第十一章：常用类的概述和使用

11.1 常用的包（熟悉）

11.1.1 包的名称和功能

- java.lang包 - 该包是Java语言的核心包，并且该包中的所有内容由Java虚拟机自动导入。

  如：System类、String类、...

- java.util包 - 该包是Java语言的工具包，里面提供了大量工具类以及集合类等。

  如：Scanner类、Random类、List集合、...

- java.io包 - 该包是Java语言中的输入输出包，里面提供了大量读写文件相关的类等。

  如：FileInputStream类、FileOutputStream类、...

- java.net包 - 该包是Java语言中的网络包，里面提供了大量网络编程相关的类等。

  如：ServerSocket类、Socket类、...

- java.sql 包 - 该包是Java语言中的数据包，里面提供了大量操作数据库的类和接口等。

  如：DriverManager类、Connection接口、...

- ... ...
- Java程序员在编程时可以使用大量类库，因此Java编程时需要记的很多，对编程能力本身要求不是特别的高。

11.2 Object类的概述（重点）

11.2.1 基本概念

- java.lang.Object类是Java语言中类层次结构的根类，也就是说任何一个类都是该类的直接或者间接子类。
- 如果定义一个Java类时没有使用extends关键字声明其父类，则其父类为 java.lang.Object 类。
- Object类定义了“对象”的基本行为, 被子类默认继承。

11.2.2 常用的方法

方法声明	功能介绍
Object()	使用无参方式构造对象
boolean equals(Object obj)	用于判断调用对象是否与参数对象相等。 该方法默认比较两个对象的地址是否相等，与 == 运算符的结果一致 若希望比较两个对象的内容，则需要重写该方法。 若该方法被重写后，则应该重写hashCode方法来保证结果的一致性。
int hashCode()	用于获取调用对象的哈希码值(内存地址的编号)。 若两个对象调用equals方法相等，则各自调用该方法的结果必须相同 若两个调用对象equals方法不相等，则各自调用该方法的结果应该不相同。 为了使得该方法与equals方法保持一致，需要重写该方法。
String toString()	用于获取调用对象的字符串形式 该方法默认返回的字符串为：包名.类名@哈希码值的十六进制 为了返回更有意义的数据，需要重写该方法 使用print或println打印引用或字符串拼接引用都会自动调用该方法
Class<?> getClass()	用于返回调用对象执行时的Class实例，反射机制使用

案例题目：

编程实现Student类的封装，特征：学号(id)和姓名，要求提供打印所有特征的方法。
编程实现StudentTest类，在main方法中使用有参方式构造两个Student类型的对象并打印特征。

题目扩展:

如何实现以姓名作为基准判断两个对象是否相等？以及以学号和姓名同时作为基准判断两个对象是否相等？

11.3 包装类（熟悉）

11.3.1 包装类的概念

通常情况下基本数据类型的变量不是对象，为了满足万物皆对象的理念就需要对基本数据类型的变量进行打包封装处理变成对象，而负责将这些变量声明为成员变量进行对象化处理的相关类，叫做包装类。

如：

```
Person p = new Person();

int  num = 10;
```

11.3.2 包装类的分类

包装类	对应的基本类型
java.lang.Byte	byte
java.lang.Short	short
java.lang.Integer	int
java.lang.Long	long
java.lang.Float	float
java.lang.Double	double
java.lang.Boolean	boolean
java.lang.Character	char

11.3.3 Integer类的概述

（1）基本概念

java.lang.Integer类内部包装了一个int类型的变量作为成员变量，主要用于实现对int类型的包装并提供int类型到String类之间的转换等方法。

（2）常用的常量

常量类型和名称	功能介绍
public static final int MAX_VALUE	表示int类型可以描述的最大值，即2^31-1
public static final int MIN_VALUE	表示int类型可以描述的最小值，即-2^31
public static final int SIZE	表示int类型采用二进制补码形式的位数
public static final int BYTES	表示int类型所占的字节个数
public static final Class TYPE	表示int类型的Class实例

（2）常用的方法

方法声明	功能介绍
Integer(int value)	根据参数指定的整数来构造对象（已过时）
Integer(String s)	根据参数指定的字符串来构造对象（已过时）
int intValue()	获取调用对象中的整数值并返回
static Integer valueOf(int i)	根据参数指定整数值得到Integer类型对象
boolean equals(Object obj)	比较调用对象与参数指定的对象是否相等
String toString()	返回描述调用对象数值的字符串形式
static int parseInt(String s)	将字符串类型转换为int类型并返回
static String toString(int i)	获取参数指定整数的十进制字符串形式
static String toBinaryString(int i)	获取参数指定整数的二进制字符串形式
static String toHexString(int i)	获取参数指定整数的十六进制字符串形式
static String toOctalString(int i)	获取参数指定整数的八进制字符串形式

（3）装箱和拆箱的概念

在Java5发布之前使用包装类对象进行运算时，需要较为繁琐的“拆箱”和“装箱”操作；即运算前先将包装类对象拆分为基本类型数据，运算后再将结果封装成包装类对象。

从Java5开始增加了自动拆箱和自动装箱的功能。

（4）自动装箱池

在Integer类的内部提供了自动装箱池技术，将-128到127之间的整数已经装箱完毕，当程序中使用该范围之间的整数时，无需装箱直接取用自动装箱池中的对象即可，从而提高效率。

11.3.4 Double类的概述

（1）基本概念

java.lang.Double类型内部包装了一个double类型的变量作为成员变量，主要用于实现对double类型的包装并提供double类型到String类之间的转换等方法。

（2）常用的常量

常量类型和名称	功能介绍
public static final int SIZE	表示double类型的二进制位数
public static final int BYTES	表示double类型的字节个数
public static final Class TYPE	表示double类型的Class实例

（3）常用的方法

方法声明	功能介绍
Double(double value)	根据参数指定的浮点数据来构造对象（已过时）
Double(String s)	根据参数指定的字符串来构造对象（已过时）
double doubleValue()	获取调用对象中的浮点数据并返回
static Double valueOf(double d)	根据参数指定浮点数据得到Double类型对象
boolean equals(Object obj)	比较调用对象与参数指定的对象是否相等
String toString()	返回描述调用对象数值的字符串形式
static double parseDouble(String s)	将字符串类型转换为double类型并返回
boolean isNaN()	判断调用对象的数值是否为非数字

- **扩展：**

  java.lang.Number类是个抽象类，是上述类的父类来描述所有类共有的成员。

11.3.5 Boolean类的概述

（1）基本概念

java.lang.Boolean类型内部包装了一个boolean类型的变量作为成员变量，主要用于实现对boolean类型的包装并提供boolean类型到String类之间的转换等方法。

（2）常用的常量

常量类型和名称	功能介绍
public static final Boolean FALSE	对应基值为false的对象
public static final Boolean TRUE	对应基值为true的对象
public static final Class TYPE	表示boolean类型的Class实例

（3）常用的方法

方法声明	功能介绍
Boolean(boolean value)	根据参数指定的布尔数值来构造对象（已过时）
Boolean(String s)	根据参数指定的字符串来构造对象（已过时）
boolean booleanValue()	获取调用对象中的布尔数值并返回
static Boolean valueOf(boolean b)	根据参数指定布尔数值得到Boolean类型对象
boolean equals(Object obj)	比较调用对象与参数指定的对象是否相等
String toString()	返回描述调用对象数值的字符串形式
static boolean parseBoolean(String s)	将字符串类型转换为boolean类型并返回

11.3.6 Character类的概述

（1）基本概念

java.lang.Character类型内部包装了一个char类型的变量作为成员变量，主要用于实现对char类型的包装并提供字符类别的判断和转换等方法。

（2）常用的常量

常量类型和名称	功能介绍
public static final int SIZE	表示char类型的二进制位数
public static final int BYTES	表示char类型的字节个数
public static final Class TYPE	表示char类型的Class实例

（3）常用的方法

方法声明	功能介绍
Character(char value)	根据参数指定的字符数据来构造对象（已过时）
char charValue()	获取调用对象中的字符数据并返回
static Character valueOf(char c)	根据参数指定字符数据得到Character类型对象
boolean equals(Object obj)	比较调用对象与参数指定的对象是否相等
String toString()	返回描述调用对象数值的字符串形式
static boolean isUpperCase(char ch)	判断参数指定字符是否为大写字符
static boolean isLowerCase(char ch)	判断参数指定字符是否为小写字符
static boolean isDigit(char ch)	判断参数指定字符是否为数字字符
static char toUpperCase(char ch)	将参数指定的字符转换为大写字符
static char toLowerCase(char ch)	将参数指定的字符转换为小写字符

11.3.7 包装类（Wrapper）的使用总结

- 基本数据类型转换为对应包装类的方式

  调用包装类的构造方法或静态方法即可
- 获取包装类对象中基本数据类型变量数值的方式

  调用包装类中的xxxValue方法即可
- 字符串转换为基本数据类型的方式

  调用包装类中的parseXxx方法即可

11.4 数学处理类（熟悉）

11.4.1 Math类的概述

（1）基本概念

java.lang.Math类主要用于提供执行数学运算的方法，如：对数，平方根。

（2）常用的方法

方法声明	功能介绍
static int max(int a, int b)	返回两个参数中的最大值
static int min(int a, int b)	返回两个参数中的最小值
static double pow(double a, double b)	返回第一个参数的幂
static int abs(int a)	返回参数指定数值的绝对值
static long round(double a)	返回参数四舍五入的结果
static double sqrt(double a)	返回参数的平方根
static double random()	返回0.0到1.0的随机数

11.4.2 BigDecimal类的概述

（1）基本概念

由于float类型和double类型在运算时可能会有误差，若希望实现精确运算则借助java.math.BigDecimal类型加以描述。

（2）常用的方法

方法声明	功能介绍
BigDecimal(String val)	根据参数指定的字符串来构造对象
BigDecimal add(BigDecimal augend)	用于实现加法运算
BigDecimal subtract(BigDecimal subtrahend)	用于实现减法运算
BigDecimal multiply(BigDecimal multiplicand)	用于实现乘法运算
BigDecimal divide(BigDecimal divisor)	用于实现除法运算

11.4.3 BigInteger类的概念

（1）基本概念

若希望表示比long类型范围还大的整数数据，则需要借助java.math.BigInteger类型描述。

（2）常用的方法

方法声明	功能介绍
BigInteger(String val)	根据参数指定的字符串来构造对象
BigInteger add(BigInteger val)	用于实现加法运算
BigInteger subtract(BigInteger val)	用于实现减法运算
BigInteger multiply(BigInteger val)	用于实现乘法运算
BigInteger divide(BigInteger val)	用于实现除法运算
BigInteger remainder(BigInteger val)	用于实现取余运算
BigInteger[] divideAndRemainder(BigInteger val)	用于实现取商和余数的运算

第十二章 String类的概述和使用

12.1 String类的概念（重点）

- java.lang.String类用于描述字符串，Java程序中所有的字符串字面值都可以使用该类的对象加以描述，如："abc"。
- 该类由final关键字修饰，表示该类不能被继承。
- 从jdk1.9开始该类的底层不使用char[]来存储数据，而是改成 byte[]加上编码标记，从而节约了一些空间。
- 该类描述的字符串内容是个常量不可更改，因此可以被共享使用。
  如：
  String str1 = “abc”; - 其中"abc"这个字符串是个常量不可改变。

str1 = “123”; - 将“123”字符串的地址赋值给变量str1。

- 改变str1的指向并没有改变指向的内容

12.2 常量池的概念（原理）

由于String类型描述的字符串内容是常量不可改变，因此Java虚拟机将首次出现的字符串放入常量池中，若后续代码中出现了相同字符串内容则直接使用池中已有的字符串对象而无需申请内存及创建对象，从而提高了性能。

12.3 常用的构造方法（练熟、记住）

方法声明	功能介绍
String()	使用无参方式构造对象得到空字符序列
String(byte[] bytes, int offset, int length)	使用bytes数组中下标从offset位置开始的length个字节来构造对象
String(byte[] bytes)	使用bytes数组中的所有内容构造对象
String(char[] value, int offset, int count)	使用value数组中下标从offset位置开始的count个字符来构造对象
String(char[] value)	使用value数组中的所有内容构造对象
String(String original)	根据参数指定的字符串内容来构造对象，新创建对象为参数对象的副本

12.4 常用的成员方法（练熟、记住）

方法声明	功能介绍
String toString()	返回字符串本身
byte[] getBytes()	将当前字符串内容转换为byte数组并返回
char[] toCharArray()	用于将当前字符串内容转换为char数组并返回

方法声明	功能介绍
char charAt(int index)	方法charAt用于返回字符串指定位置的字符。
int length()	返回字符串字符序列的长度
boolean isEmpty()	判断字符串是否为空

- 案例题目

  判断字符串“上海自来水来自海上”是否为回文并打印，所谓回文是指一个字符序列无论从左向右读还是从右向左读都是相同的句子。

方法声明	功能介绍
int compareTo(String anotherString)	用于比较调用对象和参数对象的大小关系
int compareToIgnoreCase(String str)	不考虑大小写，也就是'a'和'A'是相等的关系

- 案例题目

  编程实现字符串之间大小的比较并打印。

方法声明	功能介绍
String concat(String str)	用于实现字符串的拼接
boolean contains(CharSequence s)	用于判断当前字符串是否包含参数指定的内容
String toLowerCase()	返回字符串的小写形式
String toUpperCase()	返回字符串的大写形式
String trim()	返回去掉前导和后继空白的字符串
boolean startsWith(String prefix)	判断字符串是否以参数字符串开头
boolean startsWith(String prefix, int toffset)	从指定位置开始是否以参数字符串开头
boolean endsWith(String suffix)	判断字符串是否以参数字符串结尾

- 案例题目

  编程实现上述方法的使用。

方法声明	功能介绍
boolean equals(Object anObject)	用于比较字符串内容是否相等并返回
int hashCode()	获取调用对象的哈希码值
boolean equalsIgnoreCase(String anotherString)	用于比较字符串内容是否相等并返回，不考虑大小写，如：'A'和'a'是相等

- 案例题目

  提示用户从键盘输入用户名和密码信息，若输入”admin”和”123456”则提示“登录成功，欢迎使用”，否则提示“用户名或密码错误，您还有n次机会”，若用户输入三次后依然错误则提示“账户已冻结，请联系客服人员！”

方法声明	功能介绍
int indexOf(int ch)	用于返回当前字符串中参数ch指定的字符第一次出现的下标
int indexOf(int ch, int fromIndex)	用于从fromIndex位置开始查找ch指定的字符
int indexOf(String str)	在字符串中检索str返回其第一次出现的位置，若找不到返回-1
int indexOf(String str, int fromIndex)	表示从字符串的fromIndex位置开始检索str第一次出现的位置
int lastIndexOf(int ch)	用于返回参数ch指定的字符最后一次出现的下标
int lastIndexOf(int ch, int fromIndex)	用于从fromIndex位置开始查找ch指定字符出现的下标
int lastIndexOf(String str)	返回str指定字符串最后一次出现的下标
int lastIndexOf(String str, int fromIndex)	用于从fromIndex位置开始反向搜索的第一次出现的下标。

- 案例题目

  编写通用的代码可以查询字符串"Good Good Study, Day Day Up!"中所有"Day"出现的索引位置并打印出来。

方法声明	功能介绍
String substring(int beginIndex, int endIndex)	返回字符串中从下标beginIndex（包括）开始到endIndex（不包括）结束的子字符串
String substring(int beginIndex)	返回字符串中从下标beginIndex（包括）开始到字符串结尾的子字符串

- 案例题目

  提示用户从键盘输入一个字符串和一个字符，输出该字符(不含)后面的所有子字符串。

12.5 正则表达式的概念（了解）

正则表达式本质就是一个“规则字符串”，可以用于对字符串数据的格式进行验证，以及匹配、查找、替换等操作。该字符串通常使用^运算符作为开头标志，使用$运算符作为结尾标志，当然也可以省略。

12.6 正则表达式的规则（了解）

正则表达式	说明
[abc]	可以出现a、b、c中任意一个字符
[^abc]	可以出现任何字符，除了a、b、c的任意字符
[a-z]	可以出现a、b、c、……、z中的任意一个字符
[a-zA-Z0-9]	可以出现a~z、A~Z、0~9中任意一个字符

正则表达式	说明
.	任意一个字符（通常不包含换行符）
\d	任意一个数字字符，相当于[0-9]
\D	任意一个非数字字符
\s	空白字符，相当于[\t\n\x0B\f\r]
\S	非空白字符
\w	任意一个单词字符，相当于[a-zA-Z_0-9]
\W	任意一个非单词字符

正则表达式	说明
X?	表示X可以出现一次或一次也没有，也就是0 ~ 1次
X*	表示X可以出现零次或多次，也就是0 ~ n次
X+	表示X可以出现一次或多次，也就是1 ~ n次
X{n}	表示X可以出现恰好 n 次
X{n，}	表示X可以出现至少 n 次，也就是>=n次
X{n，m}	表示X可以出现至少 n 次，但是不超过 m 次，也就是>=n并且<=m次

12.7 正则表达式相关的方法（熟悉）

方法名称	方法说明
boolean matches(String regex)	判断当前正在调用的字符串是否匹配参数指定的正则表达式规则

- 案例题目

  使用正则表达式描述一下银行卡密码的规则：要求是由6位数字组成。

  使用正则表达式描述一下QQ号码的规则：要求是由非0开头的5~15位数组成。

  使用正则表达式描述一下手机号码的规则：要求是由1开头，第二位数是3、4、5、7、8中的一位，总共11位

  描述身份证号码的规则：总共18位，6位数字代表地区，4位数字代表年，2位数字代表月，2位数字代表日期， 3位数字代表个人，最后一位可能数字也可能是X。

方法名称	方法说明
String[] split(String regex)	参数regex为正则表达式，以regex所表示的字符串为分隔符，将字符串拆分成字符串数组
String replace(char oldChar, char newChar)	使用参数newChar替换此字符串中出现的所有参数oldChar
String replaceFirst(String regex, String replacement)	替换此字符串匹配给定的正则表达式的第一个子字符串
String replaceAll(String regex, String replacement)	将字符串中匹配正则表达式regex的字符串替换成replacement

第十三章 可变字符串类和日期相关类

13.1 可变字符串类（重点）

13.1.1 基本概念

- 由于String类描述的字符串内容是个常量不可改变，当需要在Java代码中描述大量类似的字符串时，只能单独申请和存储，此时会造成内存空间的浪费。
- 为了解决上述问题，可以使用java.lang.StringBuilder类和java.lang.StringBuffer类来描述字符序列可以改变的字符串，如："ab"。
- StringBuffer类是从jdk1.0开始存在，属于线程安全的类，因此效率比较低。
- StringBuilder类是从jdk1.5开始存在，属于非线程安全的类，效率比较高。

13.1.2 StringBuilder类常用的构造方法

方法声明	功能介绍
StringBuilder()	使用无参方式构造对象，容量为16
StringBuilder(int capacity)	根据参数指定的容量来构造对象，容量为参数指定大小
StringBuilder(String str)	根据参数指定的字符串来构造对象，容量为：16+字符串长度

13.1.3 StringBuilder类常用的成员方法

方法声明	功能介绍
int capacity()	用于返回调用对象的容量
int length()	用于返回字符串的长度，也就是字符的个数
StringBuilder insert(int offset, String str)	插入字符串并返回调用对象的引用，就是自己。
StringBuilder append(String str)	追加字符串
StringBuilder deleteCharAt(int index)	将当前字符串中下标为index位置的单个字符删除
StringBuilder delete(int start，int end)	删除字符串
StringBuilder replace(int start，int end，String str)	替换字符串
StringBuilder reverse()	字符串反转

- 注意

  作为参数传递的话，方法内部String不会改变其值，StringBuffer和StringBuilder会改变其值。

13.1.4 返回值的设计

- StringBuilder的很多方法的返回值均为StringBuilder类型。这些方法的返回语句均为：return this。
- 由此可见，这些方法在对StringBuilder所封装的字符序列进行改变后又返回了该对象的引用。基于这样设计的目的在于可以连续调用。

13.2 Java8之前的日期相关类（熟悉）

13.2.1 System类的概述

（1）基本概念

- Java.lang.System类中提供了一些有用的类字段和方法。

（2）常用的方法

方法声明	功能介绍
static long currentTimeMillis()	返回当前时间与1970年1月1日0时0分0秒之间以毫秒为单位的时间差

13.2.2 Date类的概述

（1）基本概念

- java.util.Date类主要用于描述特定的瞬间，也就是年月日时分秒，可以精确到毫秒。

（2）常用的方法

方法声明	功能介绍
Date()	使用无参的方式构造对象，也就是当前系统时间
Date(long date)	根据参数指定毫秒数构造对象， 参数为距离1970年1月1日0时0分0秒的毫秒数
long getTime()	获取调用对象距离1970年1月1日0时0分0秒的毫秒数
void setTime(long time)	设置调用对象为距离基准时间time毫秒的时间点

13.2.3 SimpleDateFormat类的概述

（1）基本概念

- java.text.SimpleDateFormat类主要用于实现日期和文本之间的转换。

（2）常用的方法

方法声明	功能介绍
SimpleDateFormat()	使用无参方式构造对象
SimpleDateFormat(String pattern)	根据参数指定的模式来构造对象，模式主要有: y-年 M-月 d-日 H-时 m-分 s-秒
final String format(Date date)	用于将日期类型转换为文本类型
Date parse(String source)	用于将文本类型转换为日期类型

13.2.4 Calendar类的概述

（1）基本概念

- java.util.Calender类主要用于描述特定的瞬间，取代Date类中的过时方法实现全球化。
- 该类是个抽象类，因此不能实例化对象，其具体子类针对不同国家的日历系统，其中应用最广泛的是GregorianCalendar（格里高利历），对应世界上绝大多数国家/地区使用的标准日历系统。

（2）常用的方法

方法声明	功能介绍
static Calendar getInstance()	用于获取Calendar类型的引用
void set(int year, int month, int date, int hourOfDay, int minute, int second)	用于设置年月日时分秒信息
Date getTime()	用于将Calendar类型转换为Date类型
void set(int field, int value)	设置指定字段的数值
void add(int field, int amount)	向指定字段增加数值

（3）多态的使用场合

- 通过方法的参数传递形成多态；
  ```
  public static void draw(Shape s){
      s.show();
  }
  draw(new Rect(1, 2, 3, 4));
  ```
- 在方法体中直接使用多态的语法格式
  ```
  Account acc = new FixedAccount();
  ```
- 通过方法的返回值类型形成多态
  ```
  Calender getInstance(){
     return new GregorianCalendar(zone, aLocale);
   }
  ```

13.3 Java8中的日期相关类（熟悉）

13.3.1 Java8日期类的由来

JDK 1.0中包含了一个java.util.Date类，但是它的大多数方法已经在JDK 1.1引入Calendar类之后被弃用
了。而Calendar并不比Date好多少。它们面临的问题是：

- Date类中的年份是从1900开始的，而月份都从0开始。
- 格式化只对Date类有用，对Calendar类则不能使用。
- 非线程安全等。

13.3.2 Java8日期类的概述

- Java 8通过发布新的Date-Time API来进一步加强对 日期与时间的处理。
- java.time包：该包日期/时间API的基础包。
- java.time.chrono包：该包提供对不同日历系统的访问。
- java.time.format包：该包能够格式化和解析日期时间对象。
- java.time.temporal包：该包包含底层框架和扩展特性。
- java.time.zone包：该包支持不同时区以及相关规则的类。

13.3.3 LocalDate类的概述

（1）基本概念

- java.time.LocalDate类主要用于描述年-月-日格式的日期信息，该类不表示时间和时区信息。

（2）常用的方法

方法声明	功能介绍
static LocalDate now()	在默认时区中从系统时钟获取当前日期

13.3.4 LocalTime类的概述

（1）基本概念

- java.time.LocalTime 类主要用于描述时间信息，可以描述时分秒以及纳秒。

（2）常用的方法

方法声明	功能介绍
static LocalTime now()	从默认时区的系统时间中获取当前时间
static LocalTime now(ZoneId zone)	获取指定时区的当前时间

13.3.5 LocalDateTime类的概述

（1）基本概念

- java.time.LocalDateTime类主要用于描述ISO-8601日历系统中没有时区的日期时间，如2007-12-03T10:15:30。

（2）常用的方法

方法声明	功能介绍
static LocalDateTime now()	从默认时区的系统时间中获取当前日期时间
static LocalDateTime of(int year, int month, int dayOfMonth, int hour, int minute, int second)	根据参数指定的年月日时分秒信息来设置日期时间
int getYear()	获取年份字段的数值
int getMonthValue()	获取1到12之间的月份字段
int getDayOfMonth()	获取日期字段
int getHour()	获取小时数
int getMinute()	获取分钟数
int getSecond()	获取秒数
LocalDateTime withYear(int year)	设置为参数指定的年
LocalDateTime withMonth(int month)	设置为参数指定的月
LocalDateTime withDayOfMonth(int dayOfMonth)	设置为参数指定的日
LocalDateTime withHour(int hour)	设置为参数指定的时
LocalDateTime withMinute(int minute)	设置为参数指定的分
LocalDateTime withSecond(int second)	设置为参数指定的秒
LocalDateTime plusYears(long years)	加上参数指定的年
LocalDateTime plusMonths(long months)	加上参数指定的月
LocalDateTime plusDays(long days)	加上参数指定的日
LocalDateTime plusHours(long hours)	加上参数指定的时
LocalDateTime plusMinutes(long minutes)	加上参数指定的分
LocalDateTime plusSeconds(long seconds)	加上参数指定的秒
LocalDateTime minusYears(long years)	减去参数指定的年
LocalDateTime minusMonths(long months)	减去参数指定的月
LocalDateTime minusDays(long days)	减去参数指定的日
LocalDateTime minusHours(long hours)	减去参数指定的时
LocalDateTime minusMinutes(long minutes)	减去参数指定的分
LocalDateTime minusSeconds(long seconds)	减去参数指定的秒

13.3.6 Instant类的概述

（1）基本概念

- java.time.Instant类主要用于描述瞬间的时间点信息。

（2）常用的方法

方法声明	功能介绍
static Instant now()	从系统时钟上获取当前时间
OffsetDateTime atOffset(ZoneOffset offset)	将此瞬间与偏移量组合以创建偏移日期时间
static Instant ofEpochMilli(long epochMilli)	根据参数指定的毫秒数来构造对象，参数为距离1970年1月1日0时0分0秒的毫秒数
long toEpochMilli()	获取距离1970年1月1日0时0分0秒的毫秒数

13.3.7 DateTimeFormatter类的概述

（1）基本概念

- java.time.format.DateTimeFormatter类主要用于格式化和解析日期。

（2）常用的方法

方法声明	功能介绍
static DateTimeFormatter ofPattern(String pattern)	根据参数指定的模式来获取对象
String format(TemporalAccessor temporal)	将参数指定日期时间转换为字符串
TemporalAccessor parse(CharSequence text)	将参数指定字符串转换为日期时间

第十四章 集合类库（上）

14.1 集合的概述（重点）

14.1.1 集合的由来

- 当需要在Java程序中记录单个数据内容时，则声明一个变量。
- 当需要在Java程序中记录多个类型相同的数据内容时，声明一个一维数组。
- 当需要在Java程序中记录多个类型不同的数据内容时，则创建一个对象。
- 当需要在Java程序中记录多个类型相同的对象数据时，创建一个对象数组。
- 当需要在Java程序中记录多个类型不同的对象数据时，则准备一个集合。

14.1.2 集合的框架结构

- Java中集合框架顶层框架是：java.util.Collection集合 和 java.util.Map集合。
- 其中Collection集合中存取元素的基本单位是：单个元素。
- 其中Map集合中存取元素的基本单位是：单对元素。

14.2 Collection集合（重点）

14.2.1 基本概念

- java.util.Collection接口是List接口、Queue 接口以及Set接口的父接口，因此该接口里定义的方法既可用于操作List集合，也可用于操作Queue集合和Set集合。

14.2.1 常用的方法（练熟、记住）

方法声明	功能介绍
boolean add(E e);	向集合中添加对象
boolean addAll(Collection<? extends E> c)	用于将参数指定集合c中的所有元素添加到当前集合中
boolean contains(Object o);	判断是否包含指定对象
boolean containsAll(Collection<?> c)	判断是否包含参数指定的所有对象
boolean retainAll(Collection<?> c)	保留当前集合中存在且参数集合中存在的所有对象
boolean remove(Object o);	从集合中删除对象
boolean removeAll(Collection<?> c)	从集合中删除参数指定的所有对象
void clear();	清空集合
int size();	返回包含对象的个数
boolean isEmpty();	判断是否为空
boolean equals(Object o)	判断是否相等
int hashCode()	获取当前集合的哈希码值
Object[] toArray()	将集合转换为数组
Iterator iterator()	获取当前集合的迭代器

14.3 Iterator接口（重点）

14.3.1 基本概念

- java.util.Iterator接口主要用于描述迭代器对象，可以遍历Collection集合中的所有元素。
- java.util.Collection接口继承Iterator接口，因此所有实现Collection接口的实现类都可以使用该迭代器对象。

14.3.2 常用的方法

方法声明	功能介绍
boolean hasNext()	判断集合中是否有可以迭代/访问的元素
E next()	用于取出一个元素并指向下一个元素
void remove()	用于删除访问到的最后一个元素

- 案例题目：

  如何使用迭代器实现toString方法的打印效果？

14.4 for each循环（重点）

14.4.1 基本概念

- Java5推出了增强型for循环语句，可以应用数组和集合的遍历。
- 是经典迭代的“简化版”。

14.4.2 语法格式

- for(元素类型 变量名 : 数组/集合名称) {
  循环体;
  }

14.4.3 执行流程

- 不断地从数组/集合中取出一个元素赋值给变量名并执行循环体，直到取完所有元素为止。

14.5 List集合（重中之重）

14.5.1 基本概念

- java.util.List集合是Collection集合的子集合，该集合中允许有重复的元素并且有先后放入次序。
- 该集合的主要实现类有：ArrayList类、LinkedList类、Stack类、Vector类。
- 其中ArrayList类的底层是采用动态数组进行数据管理的，支持下标访问，增删元素不方便。
- 其中LinkedList类的底层是采用双向链表进行数据管理的，访问不方便，增删元素方便。
- 可以认为ArrayList和LinkedList的方法在逻辑上完全一样，只是在性能上有一定的差别，ArrayList更适合于

  机访问而LinkedList更适合于插入和删除；在性能要求不是特别苛刻的情形下可以忽略这个差别。
- 其中Stack类的底层是采用动态数组进行数据管理的，该类主要用于描述一种具有后进先出特征的数据结构，叫做栈(last in first out LIFO)。
- 其中Vector类的底层是采用动态数组进行数据管理的，该类与ArrayList类相比属于线程安全的类，效率比较低，以后开发中基本不用。

14.5.2 常用的方法

方法声明	功能介绍
void add(int index, E element)	向集合中指定位置添加元素
boolean addAll(int index, Collection<? extends E> c)	向集合中添加所有元素
E get(int index)	从集合中获取指定位置元素
int indexOf(Object o)	查找参数指定的对象
int lastIndexOf(Object o)	反向查找参数指定的对象
E set(int index, E element)	修改指定位置的元素
E remove(int index)	删除指定位置的元素
List subList(int fromIndex, int toIndex)	用于获取子List

- 案例题目

  准备一个Stack集合，将数据11、22、33、44、55依次入栈并打印，然后查看栈顶元素并打印，然后将栈中所有数据依次出栈并打印。

  再准备一个Stack对象，将数据从第一个栈中取出来放入第二个栈中，然后再从第二个栈中取出并打印。

14.6 Queue集合（重点）

14.6.1 基本概念

- java.util.Queue集合是Collection集合的子集合，与List集合属于平级关系。
- 该集合的主要用于描述具有先进先出特征的数据结构，叫做队列(first in first out FIFO)。
- 该集合的主要实现类是LinkedList类，因为该类在增删方面比较有优势。

14.6.2 常用的方法

方法声明	功能介绍
boolean offer(E e)	将一个对象添加至队尾，若添加成功则返回true
E poll()	从队首删除并返回一个元素
E peek()	返回队首的元素（但并不删除）

- 案例题目

  准备一个Queue集合，将数据11、22、33、44、55依次入队并打印，然后查看队首元素并打印，然后将队列中所有数据依次出队并打印。

第十五章 集合类库（下）

15.1 泛型机制（熟悉）

15.1.1 基本概念

- 通常情况下集合中可以存放不同类型的对象，是因为将所有对象都看做Object类型放入的，因此从集合中取出元素时也是Object类型，为了表达该元素真实的数据类型，则需要强制类型转换，而强制类型转换可能会引发类型转换异常。
- 为了避免上述错误的发生，从Java5开始增加泛型机制，也就是在集合名称的右侧使用<数据类型>的方式来明确要求该集合中可以存放的元素类型，若放入其它类型的元素则编译报错。
- 泛型只在编译时期有效，在运行时期不区分是什么类型。

15.1.2 底层原理

- 泛型的本质就是参数化类型，也就是让数据类型作为参数传递，其中E相当于形式参数负责占位，而使用集合时<>中的数据类型相当于实际参数，用于给形式参数E进行初始化，从而使得集合中所有的E被实际参数替换，由于实际参数可以传递各种各样广泛的数据类型，因此得名为泛型。
- 如：

```
//其中i叫做形式参数，负责占位
//int i = 10;
//int i = 20;
public static void show(int i) {
    ...
}
//其中10叫做实际参数，负责给形式参数初始化
show(10);
show(20);
```

```
其中E叫做形式参数，负责占位
E = String;
E = Integer;
public interface List {
    ...
}
//其中String叫做实际参数
List lt1 = ...;
List lt2 = ...;
```

15.1.3 自定义泛型接口

- 泛型接口和普通接口的区别就是后面添加了类型参数列表，可以有多个类型参数，如：<E, T, .. >等。

15.1.4 自定义泛型类

- 泛型类和普通类的区别就是类名后面添加了类型参数列表，可以有多个类型参数，如：<E, T, .. >等。
- 实例化泛型类时应该指定具体的数据类型，并且是引用数据类型而不是基本数据类型。
- 父类有泛型，子类可以选择保留泛型也可以选择指定泛型类型。
- 子类必须是“富二代”，子类除了指定或保留父类的泛型，还可以增加自己的泛型。

15.1.5 自定义泛型方法

- 泛型方法就是我们输入参数的时候，输入的是泛型参数，而不是具体的参数。我们在调用这个泛型方法的时需要对泛型参数进行实例化。
- 泛型方法的格式：
  [访问权限] <泛型> 返回值类型 方法名([泛型标识 参数名称]) { 方法体; }
- 在静态方法中使用泛型参数的时候，需要我们把静态方法定义为泛型方法。

15.1.6 泛型在继承上的体现

- 如果B是A的一个子类或子接口，而G是具有泛型声明的类或接口，则G**并不是G的子类型！**
  比如：String是Object的子类，但是List并不是List的子类。

15.1.7 通配符的使用

- **有时候我们希望传入的类型在一个指定的范围内，此时就可以使用泛型通配符了。**
- **如：之前传入的类型要求为Integer类型，但是后来业务需要Integer的父类Number类也可以传入。**
- **泛型中有三种通配符形式：**

  <?> 无限制通配符：表示我们可以传入任意类型的参数。
  <? extends E> 表示类型的上界是E，只能是E或者是E的子类。
  <? super E> 表示类型的下界是E，只能是E或者是E的父类。

15.2 Set集合（熟悉）

15.2.1 基本概念

- **java.util.Set集合是Collection集合的子集合，与List集合平级。**
- **该集合中元素没有先后放入次序，且不允许重复。**
- **该集合的主要实现类是：HashSet类 和 TreeSet类以及LinkedHashSet类。**
- **其中HashSet类的底层是采用哈希表进行数据管理的。**
- **其中TreeSet类的底层是采用红黑树进行数据管理的。**
- **其中LinkedHashSet类与HashSet类的不同之处在于内部维护了一个双向链表，链表中记录了元素的迭代顺序，也就是元素插入集合中的先后顺序，因此便于迭代。**

15.2.2 常用的方法

- **参考Collection集合中的方法即可！**
- **案例题目**

  准备一个Set集合指向HashSet对象，向该集合中添加元素"two"并打印，再向集合中添加元素"one"并打印，再向集合中添加元素"three"并打印，再向集合中添加"one"并打印。

15.2.3 元素放入HashSet集合的原理

- **使用元素调用hashCode方法获取对应的哈希码值，再由某种哈希算法计算出该元素在数组中的索引位置。**
- **若该位置没有元素，则将该元素直接放入即可。**
- **若该位置有元素，则使用新元素与已有元素依次比较哈希值，若哈希值不相同，则将该元素直接放入。**
- **若新元素与已有元素的哈希值相同，则使用新元素调用equals方法与已有元素依次比较。**
- **若相等则添加元素失败，否则将元素直接放入即可。**

- **思考：为什么要求重写equals方法后要重写hashCode方法呢？**

- 解析：
  当两个元素调用equals方法相等时证明这两个元素相同，重写hashCode方法后保证这两个元素得到的哈希码值相同，由同一个哈希算法生成的索引位置相同，此时只需要与该索引位置已有元素比较即可，从而提高效率并避免重复元素的出现。

15.2.5 TreeSet集合的概念

- 二叉树主要指每个节点最多只有两个子节点的树形结构。
- 满足以下3个特征的二叉树叫做有序二叉树。
  a.左子树中的任意节点元素都小于根节点元素值；
  b.右子树中的任意节点元素都大于根节点元素值；
  c.左子树和右子树的内部也遵守上述规则；
- 由于TreeSet集合的底层采用红黑树进行数据的管理，当有新元素插入到TreeSet集合时，需要使用新元素与集合中已有的元素依次比较来确定新元素的合理位置。
- 比较元素大小的规则有两种方式：
  使用元素的自然排序规则进行比较并排序，让元素类型实现java.lang.Comparable接口；
  使用比较器规则进行比较并排序，构造TreeSet集合时传入java.util.Comparator接口；
- 自然排序的规则比较单一，而比较器的规则比较多元化，而且比较器优先于自然排序；

15.3 Map集合（重点）

15.3.1 基本概念

- java.util.Map<K,V>集合中存取元素的基本单位是：单对元素，其中类型参数如下：
  K - 此映射所维护的键(Key)的类型，相当于目录。
  V - 映射值(Value)的类型，相当于内容。
- 该集合中key是不允许重复的，而且一个key只能对应一个value。
- 该集合的主要实现类有：HashMap类、TreeMap类、LinkedHashMap类、Hashtable类、Properties类。
- 其中HashMap类的底层是采用哈希表进行数据管理的。
- 其中TreeMap类的底层是采用红黑树进行数据管理的。
- 其中LinkedHashMap类与HashMap类的不同之处在于内部维护了一个双向链表，链表中记录了元素的迭代顺序，也就是元素插入集合中的先后顺序，因此便于迭代。
- 其中Hashtable类是古老的Map实现类，与HashMap类相比属于线程安全的类，且不允许null作为key或者value的数值。
- 其中Properties类是Hashtable类的子类，该对象用于处理属性文件，key和value都是String类型的。
- Map集合是面向查询优化的数据结构, 在大数据量情况下有着优良的查询性能。
- 经常用于根据key检索value的业务场景。

15.3.2 常用的方法

方法声明	功能介绍
V put(K key, V value)	将Key-Value对存入Map，若集合中已经包含该Key，则替换该Key所对应的Value，返回值为该Key原来所对应的Value，若没有则返回null
V get(Object key)	返回与参数Key所对应的Value对象，如果不存在则返回null
boolean containsKey(Object key);	判断集合中是否包含指定的Key
boolean containsValue (Object value);	判断集合中是否包含指定的Value
V remove(Object key)	根据参数指定的key进行删除
Set keySet()	返回此映射中包含的键的Set视图
Collection values()	返回此映射中包含的值的Set视图
Set<Map.Entry<K,V>> entrySet()	返回此映射中包含的映射的Set视图

15.3.3 元素放入HashMap集合的原理

- **使用元素的key调用hashCode方法获取对应的哈希码值，再由某种哈希算法计算在数组中的索引位置。**
- **若该位置没有元素，则将该键值对直接放入即可。**
- **若该位置有元素，则使用key与已有元素依次比较哈希值，若哈希值不相同，则将该元素直接放入。**
- **若key与已有元素的哈希值相同，则使用key调用equals方法与已有元素依次比较。**
- **若相等则将对应的value修改，否则将键值对直接放入即可。**

15.3.4 相关的常量

- **DEFAULT_INITIAL_CAPACITY : HashMap的默认容量是16。**
- **DEFAULT_LOAD_FACTOR：HashMap的默认加载因子是0.75。**
- **threshold：扩容的临界值，该数值为：容量*填充因子，也就是12。**
- **TREEIFY_THRESHOLD：若Bucket中链表长度大于该默认值则转化为红黑树存储，该数值是8。**
- **MIN_TREEIFY_CAPACITY：桶中的Node被树化时最小的hash表容量，该数值是64。**

15.4 Collections类

15.4.1 基本概念

- **java.util.Collections类主要提供了对集合操作或者返回集合的静态方法。**

15.4.2 常用的方法

方法声明	功能介绍
static <T extends Object & Comparable<? super T>> T max(Collection<? extends T> coll)	根据元素的自然顺序返回给定集合的最大元素
static T max(Collection<? extends T> coll, Comparator<? super T> comp)	根据指定比较器引发的顺序返回给定集合的最大元素
static <T extends Object & Comparable<?super T>> T min(Collection<? extends T> coll)	根据元素的自然顺序返回给定集合的最小元素
static T min(Collection<? extends T> coll, Comparator<? super T> comp)	根据指定比较器引发的顺序返回给定集合的最小元素
static void copy(List<? super T> dest, List<? extends T> src)	将一个列表中的所有元素复制到另一个列表中

方法声明	功能介绍
static void reverse(List<?> list)	反转指定列表中元素的顺序
static void shuffle(List<?> list)	使用默认的随机源随机置换指定的列表
static <T extends Comparable<? super T>> void sort(List list)	根据其元素的自然顺序将指定列表按升序排序
static void sort(List list, Comparator<? super T> c)	根据指定比较器指定的顺序对指定列表进行排序
static void swap(List<?> list, int i, int j)	交换指定列表中指定位置的元素