开发工具

Javac 编译工具

Java 运行工具

Jdb 调试工具

Jhat 内存分析工具

JDK？JRE?

JDK是java开发工具包

·jvm虚拟机=>程序运行得地方

·核心类库

·开发工具

JRE是java运行环境

JVM 核心类库 部分开发工具

JDK JRE JVM 三者关系

JDK包含了JRE包含了JVM

基本数据类型

Byte -128-127

Short

Int

Long

Float

Double

Char

Boolean

java内存分配

·栈 方法运行时使用的内存，方法进栈运行，运行完毕就出栈

·堆 new出来的，都在堆中开辟一个空间

·方法区 存储可以运行的class文件

·本地方法栈 jvm在使用操作系统功能时使用

·寄存器 给cpu使用

·串池Stringtable ---->字符串常量池 只有直接赋值才会存在串池中，在jdk7之前串池是存放在方法区中，从jdk7开始从方法区挪到堆中

·静态区 当加载一个类的class文件时，就会在堆中创建一个静态区，来存放静态变量

Jdk8之前是在方法区。Jdk8以后是在堆中

从jdk8开始取消方法区新增元空间。把原来方法去的多种功能进行拆分，有的功能放到了堆中，有的功能放到了元空间中，

对于继承来说，会在方法区中加载子类的class文件和父类的class文件以及Object类得class文件，会在堆中存放父类得成员变量，以及子类的成员变量，二者共用一块地址（会先加载object类的字节码文件，再加载父类的，在加载子类的）

一个对象得内存图

Student stu = new Student();

1. 加载class文件------------在方法区内存储
2. 申明局部变量stu在栈中
3. 在堆内存开辟一个新的空间（new）---->在堆里面会有所有得成员变量还有成员方法得地址值
4. 默认初始化--->根据成员变量类型初始化赋值（string=null，int=0）
5. 显示初始化--->根据定义成员变量时是否赋值来赋值
6. 构造方法初始化---->根据new student（） 调用空参或有参来给对象中的变量初始化赋值
7. 将堆中的地址值付给stu
8. 当方法执行结束，方法出栈，栈中地址会消失
9. 当堆中得地址没有对象指向，则也会变成垃圾
10. 给引用对象赋值null时，修改的是指向地址是null，而不是将堆中地址清空

This的本质：记录方法调用者的地址值，方法中用this指向的就是成员变量

成员变量和局部变量

成员变量存在堆中，随着对象创建而创建，对象消失而消失

局部变量存在栈中，随着调用方法存在而存在，调用方法消失而消失

字符串原理

stringjoiner可以指定分隔符号，开始符号和结束符号

将数组转换成字符串

StringJoiner sj = new StringJoiner(“,”,”[”,”]”);

拓展底层原理1：字符串存储的内存原理

·直接复制会复用字符串常量池中的

·new出来不会复用，而是在堆中开辟一个新的空间

拓展底层原理2：==比较的到底是什么

·比较基本数据类型的时候比较数据值

·比较引用类型的时候比较地址

拓展底层原理3：字符串拼接的底层原理

·拼接的时候没有变量，只有字符串的时候，会触发字符串的优化机制，在编译的时候就是 最优结果

·拼接的时候有变量，每加一个对象内存就会多两个对象，一个stringbuilder一个string，底层实现使用stringbuilder+toString来实现的

·jdk8字符串拼接的底层原理：预估字符串长度

拓展底层原理4：StringBuilder提高效率原理

·所有要拼接的内容都会往StringBuilder中放，不会创建很多无用的空间，节约内存

拓展底层原理5：StringBuilder源码分析

Stringbuilder默认容量16

添加abc实际上添加的是ASCII

Stringbuilder扩容：老容量\*2+2 第一次16\*2+2 第二次 （16\*2+2）\*2+2

如果扩容数据不满足数据长度，则扩充为数据的长度

Capacity()可以获取stringbuilder容量 length()可以获取长度

容量最大值是int的最大值

Static：

静态变量：

特点：

该类所有对象共享

不属于对象，属于类

静态变量加载优先于对象的创建，随着类的加载而加载

调用方式：

类名直接调用（推荐）

对象名调用

Static的注意事项

静态方法只能访问静态变量和静态方法

非静态方法可以访问静态变量和静态方法，也可以访问非静态变量和非静态方法

静态方法中没有this

继承

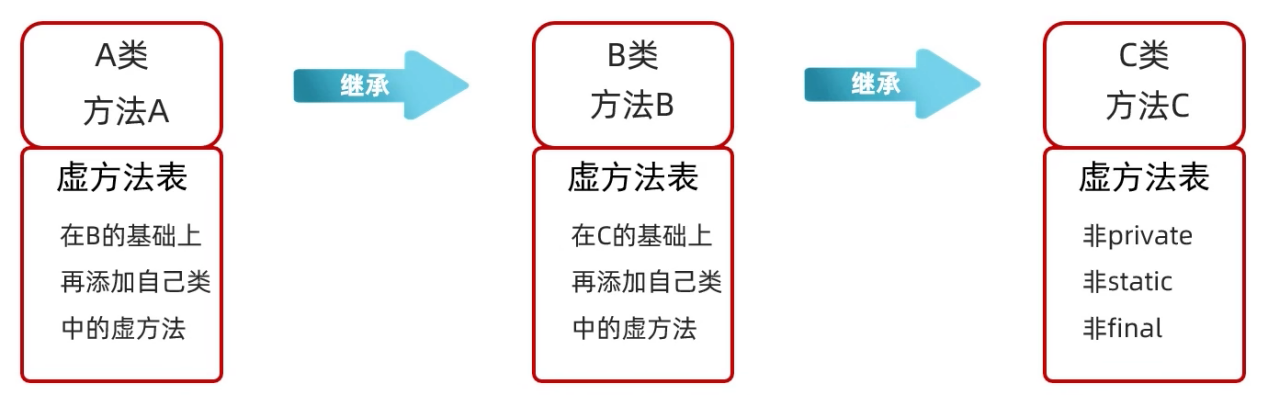
父类得方法：

通过子类调用父类的方法：会形成一个虚方法表，将常用的方法放在虚方法表中（=>

不能被private修饰

不能被static修饰

不能被final修饰）



重写

方法名和参数列表必须相同

访问修饰符必须大于父类得访问修饰符public>protected>不写

子类重写父类的方法时，返回值类型必须小于等于父类的返回值类型

私有方法不能被重写

静态方法不能被重写 重写是重写虚方法表里面的方法，这三种情况不会被加到虚方法表中

被final修饰得方法不能被重写

父类的构造方法不会被继承

在实例化子类的时候会先调用父类的无参构造，在执行自己的构造=>因为子类在初始化的时候可能会用到父类的数据，如果不调用父类的无参构造那么将无法使用父类的数据

调用父类的无参构造=>在子类的构造上super（），且必须在第一行，不写调用父类的无参jvm会默认添加

This使用进阶=>在空参构造中用this()调用有参构造，那么在空参中就不会有默认的super（），而是在用this调用的有参构造第一行，调用super（）

多态的弊端：

Fu fu = new Zi();

在子类中调用特有的方法时，编译报错

原因=>javac编译的时候会先检查等号左边也就是Fu中有没有方法

解决方案：

用类型强转成子类类型

被final修饰得基本数据类型=>值不可以被修改

被final修饰得引用数据类型=>指向的地址不可以发生改变，内容可以变

抽象类

抽象类不可以被实例化

抽象类不一定有抽象方法，有抽象方法得类一定是抽象类

可以有构造方法

JDK7

在接口中只能定义抽象方法

JDK8

在接口中可以定义静态方法默认方法

默认方法不是抽象方法，重写时需要把default去掉

Default默认方法default不能省略

如果一个实现类实现了多个接口，而且多个接口有同样的默认方法，那么实现类必须对默认方法重写

静态方法必须用接口名调用

JDK9

在接口中可以定义私有方法

**成员内部类**

内部类得访问特点：

内部类可以直接访问外部类得成员，包括私有的=>

①在内部类的方法中调用父类的成员变量且与内部类的成员变量和局部变量重名时，可以用外部类名.this.变量名

外部类如果访问内部类，必须创建对象

在成员内部类中不能定义静态变量，在JDK16以后版本才可以

获取内部类对象的方法

·在外部类中提供内部类的对象

·外部类名.内部类名 对象名= 外部类对象.内部类对象

※内部类得内存：

①内部类和外部类在方法区中加载字节码文件是相对独立的

②在堆内存中，实例化内部类

外部类.内部类 对象名 = new 外部类().new 内部类对象()

在堆内存中分别开辟一个外部类得内存和内部类得内存

而且还会在内部类的空间默认加一个this用来指向外部类在堆中得地址值

③内部类的对象实际上指向的是内部类在堆中的地址值

**静态内部类**

静态内部类只能访问外部类的静态变量或静态方法，如果想要访问外部类的非静态变量，那么需要创建外部类对象

创建静态内部类的方式

外部类.内部类 内部类对象= new 外部类.内部类（）；

调用静态内部类中静态方法

外部类.内部类.静态方法（）；

**局部内部类**

将内部类定义在方法内，就叫做局部内部类，类似于方法中的局部变量

外界无法使用，需要在方法中创建局部内部类的对象进行使用

局部内部类可以访问外部类的成员，也可以访问方法内的局部变量

**匿名内部类**

**·**隐藏了名字的内部类，可以写在成员位置，也可以写在局部位置

深拷贝/浅拷贝

浅拷贝=>object.clone

深拷贝=>用第三方插件gson

·Gson gson = new Gson();

String s = Gson.toJson(需要深拷贝的对象);

gson.fromJson(s,拷贝的类的字节码);

BigInteger

获取2的n次方-1随机数

BigInteger bd = new BigInteger(n,new Random());

获取指定的大整数a

BigInteger bd = new BigInteger(String a);

获取指定进制的整数=>n的x进制

BigInteger bd = new BigInteger(n,x);

静态方法获取bigInteger对象 内部有优化

BigInteger.valueOf(long类型);

这种方法比第二种方法取值范围小，只可以在long的取值范围内使用

在-16~16优化

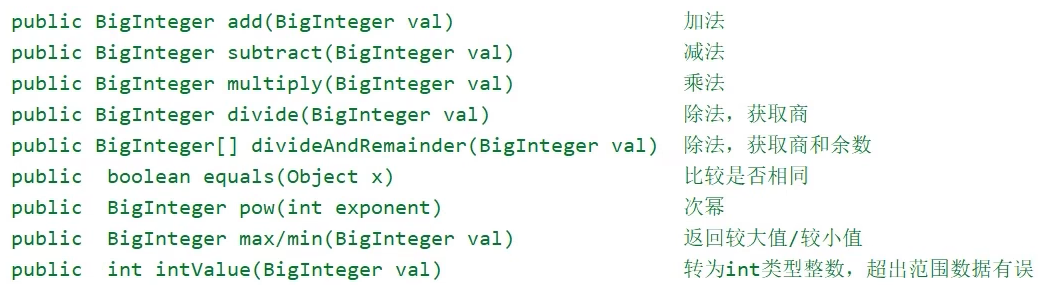
提前将-16~16对象创建好，如果多次调用不会创建新的

①如果biginteger表示的数字没有超出long的范围，可以使用静态方法valueof

②如果biginteger表示的超出long的范围，可以采用构造方法获取

③对象一旦创建，biginteger内部的值就不能发生改变

④只要进行计算都会产生一个新的biginteger对象



BigDecimal

构造器（double类型）可能是不精确的不建议使用

构造器（string类型）可以避免上述的问题

Divide方法（被除数，保留位数，Roundmode）

RoundMode=>

Up远离0的摄入

Down向零的方向舍入

Ceiling向正无穷大方向舍入

Floor向负无穷大舍入

Half-up 四舍五入

BigDicimal底层存储的是每一个字符的ASCII码

BigInteger底层存储的是整数位的二进制和小数位的二进制

BigDicimal的作用？

·表示较大的小数和解决小数运算的精度失真问题

BigDicimal的获取方式

·BigDicimal bd1 = new BigDicimal(double/string);

·BigDicimal bd2.valueof()

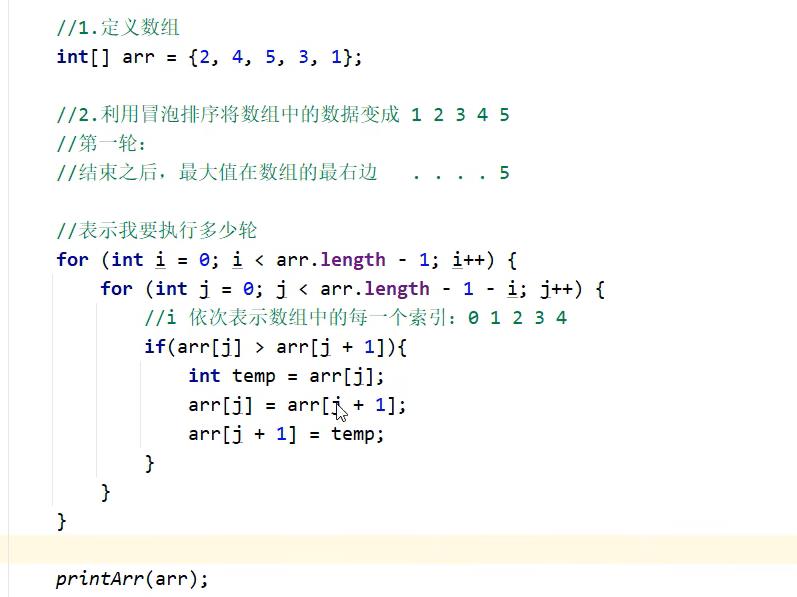
**※爬虫**

正则表达式

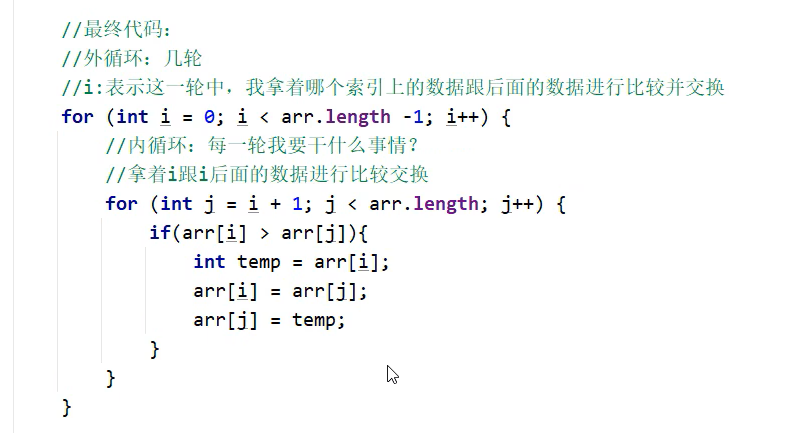
Pattern：java.util.regex包下的类=>获取正则表达式

Matcher:java.util.regex包下的类=>文本匹配器，作用按照正则表达式的规则去读取字符串，从头开始读取，在大串种去找符合匹配规则的子串。

冒泡排序



选择排序



Lambda表达式的标准规格

Lambda表达式是jdk8开始后的一种新的语法形式

注意点：

·lambda表达式可以用来简化匿名内部类的书写

·lambda表达式只能简化函数式接口的匿名内部类的写法

·函数式接口：

有且只有一个抽象方法的接口叫做函数式接口，接口上方可以加@FunctionalInterface注解

Lambda表达式的作用=>简化函数式接口的匿名内部类的写法

lambda的省略规则：

1. 参数类型可以省略不写
2. 如果只有一个参数，参数类型可以省略，同时（）也可以省略
3. 如果lambda表达式的方法体只有一行，那么大括号分号和return都可以不写，需要同时省略

Collection：

ArrayList:

List: 接口 LinkedList:

Vector:被淘汰了

HashSet：

Set: 接口 LinkedHashSet:是HashSet的子类

TreeSet:

List系列集合：添加的元素是有序、可重复、有索引的

Set系列集合：添加的元素是无序、不重复、无索引的

1. 添加元素

细节1：如果我们要往list系列的集合中添加元素，那么方法永远返回true，因为list系列是允许重复的

细节2：如果我们要往set系列的结合中添加元素，如果当前要添加的元素不存在，那么返回true，表示添加成功，如果当前要添加的元素存在，那么返回false，表示添加失败，因为set集合不允许重复

1. 删除

细节1：因为collection里面的方法是共性方法，所以不能通过索引进行删除。只能通过元素的对象进行删除

细节2：方法会返回一个布尔类型的返回值，删除成功true，失败false（删除的元素不存在就是删除失败）

1. 包含contains

细节：底层是依赖equals方法进行判断是否存在

所以，如果集合中存储的是自定义对象，也想通过contains方法来判断是否包含，那么在javaBean中一定要重写equals方法，

1. 清空clear

Collection遍历方式：

迭代器遍历：迭代器不依赖索引

1. 用集合的对象.iterator获取指针（获取迭代器对象）
2. hasNext判断是否有元素
3. next获取元素并且移动指针

细节：

1. 当指针指向最后一个hasNext的索引时，强行调用next方法，就会报NoSuchElementException异常
2. 迭代器遍历完毕，指针不会复位
3. 循环中只能用一次next方法
4. 迭代器遍历时，不能用集合的方法进行增加或者删除

增强for循环：底层就是迭代器

Lambda遍历：

Foreach循环 集合.foreach(s->{sout(s)});

List集合的特有方法

·collection的方法list都继承了

·list集合因为有索引，所以多了很多索引操作

Add(index,element)=>原来索引上的元素会依次往后移动

Remove(index)

Set(index,element)

Get(index)

遍历五种方式

Collection三种+for+列表迭代器

列表迭代器=>可以在遍历的过程中用列表迭代器中的add添加

五种方式的对比

迭代器：在遍历的过程中需要删除元素（使用迭代器中remove方法）

列表迭代器：在遍历的过程中需要添加元素（使用列表迭代器中获得add方法）

增强for/lambda:仅仅想遍历

普通for：在遍历的时候想操作索引

ArrayList底层

扩容机制：

* 1. ArrayList底层是数组结构，利用空参创建的集合，在底层创建一个默认长度为0的数组
  2. 添加第一个元素时，底层会创建一个新的长度为10的数组
  3. 存满时，会扩容1.5倍
  4. 如果一次添加多个元素，1.5倍还放不下，则新创建数组的长度以实际为主

LinkedList底层=>底层数据结构是双链表，查询慢，增删快，但是如果操作的时首尾元素，速度也是极快，所以linkedlist多了很多首尾操作的特有的api

成员变量：size，first，last

内部类node：E item, Node <E> next, Node<E>prev

迭代器源码分析

List.iterator=>底层就是new了一个ArrayList的内部类Itr()；

所以当我们调用iterator时，每调用一次就是创建一个itr的对象

Itr中的成员变量：int cursor =>光标，表示迭代器里面的指针，默认值0，所以默认指向0索引

Int lastRet=-1；表示上一次操作的索引

Int expectedModCount = modCount

ModCount：表示集合变化的次数，没add或remove一次，这个变量都会自增，当我们创建迭代器对象时，就会把这个数告诉迭代器

再next()方法中会调用checkForComodification()；方法=>会校验当前集合中最新的变化次数跟一开始的记录的次数是否相同，如果相同，证明当前集合没有发生改变，如果不一样，证明再迭代器遍历集合的过程中，使用了集合中的方法增加/删除了元素。

结论：如何避免并发修改异常

在使用迭代器或者增强for遍历集合的过程中，不要使用集合的方法去添加或删除元素，可以使用迭代器的remove方法，或者列表迭代器的add方法

泛型=>是jdk5以后引入的特性，可以在编译阶段约束操作的数据类型，并进行检查

泛型的格式=> <引用数据类型>

没有泛型的时候，集合如何存储数据=>

结论：

如果没有给集合指定泛型，默认认为所有的数据类型都是Object类型

此时可以往集合添加任意的数据类型

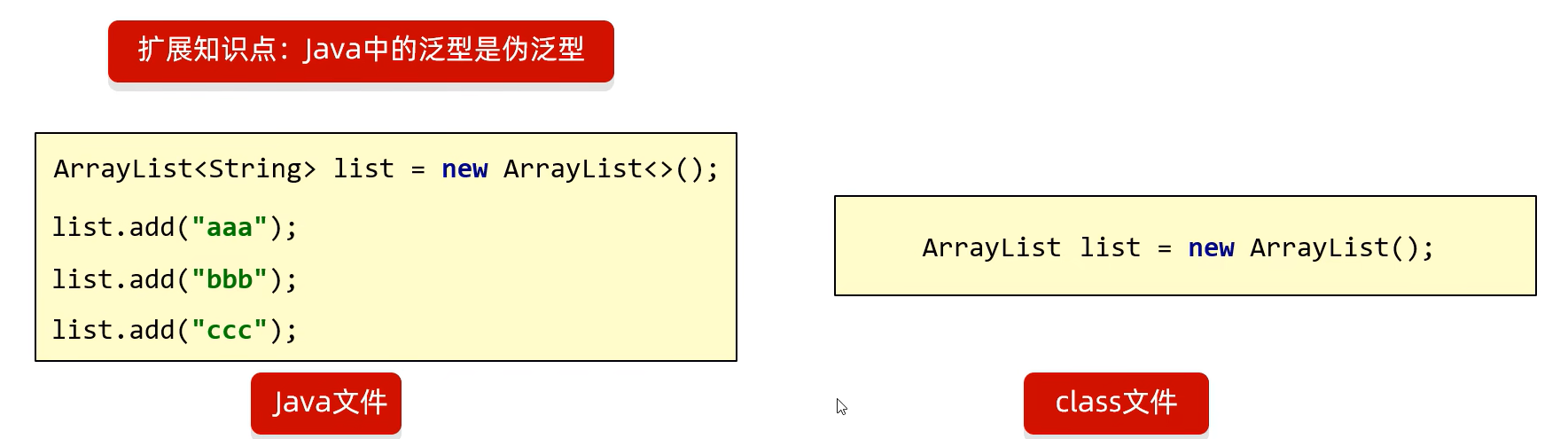
但是我们在获取数据时，无法使用他特有的行为，例Object类型的数据不能调用String类里面的方法，多态的弊端=>父类无法访问子类的方法

泛型的好处=>

1. 统一数据类型
2. 把运行时期的问题提前到了编译期间，避免了类型强转可能导致的异常。

拓展知识：java中的泛型是伪泛型

在编写.java文件时，约定添加进集合的类型，但是在编译成.class文件时，泛型就会消失 （泛型的擦除）

 set系列集合

·无序：存取顺序不一样

·不重复：可以去除重复

·无索引：没有带索引的方法，所以不能使用普通for遍历也不能通过索引来获取元素

Set集合实现类

·HashSet：无序、不重复、无索引

·LinkedHashSet：有序、不重复、无索引

·TreeSet：可排序、不重复、无索引

Set接口上的方法基本上与Collection的API一样

HashSet底层原理

·HashSet集合底层采用哈希表存储数据

·哈希表是一种对于增删改查数据性能都比较好的数据结构

哈希表的组成：

·jdk8之前：数组+链表

·jdk8以后：数组+链表+红黑树

哈希值：对象的整数值的表现形式

·根据hashCode方法算出来的int类型的证书

·该方法定义在object类中，所有对象都可以调用，默认使用地址值进行计算

·一般情况下，重写hashCode方法，利用对象内部的属性值计算哈希值

对象的哈希值特点

·如果没有重写hashCode方法，不同对象计算出的哈希值是不同的

·如果已经重写了hashCode方法，不同对象只要属性值相同，计算出的hash值也相等

·在小部分情况下，不同的属性值或者不同的地址值计算出来的哈希值也有可能是一样的（哈希碰撞）

HashSet底层实现

1. 创建一个默认长度16，默认加载因子0.75的数组，数组名table （加载因子=>当数组添加长度为当前长度的0.75倍，则扩容一倍，16X0.75=12，当数组存储了12个数据时，扩容）
2. 根据元素的哈希值跟数组的长度计算出应存入的位置

Int index = (数组长度-1)&哈希值

1. 判断当前元素是否为null，如果是null直接存入
2. 如果位置不为null，表示有元素，则调用equals方法比较属性值
3. 一样：舍弃、不添加 不一样：存入数组，形成链表

JDK8以前：新的元素存入数组，老元素挂载新元素下面

JDK8以后：新元素直接挂载老元素下面

1. JDK8以后，当链表长度大于8并且数组长度大于64时，链表自动转换为红黑树
2. 如果集合中存储的是自定义对象，必须重写hashCode和equals方法

LinkedHashSet底层原理

·有序、不重复、无索引

·这里的有序指的是保证存储和取出元素的顺序一致

·原理：底层数据结构依然是哈希表，只是每个元素又额外的多了一个双链表的机制记录存储的顺序

TreeSet底层原理

·不重复、无索引、可排序

·可排序：按照元素的默认规则（从大到小）排序

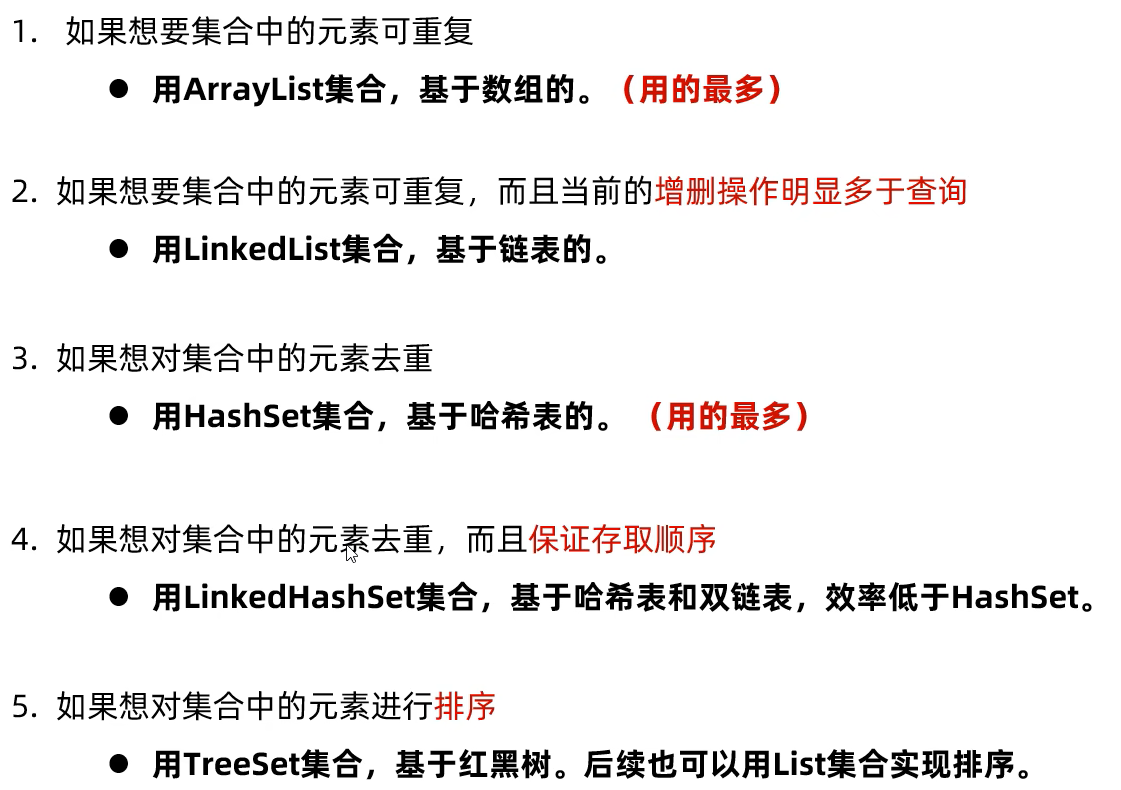
·TreeSet集合底层是基于红黑树的数据结构实现排序的，增删改查性能都很好

当TreeSet存的泛型是自定义类型时，需要声明排序规则

方式一：在JAVABEAN中实现Comparable接口，并重写抽象方法指定比较规则

重写方法中，如果返回值是0那么说明存在，就会被舍弃，在方法中this指向的是添加过来的元素，而参数表示的是已经存在红黑树里面的数据

方式二：创建TreeSet对象的时候，传递比较器comparator指定规则



**双列集合**

1. 双列集合一次需要存一对数据，分别是键和值
2. 键不能重复，值可以重复
3. 键和值是一一对应的，每个键只能找到自己的值
4. 键+值这个整体，我们称之为键值对或者键值对对象，或者entry对象

Map常见的API

**添加元素**

再添加数据的时候，如果键不存在，那么直接把键值对对象添加到map集合中，方法返回null

在添加数据的时候，如果键存在，那么会把原有的键值对对象覆盖，并将被覆盖的value返回

**删除元素**

Remove=>根据key来删除数据，返回value

Map得遍历方式

1. 键找值
2. Entry对象
3. Lambda=>底层就是增强for遍历entry对象

HashMap

1. HashMap是Map里面的一个实现类
2. 没有额外需要学习的特有方法，直接使用Map里面的方法就可以了
3. 特点都是由键决定的：无序、不重复、无索引
4. HashMap和HashSet底层原理一样，都是哈希表结构
5. 根据键得hash值来存放在hash表
6. 如果键表示的属性值一样，那么就会覆盖之前的值
7. 当数组长度大于64并且链表长度大于8的时候，会转换成红黑树
8. 依赖hashCode方法和equals方法来保证键得唯一
9. 如果键存储的是自定义对象，那么需要重写hashcode方法和equals方法

**LinkedHashMap**

1. HashMap的子类
2. 由键决定：有序、不重复、无索引
3. 这里的有序指的是保证存储和去除的元素顺序一致
4. 原理：底层数据结构依然是哈希表，只是每个键值对元素又额外的多一个双链表的机制记录存储的顺序

TreeMap

1. TreeMap和TreeSet底层原理一样，都是红黑树
2. 由键决定特性：不重复、无索引、可排序
3. 可排序：对键进行排序
4. 注意：默认按照键的从大到小进行排序，也可以自己规定键的排序规则

HashMap底层实现

1. 当利用空参构造的时候，只是制定了加载因子为0.75，此时底层数组还没有，是null
2. 当调用put方法像数组赋值的时候，数组才创建，方法返回值为被覆盖的value，如果不是覆盖，而是添加，那么返回值是null
3. Put方法中调用putval方法，四个参数
   1. 键的哈希值
   2. 键
   3. 值
   4. 如果键重复了是否保留
      1. True，表示老元素的值保留，不会覆盖
      2. False，表示老元素的值不会保留，会覆盖
4. Putval方法会调用resize方法
   1. 如果当前是第一次添加数据，底层会创建一个默认长度为16，扩容银子为0.75的数组
   2. 如果不是第一次添加数据，会看数组中的元素是否到达了扩容的条件

如果没有到达扩容条件，底层不会做任何操作

如果到了扩容条件，底层会把数组扩容为原来的两倍，并把数据全部转移到新的哈希表中

1.TreeMap添加元素的时候，键是否需要重写hashCode和equals方法？

不需要

因为底层没有调用键的hashCode和equals方法

2.HashMap是哈希表结构，JDK8开始由数组，链表，红黑树组成的

既然有红黑树，那么hashMap的键是否需要实现compareable接口或者传递比较器对象呢？

不需要，因为在HashMap的底层，默认是利用哈希值的大小关系来创建红黑树的

1. TreeMap和HashMap谁的效率更高？

如果是添加的元素形成了链表，此时TreeMap的效率更高

1. Java会提供一个如果键重复了，不会覆盖的方法么？

会

Stream流的中间方法

Fielter->过滤

Limit->获取前几个元素

Skip->跳过前几个元素

Distinct->去重（依赖hashcode和equals方法）

Concat->合并两个流

Map->转换流中的数据类型

注意点1：中间方法，返回新的stream流，原来的stream流只能使用一次，建议使用链式编程

注意点2：修改Stream流中的数据，不会影响原来集合或者数组的数据

Stream流的中介方法

Foreach=>遍历

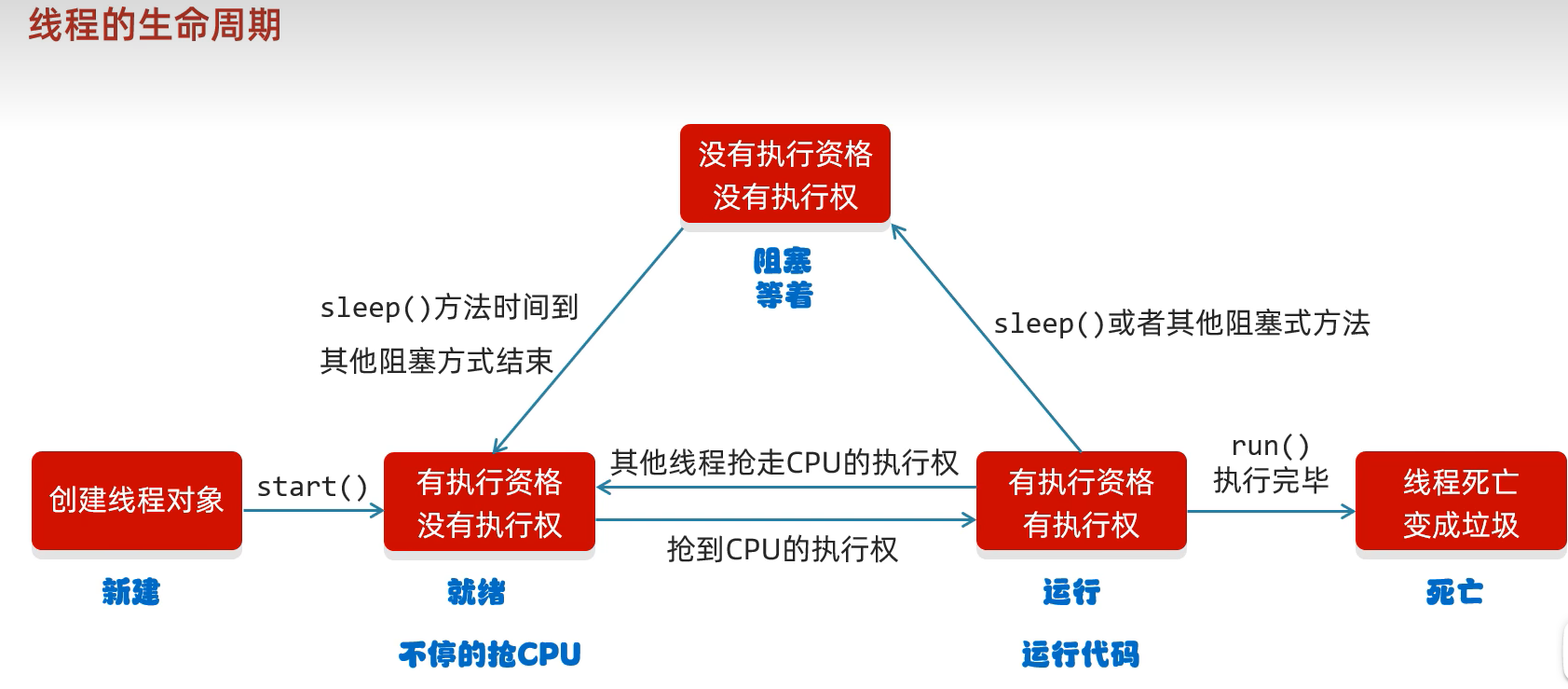
Count=>统计

ToArray()=>收集流中的数据到数组中

Collect(Collector collector)=>收集流中的数据到集合中

线程的生命周期

创建线程对象（新建）->有执行资格，没有执行权（就绪状态）->运行->死亡



线程安全的问题

同步代码块

把操作共享数据的代码锁起来

特点一：锁默认打开，有一个线程进去了，锁自动关闭

特点二：里面代码全部执行完，锁自动打开

同步方法

特点一：同步方法是锁住方法里面的所有代码

特点二：锁对象不能自己指定【非静态：this，静态：当前类的字节码文件对象】

线程的七种状态

新建->就绪->阻塞->等待->计时等待->死亡



自定义线程池

1. 核心线程数量
2. 线程池中最大线程的数量
3. 空闲时间（值）
4. 空闲时间（单位）
5. 阻塞队列
6. 创建线程的方式
7. 要执行的任务过多时的解决方案

常见软件架构

C/S client/server

在用户本地需要下载并安装客户端程序，在远程有一个服务器端程序

1. 画面可以组的非常精美，用户体验好
2. 需要开发客户端，也需要开发服务端
3. 用户需要下载和更新

B/S Browser/Server

只需要一个浏览器，用户通过不同的地址，客户访问不同的服务器

1.BS架构的优缺点，不需要开发客户端，只需要页面+服务端

2.用户不需要下载，打开浏览器就能使用

3.如果应用过大，用户体验受到影响

网络编程三要素：

IP地址：设备在网络中的地址，是唯一的标识

端口号：应用程序在设备中唯一的标识

协议：数据在网络中的传输规则，常见的协议有UDP、TCP、http、https、ftp

UDP协议

UDP是面向无连接通讯协议

速度快，有大小限制一次最多发送64KB，数据不安全，易丢失数据

UDP通讯程序（发送程序）

1. 创建发送端的DatagramSocket对象
2. 数据打包（Datagrampacket）
3. 发送数据
4. 释放资源

UDP通讯程序（接收数据）

1. 创建接受端的DatagramSocket对象
2. 接收打包好的数据
3. 解析数据包
4. 释放资源

UDP的三种通信方式

1. 单播
2. 组播
3. 广播

TCP协议

TCP协议是面向连接的通讯协议

速度慢，没有大小限制，数据安全

TCP通信程序

TCP通讯程序是一种可靠的网络协议，它在通信的两端各建立一个socket对象

通信之前要保证连接已经成立

通过Socket产生IO流来进行网络通信

客户端

1. 创建客户端的Socket对象与指定服务器端连接
2. 获取输出流，写数据
3. 释放资源

服务器端

1. 创建服务器端ServerSocket对象
2. 监听客户端连接，返回一个Socket对象
3. 获取输入流，读数据，并把数据显示在控制台
4. 释放资源

TCP传递中文乱码

因为一个中文是三个字节，默认一个字节进行解析，所以乱码

解决->将字节流变成字符流。

三次握手四次挥手

三次握手确保连接建立

第一次握手->客户端向服务器端发出连接请求，等待服务器确认

第二次握手->服务器端向客户端返回一个相应，告诉客户端收到了请求

第三次握手->客户端向服务器再次发送请求，连接建立

四次挥手确保连接断开，且数据处理完毕

第一次挥手->客户端向服务器发出取消连接请求

第二次挥手->服务器向客户端返回一个相应，表示收到客户端取消请求

第三次挥手->服务器向客户端发出确认取消信息

第四次挥手->客户端再次发送确认消息，连接取消