By mnmlist,2015-9-1 11:22:43

>

如何向GitHub提交更改的代码

1.下载安装Git for windows 就不多说了，由Git Shell进入项目所在的目录中

2.首先，在该目录中创建一个新的responsitory，将会显示"Initialized empty git repository in ....git" (... is the path).

3.现在你需要通过将文件添加进repository来告诉git，通过git add filename来实现，如果你想添加所有的文件，你可以通过git add . 来实现；如果涉及到文件的删除，可能还需要git add . -A 来实现，这样被删除的文件也会被一同提交

4.现在你添加了文件并记录了变化，你需要提交这些改变，那样git就可以跟踪它们了，输入git commit -m "second_commit".-m允许你为这次提交添加备注信息。

目前来讲，上述的步骤是必须的，即使你没有用到github，它们是开始一个respository的正常步骤。由于git是分布式的，它意味着你为了使用git不必非要有一个中央服务器。

5.现在你想将这些改变同步到github上，你只需要告诉git添加一个远程路径，你可以通过以下命令实现它：

```
git remote add origin https://github.com/yourusername/your-repo-name.get
```

6.一旦你这样做了，git知道你的远端repository，你可以让它上传你提交的文件：

```
git push -u origin master
```

一般步骤1-6是可以将文件提交到远端github上，但并不总是这样，有时候你修改的时候还必须先pull一下，将远端服务器的代码pull到本地，没有pull的情况下有可能出现如下错误，比如：! [rejected] master -> master (fetch first)

它就是告诉你，要先fetch first，因为或许已经有人先你一步将代码push到master，你的提交是在后面的，因此你必须拉取远端代码，合并改变，然后你就可以再次提交了。如果你不想这样（或者你想强制提交，通过使用--force选项），你可能搞乱提交历史记录。

所以不建议用git push origin master -f来强制提交，而是先fetch，然后再改变，然后提交。

阿里巴巴2015实习生笔试真题

1.有两个N*N的矩阵A和B，想要在PC上按矩阵乘法基本算法编程实现计算A*B。假设N较大，本机内存也很大，可以存下A、B和结果矩阵。那么，为了计算速度，A和B在内存中应该如何存储（按行存指先存储第一行，再第二行，直到最后一行；按列存指先存储第一列，再第二列，直到最后一列）？

A.A按行存，B按行存。
B.A按行存，B按列存。
C.A按列存，B按行存。
D.A按列存，B按列存。

答案：A

想到的是传统矩阵相乘的方法，时间复杂度为O（n3 ），但是这不是最优的方法，最优方法为Strassen矩阵相乘发，时间复杂度降低为O（n2.81），用分治的思想将矩阵分块计算，在这个算法中按行存储更有利。所以正确答案为A。

2.IP数据报头采用（ ）字节序，在此字节序下从低地址到高地址0x1234的表示形式为（ ）。

A.big_endian,0x12 0x34 0 0
B.little_endian,0x34 0x12 0 0
C.big_endian,0 0 0x12 0x34
D.little_endian, 0 0 0x34 0x12

答案：C

big_endian：高位在低地址；
small-endian：地位在低地址。

3.设集合A={1,2,3},A上的关系R = {(1,1),(2,2),(2,3),(3,2),(3,3)}，则R不具备 ()?

A.自反性
B.传递性
C.对称性
D.反对称性

答案：D

假设集合A，以及基于A上的关系R
自反： 如果a是A的元素，那么<a,a>是R的元素
反自反： 如果a是A的元素，那么<a,a>不是R的元素
对称：如果<a,b>是R的元素，那么<b,a>是R的元素
反对称：如果<a,b>，<b,a>是R的元素，那么a,b相等
传递：如果<a,b>，<b,c>是R的元素，那么<a,c>是R的元素

4.无锁化编程有哪些常见方法？

A.针对计数器，可以使用原子加
B.只有一个生产者和一个消费者，那么就可以做到免锁访问环形缓冲区（Ring Buffer）
C.RCU（Read-Copy-Update），新旧副本切换机制，对于旧副本可以采用延迟释放的做法
D.CAS（Compare-and-Swap），如无锁栈，无锁队列等待

答案：D

A 这方法虽然不太好，但是常见
B ProducerConsumerQueue就是这个，到处都是
C linux kernel里面大量使用
D 本质上其实就是乐观锁，操作起来很困难。。单生产者多消费者或者多生产者单消费者的情况下比较常见，也不容易遇到ABA问题。

By mnmlist,2015-9-1 11:22:43

5. 以下措施中，不可能改进分布式系统读写(IO)性能的有____。

A.网络从千兆网升级为万兆网
B.优化调度系统，尽量做到任务与数据相近（Locality）
C.数据预取机制
D.实现异步读写机制

答案：D

异步IO就是调用系统IO来完成实际的IO操作，而不需要应用程序自己写代码完成IO，每次系统IO完成后给应用程序返回一个IO完成的信号，从而实现应用程序真的的异步无阻塞式的IO，异步IO可以提高应用程序的性能，而对操作系统IO没什么影响。影响分布式系统读写(IO)性能的关键因素应该是请求数据，而数据可能在别的机器上，所以ABC都能明显改善性能。至于D，感觉在非分布式系统上都已经出现了，貌似对 分布式系统读写(IO)性能的影响不大。

6.假定x=500，求下面函数的返回值_____ 。

```
int fun(int x)
{
   int countx = 0;
   while (x)
   {
       countx++;
       x = x & (x - 1);
   }
   return countx;
```

A.2
B.3
C.5
D.6

答案：D

很有意思的一道题目，500(d)=(111110100)b，由于当x为2的整数次幂时，x&(x-1)==0，所以，500的二进制表示中有多少个二进制位就会循环多少次，问题得解

By mnmlist,2015-9-1 11:22:43

面试题目-1

1.有关会话跟踪技术描述正确的是
a.Cookie是Web服务器发送给客户端的一小段信息，客户端请求时，可以读取该信息发送到服务器端
b.关闭浏览器意味着会话ID丢失，但所有与原会话关联的会话数据仍保留在服务器上，直至会话过期
c.在禁用Cookie时可以使用URL重写技术跟踪会话

答案：a,b,c

2.下面代码的输出结果是什么？

```
public class B
{
   public static B t1 = new B();
   public static B t2 = new B();
   {
       System.out.println("");
   }
   static
   {
       System.out.println("");
   }
   public static void main(String[] args)
   {
       B t = new B();
   }
}
```

答案：

staticJVM
{}
>main()>>
publicstaticB t1 = newB();
public static B t1 = new B();

static
{
System.out.println("");

By mnmlist,2015-9-1 11:22:43

}
main

3.下面不属于HttpServletRequest接口完成功能的是？
a.读取cookie
b.读取HTTP头
c.设定响应的content类型
d.读取路径信息
答案：c
A: request.getCookies();
B: request.getHeader(String s);
C: response.setContentType("text/html;charset=utf-8");
C: request.getContextPath();/request.getServletPath();

c

4.以下关于对象序列化描述正确的是
a.使用FileOutputStream可以将对象进行传输
b.使用PrintWriter可以将对象进行传输
c.使用transient修饰的变量不会被序列化
d.对象序列化的所属类需要实现Serializable接口

答案：c d

只有以Object开头的stream才可以传输对象，C和D transient这个单词本身的意思就是瞬时的意思 transient是变量修饰符 变量定义为transient的，序列化时会忽略此字段，所以C是对的， 只有实现了Serializable接口的，才可以被序列化

5.下面哪种情况会导致持久区jvm.

a.循环上万次的字符串处理
b.在一段代码内申请上百M甚至上G的内存
c.使用CGLib技术直接操作字节码运行，生成大量的动态类
d.不断创建对象

答案：c

简单的来说 java的堆内存分为两块:permantspace（持久带） 和 heap space。
持久带中主要存放用于存放静态类型数据，如 Java Class, Method 等， 与垃圾收集器要收集的Java对象关系不大。
而heapspace分为年轻带和年老带
年轻代的垃圾回收叫 Young GC， 年老代的垃圾回收叫 Full GC。
在年轻代中经历了N次（可配置）垃圾回收后仍然存活的对象，就会被复制到年老代中。因此，可以认为年老代中存放的都是一些生命周期较长的对象
年老代溢出原因有 循环上万次的字符串处理、创建上千万个对象、在一段代码内申请上百M甚至上G的内存，既A B D选项
持久代溢出原因 动态加载了大量Java类而导致溢出

a,b,d导致堆内存溢出，而c导致permantspace区域溢出，堆区域和permantspace区域不是一个区域，permantspace包括方法区，方法区主要包括类加载信息、字面量和常量等。

6.在 myjsp.jsp 中，关于下面的代码说法错误的是：（）
<%@ page language="java" import="java.util.*" errorPage="error.jsp" isErrorPage="false" %>

a.该页面可以使用 exception 对象
b.该页面发生异常会转向 error.jsp
c.存在 errorPage 属性时，isErrorPage 是默认为 false
d.error.jsp 页面一定要有isErrorPage 属性且值为 true

答案：a

页面有isErrorPage属性且值为false，不可以使用 exception 对象

7.面向对象5大基本原则

单一职责原则（SRP）
开放封闭原则（OCP）
里氏替换原则（LSP）
依赖倒置原则（DIP）
接口隔离原则（ISP）

单一职责原则（Single-Resposibility Principle）：一个类，最好只做一件事，只有一个引起它的变化。单一职责原则可以看做是低耦合、高内聚在面向对象原则上的引申，将职责定义为引起变化的原因，以提高内聚性来减少引起变化的原因。
开放封闭原则（Open-Closed principle）：软件实体应该是可扩展的，而不可修改的。也就是，对扩展开放，对修改封闭的。
Liskov替换原则（Liskov-Substituion Principle）：子类必须能够替换其基类。这一思想体现为对继承机制的约束规范，只有子类能够替换基类时，才能保证系统在运行期内识别子类，这是保证继承复用的基础。
依赖倒置原则（Dependecy-Inversion Principle）：依赖于抽象。具体而言就是高层模块不依赖于底层模块，二者都同依赖于抽象；抽象不依赖于具体，具体依赖于抽象。
接口隔离原则（Interface-Segregation Principle）：使用多个小的专门的接口，而不要使用一个大的总接口

8.下面哪项技术可以用在WEB开发中实现会话跟踪实现？

a.session
b.Cookie
c.地址重写
d.隐藏域

答案：a.b.c.d

HTTP是“无状态”协议：客户程序每次读取 Web 页面，都打开到 Web 服务器的单独的连接，并且，服务器也不自动维护客户的上下文信息。即使那些支持持续性 HTTP 连接的服务器，尽管多个客户请求连续发生且间隔很短时它们会保持 socket 打开，但是，它们也没有提供维护上下文信息的内建支持。上下文的缺失引起许多困难。例如，在线商店的客户向他们的购物车中加入商品时，服务器如何知道购物车中已有何种物品呢？类似地，在客户决定结账时，服务器如何能确定之前创建的购物车中哪个属于此客户呢？这些问题虽然看起来十分简单，但是由于 HTTP 的不足，解答它们却异常复杂困难。对于这个问题，存在 3 种典型的解决方案：
Cookie（结合session使用）
可以使用 cookie 存储购物会话的 ID；在后续连接中，取出当前的会话 ID，并使用这个 ID 从服务器上的查找表（lookup table）中提取出会话的相关信息。 以这种方式使用 cookie 是一种绝佳的解决方案，也是在处理会话时最常使用的方式。但是，sevlet 中最好有一种高级的 API 来处理所有这些任务，以及下面这些冗长乏味的任务：从众多的其他cookie中（毕竟可能会存在许多cookie）提取出存储会话标识符的 cookie；确定空闲会话什么时候过期，并回收它们；将散列表与每个请求关联起来；生成惟一的会话标识符。
URL 重写
采用这种方式时，客户程序在每个URL的尾部添加一些额外数据。这些数据标识当前的会话，服务器将这个标识符与它存储的用户相关数据关联起来。 URL重写是比较不错的会话跟踪解决方案，即使浏览器不支持 cookie 或在用户禁用 cookie 的情况下，这种方案

也能够工作。URL 重写具有 cookie 所具有的同样缺点，也就是说，服务器端程序要做许多简单但是冗长乏味的处理任务。即使有高层的 API 可以处理大部分的细节，仍须十分小心每个引用你的站点的 URL ，以及那些返回给用户的 URL。即使通过间接手段，比如服务器重定向中的 Location 字段，都要添加额外的信息。这种限制意味着，在你的站点上不能有任何静态 HTML 页面（至少静态页面中不能有任何链接到站点动态页面的链接）。因此，每个页面都必须使用 servlet 或 JSP 动态生成。即使所有的页面都动态生成，如果用户离开了会话并通过书签或链接再次回来，会话的信息也会丢失，因为存储下来的链接含有错误的标识信息。
隐藏的表单域
HTML 表单中可以含有如下的条目：<input type="hidden" name="session" value="a1234">
这个条目的意思是：在提交表单时，要将指定的名称和值自动包括在 GET 或 POST 数据中。这个隐藏域可以用来存储有关会话的信息，但它的主要缺点是：仅当每个页面都是由表单提交而动态生成时，才能使用这种方法。单击常规的超文本链接并不产生表单提交，因此隐藏的表单域不能支持通常的会话跟踪，只能用于一系列特定的操作中，比如在线商店的结账过程。

9.String和StringBuffer的区别？

a.String是不可变的对象，StringBuffer是可以再编辑的
b.String是常量，StringBuffer是变量

c.String是可变的对象，StringBuffer是不可以再编辑的

d.以上说法都不正确

答案：a.b

10.已知如下类定义：

```
class Base {
public Base (){
//...
}
public Base ( int m ){
//...
}
public void fun( int n ){
```

```
//...
}
}
public class Child extends Base{
// member methods
}
```

如下哪句可以正确地加入子类中？

a.private void fun(int n){ //...}
b.void fun (int n){ //... }
c.protected void fun (int n) { //... }
d.public void fun (int n) { //... }

答案：d

子类继承至少要权限满足父类即可 父类是public 子类只能是public了

11.下面描述属于java虚拟机功能的是？

a.通过 ClassLoader 寻找和装载 class 文件
b.解释字节码成为指令并执行，提供 class 文件的运行环境
c.进行运行期间垃圾回收
d.提供与硬件交互的平台

答案：a.b.c

12.判断对错。在java的多态调用中，new的是哪一个类就是调用的哪个类的方法。

a.对
b.错

答案：b

java多态有两种情况：重载和覆写
在覆写中，运用的是动态单分配，是根据new的类型确定对象，从而确定调用的方法；
在重载中，运用的是静态多分派，即根据静态类型确定对象，因此不是根据new的类型确定调用的方法

12.下面有关java classloader说法错误的是?

a.Java默认提供的三个ClassLoader是BootStrap ClassLoader，Extension ClassLoader，App ClassLoader
b.ClassLoader使用的是双亲委托模型来搜索类的
c.JVM在判定两个class是否相同时，只用判断类名相同即可，和类加载器无关
d.ClassLoader就是用来动态加载class文件到内存当中用的

答案：c

由于类加载器的双亲委托机制，不同类加载器加载的类必定不是同一个类

13.下面有关java的一些细节问题，描述错误的是？

a.构造方法不需要同步化
b.一个子类不可以覆盖掉父类的同步方法
c.定义在接口中的方法默认是public的
d,容器保存的是对象的引用

答案：b

构造方法每次都是构造出新的对象，不存在多个线程同时读写同一对象中的属性的问题，所以不需要同步

子类对父类的方法都可以覆盖

By mnmlist,2015-9-1 11:22:43

14.What will be printed when you execute the following code?

```
class C {
   C() {
       System.out.print("C");
   }
}

class A {
   C c = new C();

   A() {
       this("A");
       System.out.print("A");
   }

   A(String s) {
       System.out.print(s);
   }
}

class Test extends A {
   Test() {
       super("B");
       System.out.print("B");
   }

   public static void main(String[] args) {
       new Test();
   }
}
```

a.BB
b.CBB
c.BAB
d.None of the above

答案：b，super("B") 直接调用父类的有参构造方法，而会忽略父类的无参构造方法

15.下面有关文件系统元数据的描述，说法错误的是？

a.元数据指用来描述一个文件的特征的系统数据，诸如访问权限、文件拥有者以及文件数据块的分布信息等等
b.我们可以使用stat命令来查看文件更多的元数据信息

c.Unix/Linux系统允许，多个文件名指向同一个inode号码
d.文件A和文件B的inode号码虽然不一样，但是文件A的内容是文件B的路径。读取文件A时，系统会自动将访问者导向文件B，这是文件A就称为文件B的"硬链接"

答案：c只有软链接才会指向同一个innode号码

16.Java中用正则表达式截取字符串中第一个出现的英文左括号之前的字符串。比如：北京市（海淀区）（朝阳区）（西城区），截取结果为：北京市。正则表达式为（ ）

a.".*?(?=\\()"
b.".*?(?=\()"
c.".*(?=\\()"
d.".*(?=\()"

答案：a

没什么道理的

16.以下哪些类是线程安全的（ ）

a.Vector

b.ArrayList
c.StringBuffer
d.Properties

答案：a,c,d

Vector相当于一个线程安全的List
Arraylist是非线程安全的，其对应的线程安全类是Vector
StringBuffer是线程安全的，相当于一个线程安全的StringBuilder
Properties实现了Map接口，是线程安全的

By mnmlist,2015-9-1 11:22:43

17. 问这个程序的输出结果。

```
package Wangyi;
class Base
{
   public void method()
   {
       System.out.println("Base");
   }
}
class Son extends Base
{
   public void method()
   {
       System.out.println("Son");
   }

   public void methodB()
   {
       System.out.println("SonB");
   }
}
public class Test01
{
   public static void main(String[] args)
   {
       Base base = new Son();
       base.method();
       base.methodB();
   }
}
```

a.Base SonB
b.Son SonB
c.Base Son SonB
d.编译不通过

答案：编译不通过

有关对象的上转型对象 Base base = newSon();--》
** base.method();//等同于son调用重写的method()方法
** base.methodB(); //非法，因为methodB()（是子类新加的方法 子类新加的属性也是不可访问的）--》 报错-》D
**插入Son son=new (Son)base //把上转型对象强制转化为子类对象即刻通过编译--》

By mnmlist,2015-9-1 11:22:43

B

18.电影院排队问题

有2n个人排队进电影院，票价是50美分。在这2n个人当中，其中n个人只有50美分，另外n个人有1美元（纸票子）。愚蠢的电影院开始卖票时1分钱也没有。
问： 有多少种排队方法 使得 每当一个拥有1美元买票时，电影院都有50美分找钱
注：
1美元=100美分
拥有1美元的人，拥有的是纸币，没法破成2个50美分

a.(2n)!/[n!n!]
b.(2n)!/[n!(n+1)!]
c.(2n)!/[n!(n-1)!]
d.(2n + 1)!/[n!(n-1)!]

正确答案：b

19:下面关于Spring的说法中错误的是（ ）

a.Spring是一系列轻量级Java EE框架的集合
b.Spring中包含一个“依赖注入”模式的实现
c.使用Spring可以实现声明式事务
d.Spring提供了AOP方式的日志系统

答案：d

Spring提供了AOP方式的日志系统
Spring并没有为我们提供日志系统，我们需要使用AOP（面向方面编程）的方式，借助Spring与日志系统log4j实现我们自己的日志系统。

20.下面有关forward和redirect的描述，正确的是？

By mnmlist,2015-9-1 11:22:43

a.执行forward时，浏览器不知道服务器发送的内容是从何处来，浏览器地址栏中还是原来的地址
b.执行redirect时，服务器端告诉浏览器重新去请求地址
c.forward是内部重定向，redirect是外部重定向

d.redirect默认将产生301 Permanently moved的HTTP响应

答案：a,b,c

request的forward和response的redirect
1.redirect地址栏变化，forward发生在服务器端内部从而导致浏览器不知道响应资源来自哪里
2.redirect可以重定向到同一个站点上的其他应用程序中的资源，forward 只能将请求 转发给同一个WEB应用中的组件
3.redirect默认是302码，包含两次请求和两次响应
4.redirect效率较低

Java中Comparator接口和Comparable接口的使用

一般情况下在实现对对象元素的数组或集合进行排序的时候会用到Comparator和Comparable接口，通过在元素所在的类中实现这两个接口中的一个，然后对数组或集合调用Arrays.sort或者Collentions.sort方法即可实现对数组或集合的排序。就sort方法里面的参数来说，实现了不同的接口则传递的参数也不尽相同。对于实现了Comparator接口的类来说，sort方法需要接受的参数不仅包括数组或集合，还要包括实现了该接口的类对象；而对实现了Comparable接口的类来说，参数不仅包括数组或者集合，还要包括实现了该接口的类对象。具体怎么区别呢，其实很简单，因为看两个接口所实现的方法就知道，Comparator定义的方法为compare(Object o1, Object o2)，方法涉及两个类对象，所以需要在另外一个新的类来实现对数组元素的比较，所以在调用sort方法时需要传递这个额外的实现了Comparator接口的类对象；而实现了Comparable接口的类实在元素类的内部实现了排序的逻辑，所以调用sort方法时不需要传递额外的类对象。废话少说，直接上代码。
1.通过实现Comparator接口来实现对数组或集合的比较

```
class SortCat implements Comparator<Cat1>
{
   @Override
```

By mnmlist,2015-9-1 11:22:43

```
   public int compare(Cat1 o1, Cat1 o2)//Comparatorcompare
   {
       // TODO Auto-generated method stub
       int size1=o1.getSize(),size2=o2.getSize();
       if(size1!=size2)
           return size1-size2;
       return o1.getColor().compareTo(o2.getColor());
   }
}
class Cat1
{
   private String color;
   private int size;
   public  Cat1(String color,int size)
   {
       this.color=color;
       this.size=size;
   }
   public int getSize()
   {
       return size;
   }
   public String getColor()
   {
       return color;
   }
   public String toString()
   {
       return color+" cat,size = "+size;
   }

}
public class HashMapComparatorDemo
{

   public static void main(String[] args)
   {
       // TODO Auto-generated method stub
       String colorArr[]={"black","yellow","white","colorful","gray","brown","blue","orar
       int catSizeArr[]={5,3,7,9,6,4,1,8,2};
       Random random=new Random();
       Cat1 catArr[]=new Cat1[10];
       int sizeIndex=0,colorIndex=0;
       for(int i=0;i<10;i++)
       {
           sizeIndex=random.nextInt(catSizeArr.length);
           colorIndex=random.nextInt(colorArr.length);
           catArr[i]=new Cat1(colorArr[colorIndex], catSizeArr[sizeIndex]);
           System.out.println("Color:"+catArr[i].getColor()+",size:"+catArr[i].getSize())
       }
       System.out.println("\nAfter change the order....\n");
       Arrays.sort(catArr,new SortCat());//Comparator
       for(Cat1 cat:catArr)
           System.out.println("Color:"+cat.getColor()+",size:"+cat.getSize());

   }
```

```
By mnmlist,2015-9-1 11:22:43
}
```

结果：

```
Color:white,size:4
Color:colorful,size:4
Color:colorful,size:4
Color:gray,size:5
Color:yellow,size:7
Color:orange,size:6
Color:black,size:2
Color:colorful,size:8
Color:black,size:3
Color:yellow,size:2

After change the order....

Color:black,size:2
Color:yellow,size:2
Color:black,size:3
Color:colorful,size:4
Color:colorful,size:4
Color:white,size:4
Color:gray,size:5
Color:orange,size:6
Color:yellow,size:7
Color:colorful,size:8
```

2.通过实现Comparable接口来实现对数组或集合的比较

```
class Cat2  implements Comparable<Cat2>
{
   private String color;
   private int size;
   public  Cat2(String color,int size)
   {
       this.color=color;
       this.size=size;
   }
   public int getSize()
   {
       return size;
   }
   public String getColor()
   {
       return color;
   }
   public String toString()
   {
       return color+" cat,size = "+size;
   }
```

```
   @Override
   public int compareTo(Cat2 o)//comparable
   {
       // TODO Auto-generated method stub
       int size=o.getSize();
       if(this.size!=size)
           return this.size-size;
       return this.color.compareTo(o.getColor());
   }

}
public class HashMapComparatorDemo2
{

   public static void main(String[] args)
   {
       String colorArr[]={"black","yellow","white","colorful","gray","brown","blue","oran
       int catSizeArr[]={5,3,7,9,6,4,1,8,2};
       Random random=new Random();
       Cat2 catArr[]=new Cat2[10];
       int sizeIndex=0,colorIndex=0;
       for(int i=0;i<10;i++)
       {
           sizeIndex=random.nextInt(catSizeArr.length);
           colorIndex=random.nextInt(colorArr.length);
           catArr[i]=new Cat2(colorArr[colorIndex], catSizeArr[sizeIndex]);
           System.out.println("Color:"+catArr[i].getColor()+",size:"+catArr[i].getSize())
       }
       System.out.println("\nAfter change the order....\n");
       Arrays.sort(catArr);//
       for(Cat2 cat:catArr)
           System.out.println("Color:"+cat.getColor()+",size:"+cat.getSize());

   }

}
```

结果：

```
Color:gray,size:7
Color:orange,size:9
Color:gray,size:3
Color:brown,size:5
Color:orange,size:1
Color:gray,size:8
Color:colorful,size:9
Color:white,size:1
Color:blue,size:7
Color:brown,size:1

After change the order....

Color:brown,size:1
Color:orange,size:1
```

By mnmlist,2015-9-1 11:22:43

```
Color:white,size:1
Color:gray,size:3
Color:brown,size:5
Color:blue,size:7
Color:gray,size:7
Color:gray,size:8
Color:colorful,size:9
Color:orange,size:9
```

3.另外，还可以用实现了对Comparable接口treeMap、treeSet进行排序，具体应用范围很广，包括求一个无重复无序数组的前k项元素就可以用treeSet或treeMap很好滴实现，循环建tree并维持一个大小为k的treeSet或treeMap即可，超出范围的话先插入一个元素，然后删除排在最后的元素即可。下面这段代码仅仅实现的是有序建树并将treeMap输出出来的功能。

```
class Cat implements Comparable<Cat>
{
   private String color;
   private int size;
   public  Cat(String color,int size)
   {
       this.color=color;
       this.size=size;
   }
   public int getSize()
   {
       return size;
   }
   public String getColor()
   {
       return color;
   }
   @Override
   public int compareTo(Cat o)
   {
//      //
//      String color1=o.getColor();
//      if(this.color.compareTo(color1)!=0)
//          return this.color.compareTo(color1);
//      return this.size-o.size;
       //
       int size=o.getSize();
       if(this.size!=size)
           return this.size-size;
       return this.color.compareTo(o.getColor());
   }
   public String toString()
   {
       return "Color:"+color+",size = "+size;
   }
}
public class HashMapComparableDemo
{
```

By mnmlist,2015-9-1 11:22:43

```
   public static void main(String[] args)
   {
       SortedMap<Cat, Integer>map=new TreeMap();
       String colorArr[]={"black","yellow","white","colorful","gray","brown","blue","orar
       int catSizeArr[]={5,3,7,9,6,4,1,8,2};
       Random random=new Random();
       int sizeIndex=0,colorIndex=0,count=0;;
       int mapSize=10;
       for(int i=0;i<mapSize;i++)
       {
           sizeIndex=random.nextInt(catSizeArr.length);
           colorIndex=random.nextInt(colorArr.length);
           count=random.nextInt(20)+5;
           map.put(new Cat(colorArr[colorIndex], catSizeArr[sizeIndex]), count);
       }
       Iterator<Entry<Cat, Integer>>iterator=map.entrySet().iterator();
       Entry<Cat, Integer>entry;
       while(iterator.hasNext())
       {
           entry=iterator.next();
           System.out.println(entry.getKey().toString()+",Count:"+entry.getValue().toStri
       }

   }

}
```

结果：

```
Color:black,size = 1,Count:15
Color:black,size = 2,Count:12
Color:gray,size = 2,Count:24
Color:brown,size = 5,Count:24
Color:gray,size = 5,Count:14
Color:brown,size = 6,Count:13
Color:white,size = 6,Count:7
Color:yellow,size = 6,Count:12
Color:gray,size = 7,Count:9
Color:brown,size = 8,Count:17
```

另外：对某些对象建立treeMap或hashMap的时候还需要重写一下equals方法，防止发生插入相同元素的情况，因为默认情况下treeMap或HashMap是应该是以对象的引用作为各个对象元素的标志，而不是元素的值。

对象的比较及hashCode、equals方法的使用

By mnmlist,2015-9-1 11:22:43

如何进行对象的值的比较呢？如String类型的变量，是靠调用equals方法来比较的，而其它的类似数组或普通的对象直接调用equals方法一般是不可以的，这是因为String类型的变量通过调用equals方法来比较变量是因为String类覆盖了Object的HashCode方法和equals方法。正如String调用equals方法可以比较String对象变量的值是否相等，一般的对象也可以通过覆盖Object类的方法来达到同样的目的的。如下面代码 所示：

```
import java.util.HashMap;
import java.util.Map;
/*
* @author mnmlist
* @date 8/18/2015
* @description the basic usage of hashCode and equals method
*/
class Stu
{
String sno;
String sname;
public Stu(String sno,String sname)
{
 this.sno=sno;
 this.sname=sname;
}
public String getSno()
{
 return sno;
}
public boolean equals(Object obj)
{
 if(!(obj instanceof Stu))
  return false;
 if(obj==this)
  return true;
 return this.sno.equals(((Stu)obj).getSno());
}
public int hashCode()
{
 return (sno+sname).hashCode();
}
}
public class HashcodeDemo
{

public static void main(String[] args)
{
 // TODO Auto-generated method stub
 Map<Stu, Integer>map=new HashMap<>();
 Stu st1=new Stu("123", "Sting");
 Stu st2=new Stu("234", "mnmlist");
 Stu st3=new Stu("345", "Tony");
 map.put(st1, 1);
```

By mnmlist,2015-9-1 11:22:43

```
 map.put(st2,2);
 map.put(st3, 3);
 Stu stu=new Stu("123", "Sting");
 System.out.println(map.containsKey(stu));
}

}
```

如上述代码所示，通过覆盖hashCode方法和equals方法可以实现对对象的比较结果：

```
true
```


Java中枚举类型的使用

虽然Java中有枚举类型这种数据类型，但是很少用，其实当用到星期、月份、四季等数据集的时候，无疑，枚举类型不仅很方便地解决了整数和字符串的映射问题，而且极大地提高了程序的可读性。

下面就用简单的程序代码来说明这种问题：

1.枚举类型自动赋值，默认情况下枚举变量从前到后分别被赋值为0、1、2、3、4、5....然后用循环的方式将枚举变量的名称和其对应的下标打印出来。

```
/*
 * @author mnmlist
 * @date 8/18/2015
 * @description of the basic usage of enum
 */
enum Color1
{
 RED,GREEN,BLUE,PINK;
 private Color1(){};
 public static void printAllValues()
```

```
 {
  for(Color1 color:Color1.values())
  {
   System.out.println("Name:"+color+",Index:"+color.ordinal());//
  }
 }
 public static void printOneValue()
 {
  System.out.println("Name:"+RED+",Index:"+RED.ordinal());
 }
}
public class EnumDemo
{

public static void main(String[] args)
{
//  for(Color color:Color.values())
//   color.printValue();
 System.out.println("Print one value:");
 Color1.printOneValue();
 System.out.println("Print all values:");
 Color1.printAllValues();
}

}
```

结果：

```
Print one value:
Name:RED,Index:0
Print all values:
Name:RED,Index:0
Name:GREEN,Index:1
Name:BLUE,Index:2
Name:PINK,Index:3
```

2.利用构造函数对枚举变量进行赋值，这样就可以从随便某个整型的值开始而不是仅从0开始

```
enum Color
{
RED(3),YELLOW(5),BLUE(9);
private int value;
private Color(){}
private  Color(int value)
{
 this.value=value;
```

```
}
public void printValue()
{

 System.out.println("Name:"+this.name()+",Value:"+this.value);//
}

}




public class EnumDemo
{

public static void main(String[] args)
{
 for(Color color:Color.values())
  color.printValue();
}

}
```

结果：

```
Name:RED,Value:3
Name:YELLOW,Value:5
Name:BLUE,Value:9
```

By mnmlist,2015-9-1 11:22:43

腾讯面试题目之一

描述：

假如给定一个150个整数大小的整数数组nums[]，整数的范围1~100，请问用最快的速度如何对该数组进行排序？

思路：

基数排序？假设我不用基数排序呢。。。

那我用快排、归并排序和堆排序，时间复杂度都是O(nlogn)，哪个方法更快些呢？归并排序。有什么依据么？好吧，是我猜的，囧~

回去跟同学探讨了一下，其实就是考查用空间换时间罢了。一个很经典的例子，相似的例子比如对一个堆身高数据进行排序，身高肯定就非常集中了50cm-240cm左右吧，不外乎开辟一个长度为241的整数数组count[]，遍历数组，并将count[nums[i]]++;即可，最后输出的时候按count里元素的个数输出相应的身高即可。

代码：

```
public void sortNumers(int nums[])
{
 //assume the scope of  the numbers is between 1 and 100
 if(nums==null||nums.length==0)
  return ;
 int count[]=new int[101];
 for(int i=0;i<nums.length;i++)
  count[nums[i]]++;
 for(int i=1;i<count.length;i++)
 {
  for(int j=0;j<count[i];j++)
   System.out.print(i+" ");
 }
 System.out.println();
}
```

leetcode_Palindrome Linked List

描述：

Given a singly linked list, determine if it is a palindrome.

Follow up:
Could you do it in O(n) time and O(1) space?

思路：

1.如何判断一个链表是否是回文的？很简单将链表中的元素遍历出来并放进ArrayList中，然后可以像数组一样来判断该元素是否为回文的，时间复杂度O(n)，空间复杂度O(n)，可如何用O(n)的时间复杂度和O(1)的空间复杂度来解决呢？

2.是不是可以考虑 将链表反转？可反转后还是链表啊，要是将链表分为前后两个部分呢，分为两个部分还是无法判断该链表是否为回文链表啊，那要是再将其中一个链表反转一下呢，It's done!好多时候，多想一步容易，再多想一步就困难了。

代码：

```
public boolean isPalindrome(ListNode head) {
        if(head==null||head.next==null)
            return true;
        ListNode p=head,temp=head,quick=head;
        while(temp!=null&&quick!=null)
        {
            temp=temp.next;
            if(quick.next==null)
                break;
            quick=quick.next.next;
        }
        temp=reverseList(temp);
        p=head;
        while(temp!=null&&p!=null)
        {
            if(temp.val!=p.val)
                return false;
            temp=temp.next;
            p=p.next;
```

```
        }
        return true;
    }
    public ListNode reverseList(ListNode head)
    {
        ListNode tempHead=new ListNode(-1);
        ListNode p=head.next,q=null;
        tempHead.next=head;
        head.next=null;
        while(p!=null)
        {
            q=p;
            p=p.next;
            q.next=tempHead.next;
            tempHead.next=q;
        }
        return tempHead.next;
    }
```


leetcode_Jump Game II

描述：

Given an array of non-negative integers, you are initially positioned at the first index of the array.
Each element in the array represents your maximum jump length at that position.
Your goal is to reach the last index in the minimum number of jumps.
For example:
Given array A = [2,3,1,1,4]
The minimum number of jumps to reach the last index is 2. (Jump 1 step from index 0 to 1, then 3 steps to the last index.)

思路：

1.Jump Game思路：Max Subarrayreach=Math.max(nums[i]+1,reach),i>reachi>=nums.lengthtrue,false.

2.Jump GameJump Game II edgereachi<=reachMath.max(nums[i]+i,reach)reachi>edgeedgereachminStep minStep+1

By mnmlist,2015-9-1 11:22:43

代码：

```
public int jump(int[] nums)
{
 int edge=0,reach=0;
 int minStep=0,i=0;
 for(;i<nums.length&&i<=reach;i++)
 {
  if(edge<i)
  {
   edge=reach;
   minStep=minStep+1;
  }
  reach=Math.max(nums[i]+i, reach);
 }
 if(i==nums.length)
  return minStep;
 return -1;
}
```


leetcode_Jump Game

描述：

Given an array of non-negative integers, you are initially positioned at the first index of the array.
Each element in the array represents your maximum jump length at that position.
Determine if you are able to reach the last index.
For example:
A = [2,3,1,1,4], return true.
A = [3,2,1,0,4], return false.

思路：

和求Max Subarray类似，维护一个当前元素可以跳至的最大值，每循环一次更新reach=Math.max(nums[i]+1,reach),当i>reach或i>=nums.length的时候循环终止，最后看循环是否到达了最后，到达最后则返回true,否则，返回false.

代码：

By mnmlist,2015-9-1 11:22:43

```
public boolean canJump(int[] nums) {
       int i=0;
       for(int reach=0;i<nums.length&&i<=reach;i++)
       {
        reach=Math.max(nums[i]+i,reach);
       }
       return i==nums.length;
   }
```


leetcode_Decode Ways

描述：

A message containing letters from A-Z is being encoded to numbers using the following mapping:
'A' -> 1
'B' -> 2
...
'Z' -> 26
Given an encoded message containing digits, determine the total number of ways to decode it.
For example,
Given encoded message "12", it could be decoded as "AB" (1 2) or "L" (12).
The number of ways decoding "12" is 2.

思路：

直接从后面开始算，如果str.substring(i,i+2)代表的数值小于26，那么，memo[i]=memo[i+1]+memo[i+2];否则，memo[i]=memo[i+1]，虽然看起来很简单，却是很难想，一般都是从前面开始，然后比较字符的值再做判断，这样的话控制逻辑会非常复杂。

By mnmlist,2015-9-1 11:22:43

代码：

```
public int numDecodings(String s)
{
 int strLen=s.length();
 if(strLen==0)
  return 0;
 int memo[]=new int[strLen+1];
 memo[strLen]=1;
 memo[strLen-1]=s.charAt(strLen-1)!='0'?1:0;
 for(int i=strLen-2;i>=0;i--)
 {
  if(s.charAt(i)=='0')
   continue;
  else {
   memo[i]=(Integer.parseInt(s.substring(i,i+2)))<=26?memo[i+1]+memo[i+2]:memo[i+1];
  }
 }
 return memo[0];
}
```


leetcode_Maximum Product Subarray

描述：

Find the contiguous subarray within an array (containing at least one number) which has the largest product.
For example, given the array [2,3,-2,4],
the contiguous subarray [2,3] has the largest product = 6.

思路：

1.一种思路是将数组分为被数字0分割的子数组，然后统计负数出现的次数，如果 是偶数，直接取子数组的乘积，如果是奇数，取去掉前面第一个奇数出现之前的 部分或去掉最后一个奇数及其后面数组部分后剩余部分的乘积即可，然后取maxProduct为字数组乘积最大的一段。程序的控制流程很麻烦。

2.每次取部分数组乘积的最大值或最小值，然后跟num[i]做乘法运算，再取最大值或最小值即可，取最大值理所当然，取最小值是因为最小值的绝对值 可能大于最大值，碰到负

By mnmlist,2015-9-1 11:22:43

数就变最大值了。

代码：

```
public int maxProduct(int[] nums)
{
 if (nums == null || nums.length == 0)
  return 0;
 int maxProduct=nums[0];
 int minProduct=nums[0];
 int maxSofar=nums[0];
 int tempMaxProduct,tempMinProduct;
 int numMax,numMin;
 for(int i=1;i<nums.length;i++)
 {
  numMax=maxProduct*nums[i];
  numMin=minProduct*nums[i];
  tempMaxProduct=Math.max(Math.max(numMax,numMin ), nums[i]);
  tempMinProduct=Math.min(Math.min(numMax, numMin), nums[i]);
  maxSofar=Math.max(tempMaxProduct, maxSofar);
  maxProduct=tempMaxProduct;
  minProduct=tempMinProduct;
 }
 return maxSofar;
}
```


leetcode_Sort List

描述：

Sort a linked list inO(nlogn) time using constant space complexity.

思路：

By mnmlist,2015-9-1 11:22:43

1.链表的排序一般每遇到过，让用O(nlogn)解决该问题就更不知如何下手了

2.通过参考网上的思路才知道用归并排序，采用递归的方法解决该问题还是可以的，就是理解起来有点费劲

3.重要步骤：递归，归并，查找数组有效范围内的中间节点，有序数组合并

代码：

```
public ListNode sortList(ListNode head)
{
 if(head==null||head.next==null)
  return head;
 ListNode midNode=getMidNode(head);
 ListNode newHead=midNode.next;
 midNode.next=null;
 head=mergeLists(sortList(newHead), sortList(head));
 return head;
}

public ListNode mergeLists(ListNode head1, ListNode head2)
{
 if(head1==null&&head2==null)
  return head1;
 if(head1==null)
  return head2;
 if(head2==null)
  return head1;
 ListNode newHead=new ListNode(-1);
 ListNode curListNode=newHead;
 while(head1!=null&&head2!=null)
 {
  if(head1.val<head2.val)
  {
   curListNode.next=head1;
   head1=head1.next;
  }else
  {
   curListNode.next=head2;
   head2=head2.next;
  }
  curListNode=curListNode.next;
 }
 if(head1!=null)curListNode.next=head1;
 if(head2!=null)curListNode.next=head2;
 return newHead.next;
}

public ListNode getMidNode(ListNode head)
```

By mnmlist,2015-9-1 11:22:43

```
{
 if(head==null||head.next==null)
  return head;
 ListNode p=head,q=head;
 while(q.next!=null&&q.next.next!=null)
 {
  p=p.next;
  q=q.next.next;

 }
 return p;
}
```


leetcode_Integer to Roman

描述：

Given an integer, convert it to a roman numeral.

Input is guaranteed to be within the range from 1 to 3999.

思路：

1.由于罗马数字是字符的形式堆叠而成的，所以不妨将数字的每一位转换成字符并结合相应的量级合成罗马字符串

2.相应的，对于每个字符数字在不考虑量级的情况下大概分三种情况可以解决阿拉伯数字到罗马数字的转换，分(number>=1&&number<=3)、(number>=4&&number<=8)、(number==9)来考虑，具体的实现见代码。

代码：

```
public String intToRoman(int num)
{
 StringBuilder sBuilder=new StringBuilder();
 char chArr[] ={ 'I', 'V', 'X', 'L', 'C', 'D', 'M' };
 int numArr[] ={ 1, 5, 10, 50, 100, 500, 1000 };
 HashMap<Integer,Character> map = new HashMap< Integer,Character>();
 HashMap<Character, Integer> priorityMap = new HashMap<Character, Integer>();
```

By mnmlist,2015-9-1 11:22:43

```
 for (int i = 0; i < chArr.length; i++)
 {
  map.put(numArr[i], chArr[i]);
  priorityMap.put(chArr[i], i);
 }
 String numString=String.valueOf(num);
 int tempNum=1;
 int number=0;
 int numLen=numString.length();
 for(int i=1;i<numLen;i++)
  tempNum*=10;
 char ch='0';
 for(int i=0;i<numLen;i++)
 {
  number=numString.charAt(i)-'0';
  if(number>=1&&number<=3)
  {
   ch=map.get(tempNum);
   for(int j=0;j<number;j++)
    sBuilder.append(ch);
  }
  else if(number>=4&&number<=8)
  {
   ch=map.get(tempNum);
   int pri=priorityMap.get(ch);
   char newChar=chArr[pri+1];
   for(int j=0;j<5-number;j++)
    sBuilder.append(ch);
   sBuilder.append(newChar);
   for(int j=6;j<=number;j++)
    sBuilder.append(ch);

  }else if(number==9)
  {
   ch=map.get(tempNum);
   int pri=priorityMap.get(ch);
   char newChar=chArr[pri+2];
   sBuilder.append(ch);
   sBuilder.append(newChar);
  }
  tempNum/=10;
 }

 return sBuilder.toString();
}
```


leetcode_Roman to Integer

By mnmlist,2015-9-1 11:22:43

描述：

Given a roman numeral, convert it to an integer.

Input is guaranteed to be within the range from 1 to 3999.

思路：

1.就像脑筋急转弯一样，这一题还真的搞了好久，主要是没想明白思路，现在再回过头去看代码，显而易见，或许这才是制造和创造的区别吧

2.大致过程就是一个循环，当currentChar的量级小于nextChar的量级时，直接加上nextChar的量级和currentChar的量级之差即可，其它的情况直接加上currentChar的量级，罗马数字就类似字符串的形式

3.另：这也就是有计算机才可以这么算，这对普通人来说得多麻烦啊，罗马数字很啰嗦。

代码：

```
public int romanToInt(String s) {
      char chArr[]={'I','V','X','L','C','D','M','i','v','x','l','c','d','m'};
       int numArr[]={1,5,10,50,100,500,1000,1,5,10,50,100,500,1000};
       HashMap<Character, Integer>map=new HashMap<Character, Integer>();
       for(int i=0;i<chArr.length;i++ )
        map.put(chArr[i], numArr[i]);
       int j=0;
       int strLen=s.length();
       char curCh,nextCh;
       int sum=0;
       for(int i=0;i<strLen;i++)
       {
        curCh=s.charAt(i);
        j=i+1;
        if(j<strLen)
        {
         nextCh=s.charAt(j);
         if(map.get(curCh)<map.get(nextCh))//
         {
          sum+=map.get(nextCh)-map.get(curCh);
          i=j;//
         }else {
          sum+=map.get(curCh);
   }

        }else {
```

By mnmlist,2015-9-1 11:22:43

```
   sum+=map.get(curCh);//
  }

      }
      return sum;
   }
```


leetcode_Repeated DNA Sequences

描述：

All DNA is composed of a series of nucleotides abbreviated as A, C, G, and T, for example: "ACGAATTCCG". When studying DNA, it is sometimes useful to identify repeated sequences within the DNA.

Write a function to find all the 10-letter-long sequences (substrings) that occur more than once in a DNA molecule.

For example,

```
Given s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT",

Return:
["AAAAACCCCC", "CCCCCAAAAA"].
```

思路：

1.很显然，暴力求解也是一种方法，尽管该方法是不可能的。

2. ”A" "C" “G" "T" ASCII65, 67, 71, 84 1000001, 1000011, 1000111, 10101003bit1030bitint29~00~9 30bitintkeyHashTable

代码：

```
public List<String> findRepeatedDnaSequences(String s)
{
 List<String>list=new ArrayList<String>();
 int strLen=s.length();
 if(strLen<=10)
```

By mnmlist,2015-9-1 11:22:43

```
  return list;
 HashMap<Integer, Integer>map=new HashMap<Integer,Integer>();
 int key=0;
 for(int i=0;i<strLen;i++)
 {
  key=((key<<3)|(s.charAt(i)&0x7))&0x3fffffff;//k<<3,key3
  //s.charAt(i)&0x7)s.charAt(i)3
  //&0x3fffffffkey
  if(i<9)continue;
  if(map.get(key)==null)//map
   map.put(key, 1);
  else if(map.get(key)==1)//list
  {
   list.add(s.substring(i-9,i+1));
   map.put(key, 2);//
  }
 }
 return list;
}
```


leetcode_House Robber II

描述：

Note:This is an extension ofHouse Robber.

After robbing those houses on that street, the thief has found himself a new place for his thievery so that he will not get too much attention. This time, all houses at this place are arranged in a circle.That means the first house is the neighbor of the last one. Meanwhile, the security system for these houses remain the same as for those in the previous street.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonightwithout alerting the police.

思路：

1.和House Robber的思路类似，只是本题中房子是首尾相连的，也就是说第一个房子和最后一个房子不能都被盗，否则，触发警报，小偷被抓

2.在House Robber的基础上，a:如果从第一座房子开始偷窃，那么最后一个房子肯定不能被偷；b:如果从第二座房子偷窃，那么最后一个房子可以被偷，分别对这两种情况进行计算即可。

By mnmlist,2015-9-1 11:22:43

代码：

```
public int rob(int[] nums) {
       if(nums==null)
           return 0;
       int len=nums.length;
       int maxProfit=0;
       if(len==0)
           return 0;
       else if(len==1)
           return nums[0];
       else if(len==2)
           return Math.max(nums[0],nums[1]);
       else
       {
           int tempMaxProfit1=getMaxProfit(nums, 0, nums.length-2);
           int tempMaxProfit2=getMaxProfit(nums, 1, nums.length-1);
           maxProfit=Math.max(tempMaxProfit1, tempMaxProfit2);
       }
       return maxProfit;
   }
public int getMaxProfit(int nums[],int start,int end)
{
 int maxProfit1=nums[start];//maxProfit1maxProfit2
       int maxProfit2=Math.max(nums[start],nums[start+1]);
       int maxProfit=Math.max(maxProfit1, maxProfit2);
       for(int i=start+2;i<=end;i++)
       {
           maxProfit=Math.max(maxProfit2,maxProfit1+nums[i]);
           maxProfit1=maxProfit2;
           maxProfit2=maxProfit;
       }
       return maxProfit;
}
```


leetcode_Product of Array Except Self

描述：

By mnmlist,2015-9-1 11:22:43

Given an array ofnintegers wheren> 1,`nums`, return an array`output`such that`output[i]`is equal to the product of all the elements of`nums`except`nums[i]`.

Solve itwithout divisionand in O(n).

For example, given`[1,2,3,4]`, return`[24,12,8,6]`.

Follow up:
Could you solve it with constant space complexity? (Note: The output arraydoes notcount as extra space for the purpose of space complexity analysis.)

思路：

1.该题目的要求返回一个数组，该数组的product[i]=num[0]*num[1]*.....num[i-1]*num[i+1]*...*num[num.length-1];

2.由于要求出除i位置的元素num[i]的其它所有元素的乘积，假如每次都这么循环遍历一次并作乘法运算，当到num[i]的时候跳过，这样时间复杂度就是O(n*n)

3.另外一种思路就是求出所有的元素的乘积productTotal，然后具体求某个product[i]时，直接product[i]=productTotal/num[i]

4.但3中的思路有一个问题，就是productTotal有可能溢出，为处理这种情况，将productTotal初始化为long类型，OK！

代码：

```
public int[] productExceptSelf(int[] nums)
{
 if (nums == null || nums.length == 0)
  return nums;
 long result = 1;
 for (int num : nums)
  result *= num;
 if(result!=0)
 {
  for (int i = 0; i < nums.length; i++)
   nums[i] = (int) (result / nums[i]);
 }else
 {
  int newArr[]=new int[nums.length];
  newArr=Arrays.copyOf(nums, nums.length);
  for (int i = 0; i < nums.length; i++)
  {
   if(newArr[i]!=0)
```

By mnmlist,2015-9-1 11:22:43

```
    nums[i]=0;
   else
   {
    int tempProduct=1;
    for(int j=0;j<nums.length;j++)
    {
     if(j==i)
      continue;
     else
      tempProduct*=newArr[j];

    }
    nums[i]=tempProduct;
   }
  }
 }
 return nums;
}
```


leetcode_Combination Sum III

描述：

Find all possible combinations ofknumbers that add up to a numbern, given that only numbers from 1 to 9 can be used and each combination should be a unique set of numbers.

Ensure that numbers within the set are sorted in ascending order.

Example 1:

Input:k= 3,n= 7

Output:

```
[[1,2,4]]
```

Example 2:

Input:k= 3,n= 9

Output:

```
[[1,2,6], [1,3,5], [2,3,4]]
```

By mnmlist,2015-9-1 11:22:43

思路：

1.和Combination Sum II思路类似，只是在其基础上判断一下当有满足条件的结果时，其序列长度是否为k

代码：

```
public List<List<Integer>> combinationSum3(int k, int n) {
 int candidates[]={1,2,3,4,5,6,7,8,9};
 List<List<Integer>>listResult=new ArrayList<List<Integer>>();
 ArrayList<Integer>list=new ArrayList<Integer>();
 Arrays.sort(candidates);
 combination(listResult, list, candidates, k,n,0);
 return listResult;
}
public void combination(List<List<Integer>>listResult,
  ArrayList<Integer>list,int[] candidates,int k, int target,int begin)
{
 if(target==0)
 { if(list.size()==k)<span style="font-size: 13.3333339691162px; font-family: Arial, Helv
   listResult.add(list);
  return;
 }
 for(int i=begin;i<candidates.length&&target>=candidates[i];i++)
 {
  if(i==begin||candidates[i]!=candidates[i-1])//<span style="font-size: 13.3333339691162p
  {
   ArrayList<Integer>copy=new ArrayList<Integer>(list);
   copy.add(candidates[i]);
   combination(listResult, copy, candidates,k,target- candidates[i],i+1);
  }
 }
}
```


leetcode_Combination Sum II

By mnmlist,2015-9-1 11:22:43

描述：

Given a collection of candidate numbers (C) and a target number (T), find all unique combinations inCwhere the candidate numbers sums toT.

Each number inCmay only be usedoncein the combination.

Note:

- **All numbers (including target) will be positive integers.**
- **Elements in a combination (a1,a2, … ,ak) must be in non-descending order. (ie,a1a2 … ak).**
- **The solution set must not contain duplicate combinations.**

For example, given candidate set10,1,2,7,6,1,5and target8,
A solution set is:

```
[1, 7]
[1, 2, 5]
[2, 6]
[1, 1, 6]
```

思路：

1.基本思路和Combination Sum类似。

2.Combination Sum^-^!targettarget=target-candidates[i]indexitarget==0target<candidates[i]i++

3.Combination SumCombination Sum(i==begin||candidates[i]!=candidates[i-1])

代码：

```
public List<List<Integer>> combinationSum2(int[] candidates, int target)
{
 List<List<Integer>>listResult=new ArrayList<List<Integer>>();
 ArrayList<Integer>list=new ArrayList<Integer>();
 Arrays.sort(candidates);
 combination(listResult, list, candidates, target,0);
 return listResult;

}
public void combination(List<List<Integer>>listResult,
  ArrayList<Integer>list,int[] candidates, int target,int begin)
{
 if(target==0)
 {
```

By mnmlist,2015-9-1 11:22:43

```
  listResult.add(list);
  return;
 }
 for(int i=begin;i<candidates.length&&target>=candidates[i];i++)
 {
  if(i==begin||candidates[i]!=candidates[i-1])
  {
   ArrayList<Integer>copy=new ArrayList<Integer>(list);
   copy.add(candidates[i]);
   combination(listResult, copy, candidates,target- candidates[i],i+1);
  }
 }
}
```


[置顶] 100 High-Quality Java Developers' Blogs

This list collects 100 high quality blogs from Java developers from all over the world. Some of these blogs may not be written by Java developers, but at least Java developers should find it useful or interesting. Reading those blogs should be fun and often bring some fresh ideas.

Google ranks large websites higher. That is not so fair for small high-quality blogs. There are a lot of sites that have very large traffic, but they may not have high quality. My definition of high quality is as follows:

1. Articles are readable and have originality.
2. Its author shows real interest in technology.
3. It contains creative thinking from personal understanding.
4. It should update regularly.

Therefore, a lot of blogs that Google ranks high will not appear in this list. Please leave your comment if you know some blogs that should be in this list. As this list is quickly growing, please only provide good sites.

	Name(Site/People)	Country	Key Words
0	Java SED	America	not a blog, a useful Java tool
1	Adam Bien	Germany	Java EE
2	Antonio Goncalves	France	Author of Java EE 7
3	Henrik Warne	Sweden	Thoughts on programming

4	Billy Yarosh	America	Coding Cures
5	Lars Vogel	Germany	Android and Eclipse
6	Peter Verhas	Hungary	Pure Java
7	Martin Fowler	America	Author, Speaker
8	Bozhidar Bozhanov	Bulgaria	JEE
9	Richard Warburton	UK	Java 8 Lambdas
10	Bear Giles	America	JEE
11	Marginally Interesting	Germany	Machine Learning
12	Pascal Alma	America	JEE
13	Dror Helper	America	Consultant
14	Juri Strumpflohner	Italy	JavaScript
15	Reza Rahman	America	Java EE/Glassfish
16	Phil Whelan	Canada	Web
17	Brett Porter	Australia	Co-author of Apache Maven 2
18	Ben McCann	America	Co-founder at Connectifier
19	Java Posse	America	Some useful links
20	Mark Needham	UK	Data
21	Iris Shoor	Israel	Debug
22	Yifan Peng	America	Graduate Student
23	Nikita Salnikov Tarnovski	Estonia	Memory Leaks
24	Dustin Marx	America	Actual Events
25	Bart Bakker	Netherland	Agile
26	Gunnar Peipman	America	non-java
27	Dave Fecak	America	Job Tips for Programmers
28	JOOQ	Switzerland	SQL
29	Petri Kainulainen	Finland	Web
30	Informatech CR	Costa Rica	
31	Arun Gupta	America	Java EE
32	Mechanical Sympathy	UK	Performance
33	Extreme Enthusiasm	Italy	Agile
34	Steve Blank	America	Author of The Startup Owner's Manual
35	Oliver Gierke	Germany	SpringSource
36	Nicolas Fr?nkel	Switzerland	Java EE

37	Blaise Doughan	America	XML and JSON
38	Vlad Mihalcea	Romania	Software Integration
39	Kevin Lee	Australia	Web
40	Mikhail Vorontsov	Australia	Performance
41	Jakob Jenkov	Denmark	Software Architecture
42	Jim Weaver		Rich Client Java
43	Jonathan Giles	New Zealand	Java FX
44	Stephen Chin	America	Java FX
45	Matt Raible	America	Open Source Frameworks
46	Peter Lawrey	UK	Core Java
47	Gregor Riegler	Austria	OO Design, XP
48	Jos Dirksen	Netherlands	SOA, HTML 5
49	Alexander J. Turner	UK	Information, News And Views
50	Java Advent		
51	John Purcell	Hungary	Tutorials
52	Transylvania JUG	UK	
53	Java Roots		Spring
54	Java Training	Greece	training
55	Allan Kelly	UK	Software
56	Samuel Santos	Portugal	Java EE
57	Steve Smith	UK	Agile
58	Niklas Schlimm	Germany	Multithreading
59	Shrutarshi Basu	America	PhD, Computer Science
60	Anton Arhipov	Estonia	Java EE
61	Charles Nutter	America	JVM
62	RedStack	America	SOA, JVM
63	James Bloom	America	JVM
64	Pierre-Hugues Charbonneau	Canada	Java EE
65	Eugen Paraschiv	Romania	Java Web
66	Wayne Beaton	America	Eclipse
67	Jeff Atwood	America	Stack Overflow
68	Stuart Marks	America	Oracle
69	Ben Stopford	UK	Data Platform

	Program Creek	America	Deep Understanding of Java Core

Please leave your comment if you know some high quality Java blogs or find any errors in the list above. I will keep updating this list, but limit it up to 100! As this list is being read by thousands of people, if you don't want to be on the list, I can also take your blog off from the list.

* This collection reflects my personal opinion. Not all links from comments will be added to the list.

原文出处：top-100-java-developers-blogs/

leetcode_Combination Sum

描述：

Given a set of candidate numbers (C) and a target number (T), find all unique combinations inC where the candidate numbers sums toT.

Thesamerepeated number may be chosen fromCunlimited number of times.

Note:

- **All numbers (including target) will be positive integers.**
- **Elements in a combination (a1,a2, … ,ak) must be in non-descending order. (ie,a1a2 … ak).**
- **The solution set must not contain duplicate combinations.**

For example, given candidate set`2,3,6,7`and target`7`,
A solution set is:

```
[7]
[2, 2, 3]
```

思路：

这种题目，一般要用到递归或回溯两种方法，用回溯法试过，代码规模总是越来越庞大，但最终还是没能通过所有的测试用例，^-^!

用递归的话这题目看着要容易理解的多，每递归一次target要变为target=target-candidates[i]，并将开始index赋值为i，当target==0时，条件满足，如果target<candidates[i]，这轮循环结束，方法出栈，当前的i++，继续循环。

By mnmlist,2015-9-1 11:22:43

总之，用递归来解决问题，难想到，也不太容易被看懂。

代码：

```
public List<List<Integer>> combinationSum(int[] candidates, int target)
{
 List<List<Integer>>listResult=new ArrayList<List<Integer>>();
 ArrayList<Integer>list=new ArrayList<Integer>();
 Arrays.sort(candidates);
 combination(listResult, list, candidates, target,0);
 return listResult;

}
public void combination(List<List<Integer>>listResult,
  ArrayList<Integer>list,int[] candidates, int target,int begin)
{
 if(target==0)
 {
  listResult.add(list);
  return;
 }
 for(int i=begin;i<candidates.length&&target>=candidates[i];i++)
 {
  ArrayList<Integer>copy=new ArrayList<Integer>(list);
  copy.add(candidates[i]);
  combination(listResult, copy, candidates,target- candidates[i],i);
 }
}
```


leetcode_Majority Element II

描述：

Given an integer array of size n, find all elements that appear more than ? n/3 ? times. The algorithm should run in linear time and in O(1) space.

思路：

By mnmlist,2015-9-1 11:22:43

1.跟让求一个数组中出现次数超过一半的元素类似，这次让求出现次数超过n/3次数的元素，原则上是类似的，但难度要大好多

2.因为是类似的，所以总体思路应该还是一样的，但具体怎么做呢？这次维护一个curNum1和curNum2两个锚点并用count1和count2来计算锚点出现的次数，因为求超过n/3次数的元素，所以数组中可能存在两个这样的元素，其它的就和Majority Element类似了，只有nums[i]与curNum1和curNum2均不相等时才会count1--，count2--

3.和Majority Element不同，最后剩下的curNum1和curNum2有可能不是出现次数超过n/3的那个数，所以还需要统计下curNum1和curNum2出现的真实次数，但这并不等于说会漏掉出现次数超过n/3的数字

代码：

```
public List<Integer> majorityElement(int[] nums) {
 List<Integer>list=new ArrayList<Integer>();
 if(nums==null)
  return list;
       int count1=0,curNum1=0;
       int count2=0,curNum2=1;
       for(int num:nums)
       {
        if(num==curNum1)
         count1++;
        else if(num==curNum2)
         count2++;
        else if(count1==0)
        {
         curNum1=num;
         count1=1;
        }else if(count2==0)
        {
         curNum2=num;
         count2=1;
        }else {
   count1--;
   count2--;
  }
       }
       count1=0;
       count2=0;
       for(int num:nums)
       {
        if(num==curNum1)
         count1++;
        else if(num==curNum2)
```

By mnmlist,2015-9-1 11:22:43

```
            count2++;
        }
        if(count1>nums.length/3)list.add(curNum1);
        if(count2>nums.length/3)list.add(curNum2);
        return list;
    }
```


leetcode_Implement Stack using Queues

描述：

Implement the following operations of a stack using queues.

push(x) -- Push element x onto stack.
pop() -- Removes the element on top of the stack.
top() -- Get the top element.
empty() -- Return whether the stack is empty.
Notes:
You must use only standard operations of a queue -- which means only push to back, peek/pop from front, size, and is empty operations are valid.
Depending on your language, queue may not be supported natively. You may simulate a queue by using a list or deque (double-ended queue), as long as you use only standard operations of a queue.
You may assume that all operations are valid (for example, no pop or top operations will be called on an empty stack).

思路：

1.跟用栈实现队列不同，我感觉用队列去实现栈要困难的多，以至于根本就想不起来，参考了网络上的思路才算是有了写头绪，原来是这个这个样子。。。

2.如果用栈来实现队列还算可以理解的话，但用队列来实现栈就只有两个字来形容:no zuo no die!，下面我就来描述下这种奇葩的思路：

3.用两个队列queue1和queue2来模拟栈，具体怎么模拟呢？queue1是操作队列，先进先出，queue2是中转队列，每次取元素时，将0~size-2个元素先中转到queue2中，然

By mnmlist,2015-9-1 11:22:43

后取出queue1的最后一个元素，然后，对，然后在将queue1中的元素再依次放回queue2中，直至stack2为空且没有新的元素进栈，循环终止

4.好有想法的一个思路，希望还有更简洁明快的思路出现。

代码：

```
class MyStack {
   Deque<Integer>queue1=new LinkedList<Integer>();
Deque<Integer>queue2=new LinkedList<Integer>();
// Push element x onto stack.
public void push(int x)
{
 queue1.addLast(x);;
}

// Removes the element on top of the stack.
public void pop()
{
 if(!this.empty())
 {
  int size=queue1.size()-1;
  for(int i=0;i<size;i++)
   queue2.addLast(queue1.pop());
  queue1.pop();
 }
 queue1.addAll(queue2);
 queue2.clear();
}

// Get the top element.
public int top()
{
 int num=0;
 if(!this.empty())
 {
  int size=queue1.size()-1;
  for(int i=0;i<size;i++)
   queue2.addLast(queue1.pop());
  num=queue1.pop();
  queue1.addAll(queue2);
  queue1.addLast(num);
  queue2.clear();
 }
 return num;
}

// Return whether the stack is empty.
public boolean empty()
{
```

By mnmlist,2015-9-1 11:22:43

```
 if(queue1.size()==0)
  return true;
 return false;
}
}
```


leetcode_Majority Element

描述：

Given an array of size n, find the majority element. The majority element is the element that appears more than ? n/2 ? times.
You may assume that the array is non-empty and the majority element always exist in the array.

思路：

1.对于这种问题，相信很多人都会想起来直接对数组拍下序，然后取中间值即可return nums[nums.length/2];地球人都知道，哈哈。。。Time:O(n(log(n)),Space:O(k)

2.啥？还有更好的方法？对，其实还真有，先找第一个数字curNum作为锚，count统计curNum目前为止出现的次数，依次遍历数组nums[i],如果
nums[i]==curNum，count++；如果不等呢，count--，此时若count==0；再换锚点为curNum=nums[i]，并令count==1

3.重复2中的步骤直至遍历完所有的数组元素，那么最后剩下的nagecurNum就是在数组中出现次数大于一半的数字。

代码：

By mnmlist,2015-9-1 11:22:43

```
public int majorityElement(int[] nums) {
       int count=1,curNum=nums[0];
       for(int i=1;i<nums.length;i++)
       {
           if(nums[i]==curNum)
               count++;
           else
           {
               count--;
               if(count==0)
               {
                   curNum=nums[i];
                   count=1;
               }
           }
       }
       return curNum;
   }
```


leetcode_Summary Ranges

描述：

Given a sorted integer array without duplicates, return the summary of its ranges.

For example, given `[0,1,2,4,5,7]`**, return** `["0->2","4->5","7"].`

思路：

类似字符串的压缩，就是统计下连续数组元素并判断该数组元素是否连续递增，将连续递增的系列元素用startNum->endNum进行压缩，当然对于孤立的数字点，直接原样压缩即可。

代码：

By mnmlist,2015-9-1 11:22:43

```
public List<String> summaryRanges(int[] nums) {
       List<String>listStr=new ArrayList<String>();
       if(nums==null||nums.length==0)
           return listStr;
       int start=0,end=0,i=0;
       while(i<nums.length)
       {
           start=nums[i];
           while(i+1<nums.length&&nums[i+1]-nums[i]==1)
               i=i+1;
           end=nums[i];
           if(start==end)
               listStr.add(String.valueOf(start));
           else
               listStr.add(""+start+"->"+end);//jdk1.7StringBuilderZzz
           i++;
       }
       return listStr;

   }
```


leetcode_Implement Queue using Stacks

描述：

Implement the following operations of a queue using stacks.

- **push(x) -- Push element x to the back of queue.**
- **pop() -- Removes the element from in front of queue.**
- **peek() -- Get the front element.**
- **empty() -- Return whether the queue is empty.**

Notes:

- **You must useonlystandard operations of a stack -- which means only`push to top`, `peek/pop from top`,`size`, and`is empty`operations are valid.**
- **Depending on your language, stack may not be supported natively. You may simulate a stack by using a list or deque (double-ended queue), as long as you use only standard operations of a stack.**
- **You may assume that all operations are valid (for example, no pop or peek operations will be called on an empty queue).**

思路：

By mnmlist,2015-9-1 11:22:43

1.用栈来实现一个队列，也就是用后进先出的栈实现先进先出的队列

2.这个还是很难想的，但总之还是比用队列来实现栈容易想，大概就是用两个栈stack1和stack2来模拟队列

3.所有的元素都从stack1进栈，所有元素都从stack2出栈，当stack2为空的时候，将stack1中的所有元素出栈并全部push到stack2中去，然后从stack2出栈

4.由于栈是后进先出的，两次后进先出的操作就实现了队列的功能

代码：

```
class MyQueue {
   Stack<Integer>stack1=new Stack<Integer>();
Stack<Integer>stack2=new Stack<Integer>();
// Push element x to the back of queue.
   public void push(int x) {
       stack1.push(x);
   }

   // Removes the element from in front of queue.
   public void pop() {
       if(!stack2.empty())
        stack2.pop();
       else
       {
  while(!stack1.empty())
  {
   stack2.push(stack1.pop());
  }
  stack2.pop();
 }
   }

   // Get the front element.
   public int peek() {
    int num=0;
    if(!stack2.empty())
        num= stack2.peek();
       else
       {
  while(!stack1.empty())
  {
   stack2.push(stack1.pop());
  }
```

By mnmlist,2015-9-1 11:22:43

```
  num= stack2.peek();
 }
       return num;
   }

   // Return whether the queue is empty.
   public boolean empty() {
       if(stack1.empty()&&stack2.empty())
        return true;
       return false;
   }
}
```


leetcode_Invert Binary Tree

描述：

Invert a binary tree.

```
     4
   /   \
  2     7
 / \   / \
1   3 6   9
```

to

```
     4
   /   \
  7     2
 / \   / \
9   6 3   1
```

Trivia:
This problem was inspired bythis original tweetbyMax Howell:

> **Google: 90% of our engineers use the software you wrote (Homebrew), but you can’t invert a binary tree on a whiteboard so fuck off.**

By mnmlist,2015-9-1 11:22:43

思路：

1.记得做树的问题时，有两题类似的，一题是判断二叉树是否镜像对称，另外一题是判断两个二叉树节点的值是否相等，同样都是用递归的方式来做

2.这一题跟镜像对称哪一题类似，从上至下，交换同一个父节点的左右子节点即可。

代码：

```
public TreeNode invertTree(TreeNode root) {
       if(root==null)
           return root;

       if(root.left!=null||root.right!=null)
       {
        LinkedList<TreeNode> list=new LinkedList<TreeNode>();
        list.addLast(root);
        TreeNode curNode=null;
           while(!list.isEmpty())
           {
            curNode=list.removeFirst();
            changeTree(curNode);
            if(curNode.left!=null)
             list.addLast(curNode.left);
            if(curNode.right!=null)
             list.addLast(curNode.right);
           }
       }
       return root;
   }

public void changeTree(TreeNode root)
{
 if (root.left != null || root.right != null)
 {
  TreeNode temp = root.left;
  root.left = root.right;
  root.right = temp;
 }
}
```

By mnmlist,2015-9-1 11:22:43

leetcode_Kth Smallest Element in a BST

描述：

Given a binary search tree, write a function kthSmallest to find the kth smallest element in it.
Note:
You may assume k is always valid, 1 ≤ k ≤ BST's total elements.
Follow up:
What if the BST is modified (insert/delete operations) often and you need to find the kth smallest frequently? How would you optimize the kthSmallest routine?
Hint:
Try to utilize the property of a BST.Show More Hint

思路：

1.一般涉及到树的问题用树的遍历方式来解决的可能性比较大，或者就是递归了

2.对于本题，一种很朴素的解决方式就是不是二叉查找树么，中序遍历时不就是一个有序的序列么，直接上中序遍历，遍历到第K个数即是本题要求的Kth Smallest Element in a BST

代码：

```
public int kthSmallest(TreeNode root, int k) {
  int count=0;
     Stack<TreeNode>st=new Stack<TreeNode>();
     st.push(root);
     TreeNode top=null;
     while(!st.empty())
     {
      top=st.peek();
      while(top.left!=null)
      {
       st.push(top.left);
       top=top.left;
      }
      while(top.right==null)
      {
       count++;
       if(count==k)
```

By mnmlist,2015-9-1 11:22:43

```
          return top.val;
       st.pop();
       if(!st.empty())
        top=st.peek();
       else
        break;
      }
      if(!st.empty())
      {
       count++;
       if(count==k)
          return top.val;
       st.pop();
       st.push(top.right);
      }

     }
     return -1;
   }
```


leetcode_Palindrome Linked List

描述：

Given a singly linked list, determine if it is a palindrome.
Follow up:
Could you do it in O(n) time and O(1) space?

思路：

1.有大概两种思路，第一种将linkedLIst表中的数据读取到一个ArrayList中，然后从ArrayList前后开始比较即可。Time:O(n),Space:O(n)

2.第二种用快慢指针找到中间节点，然后将链表断成两个表，并将后面的链表逆转，然后顺序比较两个链表即可。Time:O(n),Space:O(1)

By mnmlist,2015-9-1 11:22:43

3.显然只有第二种方法符合题意。

代码：

```
public boolean isPalindrome(ListNode head) {
        if(head==null||head.next==null)
            return true;
        ListNode p=head,temp=head,quick=head;
        while(temp!=null&&quick!=null)
        {
            temp=temp.next;
            if(quick.next==null)
                break;
            quick=quick.next.next;
        }
        temp=reverseList(temp);
        p=head;
        while(temp!=null&&p!=null)
        {
            if(temp.val!=p.val)
                return false;
            temp=temp.next;
            p=p.next;
        }
        return true;
    }
    public ListNode reverseList(ListNode head)
    {
        ListNode tempHead=new ListNode(-1);
        ListNode p=head.next,q=null;
        tempHead.next=head;
        head.next=null;
        while(p!=null)
        {
            q=p;
            p=p.next;
            q.next=tempHead.next;
            tempHead.next=q;
        }
        return tempHead.next;
    }
```

leetcode_Power of Two

描述：

Given an integer, write a function to determine if it is a power of two.

思路：

1.很有意思的一道题目，大致有三种方法：

2.第一种对num进行向右移位，每次移动一个Bit位并和0x1作&运算，共需移位31次，统计num中1出现的次数，有且仅出现一次的话就返回true

3.第二种方法是令tempNum=1，让num和tempNum作&运算，每次移位结束将tempNum左移位，工需移位31次，1有且仅出现一次则返回true

4.第三种方法有点惨无人道，boolean flag=((num&(num-1))==0);flag为真的话则返回true

代码：

仅贴出第三种方法的代码：

```
public boolean isPowerOfTwo(int n)
{
    if(n<=0)
        return false;
    return (n&(n-1))==0;
}
```

By mnmlist,2015-9-1 11:22:43

leetcode_Median of Two Sorted Arrays

描述：

There are two sorted arrays nums1 and nums2 of size m and n respectively. Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

思路：

1.题意大概为查找两个数组 的中位数，这个不难，O（N）的时间复杂度就可以解决，但题目的要求是用 O(log (m+n))的时间复杂度来解决该问题！

2.由于时间复杂度为 O(log (m+n))，一般会用到二分法，每次取数组nums1和nums2所剩下的部分的中间值mid1和mid2，然后比较两个数组的中间值大小，较大的一个取数组较小的一部分，较小的mid数组取较大的一部分。由于每次都要多虑到一部分不符合条件的，所以刚开始要求第k=（len1+len2)/2个数的k每次都要更新。

3.重负步骤2，直至每个数组剩下的有效部分为0，则另一个数组的nums[start+k]即为中位数；另外一种可能是最后当k==0时，两个数组有效部分的开始的那个数肯定为要求的数。

哎，表述能力不行啊，自己都快被绕晕了(^-^)!

代码：

```
public double findMedianSortedArrays(int[] nums1, int[] nums2) {
       if((nums1==null||nums1.length==0)&&(nums2==null||nums2.length==0))
           return 0;
       if(nums1==null||nums1.length==0)
           return getMidOfOneArray(nums2);
       if(nums2==null||nums2.length==0)
           return getMidOfOneArray(nums1);
 int totalLen = nums1.length + nums2.length;
 if ((totalLen & 0x1) == 1)
  return findMid(nums1, 0, nums1.length - 1, nums2, 0,
    nums2.length - 1, totalLen / 2);
 else
  return (findMid(nums1, 0, nums1.length - 1, nums2, 0,
    nums2.length - 1, totalLen / 2) + findMid(nums1, 0,
    nums1.length - 1, nums2, 0, nums2.length - 1,
```

```
    totalLen / 2 - 1)) * 0.5;
}

public double findMid(int nums1[], int start1, int end1, int nums2[],
  int start2, int end2, int k) {
 int aLen = end1 - start1 + 1;
 int bLen = end2 - start2 + 1;
 if (aLen == 0)
  return nums2[start2 + k];
 if (bLen == 0)
  return nums1[start1 + k];
 if (k == 0)
  return nums1[start1] < nums2[start2] ? nums1[start1]
    : nums2[start2];
 int aMid = aLen * k / (aLen + bLen);
 int bMid = k - aMid - 1;
 aMid = aMid + start1;
 bMid = bMid + start2;
 if (nums1[aMid] > nums2[bMid]) {
  k = k - (bMid - start2 + 1);
  end1 = aMid;
  start2 = bMid + 1;
 } else {
  k = k - (aMid - start1 + 1);
  end2 = bMid;
  start1 = aMid + 1;
 }
 return findMid(nums1, start1, end1, nums2, start2, end2, k);
}
public double getMidOfOneArray(int arr[])
{
    if((arr.length&0x1)==0)
        return (arr[arr.length/2]+arr[arr.length/2-1])*0.5;
    return arr[arr.length/2];
}
```


leetcode_Valid Anagram

描述：

Given two stringssandt, write a function to determine iftis an anagram ofs.

By mnmlist,2015-9-1 11:22:43

For example,
s= "anagram",t= "nagaram", return true.
s= "rat",t= "car", return false.

Note:
You may assume the string contains only lowercase alphabets.

思路：

anagram

true

1.StringcharArraynlognnlogn

2.26OkOn

3.2

代码：

```
public boolean isAnagram(String s, String t) {
       if(s==null&&t==null)
           return true;
       if(s==null||t==null)
           return false;
       int lens=s.length();
       int lent=t.length();
       if(lens!=lent)
           return false;
       int charCount1[]=new int[26];
       int charCount2[]=new int[26];
       for(int i=0;i<lens;i++)
       {
           ++charCount1[s.charAt(i)-'a'];
           ++charCount2[t.charAt(i)-'a'];
       }
       for(int i=0;i<26;i++)
       {
           if(charCount1[i]!=charCount2[i])
               return false;
       }
       return true;

   }
```

leetcode_Sqrt(x)

描述：

Implement int sqrt(int x).
Compute and return the square root of x.

思路：

按题意来说是实现一个api方法，对整数开根号

1.用二分的方法来对一个整数开根号，每次循环求start和end之间的值mid来计算，若mid*mid>x，则end变为mid，若mid*mid<x,则start取mid，直至mid*mid=x

2.对于1中的思路有两个不足的地方，假如x为99呢，mid*mid永远不可能为99，因为sqrt(99)==9，而当mid变的很大时mid*mid直接就溢出了

3.对于2.种提到的不足之处，可用(mid==start||mid==end)来弥补，当mid==start或mid==end任何一个条件满足即可。而mid*mid溢出的问题可以用x/mid mid来解决

4.对整数开根号能不能升级为对float、double类型的数据开根号呢，当然可以，判断条件只需要x/mid-mid小于某个数(也就是精度)即可。

代码：

Demo版本：

```
public int mySqrt(int x)
{
```

By mnmlist,2015-9-1 11:22:43

```
 if (x < 0)
  return -1;
 int start = 0, end = x, mid=x/2;
 if(x*x==x)//to judge the number 1
  return x;
 long tempNum = 0,tempMid=0;
 while (mid!= start && mid!= end)
 {
  tempMid=mid;
  tempNum=tempMid*tempMid;
  if (tempNum<x)
  {
   start=mid;
   mid =(mid+ end) / 2  ;
  }
  else if (tempNum>x)
  {
   end=mid;
   mid =(mid+ start) / 2 ;
  }else
   break;
 }
 return mid;
}
```

改进版本：

```
public int mySqrt(int x)
{
 if (x < 0)
  return -1;
 int start = 0, end = x, mid=x/2;
 if(x*x==x)//to judge the number 1
  return x;
 int tempNum = 0,tempMid=0;
 while (mid!= start && mid!= end)
 {
  tempMid=mid;
  tempNum=x/tempMid-tempMid;
  if (tempNum>0)
  {
   start=mid;
   mid =(mid+ end) / 2  ;

  }
  else if (tempNum<0)
  {
   end=(int)mid;
   mid =(mid+ start) / 2 ;
  }else if(tempNum==0) {
```

```
   break;
  }

 }
 return mid;
}
```


leetcode_Count and Say

描述：

The count-and-say sequence is the sequence of integers beginning as follows:
`1, 11, 21, 1211, 111221, ...`

`1`is read off as`"one 1"`or`11`.
`11`is read off as`"two 1s"`or`21`.
`21`is read off as`"one 2`, then`one 1"`or`1211`.

Given an integern, generate thenthsequence.

Note: The sequence of integers will be represented as a string.

思路：

很有意思的一个题目，由于题目描述的比较简单，刚开始竟然没有读懂。。。

1.题目的意思是给你一个数n，让你输出前n个按照给定规则生成的数字串

2.而生成该数字串的规则为后一个串是前一个串的读法

3.而字符串是如何读的呢？是将该数字串种连续相同的数字进行压缩编码即111读作3个1，即31,11读作2个1，即21，以此类推。

代码：

By mnmlist,2015-9-1 11:22:43

```
public String countAndSay(int n)
{
 if (n <= 0)
  return new String("");
 String strNum = "1";
 for (int i = 1; i < n; i++)//"1"
 {
  strNum = getString(strNum);//
 }
 return strNum;
}

public String getString(String strNum)
{
 int numLen = strNum.length();
 StringBuilder sb = new StringBuilder();
 int indexStart = 0, indexEnd = 0;
 char ch;
 while (indexStart < numLen)//
 {
  ch = strNum.charAt(indexStart);
  indexEnd = indexStart + 1;
  while (indexEnd < numLen && ch == strNum.charAt(indexEnd))
   indexEnd++;
  sb.append(indexEnd - indexStart);
  sb.append(ch);
  indexStart = indexEnd;
 }
 return sb.toString();
}
```


leetcode_Search a 2D Matrix II

描述：

Write an efficient algorithm that searches for a value in anmxnmatrix. This matrix has the following properties:

- **Integers in each row are sorted in ascending from left to right.**

By mnmlist,2015-9-1 11:22:43

- **Integers in each column are sorted in ascending from top to bottom.**

For example,

Consider the following matrix:

```
[
 [1,   4,  7, 11, 15],
 [2,   5,  8, 12, 19],
 [3,   6,  9, 16, 22],
 [10, 13, 14, 17, 24],
 [18, 21, 23, 26, 30]
]
```

Giventarget=`5`, return`true`.

Giventarget=`20`, return`false`.

思路：

本题有两种思路，首先想到的是对该矩阵的每一行做二分查找，后来跟人探讨时才知道还有另外一种O（ n ）时间复杂的方法，下面就细说下这个O（ n ）时间复杂度的方法

1.首先从第一行的最后一个开始，假设该值为x，当x大于target的时候，那target肯定小于该列的其它值，因为该列是递增的，colNum--；

2.同理，如果target大于x的时候，说明target大于该行的其它值，因为改行也是递增的，即rowNum++；

3.由于每次以一行的速度来排除，所以O（ n ）的时间复杂度既可以解决该问题。

代码：

O（ nlogn ）的方法，对每一行进行二分查找

```
public boolean searchMatrix(int[][] matrix, int target) {
       if(matrix==null)
           return false;
       int endIndex=0;
       if(matrix[matrix.length-1][matrix[0].length-1]<target)//target
           return false;
       for(int i=0;i<matrix.length;i++)//
       {
```

```
            if(matrix[i][0]>target)
            {
                endIndex=i;
                break;
            }
        }
        if(matrix[matrix.length-1][0]<=target)
            endIndex=matrix.length;
        for(int i=0;i<endIndex;i++)//
        {
            if(binarySearch(matrix[i],target))
                return true;
        }
        return false;
    }
    public boolean binarySearch(int []arr,int target)//
    {
        int start=0,end=arr.length-1,mid=0;
        while(start<=end)
        {
            mid=(start+end)/2;
            if(arr[mid]==target)
                return true;
            else if(arr[mid]>target)
                end=mid-1;
            else
                start=mid+1;
        }
        return false;
    }
```

O（N）时间复杂度的解法

```
public boolean searchMatrix(int[][] matrix, int target) {
        if(matrix==null)
            return false;
        int endIndex=matrix.length;
        int rowNum=0,columNum=matrix[0].length-1;
        while(rowNum<endIndex&&columNum>=0)
        {
            if(matrix[rowNum][columNum]==target)
                return true;
            else if(matrix[rowNum][columNum]<target)
                rowNum++;
            else if(matrix[rowNum][columNum]>target)
                columNum--;
        }
        return false;
    }
```

leetcode_Rectangle Area

描述：

Find the total area covered by tworectilinearrectangles in a2Dplane.

Each rectangle is defined by its bottom left corner and top right corner as shown in the figure.

Assume that the total area is never beyond the maximum possible value ofint.

思路：

1.本题目的意思是，求totalAreage=areage(A)||areage(B)，就是求两个矩形的并集。

2.这题有点绕，解决问题的关键是求出两个矩形相交部分的面积，然后两个矩形的面积之和减去两个矩形相交的部分即是最后要求的结果

By mnmlist,2015-9-1 11:22:43

3.但是如何求两个矩形相交的部分呢，假设两个矩形相交，那就要求两个矩形相交部分的左边的最大值，右边的最小值，上边的最小值，下边的最大值

4.然后再根据矩形相交时的位置（比如左上右下方向，左下右上等来判断要减去的面积）

代码：

```
public int computeArea(int A, int B, int C, int D, int E, int F, int G,
  int H)
{
 int totalArea = (D - B) * (C - A) + (H - F) * (G - E);
 int left, bottom, top, right;
 left = getLeft(A, E);
 bottom = getBottom(B, F);
 top = getTop(D, H);
 right = getRight(C, G);
 if(left<right&&bottom<top)//
 {
  if(A<=E||B<=F)
  {
      if(!(C<E&&D<F))
    totalArea -= (right - left) * (top - bottom);
  }
  if(E<A&&F<B)//
  {
   if(!(G<A&&H<B))
    totalArea -= (right - left) * (top - bottom);
  }
 }
 return totalArea;
}

public int getLeft(int y1, int y2)
{
 return y1 > y2 ? y1 : y2;
}

public int getBottom(int x1, int x2)
{
 return x1 > x2 ? x1 : x2;
}

public int getTop(int x1, int x2)
{
 return x1 < x2 ? x1 : x2;
}

public int getRight(int y1, int y2)
{
```

By mnmlist,2015-9-1 11:22:43

```
 return y1 < y2 ? y1 : y2;
}
```


leetcode_Delete Node in a Linked List

描述：

Write a function to delete a node (except the tail) in a singly linked list, given only access to that node.
Supposed the linked list is 1 -> 2 -> 3 -> 4 and you are given the third node with value 3, the linked list should become 1 -> 2 -> 4 after calling your function.

思路：

1.若不是最后一个，将当前节点的下一个节点的值赋值给当前节点，然后直接删除下一个节点即可。

2.若是最后一个，即currentNode.next==null，这就需要从头遍历一下当前链表了，找到倒数第二个节点，直接node.next=node.next.next；即可完成删除工作。

当然，本题目要求的仅仅是将当前节点删除即可。

代码：

```
public void deleteNode(ListNode node) {
       if(node==null)
           return;
       node.val=node.next.val;
       node.next=node.next.next;
   }
```

leetcode_Evaluate Reverse Polish Notation

描述：

Evaluate the value of an arithmetic expression in Reverse Polish Notation.
Valid operators are +, -, *, /. Each operand may be an integer or another expression.
Some examples:
["2", "1", "+", "3", "*"] -> ((2 + 1) * 3) -> 9
["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6

思路：

1.就像题目所描述的，计算逆波兰数学表达式的结果，循环访问字符串数组类型的表达式

2.遍历的字符是数字，将数字进栈，是数学符号的时，将连续的两个数字出栈并计算出结果，然后将结果再入栈

3.重复步骤2直至所有的数组元素被访问完毕。

代码：

//代码通俗易懂，略冗余。

```
public int evalRPN(String[] tokens) {
       if(tokens==null||tokens.length==0)
           return 0;
       if(tokens.length==1)
           return Integer.parseInt(tokens[0]);
       int parseNum=0;
```

By mnmlist,2015-9-1 11:22:43

```
       int num1=0,num2=0;
       Stack<Integer>st=new Stack<Integer>();
       for(int i=0;i<tokens.length;i++)//iterate the item of the array
       {
        if(isOperator(tokens[i]))//if the item is operator,caculate the numbers
        {
         num2=st.peek();
         st.pop();
         num1=st.peek();
         st.pop();
         parseNum=evaluteNums(num1,num2,tokens[i]);
         if(i+1==tokens.length)
          return parseNum;
         st.push(parseNum);
        }else {
   st.push(Integer.parseInt(tokens[i]));//if the item is number,push to the stack
  }

       }
       return parseNum;
   }
public int evaluteNums(int num1,int num2,String operator)
{
 if(operator.equals("+"))
  return num1+num2;
 else if(operator.equals("-"))
  return num1-num2;
 else if(operator.equals("*"))
  return num1*num2;
 else
  return num1/num2;
}
public boolean isOperator(String str)
{
 char operator=str.charAt(0);
 if(operator=='+'||operator=='-'||operator=='*'||operator=='/')
 {
  if(str.length()==1)
   return true;
 }
 return false;
}
```


leetcode_Evaluate Reverse Polish Notation

By mnmlist,2015-9-1 11:22:43

描述：

Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Valid operators are +, -, *, /. Each operand may be an integer or another expression.

Some examples:
["2", "1", "+", "3", "*"] -> ((2 + 1) * 3) -> 9
["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6

思路：

1.就像题目所描述的，计算逆波兰数学表达式的结果，循环访问字符串数组类型的表达式

2.遍历的字符是数字，将数字进栈，是数学符号的时，将连续的两个数字出栈并计算出结果，然后将结果再入栈

3.重复步骤2直至所有的数组元素被访问完毕。

代码：

//代码通俗易懂，略冗余。

```
public int evalRPN(String[] tokens) {
       if(tokens==null||tokens.length==0)
           return 0;
       if(tokens.length==1)
           return Integer.parseInt(tokens[0]);
       int parseNum=0;
       int num1=0,num2=0;
       Stack<Integer>st=new Stack<Integer>();
       for(int i=0;i<tokens.length;i++)//iterate the item of the array
       {
        if(isOperator(tokens[i]))//if the item is operator,caculate the numbers
        {
         num2=st.peek();
         st.pop();
         num1=st.peek();
         st.pop();
```

```
          parseNum=evaluteNums(num1,num2,tokens[i]);
          if(i+1==tokens.length)
           return parseNum;
          st.push(parseNum);
         }else {
   st.push(Integer.parseInt(tokens[i]));//if the item is number,push to the stack
  }

       }
       return parseNum;
   }
public int evaluteNums(int num1,int num2,String operator)
{
 if(operator.equals("+"))
  return num1+num2;
 else if(operator.equals("-"))
  return num1-num2;
 else if(operator.equals("*"))
  return num1*num2;
 else
  return num1/num2;
}
public boolean isOperator(String str)
{
 char operator=str.charAt(0);
 if(operator=='+'||operator=='-'||operator=='*'||operator=='/')
 {
  if(str.length()==1)
   return true;
 }
 return false;
}
```