跨平台： JVM（virtual machine）

运行环境：JRE（runtime environment）

JDK（develop kit）= JRE+开发工具=（JVM+核心类库）+ 开房工具

Javac（compile）

命令提示符：

Win + R --cmd

-cls:clear

注释：

单行： //

多行： /\* .... \*/

空常亮不能直接输出

/\*

Java 程序中最基本单位是类

public class 类名{

}

\*/

public class HelloWorld {

/\*

代码从main 方法开始执行

\*/

public static void main(String[] args){

//输出语句

System.out.println("HelloWorld")；

}

}

**基本数据类型：**

-类，接口，数组

-浮点数默认：double

-long类型：100000000L

-float类型：13.14F

-int：Integer

-char：Character

//定义变量

int a = 10;

//输出变量

System.out.println(a)；

//修改变量

a = 20;

小驼峰：方法和变量

大驼峰：类

数值和字符类型不兼容

强制类型转换：int k = (int)88.88

整数相除只能得到整数，要想得到小数，必须有浮点数参与

不同类型数值进行运算，short,byte,char都变成int（long,float, double）

Java中定义变量时必须要声明变量数据类型

i=10

i++:i加1

1. -：i减1

j=i++: i=11;j=10

j=++i: i=11;j=11

与：&

或：|

异或：^ 相同False；不同True

非：！

短路与：&& 只要左边是False是右边不执行

短路或：|| 只要左边是True 右边不执行

三元运算符：int max= a>b?a:b true:a是运算结果；false:b是运算结果

接收数据：

直接在代码出输入Scanner可以自动导包

//导包

import java.util.Scanner;

public class Scanner {

public static void main(String[] args){

//创建对象

Scanner sc = new Scanner(System.in);

//接收数据

int x = sc.nextInt();

//输出

system.out.println("x：" + x);

}

}

if语句：

public class IfDemo {

public static void main(String[] args){

//赋值

int a = 10;

int b = 20;

//if语句

if (a > b){

system.out.println("a大于b");

}else{

system.out.println("a不大于b");

}

}

}

奇偶案例：

import java.until.Scanner;

public class IfDemo {

public static void main(String[] args){

Scanner sc = new Scanner(system.in);

int num = sc.nextInt();

//if语句

if (num % 2 == 0){

system.out.println(num + "是偶数");

}else{

system.out.println(num + "是奇数");

}

}

}

If... else if...else

switch语句：（等于case值）

import java.until.Scanner;

public class IfDemo {

public static void main(String[] args){

Scanner sc = new Scanner(system.in);

int month = sc.nextInt();

//switch语句

switch(month) {

case 1:

case 2:

case 12

system.out.println("冬季");

break;

case 3:

case 4:

case 5:

system.out.println("春季");

break;

default:

system.out.println("秋季");

}

}

}

for语法：

for(初始化语句；条件判断语句；条件控制语句){

循环体语句

}

public class IfDemo {

public static void main(String[] args){

for (int i = 1; i <= 5; i ++){

System.out.println("Hello World");

}

}

}

public class IfDemo {

public static void main(String[] args){

int sum = 0;

for (int i = 1; i <= 100; i ++){

if (i % 2 == 0){

sum += i;

}

}

System.out.println(sum);

}

}

While:

初始化语句；

While (条件判断语句) {

循环体语句；

条件控制语句

}

数组：相同类型

-int[ ] arr = new int[3] （3个元素）

-int[ ] arr = {1, 2, 3}

-new:申请内存空间（将对象实例化）

-变量名（局部变量）：栈内存 ；数据存储：堆内存

-数组长度：arr.length

定义方法：

-public static void 方法名 {}

带返回值的方法定义：（把void改成数据类型）

-public static 数据类型 方法名{}

方法重载：方法名相同，参数不同（可共存，且系统可识别）

-println :换行（先输出后换行）

-print: 不换行

类中

-private

-无参构造

-含参构造

定义成员属性直接用数据类型+属性名（成员变量/类属性）

定义成员方法是public void 方法名 {} （无static）

类储存在堆内存中，主程序代码执行在栈内存

内部变量（局部变量）：在类属性中 private int age （提供get,set 方法）

-this.name 同于self（指对象）

-初始化：public Student(参数) {} 同于\_\_init\_\_

-初始化要有一个无参数的和一个有参数的

API：application programming interface

字符串：

==比较地址是否相同

-equal方法比较内容是否相同

获取字符串索引位置字符： line.charAt(index)

string不可变类型

stringbuilder可变类型---先建立对象，append,reverse

转换：toString;StringBuilder(String s)

集合ArrayList<E>

方法： add() 可指定元素

-remove

-set

-get

-size 同于length

系统退出：system.exit(0)

制表符\t：指的是tab

Arraylist:储存的是对象，和用类创造的对象信息相同

每个数据的属性都包含在每个元素里面

-Student s = array.get(index) s.getSid

只有if ..else...就将else换成return

-boolen类型都为小写

有return或创建对象都可以用ctrl+alt+v

继承：public class 子类名 extends 父类名{}

访问父类的成员变量：super.age 相对于this

每个子类构造方法第一条语句默认：super() 访问父类无参构造方法

表明重写父类方法：@override

权限修饰符：1private 2无（默认） 3protected 4public

同一包下：2,3,4

不同包下有继承：3,4

不同包下无继承：4

状态修饰符： final, static

final不可被重写；不可被继承；不可被赋值（意为常量）

static表示该变量被成员变量共享，若其他对象未赋值，用类名访问（所有对象相同的属性）：

-public static String s

静态成员方法只能访问静态成员

多态：

成员变量：编译看左边，执行看左边

成员方法：编译看左边，执行看右边

Animal a = new Cat()向上转型 Cat c = (Cat) a向下转型

**接口：（指定标准）**

-public class Cat implements interface

-成员变量（全局静态常量）默认：public static final

-成员方法只能是抽象的：默认public abstract 只给标准，具体怎么实现不管（之后被重写）

-静态方法：public static 类名调用

-默认方法：public default 实现类对象调用

-在类中若A和B接口的同名默认方法冲突 A.super.test()

-若接口默认方法与父类方法冲突 默认父类方法

-无构造方法

-继承只能单继承，接口可以有多个，且接口可以继承多个接口的抽象方法

-抽象类对类（事物）抽象，接口对行为抽象

-很多对象没有相同的父类，却可以有相同的行为（鸟，飞机，风筝）

-抽象类的继承是全部继承行为，接口是可以选择继承行为（将抽象类中可有可无的行为，单独作为接口实现）

-接口类型变量可与实现类的对象构成多态引用（数组应用）

-让对象之间进行比较的抽象方法：comparator接口（实现类重写compare方法）和comparable接口（重写compareTo方法）

-数组sort方法中可用comparator排序 sort(Object[] a, Comparator c)

-comparable定义一种比较方法（自然排序,sort自动调用）；comparator可定义多个

把int转换为string i+1+””

可变参数：int ... nums

结合：int num1,int ... nums

**数组：**

-String [] arr 存了一组字符串对象

-Student[] arr 存了一组学生对象 先创建数组对象，再创建学生对象

**Array方法：**

-arraycopy的参数：（可以实现同一数组内的元素移动） ——右移：插入； 左移：删除

-src：原数组对象

-srcPos：从原数组[srcPos]开始复制

-dest：目标数组对象

-destPos：从目标数组[destPos]开始储存

-length：复制元素数量 左移一位为total - index -1 右移一位为total - index

-遍历数组：toString

-Array.copyOf（all, all.length\*2） *数组扩容*

**权限修饰符：**

-限定访问权限

-缺省在本包可以使用

-属性大多private 外部通过方法访问类属性

-方法大多是public

-权限中boolean的获取方法 boolean isMarry() 不用get

**包：**

-包名：

-所有单词小写，用.分割

-用公司域名倒置+模块名

-com.mysql.xxx

-带包的编译：javac -d . HelloWorld.java d是direct的意思 在当前文件夹下创建包

**多态：继承+重写**

-左边父类类型，右边是子类对象（默认向上转型）

-eg: Person p = new Woman() 向上转型

Woman w = (Woman)p 向下转型（强转） ——为了调用子类特有方法

-向下转型，if (p isinstanceof Woman){

Woman m = (Woman) p

M.shop()}

-方法：编译看左边（父），运行看右边（子）

-属性：只看左边的编译类型（父的属性），强转属于编译行为

-this,print() this代表子类对象，若方法print重写，则代表子类的方法

-好处：（用父的方法实现代替多个子类的重复代码）

-多态数组：可以创建一个父类数组，将子类对象作为元素进行添加

-多态参数：形参为父，实参为子

**修饰符：**

-final：

-类：无后代，不能被继承（类中所有都不能重写）

-方法：方法不能被重写，但可以被继承

-变量：常量，不能被修改（不能用set方法），但对象属性可修改

——所有单词大写，在定义时手动赋值

-对象：内存地址不能修改，不能再c=new Circle赋值

-native：（不重要）

-语法格式：native 返回值类型 方法名（形参列表）；

-只修饰方法名，无方法体

-是调用了底层的C/C++代码，被编译为.dll（dynamic link library）文件，Java来执行

-子类可重写

-abstract：

-抽象类不能创建对象

-抽象类有构造器，但不为自己创建对象，只在子类创建对象时调用

-public abstract class Animal public abstract void eat()

-抽象方法必须在抽象类中

-用多态来访问抽象类中的非抽象类

-static和abstract不能一起使用（static方法不能重写，abstract需要重写）

-static

-static修饰的方法可以用“类名.方法”进行调用（类方法） ——工具类都用静态方法

-静态不能引用非静态/super/this

-静态方法不能被重写（多态引用的方法也是父类的）

-可修饰成员变量并存储在方法区，不可修饰局部变量

（表示为类属性，所有对象共享，改一个对象的属性，所有对象一起改）

-静态代码块：在类初始化的时候有且仅执行一次

-类初始化<clinit>：（一个类只有一个）

-初始化过程：

-静态变量显式赋值

-静态代码块（可为类变量赋值）

-不能被重写的方法：final,static,private,跨包缺省

**类：**

-this()表示调用无参构造

-子类需要手动调用父类的有参构造

-java中只有直接父类和间接父类（爷爷类）

-子类的返回值可比父类小，权限可比父类大

-变量在内存中的储存位置是按成员变量和局部变量分；而不是按数据类型分的

-创建对象初始化过程：

-1为成员变量显式赋值（变量的默认值）；

-2执行非静态代码（就是类中方法以外的代码）；

-3执行构造器

-1,2顺序执行，3最后执行

-继承时，先进行父类的实例初始化，再进行子类的实例初始化

-toString方法需要重写，类似于\_\_str\_\_ 可直接设定对象打印结果

-getClass：返回**运行**时的类

-finalize：类似于\_\_del\_\_

-哈希码：记录对象的信息（属性值），类似对象的身份证

-不同对象可能有相同哈希值 如：Aa.hashCode=Bb.hashCode

-哈希码相同≠对象相等（equal）

-hashCode不同，对象肯定不同

-先用hashCode（快），再用equal（准）

-String重写equals方法（快捷键）之后，只比较内容是否相同

**内部类：**

-匿名内部类，既是类也是对象（是原有类的子类） new Object()

类似于lambada，为了直接重写父类/父接口的方法

-当描述一个事物时，发现它的内部还有一个完整的结构需要用一个类来描述，并发现这个内部 的结构，如果独立存在没有意义，必须在外部类中才有意义，且只为这个外部类服务

-静态内部类不会随着外部类的初始化一起初始化，而是在用到这个静态内部类时才会初始化

-需要一个外部类对象来调用非静态内部类的构造器

-用get方法来访问内部类对象

-字节码文件形式：外部类名$【非】静态内部类名.class

-局部内部类：

-修饰符只能是abstract和final

-在Java8中，如果某个局部变量被局部内部类调用，默认为常量final

（因为局部变量在栈中会消失，转换为方法区中的常量）

**变量：**

-在内存中的位置：

-类变量：方法区

-实例变量（成员/对象变量）：堆

-native方法的局部变量：本地方法栈

-非native方法的局部变量：虚拟机栈

**枚举：**

-语法：【修饰符】 enum 枚举类型名{

常量对象列表

其他成员列表}

-常量对象可以调用有参/无参构造

-枚举是一种特殊的类，只是对象是有限的常量**对象**

-枚举类型有隐含父类Enum——不能再继承其他父类

-Enum类中有唯一构造器 protected Enum(String name, int ordinal)

编译器自动调用，自动传入参数

-方法：

-values返回所有枚举对象的数组

-valueOf返回指定常量名的对象

-int compareTo（Object obj）：按照常量对象顺序比较

**注解（Annotation）：**

-代码注释代码

-@开头

-@Override,@SuppressWarnings(“all”),@Deprecated：标记已过时，不建议继续使用

-文档注解：/\*\*

-@author，@see,@since

-@param 形参名 形参数据类型 解释

-@return 返回值类型 解释

-@throws 抛出异常类型名 解释

-@exception

-元注解：注解注解的注解

-@Target（ElementType.METHOD）：标记注解可用于的位置

-ElementType是枚举类，有常量

-常量：TYPE*（类、接口）*,METHOD,ANNOTATION,FIELD*（属性）*等

-@Retention（RetentionPolicy.RUNTIME）：留存

-SOURCE：源代码阶段.java

-CLASS：字节码阶段.class

-RUNTIME：运行时JVM

-@Document：标记该注释是否可以被javadoc.exe读取到API中

-@Inherited：继承注解

-注解声明：【修饰符】 @interface 注释名{ *注解默认是接口*

配置参数列表} *例：String name() default joe*

*-如果注解的配置参数只有一个且为value，赋值时可以省略“value=”*

-使用注解：@注释名

**静态导入：**

-语法：import static 包.类名.静态成员；

**异常：**

-不同于语法错误（编译时就无法通过）

-异常定义：因为一些用户原因，导致程序无法运行

-检测异常语法：try{}catch（异常类型 e）{} 最后一个catch （Exception e）

-如果不能捕获异常，将这个异常抛给“上级”

-异常的公共父类Throwable

-Error：不应该捕获的严重错误 需要升级程序，硬件，架构

-Exception：

-运行时异常（非受检异常）：

不建议try...catch，因为发生频率太高，且一般是可避免的异常

-编译时异常（受检异常）：

强制要求编写try...catch

-异常对象的常用方法：

-e.printStackTrace()：打印异常详细信息（跟踪信息）

-e.getMessage()：返回异常对象中简单的错误信息提示 *没有红色提醒*

-打印异常/错误信息：System.err.println(xx) *有红色提醒*

-若finally中有return，前面的return将都无效，只以finally中的返回值为最后的return结果

-finally {input.close()}

-throws 异常名 让调用者知道应该catch什么异常，提醒在该方法内未处理，需要后续处理

-方法重写：子类抛出的异常类型<=父类的 *更精确*

-throw异常名：主动抛出异常

-正常RuntimeException，如果是编译时异常，要加throws

-可以代替return语句

-自定义异常：

-创建异常类继承RuntimeException和Exception

-保留无参构造super()和super(message)

-throw new AccountException 主动抛出

**Junit测试：**

-@Test：可以作为一段独立的代码进行测试

-只能测试public void的无参代码 有参的需要进行调用

-@Before:：每个test之前执行

-@After：每个test之后执行

-@BeforeClass：类初始化前（只执行一次） public static void

-@AfterClass：类初始化后

**包装类：**

-装箱（将数据类型赋值给对象）：int a = 10 Integer aObj = a

-拆箱（将对象赋值给数据类型）：Integer i = new Integer(10) int b = i

-包装类与基本数据类型进行比较会自动拆箱

-包装类（Integer）的方法

-字符串转int等基本数据类型：Integer.parseInt（“123”） Integer.valueOf

-基本数据类型转为字符串 int a = 10 String str = a + “”

-进制转换：toBinaryString,toOctalString,toHexString

-大小写：Character.toUpperCase,Character.toLowerCase

-自动装箱的缓存范围（共同的常量对象）：

-Byte,Short,Integer,Long：-128~127 超过这个范围则另外创建对象

-Float,Double无缓存对象

-Character：0~127 最早的ASCII码的128个字符

**线程：**

-main线程是Java程序进程的第一个线程

1. -多线程方式：

-继承Thread类

-重写run()

-MyThread my = new MyThread()

-my.start() 启动线程

-实现Runnable接口

-重写run()

-MyRunnable my = new MyRunnable()

-Thread t = new Thread(my) 代理

-t.start()

1. -Runnable优势：

-无单继承限制

1. -用多线程代替多进程来进行多任务开发
2. -整个过程：新建--就绪--运行--死亡

-死亡原因：

-run任务完成

-遇到异常

-被其他线程stop结束

-运行期间被阻塞原因：

-sleep 时间到/interrupt，转到“就绪”状态

-wait notify

-join 被其他线程加塞 等到加塞线程结束

-没有锁 需要等对方释放锁

-suspense 挂起 resume

-新建到就绪：start()

-运行到就绪：时间到；yield暂停

1. -对象方法：getName,setPriority,getPriority，interrupt,join，yield

-优先级范围：[1,10] 普通优先级（默认）：5

-优先级影响的是被执行的概率，而不是对任务先后执行顺序进行排序

-用join加塞程序，必须等整个加塞的程序结束才能继续执行被加塞程序，此外的线程不受 影响

1. -类方法：currentThread
2. -volatile boolean：多个线程同时且频繁访问某个成员变量，防止缓存
3. -线程安全问题：static共享数据（Thread） 解决方式：加锁

-同步代码块synchronized(锁对象)

-锁的作用：保证某个进程可以执行完成，不被插队

-private static Object lock = new Object()

-while true synchronized(lock) if

-Runnable的对象属性就是共享的 while true synchronized(this) if

-while (total > 0) public synchronized void 方法名 if(total > 0)

每次执行方法都需要上锁

-非静态的锁对象默认：this

-静态的锁对象默认：当前class .class

1. -死锁：两个线程互相持有并占用对方想要的锁 可能会出现在有多层锁的情况下
2. -生产者与消费者问题：

-共享数据：线程安全问题，需要同步锁 超过下限

-共享区域大小固定且有限，需要协作，线程通信 超过下限

-wait()必须在synchronized的方法中 之后用notifyAll()唤醒

-while(count<=0) wait()

-sleep和wait的区别

-sleep()不释放锁，**wait()释放锁**

-Thread.sleep() wait()是非静态方法，需要用“同步锁”对象调用

1. 单例设计： *构造器私有化，导致外部无法再创建其他对象*
   1. 恶汉式： *上来就会创建出这个唯一的对象*
      1. 枚举
   2. 懒汉式： *延迟创建对象*
      1. 考虑效率和安全性，多重条件+锁
      2. 内部类创建
2. 字符串
   1. 字符串方法
      1. -equals
      2. -equalsIgnoreCase：忽略大小写
      3. -String类型是有CompareTo的方法的（实行了Comparable接口）
      4. -CompareToIgnoreCase
      5. -Collator Arrays.sort(arr, Collator.getInstance(Locale.CHINA)) 中文字典排序
   2. 字符串对象在栈中，char[]的value数组对象在常量池中（方法区）
   3. 字符串的拼接，
      1. 只有在常量池中的才是共享的，堆中变量之间拼接只会产生新的对象
      2. -xx.intern()方法表示把拼接的结果放入常量池中
      3. -String s3 = s1.concat(s2) 拼接
   4. 空字符串对象
      1. -String s = new String(“”) 总共有两个对象，一个在常量池中，一个在堆中
      2. “”
      3. -new String(“”)
      4. 判断是否为空字符串：（常量与非常量比较） “”.equals(str)
   5. - str = str.trim() 去掉字符串前后的空白符
   6. 字符char相关方法：

* -char[] toCharArray()
* -char charAt(index)
* -String(char[] arr, int offset, int length)
  1. 与byte相关
     1. 编码：字符→字节 给计算机看的
     2. 解码：字节→字符 给人看的
  2. 其他方法：
     1. 判断
        1. -boolean startsWith(xx)
        2. -boolean endsWith(xx)
        3. -boolean contains(xx)
     2. 定位
        1. -int indexOf(xx)
        2. -int lastIndexOf(xx)
     3. 截取
        1. -String substring(int beginIndex)
        2. -String substring(int beginIndex, int endIndex)
     4. 匹配：
        1. -matches(正则表达式)
           1. 正则表达式：

\d 在Java中需要写成\\d

* + 1. 替换 可用于删除（替换成空字符）
       1. -replace
       2. -replaceAll 可使用正则表达式
       3. -replaceFirst 可使用正则表达式
    2. 拆分
       1. String[] split(xx)
  1. **可变字符序列**
* -StringBuffer 线程安全
* -StringBuilder 线程不安全，但速度快
  + 1. 方法：
       1. -append(xx)
       2. -insert(index, xx)
       3. -delete(int start, int end)
       4. -deleteCharAt(int index)
       5. -reverse()
       6. -setCharAt(int index, char ch)
       7. -[setLength](mk:@MSITStore:C:\\Users\\dell\\Desktop\\JDK1.6%20API帮助文档.CHM::/java/lang/../../java/lang/StringBuilder.html" \l "setLength(int))(int newLength)
       8. -[substring](mk:@MSITStore:C:\\Users\\dell\\Desktop\\JDK1.6%20API帮助文档.CHM::/java/lang/../../java/lang/StringBuilder.html" \l "substring(int))(int start) 截取

1. 日期时间类
   1. -Date
      1. -System.currentTimeMillis() 距离1970-1-1 0:0:0
      2. -Date d = new Date(毫秒)具体时间 d.getTime获得毫秒
   2. -Calendar 抽象类，具有子类GregorianCalendar
      1. -Calendar c = Calendar.getInstance() c.get(Calendar.YEAR)
   3. -SimpleDateFormat
      1. SimpleDateFormat sf = new SimpleDateFormat(“yyyy年MM月dd日 HH时mm分ss秒 SSS毫秒 E Z”)
      2. sf.parse(“1995-01-01”)
   4. 代替Date和Canlendar

* -LocalDate LocalDate now = LocalDate.now()
* -LocalTime LocalTime now = LocalTime.now()
* -LocalDateTime LocalDateTime now = LocalDateTime.now()
  + 1. -of(xx)
    2. -getXxxx
    3. -plus
    4. -minus
    5. -isLeapYear()
  1. 日期时间格式化
     1. -DateTimeFormatter

1. 数学类
   1. -java.lang.Math

* -Math.ceil
* -Math.floor
* -Math.round
  1. -java.math
     1. BigInteger 用于特别长的整数
     2. BigDecimal 用于特别长的小数

1. 数据结构：存储数据的某种结构
   1. 底层的物理结构
      1. 数组：连续的存储空间
      2. 链式：不连续的存储空间，但需要结点来包装要储存的数据
   2. 表现出来的逻辑结构：动态数组、单向/双向链表、队列、栈、二叉树、哈希表、图等
2. 链式，二叉树
   1. 链式：（Node 节点）
      1. Object data, Node pre, Node next, (Node first)

这里的Node不是对象而是记录的地址值

* + 1. 把要存储的数据都包装成节点（Node）对象
    2. -first.data == null表示存储数据为空 first == null表示为空链表
  1. 二叉树
     1. Object data, Node parent, Node left, Node right
     2. 存储的信息较多，占用内存较多，但查询速度快

1. 集合接口
   1. -Collection:规范单值集合的接口,obj 集合中不能存储基本数据类型，只能存储对象
      1. API
         1. 添加

* -add(Object obj) 只添加一个元素
* -addAll(Collection c) 把Collection中的所有元素分别都添加到XXX
  + - 1. 获取有效元素的个数： int size()
      2. 包含
* -contains(Object o)
* -containsAll(Collection c)
  + - 1. -boolean isEmpty
      2. -remove(Object o) removeAll(Collection c)：this = this - this ∩ c clear()
      3. -retainAll(Collection c)：this = this ∩ c
      4. -Object[] toArray()
    1. 子接口：List, Set
       1. List：有序，可重复 有索引
          1. 实现类：

ArrayList(动态数组)

初始化为空数组

调用add添加元素，扩容为长度为10

如果再不够，扩容为1.5倍

-

Vector(动态数组、向量类)

与ArrayList的区别：**Vector有线程同步（线程安全）**；不够后扩容为2倍（ArrayList扩为1.5倍）；初始容量为10

如果可以预估元素个数，ArrayList和Vector都可以初始化容量

Stack(栈) Vector的子类

a）先进后出（FILO）或后进先出（LIFO）

b）方法：

i. Object peek() 访问当前栈顶元素

ii. Object pop() 弹出栈顶元素

iii. Object push(Object item) 把元素压入栈顶，等价于add

LinkedList(双向链表、双端队列、栈....)

双向链表的方法

-E getFirst() 返回此列表的第一个元素

-E getLast() 返回此列表的最后一个元素

-boolean offerFirst(E e) 在此列表的开头插入指定的元素

-boolean offerLast(E e) 在此列表末尾插入指定的元素

-int indexOf(Object o) 返回此列表中首次出现的指定元素的索引

-int lastIndexOf(Object o)

-E get(int index) 先判断index是靠前还是靠后

队列方法：（FIFO）

-offer(e) 插入队头

-poll() 移走队头元素

-peek() 检查队头元素

双端队列Deque（double ended queue）

* + - * 1. API

-void add(int index, Object element)

-boolean addAll(int index, Collection c)

-Object remove(int index) 若要调用删除对象的方法：list.remove(new Integer(1))

-Object get(int index)

-Object set(int index, Object element)

-int indexOf(Object o)

-int lastIndexOf(Object o)

-List subList(int fromIndex, int toIndex)

-ListIterator listIterator()：Iterator的子接口，增加的方法：从后向前遍历，增加修改

-boolean hasPrevious

-Object previous()

-void add(E e)

-void set(E e)

-int nextIndex()

-int previousIndex()

* + - 1. Set接口 底层是map
         1. 元素不能重复

HashSet和LinkedHashSet判断重复的方法：先比较hash值（值不同一定不相同），再用equals比较是否相同

TreeSet用大小来避免重复

* + - * 1. 实现类:

LinkedHashSet可以保证元素添加顺序

TreeSet可以保证元素大小顺序 若没有实现comparable接口，就无法添加TreeSet

HashSet完全无序

* + 1. 实现类：ArrayList 动态数组
    2. 迭代器：每个集合对象都有一个迭代器

-Iterator接口对象的方法：

* + - 1. -hasNext()
      2. -next()
      3. -remove()
      4. -for (元素数据源类型 临时变量名 ： 数组或集合名) foreach循环
         1. 凡是实现了java.lang.Iterable接口（可迭代）的集合或容器都可用foreach
         2. Iterable需要实现Iterator，foreach底层是使用迭代器来遍历集合
         3. \*\*不要使用foreach删除，因为remove是集合的方法，遍历调用的是Iterator的遍历方法，有线程安全问题
  1. -Map：规范对值集合的接口，（key, value）

1. API

-put(Object key, Object value)

-putAll(Map m)

-int size()

-boolean containsKey(Object key)

-boolean containsValue(Object value)

1. 遍历

因为Map接口没有继承Iterable接口，不支持foreach循环遍历

①Set keys = map.keySet();

for (Object key : keys) {

System.out.println(key + “→” + map.get(key));

}

**因为Set集合不可重复，而键是唯一的，且一一对应**

②Collection values = map.values();

for (Object value: values) {

System.out.println(value);

}

**value组成了一个Collection集合，可能重复**

③Set entrySet = map.entrySet();

for (Object entry:

entrySet) {

System.out.println(entry);

}

**因为key不可重复，所有映射组合也唯一，用set**

若key重复了，会进行覆盖

1. 实现类：

①HashMap和Hashtable

-Hashtable线程安全，key和value不能为null

②HashMap和LinkedHashMap

-LinkedHashMap是HashMap的子类，多记录了添加顺序（所有的linked都是有顺序功能的）

③HashMap和TreeMap

-TreeMap按照key大小排序

④Properties

是Hashtable的子类，key和value不能为null，且都是String类型

通常用于存储配置属性

-setProperty(key, value)

-getProperty(key)

-System.getProperties()

**HashMap的容器设计思路： 数组+链表**

hash和index确定位置 hash进行散列运算（capacity为2的n次方），为了保证散列，会在capacity下设置一个阈值

index均匀分配 若一个index下有多个元素，用链式储存

Index=hash(key.hashCode()) & table.length -1

key有重复替换，无重复判断扩容，扩容后index重新分配

Entry相当于一个结点类型

class Entry {

int hash;

Object key;

Object value;

Entry next;

}

**HashMap的容器设计思路： 数组+链表+树**

TREEIFY\_THRESHOLD:默认树化阈值8，当链表到达这个长度要树化

UNTREEIFY\_THRESHOLD：默认反树化阈值，当树中结点达到这个值考虑变为链表

MIN\_TREEIFY\_CAPACITY：最小树化容量64

当单个链表结点达到8，且table长度达到64，才会树化（否则先扩容）

entry表示映射关系，size表示其对数

modCount：记录当前集合被修改的次数

增加，删除

1. 泛型
   1. 泛化类型：因为代表的是数据的类型，所以分为类型形参和类型实参
   2. 在类，接口，方法后面声明类型
   3. 泛型实参只能是引用数据类型

class Student<T> {

private String name;

private T score;

} T就是score的类型，说明score类型可能包含多种类型

* 1. 泛型在创建对象时指定实参
  2. 继承泛型类、实现泛型接口是可以指定泛型实参
  3. 不能用于静态成员方法上（因为跳过了创建对象时的实参指定）
  4. 限定实参类型：（extends 上限）

class Student<T extends Number> { 只能指定数字，不能指定字符串

private String name;

private T score;

* 1. 泛型形参的代表类型

ArrayList<E>：E代表集合的元素（Element）类型

Map<K, V>：K代表key，V代表value

Comparable<T>：对象类型

* 1. 若没有指定类型，按上限类型（最高Object）

若有多个上限（用&连接），按照第一个处理

* 1. 泛型方法
     1. 若某静态方法想要使用泛型，需要单独设计

Arrays: public static <T> List<T> asList(T... a) T...是可变参数

* + 1. 泛型上的形参不适用于方法，那么该方法需要单独设计

<T> T[] toArray(T[] a)

* + 1. 泛型语法格式： 【修饰符】<泛型形参列表> 返回值类型 方法名（【数据形参列表】）

1. 通配符：wildcard（在foreach循环方法中可作为形参）
   1. 种类：
      1. <?> 代表任意类型
      2. <? extends 上限>
      3. <? super 下限>

public <T> void copy(Collection<T> src, Collection<? super T> dest) {

for (T t :

src) {

dest.add(t);

}

public static <T> int binarySearch(List<? extends Comparable<? super T>> list, T key)

list集合中元素必须实现comparable接口

接口/方法可以是目标的父类（super） 可以在子类容器中找/存父类（extends）

1. File类：代表文件或目录
   1. API

-getName

-length 返回文件长度，不能返回目录大小

-isHidden

-exists

-canRead

-canWrite

-getParent 返回父目录名称

-getParentFile 返回父目录file对象

-lastModified 最后修改时间

SimpleFormat sf = new SimpleDateFormat(“yyyy-MM-dd HH:mm:ss”)

sf.format(new Date(file.lastModified))

-getPath 指定的路径

-getAbsolutePath 可能带.././

-getCanonicalPath 规范路径

-createNewFile

-mkdir 只能创建一级目录

-mkdirs 创建多级目录

-delete：只能删除最里面的一层

-renameTo(File dest)

-isFile

-isDirectory

* 1. 文件API

-String[] list()

-File[] listFiles

-File[] listFiles(FileFilter)

**显示所有文件（迭代）**

public void listAllSub(File dir) {

if (dir.isDirectory()) {

File[] listFile = dir.listFiles();

for (File sub :

listFile) {

listAllSub(sub);

}

}

System.out.println(dir);

}

**显示文件夹大小**

public long getSize(File f) {

if (f.isFile()) {

return f.length();

}else if (f.isDirectory()) {

File[] listFile = f.listFiles();

long sum = 0;

for (File sub :

Objects.requireNonNull(listFile)) {

sum += getSize(sub);

}

return sum;

}

return 0;

}

**显示以.java结尾文件（文件过滤器）**

public void test02() {

File dir = new File("D:/atguigu/javaee");

File[] listfiles = dir.listFiles(new FileFilter() {

// File pathname 表示dir目录下的子文件

// 返回true则要保留

@Override

public boolean accept(File pathname) {

return pathname.getName().endsWith(".java");// 结果为true/false

}

});

for (File sub :

listfiles) {

System.out.println(sub);

}

}

* 1. IO（想象成从池子里用水管接水）

参照物是内存；

将数据写入文件：输出；从文件读取：输入

纯文本文件：.txt .html .xml .properties 不是：.doc .xls

字节流：可以适用任何类型文件，但用字符流存纯文本文件快

节点流：如文件流

处理流：在节点流基础上，加其他处理、装饰功能 例如：缓冲流、序列化和反序列化

四大抽象基类：

-InputStream：字节输入流

-OutputStream：字节输出流

-Reader：字符输入流

-Writer：字符输出流

Reader方法：

-int read() 读取单个字符,返回该字符的编码值

-int read(char[] cbuf) 将字符读入数组，读取本次读取的字符个数

-int read(char[] cbuf, int off, int len) 将字符读入数组的某一部分。

-close

**打印文本：**

public void test03() throws IOException {

FileReader fr = new FileReader("1.txt");

char[] arr = new char[10];

while (true) {

int len = fr.read(arr);

if (len == -1) {

break;

}

System.out.print(Arrays.toString(arr));

}

fr.close();

}

}

Writer方法：

- void write(int c) 写入单个字符

- void write(char[] cbuf) 写入字符数组

- void write(char[] cbuf, int off, int len) 写入字符数组的某一部分

- void write(String str) 写入字符串

- void write(String str, int off, int len) 写入字符串的某一部分

- void flush() 刷新该流的缓冲，立马写入

public void test04() throws IOException {

Scanner input = new Scanner(System.in);

System.out.println("请输入一句话");

String str = input.nextLine();

FileWriter fw = new FileWriter("1.txt", true);

fw.write(str);

fw.close();

input.close();

}

}

Scanner:next遇到空白符认为输入结束；nextLine遇到换行符认为输入结束

**文件复制：**

public void copy(String srcFileName, String destFileName) throws IOException {

FileReader fr = new FileReader(srcFileName);

FileWriter fw = new FileWriter(destFileName);

char[] arr = new char[10];

int len;

while ((len = fr.read(arr)) != -1) {

fw.write(arr, 0, len);

}

fw.close();

fr.close();

}

}

**BufferedReader:缓存速度更快**

public void test05() throws IOException {

FileReader fr = new FileReader("1.txt");

BufferedReader bf = new BufferedReader(fr);

String str;

while ((str = bf.readLine()) != null) {

System.out.println(str);

}

fr.close();

bf.close();

}

}

BufferedReader增加了String readLine() 读取一行

BufferedWriter增加了void newLine() 写换行符，否则写入时会全在一行

**编码与解码：（字节字符在一起用）**

public void test06() throws IOException {

FileInputStream fis = new FileInputStream("1.txt");

InputStreamReader isr = new InputStreamReader(fis, "GBK");

}

}

* 1. 有不同的数据类型，用字节流FileOutputStream;再通过DataOutputStream包装转换成字节可以输出各种数据类型，但需要用DataInputStream读取，且读取与写入的数据顺序需要一致

需要用.dat文件保存

dos.writeInt dos.writeChar ....

dis.readInt dis.readChar ....

* 1. 输出对象的IO流：ObjectOutputStream oos.writeObject
     1. 把对象转为字节序列：序列化
     2. 需要实现Serializable接口 指定serialVersionUID版本ID
  2. 读取对象的IO流：ObjectInputStream ois.readObject（会返回对象）
     1. 反序列化
     2. 读取时需要创建对象来保存
  3. 如果某个属性不需要序列化，加transient修饰 在读取的时候只会显示默认值

若某个属性是static，那么也不需要序列化

序列化一个对象，若属性是引用数据类型，也需要实现Serializable接口

* 1. Externalizable 需要重写方法 可以定义需要序列化的属性及顺序

public class User implements Externalizable{

public String brand;

private String name;

private transient int sale;

private double price;

@Override

public void writeExternal(ObjectOutput out) throws IOException {

out.writeUTF(brand);

out.writeUTF(name);

out.writeDouble(price);

out.writeInt(sale);

}

@Override

public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {

brand = in.readUTF();

name = in.readUTF();

price = in.readDouble();

sale = in.readInt();

}

}

* 1. PrintStream： System.out 相当于Reader ps.println

New PrintStream(文件名，编码)

* 1. Scanner文件扫描仪 从指定的文件，流中读取文本数据

new Scanner(new FileInputStream(“1.txt”))

new Scanner(new File(“1.txt”, “GBK”))

* 1. System.setOut(new PrintStream(“1.txt”)) 指定输出位置
  2. 自动关闭资源：try(所有文件){处理代码}catch(){}
  3. // Path 接口（相当于File），Paths 工具类（用来创建Path接口的对象）

Path file = Paths.get("testIO","java", "1.txt"); 代替/ 作为分级

* 1. Files的API
     1. 复制文件copy

Path src = Paths.get("1.txt" );

Path dest = Paths.get("2.txt");

Files.copy(src, dest, StandardCopyOption.REPLACE\_EXISTING);

* + 1. -delete 文件不存在会报错

-deleteIfExists 存在就删除，不存在什么也不做

* + 1. -move 剪切
    2. -readAllLine 读取

Path file = Paths.get("testIO", "java", "2.txt");

List<String> allLines = Files.readAllLines(file, StandardCharsets.UTF\_8);

for (String string:

allLines) {

System.out.println(string);

1. 网络
   1. 网络通信三要素：
      1. 地址：IP地址，定位到一台设备
      2. 端口号：定位到一个应用程序（进程）
      3. 网络协议：数据传输的要求/标准
   2. OSI（Open System Interconnection）
      1. 物理层：建立、维护、断开物理连接 硬件
      2. 数据链路层：硬件寻址 用MAC地址访问介质 网卡
      3. 网络层：逻辑地址寻址 IPv4 IPv6
      4. 传输层：传输数据的协议端口号
      5. 会话层：建立、管理、终止会话 本地主机与远程主机的会话
      6. 表示层：数据的解析、压缩 显示图片等
      7. 应用层：使用服务 HTTP/FTP/SMTP
   3. TCP/IP协议：
      1. 主机-网路层：硬件
      2. 网络层：IP寻址
      3. 传输层：TCP/UDP
      4. 应用层：程序员需要面对的
   4. TCP：面向连接，可靠，字节流，大数据量

UDP（User Datagram Protocol）：非面向连接，不可靠，基于用户数据报（报文），64K以内

* 1. IP地址
     1. IPv4：32位，4个整数组成，每个整数1个字节 无符号范围0~255
     2. IPv6：128位，8个16进制组成
     3. 特殊：
        1. 127.0.0.1 本地回传IP地址
        2. 224.0.0.0~239.255.255.255 广播IP
  2. 端口号： 【0，65535】之间 无符号2个字节
     1. 公认端口：0~1023
        1. HTTP：80
        2. FTP：21
        3. SMTP：25
     2. 注册端口：1024~49151
        1. Tomcat：8080
        2. JBOSS：8080
        3. Oracle：1521
        4. MySQL：3306
        5. SQL Server：1433
        6. QQ：1080
     3. 动态/私有端口：49152~65535
  3. InetAddress 用类方法返回对象
  4. Socket：是两台机器间通信的端点，可以和网卡驱动进行交流，将数据交给网卡驱动，或者从网卡驱动中提取数据
     1. 流： TCP ServerSocket/Socket
        1. ServerSocket：接受、建立连接
        2. Socket：数据传输
     2. 数据报： UDP DatagramSocket
  5. 服务器步骤：
     1. 开启服务器，指定服务器监听的端口号
     2. 等待客户端并接受客户端连接 服务器接受客户端accept
     3. 接受/发送数据 客户端接受输出流get
     4. 断开连接
     5. 关闭服务器

public void testServer() throws IOException {

ServerSocket server = new ServerSocket(9999);

Socket socket = server.accept(); // 这句代码执行一次就接受一个客户端连接

OutputStream outputStream = socket.getOutputStream();

outputStream.write("欢迎光临".getBytes());

socket.close();

server.close();

}

* 1. 客户端步骤：
     1. 连接服务器 需要IP和port
     2. 发送/接受数据
     3. 断开连接

public void testClient() throws IOException {

Socket socket = new Socket("192.168.30.142", 9999);

OutputStream outputStream = socket.getOutputStream();

outputStream.write("你好".getBytes());

socket.shutdownOutput(); // 只关闭输出

// 接受数据

InputStream inputStream = socket.getInputStream();

byte[] data = new byte[1024];

int len;

while ((len = inputStream.read(data)) != -1) {

System.out.println(new String(data, 0, len));

}

socket.close();

}

* 1. 服务器通信

public void testServer01() throws IOException {

// 1.开启服务器

ServerSocket server = new ServerSocket(8989);

// 2.接受客户端

Socket socket = server.accept();

// 3.获取输入

InputStream in = socket.getInputStream();

// 因为是单词，需要转成字符流

InputStreamReader inputStreamReader = new InputStreamReader(in);

**// 按行读取：BufferedReader/Scanner**

BufferedReader bufferedReader = new BufferedReader(inputStreamReader);

OutputStream out = socket.getOutputStream();

**// 按行写：BufferedWriter(nextLine)/PrintStream(printLine)**

PrintStream printStream = new PrintStream(out);

String str;

// 4.反转

while ((str = bufferedReader.readLine()) != null) {

if ("bye".equals(str)) {

break;

}

**// StringBuilder可以反转字符串**

StringBuilder stringBuilder = new StringBuilder(str);

stringBuilder.reverse();

printStream.println(stringBuilder.toString());

}

socket.close();

server.close();

}

* 1. 客户端通信：

public void testClient01() throws IOException {

// 1. 创建服务器

Socket socket = new Socket("192.168.30.142", 8989);

// 2. 输出

Scanner scanner = new Scanner(System.in);

PrintStream printStream = new PrintStream(socket.getOutputStream());

// 3. 接受数据

while (true) {

System.out.println("请输入单词：");

String input = scanner.next();

printStream.println(input);

if ("bye".equals(input)) {

break;

}

}

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

String result = bufferedReader.readLine();

System.out.println(result);

// 4. 关闭

scanner.close();

socket.close();

}

* 1. 多客户端--多线程 每个客户端的socket是独立的，用构造器将socket加入线程中
  2. 接受文件步骤
     1. 接受客户端连接
     2. 接收文件名.后缀名
        1. 需要单独存储在文件夹中
        2. 文件重名：时间戳/唯一编码的UUID
     3. 接受文件内容
        1. 既接收文件名又接收文件内容：需要字节流
        2. 区别文件名和文件内容：DataInputStream

public void testServer02() throws IOException {

ServerSocket server = new ServerSocket(9999);

Socket socket = server.accept();

InputStream in = socket.getInputStream();

DataInputStream dataInputStream = new DataInputStream(in);

String fileName = dataInputStream.readUTF();

int index = fileName.lastIndexOf(".");

long time = System.currentTimeMillis();

String ext = fileName.substring(index);

String name = fileName.substring(0, index);

String newFileName = "upload/" + name + time + ext;

byte[] data = new byte[1024];

int len;

FileOutputStream fileOutputStream = new FileOutputStream(newFileName);

while ((len = dataInputStream.read(data)) != -1) {// 读入且返回

fileOutputStream.write(data, 0, len);

}

OutputStream out = socket.getOutputStream();

PrintStream printStream = new PrintStream(out);

printStream.println("文件接收完毕");

fileOutputStream.close();

socket.close();

server.close();

}

* 1. 发送文件：

public void testClient02() throws IOException {

Socket socket = new Socket("192.168.30.142", 9999);

Scanner input = new Scanner(System.in);

System.out.println("请输入文件绝对路径");

String fileName = input.nextLine();

File file = new File(fileName);

// 发送文件名

DataOutputStream dataOutputStream = new DataOutputStream(socket.getOutputStream());

dataOutputStream.writeUTF(file.getName());

// 先读取文件内容，再发送

FileInputStream fileInputStream = new FileInputStream(file);

byte[] data= new byte[1024];

int len;

while ((len = fileInputStream.read(data)) != -1) {

dataOutputStream.write(data, 0, len);

}

socket.shutdownOutput();// 表示发送完毕

// 接收返回结果

InputStreamReader inputStreamReader = new InputStreamReader(socket.getInputStream());

BufferedReader bufferedReader = new BufferedReader(inputStreamReader);

String result = bufferedReader.readLine();

System.out.println("结果：" + result);

socket.close();

fileInputStream.close();// 孤立的，与socket无关

* 1. 群聊客户端：

public class TestClient {

public static void main(String[] args) throws IOException, InterruptedException {

Socket socket = new Socket("192.168.30.142", 9999);

ReceiveThread receiveThread = new ReceiveThread(socket);

SendThread sendThread = new SendThread(socket);

receiveThread.start();

sendThread.start();

sendThread.join();

receiveThread.setFlag(false);

receiveThread.join();

socket.close();

}

}

class SendThread extends Thread {

private Socket socket;

public SendThread(Socket socket) {

super();

this.socket = socket;

}

public void run() {

Scanner input = new Scanner(System.in);

PrintStream printStream = null;

try {

printStream = new PrintStream(socket.getOutputStream());

} catch (IOException e) {

e.printStackTrace();

}

while (true) {

System.out.println("请输入要发送的信息");

String content = input.nextLine();

if ("bye".equals(content)) {

break;

}

printStream.println(content);

}

input.close();

}

}

class ReceiveThread extends Thread {

private Socket socket;

private boolean flag = true;

public ReceiveThread(Socket socket) {

super();

this.socket = socket;

}

public void setFlag(boolean flag) {

this.flag = flag;

}

public void run() {

try {

InputStreamReader inputStreamReader = new InputStreamReader(socket.getInputStream());

BufferedReader bufferedReader = new BufferedReader(inputStreamReader);

while (flag) {

String line = bufferedReader.readLine();

System.out.println(line);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

* 1. fileinputstream返回的是len,bufferedinputstream返回的是string

1. UDP
   1. DatagramSocket

DatagramPacket

* 1. 分发送端和接收端
  2. 一般是服务器向客户端推送消息
  3. 发送方步骤：
     1. 发送方建立DatagramSocket，IP和端口号都是自动获取的
     2. 将要传输的数据打包成DatagramPacket
     3. datagramSocket.send(datagramPacket)

public void testSend() throws IOException {

DatagramSocket datagramSocket = new DatagramSocket();

String str = "下课了";

byte[] data = str.getBytes();

InetAddress ip = InetAddress.getByName("192.168.30.142");

DatagramPacket datagramPacket = new DatagramPacket(data, 0, data.length, ip, 9999);

datagramSocket.send(datagramPacket);

System.out.println("发送完毕");

datagramSocket.close();

}

* 1. 接收方步骤：
     1. 接收方的DatagramSocket自动获取IP，但要指定端口号
     2. 准备一个存储数据的容器打包
     3. 接收数据包datagramSocket.receive(datagramPacket)

public void testReceive() throws IOException {

DatagramSocket datagramSocket = new DatagramSocket(9999);

byte[] data = new byte[1024];

DatagramPacket datagramPacket = new DatagramPacket(data, data.length);

datagramSocket.receive(datagramPacket);

byte[] dataReceived = datagramPacket.getData();

int finalLength = datagramPacket.getLength();

datagramSocket.close();

}

* 1. 类的加载：
     1. 加载：把字节码读取到内存
     2. 连接：
        1. 验证：确保符合JVM的规范：以cafe魔术开头，没有安全问题
        2. 准备：例如给类变量（静态变量）在方法区分配内存，并赋予默认值，若是final直接赋常量值
        3. 解析：将虚拟机常量池中的符合引用（常量名）替换为直接引用（地址）
     3. 初始化：
        1. 虚拟机启动，先初始化main方法所在的类
        2. 若父类未初始化，先初始化父类
        3. -new对象
        4. 使用类的静态变量或静态方法（静态常量不会触发类初始化）
        5. 使用reflect方法反射调用
     4. 不会进行初始化的情况：
        1. 引用的是静态常量 final static
        2. 当访问一个静态域（静态变量、静态方法）时，只有声明这个域的类才会初始化。

即当子类引用父类的静态属性、静态方法时，子类不会初始化

* + - 1. 用数组定义类的引用，不会导致类的初始化
  1. 类加载器：
     1. 引导类加载器：加载Java核心库
     2. 扩展类加载器：加载Java的扩展库
     3. 应用程序类加载器：项目路径bin文件夹下的字节码，以及如果你配置了环境变量classpath
     4. 自定义类加载器
        1. 字节码需要加密解密
        2. 字节码的路径不再常规路径，有自定义的特定路径 例如：tomcat
  2. Java中类加载器的双亲委托模式：（安全，各司其职，无继承关系）

组合方式：扩展类加载器是应用程序类加载器的父加载器，应用程序类加载器有成员变量parent

* 1. API
     1. -获取类的class对象：.class
     2. 获取类加载器的对象：getClassLoader
     3. 获取父加载器：getParent
  2. 接到加载某个类的任务后：
     1. 应用-扩展-引导 先搜索是否加载过，再提交给父加载器搜索
     2. 引导-扩展-应用 都未找到，传给子加载器在其负责范围内加载
     3. 仍未找到，报错ClassNotFoundException
  3. 类加载器的作用：
     1. 本质工作：加载类
     2. 顺便加载“类路径下”的资源文件

例如：src下（编译后对应bin）的资源配置文件：xxx.properties

public class TestClassLoader {

public void test01() throws IOException {

Properties properties = new Properties();

Class<TestClassLoader> clazz = TestClassLoader.class;

ClassLoader classLoader = clazz.getClassLoader();

InputStream in = classLoader.getResourceAsStream("jdbc.properties");

// 类加载器可以加载配置文件（通过流的方式）

// 在配置属性对象中加载数据流

properties.load(in);

}

}

在项目的根目录下，不再src里面：用fileinputstream加载

在src里面/在source folder文件夹（等价于src且不同于普通文件夹）中：用类加载器加载

1. Class类型：
   1. 所有的java类型（基本数据类型、引用数据类型、void）被加载到内存后，或是编译器自动生成的class字节码，最终都会用一个class对象来表示；

即所有的java类型在内存中都表示为class对象

* 1. 获取Class对象的方法
     1. 类名.class
        1. 优点：简洁
        2. 缺点：要求编译期间就存在，不在当前类路径下的类无法读取
     2. 对象.getClass()
        1. 在java.lang.Object类型中声明的，返回对象运行时类型
        2. 适用范围：必须有对象，引用数据类型可以，基本数据类型不可以
     3. Class.forName(“类型全名称”)

类型全名称：包.类名

适用场景：此类型可以在编译期间未知，可通过以下方式指定：

1. 直接在代码中显示
2. 键盘输入
3. 配置在配置文件中
   * 1. 类加载器对象.loadClass(“类型全名称”)

适用于用自定义类加载器对象来加载指定文件路径下的类

1. 反射的作用：区别“运行时”和“编译时”
   1. 在运行时能够获取任意类型的详细信息（在spring框架中管理很多自写的类）
      1. 步骤：
         1. 获取这个类的Class对象（通过配置文件load）
         2. 获取类信息
            1. getPackage
            2. getName
            3. getModifiers 修饰符
            4. getSuperclass
            5. getInterfaces 父接口们
            6. getFields 公共（public）的属性
            7. getDeclaredFields

为t的属性对象赋值：

①Field field = clazz.getDeclaredFields(“属性名”)

②field.setAccesible(true)

③field.set(t, 属性值)

* + - * 1. getConstructor
        2. getDeclaredConstructor
  1. 在运行时能够创建任意引用数据类型对象
     1. 步骤：
        1. 获取某个类型的Class对象
        2. 通过Class对象来创建这个类型所代表的实例对象（只能是无参构造）

public void test02() throws ClassNotFoundException, IllegalAccessException, InstantiationException {

Class<?> clazz = Class.forName("com.atguigu.ect.demo.AtGuiguDemo");

// obj编译时类型是Object

// 运行时类型是AtGuiguDemo

Object obj = clazz.newInstance();

}

1. 获取某个类型的Class对象
2. 通过Class对象来获取Constructor对象 getConstructor（形参列表）

形参列表通过配置文件获取

1. 通过Constructor对象来创建这个类型所代表的实例对象
   1. 在运行时可set,get属性值
      1. 用get的属性对象set
      2. parameterType:数据类型.class
   2. 运行时可以调用任意对象方法：
      1. Method.invoke
   3. 运行时读取某个注解信息：
      1. 获取Class对象
      2. 获取注解对象getAnnotation
      3. 获取注解的配置参数的值 .value
   4. 运行时获取泛型类型实参
      1. 获取Class对象 this.getClass可以获取子类（多态创建对象包括子类）
      2. 获取泛型父类 (ParameterizedType)getGenericSuperClass 返回Type

泛型不能表示为Class对象

Type：代表Java的所有类型

1. Class：普通类型，没有泛型信息
2. ParameterizedType：Father<Integer,String>
3. GenericArrayType：T[]
4. TypeVariable：T
5. WildcardType：通配符 ArrayList<?>
   * 1. 获取类型实参：getActualTypeArguments
   1. 一切皆对象：（在内存中）
      1. 所有的类都是class对象
      2. 所有的属性都是field对象

属性有相同的特征，所有抽象为Field类

* + 1. 所有的构造器都是Constructor对象
    2. 所有方法都是Method对象
  1. Java语言是静态语言：

如果没有反射，必须在编译期间确定所有类型。

为了让Java也支持动态性，在运行期间确定某个类型的功能，才引入反射机制。准动态语言。

只是延迟了确定对象的类型而已。

* 1. 静态语言：安全，效率高

动态语言：灵活 JavaScript Python

1. 工厂模式
   1. 目的是分工：把生产（创建）对象与使用对象分开，解耦合
   2. 在工厂类中写一个创建对象的方法（一个工厂多个产品）

将产品名称用class.forName表示

* 1. 工厂接口，增加新产品，不需要修改原来的工厂类（每个产品一个工厂）

1. 代理模式
   1. 适用场景：当前这个类要完成的任务代码是多变的（可要可不要），或是重复的
   2. 主题接口：要求代理类和被代理类实现同一个接口
   3. 每个对象不同的代码内容，在代理中传入对象来执行

代理者持有被代理者的引用，因为核心的业务逻辑仍然交给被代理者自己完成

* 1. 多个主题（接口）需要代理：动态代理
     1. 用类继承IncocationHandler接口并重写invoke方法
     2. 有反射：方法对象.invoke（被代理对象，args实参列表）

1. Lambda
   1. 特征：接口，SAM（single abstract method）

例如：Runnable Comparable

* 1. SAM多种接口：
     1. 消费型接口：Consumer 有参无返回值 void accept(T t)
     2. 供给型接口：Supplier 无参有返回值 T get()
     3. 判断型接口：Predicate boolean test(T t)
     4. 功能型接口：Function 有参有返回值 R apply(T t)
        1. Unary开头的，表示一元的，泛型的类型只有一个，形参和返回值都是同一种类型
  2. Lambda表达式替代了原来使用匿名内部类的对象给函数式接口（SAM接口）的变量或形参赋值的形式
  3. SAM接口的抽象方法只有一个，因此方法签名可以省略一部分

例如：修饰符，返回值类型，方法名，但是形参列表不可省，因为形参在方法体中需要继续使用

Lambda语法：

（形参列表）-> {方法体}

优化：

1. 如果方法体只有一句可以不用加 大括号 和 ；
2. 如果（形参列表）中的形参类型已知，获取可以推断，那么数据类型可以省略
3. 如果（形参列表）只有一个形参且数据类型省略，（） 可以省略
4. 方法体只有一句语句且是return语句，return可以省略
5. 如果没有形参，（）不可省略
   1. 方法引用是用来简化lambda表达式的

条件：

1. 当方法体的实现是通过调用一个现有的方法来完成功能时
2. 函数式接口的抽象方法的形参列表与返回值与该方法的形参列表和返回值需要相同

或者

多个形参是调用的对象和方法的实参列表 如s1.compareTo(s2)

方法引用的格式：

1. 对象：：实例方法名
2. 类名：：静态方法名
3. 类名：：实例方法名
   1. 构造器引用：

条件：

1. 当方法体的实现是通过创建一个对象来实现的
2. 要求函数式接口抽象方法的返回值类型和该对象的类型一样
3. 要求函数式接口抽象方法的形参列表与该对象创建时的构造器的形参列表对应

构造器引用语法：

1. 类名：：new
2. 数组类型：：new
3. StreamAPI

特点：

* + 1. Stream本身只是处理数据，不负责储存，储存数据的是集合、数组等数据结构与容器
    2. Stream是不可变的，一旦修改，就会产生新的Stream对象
    3. Stream的操作是一个延迟操作，都要延迟到最后的终结时一起处理

1. 创建Stream：
   1. 通过集合： 集合对象.stream()
   2. 通过数组类Arrays Arrays.stream(数组对象)
   3. Stream接口of方法，产生有限流 Stream.of(......)
   4. Stream接口的静态方法，产生无限量
      1. generate
      2. Iterate
   5. 加工操作
      1. -filter
      2. -distinct
      3. -limit
      4. -skip
      5. -peek 接受Lambda表达式，对流中的每个数据进行操作

可以在peek中进行打印操作 System.out::println

* + 1. -sorted(Comparator com)
    2. -map 接受Lambda表达式，对流中的每个数据进行操作，且返回新的流
    3. -flatmap 相比于map是想将流中的数据压扁成一个个Stream，再将其组成一个大流
  1. 终结操作：
     1. -void forEach
     2. -long count
     3. -boolean allMatch

-boolean anyMatch

-boolean noneMatch

* + 1. -Optional<T> findFirst

-Optional<T> findAny

Optional包装类不会有空指针异常，若是null会返回empty

Optional实际上是一个容器，但不同于数组和集合，Optional只能装一个对象

* + - 1. 创建Optional对象的方法：
         1. Optional.of(xx) 只能装非空对象
         2. Optional.ofNullable(xx) 可以null
         3. Optional<T> empty() 直接返回一个空箱子对象
      2. 取出Optional容器中的对象
         1. -get() 配合of(xx)使用，不能为空
         2. -orElse(T other) 若对象为空，用other对象代替
         3. -orElseGet(Supplier<？ extends T> other)
         4. -orElseThrow
      3. 其他：
         1. Optional<T> filter(Predicate)

1. Mysql

**进入方式：**

mysql -hlocalhost -P3306 -uroot -p

h是host u是user p是password

**说明：**

-如果连接的是本地的mysql，-hlocalhost可省略

-如果连接的mysql的端口号是默认的3306，-P3306可省略

**可视化工具：**

SQLyog,Navicate

**数据类型：**

* 1. 整数系列：

Tinyint：类似于byte

Smallint：类似于short

Mediumint：

Int,integer

Bigint：类似于long

使用int类型：

1. int等价于int(11)
2. Int(M) 指定宽度，但也可以存储超过此范围的数字

-unsigned:表示无符号，即没有负数

-zerofill:用0填充

* 1. 浮点小数：

-float：4个字节

-double：8个字节

-float(M,D) double(M,D) M表示总位数（包括小数位数），D表示精度（小数点后几位）

* 1. 定点小数： DECIMAL(M,D) 底层用字符串存储
  2. 日期时间：

-date

-time

-datetime

-year

-timestamp：时间戳

datetime和timestamp的区别：

-timestamp存储的是毫秒，在比较和计算时效率比较高

-两者表示的时间范围不同

* 1. 字符串：mysql没有单字符类型，只有字符串类型且用char表示

-char：定长字符，长度【1,255】 char默认为char(1)

-varchar：varchar(M) 最长不超过M个字符

-text：存储文字较多的简介等，且不经常检索的信息

char和varchar的区别：

1. 内存占用空间不同：

-char（M）固定M个字符空间 ，空字符储存的是\u0000

-varchar实际字符数+1~2个字节（用于记录实际的字符个数）

1. 速度不同：

varchar每次存取都要数个数，速度慢

* 1. 少见类型：

-位类型

-blob类型

-枚举：多个中选择一个

-集合：多个中选择多个

**常用语句：**

-show databases;

-use 数据库名称;

-create database 数据库名称;

-show tables;

-select \* from 表名称;

-set names gbk; 告知服务器客户端的字符编码，修改服务器端的变量

查询：

-select salary from t\_employee;

-select \* from t\_employee where salary+1000 > 20000;

修改：

-update t\_employee set salary = salary + 1000;

**运算符：**

* + 1. **算术运算符运算符：**

- /：可以保留小数部分

-div：若是整数相除，只保留整数部分

* + 1. **比较运算符：**

**- =**：不同于java

- <=>：安全等于 用于比较null

- is null, is not null

* + 1. **逻辑运算符：**

**-**&& and

-|| or

-! not

-^ xor

**范围：**

-区间范围：between a and b; not between a and b;

-集合范围：in(...); not in(...);

**模糊查询：(用单引号表示字符)**

-like ‘%x%’： %表示0~n个字符

-like ‘\_x%’： \_表示1个字符

**SQL语句的规范：**

* 1. **区分大小写的校对规则：**

-ci：不区分大小写 case ignore

-cs：区分大小写

-bin：最严格区分大小写，以二进制方式存储 binary

* 1. 在给字段区别名的时候，要用双引号“”

-select xxx as “xxxx”

* 1. 注释：#，/\* \*/
  2. 同级不可重名
  3. 若表明或字段名和关键字重名，用飘号来引起来 `xxx`

**SQL语言分类：**

-DDL:Data Define Language show,use,create,drop,alter 定义数据结构

-DML:Manage insert,delete,update,select

-DCL:Control grant,revoke,commit,rollback 权限事务控制

* 1. DDL：

-查看某个数据库下的表格：

1. 先use，再show tables
2. -show tables from 数据库名

-创建数据库：

-create database 数据库名 charset ‘utf8’ 不是utf-8

-创建表格

1. -create table 【数据库名.】表名称（字段名1 数据类型，字段名2 数据类型....）
2. 说明：若已用use打开了数据库，可直接用当前路径的表格

-查看表结构：

1. -desc 表名称 description 类似于通过class查看类的信息

-修改表名称：

1. -rename table 旧表名 to 新表名
2. -alter table 旧表名 rename 新表名

-修改表结构：

1. 增加一个字段

-alter table 表名称 add 【column】 字段名 数据类型; 默认添加到最后

-alter table 表名称 add 【column】 字段名 数据类型 first;

-alter table 表名称 add 【column】 字段名 数据类型 after 另一个字段;

1. 修改字段的数据类型或位置

-alter table 表名称 modify 【column】 字段名 新的数据类型；

-alter table 表名称 modify 【column】 字段名 数据类型 first；

-alter table 表名称 modify 【column】 字段名 数据类型 after 另一个字段；

1. 修改列的名称

-alter table 表名称 change 【column】 旧字段名 新字段名 数据类型；

1. 删除一列

-alter table 表名称 drop 【column】 字段名；

-删除整张表：drop table 表名称;

* 1. DML:

-增：

-insert into 表名称 values（值列表1），（值列表2）

-insert into 表名称（字段列表） values（值列表1），（值列表2）

-删：

-delete from 表名称 【where 条件】

-改：

-update 表名称set 字段名1=新值，字段名2=新值

-查：

-select \* as “别名” from 表名

**导出数据库：mysqldump -hlocalhost -P3306 -uroot -pQ1184030745. --database test > d:/test.sql**

**导入数据库：source d:/test.sql**

**关系型数据库设计规则：**

* + 1. 遵循ER模型：

-E：entity 实体，对应到数据库中的一张表 类似于java程序中的bean设置

-R：relationship

* + 1. 三范式：

-列不能拆分，即一列只能表示一种字段属性

-唯一标识，即两行记录不能完全相同

-关系引用主键（至少是唯一键）

* + 1. 具体体现：
  1. 索引：相当于书的目录页，是需要单独占用数据库的存储空间的。如果数据量很大需要建立索引，查询速度变快，修改速度变慢（主键、唯一键、外键列上自动创建索引）
  2. 主键primary key非空且唯一，创建主键自动创建索引，删除索引自动删除索引
     1. 建表之后指定主键约束：alter table 表名称 add primary key(字段列表)；
     2. 若单个无法唯一，但组合是唯一的，称为复合主键约束
     3. 删除主键约束：drop primary key;
  3. 唯一键unique key：可以有多个且可为null；创建唯一键自动建立索引，删除唯一键约束的方式是通过删除对应索引实现的
     1. 删除唯一键约束：alter table 表名称 drop index 索引名;
     2. 查看索引名：show index from 【数据库名.】表名称;
  4. 非空约束not null：非空约束，针对某个字段
     1. 建表后使字段变成非空：-alter table 表名称 modify 字段名 数据类型 not null;
     2. 建表后使字段可以为空：-alter table 表名称 modify 字段名 数据类型;
  5. 默认约束default：
     1. 建表后多约束：-alter table 表名称 modify 字段名 数据类型 default 默认值 【not null】;

-modify是全部重置约束条件

* 1. 自增约束的条件：
     1. 一个表只有一个自增变量
     2. 只能是整数列
     3. 必须是键列，一般是主键 primary key auto\_increment
     4. 指定值若是0或null，按照自增
  2. 外键约束foreign key：父不能减，子不能增
     1. 约束的等级：父表有主动权（子表对于父表修改删除的反应）

-cascade：级联

-set null

-no action

-restrict：父表不能修改、删除已被引用的字段

-set default

* + 1. 语法：【constraint 外键约束名】foreign key(从表的某字段) reference 主表名(被参考字段) 【on update 等级】 【on delete 等级】

-命名规则：主表名\_从表名\_关联字段名\_fk

* + 1. 建表后指定外键：alter table 表名称 add foreign key(从表的某字段) reference 主表名(被参考字段) 【on update 等级】 【on delete 等级】
    2. 删除外键：alter table 表名称 drop foreign key 约束名
    3. 查看某个表的约束：show create table 表名;

**字符串函数：**

**-**length：求字节数，一个汉字3个字节

-char\_length：求字符数

-concat：拼接

-concat\_ws（分隔符，str1,str2...）

-upper

-lower

-left(s,n)

-right(s,n)

-ltrim

-rtrim

-trim(both/leading/trailing ‘str’ from)

-truncate：截断，确定小数点长度

**加密函数：**

-password

-md5

**多表查询的连接关系：**

-内连接：A∩B

-select 字段列表

from A表名 inner join B表名

On 关联关系

Where 其他条件

-关联条件个数=表数-1 如果少了关联条件，就会出现笛卡尔积（A表数\*B表数）

-左连接left join：A A-A∩B

-select 字段列表

from A left join B

on 关联关系

where 关联字段 is null and 其他条件等

-右连接right join：同左连接，只是相对方向不同

-全连接full join：

-A ∪ B:A union B

-A∪B-A∩B:A - A∩B union B-A∩B

**Select语句有6大子句：**

1. From：后面跟表，视图，多行多列的二维表结构 筛选的对象
2. Where：跟条件
3. Group by ：分组字段列表
4. Having：跟条件，在此对于分组后的统计结果进行筛选
5. Order by：跟字段或者表达式 DESC（降序）

-可多个排序方式：

查询每个部门的员工人数，按照人数升序，若人数相同，按照did降序

Select did, count(\*)

From t\_employee

Group by did

Order by count(\*) ASC, did DESC

1. Limit：limit m, n 从m+1开始取，取n个（分页查询）

假设page代表第几页，nums表示每页需要显示的数量

Limit (page-1)\*nums, nums

**自连接：通过给表取别名的方式，将一张表虚拟成两张表**

**查询学习的顺序：**

**-**简单查询

-联合查询

-select的6个子句

-子查询

**子查询：查询嵌套**

* + - 1. Where
         1. 单值结果：比较运算符
         2. 多值结果：in(),all(),any()
      2. From

-对于查询的多行多列的二维表结构的临时表取别名，给分组统计的结果取别名

* + - 1. Exists

**事务：**

-结束的操作：commit,rollback（DCL语句）

rollback只对insert，update,delete（增删改查）有效，对于create,drop等DDL语句无效

-事务的属性：ACID

1. Atomicity：同一事务不可分割，要么全部发生完，要么不发生
2. Consistency：多个事务同时发生的，不可只进行其中几个，需要从一个一致性状态变换到另一个一致性状态
3. Isolation：各个事务之间的执行互不干扰
4. Durability：只要事务被提交，则改变是永久性的

**默认自动提交，手动提交方式：**

1. set autocommit = false; 。。。。。set autocommit = true;
2. Start transaction; 。。。。。。 rollback;commit;

**不同隔离级别：**

1. read uncommitted
2. Read committed
3. Repeatable read：行锁
4. Serializable：表锁

**几种问题：**

1. 脏读：还未提交，其他的就可以看到了
2. 不可重复读：提交前后的查询结果不同
3. 幻读：只针对记录数有增减的修改查询前后出现不同，好像是幻觉

**查询隔离等级：select @@tx\_isolation**

**修改隔离级别：set tx\_isolation = ‘read-uncommitted’**

**用delete和truncate删除整张表的区别：**

-truncate速度快

-truncate无法回滚

因为truncate底层是将表drop掉，然后新建了一张空表

delete底层是将数据一行一行的删除

**身份认证：**

**-**root@localhost

-root @%：任意机器上

[-root@192.168.11.56](mailto:-root@192.168.11.56)

-joe@%：user@host

**查看账户权限：show grants for user@host**

**JDBC编写步骤：**

1. 注册驱动（3步）
   1. 把驱动jar包放到项目的libs中（新建一个folder）
   2. 把jar包添加到build path 中
   3. Class.forname
2. 连接数据库
3. 操作数据库

-增删改：Int len = statement.executeUpdate

-查：ResultSet rs = statement.executeQuery

1. 关闭资源

// 1. 注册驱动

Class.forName("com.mysql.jdbc.Driver");

// 2. 连接数据库

String url = "jdbc:mysql://localhost:3306/test";

Connection connection = DriverManager.getConnection(url, "root", "Q1184030745.");

// 3. 编写SQL语句

String sql = "INSERT INTO t\_stu VALUES (2, 'lin')";

// 4. 准备statement

Statement statement = connection.createStatement();

// 5. 执行SQL

int len = statement.executeUpdate(sql);

System.out.println(len>0?"添加成功":"添加失败");

ResultSet resultSet = statement.executeQuery(sql);

while (resultSet.next()) {

int sid = resultSet.getInt("sid");

String sname = resultSet.getString("sname");

System.out.println(sid + "\t" + sname);

}

// 6. 关闭资源

resultSet.close();

connection.close();

statement.close();

**出现以下问题，需要换成子接口PreparedStatement：**

-SQL拼接

-SQL注入

-无法处理blob等二进制类型的数据

// 1. 注册驱动

Class.forName("com.mysql.jdbc.Driver");

// 2. 连接数据库

String url = "jdbc:mysql://localhost:3306/test";

Connection connection = DriverManager.getConnection(url, "root", "Q1184030745.");

// 3. 编写SQL语句

// String sql = "INSERT INTO t\_stu VALUES (2, 'lin')";

String sql = "insert into t\_depart values(null, ?, ?,?)";//将所有的参数改为?

// 4. 准备statement

// Statement statement = connection.createStatement();

PreparedStatement preparedStatement = connection.prepareStatement(sql);

// 设置?的值

preparedStatement.setString(1, "dname");

preparedStatement.setString(2, "desc");

preparedStatement.setBlob(3, new FileInputStream("d:/img/15.jpg"));

// 5. 执行SQL

// int len = statement.executeUpdate(sql);

int len = preparedStatement.executeUpdate();

System.out.println(len>0?"添加成功":"添加失败");

ResultSet resultSet = preparedStatement.executeQuery(sql);

while (resultSet.next()) {

int sid = resultSet.getInt("sid");

String sname = resultSet.getString("sname");

System.out.println(sid + "\t" + sname);

}

// 6. 关闭资源

resultSet.close();

connection.close();

preparedStatement.close();

**总结：**

**-connection相当于网络编程中的socket 获取连接**

**-statement和preparedstatement相当于socket读取的inputstream和outputstream**

**-执行sql语句时，statement对象把sql语句发给服务器，服务器执行完成后再将结果返回**

**URL：Universal Resource Locator**

**-**标准格式：协议：//主机名：端口号/资源路径

-数据库格式：jdbc:mysql://localhost:3306/test

**返回自增长的键值：**

PreparedStatement preparedStatement = connection.prepareStatement(sql,Statement.RETURN\_GENERATED\_KEYS);

ResultSet generatedKeys = preparedStatement.getGeneratedKeys();

mysql通过结果集ResultSet返回

**批处理：**

String url = "jdbc:mysql://localhost:3306/test?rewriteBatchedStatements=true";

String url = "jdbc:mysql://localhost:3306/test?rewriteBatchedStatements=true

&useUnicode=true&characterEncoding=utf8"; 设置字符编码 多个参数用 & 连接

preparedStatement.addBatch(); 底层有一个缓冲区，先缓冲所有需要执行的sql

preparedStatement.executeBatch(); 在循环外部执行batch

类似于网络编程中将流批量buffer

**处理事务：**

connection.setAutoCommit(false);

connection.commit();

connection.rollback();

connection.setAutoCommit(true);

**数据库连接池：DataSource**

**-**问题：

-数据库的连接相当于TCP/IP协议中的socket，每个客户端每次连接都有一个独立的socket和线程

-每次用完释放，成本高

-在线人数多时，并发操作多，导致服务器压力大，

-如何解决：

数据库连接池：提前准备一些连接，可重复使用

*参数：*

-初始化连接数

-最大连接数

-每次增量

-如果达到100个，会怎么样：无限制等待；等待xx秒返回异常，告知客户端连接超时

-使用Druid数据库连接池方法：

-手动创建DruidDtaSource，用set配置参数

-在配置文件中写入配置参数

必备参数：

-url

-username

-password

-driverClassName

-initialSize

-maxActive

-maxWait

可选参数：

-maxIdle

-minIdle：保证池中在释放掉一部分连接后，依然有保留几个连接以备不时之需

用数据连接池对象获取连接：

Properties properties = new Properties();

// 加载资源文件

properties.load(TestDataSource.class.getClassLoader().getResourceAsStream("druid.properties"));

DataSource dataSource = DruidDataSourceFactory.createDataSource(properties);

Connection connection = dataSource.getConnection();

connectionThreadLocal = new ThreadLocal<Connection>();

**ThreadLocal:可以设置当前线程保存的共享变量（只要不关闭，线程可以多次复用一个Connection）**

public static Connection getConnection() throws SQLException {

Connection connection = connectionThreadLocal.get();

if (connection == null) {

Connection connection1 = dataSource.getConnection();

connectionThreadLocal.set(connection1);

return connection1;

}

return connection;

}

用户在视图层输入输出；程序员在业务逻辑层处理复杂业务逻辑；最后的是数据库DB层

为了防止数据库迁移，在业务层和DB层之间增加一个DAO层作为接口连接。

DAO：Data Access Object

**MetaData:**

**如果属性名和数据库中的字段名不一致了，在sql中用别名表示**

public T getBean(String sql) throws IllegalAccessException, InstantiationException, SQLException {

T o = (T) type.newInstance();

Connection connection = JDBCToolsV1.getConnection();

PreparedStatement preparedStatement = connection.prepareStatement(sql);

ResultSet resultSet = preparedStatement.executeQuery();

// 元数据指的是列表的属性名

ResultSetMetaData metaData = resultSet.getMetaData();

int columnCount = metaData.getColumnCount();

// Field declaredField = type.getDeclaredField(metaData.getColumnName("did"));

// ColumnName是数据库中的字段名

// ColumnLabel可以是别名，要获取javabean中的属性名，需要在sql中给属性名设定与代码中相同的别名

Field declaredField = type.getDeclaredField(metaData.getColumnLabel("did"));

declaredField.setAccessible(true);

declaredField.set(type, resultSet.getObject(1));

}

出现问题：double类型不可以设定为null

解决方法：跟数据库对象相关的类的属性设置——将数据类型全部改成包装类，即double->Double

**Apache Commons DbUtils:**

**QueryRunner:**

public List<Map<String, Object>> getMapList(String sql, Object... args) throws SQLException {

return queryRunner.query(JDBCToolsV1.getConnection(), sql, new MapListHandler(), args);

}

public Map<Integer, Double> getAvgPerDepartment() throws SQLException {

String sql = "select did, avg(salary) from t\_employee group by did";

Map<Integer, Double> integerDoubleHashMap = new HashMap<>(); // 容器

List<Map<String, Object>> mapList = getMapList(sql);

for (Map<String, Object> map:

mapList) {

Integer did = (Integer) map.get("did");

Double avgSalary = (Double) map.get("avg(salary)");

integerDoubleHashMap.put(did, avgSalary);

}

return integerDoubleHashMap;

}

QueryRunner中ResultSetHandler的接口：

-BeanHandler：将结果集中的第一行数据封装到JavaBean实例中

-BeanListHandler：将结果集中的每一行数据封装到JavaBean实例中，存放到List里

-ScalarHandler：查询单值对象

-MapHandler：将结果集中的第一行数据封装到Map中，key是列名，value是对应的值

-MapListHandler：将结果集中的每一行数据封装到Map中，存放到List里

* 1. -
  2. -
  3. -
  4. -

**开发设置：**

-bin中放class文件

-src中放java文件

-module类似于工作空间

**快捷键：**

快速main()方法：psvm

快速输出：sout

内容补全：ctrl +alt+space

创建对象左边的变量直接：ctrl+alt+v

定义类方法：alt+insert

重写方法，直接输入方法名，会自动补全

查看源代码:Ctrl+点击类名

跳类名：Alt+↑

跳文件：Alt+↓

Surround with：Alt+Ctrl+T

全部缩起来：shift+ctrl+-

将某部分重复代码封装refactor变成方法：CTRL + ALT +M

跳转到方法：CTRL + B

看整个代码结构：ALT+7

查看需要的参数：CTRL+P

-通用快捷键

-反撤销：Ctrl+ Y

**其他：**

-分工：解耦合

-项目的包：utils,test,bean,view,service（存储数组list）

-圆周率：Math.PI

**项目中常出现的bug：**

-有循环结果的代码块，要注意将创建的容器对象放在循环外面