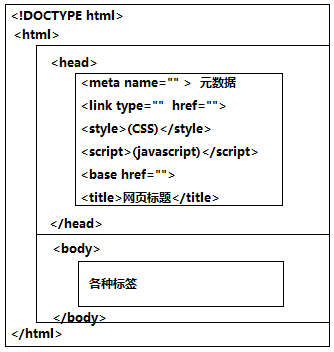
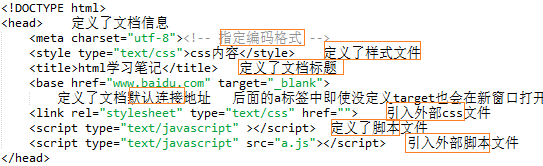
# 前端与移动 笔记

## HTML

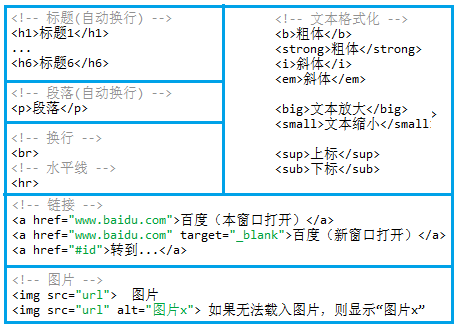
### 第一章 html 文档结构



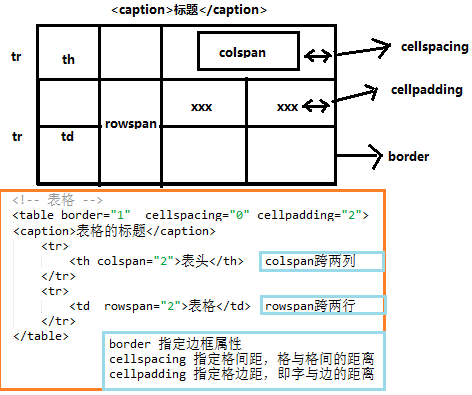
### 第二章 head 头部



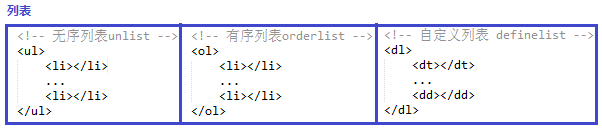
### 第三章 常用标签



### 第四章 表格



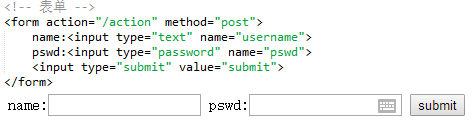
### 第五章 列表

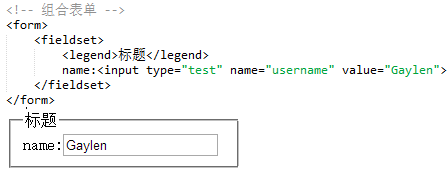


### 第六章 容器



### 第七章 表单



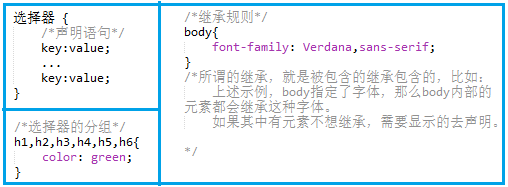




## CSS

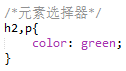
### 第一章 CSS基础

#### 1.1 CSS基础语法

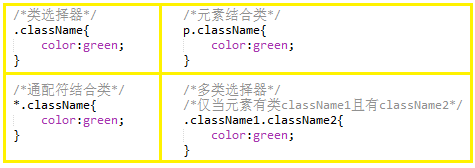


### 第二章 CSS 选择器

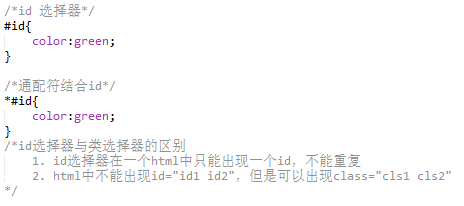
#### 2.1 元素选择器



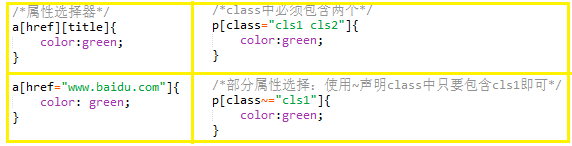
#### 2.2 类选择器

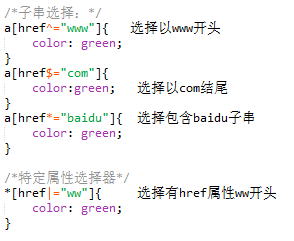


#### 2.3 id选择器

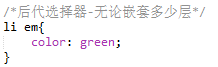


#### 2.4 属性选择器

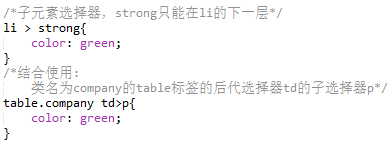




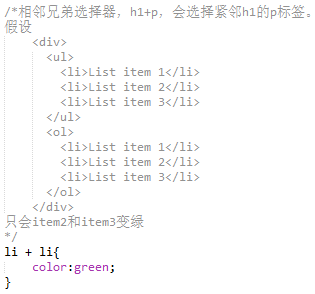
#### 2.5 后代选择器



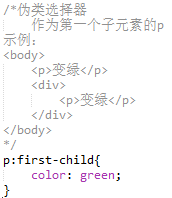
#### 2.6 子元素选择器



#### 2.7 相邻兄弟选择器



#### 2.8 伪类选择器





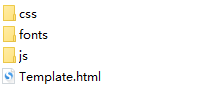
### 第三章 CSS 高级

## Bootstrap

### BootStrap的环境安装

#### 本地开发模板

本地开发模板，需要下载Bootstrap，文件夹如下：



其中，需要下载Jquery放在js中，模板代码如下：

<!DOCTYPE html>

<html lang="zh-CN">

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<!-- 为了让 Bootstrap 开发的网站对移动设备友好，确保适当的绘制和触屏缩放

以移动设备为优先，并禁止用户缩放

-->

<meta name="viewport" content="width=device-width, initial-scale=1 user-scalable=no">

<!-- 上述3个meta标签\*必须\*放在最前面，任何其他内容都\*必须\*跟随其后！ -->

<title>Bootstrap 101 Template</title>

<!-- Bootstrap -->

<link href="css/bootstrap.min.css" rel="stylesheet">

<!-- 保证老版本浏览器能识别HTML5的标签 -->

<script

src="https://cdn.bootcss.com/html5shiv/3.7.3/html5shiv.min.js">

</script>

<!-- 解决老版本的浏览器不支持CSS3 媒体查询 的问题 -->

<script

src="https://cdn.bootcss.com/respond.js/1.4.2/respond.min.js">

</script>

<!-- jQuery 必须在bootstrap.minjs 前面 -->

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

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

</head>

<body>

<h1>你好，世界！</h1>

</body>

</html>

#### 使用远程Bootstrap

<http://www.bootcdn.cn/> 稳定、快速、免费的前端开源项目 CDN 加速服务

使用远程文件的模板：

<!DOCTYPE html>

<html lang="zh-CN">

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<!-- 为了让 Bootstrap 开发的网站对移动设备友好，确保适当的绘制和触屏缩放

以移动设备为优先，但是不紧张用户缩放

-->

<meta name="viewport" content="width=device-width, initial-scale=1">

<!-- 上述3个meta标签\*必须\*放在最前面，任何其他内容都\*必须\*跟随其后！ -->

<title>Bootstrap 101 Template</title>

<!-- 最新版本的 Bootstrap 核心 CSS 文件 -->

<link rel="stylesheet" href="https://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">

<!-- 可选的 Bootstrap 主题文件（一般不用引入） -->

<link rel="stylesheet" href="https://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap-theme.min.css" integrity="sha384-rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp" crossorigin="anonymous">

<!-- 引入jQuery文件 -->

<script src="https://cdn.bootcss.com/jquery/3.2.1/jquery.min.js"></script>

<!-- 最新的 Bootstrap 核心 JavaScript 文件 -->

<script src="https://cdn.bootcss.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa" crossorigin="anonymous"></script>

</head>

<body>

<h1>你好，世界！</h1>

</body>

</html>

注意：以移动设备为优先的三个参数

① 宽度和移动设备宽度一致

② 初始缩放保持100%

③ 是否禁止用户缩放

### Bootstrap的CSS

#### 2.0 全局CSS样式的网址 <http://v3.bootcss.com/css/>

#### 2.1 Bootstrap的布局

<body>

<!-- 固定宽度为1170px -->

<div class="container" style="background: gray">

<h1>你好，世界</h1>

</div>

<!-- 如果希望宽度为100%，则类名为 container-fluid -->

</body>

总之，在Bootstrap中，很多组件已经定义好，只需要指定类名，即可使用。

#### 2.2 基本的排版元素

<div class="container" style="background: gray">

<h1>你好，世界,36px</h1>

<h2>30px</h2>

<h3>24px</h3>

<h4>18px</h4>

<h5>14px</h5>

<h6 class="page-header">12px</h6>

<!-- 其中page-header类，设置页头，并加分割线 -->

<small>本行内容是在标签内</small><br>

<strong>本行内容是在标签内,Bootstrap推荐使用的</strong><br>

<em>本行内容是在标签内，并呈现为斜体，推荐使用</em><br>

<p class="text-muted">本行内容是减弱的</p>

<p class="text-primary">本行内容带有一个 primary class</p>

<p class="text-success">本行内容带有一个 success class</p>

<p class="text-info">本行内容带有一个 info class</p>

<p class="text-warning">本行内容带有一个 warning class</p>

<p class="text-danger">本行内容带有一个 danger class</p>

<del>这是删除线</del>

</div>

#### 2.3 文本对齐方式

<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-uppercase">English Upper，英文大写</p>

<p class="text-lowercase">English Lower 英文小写</p>

<p class="text-capitalize"> english 首字母大写</p>

#### 2.4 列表

<!-- 列表，list-group会呈现类似表格的东西 -->

<ul class="list-group">

<li class="list-group-item">免费域名注册</li>

<li class="list-group-item">免费 Window 空间托管</li>

<li class="list-group-item">图像的数量

<!-- 会在列表最后出现一个“新”字 -->

<span class="badge">新</span>

</li>

<li class="list-group-item">24\*7 支持</li>

<li class="list-group-item">每年更新成本</li>

</ul>

<!-- 在列表上，添加超链接 -->

<a href="#" class="list-group-item active">

免费域名注册

</a>

<a href="#" class="list-group-item">24\*7 支持</a>

<a href="#" class="list-group-item">免费 Window 空间托管</a>

<a href="#" class="list-group-item">图像的数量</a>

<a href="#" class="list-group-item">每年更新成本</a>

显示结果:

#### 2.5 表格

<!-- .table 是一个基类，其他都在其基础上

.table-bordered 给表格加外框

.table-hover 鼠标悬停，表格变色

.table-striped 斑马线效果，隔行变色。

.table-condensed 使表格更加紧凑

-->

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

table-condensed">

<tr>

<th>first</th>

<th>second</th>

</tr>

<tr>

<td>002</td>

<td>003</td>

</tr>

<tr>

<td>004</td>

<td>005</td>

</tr>

</table>

#### 2.6 响应式表格

<!--

.table-responsive 给table的父类加，实现响应式表格

在手机端显示不全时，会出现滚动条。

-->

<div class="table-responsive">

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

table-condensed">

<tr>

<th>first</th>

<th>second</th>

</tr>

<tr>

<td>002</td>

<td>003</td>

</tr>

<tr>

<td>004</td>

<td>005</td>

</tr>

</table>

</div>

#### 2.7 响应式图片

所谓的响应式，就是能否随着界面的放大缩小而放大缩小。

<!-- .img-responsive 响应式图片 -->

<img src="" class="img-responsive" alt="">

<!--

.img-rounded 圆角矩形

.img-circle 椭圆形

.img-thumbnail 给图片加圆角矩形边框

指定图片以什么形状显示。

-->

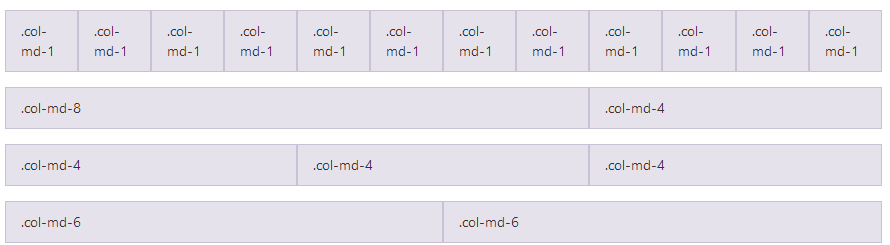
<img src="..." alt="..." class="img-rounded">

<img src="..." alt="..." class="img-circle">

<img src="..." alt="..." class="img-thumbnail">

### 栅格系统进行布局

#### 3.1 栅格系统基础知识



<!-- 栅格系统

1. 栅格系统每行row最多只能有12列col，如果超出

就会移到下一行

2. 栅格系统只能放在容器中

1）<div class="container"></div>

2）<div class="container-fluid"></div>

-->

<div class="row">

<div class="col-md-4">.col-md-4</div>

<div class="col-md-4">.col-md-4</div>

<div class="col-md-4">.col-md-4</div>

</div>

#### 3.2 栅格参数



#### 3.3 栅格列偏移——都是右移

<div class="row">

<div class="col-md-4">.col-md-4</div>

<!-- 偏移4列，每行最多12列，也就是说偏移量最多11个 -->

<div class="col-md-4 col-md-offset-4">.col-md-4 .col-md-offset-4</div>

</div>

#### 3.4 栅格排序——左移/右移

<div class="row">

<!-- 向右移3格 -->

<div class="col-md-9 col-md-push-3">.col-md-9 .col-md-push-3</div>

<!-- 向左移9格 -->

<div class="col-md-3 col-md-pull-9">.col-md-3 .col-md-pull-9</div>

</div>

### 第四章

### 第五章

## Javascript

### 第一章 JavaScript基础

#### 1.1 基础语法

JavaScript 是完全的面向对象脚本语言。

JavaScript 中一切都是对象，因此无论是数字还是字符串都有对应的方法进行操作。

1. 注释

// 单行注释

/\*

多行注释

\*/

2. 变量

var x =3;

var y =3.14;

var z ="hello";

var value; //value = undefined

3. 数据类型

// JavaScript拥有数字，字符串，布尔

// 数组，对象，Null，Undefined这几种数据类型

1. 数字 - JavaScript 只有一种数字类型。不区分整数和小数

var num = new Number(20);

var x = 34;

var y =3.14;

var z = 123e5; //12300000

2. 字符串

var str = new String("JavaScript");

var str = "JavaScript";

var sen = "my name is 'Gaylen'"; //引号嵌套

3. 布尔

var bool = new Boolean(false);

var bool = true;

var x =false;

4. 数组

var arr = new Array();

var arr = new Array(5);

var arr = new Array(1,2,3,4);

var arr =

5. 对象

var obj = new Object();

obj.name="mike";

var person={

name:"Bill",

age:23,

method:function{}

}

var obj = new Object();

6. null //清除

var value = null;

7. undefined //变量中不含有值

var value ;

#### 1.2 流程控制

运算符：

|  |  |  |  |
| --- | --- | --- | --- |
| 算术运算符 | 赋值运算符 | 比较运算符 | 逻辑运算符 |
| + 加 | = | == 等于 | && 与 |
| - 减 | += | === 全等（值和类型） | || 或 |
| \* 乘 | -= | != 不等于 | ! 非 |
| / 除 | \*= | > 大于 | (condition)?value1:value2;  三目运算符 |
| % 取余 | /= | >= 大于等于 |  |
| ++ 自增 | %= | < 小于 |  |
| -- 自减 |  | <= 小于等于 |  |

流程控制：

// if语句

//1. if语句

if(expression){

//code ...

}

//2. if-else语句

if(expression){

//code ...

}else{

//code ...

}

//3. if - else if - else语句

if(expression1){

//code ...

}else if(expression2){

//code ...

}else{

//code ...

}

//switch 语句

switch(value){

case value1:

// code ...

break;

case value2:

//code ...

break;

default:

///code ...

}

//for 语句

//1. for语句

for(var i ;i<10; i++){

//code ...

}

//2. for - in 语句

var person{

name:"mike",

age:23

};

for(var key in person){

// person[key]

}

//while 循环

//1. while 循环

var count = 0;

while(count<10){

//code

count++;

}

//2. do - while 循环

var count = 0;

do{

//code ...

count++;

}while(count<10);

//break 与 continue

break语句用于跳出整个循环

continue语句用于跳出本次循环，执行下一次循环

#### 1.3 函数

//JavaScript函数

function methodName(){

//code ...

}

//示例：

function myFunc(a,b){

return a+b;

}

#### 1.4 异常

//1. try - catch 语句

try{

// code ...

}catch(err){

// code ...

}

//2. try - catch - throw

try{

//code

if() throw "empty";

}catch(err){

// code ...

}

### 第二章 JavaScript DOM

#### 2.1 获取HTML中的元素

1. 通过id找到HTML元素

var x = document.getElementById("idName");

2. 通过标签名找到HTML元素

var x = document.getElementByTagName("h1");

3. 通过类名找到HTML元素，需要自己定义函数

4. 通过name找到HTML元素，因为name不唯一，因此返回的是元素数组

var x = document.getElementByName();

#### 2.2 改变HTML元素的内容（innerHTML）

// 改变HTML元素的内容

document.getElementById(id).innerHTML=”new HTML”;

//改变HTML元素的属性

document.getElementById(id).attributeName=“new value“；

#### 2.3 改变HTML 元素的样式(CSS)

//改变HTML元素的样式

document.getElementById(id).style.propertyName=”new style”;

//示例：

<button type="button" onclick="document.getElementById('id1').style.color='red'">

#### 2.4 对HTML DOM事件作出反应

#### 2.5 添加/删除HTML 元素

1. 添加元素

/\*\*

\* 添加元素分为两步，创建新元素，追加到已有元素

\*/

var para = document.createElement("input");

//设置属性1

para.value = "这是一个按钮";

//设置属性2

para.setAttribute("type","button");

body.appendChild(para);

//或者

document.getElementByTagName("body").appendChild(pare);

2. 删除元素

/\*\*

\*JavaScript删除元素，分为两步

\* 1. 获取父节点

\* 2. 删除子节点

\*/

var parent = document.getElementById();

var chile = document.getElementById();

parent.removeChild(child);

### 第三章 JavaScript 高级

#### 3.1 面向对象

1. 创建对象

//创建对象

var person = new Object();

person.name = "mike";

...

person.methodName = function{

//code ...

}

//创建直接的对象

var person = {name:"mike",age:23};

//使用函数构造对象

function person(firstname,lastname,age,eyecolor)

{

this.firstname=firstname;

this.lastname=lastname;

this.age=age;

this.eyecolor=eyecolor;

this.methodName = function(parameter){

// code …

};

}

myFather=new person("Bill","Gates",56,"blue");

2.

#### 3.2 常用对象

1. 数字

2. 字符串

3. 数组

4. 日期

5. Boolean

6. Math对象

7. RegExp 正则表达式

#### 3.3 JSON

JSON是指JavaScript的对象表示法

JSON 独立于编程语言

{

employees": [

{ "firstName":"Bill" , "lastName":"Gates" }, //Object1

{ "firstName":"George" , "lastName":"Bush" }, //Object2

{ "firstName":"Thomas" , "lastName":"Carter" } //Object3

]

}

## JQuery

### 第一章 JQuery 基本语法

#### 1.1 Jquery 基础语法

$(document).ready(function(){

$ (selector).action{

//code …

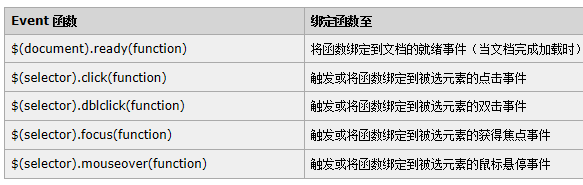
}

});

#### 1.2 Jquery 选择器—HTML的获取



#### 1.3 Jquery 事件



### 第二章 JQuery 操作HTML

#### 2.1 设置内容

•text() - 设置或返回所选元素的文本内容

•html() - 设置或返回所选元素的内容（包括 HTML 标记）

•val() - 设置或返回表单字段的值

$("#btn1").click(function(){

$("#test1").text("Hello world!");

});

$("#btn2").click(function(){

$("#test2").html("<b>Hello world!</b>");

$("#test2").html(function(param){

});

});

$("#btn3").click(function(){

$("#test3").val("Dolly Duck");

});

#### 2.2 设置属性

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

$("#w3s").attr(

"href",<http://www.w3school.com.cn/jquery>

"href",<http://www.w3school.com.cn/jquery>

);

$("#w3s").attr("href",function(param){

// code …

});

});

#### 2.3 删除元素或内容

//remove 删除被选元素以及子元素

// empty 从被选元素中删除子元素

$("#div1").remove();

$("#div1").empty();

//过滤被删除元素，如删除class = italic的所有p元素

$("p").remove(".italic");

#### 2.4 操作CSS

•addClass() - 向被选元素添加一个或多个类

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

$("#div1").addClass("important blue");

});

•removeClass() - 从被选元素删除一个或多个类

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

$("h1,h2,p").removeClass("blue");

});

•toggleClass() - 对被选元素进行添加/删除类的切换操作

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

$("h1,h2,p").toggleClass("blue");

});

•css() - 设置或返回样式属性

//设置样式属性

$("p").css("background-color","yellow");

//返回样式属性

$("p").css("background-color");

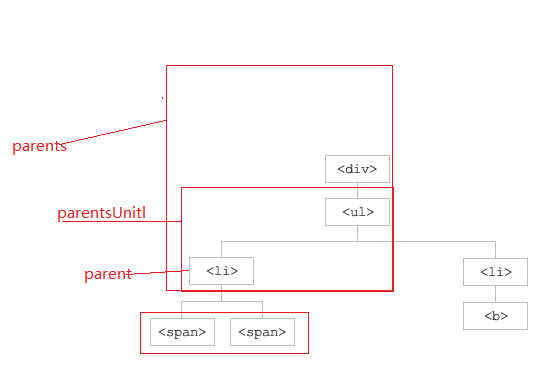
### 第三章 JQuery 实现遍历HTML

#### 3.1 祖先

•parent() 返回span的直接父元素

•parents() 返回span的所有祖先

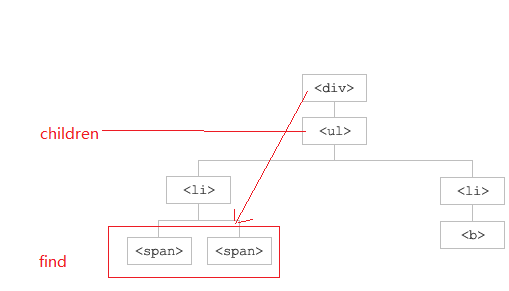
•parentsUntil() 返回span与div之间的祖先



#### 3.2 后代

•children() 返回div的直接子元素

•find() 查找div所有后代中的span元素



#### 3.3 同胞

•siblings() h2的所有同胞

•next() h2的下一个同胞

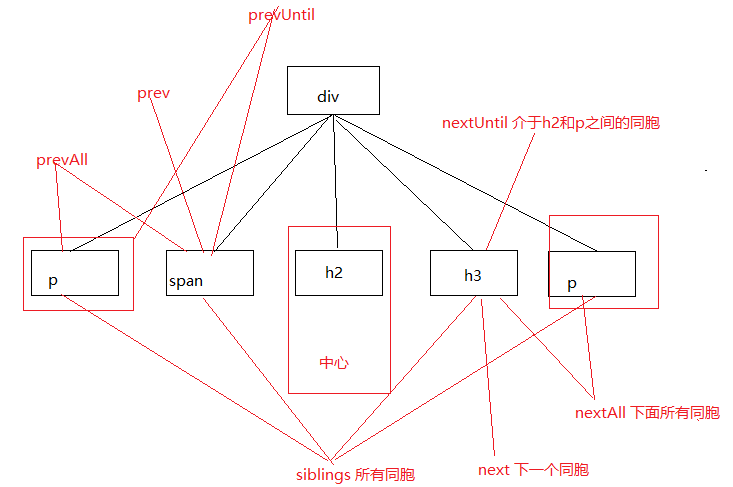
•nextAll() h2后面所有同胞

•nextUntil() h2和p之间的同胞

•prev() h2前一个同胞

•prevAll() h2前面所有同胞

•prevUntil() h2和前面p之间的同胞



#### 3.4 过滤

//选取首个div元素内部的第一个p元素

$(document).ready(function(){

$("div p").first();

});

//选取最后一个div的最后一个p元素

$(document).ready(function(){

$("div p").last();

});

//选取第二个p元素。索引号从0开始

$(document).ready(function(){

$("p").eq(1);

});

### 第四章 JQuery 中的Ajax

### 第五章 JQuery 中的特效

## 第六节 Ajax

### 第一章 Ajax简介

Ajax 是一步的JavaScript和XML。

Ajax并不是新的编程语言。

Ajax是页面与服务器进行数据交换的艺术。不加载整个页面的情况下进行局部刷新。

### 第二章 Ajax 基础

##### 7.2.1 创建XHR对象

创建XMLHttpRequest对象，并且兼容IE5，IE6.

var xmlhttp;

if (window.XMLHttpRequest)

{ // code for IE7+, Firefox, Chrome, Opera, Safari

xmlhttp=new XMLHttpRequest();

}

else

{ // code for IE6, IE5

xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");

}

##### 7.2.2 向服务器发送请求

Get请求示例：

var xmlhttp;

if (window.XMLHttpRequest)

{// code for IE7+, Firefox, Chrome, Opera, Safari

xmlhttp=new XMLHttpRequest();

}

else

{// code for IE6, IE5

xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");

}

xmlhttp.onreadystatechange=function()

{

if (xmlhttp.readyState==4 && xmlhttp.status==200)

{

document.getElementById("myDiv").innerHTML=xmlhttp.responseText;

}

}

xmlhttp.open("GET","../ajax/demo\_get.asp.htm",true);

xmlhttp.send();

##### 7.2.3 获得来自服务器的响应



示例：

document.getElementById("myDiv").innerHTML=xmlhttp.responseText;

##### 7.2.4 onreadystatechange 事件



示例：

xmlhttp.onreadystatechange=function()

{

if (xmlhttp.readyState==4 && xmlhttp.status==200)

{

document.getElementById("myDiv").innerHTML=xmlhttp.responseText;

}

}

### 第三章 Ajax 进阶

### 第四章 Ajax 高级

## 第七节 Angular2

## 第八节 Node.js

### Node.js 介绍

#### 1.1 Node.js 介绍

* 1. 单线程：消耗内存大
  2. 异步：需要函数封装事物
  3. 事件驱动：发生事件时，回调函数

#### 1.2 npm的介绍

npm自身的升级：npm install npm -g

npm 查看自身版本：npm -v

npm 安装express ： npm install express

npm 卸载express: npm uninstall express

npm 更新express ： npm update express

npm 安装express生成器：npm install express-generator -g

#### Express 创建项目

express myapp 使用express生成器创建项目

cd myapp && npm install 安装所需要的依赖

npm start 启动项目服务器

npm install supervisor -g 安装调试工具

### Node.js 基础

#### 2.1 创建第一个helloworld程序

//控制台上输出helloworld

console.log('hello world');

//文件中输出

1 建立文件 test.js

2 写入内容 console.log(“hello world”);

3 运行 node test.js

#### 2.2 创建第一个web服务器

var http = require("http");

http.createServer(function(req,res){

res.writeHead(200,{"content-Type":'text/html'});

//其他的如body和html，head标签，不需要写，会自动生成

res.write("<!DOCTYPE html>");

res.write("<title>this is a helloTest</title>");

res.write("<h1>hello world</h1>");

}).listen(3000);

console.log("正在监听3000端口：");

//启动服务器

node server.js

//使用调试模式启动服务器，要安装supervisor,可以不必每次重启服务器

Supervisor server.js

#### 2.3 回调函数与事件

##### 2.3.1 非阻塞实例

var fs = require("fs");

fs.readFile('file.txt', function (err, data) {

if (err) return console.error(err);

console.log(data.toString());

});

console.log("程序执行结束!");

##### 2.3.2 阻塞实例

var fs = require("fs");

var data = fs.readFileSync('file.txt');

console.log(data.toString());

console.log("程序执行结束!");

##### 2.3.3 事件循环

//引入事件模块

var event = require("events");

//假设建立数据库的链接操作

//数据链接，数据接收，数据库关闭三个操作

//1. 创建数据链接操作函数

var connect = function connection(){

console.log("数据链接成功");

}

//2. 创建数据接受函数

var recive = function recive(){

console.log("数据接收成功");

}

//3. 创建数据库关闭函数

var closed = function close(){

console.log("数据库关闭");

}

//创建事件对象

var eventEmitter = new event.EventEmitter();

//注册事件

eventEmitter.on("connect",connect);

eventEmitter.on("recive",recive);

eventEmitter.on("closed",closed);

//创建事件处理函数\_数据库操作函数

var dbHandle = function dataHandle(){

eventEmitter.emit("connect");

eventEmitter.emit("recive");

eventEmitter.emit("closed");

}

//注册事件处理函数

eventEmitter.on("handler",dbHandle);

//触发事件处理函数

eventEmitter.emit("handler");

### 第三章 Express获取参数列表

Node.js中的express框架获取参数

express获取参数有三种方法：

1. req.query  适合 http://localhost:3000/form?num=8888
2. req.body   适合http://localhost:3000/form，然通过form表单提交
3. req.params  适合获取form后的num：<http://localhost:3000/form/num>

示例：

#### 3.1 req.query

//常见于Get操作：localhost:3000/form?num=8888

var express = require('express');

var app = express();

//获取数据

app.get('/form', function(req, res) {

var num = req.query.num;

res.send("你获取的get数据为:" + num);

});

#### 3.2 req.body.num

//常见于form表单提交

var express = require('express');

var app = express();

//引入body-parser

var bodyParser = require('body-parser');

app.use(express.static('public'));

//需要use的

app.use(bodyParser.json()); // for parsing application/json

app.use(bodyParser.urlencoded({

extended: true

})); // for parsing application/x-www-form-urlencoded

//获取数据

app.post('/form', function(req, res) {

var num = req.body.num;

res.send("你获取的post数据为:" + num);

});

#### 3.3 req.params.num

//常见于RESTful接口设计中 localhost:3000/form/num

var express = require('express');

var app = express();

//获取数据

app.get('/form/:num', function(req, res) {

var num = req.params.num;

res.send("你获取到form/后的参数:" + num);

});

### Mongoose操作

#### 3.1 数据库链接

var mongoose = require('mongoose');

var DB\_URL = "mongodb://192.168.10.3/service";

mongoose.Promise = global.Promise;

mongoose.connect(DB\_URL,{ useMongoClient: true});

var db = mongoose.connection;

db.on('error',console.error.bind(console,'连接错误'));

module.exports=db;

#### 3.2 数据模型Schema与Model

var mongoose = require('mongoose');

var Schema = mongoose.Schema;

//定义一个Schema

var UserSchema = new Schema({

//ID

username: {type:String}, //用户昵称

password: {type:String}, //用户密码

email: {type:String}, //电子邮箱

phone: {type:String}, //手机号码

createDate: {type:Date}, //用户创建时间

right: {type:String,default:'5'} //用于权限验证

},{strict:false});

//生成一个Model

module.exports = mongoose.model('User',UserSchema);

#### 3.3 CURD操作的实现

var User = require('../schema/User');

var db = require('./dataBase');

var UserDao={};

//查询操作

//查询所有用户，可以获取用户列表

UserDao.findByName = function (username, callback) {

db.once('open',function () {

User.find({username:username},function (err,result) {

if(err){

var error = {

msg:"查询失败",

\_type:"查询错误",

status: 210

}

callback(false, error);

}else{

if(result.length == 0){

var error = {

msg:"查询为空",

\_type:"空错误",

status: 200

}

callback(false, error);

}else{

callback(true, result);

}

}

});

});

};

//通过ID获取用户信息

UserDao.findById = function (user\_id,callback) {

db.once('open',function () {

User.find({\_id:user\_id},function (err,result) {

if(err) {

var error = {

msg:"查询失败",

\_type:"查询错误"

}

callback(false,error);

}

else{

if(result.length == 0){

var error = {

msg:"查询为空",

\_type:"空错误"

}

callback(false,error);

}else{

callback(true,result);

}

}

})

})

}

//通过email获取用户信息

UserDao.findByEmail = function (email,callback) {

db.once('open',function () {

User.find({email:email},function (err,result) {

if(err){

var error = {

msg:"查询失败",

\_type:"查询错误"

}

callback(false,error);

}else{

if(result.length == 0){

var error = {

msg:"查询为空",

\_type:"空错误"

}

callback(false,error);

}else{

callback(true,result);

}

}

});

});

};

//获取用户列表

UserDao.findAll = function (callback) {

db.once('open',function () {

User.find(function (err,result) {

if(err){

var error = {

msg:"查询失败",

\_type:"查询错误"

}

callback(false,error);

}else{

if(result.length == 0){

var error = {

msg:"查询为空",

\_type:"空错误"

}

callback(false,error);

}else{

callback(true,result);

}

}

})

})

};

//通过ID删除

UserDao.removeById = function (user\_id,callback) {

db.once('open',function () {

User.remove({\_id:user\_id},function (err,result) {

if(err){

var error = {

msg:"删除失败",

\_type:"删除"

}

callback(false,error);

}

result = []

callback(true,result);

})

})

}

//删除操作,默认名字唯一

UserDao.removeByName = function (username,callback) {

db.once('open',function () {

User.remove({username:username},function (err) {

if(err){

var error = {

msg:"删除失败",

\_type:"删除"

}

callback(false,error);

}

result = [];

callback(true,result);

});

});

};

//通过ID删除用户信息

UserDao.updateById = function (user\_id,user,callback) {

db.once('open',function () {

user.\_id = user\_id;

User.remove({\_id:user\_id},function (err) {

if(err){

var error = {

msg:"删除失败",

\_type:"删除"

}

callback(false,error);

}else{

User.create(user,function (err,result) {

if(err){

var error = {

msg:"增加失败",

\_type:"增加错误"

}

callback(false,error);

}

else{

callback(true,result);

}

});

}

});

})

};

//通过名字进行，更新操作，默认名字唯一

UserDao.updateByName = function (user,callback) {

db.once('open',function () {

//按名字/id查询，删除查询到的文档，然后再添加新的文档

User.find({username:user.username},function (err,result) {

if(err){

var error = {

msg:"查询失败",

\_type:"查询错误"

}

callback(false,error);

}else if(0 == result.length){

//没查到就添加

User.create(user,function (err) {

if(err){

var error = {

msg:"增加失败",

\_type:"增加错误"

}

callback(false,error);

}

callback(true,result);

});

}else{

//查到就删除

User.remove({username:user.username},function (err) {

if(err){

var error = {

msg:"删除失败",

\_type:"删除"

}

callback(false,error);

}else{

//删除后添加新的文档

User.create(user,function (err,result) {

if(err){

var error = {

msg:"增加失败",

\_type:"增加错误"

}

callback(false,error);

}

callback(true,result);

});

}

});

}

});

});

};

//增加操作

UserDao.insert = function (user,callback) {

// db.once('open', function () {

User.find({username: user.username}, function (err,result) {

console.log("success");

// console.log(user.nickname)

if(err){

// console.log(result+",err");

var error = {

msg:"查询失败",

\_type:"查询错误"

}

callback(false,error);

}else if (0 == result.length) {

user.createDate = new Date();

User.create(user,function (err,result) {

if(err){

var error = {

msg:"增加失败",

\_type:"增加错误"

}

callback(false,error);

}else {

callback(true,result);

}

});

} else {

// console.log(result+",false");

var error = {

msg:"已经存在，不能插入",

\_type:"昵称已存在"

}

callback(false,error);

}

});

// });

};

module.exports = UserDao;

#### 3.4 多条件查询

##### 3.4.1 针对多个Key进行多条件查询

MetaDao.search = function (*str\_condition*,*callback*) {  
 Meta.find({$or:[{title:new RegExp(*str\_condition*,'gi')},{description:new RegExp(*str\_condition*,'gi')}]},function (*err*,*result*) {  
 var error;  
 if(*err*){  
 error = {  
 msg:"元数据查询失败",  
 \_type:"查询错误"  
 }  
 *callback*(false,error)  
 }else if(0!=*result*.length){  
 *callback*(true,*result*)  
 }  
 else{  
 error = {  
 msg:"查询结果为空",  
 \_type:"空错误"  
 }  
 *callback*(false,error);  
 }  
 })  
}

##### 3.4.2 针对同一个Key进行多条件查询

MetaDao.findModel = function (*subjects*,*callback*) {  
 var regExp = new RegExp("(" + *subjects*.join("|") + ")",'gi');  
 // console.log(regExp)  
 Meta.find({subject:regExp},function (*err*,*result*) {  
 var error;  
 if(*err*){  
 error = {  
 msg:"元数据查询失败",  
 \_type:"查询错误"  
 }  
 *callback*(false,error)  
 }else if(0!=*result*.length){  
 *callback*(true,*result*);  
 }else{  
 error = {  
 msg:"查询结果为空",  
 \_type:"空错误"  
 }  
 *callback*(false,error);  
 }  
 })  
}