谈谈优雅降级与渐进增强的区别。

渐进增强 progressive enhancement：针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进和追加功能达到更好的用户体验。

优雅降级 graceful degradation：一开始就构建完整的功能，然后再针对低版本浏览器进行兼容。

区别：优雅降级是从复杂的现状开始，并试图减少用户体验的供给，而渐进增强则是从一个非常基础的，能够起作用的版本开始，并不断扩充，以适应未来环境的需要。降级（功能衰减）意味着往回看；而渐进增强则意味着朝前看，同时保证其根基处于安全地带。

描述 cookies，sessionStorage 和 localStorage 的区别。

特性	Cookie	localStorage	sessionStorage
数据的生命期	可设置失效时间，默认是关闭浏览器后失效	除非被清除，否则永久保存	仅在当前会话下有效，关闭页面或浏览器后被清除
存放数据大小	4K左右	一般为5MB	一般为5MB
与服务器端通信	每次都会携带在HTTP头中，如果使用cookie保存过多数据会带来性能问题	仅在客户端（即浏览器）中保存，不参与和服务器的通信	仅在客户端（即浏览器）中保存，不参与和服务器的通信
易用性	需要程序员自己封装，源生的Cookie接口不友好	源生接口可以接受，亦可再次封装来对Object和Array有更好的支持	源生接口可以接受，亦可再次封装来对Object和Array有更好的支持

如何设计开发一个图片轮播组件？简述要点或写代码。

参考答案

给出一个比较通用的方法。

① 整个组件采用 ViewPager

② 适配器继承自 PagerAdapter

③ 重写 getCount()，isViewFromObject(View arg0, Object arg1)，destroyItem(ViewGroup container, int position, Object object)，instantiateItem(ViewGroup container, int position) 四个方法。

④ getCount 代表返回的条目，要实现无限轮播，这里就要给出一个很大的值，我们可以采 Integer.MAX_VALUE。其他的和普通 ViewPager 开发一样，在 isViewFromObject 返回 arg0 == arg1，在 destroyItem 中摧毁滑出的 View,container.removeView((View) object),在 instantiateItem 中添加对应的 item,记得添加 item,container.addView(child)。里面的 postion 都要做取余处理，避免数组越界。

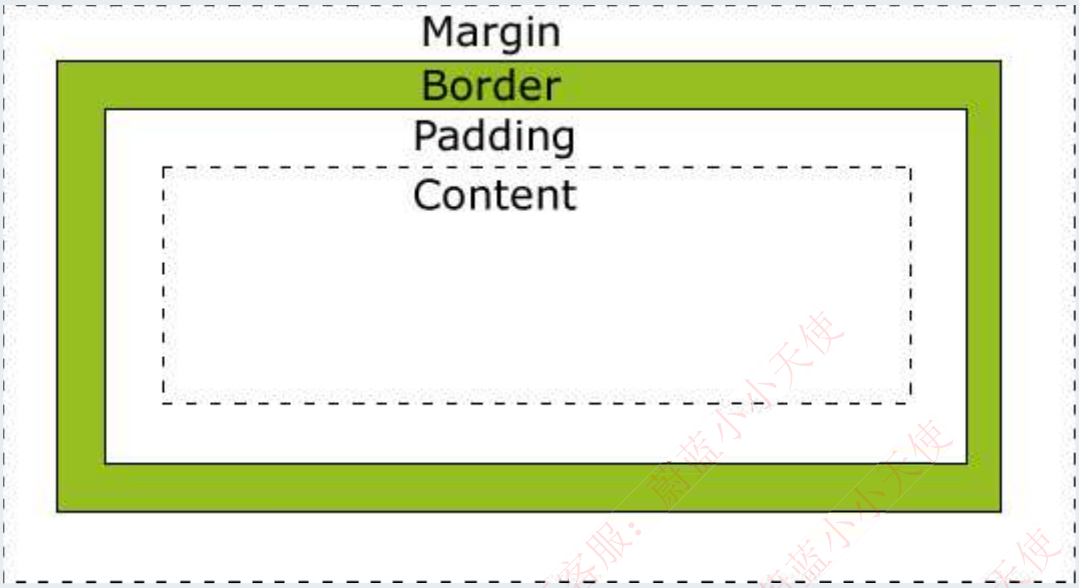
⑤ 在一开始展示的时候，把 item 定位到较中间的位置，这里我采用 vp.setCurrentItem(10000 * ids.length)。当然这样只是一个滑不到边界的轮播，并不是真正的首尾相连的轮播。

⑥ 最后，可以使用一个 handler 实现自动轮播，重写 onTouchEvent 来对自动轮播控制。还可以将这些全部封装起来，当一个自定义 view 使用。

简要解释盒模型、行内元素与块级元素的概念。

参考答案

盒模型是 CSS 的基石之一，它指定元素如何显示以及如何相互交互。页面上的每个元素被看做一个矩形框，这个框由元素的内容、内边距、边框和外边距组成。
下面的图片说明了盒子模型：



在做页面布局的时候,一般会将 html 元素分为两种，即块级元素和行内元素。
块级元素：块状元素排斥其他元素与其位于同一行，可以设定元素的宽（width）和高（height），块级元素一般是其他元素的容器，可容纳块级元素和行内元素。常见的块级元素有 div, p ,h1~h6 等。
行内元素：行内元素不可以设置宽（width）和高（height），但可以与其他行内元素位于同一行，行内元素内一般不可以包含块级元素。行内元素的高度一般由元素内部的字体大小决定，宽度由内容的长度控制。常见的行内元素有 a, em ,strong 等。

如何进行前端性能优化？简述几种常用的方法。
参考答案

代码层面：避免使用 css 表达式，避免使用高级选择器，通配选择器。
缓存利用：缓存 Ajax，使用 CDN，使用外部 js 和 css 文件以便缓存，添加 Expires 头，服务端配置 Etag，减少 DNS 查找等
请求数量：合并样式和脚本，使用 css 图片精灵，初始首屏之外的图片资源按需加载，静态资源延迟加载。
请求带宽：压缩文件，开启 GZIP，
代码层面的优化

- 用 hash-table 来优化查找
- 少用全局变量
- 用 innerHTML 代替 DOM 操作，减少 DOM 操作次数，优化 javascript 性能
- 用 setTimeout 来避免页面失去响应
- 缓存 DOM 节点查找的结果
- 避免使用 CSS Expression
- 避免全局查询
- 避免使用 with(with 会创建自己的作用域，会增加作用域链长度)

- 多个变量声明合并
- 避免图片和 iFrame 等的空 Src。空 Src 会重新加载当前页面，影响速度和效率
- 尽量避免写在 HTML 标签中写 Style 属性

移动端性能优化

- 尽量使用 css3 动画，开启硬件加速。
- 适当使用 touch 事件代替 click 事件。
- 避免使用 css3 渐变阴影效果。
- 可以用 transform: translateZ(0) 来开启硬件加速。（见下面的详细解释）
- 不滥用 Float。Float 在渲染时计算量比较大，尽量减少使用
- 不滥用 Web 字体。Web 字体需要下载，解析，重绘当前页面，尽量减少使用。
- 合理使用 requestAnimationFrame 动画代替 setTimeout
- CSS 中的属性（CSS3 transitions、CSS3 3D transforms、Opacity、Canvas、WebGL、Video）会触发 GPU 渲染，请合理使用。过渡使用会引发手机过耗电增加
- PC 端的在移动端同样适用

给你两个集合，要求{A} + {B}。 注：同一个集合中不会有两个相同的元素。

```
#include <iostream>
#include <algorithm>
#define ARRSIZE 10000
using namespace std;

int main()
{
int n,m;
int arr[ARRSIZE*2];
cin >> n >> m;
int i = 0;
for (i = 0; i < n+m; i++) {
cin >> arr[i];
}
sort(arr,arr+n+m);
cout << arr[0];
for (i = 1; i < n+m; i++) {
if(arr[i] != arr[i-1])
cout<< " " << arr[i] ;
}
cout << endl;
```

```
return 0;
}
```

尽管是一个 CS 专业的学生，小 B 的数学基础很好并对数值计算有着特别的兴趣，喜欢用计算机程序来解决数学问题，现在，她正在玩一个数值变换的游戏。她发现计算机中经常用不同的进制表示一个数，如十进制数 123 表达为 16 进制时只包含两位数 7、11（B），用八进制表示为三位数 1、7、3，按不同进制表达时，各个位数的和也不同，如上述例子中十六进制和八进制中各位数的和分别是 18 和 11,。 小 B 感兴趣的是，一个数 A 如果按 2 到 A-1 进制表达时，各个位数之和的均值是多少？她希望你能帮她解决这个问题？ 所有的计算均基于十进制进行，结果也用十进制表示为不可约简的分数形式。

```
# 不仅考察进制换算，还考察最大公约数
def f1(n): # 进制换算，计数
    ret = 0
    for i in range(2,n): # 一次除以 2 至 n-1，相当于换算成 2 至 n-1 进制
        t = n
        while 1:
            ret += t % i # 将结果加入返回值
            t = t // i
            if t == 0:
                break
    return ret
def f2(n, m): # 求最大公约数
    ret = 1
    while m > 0:
        t = n % m
        n = m
        m = t
    ret = n
    return ret
if __name__ == '__main__':
    while 1:
        try:
            n = int(raw_input())
        except:
            break
        r = f1(n) # r 是总的进制换算相加的数
        l = len(range(2, n)) # 被除数 2 至（n-1）
        s = f2(r, l) # 最大公约数
        r , l = r//s, l//s # 结果除以最大公约数
        print str(r) + '/' + str(l)
```