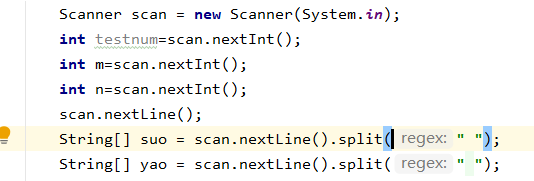
1. 输入扫描 ，想读取不确定个数的字符串或者整数，使用nextline方法，



注意nextint 和nextline使用要一个nextline给隔开，否则他会继续连续读取，使得第一个nextline失效。

补充：C++中读取不确定一行整数的方法

string num ;

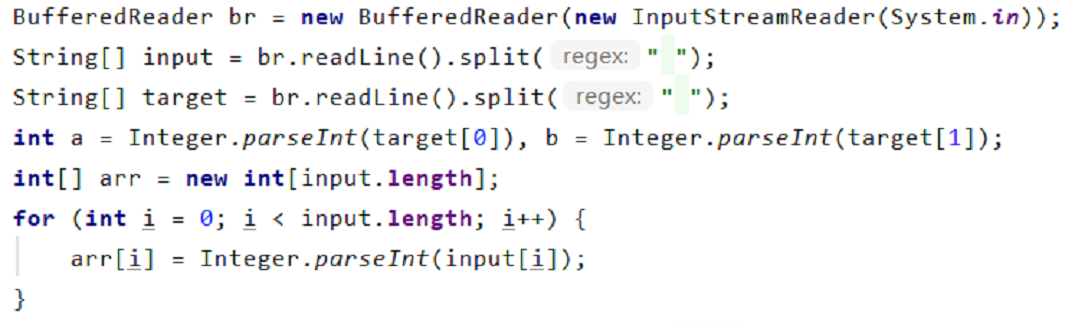
do {

cin >> num;

inputs.push\_back(num);

} while (getchar() != '\n');

但是该方法读取速度比较慢，因此有缓存流读取方法bufferedread方法



1. JAVA整型转换为字符型只需要使用（char）强制转换。
2. JAVA中List<Integer> l1=new ArrayList<>(); 这里表示一个动态数组，和C++中的vector较为相像。
3. JAVA中的String不同于C++可以之间使用【】进行索引，它需要getchar方法，同样的，数组链表、链表等结构中，也都是使用get方法进行索引，但需要注意的是链表用get方法索引速度是非常慢的。
4. 数组、链表等基本类型都进行了iterator迭代器的重写，因可以使用简化的for循环方法。

for（int i:arr）{} (arr为一个Array)。需要注意的是String在JAVA中并不像C++里一样，类似一个数组，~~所以不支持迭代器的使用。~~并不是不支持，使用应该是for(String temp : stringtemp){ 该形式

6.ASCALL码表：0是48（48-57）；A是65（65-90）；a是97。（97-122）

1. 异步回调
2. 并发：一个CPU（采用时间片）同时执行多个任务。多个人做同一件事
3. 签名：关键字和函数返回值，方法名以及一串各种类型的参数。
4. String.valueof(i)效率比单纯"" + i字符串拼接高。
5. JAVA中没有设计指针是因为指针容易出错，所以在JAVA中创建引用的方法只有一种new，改变引用的方法也只有一种：赋值语句。因此对JAVA来说，引用为安全指针。
6. Static关键字会创建静态变量，静态变量不和任何具体的对象相关联，在老的编程语言中，其被称为全局变量。静态变量存在方法区。
7. JAVA中与C++不同，它规定了每个基本数据类型的大小，而C++中的数据类型大小可能随着机器的不同（32，64位）而变化，这也是JAVA移植性较好的原因之一。

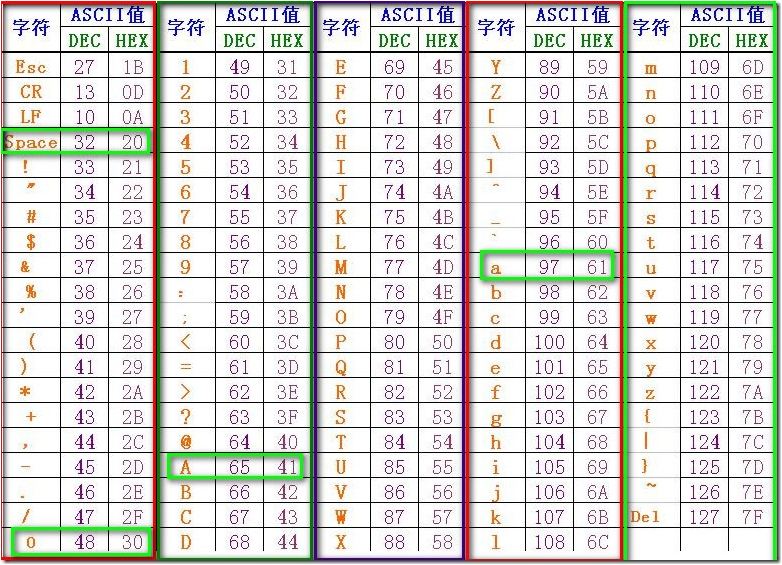
Byte:一个字节，即8位，short:2个字节，16位，char同样；int和float占4个字节，32位；long和double占64位，8个字节，boolean较为特殊，只有true和false两个取值。类型之间转换遵从从小往大转进行运算（表示值的范围大小）byte,short,char是平级的。

1. 
2. TCP三次握手时确认码为初始序列号+1：可能原因是在TCP协议中有一个超时重传机制，如果选择加1，可以区分是重传还是原始数据。
3. HTTP的404页面可以避免因为死链接导致网站不收录，设置404返回页面，一方面可以防止搜索引擎的爬取陷入死胡同，并且可以提升自己的搜索收录率，另一方面可以提升用户体验，返回到重要页面进行浏览。可能原因：原始URL失效，文件目录或文件名称更改，导致原网页链接失效。
4. HTTP协议默认的端口号是80。默认不写，所以网站看不见端口号的就是因为默认80.
5. new Integer(123) 与 Integer.valueOf(123) 的区别在于：

new Integer(123) 每次都会新建一个对象；

Integer.valueOf(123) 会使用缓存池中的对象，多次调用会取得同一个对象的引用。Integer 缓存池的大小默认为 -128~127。

1. Java 9 之后，String 类的实现改用 byte 数组存储字符串，同时使用 coder 来标识使用了哪种编码。
2. ASCLL表



1. Substring的 结束索引是不包括 的

SUBSTRING，substring(int beginIndex, int endIndex)

**beginIndex** -- 起始索引（包括）, 索引从 0 开始。

**endIndex** -- 结束索引（不包括）。

1. 线程创建的三种方法
2. 继承thread类，覆写run方法 无返回值
3. 实现runnable接口，创建新线程传入
4. 线程池创建
5. 创建Callable接口的实现类，并实现call()方法，该call()方法将作为线程执行体，并且有返回值。使用futureTask 类实例化FutureTask <String>futureTask=new FutureTask<>(callable);
6. 实现接口的方式会更好一些，因为JAVA不支持多重继承，
7. JAVA几种锁机制？他们的区别？如何使用？

两种，一个是synchronized，另一个是Reentrantlock

对于synchronized有三种同步方式，

同步一个代码块，作用于一个对象

public void func() {

synchronized (this) {

// ...

}

}

同步一个方法，作用于一个对象

public synchronized void func () {

// ...

}

同步一个类，作用于整个类，也就是说两个线程调用同一个类的不同对象上的这种同步语句，也会进行同步。

public void func() {

synchronized (SynchronizedExample.class) {

// ...

}

}

同步一个静态方法，作用于整个类

public synchronized static void fun() {

// ...

}

对于Reentrantlock，是 java.util.concurrent（J.U.C）包中的锁。

private Lock lock = new ReentrantLock();

public void func() {

lock.lock();

try {

for (int i = 0; i < 10; i++) {

System.out.print(i + " ");

}

} finally {

lock.unlock(); // 确保释放锁，从而避免发生死锁。

}

}

两种锁的区别：第一synchronized是 JVM实现的， Reentrantlock是JDK实现的，两者性能大致相同；第二Reentrantlock等待可中断（当持有锁的线程长期不释放锁的时候，正在等待的线程可以选择放弃等待），而synchronized不行；第三，synchronized中的锁是非公平的，而Reentrantlock默认情况下也是非公平的，但也可以是公平的；第四，reentrantlock可以同时绑定多个condition对象。使用选择：除非使用reentrantlock的高机功能，否则优先使用synchronized。因为synchronized是JVM实现的，原生支持，而Reentrantlock不是所有的JDK版本都支持，并且使用synchronized不用担心没有释放锁而导致死锁问题。

1. 线程Wait和sleep的区别，wait是OBJECT的方法，而sleep是Thread的静态方法；wait是释放锁，而sleep不会。
2. JUC AQS

CountDownLatch ： 用来控制一个或者多个线程等待多个线程。维护了一个计数器CNT，每次调用countdown方法会使计数器的值减1，减到0时，因为调用await方法而等待的线程就会被唤醒。

CyclicBarrier ： 用来控制多个线程互相等待，只有当多个线程都到达时，这些线程才会继续执行。（一组队伍等人一起打副本）

CyclicBarrier 和 CountdownLatch 的一个区别是，CyclicBarrier 的计数器通过调用 reset() 方法可以循环使用，所以它才叫做循环屏障。

CyclicBarrier 有两个构造函数，其中 parties 指示计数器的初始值，barrierAction 在所有线程都到达屏障的时候会执行一次。

Semaphore 类似于操作系统中的信号量，可以控制对互斥资源的访问线程数。有acquire();和release();

1. 内存模型三大特性：原子性、可见性、有序性
2. 不可变的对象是线程安全的，不可变的类型有 final关键字修饰的基本数据类型；string；枚举类型；Number部分子类，如long，double等数值包装类型；对于集合类型，使用Collections.unmodifiableXXX()方法来获取一个不可变的集合。
3. **原码**：第一位为符号位，0表示正数，1表示负数，剩余位表示数值

5的原码：00000000 00000000 00000000 00000101  
-5的原码:10000000 00000000 00000000 00000101

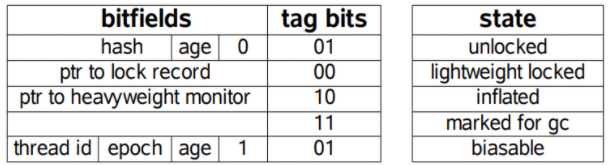
**反码**：正数原码即其反码，负数符号位不动，剩余的数字取反。

5的反码：00000000 00000000 00000000 00000101  
-5的反码：11111111 11111111 11111111 11111010

**补码**：正数的原码即其补码，负数的补码是其反码加1

5的补码：00000000 00000000 00000000 00000101  
-5的补码：11111111 11111111 11111111 11111011

**Java中所有基本数据类型均使用该数字的补码进行表示.**

1. 锁优化：**1.自旋锁**：因为互斥同步进入阻塞状态的开销很大，应该尽量避免，有时候共享数据的锁定状态只会持续很短的一段时间，自旋锁就是让一个线程再请求一个共享数据的锁时进入忙循环一段时间，如果这段时间内获得锁，就可以避免进入阻塞状态；**2.锁消除：**锁消除是指对于被检测出不可能存在竞争的共享数据进行消除；主要通过逃逸分析来支持，如果堆上的共享数据不可能逃逸出去呗其他线程访问到，就可以把他们当成私有数据对待。**3.锁粗化：**如果一系列的连续操作都对同一个对象反复加锁和解锁，频繁的加锁操作会导致性能损耗，如果虚拟机探测到由这样的一串零碎操作对同一个对象加锁，会把加锁的范围扩展（粗化）到整个操作序列的外部。**4.轻量级锁：**偏向锁和轻量级锁，使得锁有了四个状态：无锁，偏向锁、轻量级锁、重量级锁。轻量级锁使用CAS操作来避免重量级锁使用互斥量的开销。**5.偏向锁：**偏向于让第一个获取锁对象的线程，这个线程再之后获取该锁就不再需要进行同步操作，甚至连CAS操作也不再需要。
2. **注解：**它提供了一种安全的类似注释的机制，用来将任何的信息s或元数据（metadata）与程序元素（类、方法、成员变量等）进行关联。是附加在代码中的一些元信息，用于一些工具在编译、运行时进行解析和使用，起到说明、配置的功能。注解不会也不能影响代码的实际逻辑，仅仅起到辅助性的作用。 用处：  1、生成文档。这是最常见的，也是java 最早提供的注解。常用的有@param @return 等  
         2、跟踪代码依赖性，实现替代配置文件功能。比如Dagger 2 依赖注入，未来java 开发，将大量注解配置，具有很大用处;  
         3、在编译时进行格式检查。如@override 放在方法前，如果你这个方法并不是覆盖了超类方法，则编译时就能检查出。四种元注解

@Documented – 注解是否将包含在JavaDoc中  
   @Retention – 什么时候使用该注解  
   @Target – 注解用于什么地方  
   @Inherited – 是否允许子类继承该注解

1. **反射：获得class对象又三种方法**：1.类名.Class。例如String.Class；2。Class.forname()方法；3.实例.getClass()方法

用 instanceof 关键字来判断是否为某个类的实例。同时我们也可以借助反射中 Class 对象的 isInstance() 方法来判断是否为某个类的实例

**创建实例有两种方法**：1.使用Class对象的newInstance()方法来创建Class对象对应类的实例。2. 先通过Class对象获取指定的Constructor对象，再调用Constructor对象的newInstance()方法来创建实例。这种方法可以用指定的构造器构造类的实例。

**获取方法**: Class<?> c = methodClass.class;

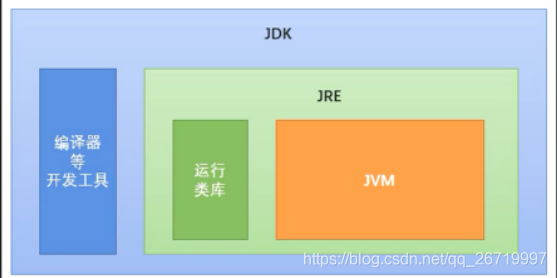
Object object = c.newInstance();

Method[] methods = c.getMethods();

Method[] declaredMethods = c.getDeclaredMethods();//ALL

//获取methodClass类的add方法

Method method = c.getMethod("add", int.class, int.class);

1. 
2. 动态获取资源路径的方法：classloader,获取某个类的src文件位置，注意路径中有空格可能会报错，因为转为字符时，空格转为了%20，造成错误

ClassLoader classLoader=jdbcutil.**class**.getClassLoader();  
URL res = classLoader.getResource(**"jdbcpro.properties"**);  
String path = res.getPath();

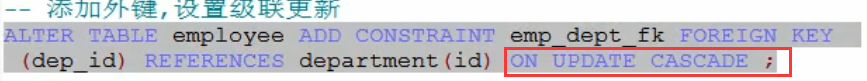
1. 解决SQL注入问题（通过SQL语法导致登录成功），使用preparedStatement解决。

String sql=**"select** *\** **from user where username = ? and password = ?"**;  
pst=conn.prepareStatement(sql);  
pst.setString(1,username);  
pst.setString(2,pwd);  
rs=pst.executeQuery();以后也基本都会使用pstmt，这个效率更高且安全

1. Hashmap的底层是数组加链表实现
2. SQL四种约束：主键约束，非空约束，唯一约束，外键约束

约束在创建表的时候添加约束，或者创建表完后，添加约束：ALTER TABLE stu MODIFY NAME VARCHAR（20） NOT NULL。对于主键可以提娜佳 AUTO\_INCREMENT来自增长主键

外键约束添加：创建表时，各名称添加好后，

开始写constrant 外键名称 FOREIGN KEY(此表中该外键列名称) REFERENCES 外键表（外键表的主键名）设置级联更新：

37.类加载机制（JVM）：把描述类的数据从Class文件加载到内存，并对数据进行**校验**、转换**解析**和**初始化**，最终形成可以被虚拟机直接使用的Java类型。个生命周期包括：加载（Loading）、验证（Verification）、准（Preparation）、解析（Resolution）、初始化（Initialization）、使用（Using）和卸载（Unloading）7个阶段。其中验证、准备、解析3个部分统称为连接（Linking）

类加载时机：

1. 隐式加载 new 创建类的实例,
2. 显式加载：loaderClass,forName等
3. 访问类的静态变量，或者为静态变量赋值
4. 调用类的静态方法
5. 使用反射方式创建某个类或者接口对象的Class对象。
6. 初始化某个类的子类
7. 直接使用java.exe命令来运行某个主类

根据类的完全限定名查找字节码文件，创建一个Class,验证class的字节流是否符合虚拟机的规范，然后分配内存空间，给该类变量进行改变量的初始值，然后解析将常量池的符号引用替换成直接引用，最后对变量进行初始化。

双亲委派：如果一个类收到了类加载的请求，它并不会自己先去加载，而是把这个请求委托给父类加载器去执行，如果父类加载器还存在父类加载器，则进一步向上委托，依次递归，请求最后到达顶层的启动类加载器，如果弗雷能够完成类的加载任务，就会成功返回，倘若父类加载器无法完成任务，子类加载器才会尝试自己去加载，这就是双亲委派模式。

Java容器，主要有collection存储对象的集合和Map存储键值对，collection中set，list,queue。Map中hashmap，treemap，hashtable. ArrayList 基于动态数组实现，LinkedList 基于双向链表实现。ArrayList 和 LinkedList 的区别可以归结为数组和链表的区别：

数组支持随机访问，但插入删除的代价很高，需要移动大量元素；

链表不支持随机访问，但插入删除只需要改变指针。

39.Arraylist和vector的比较：Vector 是同步的，因此开销就比 ArrayList 要大，访问速度更慢。最好使用 ArrayList 而不是 Vector，因为同步操作完全可以由程序员自己来控制；

Vector 每次扩容请求其大小的 2 倍（也可以通过构造函数设置增长的容量），而 ArrayList 是 1.5 倍。

40. 作为线程安全的array list的替代方案：CopyOnWriteArrayList 写操作需要加锁，防止并发写入时导致写入数据丢失。写操作结束之后需要把原始数组指向新的复制数组。

在写操作的同时允许读操作，大大提高了读操作的性能，因此很适合读多写少的应用场景。

41. hashmap与 Hashtable 的比较:

Hashtable 使用 synchronized 来进行同步。

HashMap 可以插入键为 null 的 Entry。

HashMap 的迭代器是 fail-fast 迭代器。

HashMap 不能保证随着时间的推移 Map 中的元素次序是不变的。

42.LRU缓存：以下是使用 LinkedHashMap 实现的一个 LRU 缓存：

设定最大缓存空间 MAX\_ENTRIES 为 3；

使用 LinkedHashMap 的构造函数将 accessOrder 设置为 true，开启 LRU 顺序；

覆盖 removeEldestEntry() 方法实现，在节点多于 MAX\_ENTRIES 就会将最近最久未使用的数据移除。

43.获得hashmap的键集合可以使用.entryset方法，保存为一个SET《entry< , >》形式。

44.红黑树的插入删除和遍历的时间复杂度都为logN，性能上低于哈希表，但是哈希表无法提供键值对的有序输出，红黑树因为排序插入的，可以按照键的值的大小有序输出。

每个节点可以是黑色或者红色，根节点必须为黑色了，相邻节点不能都是红色，任一节点到其子树中每个叶节点的路径都包含相同数量的黑色节点。

45.死锁：不同线程互相占用需要的同步资源不释放，都在等待对方释放锁，导致他们都进入阻塞的现象。

死锁的原因：主要是因为独占锁的特点和循环等待条件，独占锁是说，一个资源每次只能被一个线程使用，每个线程请求资源而陷入阻塞时，对他已经获得的资源不释放。或者线程已经获得的资源在没有使用完之前不允许被剥夺。 循环等待条件时说在若干线程之前形成了一种头尾相接的循环等待资源关系。

避免死锁：：在并发程序中，避免逻辑中出现复数个线程互相持有对方线程所需要的独占锁的情况，就可以避免死锁。

46.工厂模式：创建一个对象时不向客户暴露内部细节，并提供一个创建对象的通用接口

47. 