# Html 和 CSS

## 元素属性

### 内联元素和块级元素

**CSS的内联元素和块级元素：**

* Block：块级元素，是独立的一行；inline，元素前后不会换行，直到该行排满
* Display：block；默认情况下，block元素宽度自动填满其父元素宽度。block元素可以设置width,height属性。块级元素即使设置了宽度,仍然是独占一行。
* Display：inline; inline元素设置width,height属性无效。inline元素的margin和padding属性，水平方向的padding-left, padding-right, margin-left, margin-right都产生边距效果；但竖直方向的padding-top, padding-bottom, margin-top, margin-bottom不会产生边距效果。
* Display：inline-block。简单来说就是将对象呈现为inline对象，但是对象的内容作为block对象呈现。
* Flex(CSS3新增)：是flexible box的缩写，意为“弹性布局”。任何一个容器都可以指定为Flex布局。注意，设为Flex布局以后，子元素的**float、clear和vertical-align**属性将失效。子元素自动成为容器成员，成为flex项目（flex item）

**内联元素(标签)：**

* input
* span
* a
* label
* em
* img
* strong

**块级元素(标签)：**

* **div**
* **p**
* **h1~h6**
* **ul**
* **form**

#### 去除内联元素1px间距的方法

1. 可以在html中直接把元素写在一行上或把闭合标签和第二个开始标签写在一行或两行间添加注释或直接去掉闭合标签但最后一个不能去掉。
2. 设置margin-right为负值，但要考虑上下文的字体和文字大小。
3. 先设定子元素字体，再设置父元素font-size:0px;////chrome中：-webkit-text-size-adjust:none;
4. 设置父元素letter-spacing或word-spacing为负值，要考虑字体，子元素有文字也要注意设置letter-spacing或word-spacing的值。
5. float:left;

### float(浮动)

* **文档流：**将窗体自上而下分成一行行， 并在每行中按从左至右的顺序排放元素。
* **float**元素脱离于文档流，内容在文档流之内
* **清除浮动**：一般适用于两列、三列等多列布局，能够让子元素内容呈现于父元素之内,目的是不让后续元素受浮动影响。

清除浮动的方式

1. .clearfix(parent):after{content:”.”;display:block;height:0px;clear:both;overflow:hidden;visibility:hidden;}
2. .clearfix:before,.clearfix:after{content: ' ';display: none;}

### flex(弹性布局)

**基本概念**：采用Flex布局的元素，称为Flex容器（flex container），简称"容器"。它的所有子元素自动成为容器成员，称为Flex项目（flex item），简称"项目"。容器默认存在两根轴：水平的主轴（main axis）和垂直的交叉轴（cross axis）

**容器(父元素)属性：**

* flex-direction：决定主轴的方向（即项目的排列方向
* flex-wrap：定义，如果一条轴线排不下，如何换行
* flex-flow：是flex-direction属性和flex-wrap属性的简写形式
* justify-content(主轴 水平居中)：定义了项目在主轴上的对齐方式。
* align-items(辅轴 垂直居中)：定义项目在交叉轴上如何对齐
* align-content：定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。

**项目(子元素)属性：**

* order：定义项目的排列顺序。数值越小，排列越靠前，默认为0。
* flex-grow：定义项目的放大比例，默认为0，即如果存在剩余空间，也不放大。如果所有项目的flex-grow属性都为1，则它们将等分剩余空间（如果有的话）。
* flex-shrink：属性定义了项目的缩小比例，默认为1，即如果空间不足，该项目将缩小。
* flex-basis：定义了在分配多余空间之前，项目占据的主轴空间（main size）。flex-basis:350px,相当于width:350px，前提是主轴油剩余空间
* flex：是flex-grow，flex-shrink，flex-basis三个属性的简写
* align-self：允许单个项目有与其他项目不一样的对齐方式，可覆盖align-items属性

### position

**relative**元素相对于它原本的位置来做定位，但在文档流中依然保持着原有的默认位置，即使它作了偏移，周围的元素也不会占领的它的位置。

如果一个元素是相对定位（甚至它还做了偏移），它周围的元素在排版时参考的依然是那个元素原本的位置（指在没有偏移时的位置）。

**absolute:**绝对定位的元素将脱离文档流。即在排版中，绝对定位的元素不再占有它原来的位置，后面的元素将占领它原本的位置。

绝对定位的元素根据它的相对定位（Relative）的父节点的位置定位。如果它没有相对定位的父节点，则直接参考body容器定位。

**fixed ：**固定定位；类似于absolute，但不随着滚动条的移动而改变位置。

## 元素居中

### 垂直居中

1. 在块级元素下，将height和line-height设置成一样大的数值，就是垂直居中。
2. 父元素：display:table-cell(单元格);vertical(垂直)-align（排列）:middle
3. 父元素：position:relative;子元素：position:absolute;top:50%;transform:translate(-50%)。
4. 父元素：display:flex;align-items(辅轴):center

### 水平居中

1. 子元素：margin：0 auto
2. 子元素：display:inline-block;父元素：text-align:center
3. 子元素：display:table;margin:0 auto
4. 子元素：position:absolute;left:50%;transform:translateX(-50%);父元素：position:relative
5. 父元素：display:flex;justify-content(主轴):center

### 水平和垂直居中

先水平居中，再垂直居中

1. 父元素：text-align:center;display:table-cell;vertical-align:middle;子元素：display:inline-block.
2. 父元素：position:relative;子元素：position:absolute;top:50%;left:50%;transform:translate(-50%,-50%).
3. 父元素：display:flex;justify-content:center;align-items:center;

## 普通元素

宽度根据内容长度来制定(元素属性)：

* absolute
* inline
* table
* flex
* float

**overflow:** 规定当内容溢出元素框时发生的事情

**box-sizing**：border-box: 包括边框和内边距

**box-shadow：**边框添加阴影

* box-shadow:4px(水平偏移) 6px(垂直偏移) 3px(模糊半径) 3px(阴影大小) red

**vertical-align**：元素的垂直对齐方式

**z-index：**设置元素的堆叠顺序 (谁在上面，谁在下面)

**渐变(用的不多)：**

* linear-gradient 线性渐变 视频：第四章 CSS：背景(二)
* radial-gradient 径向渐变 写法：background-image(背景图片):radial-gradient(circle,red,blue);background-image:linear-gradient(to left top,red,blue)

**<dl> <dd> <dt>：**自定义列表

<em></em>：一般用于强调内容

<section></section>：section 不仅仅是一个普通的容器标签。当一个标签只是为了样式化或者方便脚本使用时，应该使用 div 。一般来说，当元素内容明确地出现在文档大纲中时，section 就是适用的。section表示节，一般有标题

visibility:hidden 与 display:none 的区别：前者不脱离文档流，后者脱离文档流，如果重新显示则需要页面的重新绘制。还有一点区别却很少人提到，如果父级设置display:none;子级设置display:block也不会显示；而如果父级设置visibility:hidden;子级设置visibility:visible时子级会显示出来

滚动条：overflow-x:scroll;overflow-y:scroll

### 文本

* **White-space：**设置处理元素间的空白
* **text-indent：**文字缩进
* **word-break：**在断字点进行换行 例如：两个单词之间的空格
* **text-shadow：**文字阴影效果
* **text-decoration：**对文本的修饰 例如：下划线
* **text-overflow：**文本溢出时发生的事情
  + 文本溢出时变成… text-overflow:ellipsis(省略);overflow:hidden;word-break:nowrap(不让文本换行，有些情况下需要) 这三个属性需要配合着使用

## 多列布局

### 两列布局(多列定宽一列自适应)

1. Float+margin
2. Left:{float:left;width:100px;};right{overflow:hidden;}
3. Parent:{display:table;width:100%(**撑满父元素**);table-layout:fixed(**布局优先**)};left,right{display:table-cell;}
4. Parent{display:flex};right{flex:1}

### 多列布局

#### 等分

1. Parent{margin-left:-20px;};col{float:left;width:25%;padding-left:20px;[box-sizing](#_元素属性):border-box;}
2. Parent-fix(是parent的父元素){margin-left:-20px;};parent{display:table;width:100%;table-layout:fixed;};col{display:table-cell;padding-left:20px;}
3. Parent{display:flex};column{flex:1}; .column+.column{margin-left:20px}

#### 等高

1. Parent{display:table;width:100px;table-layout:fixed;};.left,.right{display:table-cell};left{width:100px;padding-right:20px}
2. Parent{display:flex;}.left{width:100px;margin-right:20px;}.right{flex:1;}

## CSS选择器

### 伪类选择器(CSS3)

一般用 ： 表示

#### 动态伪类

* Link：访问前
* Visited：访问后
* Hover：鼠标悬浮
* Active：鼠标点中

一般用于链接(a标签)，顺序为：**Link--visited--hover—active**

#### UI元素状态伪类

* Enabled
* Disabled
* Checked

IE6-8不支持":checked",":enabled",":disabled"这三种选择器

#### CSS3的：nth选择器

* + - First-child (**IE6不支持:first-child选择器**)
* Last-child
* Nth-child() 表示选择某个的一个或多个特定的子元素
* ：nth-child(length); 参数是具体数字。例如：nth-child(3)，选择某元素下的第三个子元素。
* ：nth-child(n);参数是n，n从0开始计算。例如：如果是nth-child(n)，表示所有的子元素；如果是nth-child(2n)，表示选择所有的偶数的子元素，其效果也等于nth-child(even);如果是nth-child(2n-1)，就是奇数选择；如果是nth-child(n+5)，表示从第五个元素开始；以此类推。注意：注意：**IE6-8 和FF3-浏览器不支持":nth-child"选择器。**
  + - Nth-of-type()
* 只计算选择器中的指定的元素。例如，p:nth-of-type(even){….}。注意：**同样在IE6-8和FF3.0-浏览器不支持这种选择器**
  + - Only-child
      * + 表示是其父元素的唯一一个子元素。注意：**IE6-8浏览器不支持:only-child选择器；IE6-8和FF3.0-浏览器不支持:only-of-type选择器。**
    - empty

#### 否定选择器(:not)

* 定位不匹配改选择器的元素。例如，input:not([type=”submit”]){….}。注意：**IE6-8浏览器不支持:not()选择器**

#### 伪元素

* ：：befor :: after 。在元素前后插入内容。这两个常用“content”配合使用，最多的就是清除浮动。
* ：：selection。表示被用户选中的内容
* ：：first-letter。表示文本块的第一个字母。
* ：：first-line。表示元素的第一行。

### 其他选择器

后代选择器(>)

* 例如：.mian>h2，表示选中h2元素(唯一)

兄弟选择器(+)

* 例如：h2+p，表示选中p元素(唯一)
* 例如：h2~p，表示选中h2后面的所有p元素

选择器分组(,)

* 例如：h1,h2,h3 表示选中h1 h2 h3 元素(多个) ，这个简单

## CSS优先级

1. = 行内样式 例如：<p style=”color:rend”></p>
2. = ID选择器的数量
3. = 类、伪类和属性选择器的数量
4. = 标签选择器和伪元素的选择器的数量

上面的是从高到低的顺序， value=a\*1000+b\*100+c\*10+d

相同的属性会覆盖(CSS层叠)

* 改变先后顺序
* 提升选择器优先级
* ！important

# 响应式布局

弹性网格布局，弹性图片，媒体查询

弹性图片：<img>标签 属性：srcset sizes

## 优点

* 减少工作量

1. 网站、设计、代码、内容都只需要一份
2. 多出来的工作量只是JS脚本、CSS样式做一些改动

* 节省时间
* 每个设备都能得到正确的设计
* 搜索优化

## 缺点

* 会加载更多的样式和脚本资源
* 设计比较难精确定位和控制
* 老版本浏览器兼容不好

## px、em、rem(长度单位)

**em**:相对的长度单位

1.em相对参照物为父元素的font-size

2.em具有继承的特点

3.当没有设置font-size时，浏览器会有一个默认的em设置：1em = 16px.

**rem**的相对参照物为根元素html，相对于参照物固定不变，所以比较好计算

当没有设置font-size时，浏览器会有一个默认的rem这是：1rem = 16px，这点与em是一致的(IE8及以下版本不支持)

# 移动端

## 视口

1. 布局视口：原网页大小
2. 可视视口：浏览器设备大小
3. 理想视口：布局视口在一个设备的最佳尺寸

# JavaScript

## 变量命名要求

字母、数字、下划线和$符号，数字不能放前面

## 包装对象(Boolean,Number,String)

例子：var a = “123”;a.length;//3

问题：a是一个标准类型，并没有属性和方法，为什么能调用length属性？

解释：JavaScript引擎自动将a转为包装对象，并且调用包装对象上的属性和方法。

例子：a.t=”11”;a.t;//undefined

问题：既然a为包装对象，自定义一个a的属性，为什么调用时会报错？

解释：一旦调用结束，这个包装对象（临时对象）就会被销毁，所以第二次调用a对象是一个全新的对象，没有t这个属性。并且，这个临时对象是只读的，无法修改，不能添加属性和方法。

其他两个标准（Boolean Number）类型也是类似的

三种包装对象还可以在原型上添加自定义方法和属性，供原始类型的值直接调用

## 属性和方法(对象)

* valueOf() 返回包装对象实例对应的原始类型的值
* toString() 返回实例对应的字符串形式

函数有一个默认的返回值：undefined

## 数据类型的转化

**强制类型转换：**

* 主要是使用函数Number(),String(),Boolean()
* 布尔值转换为数字，true转换为1，false转换为0

**自动转换：**

* **+** 自动转换为字符串进行运算，其他的符号均转换为数值进行运算

## 实例化对象

详情请参考ES5教程，构造函数与new命令章节。

例如：

var A = function(){this.name=1};

var a = new A();

实例化过程：

1. 创建a对象(初始化对象)：a=new Object() 或者 a={};
2. 将a的内置属性\_proto\_指向A的prototype
3. A.call(a) 或者 A.apply(a);设置a的成员(对象属性和方法)，初始化a

总结： 一旦实例化完成，a对象与构造函数A再无联系，无论构造函数A如何变化，都不会影响实例化的a。

注解(第二步)：

**a.constructor===A.prototype.constructor(这个是永远成立的，即使A.prototype不再指向原来的A原型，也就是说：类的实例对象的constructor属性永远指向"构造函数"的prototype.constructor)，从而使得a.constructor.prototype指向A.prototype（obj.constructor.prototype===A.prototype，当A.prototype改变时则不成立）对**

## prototype对象(继承)

原型对象的作用：定义所有实例对象共享的属性和方法。

这也是它被称为原型对象的含义，而实例对象可以视作从原型对象衍生出来的子对象。

null没有自己的原型对象。每一个构造函数都有一个prototype属性，这个属性就是实例对象的原型对象

原型对象上的所有属性和方法，都能被派生对象共享。这就是JavaScript继承机制的基本设计。

## this关键字

### 使用场合

1. 全局环境 🡪 window
2. 构造函数 🡪 实例对象
3. 对象的方法 会改变this的指向

## 作用域链和闭包(变量)

**变量：**分为普通变量和函数变量。

* 普通变量：凡是用var标志的都是普通变量
* 函数变量：函数变量特指的是下面的这种，fun就是一个函数变量。例如：function fun(){} ;// 这是指函数变量. 函数变量一般也说成函数声明。
* 备注：var getA=function(){} //这是函数表达式

**变量声明(ES5:2011)：**

**普通变量声明：**

var x=1; //声明+赋值

var object={}; //声明+赋值

**执行顺序：**

var x = undefined; //声明

var object = undefined; //声明

x = 1; //赋值

object = {}; //赋值

总结：关于声明和赋值，请注意，声明是在函数第一行代码执行之前就已经完成，而赋值是在函数执行时期才开始赋值。所以，声明总是存在于赋值之前。而且，普通变量的声明时期总是等于undefined.

**声明提前：**所有变量的声明，在函数内部第一行代码开始执行的时候就已经完成。

### ES6(2015)

**作用域：**

js中只有函数作用域，没有块级作用域。

例：funtion(){var a=0;} //函数作用域

if(false){ var a=0;} //块级作用域

**闭包：**有权访问另一个作用域的函数

(简单点来说，就是内部函数能够访问外部函数的作用域)

例如：

function A(){

var a=1;

function B(){ //闭包函数，函数b能够访问函数a的作用域。所以，像类似这么样的函数，我们就称为闭包

}

}

## 异步编程

javascript的执行环境是“单线程”(single thread);

所谓"单线程"，就是指一次只能完成一件任务。如果有多个任务，就必须排队，前面一个任务完成，再执行后面一个任务，以此类推。

"异步模式"非常重要。在浏览器端，耗时很长的操作都应该异步执行，避免浏览器失去响应，最好的例子就是Ajax操作。在服务器端，"异步模式"甚至是唯一的模式，因为执行环境是单线程的，如果允许同步执行所有http请求，服务器性能会急剧下降，很快就会失去响应。

### 异步编程的4种方法

#### 1.回调函数

回调函数的优点是简单、容易理解和部署，缺点是不利于代码的阅读和维护，各个部分之间高度[耦合](http://en.wikipedia.org/wiki/Coupling_%28computer_programming%29)（Coupling），流程会很混乱，而且每个任务只能指定一个回调函数。

#### 2.事件监听

另一种思路是采用事件驱动模式。任务的执行不取决于代码的顺序，而取决于某个事件是否发生。首先，为f1绑定一个事件（这里采用的jQuery的[写法](http://api.jquery.com/on/)）。

　f1.on('done', f2);

上面这行代码的意思是，当f1发生done事件，就执行f2。然后，对f1进行改写：

　function f1(){

　　　　setTimeout(function () {

　　　　　　// f1的任务代码

**f1.trigger('done');**

　　　　}, 1000);

　　}

f1.trigger('done')表示，执行完成后，立即触发done事件，从而开始执行f2。

这种方法的优点是比较容易理解，可以绑定多个事件，每个事件可以指定多个回调函数，而且可以["去耦合"](http://en.wikipedia.org/wiki/Decoupling)（Decoupling），有利于实现[模块化](http://www.ruanyifeng.com/blog/2012/10/javascript_module.html)。缺点是整个程序都要变成事件驱动型，运行流程会变得很不清晰。

#### 3.发布/订阅

上一节的"事件"，完全可以理解成"信号"。

我们假定，存在一个"信号中心"，某个任务执行完成，就向信号中心"发布"（publish）一个信号，其他任务可以向信号中心"订阅"（subscribe）这个信号，从而知道什么时候自己可以开始执行。这就叫做["发布/订阅模式"](http://en.wikipedia.org/wiki/Publish-subscribe_pattern)（publish-subscribe pattern），又称["观察者模式"](http://en.wikipedia.org/wiki/Observer_pattern)（observer pattern）。

这个模式有多种[实现](http://msdn.microsoft.com/en-us/magazine/hh201955.aspx)，下面采用的是Ben Alman的[Tiny Pub/Sub](https://gist.github.com/661855)，这是jQuery的一个插件。

首先，f2向"信号中心"jQuery订阅"done"信号。

　　jQuery.subscribe("done", f2);

然后，f1进行如下改写：

　　function f1(){

　　　　setTimeout(function () {

　　　　　　// f1的任务代码

**jQuery.publish("done");**

　　　　}, 1000);

　　}

jQuery.publish("done")的意思是，f1执行完成后，向"信号中心"jQuery发布"done"信号，从而引发f2的执行。

此外，f2完成执行后，也可以取消订阅（unsubscribe）。

　　jQuery.unsubscribe("done", f2);

这种方法的性质与"事件监听"类似，但是明显优于后者。因为我们可以通过查看"消息中心"，了解存在多少信号、每个信号有多少订阅者，从而监控程序的运行。

#### 4.Promise对象

Promises对象是CommonJS工作组提出的一种规范，目的是为异步编程提供[统一接口](http://wiki.commonjs.org/wiki/Promises/A)。

简单说，它的思想是，每一个异步任务返回一个Promise对象，该对象有一个then方法，允许指定回调函数。比如，f1的回调函数f2,可以写成：

　　f1().then(f2);

f1要进行如下改写（这里使用的是jQuery的[实现](http://www.ruanyifeng.com/blog/2011/08/a_detailed_explanation_of_jquery_deferred_object.html)）：

　　function f1(){

　　　　var dfd = $.Deferred();

　　　　setTimeout(function () {

　　　　　　// f1的任务代码

　　　　　　dfd.resolve();

　　　　}, 500);

**return dfd.promise;**

　　}

这样写的优点在于，回调函数变成了链式写法，程序的流程可以看得很清楚，而且有一整套的[配套方法](http://api.jquery.com/category/deferred-object/)，可以实现许多强大的功能。

比如，指定多个回调函数：

　　f1().then(f2).then(f3);

再比如，指定发生错误时的回调函数：

　　f1().then(f2).fail(f3);

而且，它还有一个前面三种方法都没有的好处：如果一个任务已经完成，再添加回调函数，该回调函数会立即执行。所以，你不用担心是否错过了某个事件或信号。这种方法的缺点就是编写和理解，都相对比较难。

### Ajax

#### 原理

 Ajax的工作原理相当于在用户和服务器之间加了—个中间层,使用户操作与服务器响应异步化。并不是所有的用户请求都提交给服务器,像—些数据验证和数据处理等都交给Ajax引擎自己来做, 只有确定需要从服务器读取新数据时再由Ajax引擎代为向服务器提交请求。

通过XmlHttpRequest对象来向服务器发异步请求，从服务器获得数据，然后用javascript来操作DOM而更新页面

## 事件

事件：文档或者浏览器窗口中，发生的特定的交互瞬间

事件流：从页面中接收事件的顺序。

DOM事件流：事件发生时会在元素节点与根节点之间按照特定的顺序传播，路径所经过的所有节点都会收到该事件

DOM0 事件模型：

* 直接在DOM上注册事件
* 原理就是最后注册的事件要覆盖之前的，最后一次注册事件设置成null，也就解除了事件绑定

DOM2 事件模型：

* 标准：addEventListener和removeEventListener进行管理
* DOM2支持同一dom元素注册多个同种事件。
* DOM2新增了捕获和冒泡的概念。
* 参数：事件名、回调函数、布尔值。布尔值：true，捕获阶段调用事件；false，冒泡阶段调用事件
* IE9及以上，firefox,chrome,safari等都支持DOM2事件

## js跨域

跨域：不同域名之间的相互访问，协议、域名、端口不同都是跨域

在js中，我们直接用XMLHttpRequest请求不同域上的数据时，是不可以的。

### jsonp

通过script标签引入一个js文件，这个js文件载入成功后会执行我们在url参数中指定的函数，并且会把我们需要的json数据作为参数传入。所以jsonp是需要服务器端的页面进行相应的配合的。

### window.name

即在一个窗口(window)的生命周期内,窗口载入的所有的页面都是共享一个window.name的，每个页面对window.name都有读写的权限，window.name是持久存在一个窗口载入过的所有页面中的，并不会因新页面的载入而进行重置

注意，window.name的值只能是字符串的形式，这个字符串的大小最大能允许2M左右甚至更大的一个容量，具体取决于不同的浏览器，但一般是够用了。

### window.postMessage

## for循环的执行顺序

for(sta1;sta2;sta3)  
{  
   sta4;  
}  
  
1.进入循环执行sta1;//只是进入的时候执行.  
2.执行sta2;//条件为真才执行sta4,不然就跳出for了.  
3,执行sta4;  
4,执行sta3;  
5,再回到第2步开始执行

## 数组

**数组遍历的集中方法：**

JS:

forEach() IE中Array没有forEach这个方法，不支持

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

for(i in arr) 和java类似，一般用于遍历对象属性

map()

for(let i of arr) ES6新增功能

JQuery:

$().each() jQuery对象遍历

$.each(arr,function(i,n){}) 第一个参数是下标，第二个参数是值

$.grey(arr,function(i){return i>4})

**数组排序：**

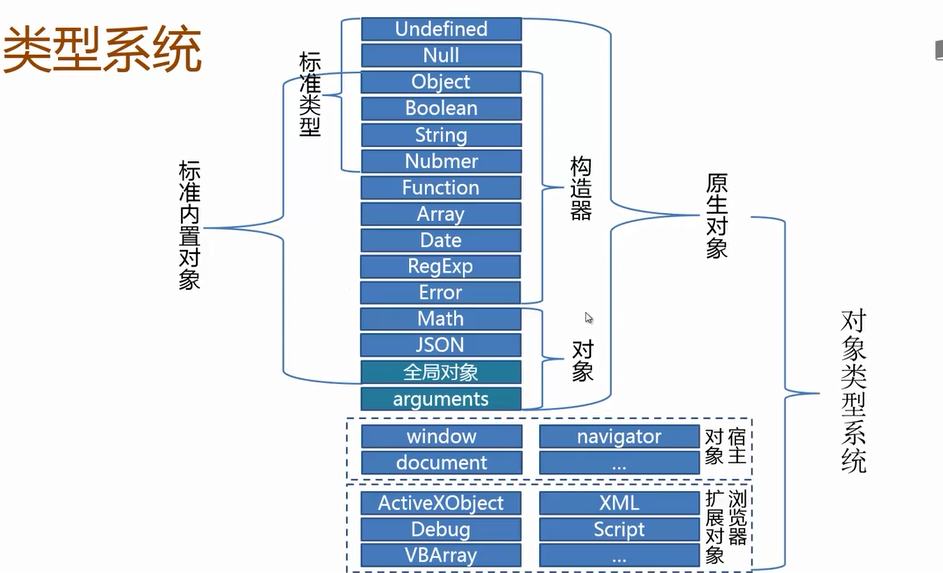
[10111, 1101, 111].sort(function (a, b) { //a,b表示数组中的任意两个元素

return a - b;

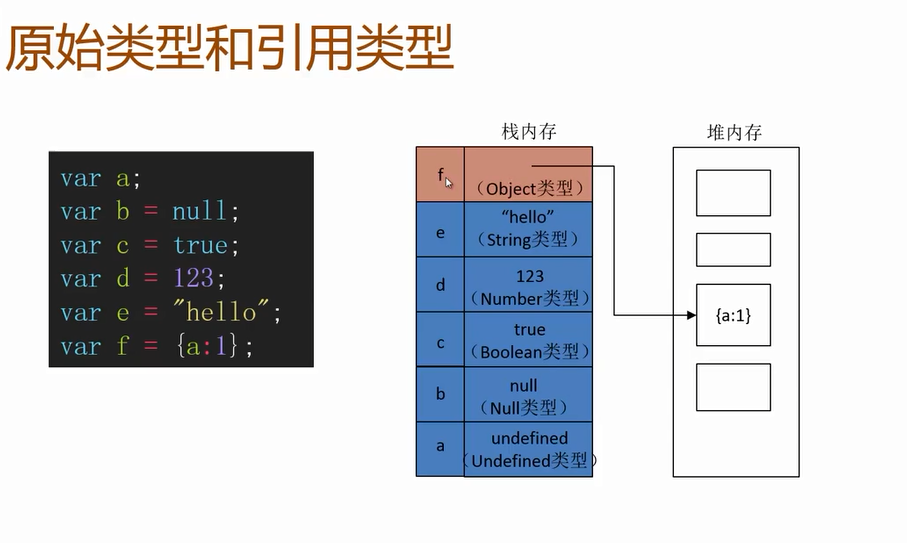
})

简化一下：a-b输出从小到大排序，b-a输出从大到小排序。

## js类型



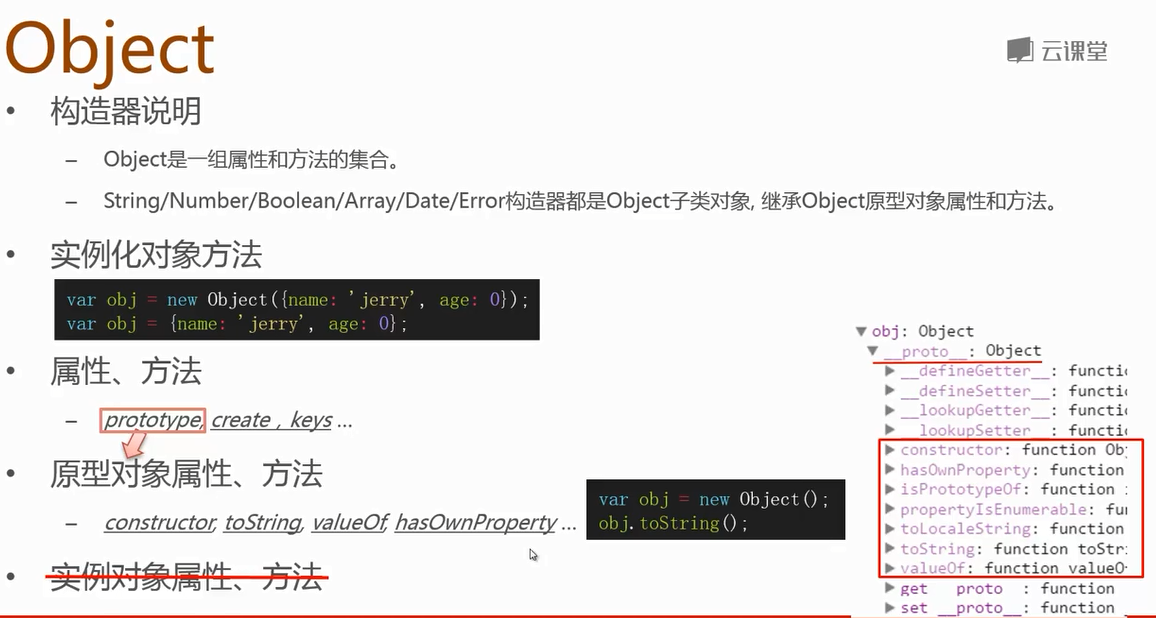
### 标准类型



## 内置对象

标准内置对象，类型系统图里有。

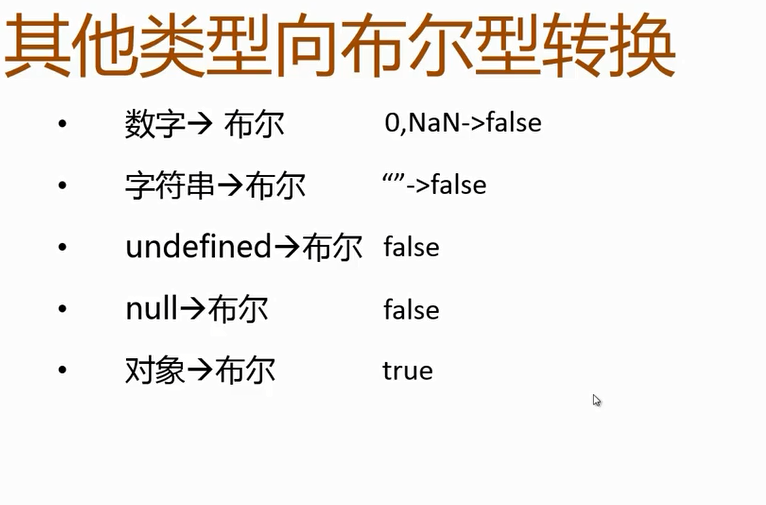
### Object



### Boolean



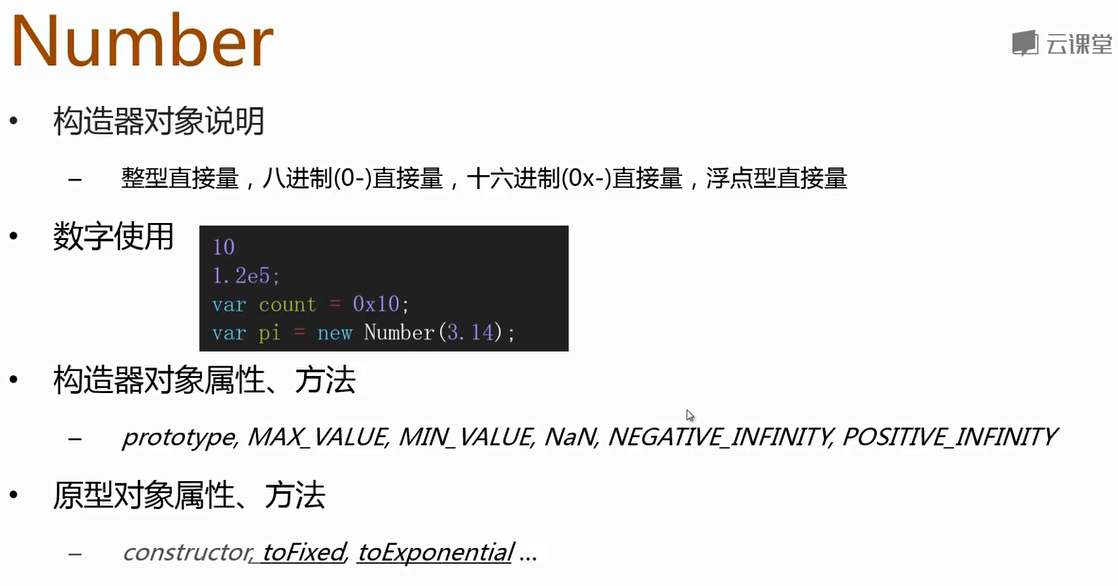
类型转换



### String



### Number

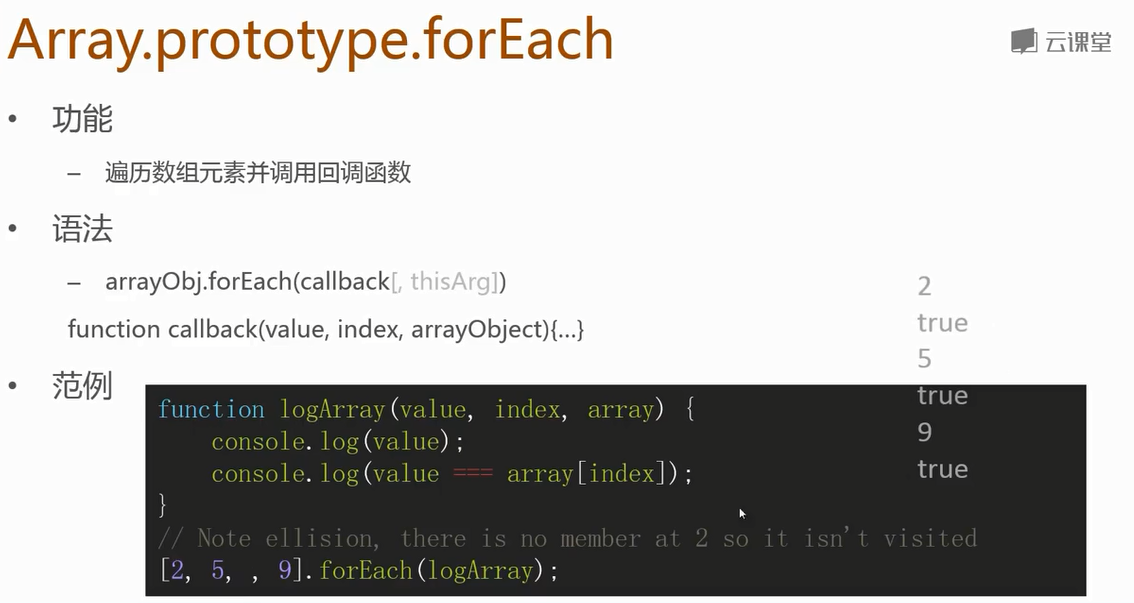


### Array



对象属性、方法

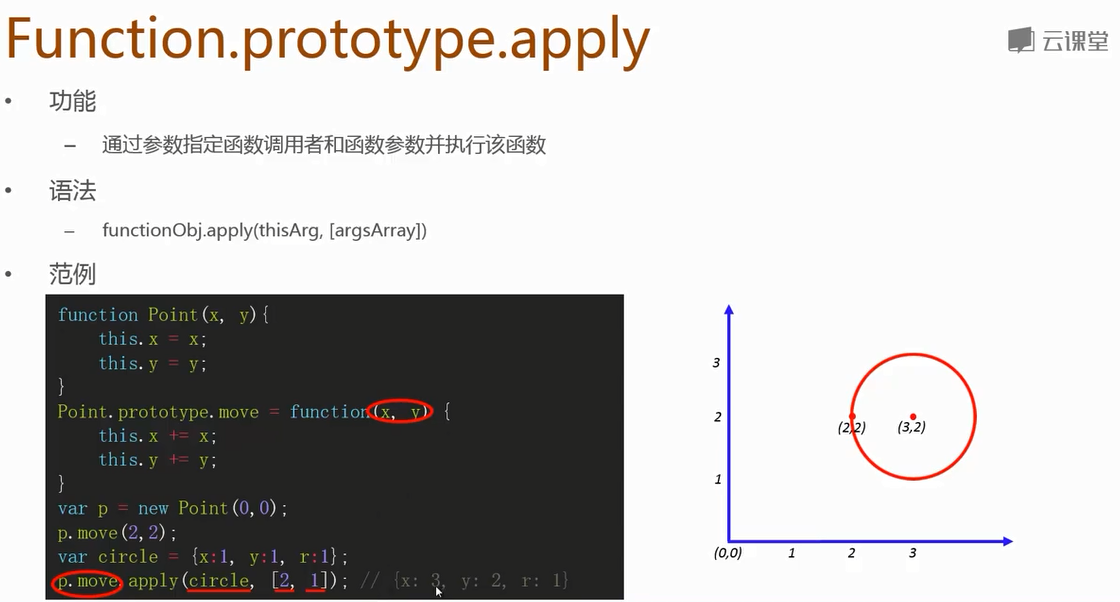




### Function



原型对象属性或方法



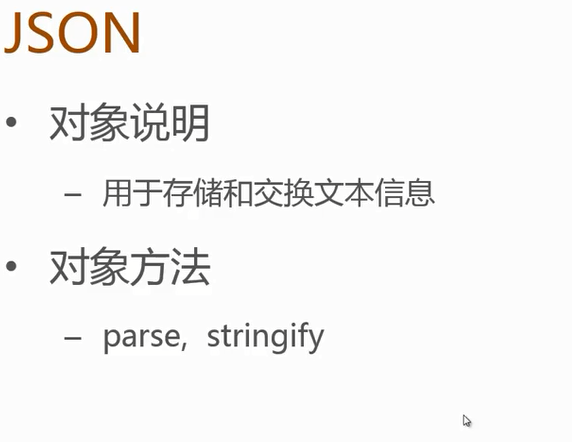
### RegExp



### Date



### Json

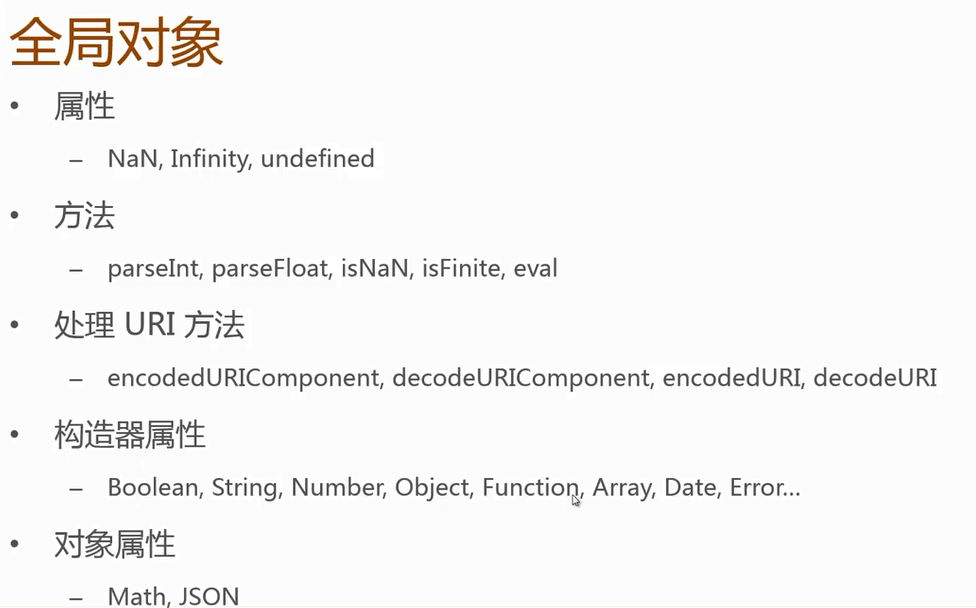


对象方法

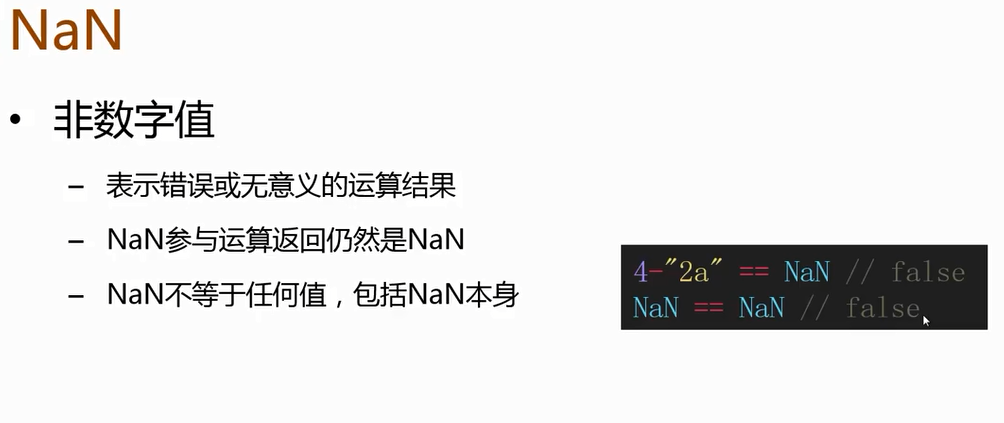




### 全局对象



#### NaN



#### parseInt



#### eval



# jQuery

jQuery五大块：选择器、DOM操作，事件，AJAX，动画。

jQuery便是一个封装得非常好的类，比如我们用 语句　$(“#btn1″) 会生成一个 jQuery类的实例。

jQuery.fn = jQuery.prototype

jQuery.extend(object)：为jQuery类添加类方法，可以理解为添加静态方法

jQuery.fn.extend(object);对jQuery.prototype进得扩展，就是为jQuery类添加“成员函数”。jQuery类的实例可以使用这个“成员函数”。

### 核心方法

* 回调系统
* 异步队列
* 数据缓存
* 队列操作
* 选择器引
* 属性操作
* 节点遍历
* 文档处理
* 样式操作
* 事件体系
* AJAX交互
* 动画引擎

jQuery.fn.init.prototype = jQuery.fn

* jQuery.fn.init.prototype = jQuery.fn = jQuery.prototype ;
* new jQuery.fn.init() 相当于 new jQuery() ;
* jQuery() 返回的是 new jQuery.fn.init()，而 var obj = new jQuery()，所以这 2 者是相当的，所以我们可以无 new 实例化 jQuery 对象。

# nodeJS

## 编程思想

一切皆异步

npm init 生成package.json文件

npm install express utility –save (--save 参数，这个参数的作用，就是会在你安装依赖的同时，自动把这些依赖写入 package.json)

var app = express();

app.use(path,callback)中的callback既可以是router对象又可以是函数

app.get(path,callback)中的callback只能是函数

## 模块

**eventProxy**:

1. 利用事件机制解耦复杂业务逻辑
2. 移除被广为诟病的深度callback嵌套问题
3. 将串行等待变成并行等待，提升多异步协作场景下的执行效率
4. 友好的Error handling
5. 无平台依赖，适合前后端，能用于浏览器和Node.js
6. 兼容CMD，AMD以及CommonJS模块环境

emit:触发事件

all：绑定事件(全部)

after:多次触发，多次回调

**superagent**：客户端请求代理模块。当你想处理get,post,put,delete,head请求时,你就应该想起该用它了。superagent 是一个轻量的,渐进式的ajax api,可读性好,学习曲线低,内部依赖nodejs原生的请求api,适用于nodejs环境下.

**cheerio**: 为服务器特别定制的，快速、灵活、实施的jQuery核心实现.

## Express

* request.params:从Express路由器获取参数 get
* request.query：获取?后的查询参数 get
* request.body：获取路由参数 post(但是express需单独引入post body 的解析器)

utility有两个很重要的方法，一个是sha1，一个是md5，通常使用他们对字符串进行加密处理。

# reactJS

为什么要在return的DOM结构的最外层包一层div标签？

表明其返回一个结果，如果返回两个或多个结果，编译就会报错

前端实现：样式和机构的分离

# 浏览器环境(知识点)

### 工作原理

网页加载流程：

1. 浏览器一边下载HTML网页，一边开始解析
2. 解析过程中，发现<script>标签
3. 暂停解析，网页渲染的控制权转交给JavaScript引擎
4. 如果<script>标签引用了外部脚本，就下载该脚本，否则就直接执行
5. 执行完毕，控制权交还渲染引擎，恢复往下解析HTML网页

Js(<scritp>标签)同步加载，css(<link>标签)异步加载

浏览器的核心是两部分：渲染引擎和JavaScript解释器（又称JavaScript引擎）。

渲染引擎的主要作用是，将网页代码渲染为用户视觉可以感知的平面文档。

渲染引擎处理网页，通常分成四个阶段。

1. 解析代码：HTML代码解析为DOM，CSS代码解析为CSSOM（CSS Object Model）
2. 对象合成：将DOM和CSSOM合成一棵渲染树（render tree）
3. 布局：计算出渲染树的布局（layout）
4. 绘制：将渲染树绘制到屏幕

### sessionStorage和localStorage

html5中的Web Storage包括了两种存储方式：sessionStorage和localStorage。

**sessionStorage**用于本地存储一个会话（session）中的数据，这些数据只有在同一个会话中的页面才能访问并且当会话结束后数据也随之销毁。因此sessionStorage不是一种持久化的本地存储，仅仅是会话级别的存储。

而**localStorage**用于持久化的本地存储，除非主动删除数据，否则数据是永远不会过期的。

Web Storage的概念和cookie相似，区别是它是为了更大容量存储设计的。Cookie的大小是受限的，并且每次你请求一个新的页面的时候Cookie都会被发送过去，这样无形中浪费了带宽，另外cookie还需要指定作用域，不可以跨域调用。

除此之外，**Web Storage**拥有setItem,getItem,removeItem,clear等方法，不像cookie需要前端开发者自己封装setCookie，getCookie。

但是Cookie也是不可以或缺的：Cookie的作用是与服务器进行交互，作为HTTP规范的一部分而存在 ，而Web Storage仅仅是为了在本地“存储”数据而生

### DOM文档加载的步骤

(1) 解析HTML结构。

(2) 加载外部脚本和样式表文件。

(3) 解析并执行脚本代码。

(4) 构造HTML DOM模型。//ready

(5) 加载图片等外部文件。

(6) 页面加载完毕。//load

# 前端开发(项目实际应用)

## 前端模块化开发

**工作原理:**

1. npm 包管理
2. spm 前端组件
3. gulp 前端工作流
4. webpack js模块管理

**工作流程：**

1. clone 克隆到本地仓库
2. npm install 相当于mvn install
3. gulp install webpack(内部去做了) 作用：实时监测js、css文件的变更
4. tomcat 启动

## 函数

Require函数：用来加载模块

## Require.js

**基于AMD标准，用来统一管理类库**

**例如：rjs—config.js文件**

requirejs.config({

//默认情况下模块所在目录为js/lib

baseUrl: 'js',

//当模块id前缀为app时，他便由js/app加载模块文件

//这里设置的路径是相对与baseUrl的，不要包含.js

paths: {

jquery: 'jslib/jquery.min',

text: 'requireJs-plugins/text',

template:'template.min',

bootstrap:'jslib/bootstrap.min',

np:'jslib/nprogress',

psbar:'jslib/progressbar/bootstrap-progressbar.min',

bsPage:'jslib/bootstrap-paginator.min',

iCheck:'jslib/icheck/icheck.min',

ztree:'jslib/zTree\_v3/js/jquery.ztree.all-3.5.min',

treeTable:'jslib/treeTable/jquery.treetable',

validate:'jslib/jquery-plugin/jquery.validate',

metadata:'jslib/jquery-plugin/jquery.metadata',

wdatePicker:"jslib/My97DatePicker/WdatePicker",

calendar:"jslib/calendar/fullcalendar.min",

moment:"jslib/moment/moment.min",

mutiSelect:"jslib/bootstrap-multiselect/bootstrap-multiselect",

uploadify:"jslib/uploadify/jquery.uploadify",

ajaxUpload:"jslib/ajaxfileupload/ajaxfileupload",

chained:"jslib/jquery-chained/jquery.chained",

kindeditor:"jslib/kindeditor/kindeditor-min",

kindeditor\_all:"jslib/kindeditor/kindeditor-all-min",

}

});

## Sea.js

**Require.async:此方法可用来异步加载模块，并在加载完成后执行回调函数**

例如：define(function (require, exports, module) {}

# Photoshop切图

**合并图层：**Ctrl+e

**选中，保存：**Ctrl+c 然后 Ctrl+n

**图片100%：**Ctrl+Alt+0 or Ctrl+1

**放大：**Ctrl+空格键+鼠标左键 or Ctrl+加号

**缩小：**Ctrl+减号

放大或缩小：Alt+鼠标滚轮(首选)

**拖拽：**空格键+鼠标左键

**标尺：**Ctrl+r

# 页面优化

## 减少请求

图片合并，css文件合并，js文件合并

## 2.加载顺序