前端开发规范

黄金定律

永远遵循同一套编码规范 -- 可以是这里列出的，也可以是你自己总结的。如果你发现本规范中有任何错误，敬请指正。

不管有多少人共同参与同一项目，一定要确保每一行代码都像是同一个人编写的。

HTML 代码规范

一、语法

- 用 4 个空格来代替制表符（tab） -- 这是唯一能保证在所有环境下获得一致展现的方法。
- 嵌套元素应当缩进一次（即 4 个空格）。
- 对于属性的定义，确保全部使用双引号，绝不要使用单引号。
- 不要在自闭合（self-closing）元素的尾部添加斜线 --HTML5 规范中明确说明这是可选的。
- 不要省略可选的结束标签（closing tag）（例如，
或 </body>）。

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <img src="images/company-logo.png" alt="Company">
    <h1 class="hello-world">Hello, world!</h1>
  </body>
</html>
```

二、HTML5 doctype

为每个 HTML 页面的第一行添加标准模式（standard mode）的声明，这样能够确保在每个浏览器中拥有一致的展现。

```
<!DOCTYPE html>
```

```
<html>
  <head>
  </head>
</html>
```

三、语言属性

根据 HTML5 规范：

强烈建议为 html 根元素指定 lang 属性，从而为文档设置正确的语言。这将有助于语音合成工具确定其所应该采用的发音，有助于翻译工具确定其翻译时所应遵守的规则等等。

更多关于 `lang` 属性的知识可以从 此规范 中了解。

这里列出了语言代码表。

```
<html lang="zh-CN">
  <!-- ... -->
</html>
```

四、IE 兼容模式

IE 支持通过特定的 `<meta>` 标签来确定绘制当前页面所应该采用的 IE 版本。除非有强烈的特殊需求，否则最好是设置为 edge mode，从而通知 IE 采用其所支持的最新的模式。

阅读这篇 stack overflow 上的文章可以获得更多有用的信息。

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

五、字符编码

通过明确声明字符编码，能够确保浏览器快速并容易的判断页面内容的渲染方式。这样做的好处是，可以避免在 HTML 中使用字符实体标记（character entity），从而全部与文档编码一致（一般采用 UTF-8 编码）。

```
<head>
  <meta charset="UTF-8">
</head>
```

六、引入 CSS 和 JavaScript 文件

根据 HTML5 规范，在引入 CSS 和 JavaScript 文件时一般不需要指定 `type` 属性，因为 `text/css` 和 `text/javascript` 分别是它们的默认值。

HTML5 spec links

- Using link
- Using style
- Using script

```
<!-- External CSS -->
<link rel="stylesheet" href="code-guide.css">

<!-- In-document CSS -->
<style>
  /* ... */
</style>

<!-- JavaScript -->
<script src="code-guide.js"></script>
```

七、实用为王

尽量遵循 HTML 标准和语义，但是不要以牺牲实用性为代价。任何时候都要尽量使用最少的标签并保持最小的复杂度。

八、属性顺序

HTML 属性应当按照以下给出的顺序依次排列，确保代码的易读性。

- `class`
- `id`, `name`
- `data-*`
- `src`, `for`, `type`, `href`
- `title`, `alt`
- `aria-*`, `role`

class 用于标识高度可复用组件，因此应该排在首位。id 用于标识具体组件，应当谨慎使用（例如，页面内的书签），因此排在第二位。

```
<a class="..." id="..." data-modal="toggle" href="#">
  Example link
```

```
</a>

<input class="form-control" type="text">

<img src="..." alt="...">
```

九、布尔（boolean）型属性

布尔型属性可以在声明时不赋值。XHTML 规范要求为其赋值，但是 HTML5 规范不需要。

更多信息请参考 [WhatWG section on boolean attributes]():

元素的布尔型属性如果有值，就是 true，如果没有值，就是 false。

如果一定要为其赋值的话，请参考 WhatWG 规范：

如果属性存在，其值必须是空字符串或 [...] 属性的规范名称，并且不要再收尾添加空白符。
简单来说，就是不用赋值。

```
<input type="text" disabled>

<input type="checkbox" value="1" checked>

<select>
  <option value="1" selected>1</option>
</select>
```

十、减少标签的数量

编写 HTML 代码时，尽量避免多余的父元素。很多时候，这需要迭代和重构来实现。请看下面的案例：

```
<!-- Not so great -->
<span class="avatar">
  <img src="...">
</span>

<!-- Better -->
<img class="avatar" src="...">
```

十一、JavaScript 生成的标签

通过 JavaScript 生成的标签让内容变得不易查找、编辑，并且降低性能。能避免时尽量避免。

CSS 代码规范

一、语法

- 用 4 个空格来代替制表符（tab） -- 这是唯一能保证在所有环境下获得一致展现的方法。
- 为选择器分组时，将单独的选择器单独放在一行。
- 为了代码的易读性，在每个声明块的左花括号前添加一个空格。
- 声明块的右花括号应当单独成行。
- 每条声明语句的 `:` 后应该插入一个空格。
- 为了获得更准确的错误报告，每条声明都应该独占一行。
- 所有声明语句都应当以分号结尾。最后一条声明语句后面的分号是可选的，但是，如果省略这个分号，你的代码可能更易出错。
- 对于以逗号分隔的属性值，每个逗号后面都应该插入一个空格（例如，`box-shadow`）。
- 不要在 `rgb()`、`rgba()`、`hsl()`、`hsla()` 或 `rect()` 值的*内部*的逗号后面插入空格。这样利于从多个属性值（既加逗号也加空格）中区分多个颜色值（只加逗号，不加空格）。
- 对于属性值或颜色参数，省略小于 1 的小数前面的 0 （例如，`.5` 代替 `0.5`；`-.5px` 代替 `-0.5px`）。
- 十六进制值应该全部小写，例如，`#fff`。在扫描文档时，小写字符易于分辨，因为他们的形式更易于区分。
- 尽量使用简写形式的十六进制值，例如，用 `#fff` 代替`#ffffff`。
- 为选择器中的属性添加双引号，例如，`input[type="text"]`。只有在某些情况下是可选的，但是，为了代码的一致性，建议都加上双引号。
- 避免为 0 值指定单位，例如，用 `margin: 0;` 代替 `margin: 0px;`。

对于这里用到的术语有疑问吗？请参考 Wikipedia 上的 syntax section of the Cascading Style Sheets article。

```
/* Bad CSS */
.selector, .selector-secondary, .selector[type=text] {
  padding:15px;
  margin:0px 0px 15px;
  background-color:rgba(0, 0, 0, 0.5);
  box-shadow:0px 1px 2px #CCC,inset 0 1px 0 #FFFFFF
}

/* Good CSS */
.selector,
.selector-secondary,
```

```
.selector[type="text"] {
  padding: 15px;
  margin-bottom: 15px;
  background-color: rgba(0,0,0,.5);
  box-shadow: 0 1px 2px #ccc, inset 0 1px 0 #fff;
}
```

二、声明顺序

相关的属性声明应当归为一组，并按照下面的顺序排列：

1. Positioning
2. Box model
3. Typographic
4. Visual

由于定位（positioning）可以从正常的文档流中移除元素，并且还能覆盖盒模型（box model）相关的样式，因此排在首位。盒模型排在第二位，因为它决定了组件的尺寸和位置。

其他属性只是影响组件的*内部（inside）*或者是不影响前两组属性，因此排在后面。

完整的属性列表及其排列顺序请参考 Recess。

```
.declaration-order {
  /* Positioning */
  position: absolute;
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;
  z-index: 100;

  /* Box-model */
  display: block;
  float: right;
  width: 100px;
  height: 100px;

  /* Typography */
  font: normal 13px "Helvetica Neue", sans-serif;
  line-height: 1.5;
  color: #333;
  text-align: center;
```

```
  /* Visual */
  background-color: #f5f5f5;
  border: 1px solid #e5e5e5;
  border-radius: 3px;

  /* Misc */
  opacity: 1;
}
```

三、不要使用 `@import`

与 `<link>` 标签相比，`@import` 指令要慢很多，不光增加了额外的请求次数，还会导致不可预料的问题。替代办法有以下几种：

- 使用多个 `<link>` 元素
- 通过 Sass 或 Less 类似的 CSS 预处理器将多个 CSS 文件编译为一个文件
- 通过 Rails、Jekyll 或其他系统中提供过 CSS 文件合并功能

请参考 Steve Souders 的文章了解更多知识。

```
<!-- Use link elements -->
<link rel="stylesheet" href="core.css">

<!-- Avoid @imports -->
<style>
  @import url("more.css");
</style>
```

四、媒体查询（Media query）的位置

将媒体查询放在尽可能相关规则的附近。不要将他们打包放在一个单一样式文件中或者放在文档底部。如果你把他们分开了，将来只会被大家遗忘。下面给出一个典型的实例。

```
.element { ... }
.element-avatar { ... }
.element-selected { ... }

@media (min-width: 480px) {
  .element { ...}
```

```
  .element-avatar { ... }
  .element-selected { ... }
}
```

五、带前缀的属性

当使用特定厂商的带有前缀的属性时，通过缩进的方式，让每个属性的值在垂直方向对齐，这样便于多行编辑。

在 Textmate 中，使用 **Text → Edit Each Line in Selection**(^⌘A)。在 Sublime Text 2 中，使用 **Selection → Add Previous Line** (^⇧↑) 和 **Selection → Add Next Line** (^⇧↓)。

```
/* Prefixed properties */
.selector {
  -webkit-box-shadow: 0 1px 2px rgba(0,0,0,.15);
          box-shadow: 0 1px 2px rgba(0,0,0,.15);
}
```

六、单行规则声明

对于**只包含一条声明**的样式，为了易读性和便于快速编辑，建议将语句放在同一行。对于带有多条声明的样式，还是应当将声明分为多行。

这样做的关键因素是为了错误检测 -- 例如，CSS 校验器指出在 183 行有语法错误。如果是单行单条声明，你就不会忽略这个错误；如果是单行多条声明的话，你就要仔细分析避免漏掉错误了。

```
/* Single declarations on one line */
.span1 { width: 60px; }
.span2 { width: 140px; }
.span3 { width: 220px; }

/* Multiple declarations, one per line */
.sprite {
  display: inline-block;
  width: 16px;
  height: 15px;
  background-image: url(../img/sprite.png);
}
.icon         { background-position: 0 0; }
.icon-home    { background-position: 0 -20px; }
.icon-account { background-position: 0 -40px; }
```

七、简写形式的属性声明

在需要显示地设置所有值的情况下，应当尽量限制使用简写形式的属性声明。常见的滥用简写属性声明的情况如下：

- padding
- margin
- font
- background
- border
- border-radius

大部分情况下，我们不需要为简写形式的属性声明指定所有值。例如，HTML 的 heading 元素只需要设置上、下边距（margin）的值，因此，在必要的时候，只需覆盖这两个值就可以。过度使用简写形式的属性声明会导致代码混乱，并且会对属性值带来不必要的覆盖从而引起意外的副作用。

MDN（Mozilla Developer Network）上一片非常好的关于 shorthand properties 的文章，对于不太熟悉简写属性声明及其行为的用户很有用。

```
/* Bad example */
.element {
  margin: 0 0 10px;
  background: red;
  background: url("image.jpg");
  border-radius: 3px 3px 0 0;
}

/* Good example */
.element {
  margin-bottom: 10px;
  background-color: red;
  background-image: url("image.jpg");
  border-top-left-radius: 3px;
  border-top-right-radius: 3px;
}
```

八、Less 和 Sass 中的嵌套

避免非必要的嵌套。这是因为虽然你可以使用嵌套，但是并不意味着应该使用嵌套。只有在必须将样式限制在父元素内（也就是后代选择器），并且存在多个需要嵌套的元素时才使用嵌套。

```
// Without nesting
```

```
.table > thead > tr > th { … }
.table > thead > tr > td { … }

// With nesting
.table > thead > tr {
  > th { … }
  > td { … }
}
```

九、注释

代码是由人编写并维护的。请确保你的代码能够自描述、注释良好并且易于他人理解。好的代码注释能够传达上下文关系和代码目的。不要简单地重申组件或 class 名称。

对于较长的注释，务必书写完整的句子；对于一般性注解，可以书写简洁的短语。

```
/* Bad example */
/* Modal header */
.modal-header {
  ...
}

/* Good example */
/* Wrapping element for .modal-title and .modal-close */
.modal-header {
  ...
}
```

十、class 命名

- class 名称中只能出现小写字符和破折号（dashe）（不是下划线，也不是驼峰命名法）。破折号应当用于相关 class 的命名（类似于命名空间）（例如，`.btn` 和 `.btn-danger`）。
- 避免过度任意的简写。`.btn` 代表 *button*，但是 `.s` 不能表达任何意思。
- class 名称应当尽可能短，并且意义明确。
- 使用有意义的名称。使用有组织的或目的明确的名称，不要使用表现形式（presentational）的名称。
- 基于最近的父 class 或基本（base） class 作为新 class 的前缀。
- 使用 `.js-*` class 来标识行为（与样式相对），并且不要将这些 class 包含到 CSS 文件中。

在为 Sass 和 Less 变量命名是也可以参考上面列出的各项规范。

```
/* Bad example */
.t { ... }
.red { ... }
.header { ... }

/* Good example */
.tweet { ... }
.important { ... }
.tweet-header { ... }
```

十一、选择器

- 对于通用元素使用 class ，这样利于渲染性能的优化。
- 对于经常出现的组件，避免使用属性选择器（例如，`[class^="..."]`）。浏览器的性能会受到这些因素的影响。
- 选择器要尽可能短，并且尽量限制组成选择器的元素个数，建议不要超过 3 。
- **只有**在必要的时候才将 class 限制在最近的父元素内（也就是后代选择器）（例如，不使用带前缀的 class 时 -- 前缀类似于命名空间）。

扩展阅读：

- Scope CSS classes with prefixes
- Stop the cascade

```
/* Bad example */
span { ... }
.page-container
#stream .stream-item .tweet .tweet-header .username
{ ... }
.avatar { ... }

/* Good example */
.avatar { ... }
.tweet-header .username { ... }
.tweet .avatar { ... }
```

十二、代码组织

- 以组件为单位组织代码段。
- 制定一致的注释规范。
- 使用一致的空白符将代码分隔成块，这样利于扫描较大的文档。
- 如果使用了多个 CSS 文件，将其按照组件而非页面的形式分拆，因为页面会被重组，而组件只会被移动。

```
/*
 * Component section heading
 */

.element { ... }


/*
 * Component section heading
 *
 * Sometimes you need to include optional context for the entire component. Do that up here if it's important enough.
 */

.element { ... }

/* Contextual sub-component or modifer */
.element-heading { ... }
```

十三、编辑器配置

将你的编辑器按照下面的配置进行设置，以避免常见的代码不一致和差异：

- 用 4 个空格代替制表符（soft-tab 即用空格代表 tab 符）。
- 保存文件时，删除尾部的空白符。
- 设置文件编码为 UTF-8。
- 在文件结尾添加一个空白行。

参照文档并将这些配置信息添加到项目的 `.editorconfig` 文件中。例如：Bootstrap 中的 .editorconfig 实例。更多信息请参考 about EditorConfig。

Bootstrap 全局 CSS 样式

一、概览

深入了解 Bootstrap 底层结构的关键部分，包括我们让 web 开发变得更好、更快、更强壮的最佳实践。

HTML5 文档类型

Bootstrap 使用到的某些 HTML 元素和 CSS 属性需要将页面设置为 HTML5 文档类型。在你项目中的每个页面都要参照下面的格式进行设置。

```
<!DOCTYPE html>
<html lang="zh-CN">
  ...
</html>
```

移动设备优先

在 Bootstrap 2 中，我们对框架中的某些关键部分增加了对移动设备友好的样式。而在 Bootstrap 3 中，我们重写了整个框架，使其一开始就是对移动设备友好的。这次不是简单的增加一些可选的针对移动设备的样式，而是直接融合进了框架的内核中。也就是说，Bootstrap 是移动设备优先的。针对移动设备的样式融合进了框架的每个角落，而不是增加一个额外的文件。

为了确保适当的绘制和触屏缩放，需要在 <head> 之中添加 viewport 元数据标签。

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

在移动设备浏览器上，通过为视口（viewport）设置 meta 属性为 user-scalable=no 可以禁用其缩放（zooming）功能。这样禁用缩放功能后，用户只能滚动屏幕，就能让你的网站看上去更像原生应用的感觉。注意，这种方式我们并不推荐所有网站使用，还是要看你自己的情况而定！

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

排版与链接

Bootstrap 排版、链接样式设置了基本的全局样式。分别是：

为 body 元素设置 background-color: #fff;
使用 @font-family-base、@font-size-base 和 @line-height-base a 变量作为排版的基本参数
为所有链接设置了基本颜色 @link-color ，并且当链接处于 :hover 状态时才添加下划线
这些样式都能在 scaffolding.less 文件中找到对应的源码。

Normalize.css

为了增强跨浏览器表现的一致性，我们使用了 Normalize.css，这是由 Nicolas Gallagher 和 Jonathan Neal 维护的一个 CSS 重置样式库。

布局容器

Bootstrap 需要为页面内容和栅格系统包裹一个 .container 容器。我们提供了两个作此用处的类。注意，由于 padding 等属性的原因，这两种 容器类不能互相嵌套。

.container 类用于固定宽度并支持响应式布局的容器。

```
<div class="container">
  ...
</div>
```

.container-fluid 类用于 100% 宽度，占据全部视口（viewport）的容器。

```
<div class="container-fluid">
  ...
</div>
```

二、栅格系统

Bootstrap 提供了一套响应式、移动设备优先的流式栅格系统，随着屏幕或视口（viewport）尺寸的增加，系统会自动分为最多 12 列。它包含了易于使用的预定义类，还有强大的 mixin 用于生成更具语义的布局。

简介

栅格系统用于通过一系列的行（row）与列（column）的组合来创建页面布局，你的内容就可以放入这些创建好的布局中。下面就介绍一下 Bootstrap 栅格系统的工作原理：

“行（row）”必须包含在 .container （固定宽度）或 .container-fluid （100% 宽度）中，以便为其赋予合适的排列（aligment）和内补（padding）。
通过“行（row）”在水平方向创建一组“列（column）”。
你的内容应当放置于“列（column）”内，并且，只有“列（column）”可以作为行（row）”的直接子元素。
类似 .row 和 .col-xs-4 这种预定义的类，可以用来快速创建栅格布局。Bootstrap 源码中定义的 mixin 也可以用来创建语义化的布局。
通过为“列（column）”设置 padding 属性，从而创建列与列之间的间隔（gutter）。通过为 .row

元素设置负值 margin 从而抵消掉为 .container 元素设置的 padding，也就间接为“行（row）”所包含的“列（column）”抵消掉了 padding。
The negative margin is why the examples below are outdented. It's so that content within grid columns is lined up with non-grid content.
Grid columns are created by specifying the number of twelve available columns you wish to span. For example, three equal columns would use three .col-xs-4.
如果一“行（row）”中包含了的“列（column）”大于 12，多余的“列（column）”所在的元素将被作为一个整体另起一行排列。
Grid classes apply to devices with screen widths greater than or equal to the breakpoint sizes, and override grid classes targeted at smaller devices. Therefore, applying any .col-md- class to an element will not only affect its styling on medium devices but also on large devices if a .col-lg- class is not present.
通过研究后面的实例，可以将这些原理应用到你的代码中。

媒体查询

在栅格系统中，我们在 Less 文件中使用以下媒体查询（media query）来创建关键的分界点阈值。

```
/* 超小屏幕（手机，小于 768px） */
/* 没有任何媒体查询相关的代码，因为这在 Bootstrap 中是默认的（还记得 Bootstrap 是移动设备优先的吗？） */

/* 小屏幕（平板，大于等于 768px） */
@media (min-width: @screen-sm-min) { ... }

/* 中等屏幕（桌面显示器，大于等于 992px） */
@media (min-width: @screen-md-min) { ... }

/* 大屏幕（大桌面显示器，大于等于 1200px） */
@media (min-width: @screen-lg-min) { ... }
```

我们偶尔也会在媒体查询代码中包含 max-width 从而将 CSS 的影响限制在更小范围的屏幕大小之内。

```
@media (max-width: @screen-xs-max) { ... }
@media (min-width: @screen-sm-min) and (max-width: @screen-sm-max) { ... }
@media (min-width: @screen-md-min) and (max-width: @screen-md-max) { ... }
@media (min-width: @screen-lg-min) { ... }
```

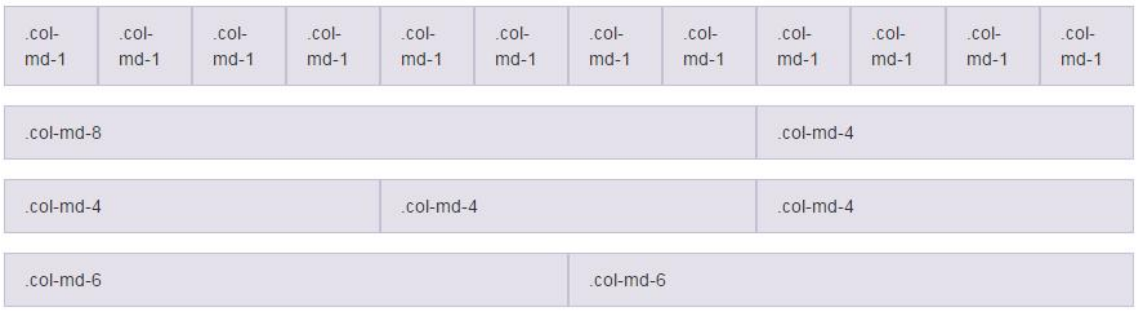
栅格参数

通过下表可以详细查看 Bootstrap 的栅格系统是如何在多种屏幕设备上工作的。

	超小屏幕 手机 (<768px)	小屏幕 平板 (≥768px)	中等屏幕 桌面显示器 (≥992px)	大屏幕 大桌面显示器 (≥1200px)
栅格系统行为	总是水平排列	开始是堆叠在一起的，当大于这些阈值时将变为水平排列 C		
`.container` 最大宽度	None （自动）	750px	970px	1170px
类前缀	`.col-xs-`	`.col-sm-`	`.col-md-`	`.col-lg-`
列（column）数	12			
最大列（column）宽	自动	~62px	~81px	~97px
槽（gutter）宽	30px （每列左右均有 15px）			
可嵌套	是			
偏移（Offsets）	是			
列排序	是			

实例：从堆叠到水平排列

使用单一的一组 .col-md-* 栅格类，就可以创建一个基本的栅格系统，在手机和平板设备上一开始是堆叠在一起的（超小屏幕到小屏幕这一范围），在桌面（中等）屏幕设备上变为水平排列。所有“列（column）必须放在 ” .row 内。




```
<div class="row">
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
</div>
<div class="row">
  <div class="col-md-8">.col-md-8</div>
  <div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
  <div class="col-md-6">.col-md-6</div>
  <div class="col-md-6">.col-md-6</div>
</div>
```

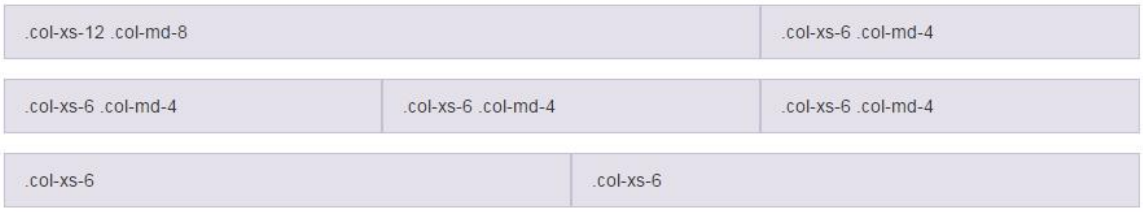
实例：流式布局容器

将最外面的布局元素 .container 修改为 .container-fluid，就可以将固定宽度的栅格布局转换为 100% 宽度的布局。

```
<div class="container-fluid">
  <div class="row">
    ...
  </div>
</div>
```

实例：移动设备和桌面屏幕

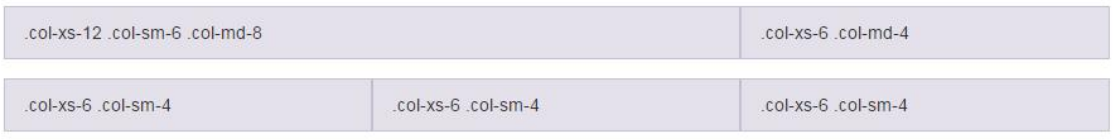
是否不希望在小屏幕设备上所有列都堆叠在一起？那就使用针对超小屏幕和中等屏幕设备所定义的类吧，即 .col-xs-* 和 .col-md-*。请看下面的实例，研究一下这些是如何工作的。




```
<!-- Stack the columns on mobile by making one full-width and the other half-width -->
<div class="row">
  <div class="col-xs-12 col-md-8">.col-xs-12 .col-md-8</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>

<!-- Columns start at 50% wide on mobile and bump up to 33.3% wide on desktop -->
<div class="row">
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>

<!-- Columns are always 50% wide, on mobile and desktop -->
<div class="row">
  <div class="col-xs-6">.col-xs-6</div>
  <div class="col-xs-6">.col-xs-6</div>
</div>
```

实例：手机、平板、桌面

在上面案例的基础上，通过使用针对平板设备的 .col-sm-* 类，我们来创建更加动态和强大的布局吧。

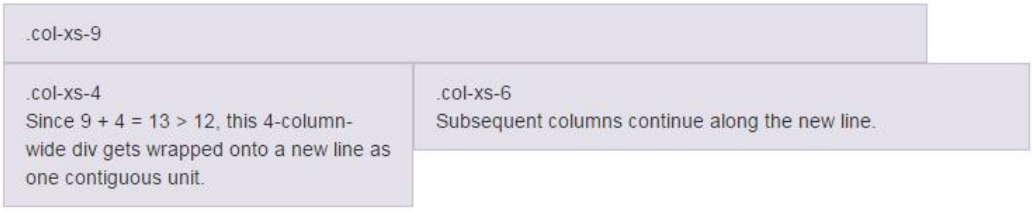



```
<div class="row">
```

```
  <div class="col-xs-12 col-sm-6 col-md-8">.col-xs-12 .col-sm-6 .col-md-8</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>
<div class="row">
  <div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>
  <div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>
  <!-- Optional: clear the XS cols if their content doesn't match in height -->
  <div class="clearfix visible-xs-block"></div>
  <div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>
</div>
```

实例：多余的列（column）将另起一行排列

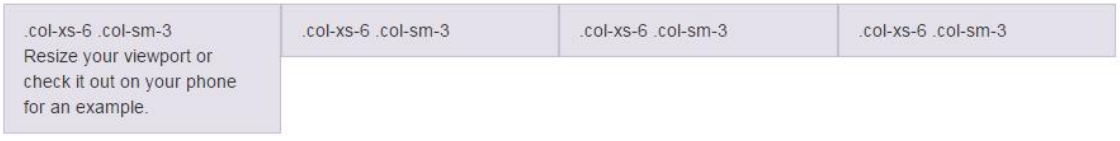
如果在一个 .row 内包含的列（column）大于 12 个，包含多余列（column）的元素将作为一个整体单元被另起一行排列。




```
<div class="row">
  <div class="col-xs-9">.col-xs-9</div>
  <div class="col-xs-4">.col-xs-4<br>Since 9 + 4 = 13 &gt; 12, this 4-column-wide div gets wrapped onto a new line as one contiguous unit.</div>
  <div class="col-xs-6">.col-xs-6<br>Subsequent columns continue along the new line.</div>
</div>
```

Responsive column resets

With the four tiers of grids available you're bound to run into issues where, at certain breakpoints, your columns don't clear quite right as one is taller than the other. To fix that, use a combination of a .clearfix and our responsive utility classes.




```
<div class="row">
  <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3</div>
  <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3</div>
```

```
  <!-- Add the extra clearfix for only the required viewport -->
  <div class="clearfix visible-xs-block"></div>

  <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3</div>
  <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3</div>
</div>
```

In addition to column clearing at responsive breakpoints, you may need to reset offsets, pushes, or pulls. See this in action in the grid example.

```
<div class="row">
  <div class="col-sm-5 col-md-6">.col-sm-5 .col-md-6</div>
  <div class="col-sm-5 col-sm-offset-2 col-md-6
col-md-offset-0">.col-sm-5 .col-sm-offset-2 .col-md-6 .col-md-offset-0</div>
</div>

<div class="row">
  <div class="col-sm-6 col-md-5 col-lg-6">.col-sm-6 .col-md-5 .col-lg-6</div>
  <div class="col-sm-6 col-md-5 col-md-offset-2 col-lg-6
col-lg-offset-0">.col-sm-6 .col-md-5 .col-md-offset-2 .col-lg-6 .col-lg-offset-0</div>
</div>
```

列偏移

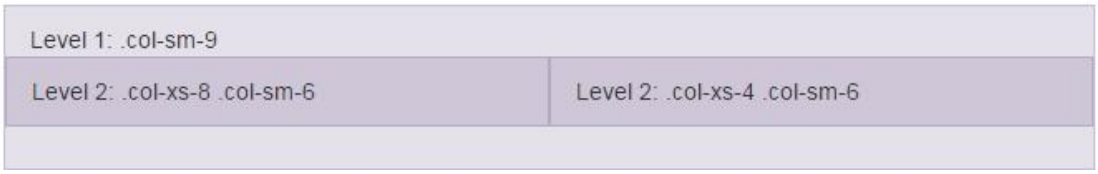
使用 .col-md-offset-* 类可以将列向右侧偏移。这些类实际是通过使用 * 选择器为当前元素增加了左侧的边距（margin）。例如，.col-md-offset-4 类将 .col-md-4 元素向右侧偏移了 4 个列（column）的宽度。


```
<div class="row">
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4 col-md-offset-4">.col-md-4 .col-md-offset-4</div>
</div>
<div class="row">
  <div class="col-md-3 col-md-offset-3">.col-md-3 .col-md-offset-3</div>
  <div class="col-md-3 col-md-offset-3">.col-md-3 .col-md-offset-3</div>
</div>
<div class="row">
  <div class="col-md-6 col-md-offset-3">.col-md-6 .col-md-offset-3</div>
```

```
</div>
```

嵌套列

为了使用内置的栅格系统将内容再次嵌套，可以通过添加一个新的 .row 元素和一系列 .col-sm-* 元素到已经存在的 .col-sm-* 元素内。被嵌套的行（row）所包含的列（column）的个数不能超过 12（其实，没有要求你必须占满 12 列）。




```
<div class="row">
  <div class="col-sm-9">
    Level 1: .col-sm-9
    <div class="row">
      <div class="col-xs-8 col-sm-6">
        Level 2: .col-xs-8 .col-sm-6
      </div>
      <div class="col-xs-4 col-sm-6">
        Level 2: .col-xs-4 .col-sm-6
      </div>
    </div>
  </div>
</div>
```

列排序

通过使用 .col-md-push-* 和 .col-md-pull-* 类就可以很容易的改变列（column）的顺序。


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

Less mixin 和变量

除了用于快速布局的预定义栅格类，Bootstrap 还包含了一组 Less 变量和 mixin 用于帮你生成简单、语义化的布局。

变量

通过变量来定义列数、槽（gutter）宽、媒体查询阈值（用于确定合适让列浮动）。我们使用这些变量生成预定义的栅格类，如上所示，还有如下所示的定制 mixin。

```
@grid-columns:              12;
@grid-gutter-width:         30px;
@grid-float-breakpoint:     768px;
```

mixin

mixin 用来和栅格变量一同使用，为每个列（column）生成语义化的 CSS 代码。

```
// Creates a wrapper for a series of columns
.make-row(@gutter: @grid-gutter-width) {
  // Then clear the floated columns
  .clearfix();

  @media (min-width: @screen-sm-min) {
    margin-left:  (@gutter / -2);
    margin-right: (@gutter / -2);
  }

  // Negative margin nested rows out to align the content of columns
  .row {
    margin-left:  (@gutter / -2);
    margin-right: (@gutter / -2);
  }
}

// Generate the extra small columns
.make-xs-column(@columns; @gutter: @grid-gutter-width) {
  position: relative;
  // Prevent columns from collapsing when empty
  min-height: 1px;
  // Inner gutter via padding
  padding-left:  (@gutter / 2);
  padding-right: (@gutter / 2);

  // Calculate width based on number of columns available
  @media (min-width: @grid-float-breakpoint) {
    float: left;
    width: percentage((@columns / @grid-columns));
  }
}
```

```
// Generate the small columns
.make-sm-column(@columns; @gutter: @grid-gutter-width) {
    position: relative;
    // Prevent columns from collapsing when empty
    min-height: 1px;
    // Inner gutter via padding
    padding-left:    (@gutter / 2);
    padding-right: (@gutter / 2);

    // Calculate width based on number of columns available
    @media (min-width: @screen-sm-min) {
        float: left;
        width: percentage((@columns / @grid-columns));
    }
}

// Generate the small column offsets
.make-sm-column-offset(@columns) {
    @media (min-width: @screen-sm-min) {
        margin-left: percentage((@columns / @grid-columns));
    }
}
.make-sm-column-push(@columns) {
    @media (min-width: @screen-sm-min) {
        left: percentage((@columns / @grid-columns));
    }
}
.make-sm-column-pull(@columns) {
    @media (min-width: @screen-sm-min) {
        right: percentage((@columns / @grid-columns));
    }
}

// Generate the medium columns
.make-md-column(@columns; @gutter: @grid-gutter-width) {
    position: relative;
    // Prevent columns from collapsing when empty
    min-height: 1px;
    // Inner gutter via padding
    padding-left:    (@gutter / 2);
    padding-right: (@gutter / 2);

    // Calculate width based on number of columns available
    @media (min-width: @screen-md-min) {
```

```
        float: left;
        width: percentage((@columns / @grid-columns));
    }
}

// Generate the medium column offsets
.make-md-column-offset(@columns) {
    @media (min-width: @screen-md-min) {
        margin-left: percentage((@columns / @grid-columns));
    }
}
.make-md-column-push(@columns) {
    @media (min-width: @screen-md-min) {
        left: percentage((@columns / @grid-columns));
    }
}
.make-md-column-pull(@columns) {
    @media (min-width: @screen-md-min) {
        right: percentage((@columns / @grid-columns));
    }
}

// Generate the large columns
.make-lg-column(@columns; @gutter: @grid-gutter-width) {
    position: relative;
    // Prevent columns from collapsing when empty
    min-height: 1px;
    // Inner gutter via padding
    padding-left:    (@gutter / 2);
    padding-right: (@gutter / 2);

    // Calculate width based on number of columns available
    @media (min-width: @screen-lg-min) {
        float: left;
        width: percentage((@columns / @grid-columns));
    }
}

// Generate the large column offsets
.make-lg-column-offset(@columns) {
    @media (min-width: @screen-lg-min) {
        margin-left: percentage((@columns / @grid-columns));
    }
}
```

```
.make-lg-column-push(@columns) {
  @media (min-width: @screen-lg-min) {
    left: percentage((@columns / @grid-columns));
  }
}
.make-lg-column-pull(@columns) {
  @media (min-width: @screen-lg-min) {
    right: percentage((@columns / @grid-columns));
  }
}
```

实例展示

你可以重新修改这些变量的值，或者用默认值调用这些 mixin。下面就是一个利用默认设置生成两列布局（列之间有间隔）的案例。

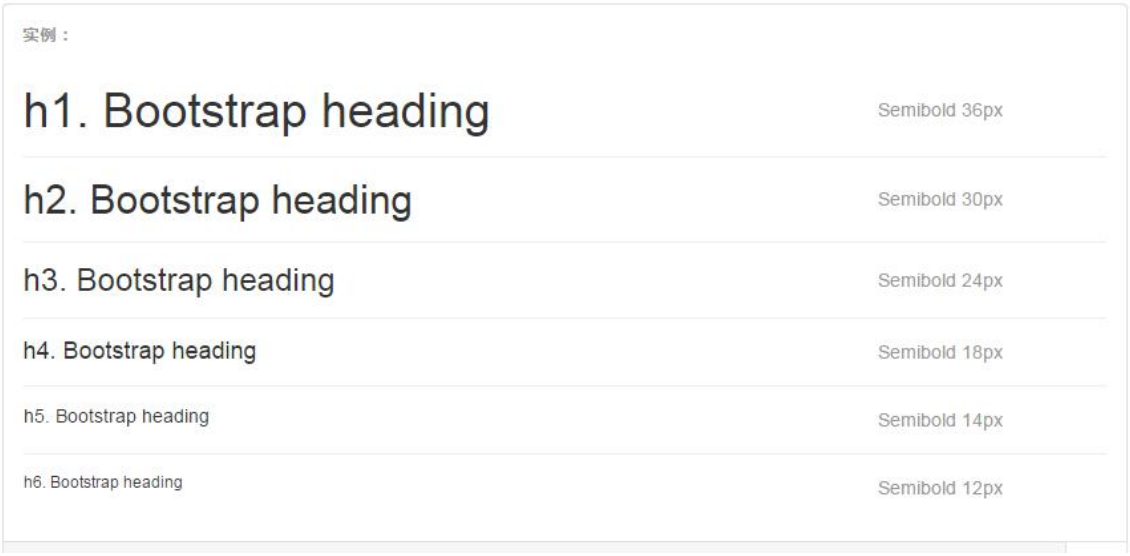
```
.wrapper {
  .make-row();
}
.content-main {
  .make-lg-column(8);
}
.content-secondary {
  .make-lg-column(3);
  .make-lg-column-offset(1);
}
```

```
<div class="wrapper">
  <div class="content-main">...</div>
  <div class="content-secondary">...</div>
</div>
```

三、排版

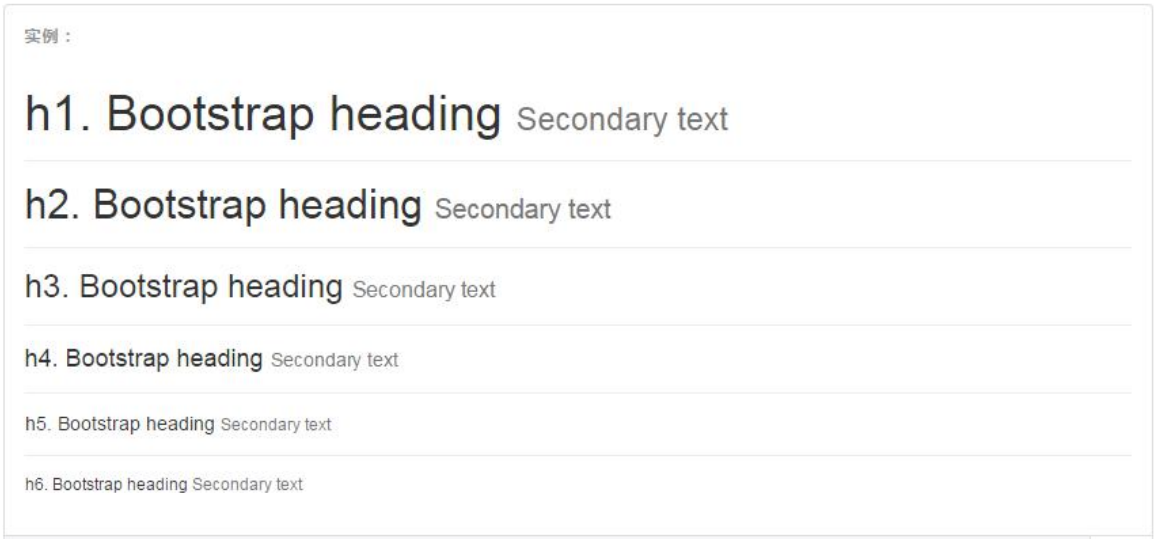
标题

HTML 中的所有标题标签，<h1> 到 <h6> 均可使用。另外，还提供了 .h1 到 .h6 类，为的是给内联（inline）属性的文本赋予标题的样式。




```
<h1>h1. Bootstrap heading</h1>
<h2>h2. Bootstrap heading</h2>
<h3>h3. Bootstrap heading</h3>
<h4>h4. Bootstrap heading</h4>
<h5>h5. Bootstrap heading</h5>
<h6>h6. Bootstrap heading</h6>
```

在标题内还可以包含 <small> 标签或赋予 .small 类的元素，可以用来标记副标题。




```
<h1>h1. Bootstrap heading <small>Secondary text</small></h1>
<h2>h2. Bootstrap heading <small>Secondary text</small></h2>
<h3>h3. Bootstrap heading <small>Secondary text</small></h3>
<h4>h4. Bootstrap heading <small>Secondary text</small></h4>
<h5>h5. Bootstrap heading <small>Secondary text</small></h5>
<h6>h6. Bootstrap heading <small>Secondary text</small></h6>
```

页面主体

Bootstrap 将全局 font-size 设置为 14px，line-height 设置为 1.428。这些属性直接赋予

<body> 元素和所有段落元素。另外，<p> （段落）元素还被设置了等于 1/2 行高（即 10px）的底部外边距（margin）。

Nullam quis risus eget urna mollis ornare vel eu leo. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam id dolor id nibh ultricies vehicula.

Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec ullamcorper nulla non metus auctor fringilla. Duis mollis, est non commodo luctus, nisi erat porttitor ligula, eget lacinia odio sem nec elit. Donec ullamcorper nulla non metus auctor fringilla.

Maecenas sed diam eget risus varius blandit sit amet non magna. Donec id elit non mi porta gravida at eget metus. Duis mollis, est non commodo luctus, nisi erat porttitor ligula, eget lacinia odio sem nec elit.

```
<p>...</p>
```

中心内容

通过添加 .lead 类可以让段落突出显示。

Vivamus sagittis lacus vel augue laoreet rutrum faucibus dolor auctor. Duis mollis, est non commodo luctus.

```
<p class="lead">...</p>
```

使用 Less 工具构建

variables.less 文件中定义的两个 Less 变量决定了排版尺寸：@font-size-base 和 @line-height-base。第一个变量定义了全局 font-size 基准，第二个变量是 line-height 基准。我们使用这些变量和一些简单的公式计算出其它所有页面元素的 margin、 padding 和 line-height。自定义这些变量即可改变 Bootstrap 的默认样式。

内联文本元素

Marked text

For highlighting a run of text due to its relevance in another context, use the <mark> tag.

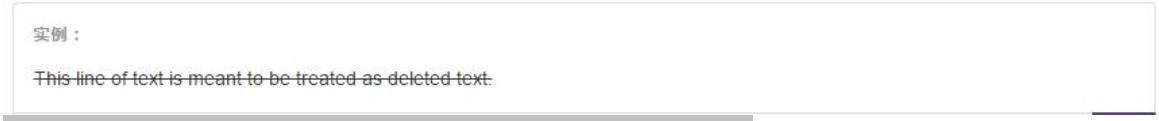
实例：

You can use the mark tag to highlight text.

```
You can use the mark tag to <mark>highlight</mark> text.
```

被删除的文本

对于被删除的文本使用 <del> 标签。




```
<del>This line of text is meant to be treated as deleted text.</del>
```

无用文本

对于没用的文本使用 <s> 标签。


```
<s>This line of text is meant to be treated as no longer accurate.</s>
```

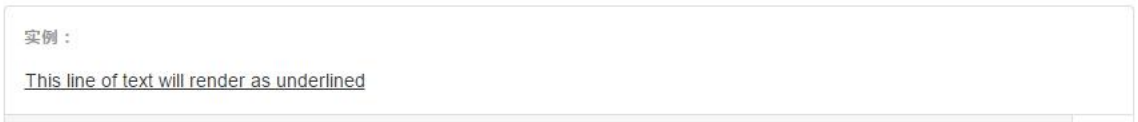
插入文本

额外插入的文本使用 <ins> 标签。


```
<ins>This line of text is meant to be treated as an addition to the document.</ins>
```

带下划线的文本

为文本添加下划线，使用 <u> 标签。




```
<u>This line of text will render as underlined</u>
```

利用 HTML 自带的表示强调意味的标签来为文本增添少量样式。

小号文本

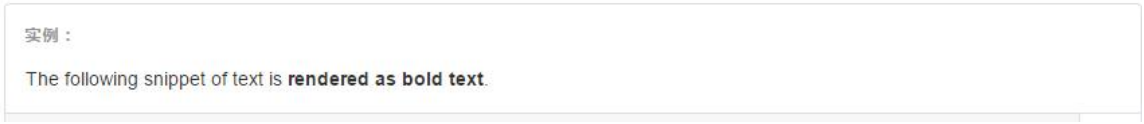
对于不需要强调的 inline 或 block 类型的文本，使用 <small> 标签包裹，其内的文本将被设置为父容器字体大小的 85%。标题元素中嵌套的 <small> 元素被设置不同的 font-size 。

你还可以为行内元素赋予 .small 类以代替任何 <small> 元素。


```
<small>This line of text is meant to be treated as fine print.</small>
```

着重

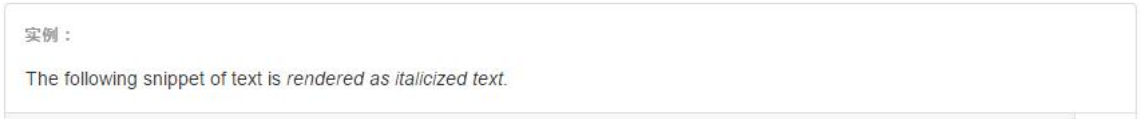
通过增加 font-weight 值强调一段文本。




```
<strong>rendered as bold text</strong>
```

斜体

用斜体强调一段文本。



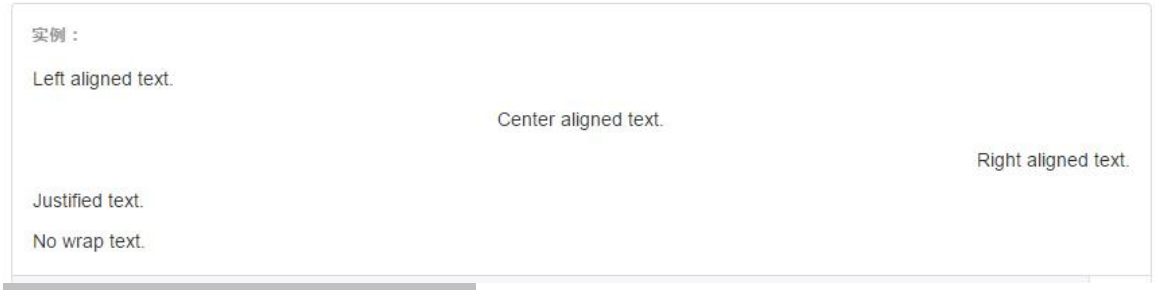

```
<em>rendered as italicized text</em>
```

Alternate elements

在 HTML5 中可以放心使用 <b> 和 <i> 标签。<b> 用于高亮单词或短语，不带有任何着重的意味；而 <i> 标签主要用于发言、技术词汇等。

对齐

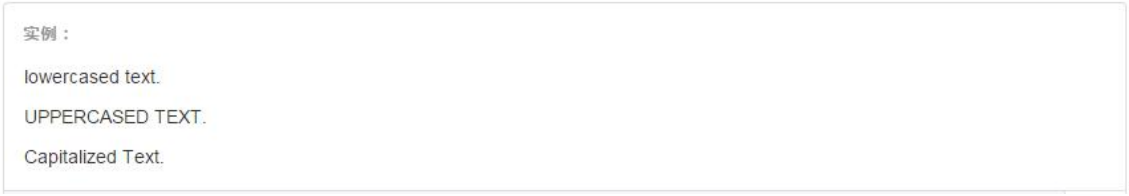
通过文本对齐类，可以简单方便的将文字重新对齐。




```
<p class="text-left">Left aligned text.</p>
<p class="text-center">Center aligned text.</p>
<p class="text-right">Right aligned text.</p>
<p class="text-justify">Justified text.</p>
<p class="text-nowrap">No wrap text.</p>
```

改变大小写

通过这几个类可以改变文本的大小写。



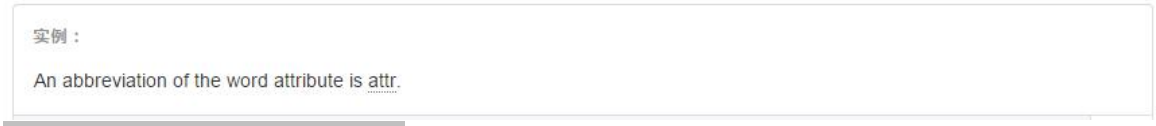

```
<p class="text-lowercase">Lowercased text.</p>
<p class="text-uppercase">Uppercased text.</p>
<p class="text-capitalize">Capitalized text.</p>
```

缩略语

当鼠标悬停在缩写和缩写词上时就会显示完整内容，Bootstrap 实现了对 HTML 的 <abbr> 元素的增强样式。缩略语元素带有 title 属性，外观表现为带有较浅的虚线框，鼠标移至上面时会变成带有“问号”的指针。如想看完整的内容可把鼠标悬停在缩略语上，但需要包含 title 属性。

基本缩略语

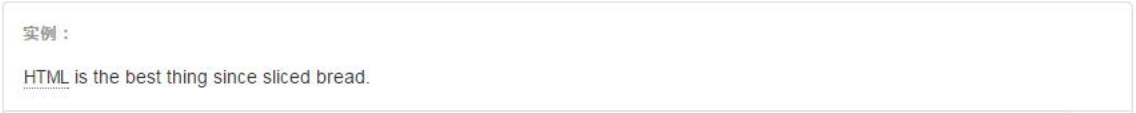
如想看完整的内容可把鼠标悬停在缩略语上，但需要为 <abbr> 元素设置 title 属性。




```
<abbr title="attribute">attr</abbr>
```

首字母缩略语

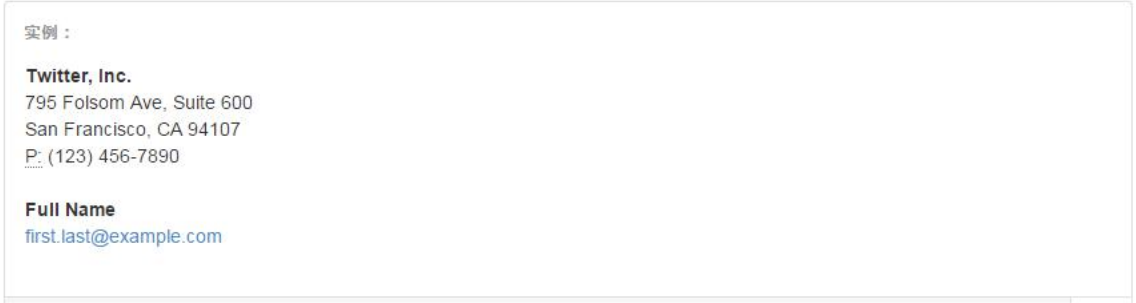
为缩略语添加 .initialism 类，可以让 font-size 变得稍微小些。




```
<abbr title="HyperText Markup Language" class="initialism">HTML</abbr>
```

地址

让联系信息以最接近日常使用的格式呈现。在每行结尾添加
 可以保留需要的样式。




```
<address>
```

```
  <strong>Twitter, Inc.</strong><br>
  795 Folsom Ave, Suite 600<br>
  San Francisco, CA 94107<br>
  <abbr title="Phone">P:</abbr> (123) 456-7890
</address>

<address>
  <strong>Full Name</strong><br>
  <a href="mailto:#">first.last@example.com</a>
</address>
```

引用

在你的文档中引用其他来源的内容。

默认样式的引用

将任何 HTML 元素包裹在 <blockquote> 中即可表现为引用样式。对于直接引用，我们建议用 <p> 标签。

实例：

> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.

```
<blockquote>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.</p>
</blockquote>
```

Blockquote options

Style and content changes for simple variations on a standard <blockquote>.

Naming a source

Add a <footer> for identifying the source. Wrap the name of the source work in <cite>.

实例：

> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.
>
> — Someone famous in *Source Title*

```
<blockquote>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.</p>
```

```
  <footer>Someone famous in <cite title="Source Title">Source Title</cite></footer>
</blockquote>
```

Alternate displays

实例：

> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.
>
> Someone famous in *Source Title* —

```
<blockquote class="blockquote-reverse">
  ...
</blockquote>
```

列表

无序列表

排列顺序无关紧要的一列元素。

实例：

- Lorem ipsum dolor sit amet
- Consectetur adipiscing elit
- Integer molestie lorem at massa
- Facilisis in pretium nisl aliquet
- Nulla volutpat aliquam velit
  - Phasellus iaculis neque
  - Purus sodales ultricies
  - Vestibulum laoreet porttitor sem
  - Ac tristique libero volutpat at
- Faucibus porta lacus fringilla vel
- Aenean sit amet erat nunc
- Eget porttitor lorem

```
<ul>
  <li>...</li>
</ul>
```

有序列表

顺序至关重要的一组元素。

实例：

1. Lorem ipsum dolor sit amet
2. Consectetur adipiscing elit
3. Integer molestie lorem at massa
4. Facilisis in pretium nisl aliquet
5. Nulla volutpat aliquam velit
6. Faucibus porta lacus fringilla vel
7. Aenean sit amet erat nunc
8. Eget porttitor lorem

```
<ol>
```

```
  <li>...</li>
</ol>
```

无样式列表

移除了默认的 list-style 样式和左侧外边距的一组元素（只针对直接子元素）。这是针对直接子元素的，也就是说，你需要对所有嵌套的列表都添加这个类才能具有同样的样式。

实例：

Lorem ipsum dolor sit amet
Consectetur adipiscing elit
Integer molestie lorem at massa
Facilisis in pretium nisl aliquet
Nulla volutpat aliquam velit
- Phasellus iaculis neque
- Purus sodales ultricies
- Vestibulum laoreet porttitor sem
- Ac tristique libero volutpat at

Faucibus porta lacus fringilla vel
Aenean sit amet erat nunc
Eget porttitor lorem

```
<ul class="list-unstyled">
  <li>...</li>
</ul>
```

内联列表

通过设置 display: inline-block; 并添加少量的内补（padding），将所有元素放置于同一行。

实例：

Lorem ipsum Phasellus iaculis Nulla volutpat

```
<ul class="list-inline">
  <li>...</li>
</ul>
```

描述

带有描述的短语列表。

实例：

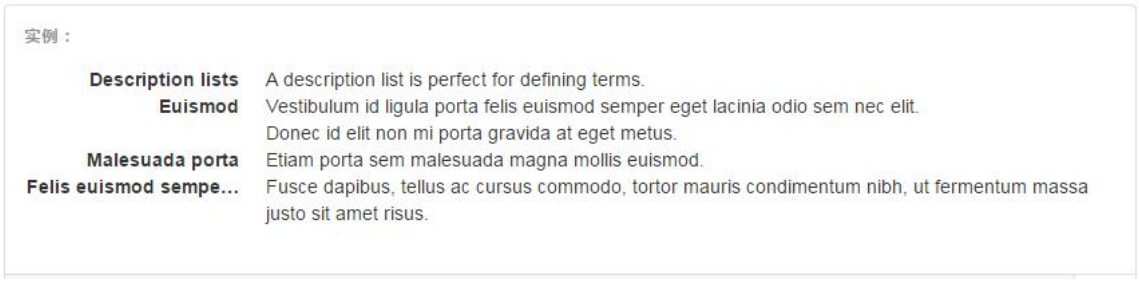
Description lists
A description list is perfect for defining terms.
Euismod
Vestibulum id ligula porta felis euismod semper eget lacinia odio sem nec elit.
Donec id elit non mi porta gravida at eget metus.
Malesuada porta
Etiam porta sem malesuada magna mollis euismod.

```
<dl>
```

```
  <dt>...</dt>
  <dd>...</dd>
</dl>
```

水平排列的描述

.dl-horizontal 可以让 <dl> 内的短语及其描述排在一行。开始是像 <dl> 的默认样式堆叠在一起，随着导航条逐渐展开而排列在一行。




```
<dl class="dl-horizontal">
  <dt>...</dt>
  <dd>...</dd>
</dl>
```

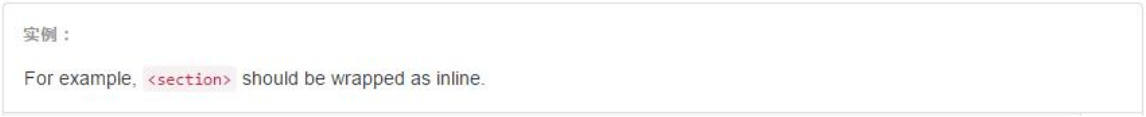
自动截断

通过 text-overflow 属性，水平排列的描述列表将会截断左侧太长的短语。在较窄的视口（viewport）内，列表将变为默认堆叠排列的布局方式。

四、代码

内联代码

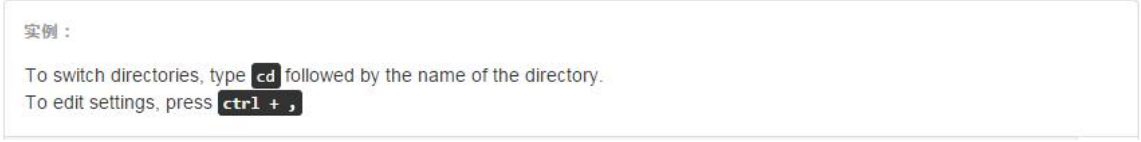
通过 <code> 标签包裹内联样式的代码片段。




```
For example, <code>&lt;section&gt;</code> should be wrapped as inline.
```

用户输入

通过 <kbd> 标签标记用户通过键盘输入的内容。=




```
To switch directories, type <kbd>cd</kbd> followed by the name of the directory.<br>
```

```
To edit settings, press <kbd><kbd>ctrl</kbd> + <kbd>,</kbd></kbd>
```

代码块

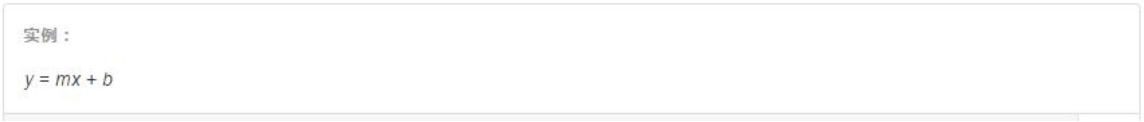
多行代码可以使用 <pre> 标签。为了正确的展示代码，注意将尖括号做转义处理。


```
<pre>&lt;p&gt;Sample text here...&lt;/p&gt;</pre>
```

还可以使用 .pre-scrollable 类，其作用是设置 max-height 为 350px ，并在垂直方向展示滚动条。

变量

通过 <var> 标签标记变量。




```
<var>y</var> = <var>m</var><var>x</var> + <var>b</var>
```

程序输出

通过 <samp> 标签来标记程序输出的内容。


```
<samp>This text is meant to be treated as sample output from a computer program.</samp>
```

五、表格

基本实例

为任意 <table> 标签添加 .table 类可以为其赋予基本的样式 — 少量的内补（padding）和水平方向的分隔线。这种方式看起来很多余！？但是我们觉得，表格元素使用的很广泛，如果我们为其赋予默认样式可能会影响例如日历和日期选择之类的插件，所以我们选择将此样式独立出来。

实例：

Optional table caption.

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

```
<table class="table">
  ...
</table>
```

条纹状表格

通过 .table-striped 类可以给 <tbody> 之内的每一行增加斑马条纹样式。

> 跨浏览器兼容性
> 条纹状表格是依赖 `:nth-child` CSS 选择器实现的，而这一功能不被 Internet Explorer 8 支持。

实例：

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

```
<table class="table table-striped">
  ...
</table>
```

带边框的表格

添加 .table-bordered 类为表格和其中的每个单元格增加边框。

实例：

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
	Mark	Otto	@TwBootstrap
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

```
<table class="table table-bordered">
```

```
  ...
</table>
```

鼠标悬停

通过添加 .table-hover 类可以让 <tbody> 中的每一行对鼠标悬停状态作出响应。

实例：

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

```
<table class="table table-hover">
  ...
</table>
```

紧缩表格

通过添加 .table-condensed 类可以让表格更加紧凑，单元格中的内补（padding）均会减半。

实例：

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

```
<table class="table table-condensed">
  ...
</table>
```

状态类

通过这些状态类可以为行或单元格设置颜色。

Class	描述
.active	鼠标悬停在行或单元格上时所设置的颜色
.success	标识成功或积极的动作
.info	标识普通的提示信息或动作
.warning	标识警告或需要用户注意
.danger	标识危险或潜在的带来负面影响的动作

实例：

#	Column heading	Column heading	Column heading
1	Column content	Column content	Column content
2	Column content	Column content	Column content
3	Column content	Column content	Column content
4	Column content	Column content	Column content
5	Column content	Column content	Column content
6	Column content	Column content	Column content
7	Column content	Column content	Column content
8	Column content	Column content	Column content
9	Column content	Column content	Column content

```
<!-- On rows -->
<tr class="active">...</tr>
<tr class="success">...</tr>
<tr class="warning">...</tr>
<tr class="danger">...</tr>
<tr class="info">...</tr>

<!-- On cells (`td` or `th`) -->
<tr>
  <td class="active">...</td>
  <td class="success">...</td>
  <td class="warning">...</td>
  <td class="danger">...</td>
  <td class="info">...</td>
</tr>
```

响应式表格

将任何 .table 元素包裹在 .table-responsive 元素内，即可创建响应式表格，其会在小屏幕设备上（小于 768px）水平滚动。当屏幕大于 768px 宽度时，水平滚动条消失。

Firefox 和 `fieldset` 元素

Firefox 浏览器对 `fieldset` 元素设置了一些影响 `width` 属性的样式，导致响应式表格出现问题。除非使用我们下面提供的针对 Firefox 的 hack 代码，否则无解：

```
@-moz-document url-prefix() {
  fieldset { display: table-cell; }
}
```

Copy

更多信息请参考 this Stack Overflow answer.

实例：

#	Table heading	Table heading	Table heading	Table heading	Table heading	Table heading
1	Table cell	Table cell	Table cell	Table cell	Table cell	Table cell
2	Table cell	Table cell	Table cell	Table cell	Table cell	Table cell
3	Table cell	Table cell	Table cell	Table cell	Table cell	Table cell

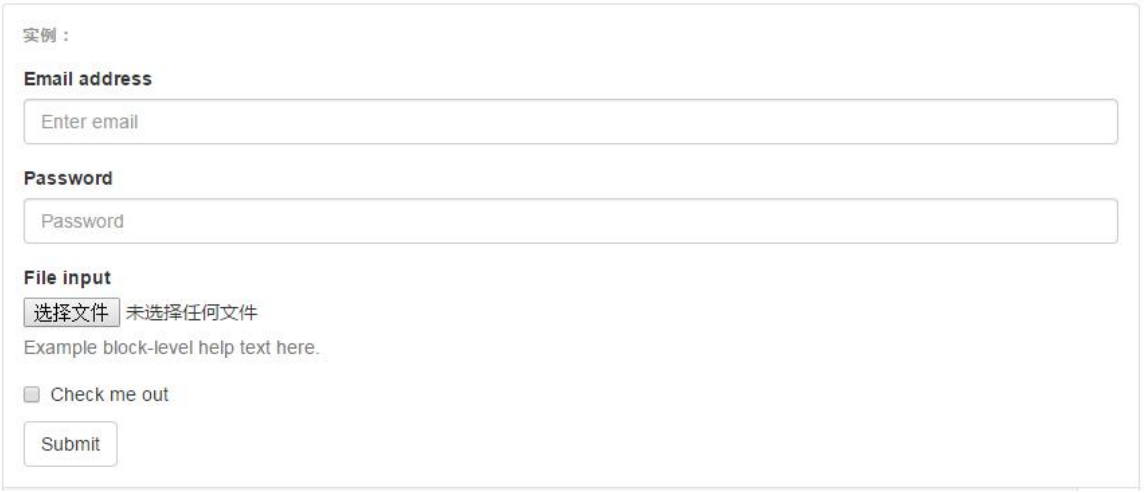
#	Table heading	Table heading	Table heading	Table heading	Table heading	Table heading
1	Table cell	Table cell	Table cell	Table cell	Table cell	Table cell
2	Table cell	Table cell	Table cell	Table cell	Table cell	Table cell
3	Table cell	Table cell	Table cell	Table cell	Table cell	Table cell

```
<div class="table-responsive">
  <table class="table">
    ...
  </table>
</div>
```

六、表单

基本实例

单独的表单控件会被自动赋予一些全局样式。所有设置了 .form-control 类的 <input>、<textarea> 和 <select> 元素都将被默认设置宽度属性为 width: 100%;。 将 label 元素和前面提到的控件包裹在 .form-group 中可以获得最好的排列。




```
<form role="form">
  <div class="form-group">
    <label for="exampleInputEmail1">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail1" placeholder="Enter email">
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">Password</label>
    <input type="password" class="form-control" id="exampleInputPassword1" placeholder="Password">
  </div>
  <div class="form-group">
    <label for="exampleInputFile">File input</label>
    <input type="file" id="exampleInputFile">
    <p class="help-block">Example block-level help text here.</p>
  </div>
  <div class="checkbox">
    <label>
      <input type="checkbox"> Check me out
    </label>
  </div>
  <button type="submit" class="btn btn-default">Submit</button>
</form>
```

不要将表单组合输入框组混合使用

不要将表单组直接和输入框组混合使用。建议将输入框组嵌套到表单组中使用。

内联表单

为 <form> 元素添加 .form-inline 类可使其内容左对齐并且表现为 inline-block 级别的控件。只适用于视口（viewport）至少在 768px 宽度时（视口宽度再小的话就会使表单折叠）。

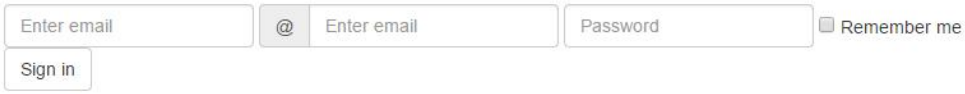
需要手动设置宽度

在 Bootstrap 中，输入框和单选/多选框控件默认被设置为 `width: 100%;` 宽度。在内联表单，我们将这些元素的宽度设置为 `width: auto;` ，因此，多个控件可以排列在同一行。根据你的布局需求，可能需要一些额外的定制化组件。

一定要添加 `label` 标签

如果你没有为每个输入控件设置 `label` 标签，屏幕阅读器将无法正确识别。对于这些内联表单，你可以通过为 `label` 设置 `.sr-only` 类将其隐藏。

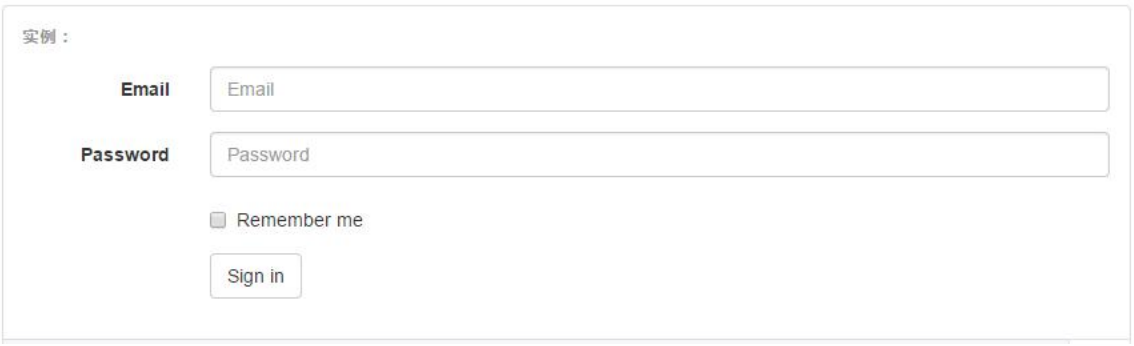
实例：




```
<form class="form-inline" role="form">
  <div class="form-group">
    <label class="sr-only" for="exampleInputEmail2">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail2" placeholder="Enter email">
  </div>
  <div class="form-group">
    <div class="input-group">
      <div class="input-group-addon">@</div>
      <input class="form-control" type="email" placeholder="Enter email">
    </div>
  </div>
  <div class="form-group">
    <label class="sr-only" for="exampleInputPassword2">Password</label>
    <input type="password" class="form-control" id="exampleInputPassword2" placeholder="Password">
  </div>
  <div class="checkbox">
    <label>
      <input type="checkbox"> Remember me
    </label>
  </div>
  <button type="submit" class="btn btn-default">Sign in</button>
</form>
```

水平排列的表单

通过为表单添加 .form-horizontal 类，并联合使用 Bootstrap 预置的栅格类，可以将 label 标签和控件组水平并排布局。这样做将改变 .form-group 的行为，使其表现为栅格系统中的行（row），因此就无需再额外添加 .row 了。




```
<form class="form-horizontal" role="form">
  <div class="form-group">
    <label for="inputEmail3" class="col-sm-2 control-label">Email</label>
    <div class="col-sm-10">
      <input type="email" class="form-control" id="inputEmail3" placeholder="Email">
    </div>
  </div>
  <div class="form-group">
    <label for="inputPassword3" class="col-sm-2 control-label">Password</label>
    <div class="col-sm-10">
      <input type="password" class="form-control" id="inputPassword3"
placeholder="Password">
    </div>
  </div>
  <div class="form-group">
    <div class="col-sm-offset-2 col-sm-10">
      <div class="checkbox">
        <label>
          <input type="checkbox"> Remember me
        </label>
      </div>
    </div>
  </div>
  <div class="form-group">
    <div class="col-sm-offset-2 col-sm-10">
      <button type="submit" class="btn btn-default">Sign in</button>
    </div>
  </div>
</form>
```

被支持的控件

表单布局实例中展示了其所支持的标准表单控件。

输入框

`text`、`password`、`datetime`、`datetime-local`、`date`、`month`、`time`、`week`、`number`、`email`、`url`、`search`、`tel` 和 `color`。

> **必须添加类型声明**
> 只有正确设置了 `type` 属性的输入控件才能被赋予正确的样式。

实例：

Text input

```
<input type="text" class="form-control" placeholder="Text input">
```

输入控件组

如需在文本输入域 <input> 前面或后面添加文本内容或按钮控件，请参考输入控件组。

文本域

支持多行文本的表单控件。可根据需要改变 rows 属性。

实例：

```
<textarea class="form-control" rows="3"></textarea>
```

多选和单选框

多选框（checkbox）用于选择列表中的一个或多个选项，而单选框（radio）用于从多个选项中只选择一个。

设置了 disabled 属性的单选或多选框都能被赋予合适的样式。对于和多选或单选框联合使用的 <label> 标签，如果也希望将悬停于上方的鼠标设置为“禁止点击”的样式，请将 .disabled 类赋予 .radio、.radio-inline、.checkbox、.checkbox-inline 或 <fieldset>。

默认外观（堆叠在一起）

实例：

- [ ] Option one is this and that—be sure to include why it's great
- [ ] Option two is disabled

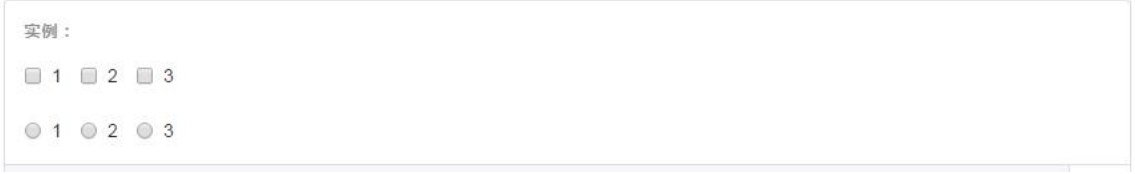
- (•) Option one is this and that—be sure to include why it's great
- ( ) Option two can be something else and selecting it will deselect option one
- ( ) Option three is disabled

```
<div class="checkbox">
  <label>
    <input type="checkbox" value="">
```

```
      Option one is this and that&mdash;be sure to include why it's great
    </label>
</div>
<div class="checkbox disabled">
    <label>
      <input type="checkbox" value="" disabled>
      Option two is disabled
    </label>
</div>

<div class="radio">
    <label>
      <input type="radio" name="optionsRadios" id="optionsRadios1" value="option1" checked>
      Option one is this and that&mdash;be sure to include why it's great
    </label>
</div>
<div class="radio">
    <label>
      <input type="radio" name="optionsRadios" id="optionsRadios2" value="option2">
      Option two can be something else and selecting it will deselect option one
    </label>
</div>
<div class="radio disabled">
    <label>
      <input type="radio" name="optionsRadios" id="optionsRadios3" value="option3" disabled>
      Option three is disabled
    </label>
</div>
```

内联单选和多选框

通过将 .checkbox-inline 或 .radio-inline 类应用到一系列的多选框（checkbox）或单选框（radio）控件上，可以使这些控件排列在一行。

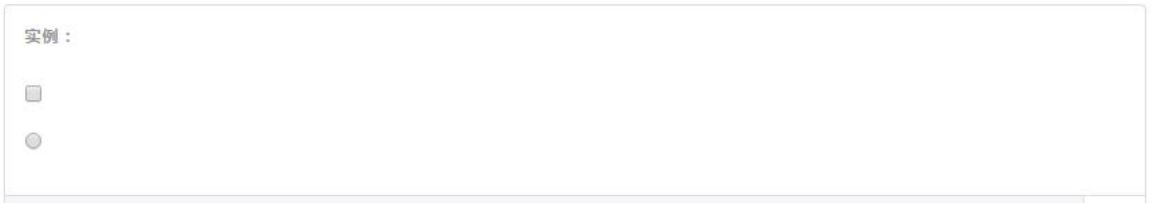



```
<label class="checkbox-inline">
    <input type="checkbox" id="inlineCheckbox1" value="option1"> 1
</label>
<label class="checkbox-inline">
    <input type="checkbox" id="inlineCheckbox2" value="option2"> 2
```

```
</label>
<label class="checkbox-inline">
  <input type="checkbox" id="inlineCheckbox3" value="option3"> 3
</label>

<label class="radio-inline">
  <input type="radio" name="inlineRadioOptions" id="inlineRadio1" value="option1"> 1
</label>
<label class="radio-inline">
  <input type="radio" name="inlineRadioOptions" id="inlineRadio2" value="option2"> 2
</label>
<label class="radio-inline">
  <input type="radio" name="inlineRadioOptions" id="inlineRadio3" value="option3"> 3
</label>
```

Checkboxes and radios without labels

Should you have no text within the <label>, the input is positioned as you'd expect. Currently only works on non-inline checkboxes and radios.




```
<div class="checkbox">
  <label>
    <input type="checkbox" id="blankCheckbox" value="option1">
  </label>
</div>
<div class="radio">
  <label>
    <input type="radio" name="blankRadio" id="blankRadio1" value="option1">
  </label>
</div>
```

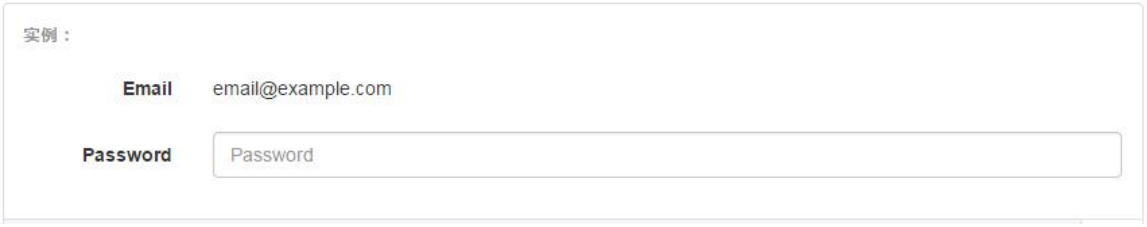
下拉列表（select）

使用默认选项或添加 multiple 属性可以同时显示多个选项。


```
<select class="form-control">
  <option>1</option>
  <option>2</option>
  <option>3</option>
  <option>4</option>
  <option>5</option>
</select>

<select multiple class="form-control">
  <option>1</option>
  <option>2</option>
  <option>3</option>
  <option>4</option>
  <option>5</option>
</select>
```

静态控件

如果需要在表单中将一行纯文本和 label 元素放置于同一行，为 <p> 元素添加 .form-control-static 类即可。




```
<form class="form-horizontal" role="form">
  <div class="form-group">
    <label class="col-sm-2 control-label">Email</label>
    <div class="col-sm-10">
      <p class="form-control-static">email@example.com</p>
    </div>
  </div>
  <div class="form-group">
    <label for="inputPassword" class="col-sm-2 control-label">Password</label>
    <div class="col-sm-10">
      <input type="password" class="form-control" id="inputPassword"
placeholder="Password">
```

```
    </div>
  </div>
</form>
```

实例：

email@example.com Password Confirm identity

```
<form class="form-inline" role="form">
  <div class="form-group">
    <label class="sr-only">Email</label>
    <p class="form-control-static">email@example.com</p>
  </div>
  <div class="form-group">
    <label for="inputPassword2" class="sr-only">Password</label>
    <input type="password" class="form-control" id="inputPassword2"
placeholder="Password">
  </div>
  <button type="submit" class="btn btn-default">Confirm identity</button>
</form>
```

输入框焦点

我们将某些表单控件的默认 outline 样式移除，然后对 :focus 状态赋予 box-shadow 属性。

实例：

Demonstrative focus state

演示 :focus 状态

在本文档中，我们为上面实例中的输入框赋予了自定义的样式，用于演示 .form-control 元素的 :focus 状态。

被禁用的输入框

为输入框设置 disabled 属性可以防止用户输入，并能对外观做一些修改，使其更直观。

实例：

Disabled input here...

```
<input class="form-control" id="disabledInput" type="text" placeholder="Disabled input here..."
disabled>
```

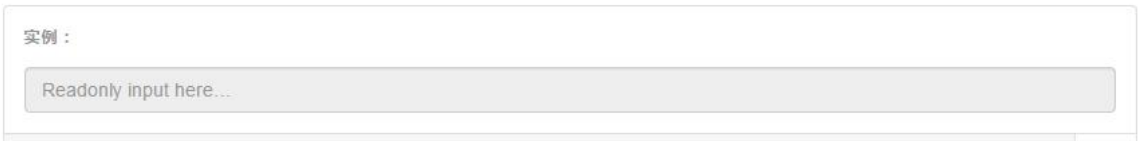
被禁用的 fieldset

为<fieldset> 设置 disabled 属性,可以禁用 <fieldset> 中包含的所有控件。


```
<form role="form">
  <fieldset disabled>
    <div class="form-group">
      <label for="disabledTextInput">Disabled input</label>
      <input type="text" id="disabledTextInput" class="form-control" placeholder="Disabled input">
    </div>
    <div class="form-group">
      <label for="disabledSelect">Disabled select menu</label>
      <select id="disabledSelect" class="form-control">
        <option>Disabled select</option>
      </select>
    </div>
    <div class="checkbox">
      <label>
        <input type="checkbox"> Can't check this
      </label>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
  </fieldset>
</form>
```

只读输入框

为输入框设置 readonly 属性可以禁止用户输入，并且输入框的样式也是禁用状态。




```
<input class="form-control" type="text" placeholder="Readonly input here…" readonly>
```

校验状态

Bootstrap 对表单控件的校验状态，如 error、warning 和 success 状态，都定义了样式。使用时，添加 .has-warning、.has-error 或 .has-success 类到这些控件的父元素即可。任何包含在此元素之内的 .control-label、.form-control 和 .help-block 元素都将接受这些校验状态的样式。


```
<div class="form-group has-success">
  <label class="control-label" for="inputSuccess1">Input with success</label>
  <input type="text" class="form-control" id="inputSuccess1">
</div>
<div class="form-group has-warning">
  <label class="control-label" for="inputWarning1">Input with warning</label>
  <input type="text" class="form-control" id="inputWarning1">
</div>
<div class="form-group has-error">
  <label class="control-label" for="inputError1">Input with error</label>
  <input type="text" class="form-control" id="inputError1">
</div>
<div class="has-success">
  <div class="checkbox">
    <label>
      <input type="checkbox" id="checkboxSuccess" value="option1">
      Checkbox with success
    </label>
  </div>
</div>
```

```
<div class="has-warning">
  <div class="checkbox">
    <label>
      <input type="checkbox" id="checkboxWarning" value="option1">
      Checkbox with warning
    </label>
  </div>
</div>
<div class="has-error">
  <div class="checkbox">
    <label>
      <input type="checkbox" id="checkboxError" value="option1">
      Checkbox with error
    </label>
  </div>
</div>
```

添加额外的图标

你还可以针对校验状态为输入框添加额外的图标。只需设置相应的 .has-feedback 类并添加正确的图标即可。

Feedback icons only work with textual <input class="form-control"> elements.

> 图标、 `label` 和输入控件组
>
> 对于不带有 `label` 标签的输入框以及右侧带有附加组件的输入框组，需要手动为其图标定位。为了让所有用户都能访问你的网站，我们强烈建议为所有输入框添加 `label` 标签。如果你不希望将 `label` 标签展示出来，可以通过添加 `sr-only` 类来实现。如果的确不能添加 `label` 标签，请调整图标的 `top` 值。对于输入框组，请根据你的实际情况调整 `right` 值。

实例：


```
<div class="form-group has-success has-feedback">
  <label class="control-label" for="inputSuccess2">Input with success</label>
  <input type="text" class="form-control" id="inputSuccess2">
  <span class="glyphicon glyphicon-ok form-control-feedback"></span>
</div>
<div class="form-group has-warning has-feedback">
```

```
  <label class="control-label" for="inputWarning2">Input with warning</label>
  <input type="text" class="form-control" id="inputWarning2">
  <span class="glyphicon glyphicon-warning-sign form-control-feedback"></span>
</div>
<div class="form-group has-error has-feedback">
  <label class="control-label" for="inputError2">Input with error</label>
  <input type="text" class="form-control" id="inputError2">
  <span class="glyphicon glyphicon-remove form-control-feedback"></span>
</div>
```

为水平排列的表单和内联表单设置可选的图标


```
<form class="form-horizontal" role="form">
  <div class="form-group has-success has-feedback">
    <label class="control-label col-sm-3" for="inputSuccess3">Input with success</label>
    <div class="col-sm-9">
      <input type="text" class="form-control" id="inputSuccess3">
      <span class="glyphicon glyphicon-ok form-control-feedback"></span>
    </div>
  </div>
</form>
```


```
<form class="form-inline" role="form">
  <div class="form-group has-success has-feedback">
    <label class="control-label" for="inputSuccess4">Input with success</label>
    <input type="text" class="form-control" id="inputSuccess4">
    <span class="glyphicon glyphicon-ok form-control-feedback"></span>
  </div>
</form>
```

可选的图标与设置 .sr-only 类的 label

通过为 label 元素添加 .sr-only 类，可以让表单控件的 label 元素不可见。在这种情况下，Bootstrap 将自动调整图标的位置。

```
<div class="form-group has-success has-feedback">
  <label class="control-label sr-only" for="inputSuccess5">Hidden label</label>
  <input type="text" class="form-control" id="inputSuccess5">
  <span class="glyphicon glyphicon-ok form-control-feedback"></span>
</div>
```

控件尺寸

通过 .input-lg 类似的类可以为控件设置高度，通过 .col-lg-* 类似的类可以为控件设置宽度。

高度尺寸

创建大一些或小一些的表单控件以匹配按钮尺寸。


```
<input class="form-control input-lg" type="text" placeholder=".input-lg">
<input class="form-control" type="text" placeholder="Default input">
<input class="form-control input-sm" type="text" placeholder=".input-sm">

<select class="form-control input-lg">...</select>
<select class="form-control">...</select>
<select class="form-control input-sm">...</select>
```

水平排列的表单组的尺寸

通过添加 .form-group-lg 或 .form-group-sm 类，为 .form-horizontal 包裹的 label 元素和表单控件快速设置尺寸。


```
<form class="form-horizontal" role="form">
  <div class="form-group form-group-lg">
    <label class="col-sm-2 control-label" for="formGroupInputLarge">Large label</label>
    <div class="col-sm-10">
      <input class="form-control" type="text" id="formGroupInputLarge" placeholder="Large input">
    </div>
  </div>
  <div class="form-group form-group-sm">
    <label class="col-sm-2 control-label" for="formGroupInputSmall">Small label</label>
    <div class="col-sm-10">
      <input class="form-control" type="text" id="formGroupInputSmall" placeholder="Small input">
    </div>
  </div>
</form>
```

调整列（column）尺寸

用栅格系统中的列（column）包裹输入框或其任何父元素，都可很容易的为其设置宽度。


```
<div class="row">
  <div class="col-xs-2">
    <input type="text" class="form-control" placeholder=".col-xs-2">
  </div>
  <div class="col-xs-3">
    <input type="text" class="form-control" placeholder=".col-xs-3">
  </div>
  <div class="col-xs-4">
    <input type="text" class="form-control" placeholder=".col-xs-4">
  </div>
</div>
```

辅助文本

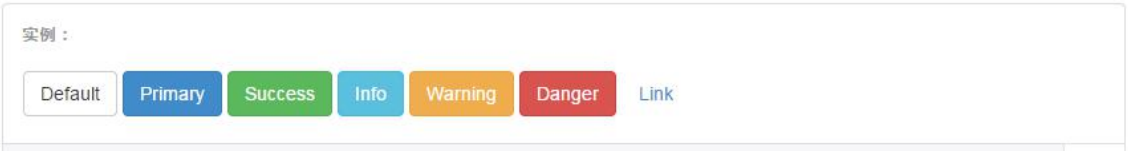
针对表单控件的“块（block）”级辅助文本。


```
<span class="help-block">A block of help text that breaks onto a new line and may extend beyond one line.</span>
```

七、按钮

预定义样式

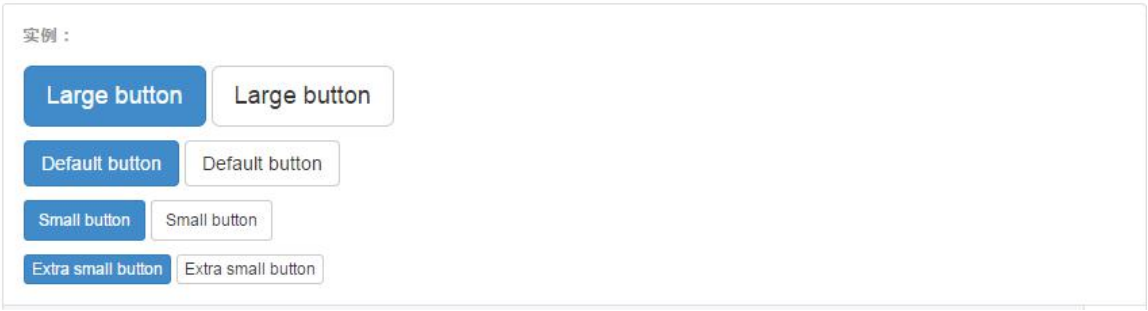
使用下面列出的类可以快速创建一个带有预定义样式的按钮。




```
<!-- Standard button -->
<button type="button" class="btn btn-default">Default</button>

<!-- Provides extra visual weight and identifies the primary action in a set of buttons -->
<button type="button" class="btn btn-primary">Primary</button>

<!-- Indicates a successful or positive action -->
<button type="button" class="btn btn-success">Success</button>

<!-- Contextual button for informational alert messages -->
<button type="button" class="btn btn-info">Info</button>

<!-- Indicates caution should be taken with this action -->
<button type="button" class="btn btn-warning">Warning</button>

<!-- Indicates a dangerous or potentially negative action -->
<button type="button" class="btn btn-danger">Danger</button>

<!-- Deemphasize a button by making it look like a link while maintaining button behavior -->
<button type="button" class="btn btn-link">Link</button>
```

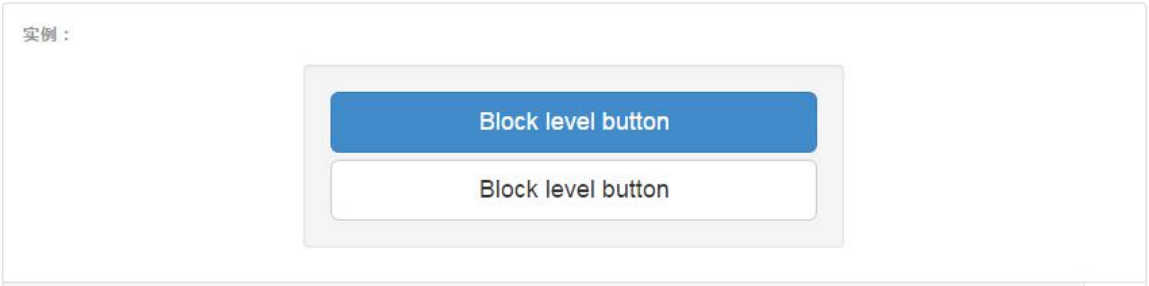
尺寸

需要让按钮具有不同尺寸吗？使用 .btn-lg、.btn-sm 或 .btn-xs 可以获得不同尺寸的按钮。




```
<p>
  <button type="button" class="btn btn-primary btn-lg">Large button</button>
  <button type="button" class="btn btn-default btn-lg">Large button</button>
</p>
<p>
  <button type="button" class="btn btn-primary">Default button</button>
  <button type="button" class="btn btn-default">Default button</button>
</p>
<p>
  <button type="button" class="btn btn-primary btn-sm">Small button</button>
  <button type="button" class="btn btn-default btn-sm">Small button</button>
</p>
<p>
  <button type="button" class="btn btn-primary btn-xs">Extra small button</button>
  <button type="button" class="btn btn-default btn-xs">Extra small button</button>
</p>
```

通过给按钮添加 .btn-block 类可以将其拉伸至父元素 100%的宽度，而且按钮也变为了块级（block）元素。



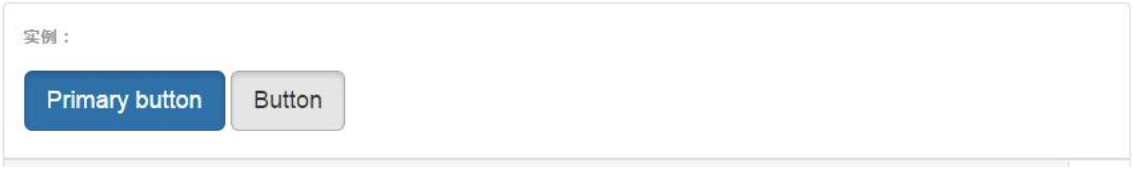

```
<button type="button" class="btn btn-primary btn-lg btn-block">Block level button</button>
<button type="button" class="btn btn-default btn-lg btn-block">Block level button</button>
```

激活状态

当按钮处于激活状态时，其表现为被按压下去（底色更深、边框夜色更深、向内投射阴影）。对于 <button> 元素，是通过 :active 状态实现的。对于 <a> 元素，是通过 .active 类实现的。然而，你还可以将 .active 应用到 <button> 上，并通过编程的方式使其处于激活状态。

button 元素

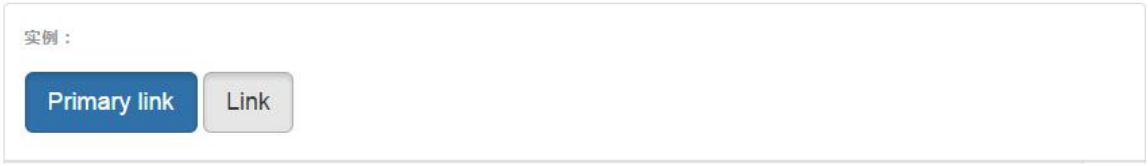
由于 :active 是伪状态，因此无需额外添加，但是在需要让其表现出同样外观的时候可以添加 .active 类。




```
<button type="button" class="btn btn-primary btn-lg active">Primary button</button>
<button type="button" class="btn btn-default btn-lg active">Button</button>
```

链接（<a>）元素

可以为基于 <a> 元素创建的按钮添加 .active 类。



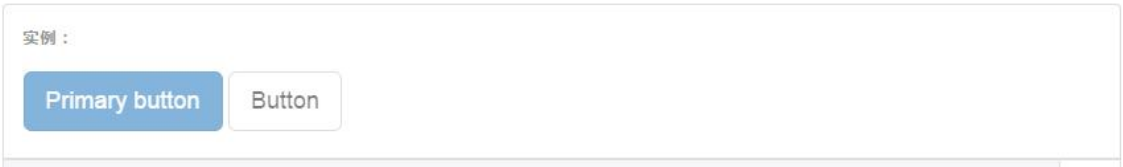

```
<a href="#" class="btn btn-primary btn-lg active" role="button">Primary link</a>
<a href="#" class="btn btn-default btn-lg active" role="button">Link</a>
```

禁用状态

通过为按钮的背景设置 opacity 属性就可以呈现出无法点击的效果。

button 元素

为 <button> 元素添加 disabled 属性，使其表现出禁用状态。



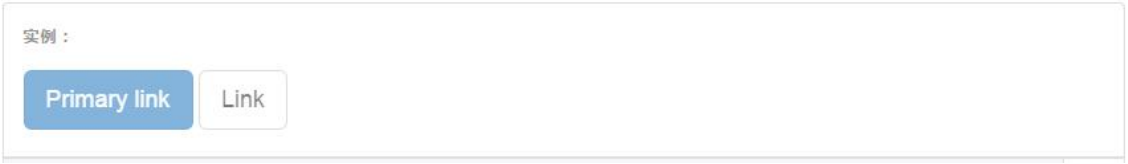

```
<button type="button" class="btn btn-lg btn-primary" disabled="disabled">Primary button</button>
<button type="button" class="btn btn-default btn-lg" disabled="disabled">Button</button>
```

跨浏览器兼容性

如果为 <button> 元素添加 disabled 属性，Internet Explorer 9 及更低版本的浏览器将会把按钮中的文本绘制为灰色，并带有恶心的阴影，目前我们还没有解决办法。

链接（<a>）元素

为基于 <a> 元素创建的按钮添加 .disabled 类。




```
<a href="#" class="btn btn-primary btn-lg disabled" role="button">Primary link</a>
<a href="#" class="btn btn-default btn-lg disabled" role="button">Link</a>
```

我们把 .disabled 作为工具类使用，就像 .active 类一样，因此不需要增加前缀。

链接的原始功能不受影响

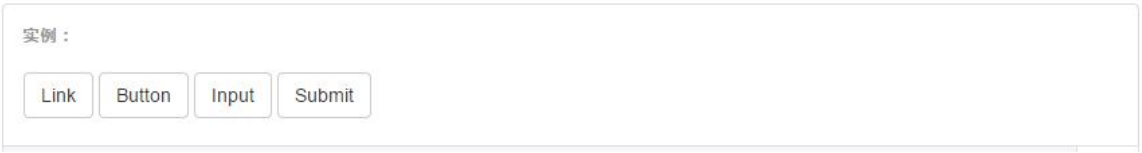
上面提到的类只是通过设置 pointer-events: none 来禁止 <a> 元素作为链接的原始功能，但是，这一 CSS 属性并没有被标准化，并且 Opera 18 及更低版本的浏览器并没有完全支持这一属性，同样，Internet Explorer 11 也不支持。因此，为了安全起见，建议通过 JavaScript 代码来禁止链接的原始功能。

Context-specific usage

虽然按钮类可以应用到 <a> 和 <button> 元素上，但是，导航和导航条只支持 <button> 元素。

按钮类

为 <a>、<button> 或 <input> 元素应用按钮类。




```
<a class="btn btn-default" href="#" role="button">Link</a>
<button class="btn btn-default" type="submit">Button</button>
<input class="btn btn-default" type="button" value="Input">
<input class="btn btn-default" type="submit" value="Submit">
```

跨浏览器展现

我们总结的最佳实践是，强烈建议尽可能使用 <button> 元素来获得在各个浏览器上获得相匹配的绘制效果。

另外，我们还发现了 a bug in Firefox <30 版本的浏览器上出现的一个 bug ：阻止我们为基于 <input> 元素创建的按钮设置 line-height 属性，这就导致在 Firefox 浏览器上不能完全和其他按钮保持一致的高度。

八、图片

响应式图片

在 Bootstrap 版本 3 中，通过为图片添加 .img-responsive 类可以让图片支持响应式布局。

其实质是为图片设置了 max-width: 100%; 和 height: auto; 属性，从而让图片在其父元素中更好的缩放。

> **SVG 图像和 IE 8-10**
>
> 在 Internet Explorer 8-10 中，设置为 `.img-responsive` 的 SVG 图像显示出的尺寸不匀称。为了解决这个问题，在出问题的地方添加 `width: 100% \9;` 即可。Bootstrap 并没有自动为所有图像元素设置这一属性，因为这会导致其他图像格式出现错乱。

```
<img src="..." class="img-responsive" alt="Responsive image">
```

图片形状

通过为 <img> 元素添加以下相应的类，可以让图片呈现不同的形状。

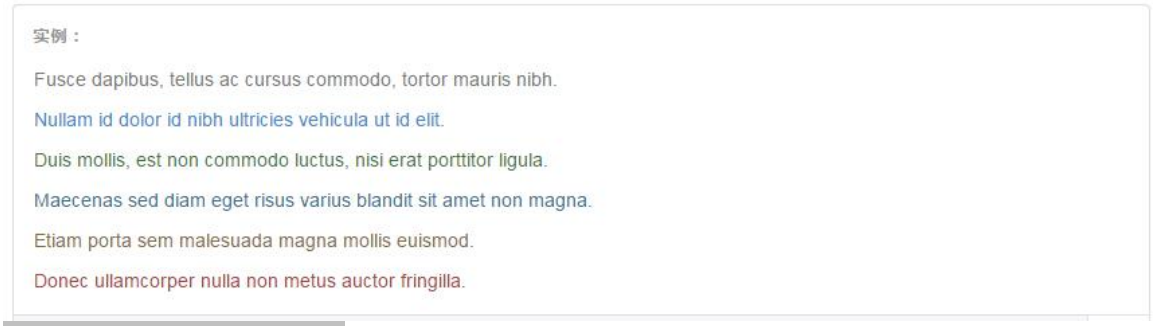
> **跨浏览器兼容性**
>
> 请时刻牢记：Internet Explorer 8 不支持 CSS3 中的圆角属性。


```
<img src="..." alt="..." class="img-rounded">
<img src="..." alt="..." class="img-circle">
<img src="..." alt="..." class="img-thumbnail">
```

九、辅助类

Contextual colors

Convey meaning through color with a handful of emphasis utility classes. These may also be applied to links and will darken on hover just like our default link styles.



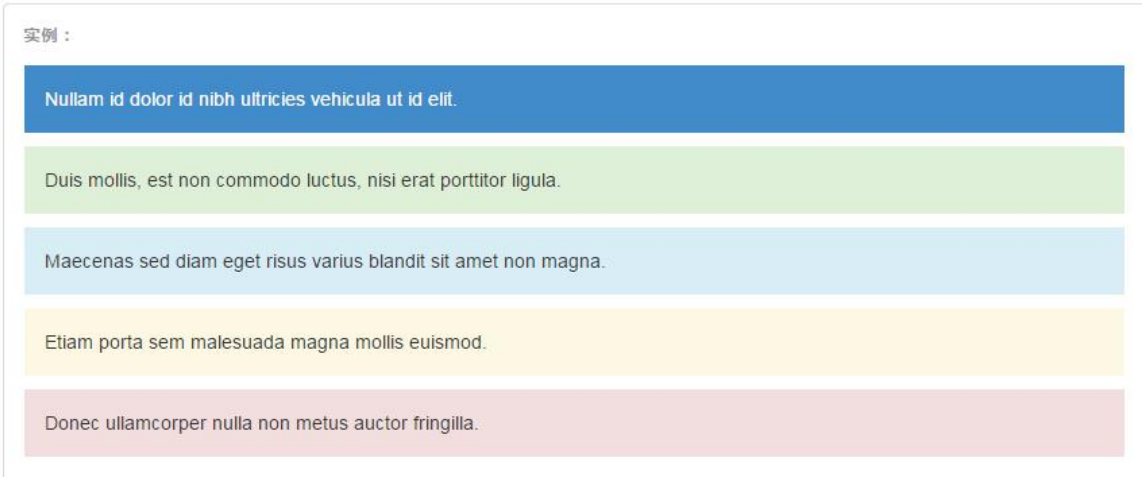

```
<p class="text-muted">...</p>
<p class="text-primary">...</p>
<p class="text-success">...</p>
<p class="text-info">...</p>
<p class="text-warning">...</p>
<p class="text-danger">...</p>
```

Dealing with specificity

Sometimes emphasis classes cannot be applied due to the specificity of another selector. In most cases, a sufficient workaround is to wrap your text in a with the class.

Contextual backgrounds

Similar to the contextual text color classes, easily set the background of an element to any contextual class. Anchor components will darken on hover, just like the text classes.



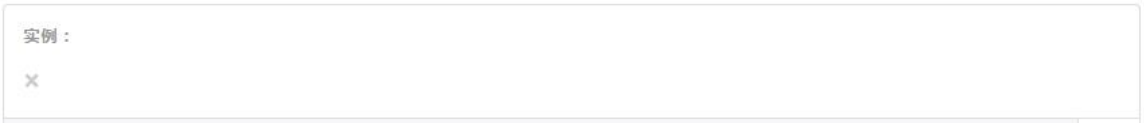

```
<p class="bg-primary">...</p>
<p class="bg-success">...</p>
<p class="bg-info">...</p>
<p class="bg-warning">...</p>
<p class="bg-danger">...</p>
```

Dealing with specificity

Sometimes contextual background classes cannot be applied due to the specificity of another selector. In some cases, a sufficient workaround is to wrap your element's content in a with the class.

关闭按钮

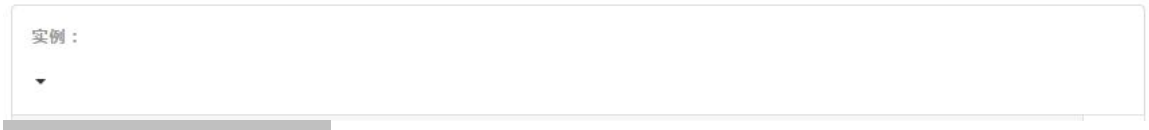
通过使用一个象征关闭的图标，可以让模态框和警告框消失。




```
<button type="button" class="close"><span aria-hidden="true">&times;</span><span class="sr-only">Close</span></button>
```

三角符号

通过使用三角符号可以指示某个元素具有下拉菜单的功能。注意，向上弹出式菜单中的三角符号是反方向的。




```
<span class="caret"></span>
```

快速浮动

Float an element to the left or right with a class. !important is included to avoid specificity issues. Classes can also be used as mixins.

```
<div class="pull-left">...</div>
<div class="pull-right">...</div>
// Classes
.pull-left {
  float: left !important;
}
.pull-right {
  float: right !important;
}

// Usage as mixins
.element {
  .pull-left();
}
.another-element {
  .pull-right();
}
```

Not for use in navbars

To align components in navbars with utility classes, use .navbar-left or .navbar-right instead. See the navbar docs for details.

Center content blocks

Set an element to display: block and center via margin. Available as a mixin and class.

```
<div class="center-block">...</div>

// Classes
.center-block {
  display: block;
  margin-left: auto;
  margin-right: auto;
}

// Usage as mixins
.element {
  .center-block();
}
```

Clearfix

Easily clear floats by adding .clearfix to the parent element. Utilizes the micro clearfix as popularized by Nicolas Gallagher. Can also be used as a mixin.

```
<!-- Usage as a class -->
<div class="clearfix">...</div>

// Mixin itself
.clearfix() {
  &:before,
  &:after {
      content: " ";
      display: table;
  }
  &:after {
      clear: both;
  }
}

// Usage as a Mixin
.element {
  .clearfix();
}
```

Showing and hiding content

Force an element to be shown or hidden (including for screen readers) with the use of .show and .hidden classes. These classes use !important to avoid specificity conflicts, just like the quick floats. They are only available for block level toggling. They can also be used as mixins.

.hide is available, but it does not always affect screen readers and is deprecated as of v3.0.1. Use .hidden or .sr-only instead.

Furthermore, .invisible can be used to toggle only the visibility of an element, meaning its display is not modified and the element can still affect the flow of the document.

```
<div class="show">...</div>
<div class="hidden">...</div>

// Classes
.show {
  display: block !important;
}
.hidden {
  display: none !important;
  visibility: hidden !important;
}
.invisible {
  visibility: hidden;
}

// Usage as mixins
.element {
  .show();
}
.another-element {
  .hidden();
}
```

Screen reader and keyboard navigation content

Hide an element to all devices except screen readers with .sr-only. Combine .sr-only with .sr-only-focusable to show the element again when it's focused (e.g. by a keyboard-only user). Necessary for following accessibility best practices. Can also be used as mixins.

```
<a class="sr-only sr-only-focusable" href="#content">Skip to main content</a>

// Usage as a Mixin
.skip-navigation {
  .sr-only();
  .sr-only-focusable();
}
```

Image replacement

Utilize the .text-hide class or mixin to help replace an element's text content with a background

image.

```
<h1 class="text-hide">Custom heading</h1>

// Usage as a Mixin
.heading {
    .text-hide();
}
```

十、响应式工具

为了加快对移动设备友好的页面开发工作，利用媒体查询功能并使用这些工具类可以方便的针对不同设备展示或隐藏页面内容。另外还包含了针对打印机显示或隐藏内容的工具类。

有针对性的使用这类工具类，从而避免为同一个网站创建完全不同的版本。相反，通过使用这些工具类可以在不同设备上提供不同的展现形式。

可用的类

通过单独或联合使用以下列出的类，可以针对不同屏幕尺寸隐藏或显示页面内容。

	超小屏幕 手机 (<768px)	小屏幕 平板 (≥768px)	中等屏幕 桌面 (≥992px)	大屏幕 桌面 (≥1200px)
.visible-xs-*	可见	隐藏	隐藏	隐藏
.visible-sm-*	隐藏	可见	隐藏	隐藏
.visible-md-*	隐藏	隐藏	可见	隐藏
.visible-lg-*	隐藏	隐藏	隐藏	可见
.hidden-xs	隐藏	可见	可见	可见
.hidden-sm	可见	隐藏	可见	可见
.hidden-md	可见	可见	隐藏	可见
.hidden-lg	可见	可见	可见	隐藏

从 v3.2.0 版本起，形如 .visible-*-* 的类针对每种屏幕大小都有了三种变体，每个针对 CSS 中不同的 display 属性，列表如下：

类组	CSS display
.visible-*-block	display: block;
.visible-*-inline	display: inline;
.visible-*-inline-block	display: inline-block;

因此，以超小屏幕（xs）为例，可用的 .visible-*-* 类是：.visible-xs-block、.visible-xs-inline 和 .visible-xs-inline-block。

.visible-xs、.visible-sm、.visible-md 和 .visible-lg 类也同时存在。但是从 v3.2.0 版本开始不再建议使用。除了 <table> 相关的元素的特殊情况外，它们与 .visible-*-block 大体相同。

打印类

和常规的响应式类一样，使用下面的类可以针对打印机隐藏或显示某些内容。

class	浏览器	打印机
.visible-print-block .visible-print-inline .visible-print-inline-block	隐藏	可见
.hidden-print	可见	隐藏

.visible-print 类也是存在的，但是从 v3.2.0 版本开始不建议使用。它与 .visible-print-block 类大致相同，除了 <table> 相关元素的特殊情况外。

测试用例

调整你的浏览器大小，或者用其他设备打开页面，都可以测试这些响应式工具类。

在...上可见

带有绿色标记的元素表示其在当前浏览器视口（viewport）中是可见的。

在...上隐藏

带有绿色标记的元素表示其在当前浏览器视口（viewport）中是隐藏的。

十一、使用 Less

Bootstrap 的 CSS 文件是通过 Less 源码编译出来的。Less 是一门预处理语言，支持变量、mixin、函数等额外功能。对于希望使用 Less 源码而非编译出来的 CSS 文件的用户，Bootstrap 框架中包含的大量变量、mixin 将非常有价值。

针对栅格系统的变量和 mixin 包含在栅格系统章节。

编译 Bootstrap

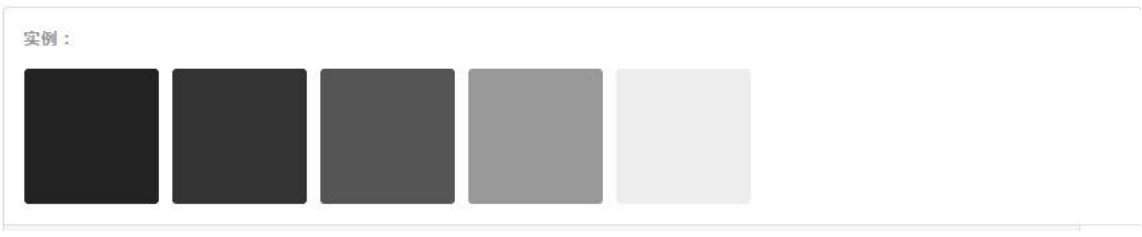
Bootstrap can be used in at least two ways: with the compiled CSS or with the source Less files. To compile the Less files, consult the Getting Started section for how to setup your development environment to run the necessary commands.

变量

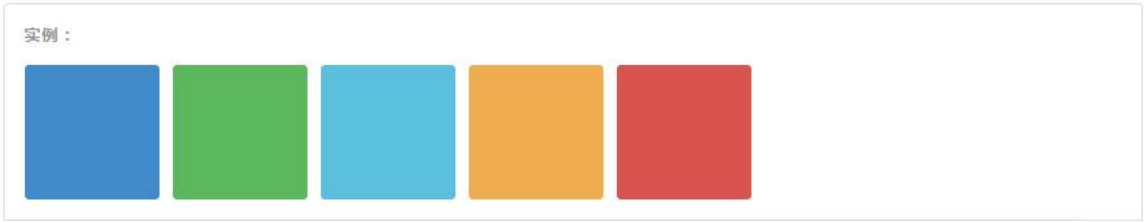
Variables are used throughout the entire project as a way to centralize and share commonly used values like colors, spacing, or font stacks. For a complete breakdown, please see the Customizer.

颜色

Easily make use of two color schemes: grayscale and semantic. Grayscale colors provide quick access to commonly used shades of black while semantic include various colors assigned to meaningful contextual values.



```
@gray-darker:    lighten(#000, 13.5%); // #222
@gray-dark:      lighten(#000, 20%);   // #333
@gray:           lighten(#000, 33.5%); // #555
@gray-light:     lighten(#000, 46.7%); // #777
@gray-lighter: lighten(#000, 93.5%); // #eee
```




```
@brand-primary: #428bca;
@brand-success: #5cb85c;
@brand-info:    #5bc0de;
@brand-warning: #f0ad4e;
@brand-danger:  #d9534f;
```

Use any of these color variables as they are or reassign them to more meaningful variables for your project.

```
// Use as-is
.masthead {
  background-color: @brand-primary;
}

// Reassigned variables in Less
@alert-message-background: @brand-info;
.alert {
  background-color: @alert-message-background;
}
```

Scaffolding

A handful of variables for quickly customizing key elements of your site's skeleton.

```
// Scaffolding
@body-bg:    #fff;
@text-color: @black-50;
```

Links

Easily style your links with the right color with only one value.

```
// Variables
@link-color:        @brand-primary;
@link-hover-color: darken(@link-color, 15%);
```

```
// Usage
a {
  color: @link-color;
  text-decoration: none;

  &:hover {
    color: @link-hover-color;
    text-decoration: underline;
  }
}
```

Note that the @link-hover-color uses a function, another awesome tool from Less, to automagically create the right hover color. You can use darken, lighten, saturate, and desaturate.

Typography

Easily set your type face, text size, leading, and more with a few quick variables. Bootstrap makes use of these as well to provide easy typographic mixins.

```
@font-family-sans-serif:  "Helvetica Neue", Helvetica, Arial, sans-serif;
@font-family-serif:       Georgia, "Times New Roman", Times, serif;
@font-family-monospace:   Menlo, Monaco, Consolas, "Courier New", monospace;
@font-family-base:        @font-family-sans-serif;

@font-size-base:          14px;
@font-size-large:         ceil((@font-size-base * 1.25)); // ~18px
@font-size-small:         ceil((@font-size-base * 0.85)); // ~12px

@font-size-h1:            floor((@font-size-base * 2.6)); // ~36px
@font-size-h2:            floor((@font-size-base * 2.15)); // ~30px
@font-size-h3:            ceil((@font-size-base * 1.7)); // ~24px
@font-size-h4:            ceil((@font-size-base * 1.25)); // ~18px
@font-size-h5:            @font-size-base;
@font-size-h6:            ceil((@font-size-base * 0.85)); // ~12px

@line-height-base:        1.428571429; // 20/14
@line-height-computed:    floor((@font-size-base * @line-height-base)); // ~20px

@headings-font-family:    inherit;
@headings-font-weight:    500;
@headings-line-height:    1.1;
@headings-color:          inherit;
```

Icons

Two quick variables for customizing the location and filename of your icons.

```
@icon-font-path:          "../fonts/";
@icon-font-name:          "glyphicons-halflings-regular";
```

Components

Components throughout Bootstrap make use of some default variables for setting common values. Here are the most commonly used.

```
@padding-base-vertical:          6px;
@padding-base-horizontal:        12px;

@padding-large-vertical:         10px;
@padding-large-horizontal:       16px;

@padding-small-vertical:         5px;
@padding-small-horizontal:       10px;

@padding-xs-vertical:            1px;
@padding-xs-horizontal:          5px;

@line-height-large:              1.33;
@line-height-small:              1.5;

@border-radius-base:             4px;
@border-radius-large:            6px;
@border-radius-small:            3px;

@component-active-color:         #fff;
@component-active-bg:            @brand-primary;

@caret-width-base:               4px;
@caret-width-large:              5px;
```

Vendor mixins

Vendor mixins are mixins to help support multiple browsers by including all relevant vendor prefixes in your compiled CSS.

Box-sizing

Reset your components' box model with a single mixin. For context, see this helpful article from Mozilla.

The mixin is deprecated as of v3.2.0, with the introduction of autoprefixer. To preserve backwards-compatibility, Bootstrap will continue to use the mixin internally until Bootstrap v4.

```
.box-sizing(@box-model) {
  -webkit-box-sizing: @box-model; // Safari <= 5
     -moz-box-sizing: @box-model; // Firefox <= 19
          box-sizing: @box-model;
}
```

Rounded corners

Today all modern browsers support the non-prefixed border-radius property. As such, there is no .border-radius() mixin, but Bootstrap does include shortcuts for quickly rounding two corners on a particular side of an object.

```
.border-top-radius(@radius) {
  border-top-right-radius: @radius;
    border-top-left-radius: @radius;
}
.border-right-radius(@radius) {
  border-bottom-right-radius: @radius;
     border-top-right-radius: @radius;
}
.border-bottom-radius(@radius) {
  border-bottom-right-radius: @radius;
    border-bottom-left-radius: @radius;
}
.border-left-radius(@radius) {
  border-bottom-left-radius: @radius;
     border-top-left-radius: @radius;
}
```

Box (Drop) shadows

If your target audience is using the latest and greatest browsers and devices, be sure to just use the box-shadow property on its own. If you need support for older Android (pre-v4) and iOS devices (pre-iOS 5), use the deprecated mixin to pick up the required -webkit prefix.

The mixin is deprecated as of v3.1.0, since Bootstrap doesn't officially support the outdated platforms that don't support the standard property. To preserve backwards-compatibility, Bootstrap will continue to use the mixin internally until Bootstrap v4.

Be sure to use rgba() colors in your box shadows so they blend as seamlessly as possible with backgrounds.

```
.box-shadow(@shadow: 0 1px 3px rgba(0,0,0,.25)) {
  -webkit-box-shadow: @shadow; // iOS <4.3 & Android <4.1
          box-shadow: @shadow;
}
```

Transitions

Multiple mixins for flexibility. Set all transition information with one, or specify a separate delay and duration as needed.

The mixins are deprecated as of v3.2.0, with the introduction of autoprefixer. To preserve backwards-compatibility, Bootstrap will continue to use the mixins internally until Bootstrap v4.

```
.transition(@transition) {
  -webkit-transition: @transition;
          transition: @transition;
}
.transition-property(@transition-property) {
  -webkit-transition-property: @transition-property;
          transition-property: @transition-property;
}
.transition-delay(@transition-delay) {
  -webkit-transition-delay: @transition-delay;
          transition-delay: @transition-delay;
}
.transition-duration(@transition-duration) {
  -webkit-transition-duration: @transition-duration;
          transition-duration: @transition-duration;
}
.transition-timing-function(@timing-function) {
  -webkit-transition-timing-function: @timing-function;
          transition-timing-function: @timing-function;
}
.transition-transform(@transition) {
  -webkit-transition: -webkit-transform @transition;
     -moz-transition: -moz-transform @transition;
       -o-transition: -o-transform @transition;
          transition: transform @transition;
}
```

Transformations

Rotate, scale, translate (move), or skew any object.

The mixins are deprecated as of v3.2.0, with the introduction of autoprefixer. To preserve backwards-compatibility, Bootstrap will continue to use the mixins internally until Bootstrap v4.

```
.rotate(@degrees) {
  -webkit-transform: rotate(@degrees);
      -ms-transform: rotate(@degrees); // IE9 only
          transform: rotate(@degrees);
}
.scale(@ratio; @ratio-y...) {
  -webkit-transform: scale(@ratio, @ratio-y);
      -ms-transform: scale(@ratio, @ratio-y); // IE9 only
          transform: scale(@ratio, @ratio-y);
}
.translate(@x; @y) {
  -webkit-transform: translate(@x, @y);
      -ms-transform: translate(@x, @y); // IE9 only
          transform: translate(@x, @y);
}
.skew(@x; @y) {
  -webkit-transform: skew(@x, @y);
      -ms-transform: skewX(@x) skewY(@y); // See
https://github.com/twbs/bootstrap/issues/4885; IE9+
          transform: skew(@x, @y);
}
.translate3d(@x; @y; @z) {
  -webkit-transform: translate3d(@x, @y, @z);
          transform: translate3d(@x, @y, @z);
}

.rotateX(@degrees) {
  -webkit-transform: rotateX(@degrees);
      -ms-transform: rotateX(@degrees); // IE9 only
          transform: rotateX(@degrees);
}
.rotateY(@degrees) {
  -webkit-transform: rotateY(@degrees);
      -ms-transform: rotateY(@degrees); // IE9 only
          transform: rotateY(@degrees);
}
.perspective(@perspective) {
  -webkit-perspective: @perspective;
     -moz-perspective: @perspective;
```

```
        perspective: @perspective;
}
.perspective-origin(@perspective) {
  -webkit-perspective-origin: @perspective;
     -moz-perspective-origin: @perspective;
        perspective-origin: @perspective;
}
.transform-origin(@origin) {
  -webkit-transform-origin: @origin;
     -moz-transform-origin: @origin;
      -ms-transform-origin: @origin; // IE9 only
        transform-origin: @origin;
}
```

Animations

A single mixin for using all of CSS3's animation properties in one declaration and other mixins for individual properties.

The mixins are deprecated as of v3.2.0, with the introduction of autoprefixer. To preserve backwards-compatibility, Bootstrap will continue to use the mixins internally until Bootstrap v4.

```
.animation(@animation) {
  -webkit-animation: @animation;
        animation: @animation;
}
.animation-name(@name) {
  -webkit-animation-name: @name;
        animation-name: @name;
}
.animation-duration(@duration) {
  -webkit-animation-duration: @duration;
        animation-duration: @duration;
}
.animation-timing-function(@timing-function) {
  -webkit-animation-timing-function: @timing-function;
        animation-timing-function: @timing-function;
}
.animation-delay(@delay) {
  -webkit-animation-delay: @delay;
        animation-delay: @delay;
}
.animation-iteration-count(@iteration-count) {
  -webkit-animation-iteration-count: @iteration-count;
```

```
        animation-iteration-count: @iteration-count;
}
.animation-direction(@direction) {
  -webkit-animation-direction: @direction;
        animation-direction: @direction;
}
```

Opacity

Set the opacity for all browsers and provide a filter fallback for IE8.

```
.opacity(@opacity) {
  opacity: @opacity;
  // IE8 filter
  @opacity-ie: (@opacity * 100);
  filter: ~"alpha(opacity=@{opacity-ie})";
}
```

Placeholder text

Provide context for form controls within each field.

```
.placeholder(@color: @input-color-placeholder) {
  &::-moz-placeholder           { color: @color; } // Firefox
  &:-ms-input-placeholder       { color: @color; } // Internet Explorer 10+
  &::-webkit-input-placeholder  { color: @color; } // Safari and Chrome
}
```

Columns

Generate columns via CSS within a single element.

```
.content-columns(@width; @count; @gap) {
  -webkit-column-width: @width;
     -moz-column-width: @width;
        column-width: @width;
  -webkit-column-count: @count;
     -moz-column-count: @count;
        column-count: @count;
  -webkit-column-gap: @gap;
     -moz-column-gap: @gap;
        column-gap: @gap;
}
```

Gradients

Easily turn any two colors into a background gradient. Get more advanced and set a direction, use three colors, or use a radial gradient. With a single mixin you get all the prefixed syntaxes you'll need.

```
#gradient > .vertical(#333; #000);
#gradient > .horizontal(#333; #000);
#gradient > .radial(#333; #000);
```

You can also specify the angle of a standard two-color, linear gradient:

```
#gradient > .directional(#333; #000; 45deg);
```

If you need a barber-stripe style gradient, that's easy, too. Just specify a single color and we'll overlay a translucent white stripe.

```
#gradient > .striped(#333; 45deg);
```

Up the ante and use three colors instead. Set the first color, the second color, the second color's color stop (a percentage value like 25%), and the third color with these mixins:

```
#gradient > .vertical-three-colors(#777; #333; 25%; #000);
#gradient > .horizontal-three-colors(#777; #333; 25%; #000);
```

Heads up! Should you ever need to remove a gradient, be sure to remove any IE-specific filter you may have added. You can do that by using the .reset-filter() mixin alongside background-image: none;.

Utility mixins

Utility mixins are mixins that combine otherwise unrelated CSS properties to achieve a specific goal or task.

Clearfix

Forget adding class="clearfix" to any element and instead add the .clearfix() mixin where appropriate. Uses the micro clearfix from Nicolas Gallager.

```
// Mixin
.clearfix() {
  &:before,
  &:after {
      content: " ";
      display: table;
  }
  &:after {
      clear: both;
```

```
    }
}

// Usage
.container {
    .clearfix();
}
```

Horizontal centering

Quickly center any element within its parent. Requires width or max-width to be set.

```
// Mixin
.center-block() {
    display: block;
    margin-left: auto;
    margin-right: auto;
}

// Usage
.container {
    width: 940px;
    .center-block();
}
```

Sizing helpers

Specify the dimensions of an object more easily.

```
// Mixins
.size(@width; @height) {
    width: @width;
    height: @height;
}
.square(@size) {
    .size(@size; @size);
}

// Usage
.image { .size(400px; 300px); }
.avatar { .square(48px); }
```

Resizable textareas

Easily configure the resize options for any textarea, or any other element. Defaults to normal browser behavior (both).

```
.resizable(@direction: both) {
  // Options: horizontal, vertical, both
  resize: @direction;
  // Safari fix
  overflow: auto;
}
```

Truncating text

Easily truncate text with an ellipsis with a single mixin. Requires element to be block or inline-block level.

```
// Mixin
.text-overflow() {
  overflow: hidden;
  text-overflow: ellipsis;
  white-space: nowrap;
}

// Usage
.branch-name {
  display: inline-block;
  max-width: 200px;
  .text-overflow();
}
```

Retina images

Specify two image paths and the @1x image dimensions, and Bootstrap will provide an @2x media query. If you have many images to serve, consider writing your retina image CSS manually in a single media query.

```
.img-retina(@file-1x; @file-2x; @width-1x; @height-1x) {
  background-image: url("@{file-1x}");

  @media
  only screen and (-webkit-min-device-pixel-ratio: 2),
  only screen and (      min--moz-device-pixel-ratio: 2),
  only screen and (          -o-min-device-pixel-ratio: 2/1),
  only screen and (                min-device-pixel-ratio: 2),
  only screen and (                                min-resolution: 192dpi),
  only screen and (                                min-resolution: 2dppx) {
    background-image: url("@{file-2x}");
    background-size: @width-1x @height-1x;
  }
```

```
}

// Usage
.jumbotron {
    .img-retina("/img/bg-1x.png", "/img/bg-2x.png", 100px, 100px);
}
```

十三、使用 Sass

虽然 Bootstrap 是基于 Less 构建的，我们还提供了一套官方支持的 Sass 移植版代码。我们将这个版本放在单独的 GitHub 仓库中进行维护，并通过脚本处理源码更新。

包含的内容

由于 Sass 移植版存放于单独的仓库，并针对不同的使用群体，这个项目中的内容与 Bootstrap 主项目有很大不同。这也是为了保证 Sass 移植版与更多基于 Sass 的系统相兼容。

路径	描述
`lib/`	Ruby gem code (Sass configuration, Rails and Compass integrations)
`tasks/`	Converter scripts (turning upstream Less to Sass)
`test/`	Compilation tests
`templates/`	Compass package manifest
`vendor/assets/`	Sass, JavaScript, and font files
`Rakefile`	Internal tasks, such as rake and convert

请访问 Sass 移植版在 GitHub 上的仓库 来了解这些文件。

安装

关于如何安装并使用 Bootstrap 的 Sass 移植版，请参考 GitHub 仓库中的 readme 文件。此仓库中包含了最新的源码以及如何与 Rails、Compass 以及标准 Sass 项目一同使用的详细信息。

Bootstrap for Sass

https://github.com/twbs/bootstrap-sass

JAVASCRIPT 语言规范

一、变量

▽ 声明变量必须加上 var 关键字.

Decision:

当你没有写 var, 变量就会暴露在全局上下文中, 这样很可能会和现有变量冲突. 另外, 如果没有加上, 很难明确该变量的作用域是什么, 变量也很可能像在局部作用域中, 很轻易地泄漏到 Document 或者 Window 中, 所以务必用 var 去声明变量.

二、常量

▽ 常量的形式如: NAMES_LIKE_THIS, 即使用大写字符, 并用下划线分隔. 你也可用 @const 标记来指明它是一个常量. 但请永远不要使用 const 关键词.

Decision:

对于基本类型的常量, 只需转换命名.

```
/**
 * The number of seconds in a minute.
 * @type {number}
 */
goog.example.SECONDS_IN_A_MINUTE = 60;
```

对于非基本类型, 使用 @const 标记.

```
/**
 * The number of seconds in each of the given units.
 * @type {Object.<number>}
 * @const
```

```
 */

goog.example.SECONDS_TABLE = {

  minute: 60,

  hour: 60 * 60,

  day: 60 * 60 * 24

}
```

这标记告诉编译器它是常量.

至于关键词 const, 因为 IE 不能识别, 所以不要使用.

三、分号

▽ 总是使用分号.

如果仅依靠语句间的隐式分隔, 有时会很麻烦. 你自己更能清楚哪里是语句的起止.

而且有些情况下, 漏掉分号会很危险:

```
// 1.

MyClass.prototype.myMethod = function() {

  return 42;

}  // No semicolon here.


(function() {

  // Some initialization code wrapped in a function to create a scope for locals.

})();
```

```
var x = {
  'i': 1,
  'j': 2
}  // No semicolon here.


// 2.  Trying to do one thing on Internet Explorer and another on Firefox.
// I know you'd never write code like this, but throw me a bone.
[normalVersion, ffVersion][isIE]();


var THINGS_TO_EAT = [apples, oysters, sprayOnCheese]  // No semicolon here.


// 3. conditional execution a la bash
-1 == resultOfOperation() || die();
```

这段代码会发生些什么诡异事呢?

报 JavaScript 错误 - 例子 1 上的语句会解释成, 一个函数带一匿名函数作为参数而被调用, 返回 42 后, 又一次被"调用", 这就导致了错误.

例子 2 中, 你很可能会在运行时遇到 'no such property in undefined' 错误, 原因是代码试图这样 x[ffVersion][isIE]() 执行.

当 resultOfOperation() 返回非 NaN 时, 就会调用 die, 其结果也会赋给 THINGS_TO_EAT.

为什么?

JavaScript 的语句以分号作为结束符, 除非可以非常准确推断某结束位置才会省略分号. 上面的几个例子产出错误, 均是在语句中声明了函数/对象/数组直

接量，但 闭括号('}'或']')并不足以表示该语句的结束. 在 JavaScript 中，只有当语句后的下一个符号是后缀或括号运算符时，才会认为该语句的结束.

遗漏分号有时会出现很奇怪的结果，所以确保语句以分号结束.

四、嵌套函数

▽可以使用

嵌套函数很有用，比如,减少重复代码，隐藏帮助函数，等. 没什么其他需要注意的地方，随意使用.

五、块内函数声明

▽不要在块内声明一个函数

不要写成:

```
if (x) {

  function foo() {}

}
```

虽然很多 JS 引擎都支持块内声明函数，但它不属于 ECMAScript 规范 (见 ECMA-262，第 13 和 14 条). 各个浏览器糟糕的实现相互不兼容，有些也与未来 ECMAScript 草案相违背. ECMAScript 只允许在脚本的根语句或函数中声明函数. 如果确实需要在块中定义函数，建议使用函数表达式来初始化变量:

```
if (x) {

  var foo = function() {}

}
```

六、异常

▽可以

你在写一个比较复杂的应用时，不可能完全避免不会发生任何异常. 大胆去用吧.

七、自定义异常

▽可以

有时发生异常了，但返回的错误信息比较奇怪，也不易读. 虽然可以将含错误信息的引用对象或者可能产生错误的完整对象传递过来，但这样做都不是很好，最好还是自定义异常类，其实这些基本上都是最原始的异常处理技巧. 所以在适当的时候使用自定义异常.

八、标准特性

▽总是优于非标准特性.

最大化可移植性和兼容性，尽量使用标准方法而不是用非标准方法，(比如，优先用 string.charAt(3) 而不用 string[3]，通过 DOM 原生函数访问元素，而不是使用应用封装好的快速接口.

九、封装基本类型

▽不要

没有任何理由去封装基本类型，另外还存在一些风险:

```
var x = new Boolean(false);

if (x) {

  alert('hi');  // Shows 'hi'.

}
```

除非明确用于类型转换，其他情况请千万不要这样做!

```
var x = Boolean(0);

if (x) {

  alert('hi');  // This will never be alerted.

}

typeof Boolean(0) == 'boolean';
```

```
typeof new Boolean(0) == 'object';
```

有时用作 number, string 或 boolean 时, 类型的转换会非常实用.

十、多级原型结构

▽不是首选

多级原型结构是指 JavaScript 中的继承关系. 当你自定义一个 D 类, 且把 B 类作为其原型, 那么这就获得了一个多级原型结构. 这些原型结构会变得越来越复杂!

使用 the Closure 库 中的 goog.inherits() 或其他类似的用于继承的函数, 会是更好的选择.

```
function D() {
  goog.base(this)
}
goog.inherits(D, B);


D.prototype.method = function() {
  ...
};
```

十一、方法定义

```
▽Foo.prototype.bar = function() { ... };
```

有很多方法可以给构造器添加方法或成员, 我们更倾向于使用如下的形式:

```
Foo.prototype.bar = function() {
  /* ... */
};
```

十二、闭包

▽可以，但小心使用.

闭包也许是 JS 中最有用的特性了．有一份比较好的介绍闭包原理的文档.

有一点需要牢记，闭包保留了一个指向它封闭作用域的指针，所以，在给 DOM 元素附加闭包时，很可能会产生循环引用，进一步导致内存泄漏．比如下面的代码:

```
function foo(element, a, b) {

  element.onclick = function() { /* uses a and b */ };

}
```

这里，即使没有使用 element，闭包也保留了 element，a 和 b 的引用，．由于 element 也保留了对闭包的引用，这就产生了循环引用，这就不能被 GC 回收．这种情况下，可将代码重构为:

```
function foo(element, a, b) {

  element.onclick = bar(a, b);

}

function bar(a, b) {

  return function() { /* uses a and b */ }

}
```

十三、eval()

▽ 只用于解析序列化串 (如: 解析 RPC 响应)

eval() 会让程序执行的比较混乱，当 eval() 里面包含用户输入的话就更加危险．可以用其他更佳的，更清晰，更安全的方式写你的代码，所以一般情况下请不要使用 eval()．当碰到一些需要解析序列化串的情况下(如，计算 RPC 响应)，使用 eval 很容易实现.

解析序列化串是指将字节流转换成内存中的数据结构．比如，你可能会将一个对象输出成文件形式:

```
users = [
  {
    name: 'Eric',
    id: 37824,
    email: 'jellyvore@myway.com'
  },
  {
    name: 'xtof',
    id: 31337,
    email: 'b4d455h4x0r@google.com'
  },
  ...
];
```

很简单地调用 eval 后，把表示成文件的数据读取回内存中.

类似的，eval() 对 RPC 响应值进行解码. 例如，你在使用 XMLHttpRequest 发出一个 RPC 请求后，通过 eval () 将服务端的响应文本转成 JavaScript 对象:

```
var userOnline = false;
var user = 'nusrat';
var xmlhttp = new XMLHttpRequest();
xmlhttp.open('GET', 'http://chat.google.com/isUserOnline?user=' + user, false);
xmlhttp.send('');
// Server returns:
// userOnline = true;
```

```
if (xmlhttp.status == 200) {
  eval(xmlhttp.responseText);
}
// userOnline is now true.
```

十四、with() {}

▽不要使用

使用 with 让你的代码在语义上变得不清晰. 因为 with 的对象, 可能会与局部变量产生冲突, 从而改变你程序原本的用义. 下面的代码是干嘛的?

```
with (foo) {
  var x = 3;
  return x;
}
```

答案: 任何事. 局部变量 x 可能被 foo 的属性覆盖, 当它定义一个 setter 时, 在赋值 3 后会执行很多其他代码. 所以不要使用 with 语句.

十五、this

▽ 仅在对象构造器, 方法, 闭包中使用.

this 的语义很特别. 有时它引用一个全局对象(大多数情况下), 调用者的作用域(使用 eval 时), DOM 树中的节点(添加事件处理函数时), 新创建的对象(使用一个构造器), 或者其他对象(如果函数被 call() 或 apply()).

使用时很容易出错, 所以只有在下面两个情况时才能使用:

在构造器中

对象的方法(包括创建的闭包)中

十六、for-in 循环

▽ 只用于 object/map/hash 的遍历

对 Array 用 for-in 循环有时会出错. 因为它并不是从 0 到 length - 1 进行遍历, 而是所有出现在对象及其原型链的键值. 下面就是一些失败的使用案例:

```
function printArray(arr) {
  for (var key in arr) {
    print(arr[key]);
  }
}
printArray([0,1,2,3]);  // This works.

var a = new Array(10);
printArray(a);  // This is wrong.

a = document.getElementsByTagName('*');
printArray(a);  // This is wrong.

a = [0,1,2,3];
a.buhu = 'wine';
printArray(a);  // This is wrong again.

a = new Array;
a[3] = 3;
printArray(a);  // This is wrong again.
```

而遍历数组通常用最普通的 for 循环.

```
function printArray(arr) {

  var l = arr.length;

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

    print(arr[i]);

  }

}
```

十七、关联数组

▽ 永远不要使用 Array 作为 map/hash/associative 数组.

数组中不允许使用非整型作为索引值, 所以也就不允许用关联数组. 而取代它使用 Object 来表示 map/hash 对象. Array 仅仅是扩展自 Object (类似于其他 JS 中的对象, 就像 Date, RegExp 和 String)一样来使用.

十八、多行字符串

▽不要使用

不要这样写长字符串:

```
var myString = 'A rather long string of English text, an error message
\

                actually that just keeps going and going -- an error \

                message to make the Energizer bunny blush (right through
\

                those Schwarzenegger shades)! Where was I? Oh yes, \

                you\'ve got an error and all the extraneous whitespace
is \

                just gravy.  Have a nice day.';
```

在编译时, 不能忽略行起始位置的空白字符; "\" 后的空白字符会产生奇怪的错误; 虽然大多数脚本引擎支持这种写法, 但它不是 ECMAScript 的标准规范.

十九、Array 和 Object 直接量

▽使用

使用 Array 和 Object 语法，而不使用 Array 和 Object 构造器.

使用 Array 构造器很容易因为传参不恰当导致错误.

```
// Length is 3.

var a1 = new Array(x1, x2, x3);

// Length is 2.

var a2 = new Array(x1, x2);

// If x1 is a number and it is a natural number the length will be x1.

// If x1 is a number but not a natural number this will throw an exception.

// Otherwise the array will have one element with x1 as its value.

var a3 = new Array(x1);

// Length is 0.

var a4 = new Array();
```

如果传入一个参数而不是 2 个参数，数组的长度很有可能就不是你期望的数值了.

为了避免这些歧义，我们应该使用更易读的直接量来声明.

```
var a = [x1, x2, x3];

var a2 = [x1, x2];

var a3 = [x1];

var a4 = [];
```

虽然 Object 构造器没有上述类似的问题，但鉴于可读性和一致性考虑，最好还是在字面上更清晰地指明.

```
var o = new Object();
```

```
var o2 = new Object();

o2.a = 0;

o2.b = 1;

o2.c = 2;

o2['strange key'] = 3;
```

应该写成:

```
var o = {};

var o2 = {

  a: 0,

  b: 1,

  c: 2,

  'strange key': 3

};
```

二十、修改内置对象的原型

▽不要

千万不要修改内置对象, 如 Object.prototype 和 Array.prototype 的原型. 而修改内置对象, 如 Function.prototype 的原型, 虽然少危险些, 但仍会导致调试时的诡异现象. 所以也要避免修改其原型.

二十一、IE 下的条件注释

▽不要使用

不要这样子写:

```
var f = function () {

    /*@cc_on if (@_jscript) { return 2* @*/  3; /*@ } @*/
```

```
};
```

条件注释妨碍自动化工具的执行，因为在运行时，它们会改变 JavaScript 语法树.

JAVASCRIPT 编码风格

一、命名

▽

通常，使用 functionNamesLikeThis，variableNamesLikeThis，ClassNamesLikeThis，EnumNamesLikeThis，methodNamesLikeThis，和 SYMBOLIC_CONSTANTS_LIKE_THIS.

属性和方法

文件或类中的 私有 属性，变量和方法名应该以下划线 "_" 开头.

保护 属性，变量和方法名不需要下划线开头，和公共变量名一样.

更多有关 私有 和 保护的信息见，visibility.

方法和函数参数

可选参数以 opt_ 开头.

函数的参数个数不固定时，应该添加最后一个参数 var_args 为参数的个数. 你也可以不设置 var_args 而取代使用 arguments.

可选和可变参数应该在 @param 标记中说明清楚. 虽然这两个规定对编译器没有任何影响，但还是请尽量遵守

Getters 和 Setters

Getters 和 setters 并不是必要的. 但只要使用它们了，就请将 getters 命名成 getFoo() 形式，将 setters 命名成 setFoo(value) 形式. (对于布尔类型的 getters，使用 isFoo() 也可.)

命名空间

JavaScript 不支持包和命名空间.

不容易发现和调试全局命名的冲突，多个系统集成时还可能因为命名冲突导致很严重的问题. 为了提高 JavaScript 代码复用率，我们遵循下面的约定以避免冲突.

为全局代码使用命名空间

在全局作用域上，使用一个唯一的，与工程/库相关的名字作为前缀标识. 比如，你的工程是 "Project Sloth"，那么命名空间前缀可取为 sloth.*.

```
var sloth = {};

sloth.sleep = function() {

  ...

};
```

许多 JavaScript 库，包括 the Closure Library and Dojo toolkit 为你提供了声明你自己的命名空间的函数. 比如:

```
goog.provide('sloth');

sloth.sleep = function() {

  ...

};
```

明确命名空间所有权

当选择了一个子命名空间，请确保父命名空间的负责人知道你在用哪个子命名空间，比如说，你为工程 'sloths' 创建一个 'hats' 子命名空间，那确保 Sloth 团队人员知道你在使用 sloth.hats.

外部代码和内部代码使用不同的命名空间

"外部代码" 是指来自于你代码体系的外部，可以独立编译. 内外部命名应该严格保持独立. 如果你使用了外部库，他的所有对象都在 foo.hats.* 下，那么你自己的代码不能在 foo.hats.*下命名，因为很有可能其他团队也在其中命名.

```
foo.require('foo.hats');
```

```
/**
 * WRONG -- Do NOT do this.
 * @constructor
 * @extend {foo.hats.RoundHat}
 */
foo.hats.BowlerHat = function() {
};
```

如果你需要在外部命名空间中定义新的 API，那么你应该直接导出一份外部库，然后在这份代码中修改. 在你的内部代码中，应该通过他们的内部名字来调用内部 API ，这样保持一致性可让编译器更好的优化你的代码.

```
foo.provide('googleyhats.BowlerHat');


foo.require('foo.hats');


/**
 * @constructor
 * @extend {foo.hats.RoundHat}
 */
googleyhats.BowlerHat = function() {
  ...
};


goog.exportSymbol('foo.hats.BowlerHat', googleyhats.BowlerHat);
```

重命名那些名字很长的变量，提高可读性

主要是为了提高可读性. 局部空间中的变量别名只需要取原名字的最后部分.

```
/**
 * @constructor
 */
some.long.namespace.MyClass = function() {
};


/**
 * @param {some.long.namespace.MyClass} a
 */
some.long.namespace.MyClass.staticHelper = function(a) {
  ...
};


myapp.main = function() {
  var MyClass = some.long.namespace.MyClass;
  var staticHelper = some.long.namespace.MyClass.staticHelper;
  staticHelper(new MyClass());
};
```

不要对命名空间创建别名.

```
myapp.main = function() {
  var namespace = some.long.namespace;
  namespace.MyClass.staticHelper(new namespace.MyClass());
};
```

除非是枚举类型，不然不要访问别名变量的属性.

```
/** @enum {string} */
some.long.namespace.Fruit = {
  APPLE: 'a',
  BANANA: 'b'
};


myapp.main = function() {
  var Fruit = some.long.namespace.Fruit;
  switch (fruit) {
    case Fruit.APPLE:
      ...
    case Fruit.BANANA:
      ...
  }
};
myapp.main = function() {
  var MyClass = some.long.namespace.MyClass;
  MyClass.staticHelper(null);
};
```

不要在全局范围内创建别名，而仅在函数块作用域中使用.

文件名

文件名应该使用小写字符，以避免在有些系统平台上不识别大小写的命名方式. 文件名以.js 结尾，不要包含除 - 和 _ 外的标点符号(使用 - 优于 _).

二、自定义 toString() 方法

▽ 应该总是成功调用且不要抛异常.

可自定义 toString() 方法，但确保你的实现方法满足：(1) 总是成功 (2) 没有其他负面影响. 如果不满足这两个条件，那么可能会导致严重的问题，比如，如果 toString() 调用了包含 assert 的函数，assert 输出导致失败的对象，这在 toString() 也会被调用.

三、延迟初始化

▽可以

没必要在每次声明变量时就将其初始化.

四、明确作用域

▽任何时候都需要

任何时候都要明确作用域 - 提高可移植性和清晰度. 例如，不要依赖于作用域链中的 window 对象. 可能在其他应用中，你函数中的 window 不是指之前的那个窗口对象.

五、代码格式化

▽主要依照 C++ 格式规范 (中文版)，针对 JavaScript，还有下面一些附加说明.

大括号

分号会被隐式插入到代码中，所以你务必在同一行上插入大括号. 例如:

```
if (something) {

  // ...
```

```
} else {
  // ...
}
```

数组和对象的初始化

如果初始值不是很长，就保持写在单行上:

```
var arr = [1, 2, 3];  // No space after [ or before ].
var obj = {a: 1, b: 2, c: 3};  // No space after { or before }.
```

初始值占用多行时，缩进 2 个空格.

```
// Object initializer.
var inset = {
  top: 10,
  right: 20,
  bottom: 15,
  left: 12
};

// Array initializer.
this.rows_ = [
  '"Slartibartfast" <fjordmaster@magrathea.com>',
  '"Zaphod Beeblebrox" <theprez@universe.gov>',
  '"Ford Prefect" <ford@theguide.com>',
  '"Arthur Dent" <has.no.tea@gmail.com>',
  '"Marvin the Paranoid Android" <marv@googlemail.com>',
  'the.mice@magrathea.com'
```

```
];


// Used in a method call.
goog.dom.createDom(goog.dom.TagName.DIV, {
  id: 'foo',
  className: 'some-css-class',
  style: 'display:none'
}, 'Hello, world!');
```

比较长的标识符或者数值，不要为了让代码好看些而手工对齐. 如:

```
CORRECT_Object.prototype = {
  a: 0,
  b: 1,
  lengthyName: 2
};
```

不要这样做:

```
WRONG_Object.prototype = {
  a          : 0,
  b          : 1,
  lengthyName: 2
};
```

函数参数

尽量让函数参数在同一行上. 如果一行超过 80 字符, 每个参数独占一行, 并以 4 个空格缩进, 或者与括号对齐, 以提高可读性. 尽可能不要让每行超过 80 个字符. 比如下面这样:

```
// Four-space, wrap at 80.  Works with very long function names, survives
// renaming without reindenting, low on space.
goog.foo.bar.doThingThatIsVeryDifficultToExplain = function(
    veryDescriptiveArgumentNumberOne, veryDescriptiveArgumentTwo,
    tableModelEventHandlerProxy, artichokeDescriptorAdapterIterator) {
  // ...
};

// Four-space, one argument per line.  Works with long function names,
// survives renaming, and emphasizes each argument.
goog.foo.bar.doThingThatIsVeryDifficultToExplain = function(
    veryDescriptiveArgumentNumberOne,
    veryDescriptiveArgumentTwo,
    tableModelEventHandlerProxy,
    artichokeDescriptorAdapterIterator) {
  // ...
};

// Parenthesis-aligned indentation, wrap at 80.  Visually groups arguments,
// low on space.
```

```
function foo(veryDescriptiveArgumentNumberOne,
veryDescriptiveArgumentTwo,

             tableModelEventHandlerProxy,
artichokeDescriptorAdapterIterator) {

  // ...

}


// Parenthesis-aligned, one argument per line.  Visually groups and

// emphasizes each individual argument.

function bar(veryDescriptiveArgumentNumberOne,

             veryDescriptiveArgumentTwo,

             tableModelEventHandlerProxy,

             artichokeDescriptorAdapterIterator) {

  // ...

}
```

传递匿名函数

如果参数中有匿名函数, 函数体从调用该函数的左边开始缩进 2 个空格, 而不是从 function 这个关键字开始. 这让匿名函数更加易读 (不要增加很多没必要的缩进让函数体显示在屏幕的右侧).

```
var names = items.map(function(item) {

                        return item.name;

                      });


prefix.something.reallyLongFunctionName('whatever', function(a1, a2) {

  if (a1.equals(a2)) {
```

```
    someOtherLongFunctionName(a1);
  } else {
    andNowForSomethingCompletelyDifferent(a2.parrot);
  }
});
```

更多的缩进

事实上，除了 初始化数组和对象 ，和传递匿名函数外，所有被拆开的多行文本要么选择与之前的表达式左对齐，要么以4个(而不是2个)空格作为一缩进层次.

```
someWonderfulHtml = '' +
                    getEvenMoreHtml(someReallyInterestingValues,
moreValues,
                                    evenMoreParams, 'a duck', true, 72,
                                    slightlyMoreMonkeys(0xfff)) +
                    '';


thisIsAVeryLongVariableName =
    hereIsAnEvenLongerOtherFunctionNameThatWillNotFitOnPrevLine();


thisIsAVeryLongVariableName = 'expressionPartOne' +
someMethodThatIsLong() +
    thisIsAnEvenLongerOtherFunctionNameThatCannotBeIndentedMore();


someValue = this.foo(
    shortArg,
```

```
    'Some really long string arg - this is a pretty common case,
actually.',
    shorty2,
    this.bar());

if (searchableCollection(allYourStuff).contains(theStuffYouWant) &&
    !ambientNotification.isActive() && (client.isAmbientSupported() ||
client.alwaysTryAmbientAnyways()) {
  ambientNotification.activate();
}
```

空行

使用空行来划分一组逻辑上相关联的代码片段.

```
doSomethingTo(x);
doSomethingElseTo(x);
andThen(x);

nowDoSomethingWith(y);

andNowWith(z);
```

二元和三元操作符

操作符始终跟随着前行, 这样就不用顾虑分号的隐式插入问题. 如果一行实在放不下, 还是按照上述的缩进风格来换行.

```
var x = a ? b : c;  // All on one line if it will fit.
```

```
// Indentation +4 is OK.
var y = a ?
    longButSimpleOperandB : longButSimpleOperandC;

// Indenting to the line position of the first operand is also OK.
var z = a ?
        moreComplicatedB :
        moreComplicatedC;
```

六、括号

▽只在需要的时候使用

不要滥用括号，只在必要的时候使用它.

对于一元操作符(如 delete, typeof 和 void)，或是在某些关键词(如 return, throw, case, new)之后，不要使用括号.

七、字符串

▽使用 ’ 优于 ″

单引号 (') 优于双引号 (").. 当你创建一个包含 HTML 代码的字符串时就知道它的好处了.

```
var msg = 'This is some HTML';
```

八、可见性 (私有域和保护域)

▽推荐使用 JSDoc 中的两个标记: @private 和 @protected

JSDoc 的两个标记 @private 和 @protected 用来指明类，函数，属性的可见性域.

标记为 @private 的全局变量和函数，表示它们只能在当前文件中访问.

标记为 @private 的构造器，表示该类只能在当前文件或是其静态/普通成员中实例化；私有构造器的公共静态属性在当前文件的任何地方都可访问，通过 instanceof 操作符也可.

永远不要为 全局变量，函数，构造器加 @protected 标记.

```
// File 1.

// AA_PrivateClass_ and AA_init_ are accessible because they are global

// and in the same file.


/**

 * @private

 * @constructor

 */

AA_PrivateClass_ = function() {

};


/** @private */

function AA_init_() {

  return new AA_PrivateClass_();

}


AA_init_();
```

标记为 @private 的属性，在当前文件中可访问它；如果是类属性私有，"拥有"该属性的类的所有静态/普通成员也可访问，但它们不能被不同文件中的子类访问或覆盖.

标记为 @protected 的属性，在当前文件中可访问它，如果是类属性保护，那么"拥有"该属性的类及其子类中的所有静态/普通成员也可访问.

注意：这与 C++，Java 中的私有和保护不同，它们是在当前文件中，检查是否具有访问私有/保护属性的权限，有权限即可访问，而不是只能在同一个类或类层次上. 而 C++ 中的私有属性不能被子类覆盖. (C++/Java 中的私有/保护是指作用域上的可访问性，在可访问性上的限制. JS 中是在限制在作用域上. PS: 可见性是与作用域对应)

```
// File 1.


/** @constructor */

  AA_PublicClass = function() {

};


/** @private */

AA_PublicClass.staticPrivateProp_ = 1;


/** @private */

AA_PublicClass.prototype.privateProp_ = 2;


/** @protected */

AA_PublicClass.staticProtectedProp = 31;


/** @protected */

AA_PublicClass.prototype.protectedProp = 4;
```

```
// File 2.


/**
 * @return {number} The number of ducks we've arranged in a row.
 */
AA_PublicClass.prototype.method = function() {
  // Legal accesses of these two properties.
  return this.privateProp_ + AA_PublicClass.staticPrivateProp_;
};


// File 3.


/**
 * @constructor
 * @extends {AA_PublicClass}
 */
AA_SubClass = function() {
  // Legal access of a protected static property.
  AA_PublicClass.staticProtectedProp = this.method();
};
goog.inherits(AA_SubClass, AA_PublicClass);


/**
 * @return {number} The number of ducks we've arranged in a row.
```

```
 */

AA_SubClass.prototype.method = function() {

  // Legal access of a protected instance property.

  return this.protectedProp;

};
```

九、JavaScript 类型

▽强烈建议你去使用编译器.

如果使用 JSDoc，那么尽量具体地，准确地根据它的规则来书写类型说明. 目前支持两种 JS2 和 JS1.x 类型规范.

JavaScript 类型语言

JS2 提议中包含了一种描述 JavaScript 类型的规范语法，这里我们在 JSDoc 中采用其来描述函数参数和返回值的类型.

JSDoc 的类型语言，按照 JS2 规范，也进行了适当改变，但编译器仍然支持旧语法.

名称	语法	描述	弃用语法
普通类型	{boolean}, {Window}, {goog.ui.Menu}	普通类型的描述方法.	
复杂类型	{Array.<string>} 字符串数组. {Object.<string, number>} 键为字符串，值为整数的对象类型.	参数化类型，即指定了该类型中包含的一系列"类型参数". 类似于 Java 中的泛型.	
联合类型	{(number|boolean)} 一个整数或者布尔值.	表示其值可能是 A 类型，也可能是 B 类型	{(number,boolean)}, {number|boolean}, {(number||boolean)}
记录类型	{{myNum: number, myObject}} 由现有类型组成的类型.	表示包含指定成员及类型的值. 这个例子中，myNum 为 number 类型，myObject 为任意类型. 注意大括号为类型语法的一部分. 比如，Array.<{length}>，表示一具有 length 属性的 Array 对象.	
可为空类型	{?number} 一个整型数或者为 NULL	表示一个值可能是 A 类型或者 null. 默认，每个对象都是可为空的. 注意：函数类型不可为空.	{number?}
非空类型	{!Object} 一个对象，但绝不会是 null 值.	说明一个值是类型 A 且肯定不是 null. 默认情况下，所有值类型 (boolean, number, string, 和 undefined) 不可为空.	{Object!}
函数类型	{function(string, boolean)} 具有两个参数 (string 和 boolean) 的函数类型，返回值未知.	说明一个函数.	
函数返回类型	{function(): number} 函数返回一个整数.	说明函数的返回类型.	
函数的 this 类型	{function(this:goog.ui.Menu, string)} 函数只带一个参数 (string)，并且在上下文 goog.ui.Menu 中执行.	说明函数类型的上下文类型.	
可变参数	{function(string, ...[number]): number} 带一个参数 (字符类型) 的函数类型，并且函数的参数个数可变，但参数类型必须为 number.	说明函数的可变长参数.	
可变长的参数 (使用 @param 标记)	@param {...number} var_args 函数参数个数可变.	使用标记，说明函数具有不定长参数.	
函数的 缺省参数	{function(?string=, number=)} 函数带一个可空且可选的字符串型参数，一个可选整型参数. = 语法只针对 function 类型有效.	说明函数的可选参数.	
函数 可选参数 (使用 @param 标记)	@param {number=} opt_argument number类型的可选参数.	使用标记，说明函数具有可选参数.	
所有类型	{*}	表示变量可以是任何类型.	

JavaScript 中的类型

类型示例	值示例	描述
number	1 1.0 -5 1e5 Math.PI	
Number	new Number(true)	Number 对象
string	'Hello' "World" String(42)	字符串值
String	new String('Hello') new String(42)	字符串对象
boolean	true false Boolean(0)	布尔值
Boolean	new Boolean(true)	布尔对象
RegExp	new RegExp('hello') /world/g	
Date	new Date new Date()	
null	null	
undefined	undefined	
void	function f() { return; }	没有返回值
Array	['foo', 0.3, null] []	类型不明确的数组
Array.<number>	[11, 22, 33]	整型数组
Array.<Array.<string>>	[['one', 'two', 'three'], ['foo', 'bar']]	字符串数组的数组
Object	{} {foo: 'abc', bar: 123, baz: null}	
Object.<string>	{'foo': 'bar'}	值为字符串的对象.
Object.<number, string>	var obj = {}; obj[1] = 'bar';	键为整数, 值为字符串的对象. 注意, JavaScript 中, 键总是被转换成字符串, 所以 obj['1'] == obj[1]. 也所以, 键在 for...in 循环中是字符串类型. 但在编译器中会明确根据键的类型来查找对象.
Function	function(x, y) { return x * y; }	函数对象
function(number, number): number	function(x, y) { return x * y; }	函数值
SomeClass	/** @constructor */ function SomeClass() {} new SomeClass();	
SomeInterface	/** @interface */ function SomeInterface() {} SomeInterface.prototype.draw = function() {};	
project.MyClass	/** @constructor */ project.MyClass = function () {} new project.MyClass()	
project.MyEnum	/** @enum {string} */ project.MyEnum = { BLUE: '#0000dd', RED: '#dd0000' };	枚举
Element	document.createElement('div')	DOM 中的元素
Node	document.body.firstChild	DOM 中的节点.
HTMLInputElement	htmlDocument.getElementsByTagName('input')[0]	DOM 中, 特定类型的元素.

可空 vs. 可选 参数和属性

JavaScript 是一种弱类型语言, 明白可选, 非空和未定义参数或属性之间的细微差别还是很重要的.

对象类型(引用类型)默认非空. 注意: 函数类型默认不能为空. 除了字符串, 整型, 布尔, undefined 和 null 外, 对象可以是任何类型.

```
/**
 * Some class, initialized with a value.
 * @param {Object} value Some value.
 * @constructor
 */
function MyClass(value) {
  /**
   * Some value.
   * @type {Object}
   * @private
   */
  this.myValue_ = value;
}
```

告诉编译器 myValue_ 属性为一对象或 null. 如果 myValue_ 永远都不会为 null, 就应该如下声明:

```
/**
 * Some class, initialized with a non-null value.
 * @param {!Object} value Some value.
 * @constructor
 */
function MyClass(value) {
  /**
   * Some value.
```

```
   * @type {!Object}
   * @private
   */
  this.myValue_ = value;
}
```

这样，当编译器在代码中碰到 MyClass 为 null 时，就会给出警告.

函数的可选参数可能在运行时没有定义，所以如果他们又被赋给类属性，需要声明成:

```
/**
 * Some class, initialized with an optional value.
 * @param {Object=} opt_value Some value (optional).
 * @constructor
 */
function MyClass(opt_value) {
  /**
   * Some value.
   * @type {Object|undefined}
   * @private
   */
  this.myValue_ = opt_value;
}
```

这告诉编译器 myValue_ 可能是一个对象，或 null，或 undefined.

注意: 可选参数 opt_value 被声明成 {Object=}, 而不是 {Object|undefined}. 这是因为可选参数可能是 undefined. 虽然直接写 undefined 也并无害处, 但鉴于可阅读性还是写成上述的样子.

最后, 属性的非空和可选并不矛盾, 属性既可是非空, 也可是可选的. 下面的四种声明各不相同:

```
/**
 * Takes four arguments, two of which are nullable, and two of which are
 * optional.
 * @param {!Object} nonNull Mandatory (must not be undefined), must not be null.
 * @param {Object} mayBeNull Mandatory (must not be undefined), may be null.
 * @param {!Object=} opt_nonNull Optional (may be undefined), but if present,
 *     must not be null!
 * @param {Object=} opt_mayBeNull Optional (may be undefined), may be null.
 */
function strangeButTrue(nonNull, mayBeNull, opt_nonNull, opt_mayBeNull) {
  // ...
};
```

十、注释

▽使用 JSDoc

我们使用 JSDoc 中的注释风格. 行内注释使用 // 变量 的形式. 另外, 我们也遵循 C++ 代码注释风格 . 这也就是说你需要:

版权和著作权的信息,

文件注释中应该写明该文件的基本信息(如, 这段代码的功能摘要, 如何使用, 与哪些东西相关), 来告诉那些不熟悉代码的读者.

类, 函数, 变量和必要的注释,

期望在哪些浏览器中执行,

正确的大小写, 标点和拼写.

为了避免出现句子片段, 请以合适的大/小写单词开头, 并以合适的标点符号结束这个句子.

现在假设维护这段代码的是一位初学者. 这可能正好是这样的!

目前很多编译器可从 JSDoc 中提取类型信息, 来对代码进行验证, 删除和压缩. 因此, 你很有必要去熟悉正确完整的 JSDoc .

顶层/文件注释

顶层注释用于告诉不熟悉这段代码的读者这个文件中包含哪些东西. 应该提供文件的大体内容, 它的作者, 依赖关系和兼容性信息. 如下:

```
// Copyright 2009 Google Inc. All Rights Reserved.


/**
 * @fileoverview Description of file, its uses and information
 * about its dependencies.
 * @author user@google.com (Firstname Lastname)
 */
```

类注释

每个类的定义都要附带一份注释，描述类的功能和用法. 也需要说明构造器参数. 如果该类继承自其它类，应该使用 @extends 标记. 如果该类是对接口的实现，应该使用 @implements 标记.

```
/**
 * Class making something fun and easy.
 * @param {string} arg1 An argument that makes this more interesting.
 * @param {Array.<number>} arg2 List of numbers to be processed.
 * @constructor
 * @extends {goog.Disposable}
 */
project.MyClass = function(arg1, arg2) {
  // ...
};
goog.inherits(project.MyClass, goog.Disposable);
```

方法与函数的注释

提供参数的说明. 使用完整的句子，并用第三人称来书写方法说明.

```
/**
 * Converts text to some completely different text.
 * @param {string} arg1 An argument that makes this more interesting.
 * @return {string} Some return value.
 */
```

```
project.MyClass.prototype.someMethod = function(arg1) {
  // ...
};


/**
 * Operates on an instance of MyClass and returns something.
 * @param {project.MyClass} obj Instance of MyClass which leads to a long
 *     comment that needs to be wrapped to two lines.
 * @return {boolean} Whether something occured.
 */
function PR_someMethod(obj) {
  // ...
}
```

对于一些简单的, 不带参数的 getters, 说明可以忽略.

```
/**
 * @return {Element} The element for the component.
 */
goog.ui.Component.prototype.getElement = function() {
  return this.element_;
};
```

属性注释

也需要对属性进行注释.

```
/**
```

```
 * Maximum number of things per pane.

 * @type {number}

 */

project.MyClass.prototype.someProperty = 4;
```

类型转换的注释

有时，类型检查不能很准确地推断出表达式的类型，所以应该给它添加类型标记注释来明确之，并且必须在表达式和类型标签外面包裹括号.

```
/** @type {number} */ (x)

(/** @type {number} */ x)
```

JSDoc 缩进

如果你在 @param, @return, @supported, @this 或 @deprecated 中断行，需要像在代码中一样，使用 4 个空格作为一个缩进层次.

```
/**

 * Illustrates line wrapping for long param/return descriptions.

 * @param {string} foo This is a param with a description too long to fit
in

 *     one line.

 * @return {number} This returns something that has a description too long
to

 *     fit in one line.

 */

project.MyClass.prototype.method = function(foo) {

  return 5;

};
```

不要在 @fileoverview 标记中进行缩进.

虽然不建议，但也可对说明文字进行适当的排版对齐. 不过，这样带来一些负面影响，就是当你每次修改变量名时，都得重新排版说明文字以保持和变量名对齐.

```
/**
 * This is NOT the preferred indentation method.
 * @param {string} foo This is a param with a description too long to fit in
 *                     one line.
 * @return {number} This returns something that has a description too long to
 *                  fit in one line.
 */
project.MyClass.prototype.method = function(foo) {
  return 5;
};
```

枚举

```
/**
 * Enum for tri-state values.
 * @enum {number}
 */
project.TriState = {
  TRUE: 1,
```

```
  FALSE: -1,
  MAYBE: 0
};
```

注意一下，枚举也具有有效类型，所以可以当成参数类型来用.

```
/**
 * Sets project state.
 * @param {project.TriState} state New project state.
 */
project.setState = function(state) {
  // ...
};
```

Typedefs

有时类型会很复杂. 比如下面的函数，接收 Element 参数:

```
/**
 * @param {string} tagName
 * @param {(string|Element|Text|Array.<Element>|Array.<Text>)}
contents
 * @return {Element}
 */
goog.createElement = function(tagName, contents) {
  ...
};
```

你可以使用 @typedef 标记来定义个常用的类型表达式.

```
/** @typedef {(string|Element|Text|Array.<Element>|Array.<Text>)} */
goog.ElementContent;


/**
* @param {string} tagName
* @param {goog.ElementContent} contents
* @return {Element}
*/
goog.createElement = function(tagName, contents) {
...
};
```

JSDoc 标记表

标记	模板 & 例子	描述	类型检测支持
@param	@param {Type} 变量名 描述 *如:* `/**` ` * Queries a Baz for items.` ` * @param {number} groupNum Subgroup id to query.` ` * @param {string|number|null} term An itemName,` ` *     or itemId, or null to search everything.` ` */` `goog.Baz.prototype.query = function(groupNum, term) {` `  // ...` `};`	给方法，函数，构造器中的参数添加说明.	完全支持.
@return	@return {Type} 描述 *如:* `/**` ` * @return {string} The hex ID of the last item.` ` */` `goog.Baz.prototype.getLastId = function() {` `  // ...` `  return id;` `};`	给方法，函数的返回值添加说明. 在描述布尔型参数时，用 "Whether the component is visible" 这种描述优于 "True if the component is visible, false otherwise". 如果函数没有返回值，就不需要添加 @return 标记.	完全支持.
@author	@author username@google.com (first last) *如:* `/**` ` * @fileoverview Utilities for handling textareas.` ` * @author kuth@google.com (Uthur Pendragon)` ` */`	表明文件的作者，通常仅会在 @fileoverview 注释中使用到它.	不需要.
@see	@see Link *如:* `/**` ` * Adds a single item, recklessly.` ` * @see #addSafely` ` * @see goog.Collect` ` * @see goog.RecklessAdder#add` ` ...`	给出引用链接，用于进一步查看函数/方法的相关细节.	不需要.
@fileoverview	@fileoverview 描述 *如:* `/**` ` * @fileoverview Utilities for doing things that require this very long` ` * but not indented comment.` ` * @author kuth@google.com (Uthur Pendragon)` ` */`	文件通览.	不需要.
@constructor	@constructor *如:* `/**` ` * A rectangle.` ` * @constructor` ` */` `function GM_Rect() {` `  ...` `}`	指明类中的构造器.	会检查. 如果省略了, 编译器将禁止实例化.
@interface	@interface *如:* `/**` ` * A shape.` ` * @interface` ` */` `function Shape() {};` `Shape.prototype.draw = function() {};` `/**` ` * A polygon.` ` * @interface` ` * @extends {Shape}` ` */` `function Polygon() {};` `Polygon.prototype.getSides = function() {};`	指明这个函数是一个接口.	会检查. 如果实例化一个接口, 编译器会警告.
@type	@type Type @type {Type} *如:* `/**` ` * The message hex ID.` ` * @type {string}` ` */` `var hexId = hexId;`	标识变量，属性或表达式的类型. 大多数类型是不需要加大括号的，但为了保持一致，建议统一加大括号.	会检查
@extends	@extends Type @extends {Type} *如:* `/**` ` * Immutable empty node list.` ` * @constructor` ` * @extends goog.ds.BasicNodeList` ` */` `goog.ds.EmptyNodeList = function() {` `  ...` `};`	与 @constructor 一起使用，用来表明该类是扩展自其它类的. 类型外的大括号可写可不写.	会检查

@implements	@implements Type @implements {Type} *如:* ``` /** * A shape. * @interface */ function Shape() {}; Shape.prototype.draw = function() {}; /** * @constructor * @implements {Shape} */ function Square() {}; Square.prototype.draw = function() { ... }; ```	与 @constructor 一起使用，用来表明该类实现自一个接口．类型外的大括号可写可不写.	会检查.如果接口不完整,编译器会警告.
@lends	@lends objectName @lends {objectName} *如:* ``` goog.object.extend(Button.prototype, /** @lends {Button.prototype} */ { isButton: function() { return true; } }); ```	表示把对象的键看成是其他对象的属性．该标记只能出现在对象语法中. 注意，括号中的名称和其他标记中的类型名称不一样，它是一个对象名，以"借过来"的属性名命名．如，@type {Foo} 表示 "Foo 的一个实例"，but @lends {Foo} 表示 "Foo 构造器". 更多有关此标记的内容见 JSDoc Toolkit docs.	会检查

@private	@private *如:* ``` /** * Handlers that are listening to this logger. * @type Array.<Function> * @private */ this.handlers_ = []; ```	指明那些以下划线结尾的方法和属性是 私有的．不推荐使用后缀下划线，而应改用 @private.	需要指定标志来开启.
@protected	@protected *如:* ``` /** * Sets the component's root element to the given element. Considered * protected and final. * @param {Element} element Root element for the component. * @protected */ goog.ui.Component.prototype.setElementInternal = function(element) { // ... }; ```	指明接下来的方法和属性是 被保护的．被保护的方法和属性的命名不需要以下划线结尾，和普通变量名没区别.	需要指定标志来开启.
@this	@this Type @this {Type} *如:* ``` pinto.chat.RosterWidget.extern('getRosterElement', /** * Returns the roster widget element. * @this pinto.chat.RosterWidget * @return {Element} */ function() { return this.getWrappedComponent_().getElement(); }); ```	指明调用这个方法时，需要在哪个上下文中．当 this 指向的不是原型方法的函数时必须使用这个标记.	会检查

@supported	@supported 描述 *如:* ``` /** * @fileoverview Event Manager * Provides an abstracted interface to the * browsers' event systems. * @supported So far tested in IE6 and FF1.5 */ ```	在文件概述中用到，表明支持哪些浏览器.	不需要.
@enum	@enum {Type} *如:* ``` /** * Enum for tri-state values. * @enum {number} */ project.TriState = { TRUE: 1, FALSE: -1, MAYBE: 0 }; ```	用于枚举类型.	完全支持.如果省略,会认为是整型.
@deprecated	@deprecated 描述 *如:* ``` /** * Determines whether a node is a field. * @return {boolean} True if the contents of * the element are editable, but the element * itself is not. * @deprecated Use isField(). */ BN_EditUtil.isTopEditableField = function(node) { // ... }; ```	告诉其他开发人员，此方法，函数已经过时，不要再使用．同时也会给出替代方法或函数.	不需要

@override	@override *如:* ``` /** * @return {string} Human-readable representation of project.SubClass. * @override */ project.SubClass.prototype.toString() { // ... }; ```	指明子类的方法和属性是故意隐藏了父类的方法和属性. 如果子类的方法和属性没有自己的文档, 就会继承父类的.	会检查
@inheritDoc	@inheritDoc *如:* ``` /** @inheritDoc */ project.SubClass.prototype.toString() { // ... }; ```	指明子类的方法和属性是故意隐藏了父类的方法和属性, 它们具有相同的文档. 注意: 使用 @inheritDoc 意味着也同时使用了 @override.	会检查
@code	{@code ...} *如:* ``` /** * Moves to the next position in the selection. * Throws {@code goog.iter.StopIteration} when it * passes the end of the range. * @return {Node} The node at the next position. */ goog.dom.RangeIterator.prototype.next = function() { // ... }; ```	说明这是一段代码, 让它能在生成的文档中正确的格式化.	不适用.
@license or @preserve	@license 描述 *如:* ``` /** * @preserve Copyright 2009 SomeThirdParty. * Here is the full license text and copyright * notice for this file. Note that the notice can span several * lines and is only terminated by the closing star and slash: */ ```	所有被标记为 @license 或 @preserve 的, 会被编译器保留不做任何修改而直接输出到最终文档中. 这个标记让一些重要的信息(如法律许可或版权信息)原样保留, 同样, 文本中的换行也会被保留.	不需要.
@noalias	@noalias *如:* ``` /** @noalias */ function Range() {} ```	在外部文件中使用, 告诉编译器不要为这个变量或函数重命名.	不需要.
@define	@define {Type} 描述 *如:* ``` /** @define {boolean} */ var TR_FLAGS_ENABLE_DEBUG = true; /** @define {boolean} */ goog.userAgent.ASSUME_IE = false; ```	表示该变量可在编译时被编译器重新赋值. 在上面例子中, BUILD 文件中指定了 --define='goog.userAgent.ASSUME_IE=true' 这个编译之后, 常量 goog.userAgent.ASSUME_IE 将被全部直接替换为 true.	不需要.
@export	@export *如:* ``` /** @export */ foo.MyPublicClass.prototype.myPublicMethod = function() { // ... }; ```	上面的例子代码, 当编译器运行时指定 --generate_exports 标志, 会生成下面的代码: ``` goog.exportSymbol('foo.MyPublicClass.prototype.myPublicMethod', foo.MyPublicClass.prototype.myPublicMethod); ``` 编译后, 将源代码中的名字原样导出. 使用 @export 标记时, 应该 1. 包含 //javascript/closure/base.js, 或者 2. 在代码库中自定义 goog.exportSymbol 和 goog.exportProperty 两个方法, 并保证有相同的调用方式.	不需要.

@const	@const *如:* `/** @const */ var MY_BEER = 'stout';` `/** * My namespace's favorite kind of beer. * @const * @type {string} */ mynamespace.MY_BEER = 'stout';` `/** @const */ MyClass.MY_BEER = 'stout';`	声明变量为只读，直接写在一行上. 如果其他代码中重写该变量值，编译器会警告. 常量应全部用大写字符，不过使用这个标记，可以帮你消除命名上依赖. 虽然 jsdoc.org 上列出的 @final 标记作用等价于 @const ，但不建议使用. @const 与 JS1.5 中的 const 关键字一致. 注意，编译器不禁止修改常量对象的属性(这与 C++ 中的常量定义不一样). 如果可以准确推测出常量类型的话，那么类型申明可以忽略. 如果指定了类型，应该也写在同一行上. 变量的额外注释可写可不写.	支持.
@nosideeffects	@nosideeffects *如:* `/** @nosideeffects */ function noSideEffectsFn1() { // ... };` `/** @nosideeffects */ var noSideEffectsFn2 = function() { // ... };` `/** @nosideeffects */ a.prototype.noSideEffectsFn3 = function() { // ... };`	用于对函数或构造器声明，说明调用此函数不会有副作用. 编译器遇到此标记时，如果调用函数的返回值没有其他地方使用到，则会将这个函数整个删除.	不需要检查.
@typedef	@typedef *如:* `/** @typedef {(string|number)} */ goog.NumberLike;` `/** @param {goog.NumberLike} x A number or a string. */ goog.readNumber = function(x) { ... }`	这个标记用于给一个复杂的类型取一个别名.	会检查
@externs	@externs *如:* `/** * @fileoverview This is an externs file. * @externs */ var document;`	指明一个外部文件.	不会检查

在第三方代码中，你还会见到其他一些 JSDoc 标记. 这些标记在 JSDoc Toolkit Tag Reference 都有介绍到，但在 Google 的代码中，目前不推荐使用. 你可以认为这些是将来会用到的“保留”名. 它们包含:

- @augments
- @argument
- @borrows
- @class
- @constant
- @constructs
- @default
- @event
- @example
- @field
- @function
- @ignore
- @inner
- @link
- @memberOf
- @name
- @namespace
- @property
- @public
- @requires
- @returns
- @since
- @static
- @version

JSDoc 中的 HTML

类似于 JavaDoc，JSDoc 支持许多 HTML 标签，如 <code>，<pre>，<tt>，，<ul>，<ol>，<li>，<a>，等等.

这就是说 JSDoc 不会完全依照纯文本中书写的格式. 所以，不要在 JSDoc 中，使用空白字符来做格式化:

```
/**
 * Computes weight based on three factors:
```

```
*   items sent

*   items received

*   last timestamp

*/
```

上面的注释，出来的结果是：

```
Computes weight based on three factors: items sent items received items received
```

应该这样写：

```
/**

 * Computes weight based on three factors:

 * <ul>

 * <li>items sent

 * <li>items received

 * <li>last timestamp

 * </ul>

 */
```

另外，也不要包含任何 HTML 或类 HTML 标签，除非你就想让它们解析成 HTML 标签.

```
/**

 * Changes <b> tags to <span> tags.

 */
```

出来的结果是:

```
Changes tags to tags.
```

另外，也应该在源代码文件中让其他人更可读，所以不要过于使用 HTML 标签:

```
/**
 * Changes &lt;b&gt; tags to &lt;span&gt; tags.
 */
```

上面的代码中，其他人就很难知道你想干嘛，直接改成下面的样子就清楚多了:

```
/**
* Changes 'b' tags to 'span' tags.
*/
```

十一、编译

▽推荐使用

建议您去使用 JS 编译器，如 Closure Compiler.

十二、Tips and Tricks

▽JavaScript 小技巧

True 和 False 布尔表达式

下面的布尔表达式都返回 false:

null

undefined

'' 空字符串

0 数字 0

但小心下面的, 可都返回 true:

'0' 字符串 0

[] 空数组

{} 空对象

下面段比较糟糕的代码:

```
while (x != null) {
```

你可以直接写成下面的形式(只要你希望 x 不是 0 和空字符串, 和 false):

```
while (x) {
```

如果你想检查字符串是否为 null 或空:

```
if (y != null && y != '') {
```

但这样会更好:

```
if (y) {
```

注意: 还有很多需要注意的地方, 如:

```
Boolean('0') == true
```

```
'0' != true
```

```
0 != null
```

```
0 == []
```

```
0 == false
```

```
Boolean(null) == false

null != true

null != false

Boolean(undefined) == false

undefined != true

undefined != false

Boolean([]) == true

[] != true

[] == false

Boolean({}) == true

{} != true

{} != false
```

条件(三元)操作符 (?:)

三元操作符用于替代下面的代码:

```
if (val != 0) {
  return foo();
} else {
  return bar();
}
```

你可以写成:

```
return val ? foo() : bar();
```

在生成 HTML 代码时也是很有用的:

```
var html = '<input type="checkbox"' +
```

```
    (isChecked ? ' checked' : '') +
    (isEnabled ? '' : ' disabled') +
    ' name="foo">';
```

&& 和 ||

二元布尔操作符是可短路的，只有在必要时才会计算到最后一项.

"||" 被称作为 'default' 操作符，因为可以这样:

```
/** @param {*=} opt_win */
function foo(opt_win) {
  var win;
  if (opt_win) {
    win = opt_win;
  } else {
    win = window;
  }
  // ...
}
```

你可以使用它来简化上面的代码:

```
/** @param {*=} opt_win */
function foo(opt_win) {
  var win = opt_win || window;
  // ...
}
```

"&&" 也可简短代码.比如:

```
if (node) {
  if (node.kids) {
    if (node.kids[index]) {
      foo(node.kids[index]);
    }
  }
}
```

你可以像这样来使用:

```
if (node && node.kids && node.kids[index]) {
  foo(node.kids[index]);
}
```

或者:

```
var kid = node && node.kids && node.kids[index];
if (kid) {
  foo(kid);
}
```

不过这样就有点儿过头了:

```
node && node.kids && node.kids[index] && foo(node.kids[index]);
```

使用 join() 来创建字符串

通常是这样使用的:

```
function listHtml(items) {
  var html = '<div class="foo">';
  for (var i = 0; i < items.length; ++i) {
    if (i > 0) {
      html += ', ';
    }
    html += itemHtml(items[i]);
  }
  html += '</div>';
  return html;
}
```

但这样在 IE 下非常慢, 可以用下面的方式:

```
function listHtml(items) {
  var html = [];
  for (var i = 0; i < items.length; ++i) {
    html[i] = itemHtml(items[i]);
  }
  return '<div class="foo">' + html.join(', ') + '</div>';
}
```

你也可以是用数组作为字符串构造器, 然后通过 myArray.join('') 转换成字符串. 不过由于赋值操作快于数组的 push(), 所以尽量使用赋值操作.

遍历 Node List

Node lists 是通过给节点迭代器加一个过滤器来实现的. 这表示获取他的属性, 如 length 的时间复杂度为 O(n), 通过 length 来遍历整个列表需要 O(n^2).

```
var paragraphs = document.getElementsByTagName('p');

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

  doSomething(paragraphs[i]);

}
```

这样做会更好:

```
var paragraphs = document.getElementsByTagName('p');

for (var i = 0, paragraph; paragraph = paragraphs[i]; i++) {

  doSomething(paragraph);

}
```

这种方法对所有的 collections 和数组(只要数组不包含 falsy 值) 都适用.

在上面的例子中, 也可以通过 firstChild 和 nextSibling 来遍历孩子节点.

```
var parentNode = document.getElementById('foo');

for (var child = parentNode.firstChild; child; child = child.nextSibling)
{

  doSomething(child);

}
```

十三、Parting Words

保持一致性.

当你在编辑代码之前，先花一些时间查看一下现有代码的风格．如果他们给算术运算符添加了空格，你也应该添加．如果他们的注释使用一个个星号盒子，那么也请你使用这种方式.

代码风格中一个关键点是整理一份常用词汇表，开发者认同它并且遵循，这样在代码中就能统一表述．我们在这提出了一些全局上的风格规则，但也要考虑自身情况形成自己的代码风格．但如果你添加的代码和现有的代码有很大的区别，这就让阅读者感到很不和谐．所以，避免这种情况的发生.

新门户项目命名

以下命名全部采用首字母小写的驼峰方式，一切命名都采用英语单词拼写方式，禁止采用汉语拼音拼写方式。

一、部件命名

（1）命名要求能反映部件的语意。

（2）假如该部件只能用于某个子门户，是非通用部件，则首单词必须是子门户名。

（3） 假如不确定具体用于哪个子门户，是通用部件，首单词无需指定子门户名。

比如 IT 服务子门户有个部件叫“最热知识”，这个部件只用于 IT 服务子门户，是非通用部件，则我们命名为“itHotKnowledge”。

又如，我的工作台和 HR 服务两个子门户都有“快捷方式”这个部件，是通用部件，我们命名为“shortcut”。

二、数据类型（Format）命名

通用部件：midea.format.general.部件名.数据类型名。

非通用部件：midea.format.子门户名.部件名.数据类型名。

规定省略数据类型名为 default。

假如有多个数据类型名，则必须指定一个数据类型名为 default，其他数据类型名再另外命名。

比如： midea.format.general.shortcut.default、
midea.format.general.shortcut.popup。

又如：midea.format.it.itHotKnowledge.default、
midea.format.it.itHotKnowledge.popup。

三、数据呈现（Render）命名

通用部件：midea.render.general.部件名.数据呈现名。

非通用部件：midea.render.子门户名.部件名.数据呈现名。

规定省略数据呈现名为 default。

假如有多个数据呈现名，则必须指定一个数据呈现名为 default，其他数据呈现名再另外命名。

比如： midea.render.general.shortcut.default、
midea.render.general.shortcut.popup。

又如：midea.render.it.itHotKnowledge.default、
midea.render.it.itHotKnowledge.popup。

四、布局（layout）命名

通用部件：midea.layout.general.部件名.布局名。

非通用部件：midea.layout.子门户名.部件名.布局名。

规定省略布局名为 default。

假如有多个布局名，则必须指定一个布局名为 default，其他布局名再另外命名。

比如： midea.layout.general.shortcut.default、
midea.layout.general.shortcut.popup。

又如：midea.layout.it.itHotKnowledge.default、
midea.layout.it.itHotKnowledge.popup。

五、数据抓取（Source）

通用部件：midea.source.general.部件名.数据抓取名。

非通用部件：midea.source.子门户名.部件名.数据抓取名。

规定省略数据抓取名为 default。

假如有多个数据抓取名，则必须指定一个数据抓取名为 default，其他数据抓取名再另外命名。

比如： midea.source.general.shortcut.default、
midea.source.general.shortcut.popup。

又如：midea.source.it.itHotKnowledge.default、
midea.source.it.itHotKnowledge.popup。

六、Portlet 命名

通用部件：midea.portlet.general.部件名.portlet 名。

非通用部件：midea.portlet.子门户名.部件名.portlet 名。

规定省略 portlet 名为 default。

假如有多个 portlet 名，则必须指定一个 portlet 名为 default，其他 portlet 名再另外命名。

比如： midea.portlet.general.shortcut.default、
midea.portlet.general.shortcut.popup。

又如：midea.portlet.it.itHotKnowledge.default、
midea.portlet.it.itHotKnowledge.popup。

七、目录名命名

命名要求能反映目录的语意。

八、JavaScript、CSS、HTML 文件命名

命名要求能反映文件的语意。

九、部件的目录及文件结构

开发一个部件通常用到以下目录及文件：

部件定义 XML：/ui-ext/midea2/design-xml/ （必需）

/ui-ext/midea2/design-xml/部件名.xml （必需）

部件主体：/ui-ext/midea2/resource/ （必需）

/ui-ext/midea2/resource/css （必需）

/ui-ext/midea2/resource/js （必需）

/ui-ext/midea2/resource/images （必需）

/ui-ext/midea2/resource/json （可选）

/ui-ext/midea2/resource/部件名/ （必需）

/ui-ext/midea2/resource/部件名/json/ （可选）

/ui-ext/midea2/resource/部件名/json/部件名.json （可选）

/ui-ext/midea2/resource/部件名/render/ （必需）

/ui-ext/midea2/resource/部件名/render/help/ （可选）

/ui-ext/midea2/resource/部件名/render/help/部件名.jpg （可选）

/ui-ext/midea2/resource/部件名/render/部件名.js （必需）

/ui-ext/midea2/resource/部件名/render/部件名.呈现数据名.js（可选）

/ui-ext/midea2/resource/部件名/render/部件名.tmpl（可选）

/ui-ext/midea2/resource/部件名/render/部件名.呈现数据名.tmpl（可选）

部件引用的 JSP：/WebContent/sys/midea2/ （必需）