API(一)

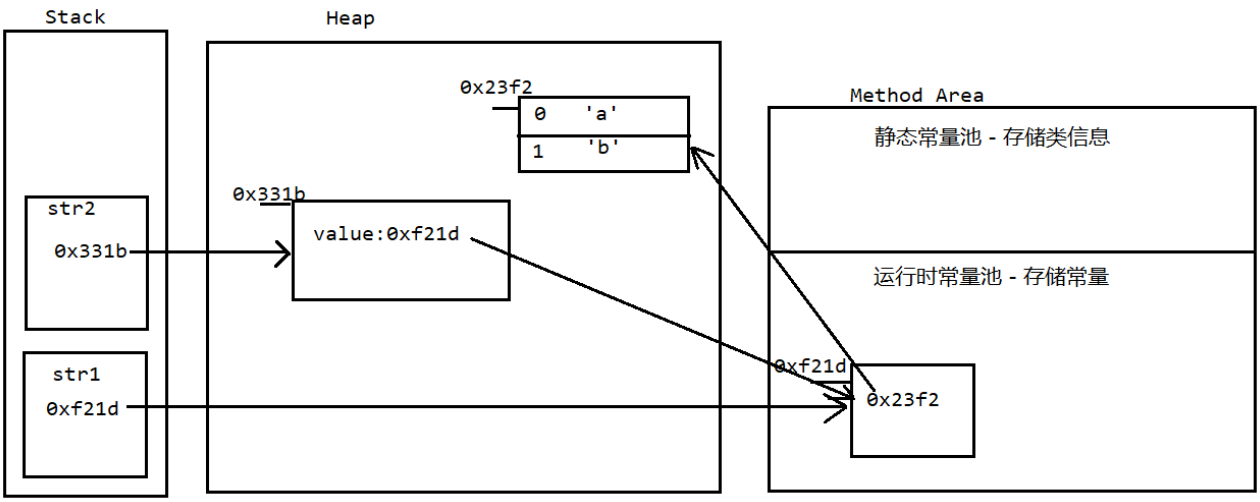
一、Object

1. Object是Java中的顶级父类，在Java中任何一个类都默认继承了Object
2. 在Java中任何一个类都是Object的子类
3. 重要方法

方法	解释
clone	1.表示创建一个和当前对象属性值一样的对象 2.这个方法在使用的时候，要求对象对应的类必须实现接口Cloneable。这个接口中没有任何的方法和属性，仅仅标记这个类的对象可以被克隆
finalize	通知GC进行垃圾回收。仅仅是起通知作用，而GC不一定启动
getClass	获取对象的实际类型
hashCode	1.获取对象的哈希码。根据哈西散列算法产生对象的哈希码 2.同一个对象的哈希码是一样的，不同对象的哈希码是不同的
toString	当直接打印一个对象的时候，底层实际上在调用这个对象的toString方法
equals	1.判断两个对象是否是同一个对象 2.equals默认是比较两个对象的地址是否一致。实际过程中往往需要重写equals方法 3.重写equals要拆分成4步： a.比较地址是否一致 b.判断参数是否为空 c.比较类型是否一致 d.判断属性值是否一样

二、String

1. String类是一个最终类
2. 字符串在Java底层是依靠字符数组来存储，字符数组是一个常量不可变
3. 字符串的内存：



4. String str = "a"; 这句话只产生了一个对象
5. String str = new String("a"); 这句话产生了2个对象
6. String str = "a"; String str2 = new String("a"); 这两句话产生了2个对象

7. 因为+在底层是调用StringBuilder中append，所以如果需要进行大量的字符串拼接的时候，建议直接使用StringBuilder
8. String类中提供了大量的操作字符串但是不改变原字符串的方法

三、正则表达式

1. 针对字符串进行操作，利用指定的规则对字符串进行匹配、筛选、替换等操作
2. 符号：

正则	解释
[xyz]	x、y、z中的任何要给都可以
[^abc]	表示除了a/b/c以外的元素
.	表示任意字符
\\\\	匹配\
\\.	匹配.
\\w	单词字符，等价于[a-zA-Z0-9_]
\\W	非单词字符
\\d	数字，等价于[0-9]
\\D	非数字
\\s	空白字符
\\S	非空白字符
+	至少一次
?	至多一次
*	零次或者多次
{n}	恰好出现n次
{n,}	至少出现n次
{n,m}	至少出现n次，但是不超过m次

3. 当()将一些字符放在一组的时候，构成了正则表达式中的捕获组
4. 捕获组的编号是从1开始递增，可以通过\\n的形式来引用对应编号的捕获组
5. 捕获组的编号计算是从(出现的顺序来依次递增

四、包装类

1. 在Java中，基本类型的变量身上没有任何的方法和属性提供使用，如果需要操着这个基本类型，那么需要自定义过程来使用。因此Java针对每种基本类型都提供了对应的类形式来便捷的操作数据。提供的类形式就称之为包装类
2. 包装类：

基本类型	byte	short	int	long	float	double	char	boolean
包装类	Byte	Short	Integer	Long	Float	Double	Character	Boolean

3. 将基本类型的变量直接赋值给引用类型的对象，这个过程称之为自动封箱。自动封箱在底层会调用对应类身上的valueOf方法

4. 将引用类型的对象直接赋值给基本类型的变量，这个过程称之为自动拆箱。自动拆箱在底层会调用对应对象身上的***Value方法

五、数字运算

1. BigDecimal是一个能够对小数进行精确运算的类。在计算的时候要求参数以字符串形式传入
2. BigInteger：能存储和计算超大整数
3. Math：针对基本类型提供了初等数学运算 - 指数、对数、幂、三角函数等

六、日期

1. Date类：这个类是属于java.util包下
   a. 如果不指定，默认获取的是当前系统的时间
   b. SimpleDateFormat负责在字符串和日期之间来进行转化的，在转化的时候需要指定格式
2. Calender类：这个类是属于java.util包下的，是JDK1.2推出来用于取代Date类，但是实际开发中依然会使用Date

格里高利历

```
java.util.GregorianCalendar[time=  从1970-01-01 00:00:00到指定时间的毫秒值
1577256697509,areFieldsSet=true,areAllFieldsSet=true,lenient=true,z
one=sun.util.calendar.ZoneInfo[id="Asia/Shanghai",offset=  时区偏移量
28800000,dstSavings=0,useDaylight=false,transitions=
19,lastRule=null],firstDayOfWeek=1,minimalDaysInFirstWeek=1,ERA=
1,YEAR=2019,MONTH=11,WEEK_OF_YEAR=52,WEEK_OF_MONTH=4,DAY_OF_MONTH=
25,DAY_OF_YEAR=359,DAY_OF_WEEK=4,DAY_OF_WEEK_IN_MONTH=4,AM_PM=
1,HOUR=2,HOUR_OF_DAY=14,MINUTE=51,SECOND=37,MILLISECOND=
509,ZONE_OFFSET=28800000,DST_OFFSET=0]
```

API(二)

一、异常

1. 异常是Java中提供的一套用于问题的反馈和处理的机制

```
public class Demo {

    public static void main(String[] args) throws FileNotExistException {
        try{
            String msg = readTxt("G:\\a.txt");
        } catch (FileNotExistException e){
            // 处理问题
        }
    }

    public static String readTxt(String path) throws FileNotExistException {
        //  路径不存在
        if(判断路径是否存在)
            // 将问题包装成一个异常对象来抛出
            throw new FileNotExistException();
        return "读取到的内容";
    }
}
```

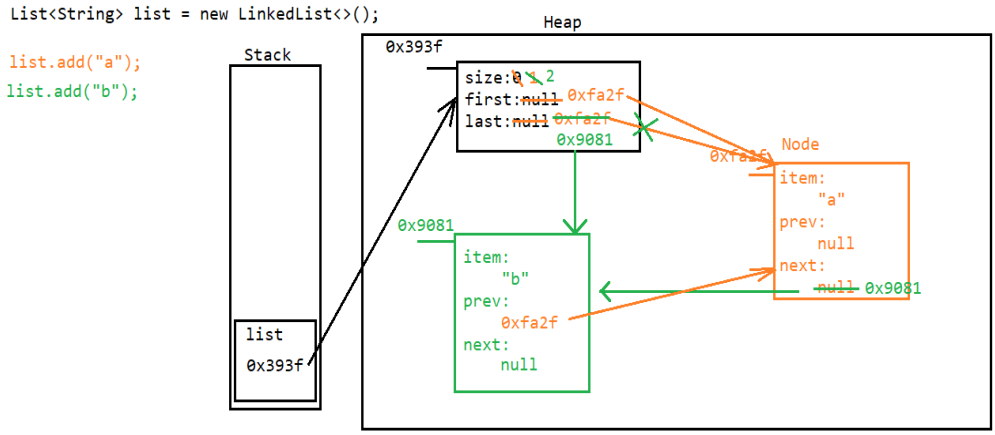
2. Java中异常的体系结构：
   a. 异常的顶级父类是Throwable，包含2个子类：Error和Exception
   b. Error：错误 - 表示在一个**合理的应用程序**中出现的**不应该捕获**的严重问题 - Error出现出现无法处理 - StackOverflowError
   c. Exception：异常。出现之后可以处理，处理方式分为两种：throws抛出或者try-catch捕获
      i. 编译时异常（已检查异常）：在编译阶段就已经出现，要求必须立即处理
      ii. 运行时异常（未检查异常）：在编译时不报错而是在运行阶段出现， 可以处理可以不处理。在Java中，所有的运行时异常都需要继承RuntimeException
3. 如果需要的异常在Java中没有提供，那么就需要自定义异常。如果需要自定义异常，那么需要写一个类继承Exception或者是其子类
4. 如果程序中抛出了异常，那么异常之后的代码就不再执行。但是异常被捕获之后，catch之后的程序可以继续执行
5. 异常的处理方式：
   a. 如果每一个异常的处理方式都不一样，那么可以分别catch分别处理
   b. 如果所有异常的处理方式都一样，那么可以捕获一个异常的父类，进行统一处理
   c. 如果异常需要分组处理，那么同一组的异常之间可以用|隔开 - 这种方式是JDK1.7提供的方式
6. 异常不影响方法的重载，但是在重写的时候，子类异常不能超过父类异常
7. try-catch-finally中的finally的特点是：无论出现异常与否，这个finally中的代码块都会执行一次

二、集合

1. 集合是Java中提供的一套容器机制，这个容器的特点是大小不固定
2. Collection是集合的顶级接口，包含了很多的子接口以及实现类：List、Set、Queue等
3. <E>表示泛型，用于规定集合中元素的类型。因为泛型的限定，所以集合中存储的类型必须是引用类型:
   Collection<Integer>，Collection<Double>等
   int[] arr; --- arr的数据类型是数组，元素类型是int
   Collection<String> c; --- c的数据类型是Collection，元素类型是String
4. List：列表
   a. 元素有序（指的是保证元素的添加顺序），允许添加重复元素
   b. 提供了下标来获取元素

c. 实现类：ArrayList底层是基于数组的，默认初始容量为10。每次扩容默认是在上一次容量的基础上增加一半的容量，也就意味着初始容量是10，那么扩容之后的容量就是15，再次扩容之后的容量就是22 - 用数组来模拟一个ArrayList - 线程不安全的列表 - ArrayList是便于查询而不便于增删

d. 实现类：LinkedList底层是基于节点的，没有初始容量，也就不需要考虑扩容 - 线程不安全的列表 - LinedList是便于增删而不便于查询



e. Vector：向量。类似于ArrayList，底层也是基于数组来存储，但是每次扩容的时候，默认是增加一倍 - Vector是一个线程安全的集合

f. Stack：栈。继承了Vector，本身是一个后进先出/先进后出的结构。

入栈/压栈：将元素放入栈中

出栈/弹栈：将元素从栈中取出

栈底元素：最先放入栈中的元素

栈顶元素：最后放入栈中的元素

5. Set：散列集合
   a. 特点：不保证元素的顺序，而且要求元素不重复
   b. HashSet：底层基于HashMap来存储
   c. TreeSet：在底层会对元素进行排序，默认是升序排序。要求存储在TreeSet中的元素对应的类必须实现Comparable接口
6. Queue：队列，遵循先进先出的原则

三、和集合相关的类

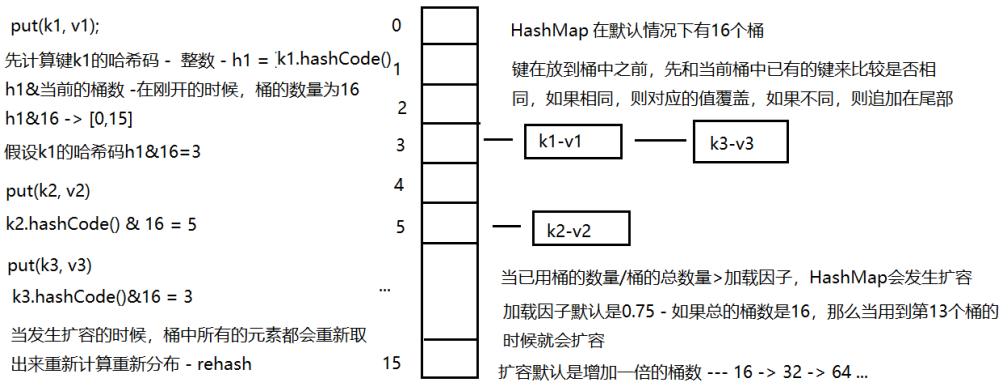
1. Iterator：迭代器。实际上是利用迭代的方式去遍历一个集合。迭代器在迭代过程中不允许直接增删原集合，而是需要通过迭代器提供的remove方法移除。增强for循环本质上是一个简化版的迭代器，增强for循环底层也是利用迭代器来迭代遍历的。要求能够被迭代的元素对应的类必须实现Iterable接口
2. Collections：集合的工具类，提供了大量的操作集合的方法

四、泛型

1. 泛型的学名是参数化类型 - ParameterizedType
2. 在JDK1.5之前，没有泛型的概念，从JDK1.5开始，出现了泛型
3. 泛型的擦除：用具体类型来替换泛型的过程。泛型的擦除发生在了编译期间
4. 如果需要定义一个泛型类，那么只需要在类名之后添加<泛型名>就可以声明一个泛型
5. 通常情况下，只使用一个大写字母来给泛型命名，最常用的大写字母：T - type，E - Element，K - key，V - Value，R - Result
6. 在Java中，也允许给方法来单独定义一个泛型，此时这个方法称之为泛型方法
7. ？ extends 类/接口，表示传入这个类/接口及其子类/子接口对象，此时称之为泛型的上限
8. ？ super 类/接口，表示传入这个类/接口及其父类/父接口对象，此时称之为泛型的下限
9. 上限和下限不能同时存在，即？ extends XXX super XXX这种格式是不允许的

五、Map<K,V>

1. Map - 映射：需要由2组元素构成，一组元素称之为键，另一组元素称之为值
2. 在Map中，每一个键必须对应一个值，把这种结构称之为键值对，即一个Map由多个键值对组成。在映射中，键是唯一的
3. 遍历映射：
   a. 方式一：可以先获取映射中所有的键，然后遍历键，根据键获取值
   b. 方式二：在Map中，将键值对封装成Entry对象，即每一个键值对实际上就是一个Entry，可以考虑直接获取所有的Entry
4. Map不是集合，但是Map是Java集合框架的成员 - Java Collections Framework（Java集合框架），包含了数组、集合、映射以及相关的工具类，例如：Collection及其来子类，Map及其子类，Arrays，Collections，Iterator，Comparable，Comparator等
5. HashMap - 哈希映射：
   a. 底层是基于数组+链表结构来存储数据。数组默认初始容量是16，数组的每一个位置都称之为是一个桶(bucket)，每一个桶中维系一个链表
   b. 默认加载因子是0.75，当已用桶数/总的桶数>0.75，会发生扩容。扩容默认是增加一倍的桶数
   c. HashMap本身是一个异步线程不安全的映射



6. Hashtable：
   a. Hashtable是Java中最早的映射之一
   b. 底层是基于数组+链表结构来存储，默认初始容量是11，默认加载因子是0.75
   c. Hashtable是一个同步线程安全的映射

API(三)

一、File

1. File是Java中提供的一套用于表示文件/目录(文件夹)的类
2. 删除目录及其子目录和子文件 -> 获取这个目录下所有的子文件和子目录(listFiles)，如果是子文件可以直接删除，如果是子目录，那么得获取这个子目录中的子文件和子目录，后续功能和当前的功能是一致的 - 递归
3. File提供了大量的操作文件/目录的方法

二、IO

1. IO -> IO流 - Input/Output Stream - 输入输出流
   a. 输入流：数据从外部流向程序。例如读取文件 - 将数据从文件读取到程序中
   b. 输出流：数据从程序流向外部。例如向文件中写数据 - 将数据从程序写到文件中
2. IO是Java中提供的一套用于进行数据传输的机制
3. 流的分类
   a. 按照传输方向：输入流和输出流
   b. 按照传输形式：字节流和字符流

	输入流	输出流
字符流	Reader	Writer
字节流	InputStream	OutputStream

4. 数据的来源/目的地：磁盘、内存、网络、输入设备
5. 向TXT文件中写入字符串 - 输出流、字符流、与文件相关的流 - FileWriter
6. 流中的异常处理：
   a. 流对象需要放在try之外定义并且赋予null值，然后放到try之内进行初始化
   b. 无论流写入数据成功与否，都需要关流，所以close操作需要放到finally里面
   c. 关流之前需要判断，判断流是否初始化成功，实际上就是判断流对象是否为空
   d. 因为关流也存在失败的可能，所以需要最后将writer置为null，那么无论如何这个writer都会被回收掉
7. JDK1.7中，提供了try-with-resources方法来处理流中的异常，但是要求try()中的对象对应的类必须实现Closable接口或者是Closable的子接口AutoClosable；如果在方法中接受到了一个流对象处理，那么需要再定义一个变量单独接收这个流
8. FileReader：读取字符文件，结合FileWriter实现文件复制的效果
9. 缓冲流：
   a. BufferedReader：提供了缓冲区，允许在读取数据的时候实现按行读取的效果 - 在实际过程中，一行实际上就是一个自然段
   b. BufferedWriter：本身比其他的字符输出流提供了更大的缓冲区
10. 文件的字节流：
    a. FileOutputStream：文件字节输出流，以字节的形式将数据写出到文件中 - 字节流没有缓冲区
    b. FileInputStream：文件字节输入流，以字节的形式将数据从文件中读取出来
11. 凡是字符流可以完成的操作，字节流都可以完成，只是传输的时候，字符流传输字符文件会比字节流要快一些
12. 字符串流 - 用于读写字符串
    a. StringReader：从内存中将字符串读取来
    b. StringWriter：将字符串写到内存中
13. 字节数组流 - 用于读写字节数组

a. ByteArrayOutputStream：产生字节数组，将这个字节数组放到内存中
b. ByteArrayInputStream：将字节数组从内存中读取出来

14. 转换流：字节流和字符流之间进行转换的桥梁，在这个过程中注意编码的转换问题
    a. OutputStreamWriter：将字符转化为字节，在底层将数据以字节形式来写出
    b. InputStreamReader：将字节转化为字符，在底层将数据以字节形式来读出来
15. 系统流：
    a. 本质上是System类中提供的静态常量，作用是从控制台来获取数据或者将数据打印到控制台上
    b. 分类

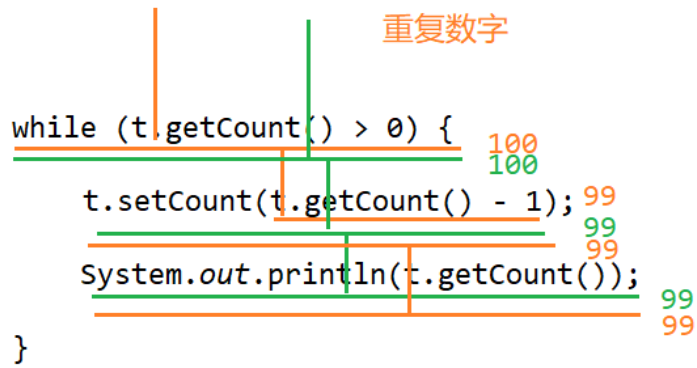
分类	说明
System.in	标准输入流
System.out	标准输出流
System.err	标准错误流

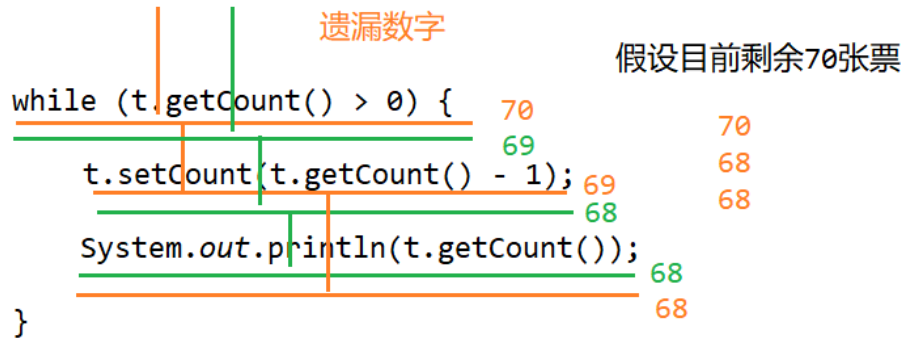
    c. 系统流全部都是字节流
    d. 系统流用完之后不需要关闭
16. 打印流 - PrintStream/PrintWriter：
    a. 是Java提供的一套能够进行便捷打印的机制，并且提供了便捷的换行方式
    b. 标准输出流/标准错误流本质上都是打印流
    c. 打印流只有输出流
17. 合并流 - SequenceInputStream：
    a. 只有输入流没有输出
    b. 合并多个字节流，将这多个字节流中的数据合并到一块，实际过程中作用往往是将多个文件合并成一个文件。在创建合并流的时候，需要获取一个Enumeration对象，所以往往是利用Vector先来存储多个字节流，然后再利用这个Vector生成一个Enumeration对象，最后再构建合并流
18. 序列化/反序列化流：
    a. 序列化：将数据按照指定格式进行转化
    b. 反序列化：将数据按照指定格式转化回来
    c. 注意问题：
        i. 如果一个对象想要被序列化，那么这个对象对应的类必须实现Serializable。这个接口中没有任何的方法和属性，仅仅是用于标记当前类的对象可以被序列化
        ii. 被static、transient修饰的属性不会被序列化出去
        iii. 当一个类实现Serializable接口之后，这个类中如果没有指定serialVersionUID(版本号)，那么在运行的时候会根据类中的属性和方法自动计算一个版本号。如果类中的属性或者方法发生改变，那么就会自动重新计算一个新的版本号。在序列化的时候，版本号会随着对象一起被序列化出去。在反序列化的时候，会先校验版本号是否一致，如果一致才能反序列化
    d. 实际过程中，序列化和反序列化会被大量的应用，只要涉及到数据的传输和存储的问题，都会考虑序列化
19. 其他：
    a. RandomAccessFile：用于操作文件的类，在操作文件的时候，将文件看作是一个大型的字节数组，可以根据下标操作文件中指定的位置
    b. Properties：
        i. Properties是一个可持久化(持久化是一种的特殊的序列化，将数据序列化到磁盘上就称之为持久化)的映射，继承了Hashtable
        ii. Properties中的键和值默认是String

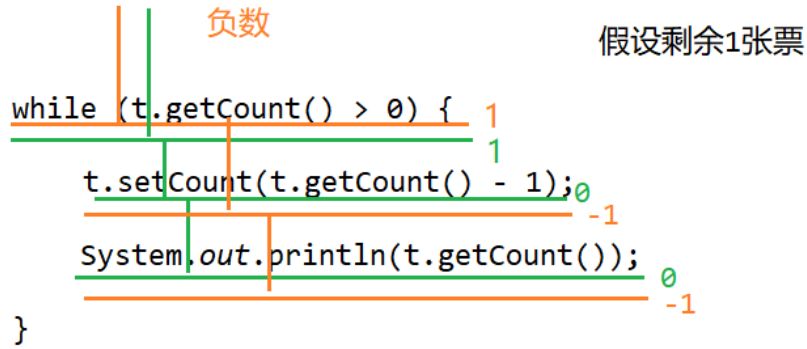
iii. Properties要求文件的后缀必须是.properties

三、线程

1. 进程和线程：
   a. 进程是计算机在执行的任务
   b. 线程是轻量级的进程。线程可以认为是进程的子任务。如果进程中只有一个任务，那么这个进程就可以看作是线程了。一个进程中至少包含1个线程，可以包含多个线程 - 线程实际上也是计算机在执行的任务
2. 多线程：计算机执行多个任务
3. 定义线程：
   a. 继承Thread类，然后覆盖run方法，将线程要执行的逻辑写到run方法中，然后调用start方法来启动线程
   b. 实现Runnable接口，然后覆盖run方法，在启动之前需要将Runnable对象传递到Thread中，然后利用Thread调用start方法来启动执行 - 现阶段要求掌握
   c. 实现Callable接口，覆盖call方法，需要利用执行器服务来启动 - 现阶段仅作了解，后续课程中会讲解
4. 多线程的并发安全问题：多个线程同时执行的时候，这多个线程之间是相互抢占执行，并且抢占会发生在线程执行的每一个步骤中。由于线程之间的抢占导致出现不合理的数据的现象称之为多线程的并发安全

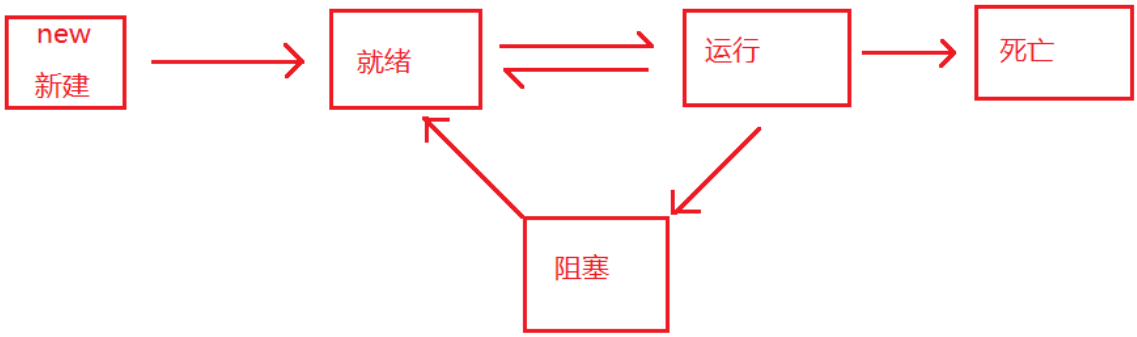






5. 并发：多个线程同时执行，这种方式称之为并发
6. synchronized：利用一个锁将一段代码锁起来，一个线程使用期间不允许其他线程使用 - 需要确定一个锁对象，锁对象要求能被所有的线程都认识：**共享资源**、方法区中的资源(类的字节码、字符串等)、this

7. 被synchronized括起来的这段代码称之为同步代码块
8. 同步和异步
    a. 同步：如果一段代码在一个时间段内只能被一个线程使用
    b. 异步：如果一段代码在一个时间段内能够被多个线程抢占使用
9. 线程安全和线程不安全：
    a. 线程安全：多个线程同时执行的时候，结果是正常的
    b. 线程不安全：多个线程同时执行的时候，结果不符合常理
10. 死锁：锁之间相互嵌套导致代码无法继续向下执行。死锁形成条件：锁形成嵌套，线程之间还需要相互的锁。死锁不是逻辑错误或者代码错误，所以死锁一旦出现无法处理。实际过程中，一般会尽量避免死锁，那么这个时候就会根据业务要求检测死锁，如果产生死锁，会考虑打破一个锁来释放其他的锁
11. 虽然锁可以有效的避免多线程并发产生的安全问题，但是实际过程中尽量减少使用锁机制。锁机制如果滥用会导致死锁的产生，会导致程序执行效率的降低。如果实在需要使用锁，那么只将有危险的片段锁起来就可以，不需要将整个类或者整个方法锁起来
12. 等待唤醒机制：wait/notify/notifyAll通过线程等待让线程不再执行而是让其他线程执行，通过这种方式调节了线程之间的执行顺序。注意，在notify的时候，是随机唤醒一个线程
13. wait和sleep区别
    a. sleep：在使用的时候需要指定睡眠时间，到了指定时间之后会自动醒过来；如果没有锁，那么线程在沉睡期间释放执行权；如果有锁，那么线程在沉睡期间不会释放执行权。sleep是被设计在Thread类上，本身还是一个静态方法
    b. wait：在使用的时候可以指定等待时间也可以不指定，如果不指定等待时间那么需要唤醒。wait必须结合锁来使用，线程在wait期间会释放执行权。wait是设计在了Object类上
14. 线程的状态：



    a. 线程在运行之前一定是先就绪
    b. 运行状态有可能转化为阻塞或者就绪状态
    c. wait、sleep等方法是将线程转化为阻塞状态
15. 线程创建的场景：
    a. 用户主动创建 - 打开微信，实际上就是主动申请创建微信运行的这个线程
    b. 系统自动启动 - 例如开机的时候，操作系统中会自己来运行一些系统的线程
    c. 关联启动
16. 线程销毁的场景：
    a. 线程自然结束 - "寿终正寝"
    b. 被其他线程给强制关闭 - "他杀"
    c. 线程在执行过程中出现以外而结束 - "意外死亡"
17. 线程的优先级：
    a. 在线程中，有优先级的区分，优先级用数字1-10表示

b. 数字越大优先级越高，理论上能抢占到资源的概率就会越大，但是注意在实际生产过程中，优先级的差别并不明显。至少要差到5个优先级及以上，区别才会稍微明显一点点
c. 如果没有设置优先级，那么线程的优先级默认为5

18. 守护线程
    a. 守护线程用于守护其他线程的，当被守护的线程结束，那么守护线程无论执行完成与否，都会随之结束
    b. 一个线程要么是守护线程要么是非守护线程
    c. 如果存在多个线程，那么只要这个线程不是守护线程，那么这个线程就是被守护。例如一共有10个线程，其中有7个守护线程，其他3个就是被守护的线程。如果存在多个被守护线程，那么守护线程要以最后一个被守护的线程的结束作为结束信号

四、网络编程

1. 网络编程实际上是一套基于网络进行数据传输的流
2. 网络模型：物理层、数据链路层、网络层、**传输层**、会话层、表示层、应用层
3. 在这七层中，提供了不同的传输协议，例如：http、FTP、POP3等这些协议都属于后三层的协议，**UDP、TCP**等协议都是传输层的协议，而Java中的网络编程也是要考虑传输层的协议
4. IP地址：在网络中用于标记主机的，在传输数据的时候需要指定IP。IP协议存在IPv4和IPv6。目前国内最常使用的是IPv4，现在国内也开始推行IPv6。IPv4是用了4组数字表示一个IP地址，例如10.35.62.189。这四组数字中的每一组数字的范围不能超过255，也就意味着IPv4的地址范围是0.0.0.0~255.255.255.255，但是这个范围的一部分IP是有特殊含义的
5. 端口：是主机和外界进行信息交互的媒介 - 如果一台电脑需要和外界进行信息的交互，那么必须通过端口来进行交互。一台主机包含了好多的端口，对端口进行编号，编号的范围是0~65535。在信息交互的时候，需要指定端口的。但是注意，实际开发中，0~1024这个范围的大部分端口被计算机内部占用，往往是从1025号端口开始使用
6. 域名：为了方便人们记忆网站地址，设计了这么一串字符串。这串字符串在网络底层要映射为IP。例如：www.baidu.com - 在实际使用过程中，域名是分级的。一级域名往往是标记域名的类型，例如com表示商业类型，org表示的机构或者组织，edu表示的教育类型；二级域名往往是公司的商标或者是品牌，例如baidu.com表示百度，163.com表示网易等；三级域名往往是应用类型，例如news.baidu.commail表示百度新闻，163.com表示的网易邮箱。在实际使用过程中，域名一般不超过三级
7. DNS解析服务器：就是将域名和IP进行了映射。在万维网上存在一个最大的DNS服务器。当在浏览器地址栏指定域名之后，在底层，DNS服务器就会将这个域名解析为IP，然后才会去访问对应的IP地址。如果当前DNS服务器无法解析这个域名，向高一层的DNS服务器来请求解析域名
8. localhost或者127.0.0.1表示本机，255.255.255.255表示的是一个广播地址(只要在这个局域网范围内的主机都会接收到这个数据)
9. InetSocketAddress：是Java提供一个用于表示IP和端口和类
10. UDP：
    a. 特点：
        i. 基于网络进行**数据传输**的流
        ii. 无连接的流，不可靠(无论有没有人接收，都会发送出去，所以可能会产生数据丢失)但是传输速度快
        iii. 适用于对传输速度依赖性较高但是对可靠性依赖相对低一些的场景，例如直播、视频聊天、语音电话等
    b. 发送端：
        i. 创建UDP发送端对象
        ii. 将数据封包，并且还需要指定接收地址和端口号
        iii. 发送数据包

iv. 关流

c. 接收端:

i. 创建UDP接收端对象，绑定监听的端口号

ii. 准备数据包

iii. 接收数据

iv. 关流

v. 解析数据

d. 案例：单人聊天室

11. TCP：

a. 特点：

i. 基于网络进行数据传输的流

ii. 有连接，经历三次握手(三次确认)，可靠(一定是在建立连接之后才会发送数据，三次确认过程中只要任何一次失败都不会发送数据，所以不会产生数据的丢失)，传输速度比较慢

iii. 适用于对可靠性依赖比较高但是对传输速度依赖相对低一些的场景，例如文件上传下载

b. 在现在的开发中，往往是TCP和UDP混用：在网络条件好的情况下，尽量选用TCP；如果网络条件产生变化，切换为UDP

c. 客户端：

i. 创建客户端套接字对象 - Socket

ii. 发起连接，绑定连接地址和端口号

iii. 获取输出流，写出数据

iv. 通知服务器端数据已经写完

v. 关流

d. 服务器端

i. 创建服务器端套接字对象 - ServerSocket

ii. 绑定监听的端口号

iii. 接收连接

iv. 获取输入流，读取数据

v. 通知客户端数据已经读完

vi. 关流

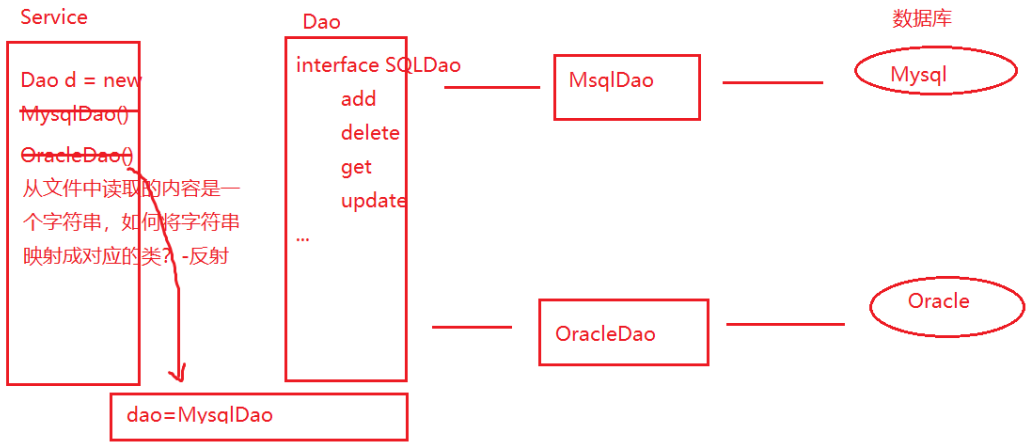
e. 案例：文件上传

五、反射

1. 反射是去剖析一个类，获取这个类的字节码对象以及实例对象
2. 如果将字节码看作对象，对这一类对象进行总结，总结出代表字节码的类 - Class
3. 基本类：

a. **Class - 代表字节码的类 - 代表类的类**
b. Field - 代表属性的类
c. Method - 代表方法的类
d. Constructor - 代表构造方法的类
e. Package - 代表包的类
f. Annotation - 代表注解的类

4. 反射在实际开发中，可以结合多态来实现解耦。如果两个或者多个模块之间产生了死死的绑定，称之为耦合。



5. 获取Class对象：
   a. 通过类名.class方式来获取指定类的字节码对象
   b. 通过对象.getClass()方式来获取对象对应的实际类型的字节码对象
   c. 通过Class.forName("全路径名")方式来获取指定类型的字节码
6. 修饰符：

修饰符	十进制	十六进制
public	1	0x00000001
private	2	0x00000002
protected	4	0x00000004
static	8	0x00000008
final	16	0x00000010
synchronized	32	0x00000020
volatile	64	0x00000040
transient	128	0x00000080
native	256	0x00000100
interface	512	0x00000200
abstract	1024	0x00000400
strictfp	2048	0x00000800

7. 案例：模拟一个clone方法 - 可以在不知道对象的类型的情况下就创建这个对象并且可以对属性进行赋值
8. 在后续学习过程中，会大量的应用反射，例如数据库的JDBC操作等 - 也正是因为有反射的存在，所以才能够更好的实现解耦过程，并且能够更好的分工合作
9. 在实际开发过程中，强调"高内聚低耦合" - 高内聚就是尽量提高自己的东西的利用率，低耦合就是尽量减少使用别人的东西

六、JDK1.5的特性

1. JDK1.5提供或者增强的主要特性：自动封箱拆箱、增强for循环、泛型、静态导入、可变参数、枚举、反射(进行了增强)、动态代理、内省、注解等
2. 静态导入：
   a. 在使用一个类中的静态方法的时候，可以在导包语句中单独导入这一个方法而不是导入整个类，这样了可以一定程度上提高加载效率
   b. 实际过程中这种方式很少使用，这种方式容易和已有的方法产生冲突，同时降低了可读性
3. 可变参数：
   a. 在调用方法的时候可以传入任意多个参数(可以传入0个参数，可以传入一个参数，也可以传入多个参数)，也可以传入一个数组
   b. 可变参数本质上就是一个数组，所以可以通过操作数组的方式来操作可变参数
   c. 可变在使用的时候只能定义一个，并且必须放在参数列表的末尾
4. 枚举：
   a. 用于表示选项固定并且能够一一列举的场景， 例如季节、等级、星期、月份等
   b. 枚举中，构造函数默认是私有的而且必须是私有的
   c. 枚举常量必须定义在枚举类的首行
   d. 在枚举类中可以定义任何的属性和方法
   e. 在Java中，枚举的顶级父类是java.lang.Enum
5. 注解：
   a. 注解可以理解为给机器看的解释说明性的文字
   b. 在Java中，注解的顶级父类是Annotation
   c. 通过@interface来定义注解
   d. 在注解中，通过定义方法的形式来定义属性，属性的类型只能是基本类型、String、枚举、Class、其他注解类型以及它们的一维数组形式
   e. 如果属性是数组，那么且调用的时候数组中只有1个值，那么在调用的时候可以省略{}
   f. 通过default来给属性定义默认值
   g. 如果一个注解中只有1个属性，并且这唯一的一个属性的名字是value，那么在调用这个属性的时候可以省略属性名不写
   h. 元注解：在之前接触过的绝大部分注解都是作用在类、方法或者属性上，但是元注解是作用在注解上的 - 元注解是对注解进行注解
      i. @Target：用于限制注解的使用范围
      ii. @Retention：用于限制注解的生命周期的
      iii. @Document：表示生成在文档中
      iv. @Inherited：表示当前注解要作用在子类上

七、其他

1. Debug模式/断点调试：允许通过断点来一行行调式代码
2. 单元测试/Junit测试：
   a. 要求测试方法必须在公共类
   b. 需要选中要测试的方法的方法名，如果不选中方法名，则测试的时候会将所有的方法都进行测试
   c. 要求被测试的方法必须没有参数，没有返回值以及是一个非静态方法