# 不能直接输出null（空常量）

| **byte** | **short** | **int(default)** | **long** | **float** | **double(default)** | **char** | **boolean** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1(c++:none) | 2 | 4 | 8(c++:4) | 4 | 8 | 2(c++:1) | 1(c++:bool) |

# 数据类型：

# 整数数字——默认为int,此时定义long需加L

**浮点数字**——默认为double。。

long a = 100L;  
float b = 1.2f;

# 标识符可为\_和$

# 小驼峰：方法、变量~firstName

**大驼峰**：类~GoodStudent

# 赋值时----小数据类型自动转换大数据类型(byte不能转char)；大->小需强转(elemType)

1. **算术运算----自动往int->long->float->double转换**
2. **字符串**

相+：从左到右：

”abc“ + 22 + 1 = abc221;  
22 + 1 + "abc" = 23abc;

# 逻辑运算符

逻辑& | !

短路 && || : 从左到右，左边一旦为假或真，右边将不执行

# Scanner 对象需要close()

# switch（表达式）

表达式：取值为byte；short；int；char；

JDK5以后可以是枚举，JDK7以后可以是String

case穿透：加入a==2，case3 及以后都不会判断，直接执行语句，直到break或结束

switch (a) {  
          case 1:  
              System.out.println("春季");  
              break;  
          case 2:  
              System.out.println("春季");  
              // break;  
          case 3:  
              System.out.println("春季");  
              // break;  
          case 4:  
              System.out.println("夏季");  
              // break;  
          case 5:  
              System.out.println("夏季");  
              // break;  
          case 6:  
              System.out.println("夏季");  
              // break;  
          default:  
              System.out.println("invalid number!");  
switch (a) {  
          case 1:  
          case 2:  
          case 3:  
              System.out.println("春季");  
              break;  
          case 4:  
          case 5:  
          case 6:  
              System.out.println("夏季");  
              break;  
          default:  
              System.out.println("invalid number!");  
      }

# 语句 do{} while(false)

# cmd指令：

cd e:

cd ..

cd \

cd directory

# Random:

Random r = new Random();  
      int num = r.nextInt(10); // [0,10)  
      System.out.println(num);

# IDEA：1.创建项目2.创建模块3.src下创建package-com.\*\*\*4.package 下新建一个类5.类中写代码

项目》模块》包》类

# 栈内存：类中的方法、局部变量--使用完毕即消失

堆内存：new出来的内容（实体、对象）-- 使用完毕，垃圾回收器空闲时被回收

new 出来的成员变量有默认值 null 0 etc；而局部变量无默认值，需定义才能用

# 方法 public static void a{return;}可直接写return

# 方法重载：

1.同一类；

2.相同方法名；

3.参数不同或数量不同；

# 类的方法，多个对象公用。

# 类的成员变量和方法的局部变量重名时。方法里面的变量只代表局部变量。此时可用this.变量名

# 面对对象三大特征：封装，继承，多态

# 封装：类中的隐藏信息，用类中的getxxx/setxxx方法来操作这些信息

优点：提高代码的安全性、复用性

# 构造方法：无参构造方法需写public 类名（）{}

# java.lang无需导入（自动导入）

# 字符串效果上相当于字符数组char,但是底层原理是字节数组byte[]

# String构建方法

String str1 = new String();  
      System.out.println(str1);  
​  
      char[] chs = {'a', 'b', 'c'};  
      String str2 = new String(chs);  
      System.out.println(str2);  
​  
      byte[] bys = {97, 98, 99};  
      String str3 = new String(bys);  
      System.out.println(str3);  
         
      String str4 = "nihao"; // 推荐使用  
      System.out.println(str4);

# String new出来的对象无论内容是否相同都一样。双引号“abc”直接构造建立多个对象地址都一样。

# str1 == str2 比较的是地址；str1.equals(str2)比较的是字符串内容

# String 对象 为一个地址

# for循环执行流程：1.执行初始化 2.执行判断 3.执行循环体 4.执行递增-》之后从2开始

# String (不可改变)字符串拼接后会构建新的对象，可用StringBuilder（可改变）

# new StringBuilder(new String("ok")).reverse().toString();

# 集合：ArrayList<String> array = new ArrayList<> ();

集合add的索引不能超出其元素个数-1

# new 类名（） 右边的为构造函数

# alt insert 自动创建成员变量的get和set 方法

# 父类、超类、基类

子类、派生类

# 继承：优点：提高代码的复用性（多个类相同的成员可以放到同一个类中）、维护性（如果方法的代码需要修改，修改一处即可）

缺点：类的耦合性增强了，当父类发生变化时子类也跟着变化，削弱了子类的独立性

# 局部变量》this成员变量》super父类变量（不考虑父父变量）

|  | **变量** | **构造方法** | **成员方法** |
| --- | --- | --- | --- |
| this | this. | this(...) | this.成员方法(...) |
| super | super. | super(...) | supter.成员方法(...) |

# 子类带参、不带参构造方法都需要先完成父类初始化（执行父类的无参构造方法）

# 如果父类无无参构造方法、子类将会报错，需要在子类所有构造方法加入父类带参构造方法

# @Override--底下的方法为父类方法的重写

子类的方法访问权限不能比父类低（public 》 默认 》 私有）

# Java 只能单继承，支持多层继承

# 权限修饰符

| **修饰符** | **同类** | **同包** | **不同包|子类** | **不同包|无关类** |
| --- | --- | --- | --- | --- |
| private（同类） | 1 |  |  |  |
| 默认（同包） | 1 | 1 |  |  |
| protected（同包或子类） | 1 | 1 | 1 | 1 |
| public（都可以） | 1 | 1 | 1 | 1 |

# 状态修饰符

final：修饰方法：不能被重写；修饰变量：不能被再次赋值；修饰类：不能被继承；

1.final修饰局部变量 ：变量基本类型->数据值不能改变

变量引用类型->地址值不能改变，地址里面的内容可以改变

static：修饰成员方法，成员变量（被类的所有对象共享）（可通过类名调用：建议）

1.静态成员方法--只能访问-->静态成员变量|静态成员方法

# 多态polymorphism：需要override 的方法才能调用 优点：提高了程序的扩展性（定义方法时，使用父类型作为参数，将来使用时，使用具体的子类参与操作）

缺点：不能使用子类中特有的属性

# 子类之间不能强转，父类可强转为子类。

向上转型：父类 a = new 子类（）

向下转型：子类 a = （子类）父类对象；

# 抽象类：（一般有抽象方法）的类； 抽象方法：没有方法体的方法

# 子类继承抽象类，必须重写所有抽象方法或者子类也为抽象类

# 抽象多态：抽象类无法实例化，按照多态形式，通过子类实例化

# 抽象方法作用：限定子类必须要有某种方法

# 多态的形式：抽象类多态、接口多态、具体类多态（很少使用）

# 接口中变量为常量：默认有public static final

# 接口没有构造方法、只能有抽象方法。

接口写的方法void show()；默认带修饰符 public abstract

一个类没有父类默认继承Object类

# 内部类可以访问外部类成员，外部类需要创建内部类对象才能访问内部类

# 匿名内部类new 开头，局部内部类class开头

# (auto)装箱：基本类型转包装类类型（Integer i = Integer.valueOf(100);)

(auto)拆箱： 包装类类型转基本类型（i.intValue()）

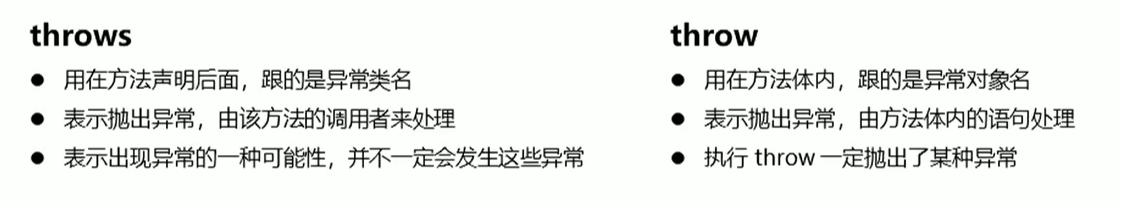
# 异常：



Try{}catch(){}处理异常：程序可运行下去

Throws处理异常：将异常扔给上一级，上一级再调用throw catch处理

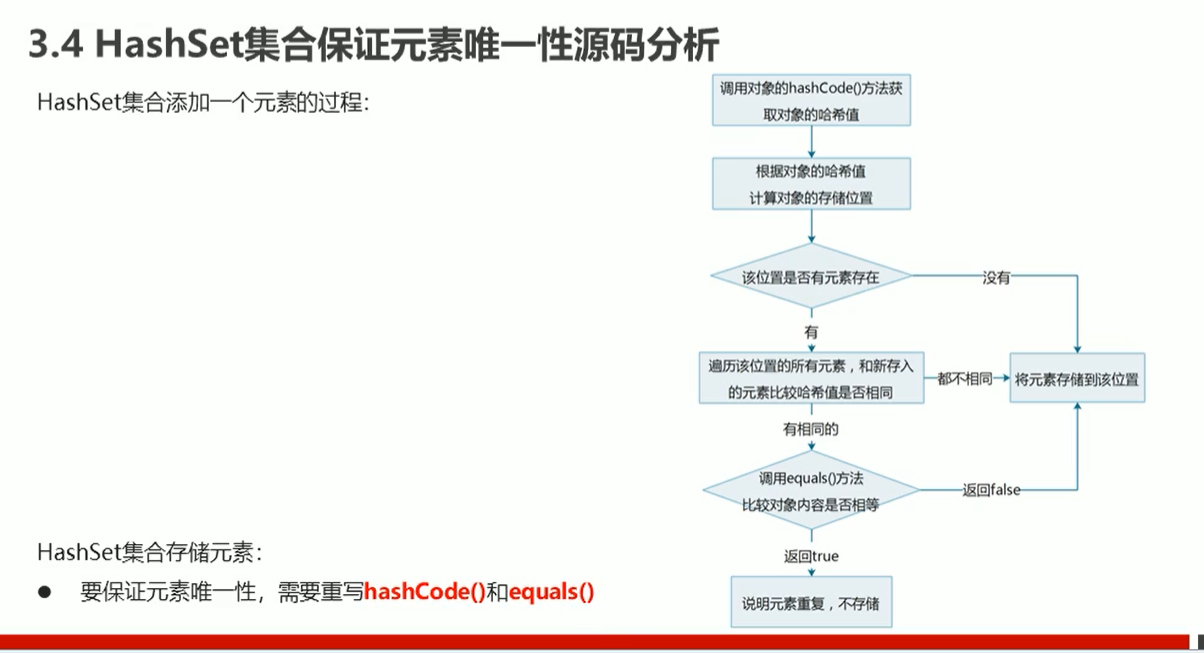
# throw 和 throws 的区别



# Alt+ 7 可以查看类的所有信息

# ConcurrentModificationException Iterator的并发修改异常

# HashSet集合保证元素唯一性源码分析



# Hashset由初始大小为16的数组和链表构成。

**Hash%16 = 该元素位置**

# 接口Comparable的实现类可进行排序

int compareTo([T](https://www.matools.com/file/manual/jdk_api_1.8_google/java/lang/Comparable.html" \o "type parameter in Comparable) o)

将此对象与指定的对象进行比较以进行排序。 返回一个负整数，零或正整数，因为该对象小于，等于或大于指定对象。

实现类要对其进行重写；

# Collection 大总结：

Set接口

* HashSet：1.元素不重复

判断元素唯一性（1.hashCode()方法 2.equals方法）

* TreeSet：1.元素不重复 2.自带排序

类似自带Collections.sort();

判断元素唯一性：将此元素与集合中所有元素判断

T implements Comparable

（compareTo(T o) return 0；）

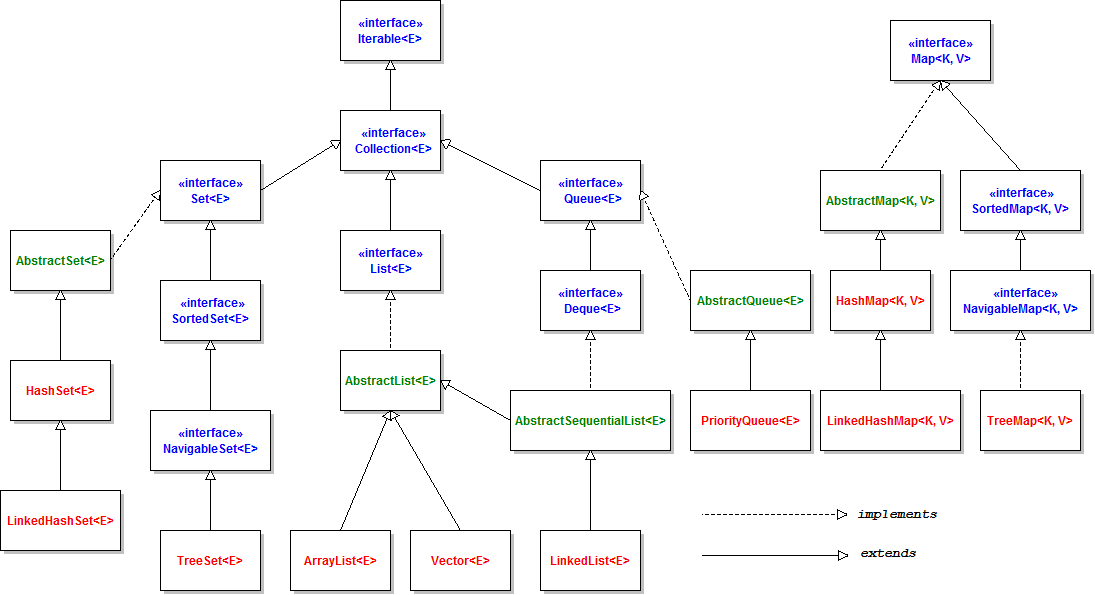
comparator implecomts Comparator

（compare(T o1, T o2) return 0;)

* LinkedHashSet：1.元素不重复 2.按存储顺序排序

判断元素唯一性：根据引用相等性（双方哈希值）o1 == o2

List接口 可用Collections.sort()来排序

* ArrayList：1.元素重复 2.按存储顺序排序
* LinkedList：1.元素重复 2.按存储顺序排序

# 泛型：

泛型类：public class name<T>{}

泛型方法：public <T> void name(T t){}

泛型接口：public interface<T>{}

**类型通配符：<?>**

**<? extends 类型> 限定？的上限（extends 有实现与继承之意）**

**<? super 类型> 限定? 的下限**

# 可变参数

**public void name(int... a){}**

其中a为数组

sout(a) 为[I@324234

**若有参数b**

public void name(int b, int... a){}

# Map

HashMap

**不允许键相同**

遍历HashMap

1. keySet()->get(key)
2. entrySet()->entry.getValue() or entry.getKey()

TreeMap

带排序的HashMap

# IO

## 字节流：InputStream、OutputStream

new FileOutputStream()：

1. 调用系统功能创建文件（有则覆盖）

2. 创建字节输出流对象

3. 对象指向创建好的文件

换行符：

Windows \r\n

Linux \n

Mac \r

方法：

FileInputStream chains BufferedInputStream

FileOutputStream chains BufferedOutputStream

## 字符流 Reader Writer

方法：

InputStreamReader、OutputStreamWriter

FileReader FileWriter

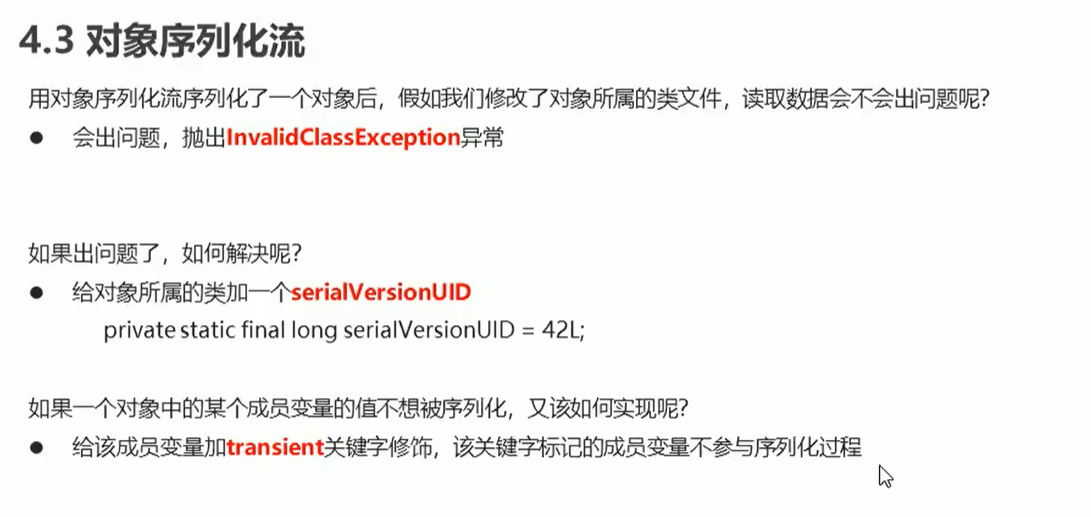
FileReader chains BufferedReader

FileWriter chains BufferedWriter

## 异常



## 注意



## Properties

存储键值对

特有方法：load(reader), store(writer, comment)

## IO大总结



字节输入、输出流

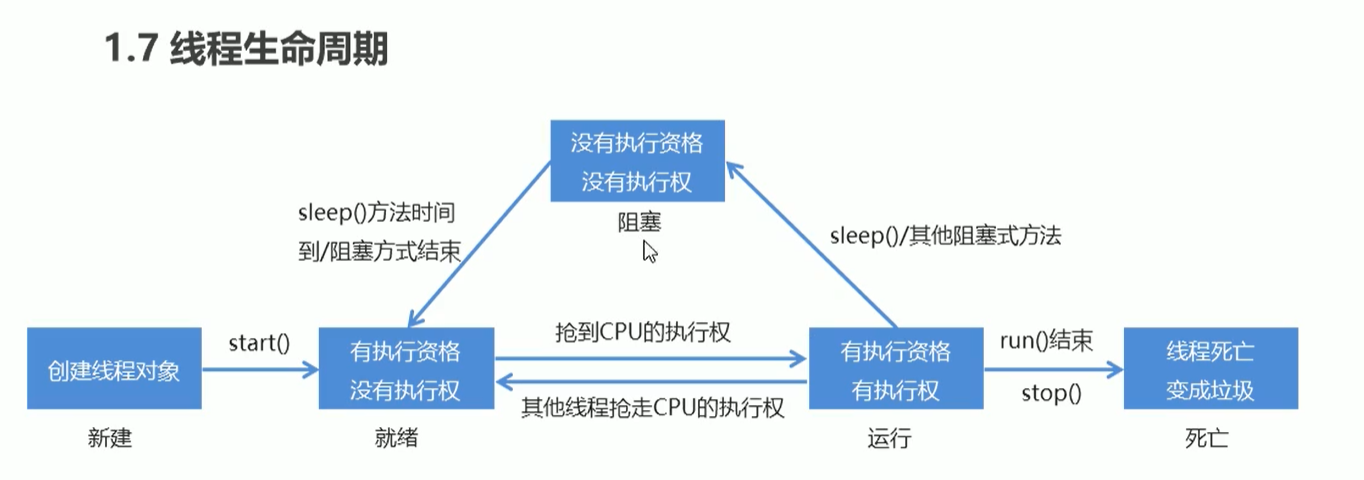
System.in

System.out

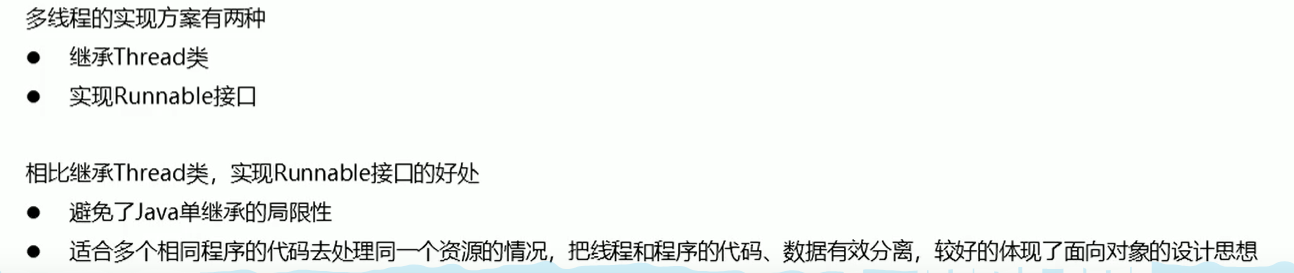
# 进程与线程

## 方法

## 线程的声明周期



## 关于Runnable



## 同步

锁的对象：obj 需要同意把锁才能使用

同步代码块 synchronized (Object obj) {...}

同步方法 : public synchronized void method() {...}

1. 非静态方法 obj 为 this
2. 静态方法 obj 为 类名.class

## 线程安全类

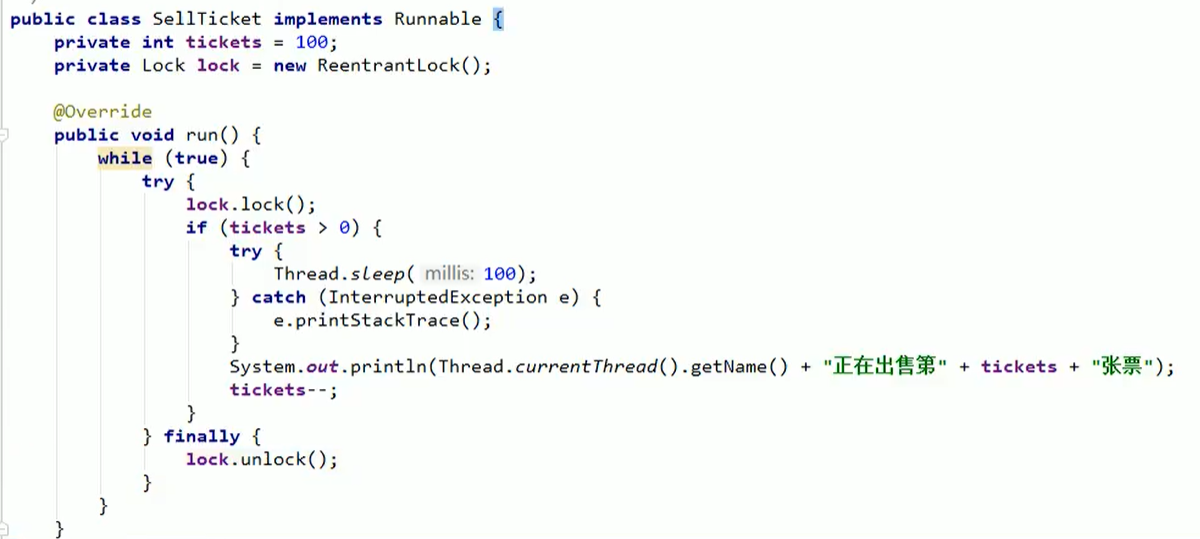


Vector和Hashtable 都会被Collections.synchronizedSet\Map\List 替代

## Lock接口

具体的锁

ReentrantLock 来实例化



防止代码块中出现错误未能解锁，需加try finally

## 生产者与消费者问题

# 网络

## 网络编程三要素：1. IP地址 2.端口 3.协议

**UDP**

## DatagramSocket() \ DatagramPacket()

**TCP**

## 服务器端ServerSocket

## 客户端Socket

# 字符集

GBK ：一个中文文字及标点符号占2字节

utf-8：占3字节；

英文数字标点符号占1个字节

# Lambda

将方法定义给 只有一个抽象方法的接口的方法

内部类：编译后，产生单独的.class字节码文件

Lambda：编译后，没有单独的.class字节码文件，对应的字节码文件只在运行的时候产生

* 方法的引用 ： System.in（对象）::println
* 构造器new的引用： 类名：：new
* 函数式接口（@FunctionalInterface 写在接口定义的上部）：有且仅有一个抽象方法的接口

**常用函数式接口：**

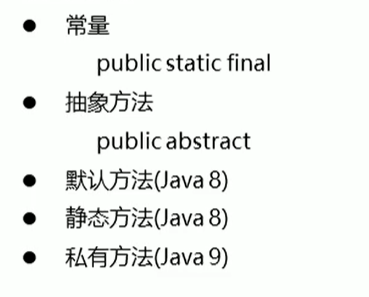
1. **Supplier<T>：return T数据的**
2. **Consumer<T>：接收T数据，并作出反应**

**andThen()：返回一个Consumer对象，其方法中有两个对象的accept**

1. **Function<T, R>：接收T数据，转换成R数据返回**
2. **Predicate<T>：接收T数据，进行一定判断，返回boolean**

# 接口

## 接口组成



## 默认方法

default doSomething(){...}

用于接口的升级；该方法可被重写，但不必重写

默认带public，可不用写public

## 静态方法

static doSomethin() {..}

只能通过类名调用

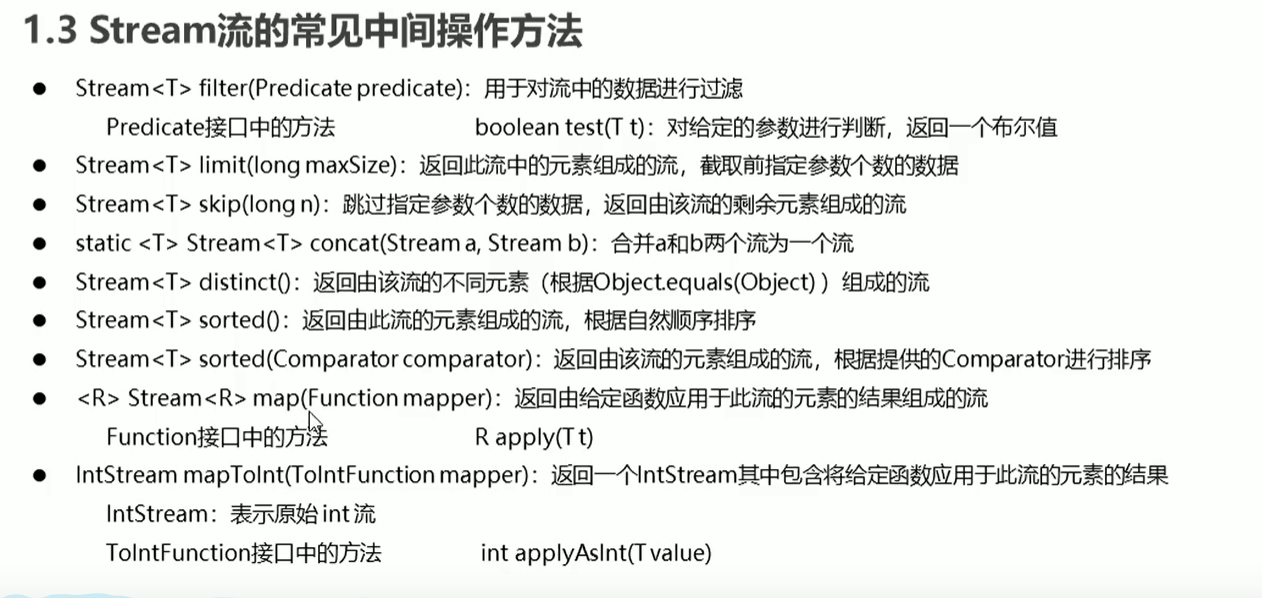
## 私有方法

private static void mutual() {...}

用于将default方法和static方法中的共同部分提取出来

# Stream流

* 生成流stream() of()
* 中间操作



* 终结操作forEach()

