**布局类型：**



**布局实现：**

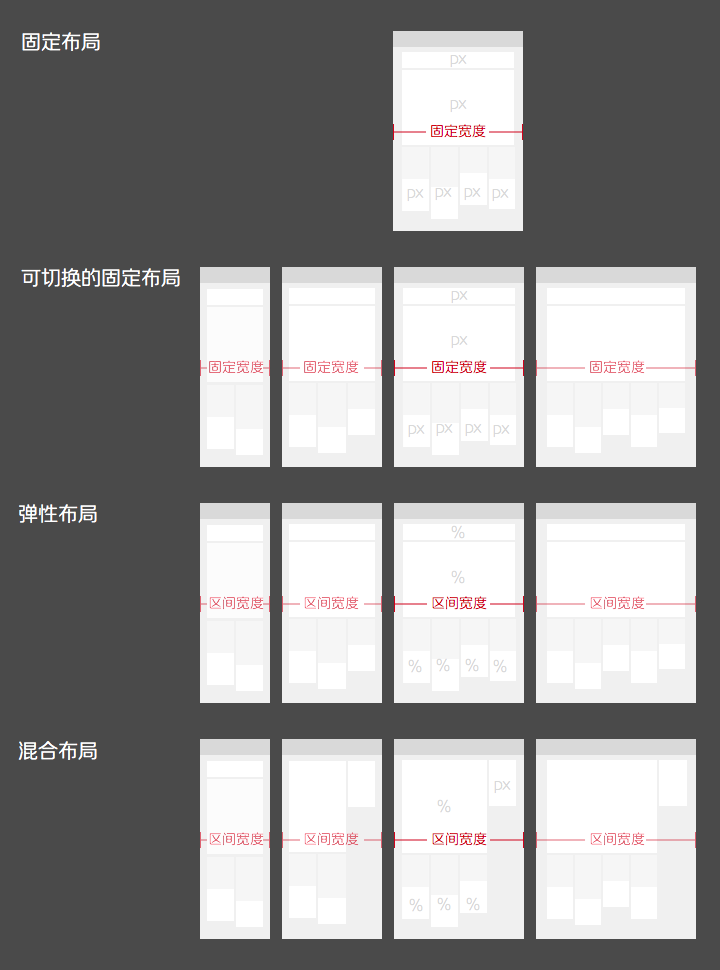
基于页面的实现单位而言，有四种方式实现布局设计，分别是：固定布局，可切换的固定布局，弹性布局，混合布局。

**固定布局**：以像素作为页面的基本单位，不管设备屏幕及浏览器宽度，只设计一套尺寸；

可切换的固定布局：同样以像素作为页面单位，参考主流设备尺寸，设计几套不同宽度的布局。通过设别的屏幕尺寸或浏览器宽度，选择最合适的那套宽度布局；

**弹性布局**：以百分比作为页面的基本单位，可以适应一定范围内所有尺寸的设备屏幕及浏览器宽度，并能完美利用有效空间展现最佳效果；

**混合布局**：同弹性布局类似，可以适应一定范围内所有尺寸的设备屏幕及浏览器宽度，并能完美利用有效空间展现最佳效果；只是混合像素、和百分比两种单位作为页面单位。



可切换的固定布局、弹性布局、混合布局都是目前可被采用的响应式布局方式。

其中可切换的固定布局的实现成本最低，但拓展性比较差；而弹性布局与混合布局效果具响应性，都是比较理想的响应式布局实现方式。只是对于不同类型的页面排版布局实现响应式设计，需要采用不用的实现方式。通栏、等分结构的适合采用弹性布局方式、而对于非等分的多栏结构往往需要采用混合布局的实现方式。



布局响应 对页面进行响应式的设计实现，需要对相同内容进行不同宽度的布局设计，有两种方式：桌面优先（从桌面端开始向下设计）；移动优先（从移动端向上设计）； 无论基于那种模式的设计，要兼容所有设备，布局响应时不可避免地需要对模块布局做一些变化（发生布局改变的临界点称之为断点）， 我们通过JS获取设备的屏幕宽度，来改变网页的布局，这一过程我们可以称之为布局响应屏幕。常见的主要有如下几种方式：

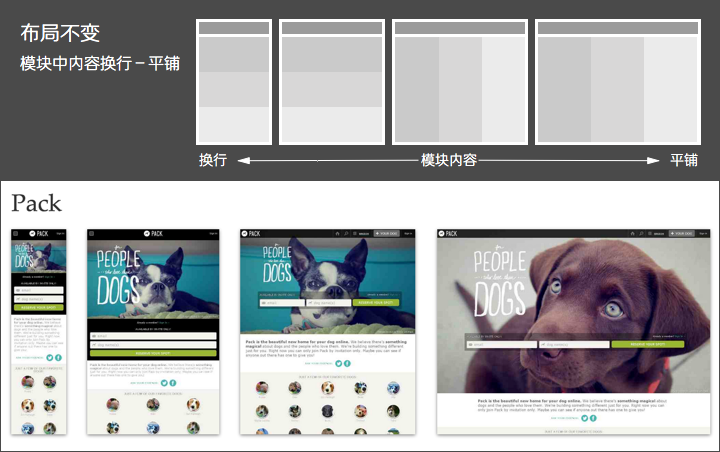
布局不变，即页面中整体模块布局不发生变化，主要有：

模块中内容：挤压－拉伸；

模块中内容：换行－平铺；

模块中内容：删减－增加；







布局改变，即页面中的整体模块布局发生变化，主要有：

模块位置变换；

模块展示方式改变：隐藏－展开；

模块数量改变：删减－增加；

很多时候，单一方式的布局响应无法满足理想效果，需要结合多种组合方式，但原则上尽可能时保持简单轻巧，而且同一断点内（发生布局改变的临界点称之为断点）保持统一逻辑。否则页面实现得太过复杂，也会影响整体体验和页面性能。





requirejs是一个js模块化工具。

为了充分利用优化工具，建议将所有script标签放在html外面，然后只引用require.js来请求加载其他的script： <script data-main="scripts/main" src="scripts/require.js"></script>

在main.js，可以用require()来加载所有需要运行的scripts。这可以确保所有的scripts都是在这里加载的，可以指定data-main script使用异步加载。

API

1.加载JavaScript文件

requirejs主要目的是为了代码的模块化，它鼓励在使用脚本时以module ID代替URL地址。

requirejs以一个相对URL来加载所有代码。页面顶层<script>标签含有一个特殊属性data-main，require.js用它来启动脚本加载过程，而baseURL一般设置到与该属性相一致的目录：<script data-main="scripts/main.js" src="scripts/require.js"></script>。这句话将baseURL设置为scripts，并且加载模块ID为main的脚本文件。

baseURL也可以通过requirejs config手动设置。如果没有显式指定config及data-main，则默认baseURL为包含requirejs的那个HTML所在的目录。

由于requirejs默认假定所有的依赖资源文件都是js文件，因此无需在module ID后加上.js后缀。

一般来说，最好使用baseURL及path config去设置module ID，同时，为了避免凌乱的配置，最好不要使用多级嵌套的目录层次来组织代码，而是要么将所有脚本都放在baseURL中，要么分置为项目库/第三方库的一个扁平化结构。

www/

index.html

js/

app/

sub.js

lib/

jquery.js

canvas.js

app.js

index.html:

<script data-main="js/app.js" src="js/require.js"></script>

app.js:

requirejs.config({

//By default load any module IDs from js/lib

baseUrl: 'js/lib',

//except, if the module ID starts with "app",

//load it from the js/app directory. paths

//config is relative to the baseUrl, and

//never includes a ".js" extension since

//the paths config could be for a directory.

paths: {

app: '../app'

}

});

// Start the main app logic.

requirejs(['jquery', 'canvas', 'app/sub'],

function ($,canvas,sub) {

//jQuery, canvas and the app/sub module are all

//loaded and can be used here now.

});

2.data-main入口点

require.js 在加载的时候会检察data-main 属性。

3.定义模块

模块不同于其他的脚本，它良好的定义了一个作用域空间来避免全局命名空间污染。它可以显式地列出其依赖关系，并以函数（定义此模块的那个函数）参数的形式将这些依赖进行注入，而无需引用全局变量。requirejs的模块是模块模式的一个扩展，其好处是无需全局的引用其他模块。

RequireJS的模块语法允许它尽快地加载多个模块，虽然加载的顺序不定，但依赖的顺序最终是正确的。同时因为无需创建全局变量，甚至可以做到在同一个页面上同时加载同一模块的不同版本。

简单的值对：如果一个模块只包含值对，没有任何依赖，则在define()中定义这些值对就好了

define({

color: "black",

size: "unsize"

});

函数式定义：如果一个模块没有任何依赖，但需要做一个setup工作函数，则在define()中定义该函数，并将其传给define()：

define(function () {

//Do setup work here

return {

color: "black",

size: "unisize"

}

});

存在依赖的函数式定义：如果模块存在依赖，则第一个参数是依赖的名称数组，第二个参数是函数，在模块的所有依赖加载完毕后，该函数会被调来定义该模块，因此该模块返回一个定义了本模块的object。依赖关系会以参数的形式注入到该函数上，参数列表与依赖名称列表一一对应。

//my/shirt.js now has some dependencies, a cart and inventory

//module in the same directory as shirt.js

define(["./cart", "./inventory"], function(cart, inventory) {

//return an object to define the "my/shirt" module.

return {

color: "blue",

size: "large",

addToCart: function() {

inventory.decrement(this);

cart.add(this);

}

}

}

);

本示例创建了一个my/shirt模块，它依赖于my/cart及my/inventory。磁盘上各文件分布如下：

my/cart.js

my/inventory.js

my/shirt.js

模块函数以参数"cart"及"inventory"使用这两个以"./cart"及"./inventory"名称指定的模块。在这两个模块加载完毕之前，模块函数不会被调用。

严重不鼓励模块定义全局变量。遵循此处的定义模式，可以使得同一模块的不同版本并存于同一个页面上(参见 高级用法 )。另外，函参的顺序应与依赖顺序保存一致。

返回的object定义了"my/shirt"模块。这种定义模式下，"my/shirt"不作为一个全局变量而存在。

将模块定义为一个函数：对模块的返回值类型并没有强制为一定是个object，任何函数的返回值都是允许的。此处是一个返回了函数的模块定义：

//A module definition inside foo/title.js. It uses

//my/cart and my/inventory modules from before,

//but since foo/title.js is in a different directory than

//the "my" modules, it uses the "my" in the module dependency

//name to find them. The "my" part of the name can be mapped

//to any directory, but by default, it is assumed to be a

//sibling to the "foo" directory.

define(["my/cart", "my/inventory"],

function(cart, inventory) {

//return a function to define "foo/title".

//It gets or sets the window title.

return function(title) {

return title ? (window.title = title) :

inventory.storeName + ' ' + cart.name;

}

}

);

注意事项：一个文件一个模块；

define()中的相对文件名：为了可以在define()内部使用诸如require("./relative/name")的调用以正确解析相对名称，记得将"require"本身作为一个依赖注入到模块中：

define(["require", "./relative/name"], function(require) {

var mod = require("./relative/name");

});

相对路径在一些场景下格外有用，例如：为了以便于将代码共享给其他人或项目，你在某个目录下创建了一些模块。你可以访问模块的相邻模块，无需知道该目录的名称。

生成相对于模块的URL地址：你可能需要生成一个相对于模块的URL地址。你可以将"require"作为一个依赖注入进来，然后调用require.toUrl()以生成该URL:

define(["require"], function(require) {

var cssUrl = require.toUrl("./style.css");

});

控制台调试:如果你需要处理一个已通过require(["module/name"], function(){})调用加载了的模块，可以使用模块名作为字符串参数的require()调用来获取它:

require("module/name").callSomeFunction()

注意这种形式仅在"module/name"已经由其异步形式的require(["module/name"])加载了后才有效。只能在define内部使用形如"./module/name"的相对路径。

循环依赖：如果你定义了一个循环依赖(a依赖b，b同时依赖a)，则在这种情形下当b的模块函数被调用的时候，它会得到一个undefined的a。b可以在模块已经定义好后用require()方法再获取(记得将require作为依赖注入进来)：

//Inside b.js:

define(["require", "a"],

function(require, a) {

//"a" in this case will be null if a also asked for b,

//a circular dependency.

return function(title) {

return require("a").doSomething();

}

}

);

有一个全局函数requirejs.undef()用来undefine一个模块。它会重置loader的内部状态以使其忘记之前定义的一个模块。

但是若有其他模块已将此模块作为依赖使用了，该模块就不会被清除，所以该功能仅在无其他模块持有该模块时的错误处理中，或者当未来需要加载该模块时有点用。参见备错(errbacks)段的示例。

如果你打算在undefine时做一些复杂的依赖图分析，则半私有的onResourceLoad API可能对你有用。

工作机制：

RequireJS使用head.appendChild()将每一个依赖加载为一个script标签。

RequireJS等待所有的依赖加载完毕，计算出模块定义函数正确调用顺序，然后依次调用它们。

在同步加载的服务端JavaScript环境中，可简单地重定义require.load()来使用RequireJS。build系统就是这么做的。该环境中的require.load实现可在build/jslib/requirePatch.js中找到。

未来可能将该部分代码置入require/目录下作为一个可选模块，这样你可以在你的宿主环境中使用它来获得正确的加载顺序。

Ajax同源策略：Ajax发送请求的url地址与服务器地址必须是同一域名下。

onclick和addEventListener，attachEvent：onclick只能为元素绑定一个事件，绑定多个事件时前面的事件会被后面的事件覆盖掉；后两个方法可以为元素绑定多个事件，依次执行。

万维网联盟创建于1994年，是Web技术领域最具权威和影响力的国际中立性技术标准机构。到目前为止，W3C已发布了200多项影响深远的Web技术标准及实施指南，如广为业界采用的超文本标记语言（标准通用标记语言下的一个应用）、可扩展标记语言（标准通用标记语言下的一个子集）以及帮助残障人士有效获得Web内容的信息无障碍指南（WCAG）等，有效促进了Web技术的互相兼容，对互联网技术的发展和应用起到了基础性和根本性的支撑作用。万维网联盟（外语缩写：W3C）标准不是某一个标准，而是一系列标准的集合。网页主要由三部分组成：结构（Structure）、表现（Presentation）和行为（Behavior）。

对应的标准也分三方面：结构化标准语言主要包括XHTML和XML，表现标准语言主要包括CSS，行为标准主要包括对象模型（如W3C DOM）、ECMAScript等。这些标准大部分由W3C起草和发布，也有一些是其他标准组织制订的标准，比如ECMA（European Computer Manufacturers Association）的ECMAScript标准。

结构标准语言

一、可扩展标记语言（标准通用标记语言下的一个子集、外语缩写：XML）。现推荐遵循的是万维网联盟于2000年10月6日发布的XML1.0。和HTML一样，XML同样来源于标准通用标记语言，可扩展标记语言和标准通用标记语言都是能定义其他语言的语言。XML最初设计的目的是弥补HTML的不足，以强大的扩展性满足网络信息发布的需要，后来逐渐用于网络数据的转换和描述。关于XML的好处和技术规范细节这里就不多说了，网上有很多资料，也有很多书籍可以参考。

二、可扩展超文本标记语言（外语缩写：XHTML）。现推荐遵循的是万维网联盟于2000年1月26日推荐XML1.0。XML虽然数据转换能力强大，完全可以替代HTML，但面对成千上万已有的站点，直接采用XML还为时过早。因此，我们在HTML4.0的基础上，用XML的规则对其进行扩展，得到了XHTML。简单的说，建立XHTML的目的就是实现HTML向XML的过渡。

表现标准语言

一、层叠样式表（外语缩写：CSS）。现推荐遵循的是万维网联盟于1998年5月12日推荐CSS2，CSS3已发布，主流浏览器正在逐渐支持，程序员也开始利用CSS3代替以往冗长的旧代码。万维网联盟创建CSS标准的目的是以CSS取代HTML表格式布局、帧和其他表现的语言。纯CSS布局与结构式XHTML相结合能帮助设计师分离外观与结构，使站点的访问及维护更加容易。

行为标准

一、文档对象模型（外语缩写：DOM）根据W3C DOM规范，DOM是一种与浏览器，平台，语言的接口，使得你可以访问页面其他的标准组件。简单理解，DOM解决了Netscaped的Javascript和Microsoft的Jscript之间的冲突，给予web设计师和开发者一个标准的方法，让他们来访问他们站点中的数据、脚本和表现层对象。

二、ECMAScript　ECMAScript是ECMA(European Computer Manufacturers Association)制定的标准脚本语言（JAVAScript）。现推荐遵循的是ECMAScript 262

一、目的: 为什么要遵循标准

我们作为生产者实际上只是位于中游，既不是上游的浏览器制造商，他们是标准的真正制定者，也不算是下游，他们是浏览器的终端使用者。这个角色就意味着我们位于一个接口的位置，我们需要想办法满足下游的用户使用上游不同浏览器时看到的是同样的内容。用一个程序的语言说，我们是转换器……adapter，我们想方设法让我们的页面、我们的程序能够支持所有的浏览器、能够满足尽可能多的用户。我们要满足所有的客户，即使做不到，我们也要满足我们技术范围之内的所有用户。

二、目标

一个标准制作的网站，让你压根感觉不到跟标准有关。

三、受众

所有ui设计师、技术工程师、运营维护人员。

四、标准内容

抛弃声明：以后我们将抛弃font标签，新的页面中不应该再出现如<font color=”red”></font>，已经存在的老的页面也应该在修改时尽量替代，替代方法：<span

class=”red\_tex”></span>。

一个标准XHTML头信息格式如下：

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="gb2312">

<head>

<title>慧聪网\_中国领先的B2B电子商务平台、电子商务网站</title>

<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />

<meta name="description" content="慧聪网(hc360.com)是为中小企业提供B2B电子商务服务的网上贸易平台，是企业寻求电子商务网络贸易信息的首选行业门户。中小企业通过慧聪网可以快速发布产品供求信息并达成交易。慧聪网为您提供全面的B2B行业资讯，供应，求购，库存信息，努力让每个老板都成为B2B电子商务活动的一员。" />

<meta name="keywords" content="B2B,hc360,电子商务,供求信息,加工,库存,二手,网上贸易,行业门户,商机,网上交易,行业资讯,在线交易,中小企业,供应,求购,商务网,商业贸易,贸易平台" />

</head>

1、什么是DOCTYPE

DOCTYPE是document type(文档类型)的简写，用来说明你用的XHTML或者HTML是什么版本。其中的DTD(例如xhtml1-transitional.dtd)叫文档类型定义，里面包含了文档的规则，浏览器就根据你定义的DTD来解释你页面的标识，并展现出来。要建立符合标准的网页，DOCTYPE声明是必不可少的关键组成部分；除非你的XHTML确定了一个正确的DOCTYPE，否则你的标识和CSS都不会生效。

XHTML 1.0 提供了三种DTD声明可供选择：

i) 过渡的(Transitional)：要求非常宽松的DTD，它允许你继续使用HTML4.01的标识(但是要符合xhtml的写法)。

完整代码如下：

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

ii) 严格的(Strict)：要求严格的DTD，你不能使用任何表现层的标识和属性，例如<br>。

完整代码如下：

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

iii) 框架的(Frameset)：专门针对框架页面设计使用的DTD，如果你的页面中包含有框架，需要采用这种DTD。

完整代码如下：

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">

注：DOCTYPE声明必须放在每一个XHTML文档最顶部，在所有代码和标识之上。

2、名字空间 namespace

<html xmlns="http://www.w3.org/1999/xhtml" lang="gb2312">

通常我们HTML4.0的代码只是<html>，这里的"xmlns"是什么呢？

这个“xmlns”是XHTML namespace的缩写，叫做“名字空间”声明。XHTML是HTML向XML过渡的标识语言，它需要符合XML文档规则，因此也需要定义名字空间。又因为XHTML1.0不能自定义标识，所以它的名字空间都相同，就是"http://www.w3.org/1999/xhtml"。目前阶段我们只要照抄代码就可以了。

3、定义语言编码

<meta http-equiv=“Content-Type” content=“text/html; charset=gb2312” />

为了被浏览器正确解释和通过W3C代码校验，所有的XHTML文档都必须声明它们所使用的编码语言，我们一般使用gb2312(简体中文)，制作多国语言页面也有可能用Unicode、ISO-8859-1等，根据你的需要定义。

注：如果忘记了定义语言编码,可能就会出现,你在DW(dreamweaver)做完一个页面，第二次打开时所有的中文变成了乱码。

4、Javascript定义

Js必须要用<script language="javascript" type="text/javascript">来开头定义，而不是原来的<script language=javascript>或干脆直接<script>，并且需要加个注释符<!-- -->，已经存在的老的页面也应该在修改时尽量替代，替代方法：保证不在不支持js的浏览器上直接显示出代码来。

<script language="javascript" type="text/javascript">

//<![CDATA[

function show\_layout(selObj){

var n = selObj.options[selObj.selectedIndex].value;

document.getElementById('stylesheet').href = n;

}

//]]>

</script>

5、CSS定义

CSS必须要用<style type=“text/css”>开头来定义，而不是原来的直接<style>，也不建议直接写在内容代码里如：<div style=”padding-left:20px;”></div>，并需要加个注释符<!-- -->

例如：

<style type="text/css" media="screen">

<!--

body {margin:0px;padding:0px;font-size:12px;text-align:center}

-->

</style>

为保证各浏览器的兼容性，在写CSS时请都写上数量单位，例如：错误：.space\_10{padding-left:10} 正确：.space\_10 {padding-left:10px}

6、不要在注释内容中使用“--”

“--”只能发生在XHTML注释的开头和结束，也就是说，在内容中它们不再有效。

例如下面的代码是无效的：<!--这里是注释-----------这里是注释-->

正确的应用等号或者空格替换内部的虚线。<!--这里是注释============这里是注释-->

7、所有标签的元素和属性的名字都必须使用小写

与HTML不一样，XHTML对大小写是敏感的，<title>和<TITLE>是不同的标签。XHTML要求所有的标签和属性的名字都必须使用小写。例如：<BODY>必须写成<body>。大小写夹杂也是不被认可的，通常dreamweaver自动生成的属性名字"onMouseOver"也必须修改成"onmouseover"。

8、所有的属性必须用引号""括起来

在HTML中，你可以不需要给属性值加引号，但是在XHTML中，它们必须被加引号。

例如：<height=80>必须修改为：<height="80">。

特殊情况，你需要在属性值里使用双引号，你可以用"，单引号可以使用&apos;，例如：<alt="say&apos;hello&apos;">

9、把所有<和&特殊符号用编码表示

任何小于号（<），不是标签的一部分，都必须被编码为 &lt;

任何大于号（>），不是标签的一部分，都必须被编码为 &gt;

任何与号（&），不是实体的一部分的，都必须被编码为 &amp;

错误：

http://club.china.alibaba.com/forum/thread/search\_forum.html?action=SearchForum&doSearchForum=true&main=1&catcount=10&keywords=mp3

正确：

http://club.china.alibaba.com/forum/thread/search\_forum.html?action=SearchForum&amp;doSearchForum=true&amp;main=1&amp;catcount=10&amp;keywords=mp3

10、给所有属性赋一个值

XHTML规定所有属性都必须有一个值，没有值的就重复本身。例如：

<td nowrap><input type="checkbox" name="shirt" value="medium" checked>必须修改为：

<td nowrap="nowrap"><input type="checkbox" name="shirt" value="medium" checked="checked" />

11、所有的标记都必须要有一个相应的结束标记

以前在HTML中，你可以打开许多标签，例如<p>和<li>而不一定写对应的</p>和</li>来关闭它们。但在XHTML中这是不合法的。XHTML要求有严谨的结构，所有标签必须关闭。如果是单独不成对的标签，在标签最后加一个"/"来关闭它。

例如：<br />

<img height="80" alt="网页" title=”网页” src="logo.gif" width="200" />

特殊结束标记

错误：

Document.write(“<td width=\"300\"><a href=\"1.html\">ok</a></td>”);

正确:

Document.write(“<td width=\"300\"><a href=\"1.html\">ok<\/a><\/td>”);

在js中，原已结束的标签需要再转义再结束。

12、所有的标记都必须合理嵌套

同样因为XHTML要求有严谨的结构，因此所有的嵌套都必须按顺序，以前我们这样写的代码：

<p><b></p></b>必须修改为：<p><b></b></p>

就是说，一层一层的嵌套必须是严格对称。

错误：

<table><tr><form><td></td></form></tr></table>

正确：

<form><table><tr><td></td></tr></table></form>

13、图片添加有意义的alt属性

例如：<img src="logo.gif" width="100" height="100" align="middle" boder="0" alt="阿里巴巴" />

尽可能的让作为内容的图片都带有属于自己的alt属性。

同理：添加文字链接的title属性。

<a href="#" target="\_blank" title="新闻新闻新闻新闻">新闻新闻…</a>，在一些限定字数的内容展示尤为重要，帮助显示不完成的内容显示完整，而不用考虑页面会因此而撑大。

14、在form表单中增加lable，以增加用户友好度

例如：

<form action="http://somesite.com/prog/adduser" method="post">

<label for="firstname">first name: </label>

<input type="text" id="firstname" />

<label for="lastname">last name: </label>

<input type="text" id="lastname" />

</form>

Pixel像素

px：逻辑像素，浏览器使用的抽象单位。

dp，pt：设备无关像素。

dpr：设备像素缩放比。

Viewport：手机浏览器默认做了两件事：页面渲染在一个980px（ISO）的页面内；缩放。Meta标签：<meta name="viewport" content="name=value,name=value">

width:设置布局viewport的特定值（“device-width”！表示布局viewport等于设备宽度）

initial-scale:设置页面的初始缩放

minimum-scale:最小缩放

maximum-scale:最大缩放

user-scalable:用户是否能缩放

移动web最佳viewport设置：布局viewport = 设备宽度 = 度量viewport

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

设计移动web：

方案一：根据设备的实际宽度来设计（常用），手机宽320px，我们就拿320px来设计。

方案二：1px = 1dp，缩放0.5，根据设备的物理像素dp等于抽象像素px来设计。1像素边框和高清图片都不需要额外处理。

**布局优化**

Flexbox弹性盒子布局

父：display:-webkit-flex;//表示使用弹性布局

子：flex:num;//占容器的比例

不定宽高的水平垂直居中方案：

div{

position:absolute;

top:50%;

left:50%;

z-index:3;

-webkit-transform:translate(-50%,-50%);

}

用Flexbox:

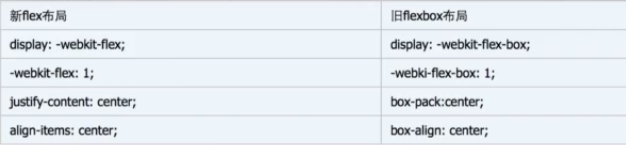
.parent{

justify-content:center;//子元素水平居中

align-items:center;//子元素垂直居中

display:-webkit-flex;

}



**响应式设计**

核心是媒体查询

设计点1：百分比设计

设计点2：弹性图片 max-width:100%

设计点3:重新布局，显示与隐藏

**相对单位：rem**

**em：**是根据父节点为相对单位；

**rem:**是根据HTML为相对单位

em在多层嵌套下会变得非常难用；rem更加能作为全局统一设置的度量。

为了适应各大手机屏幕，rem基值设置为rem=screen.width/20

//单行文本溢出

.inaline{

overflow:hidden;

white-space:nowrap;

text-overflow:ellipsis;

}

//多行文本溢出

.intwoline{

display:-webkit-box !important;

overflow:hidden;

text-overflow:ellipsis;

word-break:break-all;

-webkit-box-orient:vertical;

-webkit-line-clamp:2;

}

语法：

text-shadow ： none | <length> none | [<shadow>, ] \* <shadow> 或none | <color> [, <color> ]\*

也就是：

text-shadow:[颜色(Color) x轴(X Offset) y轴(Y Offset) 模糊半径(Blur)],[颜色(color) x轴(X Offset) y轴(Y Offset) 模糊半径(Blur)]...

或者

text-shadow:[x轴(X Offset) y轴(Y Offset) 模糊半径(Blur) 颜色(Color)],[x轴(X Offset) y轴(Y Offset) 模糊半径(Blur) 颜色(Color)]...

例子和效果：

公共样式

.demo {

background: #666666;

width: 440px;

padding: 30px;

font: bold 55px/100% "微软雅黑", "Lucida Grande", "Lucida Sans", Helvetica, Arial, Sans;;

color: #fff;

text-transform: uppercase;

}

样式1：

.demo1 {

text-shadow: red 0 1px 0;

}



**效果一：Glow and Extra Glow effect(也就是NEON effect)**

.demo2 {

text-shadow: 0 0 20px red;

}



.demo3 {

text-shadow: 0 0 5px #fff, 0 0 10px #fff, 0 0 15px #fff, 0 0 40px #ff00de, 0 0 70px #ff00de;

}



**效果二：Apple Style Effect**

.demo4 {

color: #000;

text-shadow: 0 1px 1px #fff;

}



**效果三：Photoshop Emboss Effect**

.demo5 {

color: #ccc;

text-shadow: -1px -1px 0 #fff,1px 1px 0 #333,1px 1px 0 #444;

}



**效果四：Blurytext Effect**

.demo6 {

color: transparent;

text-shadow: 0 0 5px #f96;

}



.demo7 {

color: transparent;

text-shadow:0 0 6px #F96, -1px -1px #FFF, 1px -1px #444;

}



**效果五：Inset text effect**

.demo8 {

color: #566F89;

background: #C5DFF8;

text-shadow: 1px 1px 0 #E4F1FF;

}



**效果六：Stroke text effect**



**效果七：3D text effect**

.demo10 {

color: #fff;

text-shadow: 1px 1px rgba(197, 223, 248,0.8),2px 2px rgba(197, 223, 248,0.8),3px 3px rgba(197, 223, 248,0.8),4px 4px rgba(197, 223, 248,0.8),5px 5px rgba(197, 223, 248,0.8),6px 6px rgba(197, 223, 248,0.8);

}



.demo11 {

color: #fff;

text-shadow: -1px -1px rgba(197, 223, 248,0.8),-2px -2px rgba(197, 223, 248,0.8),-3px -3px rgba(197, 223, 248,0.8),-4px -4px rgba(197, 223, 248,0.8),-5px -5px rgba(197, 223, 248,0.8),-6px -6px rgba(197, 223, 248,0.8);

}



**效果八：Vintge/Retro text effect**

.demo11 {

color: #eee;

text-shadow: 5px 5px 0 #666, 7px 7px 0 #eee;

}



**效果九：Anaglyphic text effect**

.demo13 {

color: rgba(255, 179, 140,0.5);

text-shadow: 3px 3px 0 rgba(180,255,0,0.5);

}



R：红色值，正整数|百分数。G：绿色值，正整数|百分数。B：蓝色值，正整数|百分数。A：透明度，取值0~1之间。

只要在需要使用颜色的地方都可以用这个样式。

兼容性：IE9+；FireFox：13.0.10+；chrome：2.0.x+；opera：9.64+；safari：4+

圆角半径：

-moz-border-radius: none | <length>{1,4} [/ <length>{1,4} ]?

-webkit-border-radius: none | <length>{1,4} [/ <length>{1,4} ]?

border-radius: none | <length>{1,4} [/ <length>{1,4} ]?

-moz-border-radius-topleft: <length> <length> //左上角

-moz-border-radius-topright: <length> <length> //右上角

-moz-border-radius-bottomright: <length> <length> //右下角

-moz-border-radius-bottomleft: <length> <length> //左下角

-webkit-border-top-left-radius: <length> <length> //左上角

-webkit-border-top-right-radius: <length> <length> //右上角

-webkit-border-bottom-right-radius: <length> <length> //右下角

-webkit-border-bottom-left-radius: <length> <length> // 左下角

border-top-left-radius: <length> <length> //左上角

border-top-right-radius: <length> <length> //右上角

border-bottom-right-radius: <length> <length> //右下角

border-bottom-left-radius: <length> <length> //左下角

需要特别注意的是，border-radius一定要放置在-moz-border-radius和-webkit-border-radius后面。

border-radius设置三个时，top-left取第一个值，top-right等于bottom-left并且他们取第二个值，bottom-right取第三个值。

对于border-radius还有一个内半径和外半径的区别，它主要是元素 边框值较大时，效果就很明显，当我们border-radius半径值小于或等于border的厚度时，我们边框内部就不具有圆角效果。

面向对象

优点：可维护（要修改时只需要修改需要修改的地方）、可复用（重复使用）、可扩展（添加功能）、灵活性好

通过封装继承多态使程序耦合度降低；使用设计模式使得程序更加灵活，容易修改，并且易于复用。

业务逻辑和界面逻辑分开。

简单工厂模式：又称为静态工厂方法，可以根据传递参数的不同，返回不同的实例。

该模式将对象的创建和对象本身的业务处理分离了，可以降低系统的耦合度。

call和apply

1、方法定义

call方法:

语法：call([thisObj[,arg1[, arg2[, [,.argN]]]]])

定义：调用一个对象的一个方法，以另一个对象替换当前对象。

说明：

call 方法可以用来代替另一个对象调用一个方法。call 方法可将一个函数的对象上下文从初始的上下文改变为由 thisObj 指定的新对象。

如果没有提供 thisObj 参数，那么 Global 对象被用作 thisObj。

apply方法：

语法：apply([thisObj[,argArray]])

定义：应用某一对象的一个方法，用另一个对象替换当前对象。

说明：

如果 argArray 不是一个有效的数组或者不是 arguments 对象，那么将导致一个 TypeError。

如果没有提供 argArray 和 thisObj 任何一个参数，那么 Global 对象将被用作 thisObj， 并且无法被传递任何参数。

call和apply方法用于改变函数执行的上下文，即this的指向。第一个实参是要调用函数的母对象，它是调用上下文，在函数体内通过this来获得对它的引用。如：

f.call(o);

f.apply(o);

这两行代码分别相当于：

o.m = f;//将f存储为o的临时方法

o.m();//调用它，不传入参数

delete o.m;//将临时方法删除

对于call来说，第一个调用上下文实参后的所有实参就是要传入待调用函数的值。而apply的实参都被放到一个数组中：

f.call(o,1,2,3);

f.apply(o,[1,2,3]);

几种常见的网页布局形式

网页常见的布局结构有“国”字形布局、“匡”字形布局、“三”字形布局、“川”字形布局、封面型布局、Flash布局、标题文本型布局、框架型布局和变化型布局等。

1.“国”字形布局

也可以称为“同”字型，是一些大型网站所喜欢的类型，即最上面是网站的标题以及横幅广告条，接下来就是网站的主要内容，左右分列两小条内容，中间是主要部分，与左右一起罗列到底，最下面是网站的一些基本信息、联系方式、版权声明等。这种结构是我们在网上见到的差不多最多的一种结构类型。



2.“匡”字形布局

这种结构与上一种其实只是形式上的区别，它去掉了“国”字形布局的最右边的部分，给主内容区释放了更多空间。这种布局上面是标题及广告横幅，接下来的左侧是一窄列链接等，右列是很宽的正文，下面也是一些网站的辅助信息。

3.“三”字形布局

这是一种简洁明快的网页布局，在国外用的比较多，国内比较少见。这种布局的特点是在页面上由横向两条色块将网页整体分割为3部分，色块中大多放置广告条与更新和版权提示。如图所示即是一种三字形布局的网页。

4.“川”字形布局

整个页面在垂直方向分为三列，网站的内容按栏目分布在这三列中，最大限度地突出主页的索引功能。如图1-19所示的“Beijing 2008”网站就是一种川字形的布局。

5.海报型布局

这种类型基本上是出现在一些网站的首页，大部分为一些精美的平面设计结合一些小的动画，放上几个简单的链接或者仅是一个“进入”的链接甚至直接在首页的图片上做链接而没有任何提示。这种类型大部分出现在企业网站和个人主页，如果说处理的好，会给人带来赏心悦目的感觉。

6.Flash布局

这种布局是指整个网页就是一个Flash动画，它本身就是动态的，画面一般比较绚丽、有趣，是一种比较新潮的布局方式。其实这种布局与封面型结构是类似的，不同的是由于Flash强大的功能，页面所表达的信息更丰富。其视觉效果及听觉效果如果处理得当，会是一种非常有魅力的布局。

7.标题文本型布局 标题文本型布局是指页面内容以文本为主，这种类型页面最上面往往是标题或类似的一些东西，下面是正文，比如一些文章页面或注册页面等就是这一类。

8.框架型布局 采用框架布局结构，常见的有左右框架型、上下框架型和综合框架型。由于兼容性和美观等因素，这种布局目前专业设计人员采用的已不多，不过在一些大型论坛上还是比较受青睐的，有些企业网站也有应用。 9.变化型布局 即上述几种类型的结合与变化。

console方法整理

console.assert(expression,object[object])：接受至少两个参数，若第一个表达式返回的值为false，则会在控制台输出第二个参数。

console.count([label])：执行到该行的次数。

console.dir(object)：将传入对象的属性以列表的形式输出。

console.error(object[,object])：用于输出错误信息，输出的信息会标记为错误样式，便于分辨。

console.group(),console.groupEnd()：让控制台语句产生不同层级嵌套关系，每个group()会增加一层嵌套，每个groupEnd()减少一层嵌套。

和console.group()相似的方法是console.groupCollapsed()作用相同，不同点是嵌套的输出内容是折叠状态，在有大段内容输出的时候使用这个方法可以使输出版面不至于太长。console.info(object[, object...])：用于输出信息。

console.table()：将传入的数组或对象以表格的形式输出。

console.profile([profileLabel])和console.profileEnd()：用于性能分析。在 Profiles 面板下看到这两个方法之间代码运行过程中的消耗时间。

console.time()和console.timeEnd()：将两个方法之间的代码的运行时间输出到控制台。

console.trace()用来追踪函数的调用过程。

console.warn(object[, object...])：输出参数的内容，作为警告提示。

console对象上的五个直接输出方法，console.log(),console.warn(),console.error(),console.exception()(等同于console.error())和console.info()，都可以使用占位符。支持的占位符有四种，分别是字符(%s)、整数(%d 或%i)、浮点数(%f)和对象(%o)。

前端的MVC框架：AngularJS、Backbone.js、Ember.js

display为none的图片会加载。

CSS中不可继承的属性：display、margin、border、padding、background、height、min-height、max- height、width、min-width、max-width、overflow、position、left、right、top、 bottom、z-index、float、clear、table-layout、vertical-align、page-break-after、 page-bread-before和unicode-bidi。

所有元素可继承：visibility和cursor。

内联元素可继承：letter-spacing、word-spacing、white-space、line-height、color、font、 font-family、font-size、font-style、font-variant、font-weight、text- decoration、text-transform、direction。

块状元素可继承：text-indent和text-align。

列表元素可继承：list-style、list-style-type、list-style-position、list-style-image。

表格元素可继承：border-collapse。

脱离文档流：float，position为absolute或fixed。

想要用UDP达到TCP一样安全的效果，可以在会话层做文章。

js绑定事件监听函数：

1.传统方法：document.getElementById("btn").onclick = function(){ //do something };

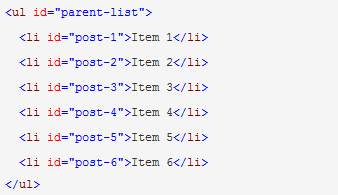
2.DOM绑定(W3C）：document.getElementById("btn").addEventListener('click',function(){ //do something },false);最后一个参数false表示冒泡，true表示捕获

3.DOM绑定（IE）：document.getElementById("btn").attachEvent('onclick',function(){ //do something });

**delegate（事件代理）**

当我们需要对很多元素添加事件的时候，可以通过将事件委托给他们的父元素来触发处理函数。这主要得益于事件的冒泡机制。

假设有一个ul，包含了很多li子节点：



要对li添加事件，通常情况下会这样写：

function addListeners4Li(liNode) {

liNode.onclick = function clickHandler() {

alert(this.id);

};

}

window.onload = function() {

var ulNode = document.getElementById("parent-list-1");

var liNodes = ulNode.getElementsByTagName("li");

for(var i = 0, l = liNodes.length; i < l; i++) {

addListeners4Li(liNodes[i]);

}

}

如果这个ul中的li子元素会频繁地添加或者删除，我们就需要在每次添加li的时候都调用这个addListeners4Li方法来为每个li节点添加事件处理函数。这就添加的复杂度和出错的可能性。

更简单的方法是使用事件代理机制，当事件被抛到更上层的父节点的时候，我们通过检查事件的目标对象（target）来判断并获取事件源li。下面的代码可以完成我们想要的效果：

document.getElementById("parent-list-2").addEventListener("click", function(e) {

// 检查事件源e.targe是否为li

if(e.target != null && e.target.nodeName.toUpperCase() == "LI") {

// 真正的处理过程在这里

alert("List item "+ e.target.id +" was clicked!");

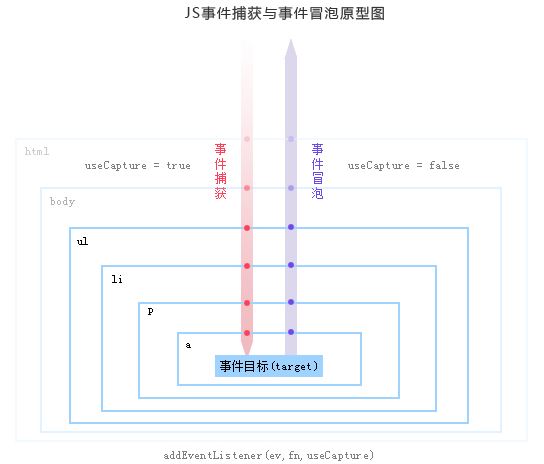
}

});

为父节点添加一个click事件，当子节点被点击的时候，click事件会从子节点开始向上冒泡。父节点捕获到事件之后，通过判断 e.target.nodeName来判断是否为我们需要处理的节点。并且通过e.target拿到了被点击的li节点。从而可以获取到相应的信息，并作处理。

**事件冒泡及捕获**

DOM2.0模型将事件处理流程分为三个阶段：事件捕获阶段，事件目标阶段，事件起泡阶段。



事件捕获：当某个元素触发某个事件（如onclick），顶层对象document就会发出一个事件流，随着DOM树的节点向目标元素节点流去，直到到达事件真正发生的目标元素。在这个过程中，事件相应的监听函数是不会被触发的。

事件目标：当到达目标元素之后，执行目标元素该事件相应的处理函数。如果没有绑定监听函数，那就不执行。

事件起泡：从目标元素开始，往顶层元素传播。途中如果有节点绑定了相应的事件处理函数，这些函数都会被依次触发。如果想阻止事件起泡，可以使用e.stopPropagation()（Firefox）或者e.cancelBubble=true（IE）来组织事件的冒泡传播。

**事件代理的优点：**

1.管理的函数变少了。不需要为每个元素都添加监听函数。对于同一个父节点下面类似的子元素，可以通过委托给父元素的监听函数来处理事件。

2.可以方便地动态添加和修改元素，不需要因为元素的改动而修改事件绑定。

3.JavaScript和DOM节点之间的关联变少了，这样也就减少了因循环引用而带来的内存泄漏发生的概率。

**jQuery中的事件代理：**

$("#link-list").delegate("a", "click", function(){

// "$(this)" is the node that was clicked

console.log("you clicked a link!",$(this));

});

jQuery的delegate的方法需要三个参数，一个选择器，一个事件名称，和事件处理函数。

引用传递和值传递（Java）

按值传递：指的是在调用方法时，传递的参数是按值的拷贝传递。传递的是值的拷贝，也就是说传递完之后就互不相干了。

public class TempTest {

private void test1(int a){

a = 5;

System.out.println("test1方法中的a="+a);

}

public static void main(String[] args) {

TempTest t = new TempTest();

int a = 3;

t.test1(a);//传递后，test1方法对变量值的改变不影响这里的a

System.out.println(”main方法中的a=”+a);

}

}

按引用传递：指在方法调用时，传递的参数是按引用进行传递，其实传递的是引用的地址，也就是变量所对应的内存空间的地址。传递的是值的引用，也就是说传递前和传递后都指向同一个引用（也就是同一个内存空间）。

public class TempTest {

private void test1(A a){

a.age = 20;

System.out.println("test1方法中的age="+a.age);

}

public static void main(String[] args) {

TempTest t = new TempTest();

A a = new A();

a.age = 10;

t.test1(a);

System.out.println(”main方法中的age=”+a.age);

}

}

class A{

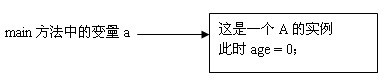
public int age = 0;

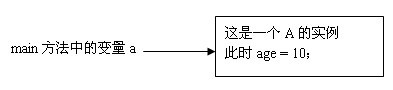
}

理解按引用传递的过程——内存分配示意图

要想正确理解按引用传递的过程，就必须学会理解内存分配的过程，内存分配示意图可以辅助我们去理解这个过程。

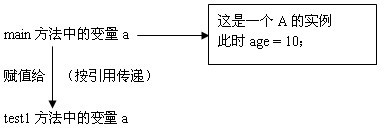
用上面的例子来进行分析：

（1）：运行开始，运行第8行，创建了一个A的实例，内存分配示意如下：

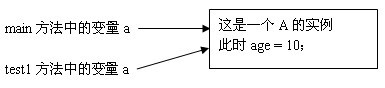
（2）：运行第9行，是修改A实例里面的age的值，运行后内存分配示意如下：

（3）：运行第10行，是把main方法中的变量a所引用的内存空间地址，按引用传递给test1方法中的a变量。请注意：这两个a变量是完全不同的，不要被名称相同所蒙蔽。

内存分配示意如下：

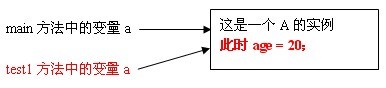


由于是按引用传递，也就是传递的是内存空间的地址，所以传递完成后形成的新的内存示意图如下：



也就是说：是两个变量都指向同一个空间。

（4）：运行第3行，为test1方法中的变量a指向的A实例的age进行赋值，完成后形成的新的内存示意图如下：



此时A实例的age值的变化是由test1方法引起的

（5）：运行第4行，根据此时的内存示意图，输出test1方法中的age=20

（6）：运行第11行，根据此时的内存示意图，输出main方法中的age=20

对上述例子的改变

理解了上面的例子，可能有人会问，那么能不能让按照引用传递的值，相互不影响呢？就是test1方法里面的修改不影响到main方法里面呢？

方法是在test1方法里面新new一个实例就可以了。改变成下面的例子，其中第3行为新加的：

public class TempTest {

private void test1(A a){

a = new A();//新加的一行

a.age = 20;

System.out.println("test1方法中的age="+a.age);

}

public static void main(String[] args) {

TempTest t = new TempTest();

A a = new A();

a.age = 10;

t.test1(a);

System.out.println(”main方法中的age=”+a.age);

}

}

class A{

public int age = 0;

}

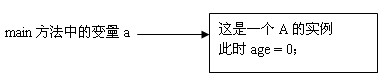
运行结果为：

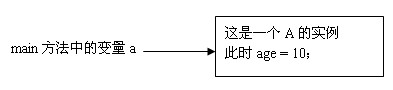
test1方法中的age=20

main方法中的age=10

为什么这次的运行结果和前面的例子不一样呢，还是使用内存示意图来理解一下

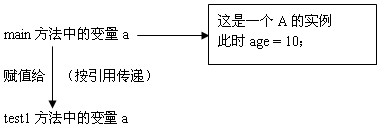
再次理解按引用传递

（1）运行开始，运行第9行，创建了一个A的实例，内存分配示意如下：

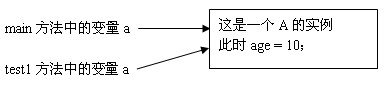
（2）运行第10行，是修改A实例里面的age的值，运行后内存分配示意如下：

（3）运行第11行，是把main方法中的变量a所引用的内存空间地址，按引用传递给test1方法中的a变量。请注意：这两个a变量是完全不同的，不要被名称相同所蒙蔽。

内存分配示意如下：

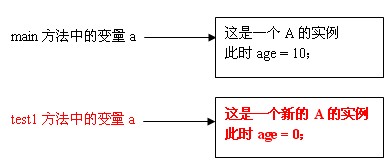


由于是按引用传递，也就是传递的是内存空间的地址，所以传递完成后形成的新的内存示意图如下：

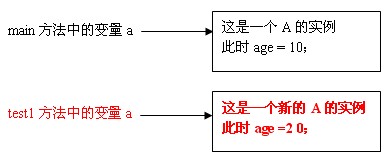


也就是说：是两个变量都指向同一个空间。

（4）运行第3行，为test1方法中的变量a重新生成了新的A实例的，完成后形成的新的内存示意图如下：



（5）运行第4行，为test1方法中的变量a指向的新的A实例的age进行赋值，完成后形成的新的内存示意图如下：



注意：这个时候test1方法中的变量a的age被改变，而main方法中的是没有改变的。

（6）运行第5行，根据此时的内存示意图，输出test1方法中的age=20

（7）运行第12行，根据此时的内存示意图，输出main方法中的age=10

说明

（1）“在Java里面参数传递都是按值传递”这句话的意思是：按值传递是传递的值的拷贝，按引用传递其实传递的是引用的地址值，所以统称按值传递。

（2）在Java里面只有基本类型和按照下面这种定义方式的String是按值传递，其它的都是按引用传递。就是直接使用双引号定义字符。

1.input作为按钮：type属性设置为submit或者button。局限性：由于它是自闭合标签，故不能设置复杂的样式，无法实现图文混排的效果。需要进行图文混排时，应使用button标签。

2.搜索框中的form：action决定输入的内容会被提交到服务器的哪个端口；target属性决定搜索结果的展示方式，如取\_blank表示搜索结果会在新的页面显示；method决定提交的方式。

当input的type="submit"时，可以提交表单内容到服务器。此时按钮必须放在表单内。

1.AngularJS表达式：

angular表达式写在双大括号内：{{expression}}

angular把数据绑定到HTML，这与ng-bind有异曲同工之妙

angular将在表达式书写的位置“输出”数据

angular表达式可以包含文字、运算符、变量

数字： <div ng-app="" ng-init="quantity=1;cost=5">

<p>总价： {{ quantity \* cost }}</p>

</div>

字符串：<div ng-app="" ng-init="firstName='John';lastName='Doe'">

<p>姓名： {{ firstName + " " + lastName }}</p>

</div>

对象： <div ng-app="" ng-init="person={firstName:'John',lastName:'Doe'}">

<p>姓为 {{ person.lastName }}</p>

</div>

数组： <div ng-app="" ng-init="points=[1,15,19,2,40]">

<p>第三个值为 {{ points[2] }}</p>

</div>

2.AngularJS指令：

angular通过被称为“指令”的新属性来扩展HTML，通过内置的属性来为应用添加功能，允许自定义属性。

angular指令是扩展的HTML属性，带有前缀ng-

ng-app指令初始化一个AngularJS应用程序，一个网页可以包含多个运行在不同元素中的 AngularJS 应用程序。。

ng-init指令初始化应用程序数据

ng-model指令把元素值（比如输入域的值）绑定到应用程序。

angular指令：<div ng-app="" ng-init="firstName='John'">

<p>在输入框中尝试输入：</p>

<p>姓名：<input type="text" ng-model="firstName"></p>

<p>你输入的为： {{ firstName }}</p>

</div>

ng-app 指令告诉 AngularJS，<div> 元素是 AngularJS 应用程序 的"所有者"。

数据绑定： <div ng-app="" ng-init="quantity=1;price=5">

<h2>价格计算器</h2>

数量： <input type="number" ng-model="quantity">

价格： <input type="number" ng-model="price">

<p><b>总价：</b> {{ quantity \* price }}</p>

</div>

重复HTML元素：<div ng-app="" ng-init="names=['Jani','Hege','Kai']">

<p>使用 ng-repeat 来循环数组</p>

<ul>

<li ng-repeat="x in names">

{{ x }}

</li>

</ul>

</div>

ng-app定义了AngularJS应用程序的根元素，它在网页加载完成时会自动引导（自动初始化）应用程序。

ng-init指令为AngularJS应用程序定义了初始值。

ng-model指令绑定HTML元素到应用程序数据，也可以：

为应用程序数据提供类型验证（number、email、required）。

为应用程序数据提供状态（invalid、dirty、touched、error）。

为 HTML 元素提供 CSS 类。

绑定 HTML 元素到 HTML 表单。

ng-repeat 指令对于集合中（数组中）的每个项会 克隆一次 HTML 元素。

3.ng-model指令

ng-model指令用于绑定应用程序数据到HTML控制器（input、select、textarea）的值。

1）ng-model指令可以将输入域的值与AngularJS创建的变量绑定。

<div ng-app="myApp" ng-controller="myCtrl">

名字: <input ng-model="name">

</div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

$scope.name = "John Doe";

});

</script>

2）双向绑定：在修改输入域的值时，AngularJS的值也会修改。

<div ng-app="myApp" ng-controller="myCtrl">

名字: <input ng-model="name">

<h1>你输入了: {{name}}</h1>

</div>

3）验证用户输入

<form ng-app="" name="myForm">

Email:

<input type="email" name="myAddress" ng-model="text">

<span ng-show="myForm.myAddress.$error.email">不是一个合法的邮箱地址</span>

</form>

4）应用状态

<form ng-app="" name="myForm" ng-init="myText = 'test@runoob.com'">

Email:

<input type="email" name="myAddress" ng-model="myText" required></p>

<h1>状态</h1>

{{myForm.myAddress.$valid}}

{{myForm.myAddress.$dirty}}

{{myForm.myAddress.$touched}}

</form>

5）CSS类

<style>

input.ng-invalid {

background-color: lightblue;

}

</style>

<body>

<form ng-app="" name="myForm">

输入你的名字:

<input name="myAddress" ng-model="text" required>

</form>

ng-model指令根据表单域的状态添加/移除以下类：

ng-empty

ng-not-empty

ng-touched

ng-untouched

ng-valid

ng-invalid

ng-dirty

ng-pending

ng-pristine

4.AngularJS Scope（作用域）

Scope（作用域）是应用在HTML（视图）和JavaScript（控制器）之间的纽带。它是一个对象，有可访问的方法和属性。它可以应用在视图和控制器上。

当在angularJS创建控制器时，可以将$scope对象当做一个参数传递。

<div ng-app="myApp" ng-controller="myCtrl">

<h1>{{carname}}</h1>

</div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

$scope.carname = "Volvo";

});

</script>

在视图中添加$scope对象时，视图（HTML）可以获取这些属性。视图中不需要添加$scope前缀，只需添加属性名即可，如：{{carname}}。

AngularJS应用组成如下：

view：视图，即HTML

model：模型，是当前视图中可用的数据

controller：控制器，即JavaScript函数，可以添加或修改属性

scope是模型，是一个JavaScript对象，带有属性和方法，这些属性和方法可以在视图和控制器中使用。

如果修改了视图，模型和控制器也会相应的更新。

根作用域：所有应用都有一个$rootScope，它可以作用在ng-app指令包含的所有HTML元素中。

<div ng-app="myApp" ng-controller="myCtrl">

<h1>{{lastname}} 家族成员:</h1>

<ul>

<li ng-repeat="x in names">{{x}} {{lastname}}</li>

</ul>

</div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope, $rootScope) {

$scope.names = ["Emil", "Tobias", "Linus"];

$rootScope.lastname = "Refsnes";

});

</script>

html5新增的表单元素：datalist，keygen，output。datalist规定输入域的选项列表，通过option创建！ keygen提供一种验证用户的可靠方法，密钥对生成器，私钥存于客户端，公钥发到服务器，用于之后验证客户端证书！ output元素用于不同类型的输出！

在HTML中，（ ）可以在网页上通过链接直接打开邮件客户端发送邮件。

<a href=”telnet:ming.zhou@nowcoder.com”>发送邮件</a>

<a href=”mail:ming.zhou@nowcoder.com”>发送邮件</a>

<a href=”mailto:ming.zhou@nowcoder.com”>发送邮件</a>»

<a href=”ftp:ming.zhou@nowcoder.com”>发送邮件</a>

置换元素和不可置换元素：

a) 置换元素：浏览器根据元素的标签和属性，来决定元素的具体显示内容。

例如：浏览器会根据<img>标签的src属性的 值来读取图片信息并显示出来，而如果查看(x)html代码，则看不到图片的实际内容；<input>标签的type属性来决定是显示输入 框，还是单选按钮等。 (x)html中 的<img>、<input>、<textarea>、<select>、<object> 都是置换元素。这些元素往往没有实际的内容，即是一个空元素。

置换元素在其显示中生成了框，这也就是有的内联元素能够设置宽高的原因。

b) 不可替换元素：(x)html 的大多数元素是不可替换元素，即其内容直接表现给用户端（如浏览器）。

例如： <label>label中的内容</label> 标签<label>是一个非置换元素，文字label中的内容”将全被显示。

repaint(重绘) ，repaint发生更改时，元素的外观被改变，且在没有改变布局的情况下发生，如改变outline,visibility,background color，不会影响到dom结构渲染。

reflow(渲染)，与repaint区别就是他会影响到dom的结构渲染，同时他会触发repaint，他会改变他本身与所有父辈元素(祖先)，这种开销是非常昂贵的，导致性能下降是必然的，页面元素越多效果越明显。

所以display:none才会产生reflow，visibility:hidden只会出发repaint。

标准盒子模型 ＝ margin + border + padding + content （content = width | height）

IE盒子模型 ＝ margin + content （content = border + padding + width | height）

现在有两堆石子,小今与小条玩游戏,2个人都足够聪明,两个人规定:每次每人只能从其中一堆中取走1个或2个或3个石子,最后将石子全部取完的人胜利.现在两堆石子的个数为8和9,请问如何安排才能让小今必胜?

让小今先取

让小条先取

没有策略能够让小今必胜

以上说法都不正确

和一些抢数字的游戏有点像，不妨这么想，后手的可以把一轮取走的石子数控制在4。如果小今先从9的堆里面拿走一个，剩下两个8，那他控制着每轮每边（小条拿哪边就拿哪边）拿4个，一定能拿走最后一个。

以下描述正确的：

Http协议所使用的运输层协议是UDP

Https的端口号是443

TCP注重数据安全性，UDP注重数据传输快

传输层提供端到端的可靠报文传递和错误恢复

正确答案: B C D

查询title中包含cv且type是1的记录（）

select \* from where title = 'cv' and type='1'

select \* from where title = '%cv%' and type='1'

select \* from where title = '\*cv' and type='1'

select \* from where title ='\*cv\*' and type='1

SQL提供了四种匹配模式：1. % 表示任意0个或多个字符。如下语句：Select \* FROM user Where name LIKE '%三%'; 将会把name为“张三”，“三脚猫”，“唐三藏”等等有“三”的全找出来。%三：表示左匹配。三%：表示右匹配。

%三%：表示模糊查询。

2. \_ 表示任意单个字符。语句： Select \* FROM user Where name LIKE '\_三\_'；只找出“唐三藏”。这样name为三个字且中间一个字是“三”的； Select \* FROM user Where name LIKE '三\_\_'； 只找出“三脚猫”这样name为三个字且第一个字是“三”的；

3. [ ] 表示括号内所列字符中的一个（类似与正则表达式）。语句：Select \* FROM user Where name LIKE '[张李王]三'; 将找出“张三”、“李三”、“王三”（而不是“张李王三”）； 如 [ ] 内有一系列字符（01234、abcde之类的）则可略写为“0-4”、“a-e“。Select \* FROM user Where name LIKE '老[1-9]'；将找出“老1”、“老2”、……、“老9”；如要找“-”字符请将其放在首位：'张三[-1-9]'；

4. [^ ] 表示不在括号所列之内的单个字符。语句：Select \* FROM user Where name LIKE '[^张李王]三'；将找出不姓“张”、“李”、“王”的“赵三”、“孙三”等；Select \* FROM user Where name LIKE '老[^1-4]'; 将排除“老1”到“老4”寻找“老5”、“老6”、……、“老9”。

5.\* 表示查找的是所有信息,例如select \* from tbl\_user

下列关于操作系统进程与线程的区别正确的是（）

进程是资源分配的基本单位

线程是资源分配的基本单位

进程是资源调度的基本单位

线程是资源调度的基本单位

选AD

排序算法中不稳定的有：希尔排序、堆排序、快速排序、选择排序

写出如下代码弹出的结果:

<script>

var m= 1, j = k = 0;

function add(n) {

return n = n+1;

　 }

y = add(m);

function add(n) {

return n = n + 3;

}

z = add(m);

</script>

y和z的最终结果为（）

js里面没有函数重载的概念，在其他语言中（如java）java中，可以存在同名函数，只要传入的参数数量或者类型不同即可。在js中，定义了两个同名函数后，后面的函数会覆盖前面定义的函数。结合这道题来说，由于函数声明提升，所以函数声明会提前，由于存在同名函数，后面的add函数将覆盖第一个add函数，所以两次调用add()返回的值是相同的。也就是y,z都为4.

页面有一个按钮button id为 button1，通过原生的js如何禁用？

document.getElementById("button1").readolny= true;

document.getElementById("button1").setAttribute(“readolny”,”true”);

document.getElementById("button1").disabled = true;

document.getElementById("button1").setAttribute(“disabled”,”true”);

选C

Readonly只针对input(text/password)和textarea有效，而disabled对于所有的表单元素有效，包括select,radio,checkbox,button等。

js禁用

document.getElementByIdx\_x\_x("btnShow").disabled=true;（这样就把按钮禁用了）

如果要重新开启按钮，则可以让disabled=false;即可

jquery禁用

利用jquery的attr的方法即可

$("#btnShow").attr({"disabled":"disabled"});

如要让按钮恢复可用，可以采用removeAttr方法把disabled属性删除即可。

$("#btnShow").removeAttr("disabled");

页面有一个按钮button id为 button1，通过原生的js 设置背景色为红色？

document.getElementById('button1').style.backgroundColor="red";

document.getElementById('button1').style.backgroundcolor="red";

document.getElementById('button1').style.backGroundColor="red";

document.getElementById('button1').style.bgcolor="red";

选A

将/home/stud1/wang目录做归档压缩，压缩后生成wang.tar.gz 文件，并将此文件保存到/home 目录下，实现此任务的tar 格式为

A.tar zcvf /home/wang.tar.gz /home/stud1/wang

B. tar xcvf /home/stud1/wang /home/wang

C. tar zcv /home/stud1/wang /home/wang

D. tar zcvf /home/stud1/wang /home/wang.tar.gz

选A，option z表示压缩，所以才有后面的.gz， c表示创建这个压缩包，v是可视，能看到其打包和压缩的过程，f表示文件。把A压缩为B文件命令：tar -zcvf B A

下列不是Linux系统进程类型的是 。

交互进程 批处理进程 守护进程 就绪进程

Linux操作系统包括三种不同类型的进程，每种进程都有自己的特点和属性。 1.交互进程——由一个shell启动的进程。交互进程既可以在前台运行，也可以在后台运行。 2.批处理进程——这种进程和终端没有联系，是一个进程序列。 3.监控进程（也称守护进程）——Linux系统启动时启动的进程，并在后台运行。

linux下给文件start.sh设置权限为自己可读可修改可执行,组内用户为可读可执行不可修改,其余用户没有任何权限,那么设置该文件权限的命令为()

chmod start.sh 706

chmod start.sh 750

chmod start.sh 705

chmod start.sh 777

linux中权限有九位二进制数表示XXX YYY ZZZ，前一个XXX表示owner的权限，YYY表示group的权限（也就是同组用户的权限），ZZZ表示other的权限（非同组用户的权限）。根 据本题要求，XXX=111，YYY=101，ZZZ=000，所以是750

下面不属于ajax事件的是？

ajaxComplete(callback)

ajaxSuccess(callback)

$.post(url)

ajaxSend(callback)

$.post(url)是ajax请求；

ajax的事件是：

ajaxComplete(callback)

ajaxError(callback)

ajaxSend(callback)

ajaxStart(callback)

ajaxStop(callback)

ajaxSuccess(callback)

在jquery中，如果想要获取当前窗口的宽度值，下面哪个是实现该功能的？

width()

width(val)

width

innerWidth()

width() 方法设置或返回元素的宽度（不包括内边距、边框或外边距）。

height() 方法设置或返回元素的高度（不包括内边距、边框或外边距）。

innerWidth() 方法返回元素的宽度（包括内边距）。

innerHeight() 方法返回元素的高度（包括内边距）。

outerWidth() 方法返回元素的宽度（包括内边距和边框）。

outerHeight() 方法返回元素的高度（包括内边距和边框）。

**渐变**

[CSS3 Gradient](http://dev.w3.org/csswg/css3-images/)分为[linear-gradient](http://lists.w3.org/Archives/Public/www-style/2011Apr/0083.html)(线性渐变)和[radial-gradient](http://lists.w3.org/Archives/Public/www-style/2011Feb/0537.html)(径向渐变)。先了解一下目前的几种现代浏览器的内核，主流内容主要有[Mozilla](http://www.mozilla.org/)（Gecko）（熟悉的有Firefox，Flock等浏览器）、[WebKit](http://www.webkit.org/)（熟悉的有Safari、Chrome等浏览器）、[Opera](http://www.opera.com/)(presto)（Opera浏览器）、Trident（讨厌的IE浏览器）。

CSS3线性渐变

语法：-moz-linear-gradient( [<point> || <angle>,]? <stop>, <stop> [, <stop>]\* )

参数：其共有三个参数，第一个参数表示线性渐变的方向，top是从上到下、left是从左到右，如果定义成left top，那就是从左上角到右下角。第二个和第三个参数分别是起点颜色和终点颜色。你还可以在它们之间插入更多的参数，表示多种颜色的渐变。

例子：

<div class="example example1"></div>

.example {

width: 150px;

height: 80px;

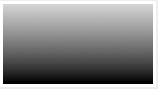
}

添加样式：

.example1 {

background: -moz-linear-gradient( top,#ccc,#000);

}



线性渐变在webkit下的应用：

-webkit-linear-gradient( [<point> || <angle>,]? <stop>, <stop> [, <stop>]\* )//最新发布书写语法

-webkit-gradient(<type>, <point> [, <radius>]?, <point> [, <radius>]? [, <stop>]\*) //老式语法书写规则

参数：-webkit-gradient是webkit引擎对渐变的实现参数，一共有五个。第一个参数表示渐变类型（type），可以是linear（线性渐变）或者radial（径向渐变）。第二个参数和第三个参数，都是一对值，分别表示渐变起点和终点。这对值可以用坐标形式表示，也可以用关键值表示，比如 left top（左上角）和left bottom（左下角）。第四个和第五个参数，分别是两个color-stop函数。color-stop函数接受两个参数，第一个表示渐变的位置，0为起点，0.5为中点，1为结束点；第二个表示该点的颜色。

例子：-webkit-linear-gradient(top,#ccc,#000);

在Mozilla和Webkit下两者的学法都基本上一致，只是其前缀的区别。

线性渐变在Opera下的应用

-o-linear-gradient([<point> || <angle>,]? <stop>, <stop> [, <stop>]);

参数：-o-linear-gradient有三个参数。第一个参数表示线性渐变的方向，top是从上到下、left是从左到右，如果定义成left top，那就是从左上角到右下角。第二个和第三个参数分别是起点颜色和终点颜色。你还可以在它们之间插入更多的参数，表示多种颜色的渐变。

例子：background: -o-linear-gradient(top,#ccc, #000);

从上面的语法中我们可以很清楚的知道，要创建一个线性渐变，我们需要创建一个起点和一个渐变方向（或角度），定义一个起始色：

-moz-linear-gradient( [<point> || <angle>,]? <stop>, <stop> [, <stop>]\* )

-webkit-linear-gradient( [<point> || <angle>,]? <stop>, <stop> [, <stop>]\* )

-o-linear-gradient( [<point> || <angle>,]? <stop>, <stop> [, <stop>]\* )

具体应用如下：

background:-moz-linear-gradient(left,#ace,#f96);/\*Mozilla\*/

background:-webkit-gradient(linear,0 50%,100% 50%,from(#ace),to(#f96));/\*Old gradient for webkit\*/

background:-webkit-linear-gradient(left,#ace,#f96);/\*new gradient for Webkit\*/

background:-o-linear-gradient(left,#ace,#f96); /\*Opera11\*/

工作区：电脑里能看到的目录

版本库：工作区里的隐藏目录.git，是Git的版本库。

Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创

建的第一个分支master，以及指向master的一个指针叫HEAD。

前面讲了我们把文件往Git版本库里添加的时候，是分两步执行的：

第一步是用git add把文件添加进去，实际上就是把文件修改添加到暂存区；

第二步是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支。

因为我们创建Git版本库时，Git自动为我们创建了唯一一个master分支，所以，现在，git commit就是往

master分支上提交更改。

你可以简单理解为，需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改。

Git跟踪并管理的是修改，而不是文件。

撤销修改：

命令git checkout -- readme.txt意思就是，把readme.txt文件在工作区的修改全部撤销，这里有两

种情况 一种是readme.txt自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；

一种是readme.txt已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。

总之，就是让这个文件回到最近一次git commit或git add时的状态。

用命令git reset HEAD file可以把暂存区的修改撤销掉（unstage），重新放回工作区

删除文件：

新建一个文件并提交：

$ git add test.txt

$ git commit -m "add test.txt"

在文件管理器中删除文件：

$ rm test.txt

这个时候，Git知道你删除了文件，因此，工作区和版本库就不一致了，git status命令会立刻告诉

你哪些文件被删除了

现在你有两个选择，一是确实要从版本库中删除该文件，那就用命令git rm删掉，并且git commit：

$ git rm test.txt

$ git commit -m "remove test.txt"

另一种情况是删错了，因为版本库里还有呢，所以可以很轻松地把误删的文件恢复到最新版本：

$ git checkout -- test.txt

远程仓库

在GitHub上免费托管的Git仓库，任何人都可以看到（但只有你自己才能改）。

代码推到远程

由于你的本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以，需要一点设置：

第1步：创建SSH Key。在user/liuyawin下，看看有没有.ssh目录，如果有，再看看这个目

录下有没有id\_rsa和id\_rsa.pub这两个文件，如果已经有了，可直接跳到下一步。如果没有，打开Shell

（Windows下打开Git Bash），创建SSH Key：

$ ssh-keygen -t rsa -C "youremail@example.com"

报错：fatal: remote origin already exists.

解决： 1、先输入$ git remote rm origin

2、再输入$ git remote add origin

git@github.com:djqiang/gitdemo.git

报错：failed to push some refs to 'git@git.oschina.net:summving/Python-Study-

Note---2013.git'

hint: Updates were rejected because the remote contains work that you do

hint: not have locally. This is usually caused by another repository

pushing

hint: to the same ref. You may want to first merge the remote changes

(e.g.,

hint: 'git pull') before pushing again.

hint: See the 'Note about fast-forwards' in 'git push --help' for

details.

解决：因为远程仓库比你的本地库更新，所以你得先pull下，再push。没有设定远程跟踪分

支的话 你要使用这个命令 git pull origin master。

最后：$ git push -u origin master

从远程克隆

要克隆一个仓库，首先必须知道仓库的地址，然后使用git clone命令克隆。

Git支持多种协议，包括https，但通过ssh支持的原生git协议速度最快。

$ git clone git@github.com:michaelliao/gitskills.git

分支管理 分支在实际中有什么用呢？假设你准备开发一个新功能，但是需要两周才能完成，第一周你

写了50%的代码，如果立刻提交，由于代码还没写完，不完整的代码库会导致别人不能干活了。如果等代码全

部写完再一次提交，又存在丢失每天进度的巨大风险。

现在有了分支，就不用怕了。你创建了一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作

，而你在自己的分支上干活，想提交就提交，直到开发完毕后，再一次性合并到原来的分支上，这样，既安全

，又不影响别人工作。

1.创建dev分支：$ git checkout -b dev

git checkout命令加上-b参数表示创建并切换，相当于以下两条命令：

$ git branch dev

$ git checkout dev

2.然后可以用git branch命令来查看分支，git branch命令会列出所有分支，当前分支前面会标一个

\*号。

3.在dev分支上正常提交

$ git add readme.txt

$ git commit -m "branch test"

4.现在切换回master分支：$ git checkout master，并查看刚刚修改的文件，发现并没有变化。因

为那个提交是在dev分支上，而master分支此刻的提交点并没有变。

5.把dev分支的工作成果合并到master分支：$ git merge dev

6.完成后删除dev分支：$ git branch -d dev

Git鼓励大量使用分支：

查看分支：git branch

创建分支：git branch <name>

切换分支：git checkout <name>

创建+切换分支：git checkout -b <name>

合并某分支到当前分支：git merge <name>

删除分支：git branch -d <name>

冲突

新建分支feature1，修改readme.txt并提交；切换到master分支，再次修改readme.txt并提交。这种

情况下，Git无法执行“快速合并”，只能试图把各自的修改合并起来，但这种合并就可能会有冲突。用git

log --graph命令可以看到分支合并图：$ git log --graph --pretty=oneline --abbrev-commit

分支策略

通常，合并分支时，如果可能，Git会用Fast forward模式，但这种模式下，删除分支后，会丢掉分

支信息。

如果要强制禁用Fast forward模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就

可以看出分支信息。

$ git merge --no-ff -m "merge with no-ff" dev

之后可以查看分支历史：$ git log --graph --pretty=oneline --abbrev-commit

在实际开发中，我们应该按照几个基本原则进行分支管理：

首先，master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；

那在哪干活呢？干活都在dev分支上，也就是说，dev分支是不稳定的，到某个时候，比如1.0版本发

布时，再把dev分支合并到master上，在master分支发布1.0版本；

你和你的小伙伴们每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并就

可以了。

合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合

并，而fast forward合并就看不出来曾经做过合并。

bug分支：

当你接到一个修复一个代号101的bug的任务时，很自然地，你想创建一个分支issue-101来修复它，

但是当前正在dev上进行的工作还没有提交，可以使用stash功能将当前工作现场“储藏”起来，等以后恢复现

场后继续工作。

储存工作现场：$ git stash

切换到master分支：$ git checkout master

创建并切换到修复错误分支：$ git checkout -b issue-101

修正bug并提交：add，commit

切换回master分支：$ git checkout master

将bug修复分支合并到master分支：$ git merge --no-ff -m "merged bug fix 101" issue-101

删除bug修复分支：$ git branch -d issue-101

切换回dev分支：$ git checkout dev

查看刚才的工作现场：$ git stash list

恢复现场并删除stash内容：$git stash pop

如果要丢弃一个没有被合并过的分支，可以通过git branch -D <name>强行删除。

多人协作

当你从远程仓库克隆时，实际上是把本地的master分支和远程master分支对应起来了，并且远程仓库

的默认名称是origin。可以用$ git remote查看远程库的信息。用$ git remote -v查看更详细的信息，查看

到的都是可以抓取的推送的origin地址。若没权限推送，就看不到push地址。

推送分支：$ git push origin master

如果要推送其他分支，比如dev，就改成：$ git push origin dev

标签管理：

发布一个版本时，我们通常先在版本库中打一个标签，这样，就唯一确定了打标签时刻的版本。将来

无论什么时候，取某个标签的版本，就是把那个打标签的时刻的历史版本取出来。所以，标签也是版本库的一

个快照。

Git的标签虽然是版本库的快照，但其实它就是指向某个commit的指针（，所以，创建和删除标签都

是瞬间完成的。

提交时的报错：

提交命令： git remote add master https://github.com/liuyawin/require.git

git push -u origin master

Pull is not possible because you have unmerged files.

解决：手工编辑好有conflict的文件，再commit。一个一个文件提交

cannot do a partial commit during a merge.

解决：不能部分提交，$ git commit -a

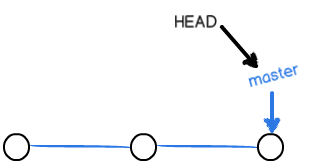
failed to push some refs to 'https://github.com/liuyawin/require.git'

解决：出现错误的主要原因是github中的README.md文件不在本地代码目录中。可以通过如下命令进

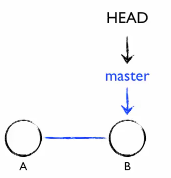
行代码合并[注：pull=fetch+merge] git pull --rebase origin master

在版本回退里，你已经知道，每次提交，Git都把它们串成一条时间线，这条时间线就是一个分支。截止到目前，只有一条时间线，在Git里，这个分支叫主分支，即master分支。HEAD严格来说不是指向提交，而是指向master，master才是指向提交的，所以，HEAD指向的就是当前分支。

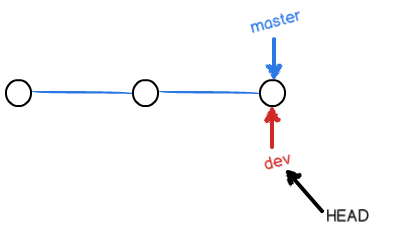
一开始的时候，master分支是一条线，Git用master指向最新的提交，再用HEAD指向master，就能确定当前分支，以及当前分支的提交点：



每次提交，master分支都会向前移动一步，这样，随着你不断提交，master分支的线也越来越长：

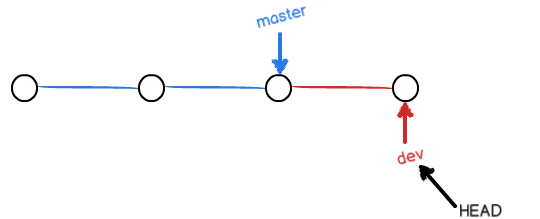


当我们创建新的分支，例如dev时，Git新建了一个指针叫dev，指向master相同的提交，再把HEAD指向dev，就表示当前分支在dev上：

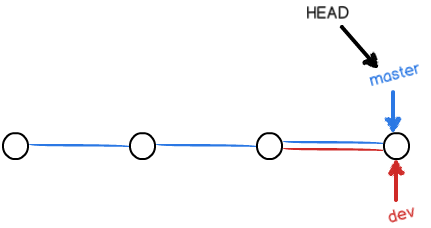


你看，Git创建一个分支很快，因为除了增加一个dev指针，改改HEAD的指向，工作区的文件都没有任何变化！

不过，从现在开始，对工作区的修改和提交就是针对dev分支了，比如新提交一次后，dev指针往前移动一步，而master指针不变：

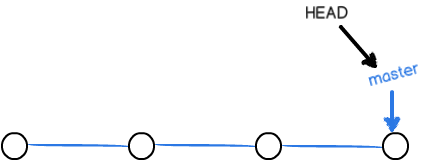


假如我们在dev上的工作完成了，就可以把dev合并到master上。Git怎么合并呢？最简单的方法，就是直接把master指向dev的当前提交，就完成了合并：

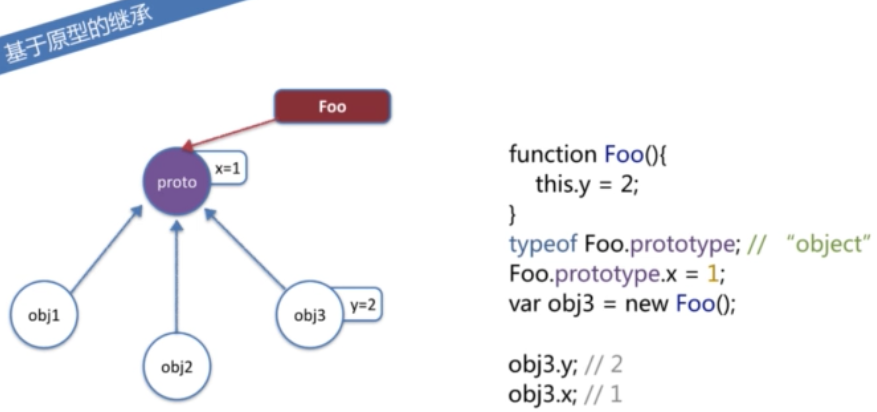


所以Git合并分支也很快！就改改指针，工作区内容也不变！

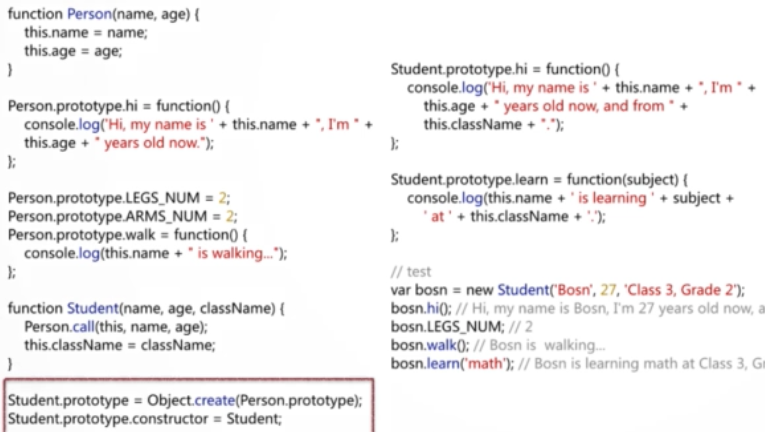
合并完分支后，甚至可以删除dev分支。删除dev分支就是把dev指针给删掉，删掉后，我们就剩下了一条master分支：



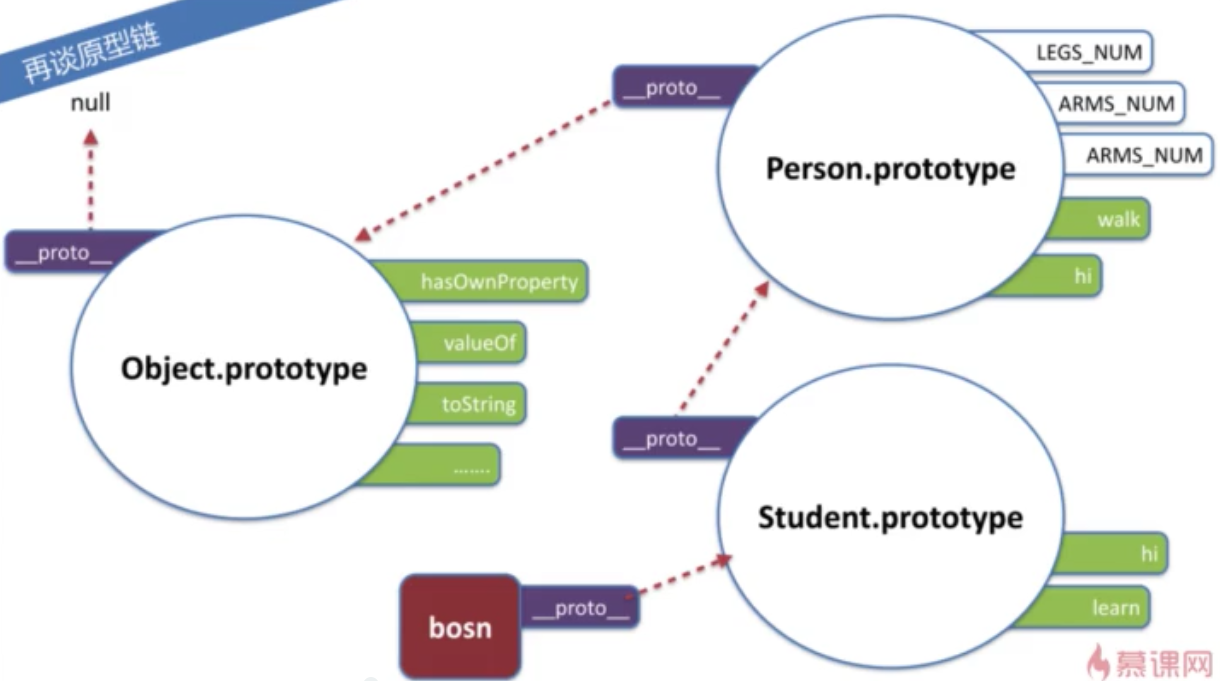
基于原型的继承：



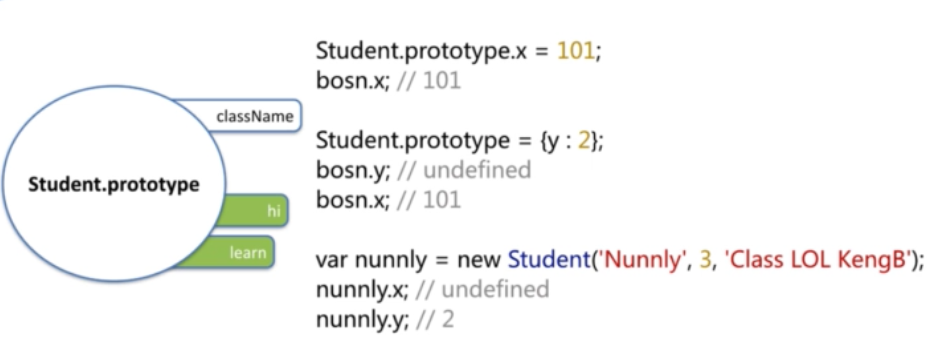


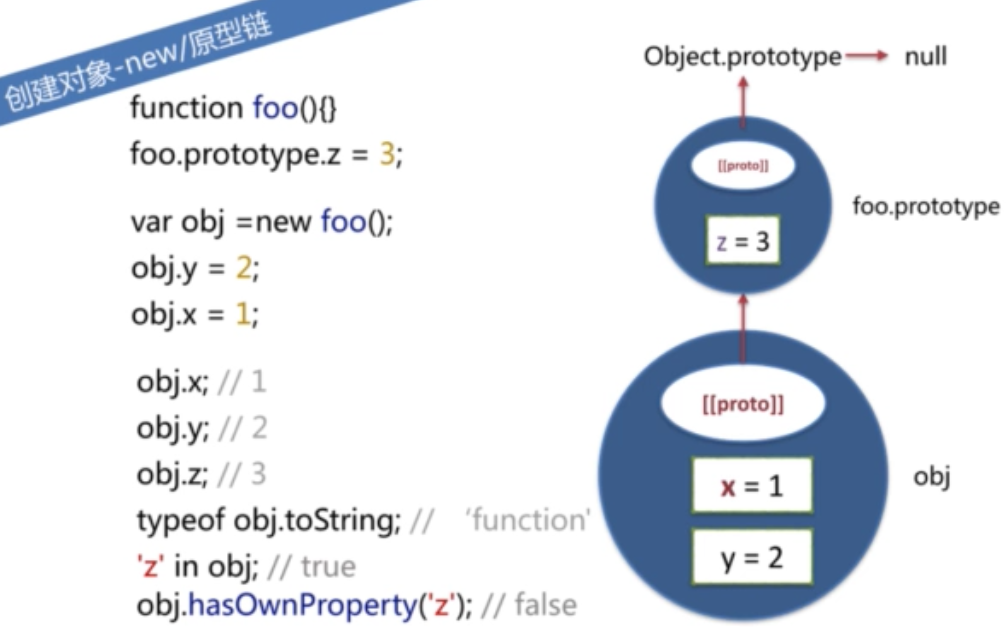


对象bosn的原型指向构造器的prototype属性Student.prototype。Student.prototype实际上是一个空对象，这个空对象的原型指向Person.prototype，随后又给这个空对象添加了hi方法和learn方法。Person.prototype是一个普通的内置对象，它的原型是Oblect.prototype。



修改prototype的属性会影响所有已创建和未创建的实例；而直接修改prototype不会影响已创建的实例，只会影响未创建的实例。





instanceof：左边是一个对象，右边是一个构造器或函数。instanceof会判断右边的构造器的prototype属性是否出现在左边对象的原型链上。

[1,2] instanceof Array === true;

new Object() instanceof Array === false;

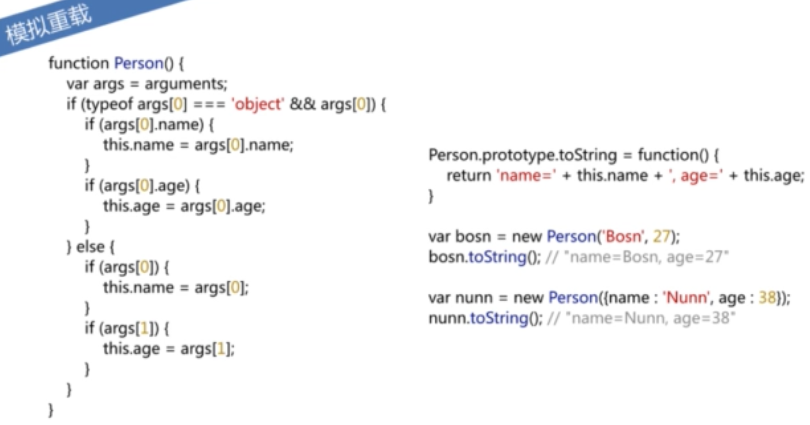
实现继承的三种方式：

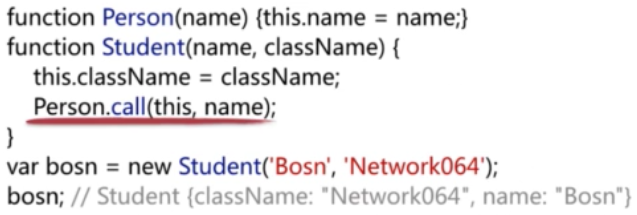
Student.prototype = Person.prototype;//不可取，因为这样改变Student的属性也会改变Person

Student.prototype = new Person();//

Student.prototype = Object.creat(Person.prototype);//ES5之后才支持

Student.prototype.constructor = Person;





链式调用：



1.函数封装

JavaScript的作用域是基于函数的，所以可以把函数作为模块化的第一步，然后在需要的时候加载函数的文件，调用函数就可以了。

function fn1(){

statement

}

function fn2(){

statement

}

缺点：污染了全局的变量，无法保证不与其他模块发生变量名冲突，而且模块成员之间没什么关系。

2.对象

将所有模块化成员封装在对象中，在需要调用模块的时候引用对应的文件。

var myModule = {

var1: 1,

var2: 2,

fn1: function(){

},

fn2: function(){

}

}

这样避免了全局污染，只要保证模块名唯一即可，同时同一模块内的成员也有了关系。但是，外部可以随意修改内部成员，这样会产生意外的安全问题。

3.立即调用函数

可以通过立即调用函数来隐藏细节。

var myModule = (function(){

var var1 = 1;

var var2 = 2;

function fn1(){

}

function fn2(){

}

return {

fn1: fn1,

fn2: fn2

};

})();

4.放大模式

如果一个模块很大，需要分成几个部分，或者一个模块要继承另一个模块，这时就需要使用放大模式：

var module1 = (function (mod){

　　　　mod.m3 = function () {

　　　　　　//...

　　　　};

　　　　return mod;

　　})(module1);

上面的代码为module1添加了一个新方法，并返回新的module1模块。

5.宽放大模式

在浏览器环境中，模块的各个部分通常都是从网上获取的，有时无法知道哪个模块会先加载，采用如上的写法，第一个执行的部分有可能加载一个不存在的对象，这时就要采用“宽放大模式”

var module1 = ( function (mod){

　　　　//...开门，

　　　　return mod;

　　})(window.module1 || {});

与"放大模式"相比，＂宽放大模式＂就是"立即执行函数"的参数可以是空对象。

6.输入全局变量

独立性是模块的重要特点，模块内部最好不与程序的其他部分直接交互。

为了在模块内部调用全局变量，必须显示的将其他变量输入模块。

var module1 = (function ($, YAHOO) {

　　　　//...

　　})(jQuery, YAHOO);

CommonJS中，有一个全局性方法require()，用于加载模块。假设有一个数学模块matn.js，可以这样加载：var math = require('math');然后就可以调用模块提供的方法：math.add(2,3);

但是这种方法在浏览器中运行时会有问题，就是第二行必须在第一行的math.js加载完成之后才能运行。也就是说，如果加载时间很长，整个应用就会停在那里。

这对服务器端不是一个问题，因为所有的模块都存放在本地硬盘，可以同步加载完成，等待时间就是硬盘的读取时间。但是，对于浏览器，这却是一个大问题，因为模块都放在服务器端，等待时间取决于网速的快慢，可能要等很长时间，浏览器处于"假死"状态。

浏览器中的模块不能使用同步加载，而应该使用异步加载。这就是AMD产生的背景。

AMD：异步模块定义。采用异步方式加载模块，模块加载不影响后面语句的运行。所有依赖于这个模块的语句，都定义在一个回调函数中，等到加载完成后，这个回调函数才会运行。

AMD也采用require()函数加载模块，但是不同于CommonJS，它需要两个参数：require([module],callback);

目前，主要有两个JavaScript库实现了AMD规范：require.js和curl.js。

原始的JS加载方法： <script src="1.js"></script>

　　<script src="2.js"></script>

　　<script src="3.js"></script>

　　<script src="4.js"></script>

　　<script src="5.js"></script>

　　<script src="6.js"></script>

这样写的缺点 1.加载的时候浏览器会停止渲染，加载文件越多，网页失去响应的时间就会越长；2.由于js之间存在依赖关系，因此必须保证加载顺序，依赖性最大的一定要放在最后加载，当依赖关系很复杂时，代码的编写和维护都会变得很困难。

第一步：下载并引入require.js

<script src="js/require.js" defer async="true" ></script>

async属性表明这个文件需要异步加载，避免网页失去响应。IE不支持这个属性，只支持defer，所以把defer也写上。

第二步：加载自己的代码。假定我们自己的代码文件是main.js，也放在js目录下面。那么，只需要写成下面这样就行了：

<script src="js/require.js" data-main="js/main"></script>

data-main属性的作用是，指定网页程序的主模块。在上例中，就是js目录下面的main.js，这个文件会第一个被require.js加载。由于require.js默认的文件后缀名是js，所以可以把main.js简写成main。

主模块的写法

主模块是整个网页的入口代码，相当于C语言的main函数，所有代码都从这儿开始运行。

常见的情况是：主模块依赖于其他模块，这时就要使用AMD规范定义的require()函数：

// main.js

　　require(['moduleA', 'moduleB', 'moduleC'], function (moduleA, moduleB, moduleC){

　　　　// some code here

　　});

require()函数接受两个参数。第一个参数是一个数组，表示所依赖的模块，上例就是['moduleA', 'moduleB', 'moduleC']，即主模块依赖这三个模块；第二个参数是一个回调函数，当前面指定的模块都加载成功后，它将被调用。加载的模块会以参数形式传入该函数，从而在回调函数内部就可以使用这些模块。

require()异步加载moduleA，moduleB和moduleC，浏览器不会失去响应；它指定的回调函数，只有前面的模块都加载成功后，才会运行，解决了依赖性的问题。

假定主模块依赖jquery、underscore和backbone这三个模块，main.js就可以这样写：

　　require(['jquery', 'underscore', 'backbone'], function ($, \_, Backbone){

　　　　// some code here

　　});

require.js会先加载jQuery、underscore和backbone，然后再运行回调函数。主模块的代码就写在回调函数中。

模块的加载

上一节最后的示例中，主模块的依赖模块是['jquery', 'underscore', 'backbone']。默认情况下，require.js假定这三个模块与main.js在同一个目录，文件名分别为jquery.js，underscore.js和backbone.js，然后自动加载。

使用require.config()方法，我们可以对模块的加载行为进行自定义。require.config()就写在主模块（main.js）的头部。参数就是一个对象，这个对象的paths属性指定各个模块的加载路径。

　　require.config({

　　　　paths: {

　　　　　　"jquery": "jquery.min",

　　　　　　"underscore": "underscore.min",

　　　　　　"backbone": "backbone.min"

　　　　}

　　});

上面的代码给出了三个模块的文件名，路径默认与main.js在同一个目录（js子目录）。如果这些模块在其他目录，比如js/lib目录，则有两种写法。一种是逐一指定路径。

　　require.config({

　　　　paths: {

　　　　　　"jquery": "lib/jquery.min",

　　　　　　"underscore": "lib/underscore.min",

　　　　　　"backbone": "lib/backbone.min"

　　　　}

　　});

另一种则是直接改变基目录（baseUrl）。

　　require.config({

　　　　baseUrl: "js/lib",

　　　　paths: {

　　　　　　"jquery": "jquery.min",

　　　　　　"underscore": "underscore.min",

　　　　　　"backbone": "backbone.min"

　　　　}

　　});

如果某个模块在另一台主机上，也可以直接指定它的网址，比如：

　　require.config({

　　　　paths: {

　　　　　　"jquery": "https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min"

　　　　}

　　});

require.js要求，每个模块是一个单独的js文件。这样的话，如果加载多个模块，就会发出多次HTTP请求，会影响网页的加载速度。因此，require.js提供了一个优化工具，当模块部署完毕以后，可以用这个工具将多个模块合并在一个文件中，减少HTTP请求数。

AMD模块的写法：

require.js加载的模块，采用AMD规范。也就是说，模块必须按照AMD的规定来写。

具体来说，就是模块必须采用特定的define()函数来定义。如果一个模块不依赖其他模块，那么可以直接定义在define()函数之中。

假定现在有一个math.js文件，它定义了一个math模块。那么，math.js就要这样写：

　　// math.js

　　define(function (){

　　　　var add = function (x,y){

　　　　　　return x+y;

　　　　};

　　　　return {

　　　　　　add: add

　　　　};

　　});

加载方法如下：

　　// main.js

　　require(['math'], function (math){

　　　　alert(math.add(1,1));

　　});

如果这个模块还依赖其他模块，那么define()函数的第一个参数，必须是一个数组，指明该模块的依赖性。

　　define(['myLib'], function(myLib){

　　　　function foo(){

　　　　　　myLib.doSomething();

　　　　}

　　　　return {

　　　　　　foo : foo

　　　　};

　　});

当require()函数加载上面这个模块的时候，就会先加载myLib.js文件。

加载非规范化的模块

理论上，require.js加载的模块，必须是按照AMD规范、用define()函数定义的模块。但是实际上，虽然已经有一部分流行的函数库（比如jQuery）符合AMD规范，更多的库并不符合。那么，require.js是否能够加载非规范的模块呢？

回答是可以的。

这样的模块在用require()加载之前，要先用require.config()方法，定义它们的一些特征。

举例来说，underscore和backbone这两个库，都没有采用AMD规范编写。如果要加载它们的话，必须先定义它们的特征。

　　require.config({

　　　　shim: {

　　　　　　'underscore':{

　　　　　　　　exports: '\_'

　　　　　　},

　　　　　　'backbone': {

　　　　　　　　deps: ['underscore', 'jquery'],

　　　　　　　　exports: 'Backbone'

　　　　　　}

　　　　}

　　});

require.config()接受一个配置对象，这个对象除了有前面说过的paths属性之外，还有一个shim属性，专门用来配置不兼容的模块。具体来说，每个模块要定义（1）exports值（输出的变量名），表明这个模块外部调用时的名称；（2）deps数组，表明该模块的依赖性。

比如，jQuery的插件可以这样定义：

　　shim: {

　　　　'jquery.scroll': {

　　　　　　deps: ['jquery'],

　　　　　　exports: 'jQuery.fn.scroll'

　　　　}

　　}

require.js插件

require.js还提供一系列插件，实现一些特定的功能。

domready插件，可以让回调函数在页面DOM结构加载完成后再运行。

　　require(['domready!'], function (doc){

　　　　// called once the DOM is ready

　　});

text和image插件，则是允许require.js加载文本和图片文件。

　　define([

　　　　'text!review.txt',

　　　　'image!cat.jpg'

　　　　],

　　　　function(review,cat){

　　　　　　console.log(review);

　　　　　　document.body.appendChild(cat);

　　　　}

　　);

类似的插件还有json和mdown，用于加载json文件和markdown文件。

函数可以定义，也可以调用。在js中，函数不仅是一种语法，也是一种值，也就是说，可以将函数赋值给变量，存储在对象的属性或是数组的元素中，作为参数传给另一个函数等。

登录：mysql -uroot -ppassword

若提示mysql不是内部命令，则配置环境变量，将mysql/bin的路径添加到环境变量的PATH下。控制台要以管理员身份运行。

创建数据库：create database *dbname*;  
**mysql权限系统**

最小权限原则：一个用户（或者一个进程）应当拥有能够执行分配给它的任务的最低级别的权限。

GRANT和REVOKE命令分别用于授权和取消MySQL用户的权限，这些权限分四个级别：全局、数据库、表、列。

GRANT命令的常见形式：

GRANT *privileges* [*columns*]

ON *item*

TO *user\_name* [IDENTIFIED BY '*password*'

[REQUIRE *ssl*\_*options*]

[WITH [GRANT OPTION] | *limit\_options*]

中括号中的子句是可选的。

REVOKE命令的常见形式：

REVOKE *privileges* [ (*columns)* ]

ON *item*

FROM user\_name

若已经给出了WITH [GRANT OPTION]子句，则按如下方式撤销：

REVOKE ALL PRIVILEGES,GRANT

FROM user\_name

使用grant和revoke的一个例子：

mysql>grant all

->on \*

->to fred identified by '123456'

->with grant option;

这句话授予用户名为fred密码为123456的用户使用数据库的所有权限，并允许他向其他人授予这些权限。

mysql>revoke all privileges, grant

->from fred;

这句话取消了fred的所有权限。

报错：The MySQL server is running with the --skip-grant-tables option so it cannot execute this statement

解决方法：

先刷新一下权限表。

mysql> flush privileges;

Query OK, 0 rows affected (0.01 sec)

mysql> grant all on cactidb.\* to dbuser@'localhost' identified by '123';

Query OK, 0 rows affected (0.00 sec)

退出MySQL：exit，回车

指定使用的数据库：use database

创建数据库表：create table *tablename* （*columns*）

create table tutorials\_tbl(

tutorial\_id INT NOT NULL AUTO\_INCREMENT,

tutorial\_title VARCHAR(100) NOT NULL,

tutorial\_author VARCHAR(40) NOT NULL,

submission\_date DATE,

PRIMARY KEY ( tutorial\_id )

);

运行sql脚本：[Mysql的bin目录]\mysql –u用户名 –p密码 –D数据库<[sql脚本文件路径全名]，示例：D:\mysql\bin\mysql –uroot –p123456 -Dtest<d:\test\ss.sql

查看数据库中所有表：show tables;

查看特定表的详细信息：describe *表名*;

创建外键：语法:alter table 表名 add constraint FK\_ID foreign key(你的外键字段名) REFERENCES 外表表名(对应的表的主键字段名);

例: alter table tb\_active add constraint FK\_ID foreign key(user\_id) REFERENCES tb\_user(id)

SQL：关系数据库管理系统。

SQL包括DDL（用于定义数据库的数据定义语言）和DML（用于查询数据库的数据操作语言）两部分。

**在数据库中插入数据**：

insert into customers (name,city) values ('Tom','New York');

insert into customers set name='Liyang',address='Beijing';

**从数据库中获取数据**：

SELECT [options] items

[into file\_details]

FROM tables

[WHERE conditions]

[GROUP BY group\_type]

[ HAVING where\_definition]

[ORDER BY order\_type]

[LIMIT limit\_criteria]

[PROCEDURE proc\_name(arguments)]

[lock\_options]

;

where子句的比较运算符

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 运算符 | 名称 | | 举例 | 描述 | | | | |
| =，>,<,>=,<= | 等于... | stuId=1... | | 测试两个值是否相等... | | | | |
| !=或<> | 不等于 | stuId!=1 | | 测试两个值是否不等 | | | | |
| is not null | 不为空 | address is not null | | | 测试一个字段是否包含一个值 | | | |
| is null | 为空 | address is null | | | | 测试一个字段是否不包含一个值 | | |
| between and | 介于 | ID between 1 and 3 | | | | | | 测试一个数是否介于两者之间 |
| in |  | city in ('SX','BJ') | | | | | | 测试一个值是否在特定集合里 |
| Not in |  | City not in ('SX','BJ') | | | | 测试一个值是否不在特定集合里 | | |
| Like | 模式匹配 | Name like ('Jim%') | | | | | 检查一个值是否匹配于一个模式 | |
| Not like | 模式匹配 | Name not like ('Jim%') | | | | 检查一个值是否不匹配于一个模式 | | |
| Regexp | 常规表达式 | | Name regexp | | | 检查一个值是否匹配一个常规表达式 | | | |

双表关联、多表关联：

select student.stuName from student,user where student.stuId = user.stuId;

查找不匹配行（左关联）：

select \* from student left join clazz on student.clazzId = clazz.clazzId;

使用表的别名：

select s.name from student as s,c.clazzName from clazz as c where c.clazzId = s.clazzId;

以特定顺序获取数据：order by（asc：升序；desc：降序）

select \* from student order by name;

分组与合计数据（group by）

选择要返回的行：

select stuName from student limit 2 3;

两个参数2,3分别表示开始的行号和行数。这句话表示返回3行，从返回结果的第2行开始。

子查询：

select \* from student where clazzId=(select max(clazzId) from clazz);

**子查询操作符**

|  |  |
| --- | --- |
| **名称** | **示例语法** |
| ANY | Select c1 from t1 where c1>ANY(select c1 from t2) |
| IN | Select c1 from t1 where c1>IN(select c1 from t2) |
| SOME | Select c1 from t1 where c1>SOME(select c1 from t2) |
| ALL | Select c1 from t1 where c1>ALL(select c1 from t2) |

关联子查询：select stuName from student where not exists (select \* from clazz where clazzId=3);

行子查询：select c1,c2 from t1 where (c1,c2) in (select c1,c2 from t2);

使用子查询作为临时表：

select \* from (select stuName,stuId from student where clazzId=1) as students;

**更新数据库记录**：update。

**创建后修改表**：

alter table student modify stuName varchar(50) not null;

alter table student add stuGender bool after stuNum;

alter table student drop stuGender;

删除数据表中的数据：

delete from student;删除student表中的所有数据

delete from student where stuId =1;

删除表：drop table student;

删除数据库：drop database schoolsystem003;

排序：order by + ASC 或 DESC按升序或降序

分组：SELECT name, COUNT(\*) FROM employee\_tbl GROUP BY name;

正则表达式：



查找name字段中以'st'为开头的所有数据：

mysql> SELECT name FROM person\_tbl WHERE name REGEXP '^st';

查找name字段中以'ok'为结尾的所有数据：

mysql> SELECT name FROM person\_tbl WHERE name REGEXP 'ok$';

查找name字段中包含'mar'字符串的所有数据：

mysql> SELECT name FROM person\_tbl WHERE name REGEXP 'mar';

查找name字段中以元音字符开头且以'ok'字符串结尾的所有数据：

mysql> SELECT name FROM person\_tbl WHERE name REGEXP '^[aeiou]|ok$';

**深入理解权限系统**

以管理员身份登陆后，输入如下命令：use mysql; show tables;可以查看数据库中的表。每个表都存储了关于系统的信息。保存权限信息的6个表分别是：user、host、db、tables\_priv、columns\_priv、procs\_priv。这些表称为授权表。

**user表**

该表包含全局用户权限的详细信息。它可以确定一个用户是否有权限连接MySQL数据库，是否具有全局级别的权限，即是否有使用系统中每个数据库的权限。

可以通过describe user;语句来浏览该表的结构。

**db表和host表**

普通用户的权限大都保存在db表和host表中。

db表可以确定哪些用户可以访问哪些主机和哪些数据库。此表中给出的权限应用于一个特定行中指定的任意数据库。

host表是db表的补充。如果一个用户从多个主机连接到数据库，在db表中将不会列出该用户的主机名。而与此项相对应，该用户将在host表中获得一组记录，每个记录都指定了每个用户-主机对的权限。

**tables\_priv表，columns\_priv表和procs\_priv表**

这三个表分别储存表级别的权限、列级别的权限以及针对存储过程的权限。

**事务管理**

1.在MySQL中只有使用了Innodb数据库引擎的数据库或表才支持事务

2.事务处理可以用来维护数据库的完整性，保证成批的SQL语句要么全部执行，要么全部不执行

3.事务用来管理insert,update,delete语句

一般来说，事务是必须满足4个条件（ACID）： Atomicity（原子性）、Consistency（稳定性）、Isolation（隔离性）、Durability（可靠性）

1、事务的原子性：一组事务，要么成功；要么撤回。

2、稳定性 ： 有非法数据（外键约束之类），事务撤回。

3、隔离性：事务独立运行。一个事务处理后的结果，影响了其他事务，那么其他事务会撤回。事务的100%隔离，需要牺牲速度。

4、可靠性：软、硬件崩溃后，InnoDB数据表驱动会利用日志文件重构修改。可靠性和高速度不可兼得， innodb\_flush\_log\_at\_trx\_commit选项 决定什么时候吧事务保存到日志里。

**MySQL索引**

索引分为单列索引和组合索引，组合索引是指一个索引包含多个列。

创建索引时，应该保证该索引是应用在sql语句中的条件（一般作为where子句条件）。实际上索引也是一张表，该表保存了主键与索引字段，并指向实体表的记录。

优点：可以大大提高MySQL的检索速度。

缺点：降低了表的更新速度，如insert、update和delete。因为进行这些操作不仅要保存数据，还要保存索引文件；建立的索引文件会占用磁盘空间。

普通索引

创建索引：CREATE INDEX indexName ON mytable(username(length));

如果是CHAR，VARCHAR类型，length可以小于字段实际长度；如果是BLOB和TEXT类型，必须指定 length。

修改表结构：

ALTER mytable ADD INDEX [indexName] ON (username(length))

也可以在创建表的时候指定：

CREATE TABLE mytable(

ID INT NOT NULL,

username VARCHAR(16) NOT NULL,

INDEX [indexName] (username(length))

);

删除索引：DROP INDEX [indexName] ON mytable;

唯一索引

创建索引：CREATE UNIQUE INDEX indexName ON mytable(username(length))

修改表结构：ALTER mytable ADD UNIQUE [indexName] ON (username(length))

也可以在创建表的时候直接指定：

CREATE TABLE mytable(

ID INT NOT NULL,

username VARCHAR(16) NOT NULL,

UNIQUE [indexName] (username(length))

);

显示索引信息：SHOW INDEX FROM table\_name\G

**MySQL处理重复数据**

1.统计重复数据，统计表中 first\_name和last\_name的重复记录数：

mysql> SELECT COUNT(\*) as repetitions, last\_name, first\_name

-> FROM person\_tbl

-> GROUP BY last\_name, first\_name

-> HAVING repetitions > 1;

2.过滤重复数据：如果你需要读取不重复的数据可以在 SELECT 语句中使用 DISTINCT 关键字来过滤重复数据。

mysql> SELECT DISTINCT last\_name, first\_name

-> FROM person\_tbl

-> ORDER BY last\_name;

或者：

mysql> SELECT last\_name, first\_name

-> FROM person\_tbl

-> GROUP BY (last\_name, first\_name);

3.删除重复数据：

mysql> CREATE TABLE tmp SELECT last\_name, first\_name, sex

-> FROM person\_tbl;

-> GROUP BY (last\_name, first\_name);

mysql> DROP TABLE person\_tbl;

mysql> ALTER TABLE tmp RENAME TO person\_tbl;

也可以在数据表中添加 INDEX（索引） 和 PRIMAY KEY（主键）这种简单的方法来删除 表中的重复记录。方法如下：

mysql> ALTER IGNORE TABLE person\_tbl

-> ADD PRIMARY KEY (last\_name, first\_name);

**SQL注入**

SQL注入是指通过把SQL命令插入到web表单提交或输入域名或页面请求的查询字符串，最终欺骗服务器执行恶意的SQL命令。

防止SQL注入，需要注意以下几点：

1.永远不要信任用户的输入。对用户的输入进行校验，可以通过正则表达式，或限制长度；对单引号和 双"-"进行转换等。

2.永远不要使用动态拼装sql，可以使用参数化的sql或者直接使用存储过程进行数据查询存取。

3.永远不要使用管理员权限的数据库连接，为每个应用使用单独的权限有限的数据库连接。

4.不要把机密信息直接存放，加密或者hash掉密码和敏感的信息。

5.应用的异常信息应该给出尽可能少的提示，最好使用自定义的错误信息对原始错误信息进行包装

6.sql注入的检测方法一般采取辅助软件或网站平台来检测，软件一般采用sql注入检测工具jsky，网站平台就有亿思网站安全平台检测工具。MDCSOFT SCAN等。采用MDCSOFT-IPS可以有效的防御SQL注入，XSS攻击等。

**导出数据**

使用 SELECT ... INTO OUTFILE 语句导出数据

mysql> SELECT \* FROM runoob\_tbl

-> INTO OUTFILE '/tmp/tutorials.txt';

用逗号隔开：

SELECT a,b,a+b INTO OUTFILE '/tmp/result.text'

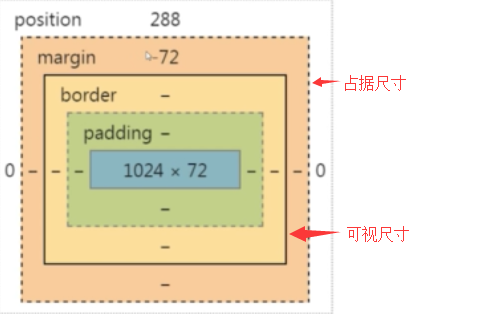
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'

LINES TERMINATED BY '\n'

FROM test\_table;

margin可以改变容器的可视尺寸，这一点适用于没有设定width/height的普通block元素，不适用于float元素，absolute/fixed元素，inline水平，table-cell元素。只适用于水平方向尺寸。利用这一特性可以实现一边固定，一遍自适应布局。

margin与占据尺寸：block/inline-block水平元素均适用；与有没有设定width/height无关；适用于水平方向和垂直方向。



普通元素的百分比margin都是相对于容器的宽度计算的（水平和垂直都是）。

绝对定位元素的百分比margin是相对于第一个定位祖先元素（relative/absolute/fixed）的宽度计算的。

margin重叠

1.只发生在block水平元素（不包括float和absolute元素）

2.不考虑writing-mode，只发生在垂直方向（margin-top，margin-bottom）

3种情形：相邻的兄弟元素；父级和第一个/最后一个子元素；空的block元素（和自己）

对第二种情况：

<div class="father">

<div class="son" style="margin-top: 80px"></div>

</div>

<div class="father" style="margin-top: 80px">

<div class="son"></div>

</div>

<div class="father" style="margin-top: 80px">

<div class="son" style="margin-top: 80px"></div>

</div>

以上三种写法其效果是一样的。

margin-top重叠：1.父元素为非块状格式化上下文元素；2.父元素没有border-top设置； 3.父元素没有padding-top设置；4.父元素和第一个子元素之间没有内联元素（inline） 分隔。

margin-bottom重叠：1.父元素为非块状格式化上下文元素；2.父元素没有border-top设 置； 3.父元素没有padding-top设置；4.父元素和第一个子元素之间没有内联元素 （inline）分隔；5.父元素没有height，min-height，max-height限制

对第三种情况：1.元素没有border设置；2.元素没有padding值；3.里面没有inline元素； 4.没有height或者min-height

重叠规则：正正取正值，正负值相加，负负取负值。

margin重叠的目的：1.连续段落或者列表之类，若果，没有margin重叠，首尾项的间距会和其他兄弟标签成2:1关系，排版不自然；2.web中任何地方嵌套或直接放入任何裸div，都不会影响原来的布局；3.遗落的空任意多个<p>元素，不要影响原来的阅读排版。

margin auto

元素有的时候就算没有设置width和height也会自动填充相应的方位。

左右margin设为auto时元素居中必须要保证元素的display:block;并且保证margin值不为负即子元素宽度小于父元素。上下margin设为auto不能垂直居中。

修改writing-mode:vertical-lr;可以改变流方式为垂直布局，此时左右margin设为auto时元素不居中，而上下margin设为auto垂直居中。

拉伸特性：

.father {height: 200px; position: relative; width: 50px;}

.son {position:absolute; top: 0; right: 0; bottom: 0; left: 0;}

此时子元素会自动填满父元素。

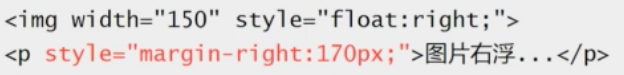
margin负值定位

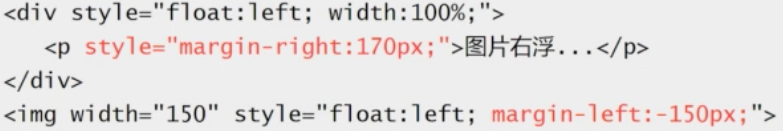
margin负值下的两端对齐

等高布局：



两栏自适应布局





margin无效的情况

1.inline水平元素的垂直margin无效，两个条件：非替换元素；正常书写模式

2.margin重叠

3.display:table-cell/display:table-row等声明的margin无效。替换元素除外。

4.绝对定位的元素在非定位方向（没有设置top/left/bottom/right的方向）上的margin“无效”，定位方向上的margin有效。

margin-start和margin-end

正常的流，margin-start等同于margin-left，两者重叠不累加

如果水平流是从右往左，margin-start等同于margin-right

在垂直流下，（writing-mode:vertical-\*;），margin-start等同于margin-top。

伪类的语法：

selector : pseudo-class {property: value}

CSS 类也可与伪类搭配使用。

selector.class : pseudo-class {property: value}

锚伪类：

a:link {color: #FF0000} /\* 未访问的链接 \*/

a:visited {color: #00FF00} /\* 已访问的链接 \*/

a:hover {color: #FF00FF} /\* 鼠标移动到链接上 \*/

a:active {color: #0000FF} /\* 选定的链接 \*/

动态伪类：

:focus；:hover；:active；

选择第一个子元素：:first-child

根据语言选择：:lang(fr)表示选择所有的法语元素

伪元素的语法：

selector:pseudo-element {property:value;}

CSS 类也可以与伪元素配合使用：

selector.class:pseudo-element {property:value;}

选择首字母：:first-letter应用于所有元素

选择首行：:first-line只用于标记或段落之类的块级元素

设置之前或之后元素的样式：:before,:after

css2允许插入生成的内容，然后使用伪元素:before和:after直接设置样式，它所生成的内容是一个单独的主体。

emmet编辑器语法:

后代：>

缩写：nav>ul>li

<nav>

<ul>

<li></li>

</ul>

</nav>

兄弟：+

缩写：div+p+bq

<div></div>

<p></p>

<blockquote></blockquote>

上级：^

缩写：div+div>p>span+em^bq

<div></div>

<div>

<p><span></span><em></em></p>

<blockquote></blockquote>

</div>

缩写：div+div>p>span+em^^bq

<div></div>

<div>

<p><span></span><em></em></p>

</div>

<blockquote></blockquote>

分组：()

缩写：div>(header>ul>li\*2>a)+footer>p

<div>

<header>

<ul>

<li><a href=""></a></li>

<li><a href=""></a></li>

</ul>

</header>

<footer>

<p></p>

</footer>

</div>

缩写：(div>dl>(dt+dd)\*3)+footer>p

<div>

<dl>

<dt></dt>

<dd></dd>

<dt></dt>

<dd></dd>

<dt></dt>

<dd></dd>

</dl>

</div>

<footer>

<p></p>

</footer>

乘法：\*

缩写：ul>li\*5

<ul>

<li></li>

<li></li>

<li></li>

<li></li>

<li></li>

</ul>

自增符号：$

缩写：ul>li.item$\*2

<ul>

<li class="item1"></li>

<li class="item2"></li>

</ul>

缩写：h$[title=item$]{Header $}\*3

<h1 title="item1">Header 1</h1>

<h2 title="item2">Header 2</h2>

<h3 title="item3">Header 3</h3>

缩写：ul>li.item$$$\*4

<ul>

<li class="item001"></li>

<li class="item002"></li>

<li class="item003"></li>

<li class="item004"></li>

</ul>

缩写：ul>li.item$@-\*5

<ul>

<li class="item5"></li>

<li class="item4"></li>

<li class="item3"></li>

<li class="item2"></li>

<li class="item1"></li>

</ul>

缩写：ul>li.item$@3\*5

<ul>

<li class="item3"></li>

<li class="item4"></li>

<li class="item5"></li>

<li class="item6"></li>

<li class="item7"></li>

</ul>

ID和类属性

缩写：#header

<div id="header"></div>

缩写：.title

<div class="title"></div>

缩写：form#search.wide

<form id="search" class="wide"></form>

缩写：p.class1.class2.class3

<p class="class1 class2 class3"></p>

自定义属性

缩写：p[title=”Hello world”]

<p title="Hello world"></p>

缩写：td[rowspan=2 colspan=3 title]

<td rowspan="2" colspan="3" title=""></td>

缩写：[a=’value1′ b=”value2″]

<div a="value1" b="value2"></div>

文本：{}

缩写：a{Click me}

<a href="">Click me</a>

缩写：p>{Click }+a{here}+{ to continue}

<p>Click <a href="">here</a> to continue</p>

隐式标签

缩写：.class

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

缩写：em>.class

<em><span class="class"></span></em>

缩写：ul>.class

<ul>

<li class="class"></li>

</ul>

缩写：table>.row>.col

<table>

<tr class="row">

<td class="col"></td>

</tr>

</table>

HTML

所有未知的缩写都会转换成标签，例如，foo → <foo></foo>

缩写：!

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

</head>

<body>

</body>

</html>

缩写：a

<a href=""></a>

缩写：a:link

<a href="http://"></a>

缩写：a:mail

<a href="mailto:"></a>

缩写：abbr

<abbr title=""></abbr>

缩写：acronym

<acronym title=""></acronym>

缩写：acronym

<acronym title=""></acronym>

缩写：base

<base href="" />

缩写：basefont

<basefont />

缩写：br

<br />

缩写：frame

<frame />

缩写：hr

<hr />

缩写：bdo

<bdo dir=""></bdo>

缩写：bdo:r

<bdo dir="rtl"></bdo>

缩写：bdo:l

<bdo dir="ltr"></bdo>

缩写：col

<col />

缩写：link

<link rel="stylesheet" href="" />

缩写：link:css

<link rel="stylesheet" href="style.css" />

缩写：link:print

<link rel="stylesheet" href="print.css" media="print" />

缩写：link:favicon

<link rel="shortcut icon" type="image/x-icon" href="favicon.ico" />

缩写：link:touch

<link rel="apple-touch-icon" href="favicon.png" />

缩写：link:rss

<link rel="alternate" type="application/rss+xml" title="RSS" href="rss.xml" />

缩写：link:atom

<link rel="alternate" type="application/atom+xml" title="Atom" href="atom.xml" />

缩写：meta

<meta />

缩写：meta:utf

<meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />

缩写：meta:win

<meta http-equiv="Content-Type" content="text/html;charset=windows-1251" />

缩写：meta:vp

<meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0" />

缩写：meta:compat

<meta http-equiv="X-UA-Compatible" content="IE=7" />

缩写：style

<style></style>

缩写：script:src

<script src=""></script>

缩写：img

<img src="" alt="" />

缩写：iframe

<iframe src="" frameborder="0"></iframe>

缩写：embed

<embed src="" type="" />

缩写：object

<object data="" type=""></object>

缩写：param

<param name="" value="" />

缩写：map

<map name=""></map>

缩写：area

<area shape="" coords="" href="" alt="" />

缩写：area:d

<area shape="default" href="" alt="" />

缩写：area:c

<area shape="circle" coords="" href="" alt="" />

缩写：area:r

<area shape="rect" coords="" href="" alt="" />

缩写：area:p

<area shape="poly" coords="" href="" alt="" />

缩写：form

<form action=""></form>

缩写：form:get

<form action="" method="get"></form>

缩写：form:post

<form action="" method="post"></form>

缩写：label

<label for=""></label>

缩写：input

<input type="text" />

缩写：inp

<input type="text" name="" id="" />

缩写：input:hidden 别名：input[type=hidden name]

<input type="hidden" name="" />

缩写：input:h 别名：input:hidden

<input type="hidden" name="" />

缩写：input:text, input:t 别名：inp

<input type="text" name="" id="" />

缩写：input:search 别名：inp[type=search]

<input type="search" name="" id="" />

缩写：input:email 别名：inp[type=email]

<input type="email" name="" id="" />

缩写：input:url 别名：inp[type=url]

<input type="url" name="" id="" />

缩写：input:password 别名：inp[type=password]

<input type="password" name="" id="" />

缩写：input:p 别名：input:password

<input type="password" name="" id="" />

缩写：input:datetime 别名：inp[type=datetime]

<input type="datetime" name="" id="" />

缩写：input:date 别名：inp[type=date]

<input type="date" name="" id="" />

缩写：input:datetime-local 别名：inp[type=datetime-local]

<input type="datetime-local" name="" id="" />

缩写：input:month 别名：inp[type=month]

<input type="month" name="" id="" />

缩写：input:week 别名：inp[type=week]

<input type="week" name="" id="" />

缩写：input:time 别名：inp[type=time]

<input type="time" name="" id="" />

缩写：input:number 别名：inp[type=number]

<input type="number" name="" id="" />

缩写：input:color 别名：inp[type=color]

<input type="color" name="" id="" />

缩写：input:checkbox 别名：inp[type=checkbox]

<input type="checkbox" name="" id="" />

缩写：input:c 别名：input:checkbox

<input type="checkbox" name="" id="" />

缩写：input:radio 别名：inp[type=radio]

<input type="radio" name="" id="" />

缩写：input:r 别名：input:radio

<input type="radio" name="" id="" />

缩写：input:range 别名：inp[type=range]

<input type="range" name="" id="" />

缩写：input:file 别名：inp[type=file]

<input type="file" name="" id="" />

缩写：input:f 别名：input:file

<input type="file" name="" id="" />

缩写：input:submit

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

缩写：input:s 别名：input:submit

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

缩写：input:image

<input type="image" src="" alt="" />

缩写：input:i 别名：input:image

<input type="image" src="" alt="" />

缩写：input:button

<input type="button" value="" />

缩写：input:b 别名：input:button

<input type="button" value="" />

缩写：isindex

<isindex />

缩写：input:reset 别名：input:button[type=reset]

<input type="reset" value="" />

缩写：select

<select name="" id=""></select>

缩写：option

<option value=""></option>

缩写：textarea

<textarea name="" id="" cols="30" rows="10"></textarea>

缩写：menu:context 别名：menu[type=context]>

<menu type="context"></menu>

缩写：menu:c 别名：menu:context

<menu type="context"></menu>

缩写：menu:toolbar 别名：menu[type=toolbar]>

<menu type="toolbar"></menu>

缩写：menu:t 别名：menu:toolbar

<menu type="toolbar"></menu>

缩写：video

<video src=""></video>

缩写：audio

<audio src=""></audio>

缩写：html:xml

<html xmlns="http://www.w3.org/1999/xhtml"></html>

缩写：keygen

<keygen />

缩写：command

<command />

缩写：bq 别名：blockquote

<blockquote></blockquote>

缩写：acr 别名：acronym

<acronym title=""></acronym>

缩写：fig 别名：figure

<figure></figure>

缩写：figc 别名：figcaption

<figcaption></figcaption>

缩写：ifr 别名：iframe

<iframe src="" frameborder="0"></iframe>

缩写：emb 别名：embed

<embed src="" type="" />

缩写：obj 别名：object

<object data="" type=""></object>

缩写：src 别名：source

<source></source>

缩写：cap 别名：caption

<caption></caption>

缩写：colg 别名：colgroup

<colgroup></colgroup>

缩写：fst, fset 别名：fieldset

<fieldset></fieldset>

缩写：btn 别名：button

<button></button>

缩写：btn:b 别名：button[type=button]

<button type="button"></button>

缩写：btn:r 别名：button[type=reset]

<button type="reset"></button>

缩写：btn:r 别名：button[type=reset]

<button type="reset"></button>

缩写：btn:s 别名：button[type=submit]

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

sublime快捷键

Ctrl+Shift+D ：复制当前行

Ctrl+Shift+K ：删除当前行

Ctrl+/ ：行注释。

Ctrl+Shift+/ ：块注释

Shift+Alt+1，2，3，4，5 ：开启对应数字的多栏编辑

Ctrl+P : 查找项目中的文件：

Ctrl+D ：选中当前光标所在位置的 单词 。连续使用时，进行多光标选择，选中下一个同名单词。

Ctrl+K ：配合 Ctrl+D 可以 跳过 下一个同名单词。

Ctrl+L ：选择当前光标所在位置的 行 。连续使用时，继续选中下一行。

Ctrl+Shift+L ：在多行选中后，在所有选中的行后产生游标。

Alt+F3 ：选中文档中所有的同名单词。

Shift+鼠标右键 ：向下拖动，产生多个光标。

Ctrl+`调出console面板

js代码格式化：先安装JsFormat插件，然后快捷键： Ctrl + Alt + f