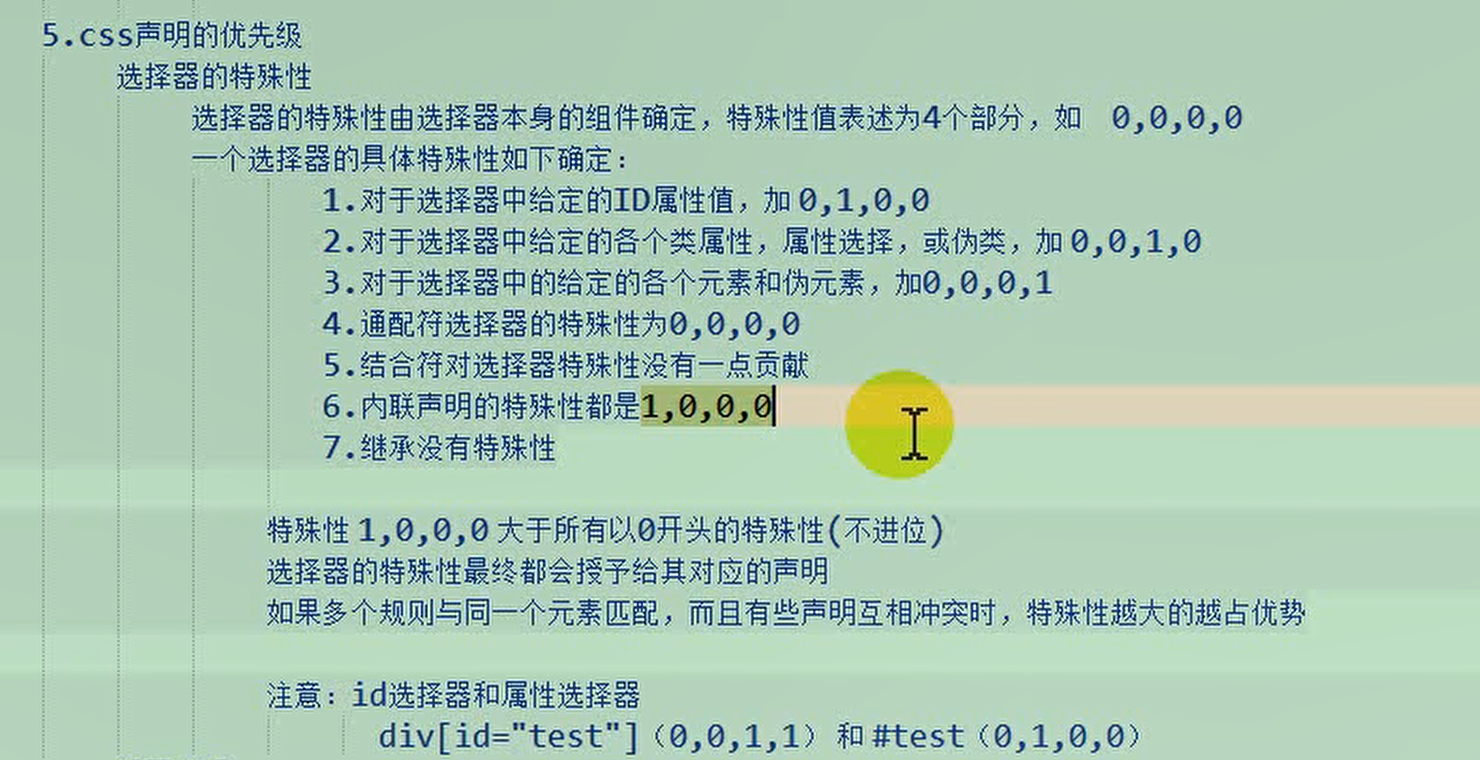
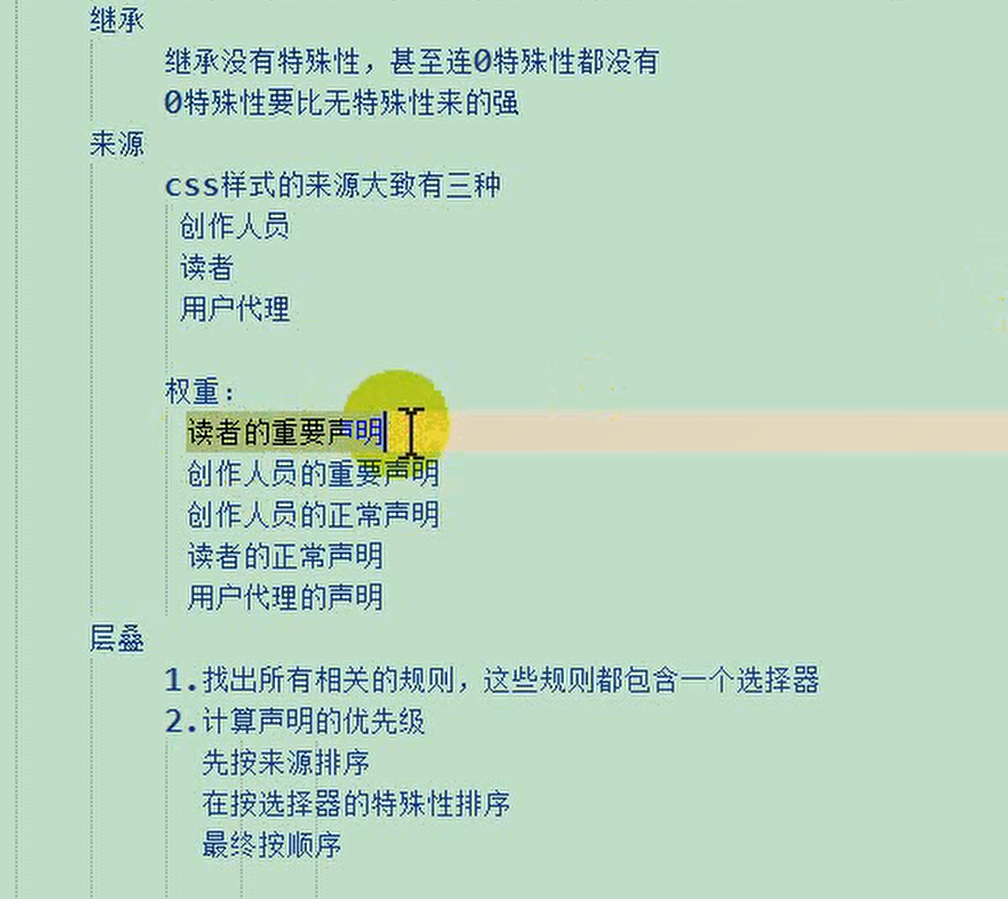
# CSS3的选择器

* 属性选择器
* 存在与值选择器：
  + div[name]
  + div[name=”xxxx”]
  + div[name~=”xxxxx”] →带有xxxxx的属性,且以空格作为分隔符的值
* 字串值选择器：
  + Div[name|=”xxxx”] →值为xxxx 或者以xxxx-开头的选择器
  + Div[name^=”xxxx”] →以xxxx开头的
  + Div[name$=”xxxx”] →以xxxx结尾
  + Div[name\*=“xxxx”] → 包含xxxx的元素
* 伪类与伪元素的选择器
  + 伪类的目的是拿到元素的状态
  + 伪元素的目的是拿到dom以外的元素
  + 链接伪类：
    - :link
    - :visited
    - :target →需要有a标签href指向一个div
  + 动态伪类
    - :hover
    - :active
  + 表单伪类：
    - :enabled
    - :disable →表单属性disable=”disabled”的时候生效
    - :checked →checkbox，redio可以用
    - :focus
    - 练习题：自定义单选按钮
  + 结构性伪类
    - :Nth-child（index）系列
      * Index是从1开始找的！！！！！
      * .wrap li:nth-child(1) → 找到.wrap底下的所有子元素，并且选中第一个子元素，并且这个子元素是li，才能被选中
      * :First-child
        + .wrap:first-child →找到是第一个元素的.wrap
      * :Last-child
        + .wrap:last-child →找到是最后一个元素的.wrap
      * :nth-last-child(index) → nth-child()反过来
      * :Only-child → 只能是独生子的元素，不能有兄弟元素
    - :Nth-of-type(index)系列
      * Index是从1开始找的！！！
      * .wrap li:nth-of-type(1) → 找到wrap底下的第一个li子元素。
      * p:first-of-type →找到第一个p，不管在哪个位置
      * p:last-of-type →找到最后一个p，不管在哪个位置
      * p:nth-last-of-type（index） → 找倒数第一个p，不管在哪个位置
      * p:only-of-type 只能找独生的p,存在两个则不行。
    - nth-child系列和nth-of-type系列的重要区别
      * nth-of-type是以元素为中心进行选择，所以最好不要用class进行选择。，用class的话会把class所在的元素全部选中。
      * div a:not(:last-of-type) 选择div下的所有的a，但是不选择最后一个
      * :empty 选择空的元素，必须是全空，有空格都不行。
    - Index可以有变量，可以有n ，2n ,2n+1 ,even,odd
  + 伪元素选择器
    - 伪元素最好用双冒号：： 单冒号也行。
    - 一个元素只有一个before和after，多写会覆盖
    - ::before
    - ::after
    - ::firstLetter
    - ::firstLine
    - ::section → 选中的部分进行渲染。

# 声明的优先级

* 
* 
* 计算声明的优先级：来源（一般都是开发人员） →选择器特殊性（决定优先级的最重要因素） → 声明顺序（最容易出bug的地方）

# 自定义字体

* + - * @Font-face{font-family:”xxx”; src:url()}
      * 好处：允许网页开发者为其网页指定在线字体，可以消除用户对本机电脑的依赖性
      * 坏处：相比本机字体增加网络负担
* 字体图标
* 可以用矢量图做出字体图标，不失真
* 相对于图片图标，较少网络负担。
* 制作字体图标流程：
  + 画出矢量图
  + 将矢量图与字符绑定（可以上网站上自动生成一系列字体图标包）
  + 矢量图与字符的绑定通过fony-face属性，class属性，伪元素的before属性进行绑定。

# 新增UI样式

* 文本新增样式
* Opacity
* 新增颜色模式Rgba → 在rgb(xx,xx,xx)中多加一个属性 rgba(xx,xx,xx,xx) 选择透明度
  + 可以用在背景透明，字体不透明这类情况
* Text-shadow 文字阴影
  + text-shadow:color offset-x offset-y blur-radius
    - 可以添加多层阴影，用逗号隔开
  + 浮雕文字
    - Color:white; text-shadow:black 1px 1px 10px
  + 文字模糊
    - H1{color:black ; transition:1s} h1:hover{color:white; text-shadow:black 0px 0px 200px }
* 模糊背景：
  + 配合一层半透明图片（rgba），和一层不透明背景图用filter:blur(xxpx)，z-index进行背景模糊
* 文字描边：
  + H1{Color:white ;Text-stroke :black 4px}
* 文字排版
  + Direction:ltr or rtr
* 溢出显示省略号：
  + 条件1：White-space:nowrap(不换行) ；overflow:hidden; text-overflow:ellipsis; (绑在一起)
  + 条件2：盒子不能靠内容撑开，只能设置宽高。
* 盒模型新增样式
* 盒模型阴影：box-shadow
  + 可选值：
    - Inset 阴影在边框内
    - Offset-x
    - Offset-y
    - Blur-radius 模糊程度（无法复制）
    - Spread-radius 阴影大小的值（正值时阴影放大，负值阴影缩小）
    - Color
* 倒影：box-reflect
  + 一般只有webkit才能使用（谷歌浏览器）
  + 可选值：
    - Left , right
    - 倒影之间的距离
* Resize：允许用户调整元素的大小
  + 需要配合Overflow:auto 一起使用
  + 可选值：
    - Both
    - Horizontal
    - Vertical
* Box-sizing
  + 可选值：
    - Content-box（默认值）
    - border-box
      * 设置为border-box后，width和height的值将包括，content,padding,border
      * 此方法可以解决加上padding后，盒子可视区发生变化的问题
* 新增·UI样式元素（border-radius,border-image,background）
* Border-radius:圆角
  + 可选值：
    - 圆角的形成是在盒模型的中心画所设置的数值为半径的园，然后将其慢慢扩张
    - Xxpx：四个角全部设置
    - Xxpx yypx ：xx代表正斜杠角，yy代表反斜杠的角
    - Xxpx yypx zzpx ：xx代表左上，yy代表反斜杠，zz代表右下
    - Xxpx yypx zzpx jjpx ：左上，右上，右下，左下。
    - Xxpx/yypx ：画一个横轴半径为xxpx 纵轴半径为yypx的椭圆 然后往四个角设置
  + 画椭圆一般都是先画一个长方形再设置border-radius为xxpx/yypx（xx为宽的一半，yy为长的一半）
  + 在移动端开发尽量不要使用百分比设置，因为大多数不支持
  + 风车练习：
    - 画一个风车，详情参照代码：css3>风车练习.html
  + 对话框练习：
    - 画一个对话框，详情参照代码：css3> 对话框练习.html
* 边框图片：border-image
  + 元素：
    - border-image-source：选择图片
    - border-image-slice:对图片进行切割
    - border-image-repeat:round 可以平铺
    - border-image-width: 图片的宽度发生变化，平铺时，数量会变多
    - border-image-outset: 整体图形发生大小变化
* css3的背景图片：
  + 附：css2的背景属性有:background-image,background-position.background-repeat,background-attachment,background-color
  + background-origin:设置背景渲染的起始位置
    - 可选值：
      * Padding-box（默认位置）
      * Border-box
      * Content-box
  + Background-clip：设置背景图片从哪里开始剪裁
    - 可选值：
      * Border-box（默认值）
      * Content-box
      * Padding-box
      * Text(只能用于-webkit)
  + Background-size:设置图片的大小
    - 可选值
      * Cover覆盖页面，配合fixed定位使用有奇效
      * Auto，图片自身大小
* 线性渐变：
  + - * 渐变是背景图片，非颜色
        + Background-image:linear-gradient(red,green,blue) 线性渐变（默认从上到下）
        + 可以指定方向渐变，linear-gradient(to left ,red,green,blue) 也可以设置to right ,to top ,to bottom
        + 可以指定角度渐变

，linear-gradient(0 deg ,red,green,blue) ：从上到下渐变

linear-gradient(90 deg ,red,green,blue) 从左到右渐变

* + - * + 可以指定颜色的长度

linear-gradient(to left ,red 10px,green 20px,blue 30px) 红色长10px 绿色长20px 蓝色长 30px

* + - * + 颜色透明度也可以指定渐变

linear-gradient(to left ,rgba(0,0,0,0),ragb(0,0,0,1)) 黑色颜色从完全透明到完全不透明进行渐变

* + - * + 渐变也可以重复

Repeating-linear-gradient(to left ,red,green,blue)

* + - * + 渐变有长度，其长度为最后一个渐变色的长度减去第一个渐变色的长度
        + 发廊灯练习：

黑白渐变

使用js的setinterval来设定持续动画

* + - * + 光斑动画：

黑色底色，字体白色，半透明

光斑的设置需要用到linear-gradient，然后从白色透明到白色不透明再到白色透明

渐变需要从Text进行裁剪（background-clip的属性）

渐变也是背景图片，为了不出现两个光斑，需要设置no-repeat

* 径向渐变
* 默认是以椭圆(ellipse)进行扩散的
  + 可以设置为circle
    - Radial-gradient(circle,yellow 10%,green 30%, pink 70%)
    - 可以设置以哪个边,哪个角进行扩散（closest-side ,fatherst-side，closest-corner,fatherest-corner（默认值））：
      * Radial-gradient(circle，closest-side,yellow 10%,green 30%, pink 70%)
    - 可以设置圆心的位置进行扩散（circle at xxpx yypx）
      * Radial-gradient(fatherest-corner,circle at 100px 100px,yellow 10%,green 30%, pink 70%)

# 过渡

Transition-property 指定属性进行过渡，默认值为all。，可指定多个 :background,height,width

Transition-duration 指定过渡的所用时间，可指定多个 :5s ,2s,5s

Transition-time-function : 默认为ease

Ease （加速然后减速）

Liner(匀速)

Ease-in（加速）

Ease-out(减速)

Ease-in-out(加速，然后减速)

Cubic-bezier(贝赛尔曲线)

Transition-delay: 延迟过渡开始的时间

Transition在元素首次渲染还没有结束的情况下是不会触发的

Windows.onload（）是在整个页面的css和html渲染已经结束的时候的调用的，也就是说在onload里面设置某元素的长度过渡的时候是可以显示在页面上的，而在外面的话是没有过渡的 。

Transition的简写有顺序要求：

第一个时间为duration

第二个时间为delay

第一个非时间为property

不同属性逗号隔开如：2s witdh,3s height

# 变形

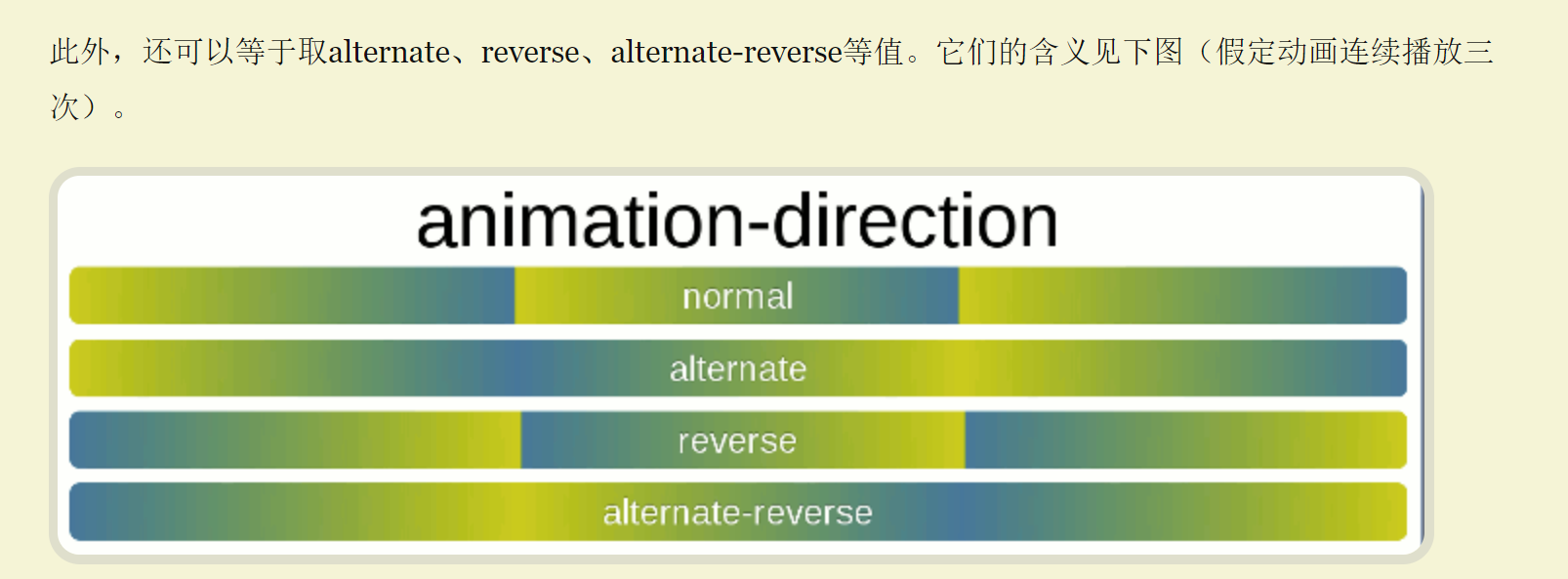
* Rotate(360deg) 旋转
* 平移
  + TranslateX(300px)
  + translateY(200px)
  + Translate（200px,200px） 斜着走
* 倾斜
  + Skew（45deg），倾斜45度 鸡肋，不怎用
* 缩放
  + Scale(2) 放大两倍
* 变形的基点：
  + Transform-origin : 默认为50% 50% 50%
  + 按照左上角开始算，左上角为0% 0%
  + 右下角为100% 100%
  + 除了位移，都可以用到基点’
* 变换组合：
  + 先缩放再位移 和 先位移再缩放的变形是不一样的。（除了缩放和旋转的变形组合都是不一样的）
    - 因为变换组合时，变换的选项时是从右往左的
    - 底层是根据矩阵去变换的
* 扇形导航实际操作

  + 需要理解的数学知识： 勾股定理，弧度与角度的转换，三角函数
  + 进行绑定过度事件时，如果其他地方不需要使用需要进行解绑
* 时钟的实际操作
  + 表盘为大div，时针，分针，秒针，中心都为小div
  + 每个刻度都是用li来表示，用Js布局和样式
  + 刻度在5n+1的时候变长
  + 获取时间用date函数，然后获取到的时间去进行角度变化。

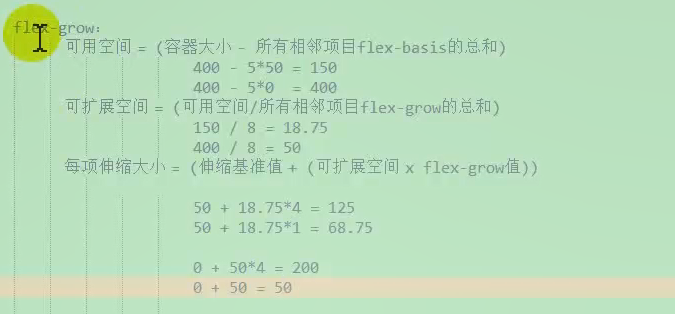
# 3d变形：

* + - * 3d变形一般配合景深（perspective）一起使用，景深相当于肉眼到屏幕的距离
        + 景深作用：构建3d舞台，决定用户的肉眼距离到屏幕的水平距离！
        + 景深越大，灭点越远，元素变形越小
        + 景深是一个不可继承属性，但是可以作用于所有的后代元素
      * 3d旋转
        + Rotate(1,1,1,deg) 绕着x,y,z都转
        + RotateX():绕着杠杆转
        + RotateY()：绕着钢管
        + RotateZ()：圆盘在转
      * 3d平移
        + TranslateZ()：因为元素没有厚度，所以不能写100%
        + Translate3D(x,y,z)：如果景深为100px.，此时z为-100px时，元素就会缩小一倍（营造近大远小的感觉）
      * 3d缩放
        + scaleZ（），单独使用没有任何意思，因为元素没有厚度
        + scaleZ配合translateZ一起使用时，先translate的话，scaleZ也会生效
      * 景深：越大，灭点越远，元素的变形幅度越小(作用于所有后代元素，不可继承)
        + 决定用户的肉眼距离屏幕的水平距离！
        + 景深的基点：perspective-origin :默认为50% 50%
        + 景深不要写在多处父元素上，景深会有叠加的
      * Transform-style:perspective-3d 营造有层级的3d舞台，如：可以让旋转转到背景的后面((写在父元素，作用于子元素，不可继承，一般与景深写在同一个地方)
      * Backface-visibility
        + 隐藏背面的元素
      * 立方体实战
      * 立方体如果需要隐藏背面可以用backface-visibility : hidden
      * 三棱柱实战，然后封装做出多棱柱，笔记总结
        + 棱柱的内角等于180-360/n
        + 棱柱的外角等于 360/n
        + 改变各个面的基点来旋转各个面
        + 改变棱柱的基点来自转整个棱柱
        + 以上两个基点是同一个，
        + 棱柱的Z轴基点是用offsetwidth的一半和tan(内角一半)求出来的

# 动画（animation）

* 可以用来替代transform，弥补transform的不足
* 动画属性：
  + Animation-name : 关键帧(keyframe)的名字
  + Animation-duration : 动画执行的时间，默认为0
  + Animation-time-function : 动画执行效果，如：先快后慢
  + Animation-delay （动画外的属性）
  + Anitionmation-iteration-count 重复关键帧 指定执行几次动画（动画外的属性）
  + Animation-direction 改变关键帧的执行方向和Animation-time-function :
    - (默认是nomal)from >>to
    - Reverse (to>> from)
    - Alternate (from >> to >>from)
    - Alternate-reverse (to >> from >u> to )
    - 
  + Animation-fill-mode:控制元素在动画外的状态：
    - Backwards动画结束后，让动画回到第一帧的状态。
    - Forwards 动画结束后，forwards表示让动画停留在结束状态
    - both: 根据animation-direction（见后）轮流应用forwards和backwards规则。
  + animation-play-state:paused 暂停动画(默认是running)
* 动画关键帧（keyframe）：
  + Animation-time-function的动画效果是在每一帧生效，有几帧生效几次

# Flex布局：

* 老版本容器
* 老版本弹性容器：display :-webkit-box
* -webkit-orient （控制主轴是谁）: horizontal 主轴是水平轴
* -webkit-box-direction（决定主轴的方向） : reverse 排列方向是反着的
* -webkit-box-pack (主轴富裕空间管理) ：
  + start(项目靠近开始位置，富裕空间在后边)
  + end 项目靠近结束位置，富裕空间在前边
  + center 项目靠中，富裕空间在两边
  + justify 项目间隔排列，富裕空间在项目间隔
* -webkit-box-align（侧轴富裕空间管理） :
  + Start 富裕空间在下边
  + End 富裕空间在上边
  + Center 富裕空间在两边
* 新版本容器
* 新版本弹性容器：display : flex
* Flex-direction :
  + Row
  + Column
  + Row-reverse
  + Column-reverse
* Justify-content :
  + Flex-start ：富裕空间在主轴的正方向
  + Flex-end ：富裕空间在主轴的反方向
  + Center ：富裕空间在主轴的两边
  + Space-between ： 富裕空间在项目之间
  + Space-around ：(旧版本没有的) ：富裕空间在项目两边
* Align-items:
  + Flex-start ：富裕空间在侧轴的正方向
  + Flex-end ： 富裕空间在侧轴的反方向
  + Center ： 富裕空间在侧轴的两边
  + Baseline(旧版本没有的) ： 按基线对齐
  + Stretch(旧版本没有的) ： 等高布局（设置项目的高度，也会拉伸到容器的高度）
* Flex基础点：
  + 容器，项目，主轴，侧轴 是理解flex的关键
  + 项目永远排列在主轴的正方向上
* 富裕空间的管理
  + 只决定富裕空间的位置，不会给项目区分配空间
  + 主轴
    - Justify-content
      * Flex-start :在主轴的正方向
      * Flex-end 在主轴的反方向
      * Center : 在两边
      * Space-between 在项目之间
      * Space-around 在项目两边
  + 侧轴
    - Align-items
      * Start 在侧轴的正方向
      * End 在侧轴的反方向
      * Center 在两边
      * Baseline
      * Stretch
* 实战：老版本容器项目（弹性空间管理）
  + 弹性空间管理：-webkit-box-flex ：1 （弹性因子） 默认0
  + 新版本弹性因子为flex-grow 默认0
  + 将主轴上的富裕空间按比例分配到项目的width（主轴是x）/height（主轴是y）上
* 新版本flex布局详解
* Flex-wrap :控制侧轴的方向
  + Wrap
  + No-wrap
  + Wrap-reverse
* Align-content 多行多列时才有用，否则align-items生效，富裕空间的管理，会把所有行，列看成一个整体
* Flex-flow是flex-direction和flex-wrap的简写，也就是同时控制主轴，副轴的方向
* 不止容器，项目也有属性：
  + Order :给项目进行排序(在具体项目中给属性)
  + Align-self 管理单个项目的侧轴方向上的富裕空间
  + Flex-shrink 收缩因子
    - 计算收缩因子于基准值乘积的总和：a=每一项项目的flex-shrink\*flex-basis 之和
    - 计算收缩因子 b=flex-shrink\*flex-basis/a
    - 移除空间的计算 : c= b\*溢出空间
  + Flex-basis ：伸缩基准值，如果不写，默认就是项目的长宽。配合flex-grow一起使用
    - 
    - 上图中，flex-grow =8，且第一行项目宽为50，第二行数据项目flex-basis为０
* Flex简写：
  + Flex ：1 相当于 flex-grow :1 ;flex-shrink:1 ；flex-basis :0%
    - 可以等宽

# 响应式布局

1. Css3的媒体查询
   1. Link 中的media属性
      1. Print是在打印才显示
      2. Screen是打印时不会显示
   2. Css中的媒体查询选择器（规则，非常规选择器）
      1. 媒体选择器类型和属性
         1. 类型：
            1. @media screen{} 在选择器中具体在写各种Html标签的选择器。
            2. All 所有设备
            3. Print 打印
            4. Projection 手持设备
            5. Tv 电视
            6. Braille 盲文触觉设备
            7. Embossed 盲文打印机
            8. Speech 听觉类似的媒体设备
            9. Tty 不适用像素的设备
         2. @media width
         3. @meida min-width
         4. @media max-width
         5. @media device-width：414px （表示设备的分辨率的width为414px时生效，如果iphone7）
         6. @media min-device-width:
         7. @media max-device-width
         8. @media device-pixel-ratio(必须加-webit- 前缀) 像素比
            1. Pc端 ：1
            2. 移动端看手机参数，一般大于1
         9. @media orientation
            1. 竖屏 ：portrait （竖屏的时候生效）
            2. 横屏 ： landscape (横屏的时候生效)
      2. 媒体选择器之间的关键字
         1. And 表示且的意思，一般用and来连接媒体类型和媒体属性
         2. Only ：和浏览器兼容性有关
            1. 老版本的浏览器只支持媒体类型，不支持带媒体属性的查询
            2. 所以老版本如果不加only的时候，只解析类型，不解析属性，一般加only来让老版本浏览器忽略此选择器
         3. ， ：表示Or
         4. Not : @media not screen 表示只要不是彩色屏幕

# 多列布局（不经常用）

1. Column-width 每个栏的宽度
2. Column-count 栏的个数
3. Column-gap 栏之间的间隙大小
4. Column-rule 栏的分割线

# 网格布局（新布局样式）

基础概念

1. 跟flex一样，也有容器和项目之分
2. 项目只能是容器的顶层子元素，不包含项目内的元素
3. 容器水平区域为行“row” ，垂直区域为列 “column
4. 行列交叉的地方为单元格（cell）
5. 划分网格的线为网格线 （grid line）
   1. 正常情况下，n行就有n+1根网格线

容器

1. Display : grid 默认情况下是块级元素，但是也可以把容器当成行内元素进行处理： display : inline-grid
2. 设定网格布局以后，容器内的项目的下列属性将失效
   1. float、
   2. display: inline-block、
   3. display: table-cell、
   4. vertical-align
   5. column-\*
3. grid-template-rows grid-template-columns来划分列和行
   1. 下面代码表示分成三行,每行100px，三列，每列100px，9个单元格，每个单元格100px”100px
      1. grid-template-rows : 100px 100px 100px；
      2. grid-template-rows : 100px 100px 100px；
   2. 也可以使用百分单位，下面代码表示分成等分成4行
      1. 如：grid-template-rows : 25% 25% 25% 25%
   3. 也可以使用repeat（）来减少重复值的指定，参数1 是重复的次数，参数2是重复的值
      1. 表示分成等分成5行 ：
         1. Grid-template-rows : repeat(5,20%)
      2. 表示按20px 40px的模式分成4份，共8行 ：
         1. grid-template-rows :repeat(4, 20px 40px )
   4. repeat可以配合auto-fill使用
      1. 表示每列宽度100px,然后自动填充
         1. Grid-template-column : repeat(auto-fill, 100px)
      2. 表示列的宽度会随着容器的大小改变，项目最小宽度为60px，在可以插入一个60px宽的列之前，当前列会一直拉伸，当容器大小大于项目之和时，容器会持续加入空行或者空列，这样就会使所有项目挤到另一边。
         1. Grid-template-column : repeat(auto-fill, minmax(60px,1fr))
   5. repeat可以配合auto-fit使用
      1. auto-fit效果几乎和auto-fill一样。不同点仅在于，当容器的大小大于各项目之和时，auto-fill将会持续地在一端放入空行或空列，这样就会使所有项目挤到另一边；而auto-fit则不会在一端放入空行或空列，而是会将所有项目拉伸至合适的大小。
      2. 当容器大小大于项目之和时，会拉伸所有项目
         1. Grid-template-column : repeat(auto-fill, minmax(60px,1fr))
   6. 还可以使用fr关键字来指定每行每列的宽度
      1. Fr表示比例关系（fraction片段）
      2. 表示第一列是第二列的两倍宽度
         1. Grid-template-columns : 2fr 1fr
      3. Fr可以配合绝对长度一起使用，表示100px以后的空间分成三份，2fr两份，1fr一份
         1. Grid-template-rows : 100px 2fr 1fr
   7. 也可以使用minmax()来产生一个长度范围，参数1最小值，参数2最大值
      1. 表示1，2列按比例分配，第三列宽度可变
         1. Grid-template-columns : 1fr 1fr minmax(100px,1fr)
   8. 也可以用auto让浏览器来决定长度
      1. 表示两边固定长度，第二列由浏览器决定(此时，基本上等于该列单元格的最大宽度)
         1. Grid-template-colunms : 100px auto 100px
   9. 网格线名称，用[ ]来表明
      1. 表示列中的每个网格线
         1. Grid-template-columns : [c1] 100px [c2] 1fr [c3] 1r [c4]
      2. 网格布局允许一条网格线有多个名字
   10. 网格布局对网页布局非常方便，两列布局只需要一行代码
       1. Grid-template-columns : 70% 30%
4. Grid-row-gap grid-column-gap grid-gap属性
   1. 设置行间距或者列间距的属性
   2. Grid-gap是grid-row-gap 和grid-column-gap的合并
      1. 表示行间距20px 列间距30px
         1. Grid-gap : 20px 30px
   3. 如果grid-gap省略第二个值，则默认第二个值等于第一个值，也就是行列间距都是同一个值
5. grid-template-areas
   1. 网格允许指定区域，一个区域由单个或者多个单元格组成·
      1. 表示6个单元格分别为6个区域，名字为abcdef
         1. Grid-template-columns : 100px 100px 100px
         2. Grid-template-rows : 100px 100px
         3. Grid-tempalte-areas: ‘a b c’  
             ‘d e f’ ;
      2. 如果需要多个单元格设为一个区域，则把其设置成同一个名字就行
      3. 如果需要某个单元格不属于某个区域，则把其设置成 .
   2. 区域的命名，会影响到网格线的命名，
      1. 每个区域的起始网格线为 区域名-start
      2. 每个区域的终止网格线为 区域名-end
6. Grid-auto-flow属性
   1. 决定单元格排列的顺序
   2. 默认值row ：先行后列
   3. Column ：先列后行
   4. row dense : 先行后列，并且尽量紧密
   5. column dense ： 先列后行，并且尽量紧密
7. justify-items align-items place-items属性
   1. justify-items设置所有单元格里面的内容的水平位置（左中右）
      1. start
      2. end
      3. center
      4. stretch 默认值
   2. align-items设置所有单元格里面的内容的垂直位置(上中下)
      1. start
      2. end
      3. center
      4. stretch 默认值
   3. place-items是上面两个属性的合并
      1. 表示水平居中，垂直居中
         1. Place-items : center center
      2. 如果省略第二个值，则默认与第一个值相等
8. Justify-content align-content place-content 属性
   1. Justify-content是整个内容区域在容器中的水平位置
      1. Start 默认值
      2. End
      3. Center
      4. Stretch : 项目大小没有指定时，拉伸占据整个网格容器。
      5. Space-around:
         1. 每个项目两侧的间隔相等。所以，项目之间的间隔比项目与容器边框的间隔大一倍。
      6. Space-between :
         1. 项目与项目的间隔相等，项目与容器边框之间没有间隔。
      7. Space-evenly
         1. 项目与项目的间隔相等，项目与容器边框之间也是同样长度的间隔。
   2. Align-content是整个内容区域在容器中的垂直位置
   3. Place-content是上面两者的缩写
9. Grid-auto-columns grid-auto-rows 属性
   1. 有时候，一些项目的指定位置，在现有网格的外部。比如网格只有3列，但是某一个项目指定在第5行。这时，浏览器会自动生成多余的网格，以便放置项目。
   2. 如果不指定grid-auto-columns grid-auto-rows的话，浏览器默认按照多余网格上的项目大小生成网格
   3. 此属性用法与grid-template-columns grid-template-rows完全一样
10. Grid-template属性是grid-template-colunms grid-template-rows grid-template-area属性的简写
    1. 行列中间用/隔开 如 300px 300px 300px / 100px 100px 100px
11. Grid属性是grid-template-rows、grid-template-columns、grid-template-areas、 grid-auto-rows、grid-auto-columns、grid-auto-flow这六个属性的合并简写形式。

项目属性

1. Grid-column-start grid-column-end grid-row-start grid-row-end属性
   1. 属性基本
      1. Grid-column-start : 左边框所在的垂直网格线
      2. Grid-column-end : 右边框所在的垂直网格线
      3. Grid-row-start ： 上边框所在的水平网格线
      4. Grid-row-end ： 下边框所在的水平网格线
   2. 下列两行代码表示项目的位置在垂直第二根网格线和第四根网格线之间，水平网格线默认（1和2之间，即第一行）
      1. Grid-column-start : 2
      2. Grid-column-start : 4
   3. 此属性还能指定网格线的名字（如果在grid-template-row[columns]上有设置网格线的名字的话）
   4. 此属性还能使用span，表示跨越多条网格线
      1. 下面代码表示项目水平上从第一根网格线开始，跨越两条网格线到第三根网格线
         1. Grid-row-start : 1
         2. Grid-row-end : span 2
      2. Span指定网格线位置的优先级小于数字指定，
         1. 如果start,end其中一个有数字指定的话，就以数字指定的为主，进行跨越。如果没有数字，以默认所在的网格位置进行跨越，如果跨越幅度超过指定网格区域的话，浏览器不会自动创建新网格，而是换行从以start 为1进行跨越
      3. Start和end都是span指定的情况下
         1. 只参照start，无视end
2. Grid-column grid-row属性
   1. grid-column属性是grid-column-start和grid-column-end的合并简写形式，
   2. grid-row属性是grid-row-start属性和grid-row-end的合并简写形式。
   3. 下列1代码表示项目的位置在垂直第二根网格线和第四根网格线之间，水平网格线默认（1和2之间，即第一行）
      1. Grid-column ：2 / 4
   4. 斜杆跨越省略，默认跨越一个网格线
      1. 表示占据左上角第一个单元格
         1. Grid-column : 1
         2. Grid-row : 1
3. Grid-area 属性
   1. 指定放在哪个区域（如果在grid-template-areas中有定义区域的话）
   2. Grid-area还可以用作grid-column-start grid-column-end grid-row-start grid-row-end 这四个属性的缩写属性 ，直接精确指定项目的位置
      1. 表示项目在第二行第三个
         1. Grid-area : 2/3/3/4
            1. 如果end 是start+1的话 end可以省略不写

所以Grid-area ：2/3

1. Justify-self align-self place-self 属性
   1. 设置单个单元格里面的内容的水平，垂直位置
   2. 与justify-items align-items place-items的用法完全一样。