ）。

注入器是一个维护服务实例的容器，存放着以前创建的实例。

如果容器中还没有所请求的服务实例，注入器就会创建一个服务实例，并且添加到容器中，然后把这个服务返回给 Angular 。

当所有的服务都被解析完并返回时， Angular 会以这些服务为参数去调用组件的构造函数。 这就是依赖注入 。

Angular学习网址

- https://www.runoob.com/angularjs2/angularjs2-tutorial.html
- https://www.runoob.com/angularjs/angularjs-tutorial.html
- http://docs.angularjs.org
- http://www.angularjs.net.cn/tutorial/1.html
- https://github.com/aztack/AngularJS-translation
- https://github.com/dolymood/learning-angular
- http://www.angularjs.cn/