CSS 的一些技巧

@(FontEnd)[CSS]

好的CSS代码应该是怎样的？

- DRY (Don't Repeat Yourself)
- 灵活,可维护
- 兼容,最好不要使用未标准化的属性
- 少依赖冗余的dom节点

一些小技巧

1.适当使用相对单位

- em，rem，%
- vw，vh ：相对视口宽高的百分比
- vmin，vmax ：相对于视口的宽度或高度中较小（大）的一个。将较 小（大）的那个均分为100单位的vmax

```
/*bad(复制蓝湖就会出现这样的代码)*/
font-size: 20px;
line-height: 30px;

/*good*/
font-size: 20px;
line-height: 1.5;
```

2.适当的简写

- 16进制色值，如果每两位相同可缩写一半： #112233 = #123

```
/*bad*/
margin: 10px 10px 10px 0;

/*good*/
margin: 10px;
margin-left: 0px;
```

```
/* bad */
margin: 10px 20px 0 20px;
/* or */
margin: 10px 20px;
margin-bottom: 0px;

/* good */
margin: 10px 20px 0;

/*bad 设置背景颜色 假如需求改变，这里的背景色要换为背景图片时，如果不注意忘记删除原来背景色的代码，
background-color与background-image会同时生效，这并不是我们所期望的（背景透明的图片会有底色），
同样如果要将原来设置的背景图片改为纯色，background-color并不会覆盖background-image*/
background-color: red;
/*good 不论是图片背景还是纯色背景都使用background简写就可以避免以上问题*/
background: red;

/* 不会将 background 的 color 值设置为 red，而是 background-color 的默认值 transparent，因为第二条规
则优先 */
background-color: red;
background: url(images/bg.gif) no-repeat top right;
```

3.使用inherit

```
/*让链接的颜色与父元素一致*/
/*good*/
a{
  color: inherit;
}
/*bad*/
a{
  color: #000;
}
/*伪元素与父元素的颜色保持一致*/
.select{
  background-color: gray;
}
.select::after{
  /*bad*/
  background-color: gray;
  /*good*/
  background-color: indert;
}
```

4.利用一些新的CSS属性实现一些常见的图形，而不是添加大量的伪元素

- 图片边框
- 短线效果

- 菱形边框
- 切角效果
- 梯形标签

5.使用预处理器(Less)提供的函数减少代码重复（DRY）

```
.clipcircle(@radius:15px,@bgcolor:red){
  background: radial-gradient(circle at top left,transparent @radius, @bgcolor 0) top left,
  radial-gradient(circle at top right,transparent @radius, @bgcolor 0) top right,
  radial-gradient(circle at bottom right, transparent @radius, @bgcolor 0) bottom right,
  radial-gradient(circle at bottom left, transparent @radius, @bgcolor 0) bottom left;
  background-size: 51% 51%;
  background-repeat: no-repeat;
}
.clip-circle{
  .clipcircle
  /* .clipcircle(20px,gray) */
}
```

6.居中的小技巧

- 垂直居中
- 基于视口的居中
- 100%宽度背景，固定宽度内容居中

7.将css属性按一定的规律顺序排列（提高代码可维护性）

参考：

1. 影响文档流的属性（比如：display, position, float, clear, visibility, table-layout等）
2. 自身盒模型的属性（比如：width, height, margin, padding, border等）
3. 排版相关属性（比如：font, line-height, text-align, text-indent, vertical-align等等）
4. 装饰性属性（比如：color, background, opacity, cursor等）
5. 生成内容的属性（比如：content, list-style等）

- 在使用中需要灵活使用，例如在不能将border属性简写的情况下，width与color，前者属于盒模型，后者属于装饰性的属性，我们最好还是将他们写在一起

```
/* good * /
.selector{
    display: inline-block;
    position: absolute;
    top:-20px;
    right: 0;
    width: 100px;
```

```
    height: 20px;
    padding: 5px 10px;

    font-size: 12px;
    line-height: 1.5;
    background-color: aquamarine;
    color: white;
    font-size: 0;
    font-weight: bold;
}
/* bad */
.selector{
    display: inline-block;
    top:-20px;
    right: 0;
    padding: 5px 10px;
    font-size: 12px;
    width: 100px;
    height: 20px;
    background-color: aquamarine;
    line-height: 1.5;
    color: white;
    font-weight: bold;
    position: absolute;
}
```

8. clip-path的一些用法

```
inset(<shape-arg>{1,4} [round <border-radius>]?)
```

前四个参数分别代表了插进的长方形与相关盒模型的上，右，下与左边界和顶点的偏移量 可选参数用于定义插进长方形顶点的圆弧角度

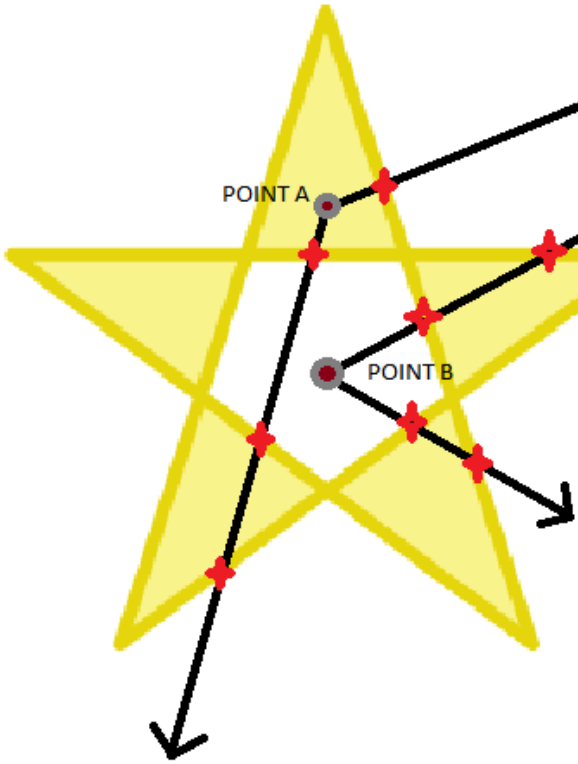
```
 circle( [<shape-radius>]? [at <position>]? )
```

参数代表了 r， 即圆形的半径， 不接受负数作为该参数的值，一个以百分比表示的值将以公式sqrt(width^2+height^2)/sqrt(2)计算 参数定义了圆心的位置。省缺值为盒模型的中心

```
polygon( [<fill-rule>,]? [<shape-arg> <shape-arg>]# )
```

代表了填充规则（ filling rule ），即，如何填充该多边形。 可选值为 nonzero 和 evenodd。 该参数的省缺值为nonzero。

- 要判断一个点是否在图形内，需要从该点作任意方向的一条射线，然后检测射线与图形路径的交点值，假设为x，x初始值为0。
- 对于evenodd，每相交一次，x+1，最后的结果如果x为奇数则该点在图形内，如果为偶数则在图形外
- 对于nonzero, (路径与射线需要考虑方向)任意假设一个方向为正,计算x的方法是，正相交+1，负相交-1，看最后x的值，如果为0，则该点在图形外，如果不为0则在图形内



Trial Ray 1 Crosses 1 path -> "ODD NUMBER" -> Point A is "inside" -> fill

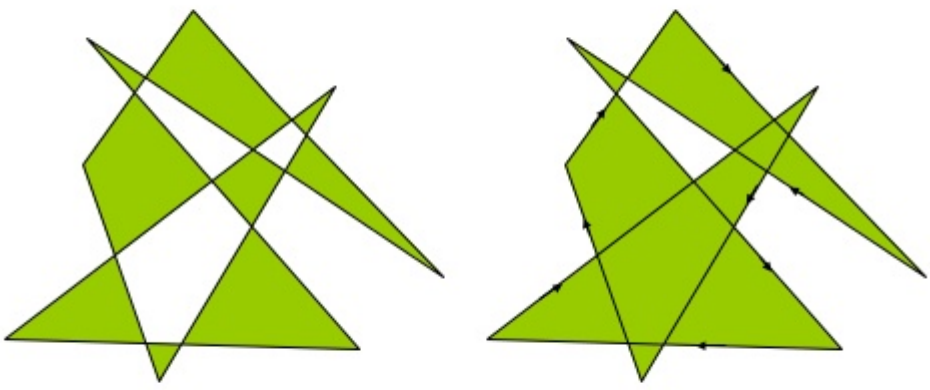
Trial Ray 3 Crosses 2 paths -> "EVEN NUMBER" -> Point B is "outside" -> don't fill

Trial Ray 4 Crosses 2 paths -> "EVEN NUMBER" -> Point B is "outside" -> don't fill

Trial Ray 2 Crosses 3 Paths -> "ODD NUMBER" -> Point A is "inside" -> Fill

General polygons

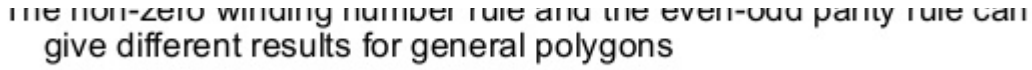
Can be self intersecting
Can have interior holes



Even-odd parity

Non-zero winding

The non-zero winding number rule and the even-odd parity rule can



9.图片设置宽高

- 网页中的图片，如果没有设置width和height，在图片载入之前，他所占的空间为0，但是当他加载完毕之后，那块为0的空间突然被撑开了，这样会导致，他下面的元素重新排列和渲染，造成重绘（repaint）和回流（reflow也称为重排），影响dom性能
- 重绘：是一个元素外观的改变所触发的浏览器行为，例如改变visibility、outline、背景色等属性。浏览器会根据元素的新属性重新绘制，使元素呈现新的外观。重绘不会带来重新布局，并不一定伴随重排。
- 重排：DOM树的增删移动会出发重排，浏览器引擎布局过程是从上到下的，从左到右的过程。 所以，如果在body最前面插入一个元素，会导致整个文档的重新渲染，而在其后插入一个元素，则不会影响到前面的元素

DOM渲染

- HTML解析器将HTML解析为DOM树（html解析器最多支持20层同类型标记的嵌套）
- 同时会解析CSS中样式数据
- 附加，DOM树与样式数据创建出呈现树（呈现树与DOM树相对应，但是非一一对应，display:none不会处在在呈现树，呈现树还不包括位置大小信息）
- 布局（重排），计算位置大小，分配坐标（流式布局模型，从上到下，从左到右）
  - 从html元素（根呈现器）开始递归遍历，所有的呈现器都有layout或reflow方法，每一个呈现器都会调用子代的layout
  - 全局布局，影响所有呈现器的全局样式更改（html字体大小更改） 屏幕大小变化
  - 增量布局， 只对dirty呈现器布局（为了避免细小改动引发全局布局，浏览器会将发生更改的呈现器标记为dirty）（往往异步执行，webkit有用于增量布局的定时器，遍历呈现树，对dirty呈现器布局）
- 绘制，遍历呈现树，绘制每个节点
  - 增量绘制 某些呈现器发生了更改但是不会影响到整个树
  - 全局绘制