Java SE

1. 全局搜索关键字VSCode

Edit->find in files

1. 正则表达式：描述符合某些复杂规则的字符串的代码
2. JIT：运行时编译

支持动态链接，编译占用运行时资源

AOT：运行前编译

避免运行时的编译性能消耗和内存消耗，加快程序启动

使程序安装的时间增加，牺牲Java的一致性

1. &按位与

&&短路运算

1. 自动装箱

Integer a = 3; //将3自动装箱成Integer类型

1. String s = “constant”;//s是常量池中”constant”的引用

String s2 = new String(“constant”);//s2无法在编译期确定，是运行时创建的新对象” constant”的引用

s2.intern();//查找常量池，没有则新增一个

1. Array可以包含基本类型和对象类型，大小固定

ArrayList只能包含对象类型，大小动态变化

8． 值传递：对基本数据类型而言，传递的是变量的一个副本，改变副本不影响原变量

引用传递：对于对象型变量而言，传递的是对象地址的一个副本

9. 4.0-3.6=0.4000001？二进制的小数无法精确地表达十进制小数，计算机在计算十进制小数的过程中要先转换为二进制进行运算，这个过程中出现了误差

10. 十进制的数在内存中以二进制补码的形式存储，最高位是符号位

正数的反码是其本身

负数的反码是在其原码的基础上, 符号位不变，其余各个位取反.

正数的补码就是其本身

负数的补码是在其原码的基础上, 符号位不变, 其余各位取反, 最后+1. (即在反码的基础上+1)

11. Lambda表达式：允许把函数作为一个方法的参数

// 5. 接受一个 string 对象,并在控制台打印,不返回任何值(看起来像是返回void)

(String s) -> System.out.print(s)

12. hashcode的主要作用是用于查找的快捷性（哈希算法，将数据依特定算法直接指定到一个地址上，当集合要添加新元素时，先调用这个元素的hashcode方法，就能定位到它的物理位置）

1）如果两个对象相同，equals返回true

2）两个对象的hashcode 相同，并不一定表示两个对象就相同，只能说明这两个对象在一个散列存储结构中

、

13. Map四个实现类

①HashMap:根据键的hashcode值存储数据，根据键可以直接获取它的值，遍历时，取得数据的顺序是随机的，遍历顺序与容量有关,不支持线程同步，最多只允许一条记录的键为NULL，允许多条记录的值为NULL

②HashTable：键和值不能为空,支持线程同步（即任一时刻只能有一个线程写入HashTable）

③LinkedHashMap

④TreeMap：能把它保存的值根据键排序

14. 保证任一时刻最多只有一个线程执行该段代码

Synchronized是关键字：发生异常时，会自动释放线程占有的锁，因此不会导致死锁的发生

Lock是接口：发生异常时，得通过在finally块中用unLock()释放锁

15. volatile关键**字：被volatile修饰的变量**保证每个线程能够获取该变量的最新值

16．重载（Overloading）发生在同一个类里面多个方法的方法名相同但是参数不同的情况

无法以返回型类别作为重载函数的区分标准

覆盖（overriding）子类重新定义父类的方法，可重新定义访问限制

17.

抽象：从多个事物中将共性的，本质的内容抽象出来，忽略一个主题中与当前目标无关的那些方面，以便更充分地注意与当前目标有关的方面

抽象类：可以包含抽象和非抽象方法

接口：特殊的抽象类，接口更强调功能

继承：新类继承原始类的特性

封装：把过程和数据包围起来成为一个对象，这些对象通过一个受保护的接口访问其他对象

多态

18. Comparator

19. 反射：允许运行中的Java程序获取自身的信息，并且可以操作类或对象的内部属性

所有类在加载后，JVM会为其在堆中创建一个Class<类名称>的对象，并且每个类只会有一个Class对象，这个类的所有对象都要通过Class<类名称>来进行实例化。

可通过Class.newInstance()方法创建Class对象对应类的实例（调用的类要有无参构造函数）

Class对象的getDeclaredMethods()方法返回类或接口声明的所有方法、

getMethods()方法返回某个类的所有公用（public）方法

getDeclaredConstructors()返回构造函数

通过method1.invoke()方法调用method1()

20. static静态修饰符

静态：被static修饰的成员在编译后所分配的内存会一直存在，直到程序退出内存才会释放这个空间

不依赖类特定的实例，被类的所有实例共享；static对象可以在它的任何对象创建之前访问，无须引用任何对象

static final 修饰的常量 ->全局常量、

修饰的容器类型变量->所装的值可变

static静态代码块在项目启动的时候就执行，自动执行

静态方法里只能直接调用同类中其他的静态成员，而不能直接访问类中的非静态成员，因为非静态方法和变量只有在创建完类的实例对象后才可使用

静态方法内部无法使用this,super，因为静态方法不属于某个实例对象

21. 上界<? extend Fruit> ，表示所有继承Fruit的子类（上界是fruit）

编译器支持向上转型

22. 泛型：“参数化”类型，将类型由原来的具体的类型参数化，在使用/调用时传入具体的类型  
23. StringBuffer比StringBuilder线程安全，多了个Synchronized关键字

24. Object

-clone()//创建并返回此对象的一个副本

-equals()//比较两个对象是否相等

-finalize()//当垃圾回收器确定不存在对该对象的等多引用时，由对象的垃圾回收器调用此方法

-getClass()//返回一个对象的运行时类

-hashcode()//返回该对象的哈希码

-wait()//导致当前的线程等待

-wait(long timeout, int nanos) 导致当前的线程等待，并释放所持有的对象的锁，直到其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者其他某个线程中断当前线程，或者已超过某个实际时间量

-notify()//唤醒在此对象监视器上等待的单个线程

-notifyAll()//唤醒在此对象监视器上等待的所有线程，让线程竞争，只有获得锁的线程才能进入就绪状态

25. 类是对某一类事物的描述，是抽象的

对象是实实在在的个体，是类的一个实例

26. String在JDK中被声明为final类(final char[])，所以不可变

为什么HashMap中的键往往使用String：不可变，hashcode一致性，处理速度快，安全，多线程安全

27. List Set Map

List可以有重复元素，Set不能有重复元素，Map键值对是一对一或多对一的关系

List以特定索引来存取元素

Set和Map容器都有基于哈希存储和排序树两种实现版本

基于哈希存储存取时间复杂度为O(1)，而基于排序树的实现在插入或删除元素会按照元素的键构成排序树从而达到排序和去重的效果平均时间复杂度为O(logN)

28. ArrayList和Vector使用数组方式存储数据，数组元素数大于实际存储的数据以便增加和插入元素，插入元素时需要重新计算大小和更新索引，Vector底层实现多了Synchronized关键字，线程安全但效率较低,ArrayList在并发操作下可能出现数组越界问题

LinkedList使用双向链表实现存储，每个元素都和它前一个元素和后一个元素链接在一起，非线程安全，存取时间复杂度为O(n)

29．Collection是集合类的上层接口，List、Set等实现了这个接口

Collections提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作

30. Iterator提供了统一遍历操作集合元素的统一接口，Collection接口实现Iterator接口

遍历时可通过remove()方法删除元素

Iterator可用来遍历List和Set，只能前向遍历

ListIterator既可以前向也可以后向遍历，还可以增加元素，替换元素，获取前一个和后一个元素的索引

31. 实现Serializable接口，可以把对象存到字节流，进行网络传输，然后可以恢复，在分布式应用中，得实现序列化

32. ConcurrentHashMap类内部使用Segment来充当锁的角色

33. TreeMap是一个有序的键值对集合，基于红黑树（自平衡的排序二叉树）实现 O(logn)

34. 如何保证线程安全

通过合理的时间调度，避开共享资源的存取冲突  
35. 新建、阻塞、就绪、运行、死亡

36. 线程池

事先创建若干个可执行的线程放入一个容器池中，需要的时候从池中获取线程而不用自行创建，使用完毕不需要销毁线程而是放回池中，从而减少创建和销毁线程对象的开销，提高响应速度，提高线程的可管理性

37. 同步和异步

数据在线程间共享，多个线程对同一数据的读写，存取策略

同步阻塞式 异步非阻塞（不想让程序等待方法的返回）

38. sleep()//不考虑线程优先级，使线程进入阻塞状态，要捕捉InterruptedException异常

yield()只会给相同优先级或更高优先级的线程以运行的机会，使线程进入就绪状态

39. 多线程实现方式

继承Thread类（重写run()，start()启动）、

实现Runnable接口（重写run()，作为Thread对象的参数传入，使用实现Runnable接口的方式创建的线程可以处理同一资源，也就是实现Runnable的实例，从而实现资源的共享.）、

创建线程池

同步实现方法

Synchronized关键字，wait(),notify( )方法

40. thread.start()方法使线程进入可运行状态，并不代表线程立刻进入运行状态，得由JVM调度执行

41. 监视器实现Java线程同步，每一个监视器都和一个对象引用相关联，线程在获取锁之前不允许执行同步代码

42. i++不是原子性操作。i++分为读取i值，对i值加一，再赋值给i++，执行期中任何一步都是有可能被其他线程抢占的

43. 死锁：两个或两个以上的线程都在等待对方执行完毕才能继续往下执行，进入无限的等待的情况（如线程1锁住了资源A，然后尝试对B加锁，但是线程2已经锁住了B，还尝试对A加锁，就发生阻塞情况）

44. Synchronized既可以加在方法上，也可以加在特定代码块上，托管给JVM执行

Lock需要显式地指定起始位置和终止位置

45. 反射通过.class字节码文件找到某一个类

Class：类的对象 Constructor：构造方法 Field：类的属性对象 Method：方法对象

46. ClassLoader和它的子类把class文件从磁盘读取到内存中（加载到JVM）

类的加载是动态的

47. 一个进程可以启动多条线程，各个线程可以共享进程的资源，又可以独立由CPU调度

48. Java虚拟机的ROOT对象

虚拟机栈中的引用对象

方法区中类静态属性引用的对象

方法区中常量引用对象

本地方法栈中JNI引用对象

49. 虚拟机将堆分为新生代和老生代

新生代：Eden、From Survivor、To Survivor（8：1：1）

50. GC采用有向图的方式记录和管理堆中的所有对象，通过这种方式确定哪些对象是“可达的”，哪些对象是“不可达”的

51. Java中的异常和错误都继承于Throwable（子类Exception、Error）

unchecked exception：非受检异常，包括运行时异常和Error及其子类

checked exception

运行时异常（RuntimeException）

ArithmeticException、ClassCastException、NullPointerException

52. 字节流：InputStream、OutputStream

字符流：InputStreamReader、OutputStreamReader

53. 异常处理机制：当Java程序违反了Java的语义规则时，Java虚拟机就会将发生的错误表示为一个异常；一种是Java内置的语义检查，如IndexOutOfBoundsException,另一种是用户自定义的异常，并用throw关键字抛出异常

54.

<<      :     左移运算符，num << 1,相当于num乘以2

>>      :     右移运算符，num >> 1,相当于num除以2

55.

实现Serializable接口的类，通常加一个serialVersionUID

相当于一个独特的指纹身份

JVM会把传来的字节流中的serialVersionUID与本地相应实体（类）的serialVersionUID进行比较

若不加serialVersionUID(系统会自动分配一个)，则修改原来的类之后再序列化会出错

56.

使用ThreadLocal创建的变量只能被当前线程访问，其他线程则无法访问和修改

private static ThreadLocal<UserInfoVO> loginEntityThreadLocal=new ThreadLocal<>();

private static ThreadLocal<UserInfoVO> loginEntityThreadLocal = new ThreadLocal<>();

loginEntityThreadLocal.get();

loginEntityThreadLocal.set(entity);

loginEntityThreadLocal.remove();

避免同步锁，保证读写高效(多个UserInfoVO)

57.

反射的用处

获取Class对象

obj.getClass()

Class.forName(接口名或类名)

Class.getDeclaredField

预先处理未编写的代码，如 web.xml

使得程序更“动态”

Junit用来寻找@Test标签的方法,在单元测试的时候再调用这些方法

58.

//Arrays

public static <T> List<T> asList(T... a) {

        return new ArrayList<>(a);

    }

可变参数…表示可以接受多个参数值

//Arrays.aslist(“a”,”b”,”c”);

参数不能是原始数据类型，会被当成一个数组对象

这里的ArrayList是Arrays的内部类，并不是util包里的类

private final E[] a;

静态数组

59.用transient关键字标记的成员变量不参与序列化过程。没必要持久化这些字段，节省空间保护敏感数据

ArrayList中的elementData被transient修饰，同时类中实现了readObject和writeObject方法，避免序列化数组中没有元素的下标；同时避免并发带来的ConcurrentModificationException(Vector中能保证同步操作，序列化实现不一样)

60.

多线程安全：一个方法或类对象可以被多个线程同时使用而不发生任何问题

当一个共享变量被volatile修饰时，它会保证修改的值会立即被更新到主存，当有其他线程需要读取时，它会去内存中读取新值。保证了可见性

Java EE

1. Spring自动装配的方式

手动配置Bean的依赖关系

byName

byType

Constructor

autodetect

1. IoC（Inversion of Control）控制反转，通过容器（BeanFactory和它的子接口ApplicationContext）来实现对象生命周期的控制并管理对象之间的依赖关系，降低程序的耦合

DI（Dependency Injection）依赖注入，是对IoC更简单的诠释，容器将某种依赖关系注入到组件之中

1. 装配Bean

@Autowired byType自动装配对象

@Resource(name=”xxDao”) byname

1. 横向散布在所有对象层次中，与对象的核心功能毫无关系的代码称为横向，不利于各模块的重用

AOP将那些影响了多个类的公共行为封装到一个可重用模块，即切面，减少系统的重复代码，降低模块之间的耦合度

1. 事务
   1. 原子性：事务是最小的执行单位，不允许分割。确保动作要么全部完成，要么都不完成
   2. 一致性：执行事务前后，数据保持一致
   3. 隔离性：访问数据库时，各并发事务之间数据库是独立的
   4. 持久性：一个事务提交后，它对数据库中数据的改变是持久的
2. SpringMVC

DispatcherServlet收到请求，转发到请求对应的Handler,Handler返回ModelAndView给DispatcherServlet利用ViewResolver进行解析渲染

1. Servlet生命周期
   1. init()方法初始化（单实例，减少开销）
   2. service()方法处理客户端的请求
   3. destroy()方法终止
   4. JVM回收
2. forward：一次请求，保留request的数据,url不变

redirect(302):两次请求

1. JNDI（Java Nameing & Directory Interface）

Java命名目录服务

1. Web Service

一个应用程序，向外界暴露出一个能够通过Web进行调用的API