JavaSE 常用类

第6章 JavaSE常用类

- 6.1 Java SE API概述
- 6.2 java.lang包
- 6.3 java.util包
- 6.4 国际化及格式化相关类
- 6.5 正则表达式
- 6.6 大数字操作

6.1 Java SE API

- API(Application Programming Interface)应用程序编程接口。
- Java SE API：JDK中提供的各种功能的java类和接口

6.2 java.lang包

- java.lang包中提供的是利用Java编程语言进行程序设计的基础类。
  - java.lang包中的类由编译器自动导入。
- 主要常用类
  - Object类
  - 基本数据类型的包装类
  - 枚举类型
  - 数学函数类
  - System类
  - Runtime类
  - 字符串相关类

6.2.1 Object类

- 所有Java类的根类。如果在类的声明中未明确使用extends关键字指定父类，则默认为继承自Object类
- 查询Java SE API 中Object类的方法说明
  - String toString()：返回代表该对象值的字符串。
    - Object类中返回的串是“类的全限定名@对象哈希码的十六进制整数值"
    - 建议在自定义类中重写
  - boolean equals(Object obj)：测试其它某个对象是否与此对象"相等"
    - Object类中是通过判断两个对象变量是否指向同一块内存区域
    - 建议在自定义类中重写
  - int hashCode()：返回该对象的哈希码值
    - 在重写equals()方法时，建议同时重写hashCode()方法，因为在某些场合需要比较两个对象是否为相同的对象时，会调用到这两个方法来判断

示例

```
public class TestObject{

    public static void main(String[] args){
        TestObject obj = new TestObject();
        System.out.println(obj.toString());
        System.out.println(obj.hashCode());

        TestObject obj2 = new TestObject();

        System.out.println(obj.equals(obj2));
    }
}
```

6.2.2 基本数据类型的包装类

- Java语言针对所有的基本数据类型都提供了对应的包装类。

基本数据类型	包装类
byte（字节）	java.lang.Byte
char（字符）	java.lang.Character
short（短整型）	java.lang.Short
int（整型）	java.lang.Integer
long（长整型）	java.lang.Long
float（浮点型）	java.lang.Float
double（双精度）	java.lang.Double
boolean（布尔）	java.lang.Boolean

- 所有基本数据类型都能很方便的和对应的包装类相互转换。

示例

```
public class NumberWrap {
    public static void main(String[] args) {
        String str = args[0];
        //------------- String转换成Ingeter
        Integer integer = new Integer(str);     // 方式一
        // Integer integer = Integer.valueOf(str); //方式二
        //------------- Ingeter转换成String
        String str2 = integer.toString();
        //------------- 把Ingeter转换成int
        int i = integer.intValue();
        //------------- 把int转换成Integer
        Integer integer2 = new Integer(i);//
        //Integer integer2 = Integer.valueOf(i);
        //------------- String转换成int
        int i2 = Integer.parseInt(str);
        //------------- 把int转换成String
        String str3 = String.valueOf(i2);  // 方式一
        String str4 = i2 + "";   // 方式二
    }
}
```

6.2.3 自动装箱、拆箱

- JDK5.0中为基本数据类型提供了自动装箱(boxing)、拆箱(unboxing)功能：
  - 装箱：将基本数据类型包装为对应的包装类对象
  - 拆箱：将包装类对象转换成对应的基本数据类型
- java编译器在编译时期会根据源代码的语法来决定是否进行装箱或拆箱。
- 示例：AutoBoxTest.java

6.2.4 枚举类型

- Java SE 5.0以后版本中引入的一个新的引用类型。
- 枚举类型是指由一组固定的常量组成合法值的类型，例如：一年的四季、一个星期的七天、红绿蓝三基色、状态的可用和不可用等。
- 使用enum关键字来定义，如：
  - public enum Year{ SPRING, SUMMER, AUTUMN, WINTER}
- 所有枚举类型对应的类是java.lang.Enum类的子类。在枚举类型定义中可以添加属性、构造器和方法，并可以实现任意的接口。

示例

```
enum Status{
    ACTIVE("可用"),INACTIVE("不可用");  //枚举值列表
    private final String name;//final成员
    Status(String name){//构造器。只能在内部定义枚举值时使用
        this.value = value;
    }
    public String getName(){   //提供get方法访问Name属性的值
        return name;
    }
}
public class StatusEnumTest {
    public static void main(String[] args) {
        Status status = Status.ACTIVE;
        System.out.println("文章状态：" + status);
        System.out.println("此状态对应的名称：" + status.getName());
    }
}
```

6.2.5 数学函数类

- java.lang.Math类提供了一序列的基本数学运算和几何函数的方法。
- Math类是final类，并且它的所有成员变量和成员方法都是静态的。

Math类的常用属性和方法

- 静态常量：
  - E ：自然对象底数的double值
  - PI ：圆周率的double值
- 常用静态方法

double sin(double a)	计算角 a的正弦值
double cos(double a)	计算角 a的余弦值
double pow(double a, double b)	计算 a 的 b 次方
double sqrt(double a)	计算给定值的平方根
int abs(int a)	计算int类型值a的绝对值.也提供long/float/double的
double ceil(double a)	返回大于等于 a的最小整数的double值
double floor(double a)	返回小于等于 a的最大整数的double值
int max(int a, int b)	返回int型值a和b中的较大值.也提供long/float/double的
int min(int a, int b)	返回 a 和 b 中的较小值。也提供long/float/double的
int round(float a);	四舍五入返回整数
double random()	返回带正号的double值，该值大于等于0.0且小于1.0

示例

```
public class MathTest {
    public static void main(String[] args) {
        int num = 38;
        float num1 = -65.7f;
        System.out.println(Math.ceil(num));
        System.out.println(Math.ceil(num1));
        System.out.println(Math.floor(num));
        System.out.println(Math.floor(num1));
        System.out.println(Math.round(num));
        System.out.println(Math.round(num1));
        System.out.println((int)(Math.random()*10 + 1));
    }
}
```

静态导入

- 在JDK5.0以上版本，使用静态导入可以使被导入类的所有静态变量和静态方法在当前类直接可见，使用这些静态成员无需再给出他们的类名。

```
import static java.lang.Math.*;
public class StaticImportTest {
    public static void main(String[] args) {
        int num = 38;
        System.out.println(PI);
        System.out.println(ceil(num));
        System.out.println(floor(num));
        System.out.println(round(num));
    }
}
```

6.2.6 System类

- System类代表与操作系统平台进行沟通的类。
  - 它和Math类有类似之处，也定义成final的，构造器也定义成了私有的，所有的属性和方法都是static的。
- 常用方式
  - 获取标准输入、输出和错误输出流：in/out/err属性
  - 获取当前时间值：currentTimeMillis()/nanoTime()
  - 获取或设置属性：getProperties()/getProperty()/setProperty()
  - 获取操作系统的环境变量：getenv(String name)

6.2.7 Runtime类

- Runtime类提供的方法用于本应用程序与其运行的环境进行信息的交互。
  - 这个类使用单例模式构建，即应用程序只能通过它提供的getRuntime()静态方法来获取唯一的实例。
- 常用
  - 获取运行时的内存情况：maxMemory()/totalMemory()/freeMemory()
  - 执行指定的命令：

```
Process proc = Runtime.getRuntime().exec(cmd); //执行系统的一些命令
if (proc.waitFor() != 0) {  //等待子进程结束，判断出口值
}
```

6.2.8字符串之String类

- java.lang.String代表不可变的字符序列。是final类
- 创建字符串实例的方式：(分析内存)
  - String a = "hello";
  - String b = new String("hello");
- Java中允许使用符号"+"把两个字符串连接起来组合成一个新字符串：String str = "你好" + "世界";
  - "+"号也能将字符串与其它的数据类型相连成一个新的字符串。String str = "abc" + 12;

字符串长度

- String类中有length()方法可以获得此字符串的长度。
  - 注意跟数组的length属性区分开来。
- 例：
  - String str = "abc你好吗";　str.length();
  - "hello world".length();

字符串比较

- ==号用来比较两个字符串是否存储在同一内存位置
- String类的equals()方法用来比较两字符串的内容是否相等

```
public class TestStringSign {
    public static void main(String[] args) {
        String s1 = "abc中文";
        String s2 = "abc中文";
        String s3 = "abc";
        System.out.println(s1 == s2); // ?
        System.out.println(s1 == s3); // ?
        System.out.println(s1.equals(s2)); // ?
        System.out.println(s1.equals(s3)); // ?
    }
}
```

```
public class TestStringEquals {
    public static void main(String[] args) {
        String ss1 = new String("abc中文");
        String ss2 = new String("abc中文");
        String ss3 = new String("abc");
        System.out.println(ss1 == ss2); // ?
        System.out.println(ss1 == ss3); // ?
        System.out.println(ss1.equals(ss2)); // ?
        System.out.println(ss1.equals(ss3)); // ?
    }
}
```

字符串方法-比较相关

方法	说明
boolean equalsIgnoreCase(String val)	此方法比较两个字符串，忽略大小写形式
int compareTo(String value)	按字典顺序比较两个字符串。 如果两个字符串相等，则返回 0； 如果字符串在参数值之前，则返回值小于 0； 如果字符串在参数值之后，则返回值大于 0
int compareToIgnoreCase(String val)	按字典顺序比较两个字符串，不考虑大小写
boolean startsWith(String value)	检查一个字符串是否以参数字符串开始。
boolean endsWith(String value)	检查一个字符串是否以参数个字符串结束。

字符串搜索相关

方法	说明
public int indexOf(int ch)	返回指定字符ch在此字符串中第一次出现处的索引值；如果未出现该字符，则返回 -1。
public int indexOf(String str)	返回第一次出现的指定子字符串str在此字符串中的索引值；如果未出现该字符串，则返回 -1。
public int lastIndexOf(int ch)	返回最后一次出现的指定字符在此字符串中的索引值；如果未出现该字符，则返回 -1。
public int lastIndexOf(String str)	返回最后一次出现的指定子字符串str在此字符串中的索引值；如果未出现该字符串，则返回 -1。
public char charAt(int index)	从指定索引index处提取单个字符，索引中的值必须为非负

示例

```
public class SearchString {
    public static void main(String[] args) {
        String name = "test@126.com";
        System.out.println("我的Email是: " + name);

        System.out.println("@ 的索引是:" + name.indexOf('@'));
        System.out.println(".com 的索引是:" + name.indexOf(".com"));

        if (name.indexOf('@') < name.indexOf(".com")) {
            System.out.println("该电子邮件地址有效");
        } else {
            System.out.println("该电子邮件地址无效");
        }
    }
}
```

修改

方法	说明
public String substring(int index)	提取从位置索引index开始直到此字符串末尾的这部分子字符串
public String substring(int beginIndex, int endIndex)	提取从 beginindex开始直到 endindex（不包括此位置）之间的这部分字符串
public String concat(String str)	将指定字符str串联接到此字符串的结尾成为一个新字符串返回。
public String replace(char oc, char nc)	返回一个新的字符串，它是通过用 newChar 替换此字符串中出现的所有 oldChar 而生成的。
public String replace(CharSequence target, CharSequence replacement)	使用指定的字面值替换序列替换此字符串所有匹配字面值目标序列的子字符串
public String trim()	返回字符串的副本，忽略前导空白和尾部空白
public String toUpperCase();	将此字符串中的所有字符都转换为大写
public String toLowerCase();	将此字符串中的所有字符都转换为小写

示例

```
public class StringMethods {
   public static void main(String [] args) {
      String s1 = "Hello world";
      String s2 = "Hello";
      String s3 = " Hello world ";
      System.out.println(s1.charAt(7));
      System.out.println(s1.substring(3, 8));
      System.out.println(s2.concat("World"));
      System.out.println(s2.replace('l', 'w'));
      System.out.println(s3.trim());
   }
}
```

```
public class StringCase {
   public static void main(String [] args) {
      String str = "Hello World";
      System.out.println(str.toUpperCase());
      System.out.println(str.toLowerCase());
   }
}
```

其它类型转换成字符串

- 在String类中定义了一些静态的重载方法
  - public static String valueOf(…)可以将基本类型数据、Object类型转换为字符串。如：
    - public static String valueOf(double d) 把double类型数据转成字符串
    - public static String valueOf(Object obj) 调用obj的toString()方法得到它的字符串表示形式。
    - int I = 100;
    - String s = I + “”;

与正则表达式相关的方法

方法	说明
public boolean matches(String regex)	此字符串是否匹配给定的正则表达式
public String[] split(String regex)	根据给定正则表达式的匹配拆分此字符串
public String replaceAll(String regex, String replacement)	使用给定的 replacement 替换此字符串所有匹配给定的正则表达式的子字符串

练习

- 编写一个java方法，用来统计所给字符串中大写英文字母的个数，小写英文字母的个数以及非英文字母的个数。

字符串的不变性

- 一个String对象的长度是固定的，不能改变它的内容，或者是附加新的字符到String对象中。
- 您也许会使用+来串联字符串以达到附加新字符或字符串的目的，但+会产生一个新的String对象。
- 如果程序对这种附加字符串的需求很频繁，并不建议使用+来进行字符串的串联。应该使用java.lang.StringBuilder类。

6.2.9 StringBuilder/StringBuffer类

- java.lang.StringBuilder/StringBuffer代表可变的字符序列。它们提供了相同的操作方法。
  - StringBuilder类的方法不保证线程同步，在非线程的情况下使用会有较好的效率。
  - StringBuffer类的方法保证线程同步。
- StringBuilder类的常用构造方法：
  - StringBuilder()
    - 构造一个其中不带字符的字符串缓冲区，初始容量为 16 个字符
  - StringBuilder(String str)
    - 构造一个字符串缓冲区，并将其内容初始化为指定字符串内容

StringBuilder常用方法

方法	说明
StringBuilder append(String str);	**将指定的字符串追加到此字符序列**
StringBuilder insert(int offset, String str)	将字符串str插入此字符序列指定位置中
int length( )	确定 StringBuffer 对象的长度
void setCharAt(int pos, char ch)	使用 ch 指定的新值设置 pos 指定的位置上的字符
String toString( )	**转换为字符串形式**
StringBuilder reverse()	反转字符串
StringBuilder delete(int start, int end)	此方法将删除调用对象中从 start 位置开始直到 end 指定的索引 – 1 位置的字符序列
StringBuilder deleteCharAt(int pos)	此方法将删除 pos 指定的索引处的字符
StringBuilder replace(int start, int end, String s)	此方法使用一组字符替换另一组字符。将用替换字符串从 start 指定的位置开始替换，直到 end 指定的位置结束

示例

```
public class StringBuilderTest {
    public static void main(String []args) {

        StringBuilder sb = new StringBuilder("Java");
        sb.append(" action ");
        sb.append(1.0);

        sb.insert(5, "in ");

        String s = sb.toString();  //转换为字符串

        System.out.println(s);
    }
}
```

6.3 java.util包

- 包含collection框架、事件模型、日期和时间设施、国际化和各种实用工具类。
- 常用类
  - Random类
  - Arrays类
  - 日期和时间相关类

6.3.1 java.util.Random类

- 此类用于生成随机数（伪随机）。
- 两种构造方法
  - Random(); 创建一个新的随机数生成器(使用当前时间毫秒值为种子)
  - Random(long seed); 使用单个long种子创建一个新随机数生成器
    - 如果用相同的种子创建两个 Random 实例，则对每个实例进行相同的方法调用序列，它们将生成并返回相同的数字序列
- Random类的方法:
  - nextInt(int n); 返回一个0到n(不包括n)的随机整数值。
  - nextInt()
  - nextFloat()
  - nextDouble()　在 0.0 和 1.0 之间均匀分布的 double 值
  - nextBoolean()

示例

```
import java.util.Random;


public class TestRandom {
    public static void main(String[] args) {
        Random random = new Random();
        for(int i = 0; i < 5; i++){
            System.out.println(random.nextInt(11));
        }
        Random random2 = new Random(123);
        System.out.println(random2.nextDouble());
        Random random3 = new Random(123);
        System.out.println(random3.nextDouble());
    }
}
```

6.3.2 Arrays类

- Arrays类提供了用来操作数组的各种静态方法，被称为数组操作工具类。
- 常用方法：
  - static String toString(int[] a); 返回数组内容的字符串表示形式
  - static int[] sort(int[] a); 使用快速排序法对指定的int型数组按数字升序进行排序
  - static int binarySearch(int[] a, int key); 使用二分搜索法搜索指定的int型数组，获得指定值所在的索引值
  - static int[] copyOf(int[] original, int newLength);复制指定的数组

6.3.3 日期和时间相关

- 世界时间标准
  - UTC(Coordinated Universal Time)：世界协调时间、世界统一时间或世界标准时间
  - GMT(Greenwich Mean Time)：格林威治标准时间或格林威治平均时间
  - 严格来讲，UTC比GMT更加精确，不过它们的差值不会超过0.9秒

6.3.4 java.util.Date类

- java.util.Date类表示特定的瞬间，精确到毫秒。
- 常用构造方法
  - Date()　使用系统当前的时间创建 一个Date实例
    - 内部就是使用System. currentTimeMillis()获取系统当前时间的毫秒数来创建Date对象
  - Date(long dt)　使用自1970年1月1日00:00:00 GMT以来的指定毫秒数创建 一个Date实例
- 常用方法
  - getTime()　返回自 1970 年 1 月 1 日 00:00:00 GMT 以来此 Date 对象表示的毫秒数。
  - toString()　把此 Date 对象转换为以下形式的 String：
    - dow mon dd hh:mm:ss zzz yyyy ：星期 月 日 时:分:秒 时区 年
- Date类中的API方法不易于实现国际化，大部分被废弃了。

6.3.5 java.util.Calendar 类

- Calendar类(日历)是一个抽象基类，主要用于完成日历字段之间相互操作的功能。即可以设置和获取日历数据的特定部分。
  - 获取Calendar类的实例
    - 使用Calendar.getInstance();
    - 调用它的子类GregorianCalendar的构造方法
  - 获取指定日历字段值
    - public int get(int field)
  - 更改指定日历字段值
    - void set(int field, int value) //不会重新计算日历的时间值
    - void add(int field, int amount) //强迫日历系统立即重新计算日历的毫秒数和所有字段

示例

```
import java.util.Calendar;
import java.util.GregorianCalendar;
public class CalendarTest {
    public static void main(String[] args) {
        // Calendar cal = Calendar.getInstance();
        Calendar cal = new GregorianCalendar();
        System.out.println("Date 和 Time 的各个组成部分: ");
        System.out.println("年: " + cal.get(Calendar.YEAR));
        // 一年中的第一个月是JANUARY，值为0
        System.out.println("月: " + (cal.get(Calendar.MONTH)));
        System.out.println("日: " + cal.get(Calendar.DATE));
        // Calendar的星期常数从星期日Calendar.SUNDAY是1,星期六Calendar.SATURDAY是7
        System.out.println("星期: " + (cal.get(Calendar.DAY_OF_WEEK)));
        System.out.println("小时: " + cal.get(Calendar.HOUR_OF_DAY));
        System.out.println("分钟: " + cal.get(Calendar.MINUTE));
        System.out.println("秒: " + cal.get(Calendar.SECOND));
    }
}
```

示例

```
import java.util.Calendar;
import java.util.Date;
public class CalendarChangeFieldTest {
    public static void main(String[] args) {
        Calendar calendar = Calendar.getInstance();
        Date date = calendar.getTime(); // 从一个 Calendar 对象中获取 Date 对象
        calendar.setTime(date); //使用给定的 Date 设置此 Calendar 的时间

        calendar.set(Calendar.DAY_OF_MONTH, 8);
        System.out.println("当前时间日设置为8后,时间是:" + calendar.getTime());

        calendar.add(Calendar.HOUR, 2);
        System.out.println("当前时间加2小时后,时间是:" + calendar.getTime());

        calendar.add(Calendar.MONTH, -2);
        System.out.println("当前日期减2个月后,时间是:" + calendar.getTime());
    }
}
```

Calendar与Date的转换

```
public class Calendar2DateTest {
    public static void main(String[] args) {
        Calendar cal = Calendar.getInstance();
        //Calendar转换成Date
        Date date = cal.getTime();

        //Date转换成Calendar
        Date date2 = new Date();
        Calendar cal2 = Calendar.getInstance();
        cal2.setTime(date2);
    }
}
```

6.4 国际化和格式化相关

- 国际化(I18N)：是指某个软件应用程序运行时，在不改变它们程序逻辑的前提下支持各种语言和区域。
- 本地化(简称L10N)：是指某个软件应用程序在运行时，能支持特定地区。

- 格式化是指对一些语言和区域敏感的数据按照特定的要求进行特定输出。

6.4.1 国际化相关类

- java.util.Locale类：代表特定的地理、政治和文化地区
  - 国际标准语言代码是小写的两个字母。中文zh、英文en
  - 区域代码是大写的两个字母。大陆CN、台湾TW、美国US。
- java.util.ResourceBundle：用于加载一个资源包
  - static ResourceBundle getBundle(String baseName, Locale lo)
  - String getString(String key)
- java.text.MessageFormat类: 提供了与语言无关的生成连接消息的方式。格式化字符串
  - public static String format(String pattern, Object... args)
    - pattern用来指定消息模式串，可用"{索引}"来预设占位符。
    - args是为消息模式中的指定位符传值

示例

- classpath下的资源文件
  - msgs.properties
  - msgs_zh_CN.properties
    - 需要通过native2ascii.exe把本地字符转换成Unicode码
- 示例

```
public class I18NHelloWorldTest {
    public static void main(String[] args) {
        Locale myLocale = Locale.getDefault(); //取得系统默认的国家/语言环境
        //根据指定的国家/语言环境加载资源包
        ResourceBundle bundle = ResourceBundle.getBundle("msgs", myLocale);
        //从资源包中取得key所对应的消息
        String msg = bundle.getString("hello");
        //为带占位符的字符串传入参数
        String s = MessageFormat.format(msg, new Object[]{"张三", new Date()});
        System.out.println(s);
    }
}
```

6.4.2 java.text.DateFormat类

- DateFormat用来格式化(日期→文本)、解析(文本→日期)日期/时间。
  - 获取实例：static getInstance()
    - 也可通过SimpleDateFormat类的构造方法指定模式字符串来创建
  - 格式化：String format(Date d);
  - 解析：Date parse(String s);

字母	日期或时间元素	表示	示例
G	Era 标志符	Text	AD
y	年	Year	1996; 96
M	年中的月份	Month	July; Jul; 07
w	年中的周数	Number	27
W	月份中的周数	Number	2
D	年中的天数	Number	189
d	月份中的天数	Number	10
F	月份中的星期	Number	2
E	星期中的天数	Text	Tuesday; Tue
a	Am/pm 标记	Text	PM
H	一天中的小时数（0-23）	Number	0
k	一天中的小时数（1-24）	Number	24
K	am/pm 中的小时数（0-11）	Number	0
h	am/pm 中的小时数（1-12）	Number	12
m	小时中的分钟数	Number	30
s	分钟中的秒数	Number	55
S	毫秒数	Number	978
z	时区	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	时区	RFC 822 time zone	-0800

示例

```
public class TestDateFormat {
    public static void main(String[] args) {
        Date date = new Date();
        DateFormat formater = new SimpleDateFormat();
        System.out.println(formater.format(date));
        DateFormat f2 = new SimpleDateFormat("yyyy年MM月dd日 HH:mm:ss");
        System.out.println(f2.format(date));

        try {
            Date date2 = f2.parse("2008年08月08日 08:08:08");
            System.out.println(date2.toString());
        } catch (ParseException e) {
             e.printStackTrace();
        }
    }
}
```

6.4.3 java.text.NumberFormat 类

- NumberFormat类用来格式化和解析数值。
  - 获取实例：static getInstance()
    - 也可通过DecimalFormat类的构造方法指定模式字符串来创建
  - 格式化：String format(long n)
    - String format(double d)
  - 解析：Number parse(String s);

符号	位置	含义
0	数字	阿拉伯数字，如果不存在则显示为 0
#	数字	阿拉伯数字
.	数字	小数分隔符或货币小数分隔符
-	数字	减号
,	数字	分组分隔符
E	数字	分隔科学计数法中的尾数和指数。*在前缀或后缀中无需加引号。*
;	子模式边界	分隔正数和负数子模式
%	前缀或后缀	乘以 100 并显示为百分数
\u2030	前缀或后缀	乘以 1000 并显示为千分数
¤(\u00A4)	前缀或后缀	货币记号，由货币符号替换。
'	前缀或后缀	用于在前缀或或后缀中为特殊字符加引号，例如 "'#'#" 将 123 格式化为 "#123"。

示例

```
public class DecimalFormatTest {
    public static void main(String[] args) {
        System.out.println(formatDecimal("#,###.##", 12345.6));
        System.out.println(formatDecimal("0,000.00", 12345.678));
        System.out.println(formatDecimal("#0.00%", 0.123456789));
        System.out.println(formatDecimal("¤#,##0.00", 12345.6789));
    }
    public static String formatDecimal(String pattern, double num){
        DecimalFormat df = new DecimalFormat(pattern);
        return df.format(num);
    }
}
```

格式化字符串

- String类提供的public static String format(String format, Object... args)

6.5 正则表达式

- 正则表达式(regular expression)描述了一种字符串匹配的模式。
  - 正则表达式作为一个模板，与所搜索的字符串进行匹配。
  - 常用于：字符串的匹配、查找、替换、分割等

6.5.1 正则表达式相关

- java.util.regex包中提供了相关类
  - Pattern：模式类。正则表达式的编译表示形式
  - Matcher：匹配器。用于匹配字符序列与正则表达式模式的类
  - PatternSyntaxException：正则表达式模式中的语法错误
- String类中与正则相关的方法
  - public boolean matches(String regex); 告知此字符串是否匹配给定的正则表达式
  - String[] split(String regex); 根据给定正则表达式的匹配拆分此字符串
  - String replaceAll(String regex, String replacement);使用给定的replacement替换此字符串所有匹配给定的正则表达式的子字符串

6.5.2 Pattern类常用方法

- public static Pattern compile(String regex)
  - 将给定的正则表达式编译到模式中
- public static Pattern compile(String regex, int flags)
  - 将给定的正则表达式编译到具有给定标志的模式中
  - flags参数表示匹配时的选项，常用的flags参数值有：
    - CASE_INSENSITIVE：启用不区分大小写的匹配。
    - COMMENTS：模式中允许空白和注释。
    - MULTILINE：启用多行模式。
- public Matcher matcher(CharSequence input)
  - 生成一个给定命名的Matcher对象
- static boolean matches(String regex, CharSequence input)
  - 直接判断字符序列input是否匹配正则表达式regex。
  - 该方法适合于该正则表达式只会使用一次的情况

6.5.3 Matcher类常用方法

- public boolean matches()
  - 尝试将整个输入序列与该模式匹配。 (开头到结尾)
- public boolean find()
  - 扫描输入序列以查找与该模式匹配的下一个子序列
- public String replaceAll(String replacement)
  - 替换模式与给定替换字符串相匹配的输入序列的每个子序列
- public String replaceFirst(String replacement)
  - 替换模式与给定替换字符串匹配的输入序列的第一个子序列

- public String group()
  - 返回由以前匹配操作所匹配的输入子序列
- public String group(int group)
  - 返回在以前匹配操作期间由给定组捕获的输入子序列

6.5.4 正则表达式语法

- 正则表达式的模式串：是由普通字符（如字符a到z）以及一些特殊字符（称为元字符）组成
- 元字符从功能上分为：限定符、选择匹配符、特殊字符、字符匹配符、定位符、分组组合符、反向引用符。

6.5.5 元字符—限定符

- 1. 限定符：用于指定其前面的单个字符或组合项连续出现多少次

字符	说明
*	匹配前面的字符或子表达式零次或多次。 例如："ja*"匹配"j"和"jaaaaaaaa"。
+	匹配前面的字符或子表达式一次或多次。 例如："ja+"与"ja"和"jaaaaa"匹配，但与"j"不匹配。
?	匹配前面的字符或子表达式零次或一次。 例如："core?"匹配"cor"或"core"。不能匹配"coreeeee"
{n}	正好匹配n次。n是一个非负整数。 例如："o{2}"与"Bob"中的"o"不匹配，与"food"中的两个"o"匹配。
{n,}	至少匹配n次。 例如："o{2,}"与"Bob"中的"o"不匹配，与"foooood"中的所有"o"匹配。
{n,m}	最少匹配n次，且最多匹配m次。m和n均为非负整数，其中n<=m。 例如："o{1,3}"匹配"fooooood"中的头三个 o。注意：不能将空格插入逗号和数字之间。

贪婪匹配 &非贪婪匹配

- 限定符默认都是贪婪匹配，即尽可能多的去匹配字符。
- 当“?”紧跟在任何一个其他限定符(*，+，?，{n}，{n,}，{n,m})后面时，就是非贪婪匹配模式。

6.5.6 元字符—选择匹配符

- 2.选择匹配符："|"，它用于选择匹配两个选项之中的任意一个。
  - 例如：“j|qava” 匹配“j”或“qava”或“java”，"(j|q)ava"匹配"java"或"qava"。

6.5.7 元字符—特殊字符

- 普通字符可以直接用来表示它们本身，也可以用它们的ASCII码或Unicode代码来代表。
  - ASCII码：两位的十六进制值，前面加"\x"
    - 字符b的ASCII码为98(十六进制是62)。所以表示b字符可用"\x62"
  - Unicode码：四位十六进制值，前面加"\u"
    - 匹配字符b，可以用它的Unicode代码"\u0062"
    - 最常用情况是用来表示中文字符的范围："\u4E00-\u9FA5"
- 元字符中用到的特别字符，当作普通字符使用时需要用"\"进行转义：
  - 使用[]来引用也可以当作普通字符来使用
  - "*"匹配"*"字符、"\?"匹配问号字符、
  - "\n"匹配换行符、"\r"匹配回车符、
  - "\t"匹配制表符、\v 匹配垂直制表符、 "\f"匹配换页符

6.5.8 元字符—字符匹配符

- 用于匹配指定字符集合中的任意一个字符

字符	说明
[...]	字符集。匹配指定字符集合包含的任意一个字符。 例如："[abc]"可以与"a"、"b"、"c"三个字符中的任何一个匹配。
[^...]	反向字符集。匹配指定字符集合未包含的任意一个字符。 例如："[^abc]"匹配"a"、"b"、"c"三个字符以外的任意一个字符。
[a-z]	字符范围。匹配指定范围内的任意一个字符。 例如："[a-z]"匹配"a"到"z"范围内的任何一个小写字母。"[0-9]"匹配"0"到"9"范围内容的任何一个数字。
[^a-z]	反向字符范围。匹配不在指定范围内的任何一个字符。 例如："[^a-z]"匹配任何不在"a"到"z"范围内的任何一个字符。"[^0-9]"匹配任何不在"0"到"9"范围内容的任何一个字符。
.	匹配除"\n"之外的任何单个字符。若要匹配包括"\n"在内的任意字符，使用如"[\s\S]"之类的模式。
\d	数字字符匹配。等效于[0-9]。
\D	非数字字符匹配。等效于[^0-9]。
\w	匹配任何单词字符，包括下划线。与"[A-Za-z0-9_]"等效。
\W	与任何非单词字符匹配。与"[^A-Za-z0-9_]"等效。
\s	匹配任何空白字符，包括空格、制表符、换页符等。与"[\f\n\r\t\v]"等效。
\S	匹配任何非空白字符。与"[^\f\n\r\t\v]"等效。

6.5.9 元字符—定位符

- 用于规定匹配模式在目标字符串中的出现位置

字符	说明
^	匹配输入字符串的开始位置。^必须出现在正则模式文本的最前才起定位作用。
$	匹配输入字符串的结尾位置。$必须出现在正则模式文本的最面才起定位作用。
\b	匹配一个单词边界。 例如："er\b"匹配"never love"中的"er"，但不匹配"verb"中的"er"。
\B	非单词边界匹配。 例如："er\B"匹配"verb"中的"er"，但不匹配"never"中的"er"。

6.5.9 元字符—分组组合符

- 用()将正则表达式中的某一部分定义为"组"，并且将匹配这个组的字符保存到一个临时区域。

字符	说明
(pattern)	捕获性分组。将圆括号中的pattern部分组合成一个组合项当作子匹配，每个捕获的子匹配项按照它们在正则模式中从左到右出现的顺序存储在缓冲区中，编号从1开始，可供以后使用。 例如：“(dog)\\1"可以匹配"dogdogdog"中的"dogdog"。
(?:pattern)	非捕获性分组。即把pattern部分组合成一个组合项，但不能捕获供以后使用。
(?=pattern)	正向预测匹配分组。 例如："Windows(?=95|98|NT|2000)"能匹配"Windows2000"中的"Windows"，但不能匹配"Windows3.1"中的"Windows"。
(?!pattern)	正向否定预测匹配分组
(?<=pattern)	负向预测匹配分组。例如："(?<=95|98|NT|2000)Windows"
(?<!pattern)	负向否定预测匹配分组

6.5.10 元字符—其它

- 反向引用符：用于对捕获分组后的组进行引用的符号。格式为"\组编号"
  - 例如：要匹配"goodgood study, dayday up!"中所有连续重复的单词部分，可使用"\b([a-z]+)\1\b"来匹配。
- "$分组编号"可用来引用分组匹配到的结果字符串

- 非贪婪模式尽可能少的匹配所搜索的字符串
  - 当“?”紧跟在任何一个其他限制符(*,+,?，{n}，{n,}，{n,m})后面时，就是使用非贪婪匹配模式。
  - 例如：对于"oooo"串，"o+?"将匹配单个"o"，而"o+"将匹配所有"o"。

示例

- 验证中文名是否合法
- 验证Email地址
- 验证手机号
- 验证邮政编码
- 验证IPv4的ip地址
- 验证国内电话号码
- 从"c:/abc/bcd/def.txt"中提取出文件名
- 把"我我我要要学java"替换成"我要学java"

```
String str = "我我我要要学java";
Matcher m = Pattern.compile("(.)\\1+").matcher(str);
str = m.replaceAll("$1");
System.out.println(str);
```

6.6 大数字操作

- 666666666666666666666666666 * 888888888888 = ?
- Java语言支持大数字的操作。在java.math包中提供了两个专门的类：BigInteger和BigDecimal类。

6.6.1 java.math.BigInteger类

- BigInteger类代表任意精度的整数
  - public BigInteger add(BigInteger val)：加运算。
  - public BigInteger subtract(BigInteger val)：减运算。
  - public BigInteger multiply(BigInteger val)：乘运算。
  - public BigInteger divide(BigInteger val)：除运算。
  - public BigInteger remainder(BigInteger val)：模运算。
  - public BigInteger[] divideAndRemainder(BigInteger val)：除运算

6.6.2 java.math.BigDecimal

- BigDecimal类代表任意精度的浮点数。
  - public BigDecimal add(BigDecimal val)：加运算。
  - public BigDecimal subtract(BigDecimal val)：减运算。
  - public BigDecimal multiply(BigDecimal val)：乘运算。
  - public BigDecimal divide(BigDecimal divisor, int scale, int roundingMode)：除运算
    - 标度指的是要保留的小数位位数。
    - BigDecimal.ROUND_HALF_UP 就是我们常用的四舍五入

示例

```
public class BigDecimalTest {
public static void main(String[] args) {
        BigDecimal bd = new BigDecimal("66666666666.666666666");
        BigDecimal bd2 = new BigDecimal("1234567890.123456789");

        System.out.println("和:"+ bd.add(bd2));
        System.out.println("差:"+ bd.subtract(bd2));
        System.out.println("积:"+ bd.multiply(bd2));
        System.out.println("商:"+bd.divide(bd2, 10, BigDecimal.ROUND_HALF_UP));
    }
}
```

总结

- java.lang包
  - Object
  - Byte/Short/Integer/Long/Character/Boolean/Float/Double
  - Enum
  - Math
  - System & Runtime
  - String、StringBuilder/StringBuffer
- java.util包
  - Random
  - Arrays
  - Date & Calendar

总结2

- 国际化和格式化(java.text包)
  - Locale/ResourceBunlde/MessageFormat
  - DateFormat/SimpleDateFormat
  - NumberFormat/DecimalFormat
  - String.format(String pattern, Object… args)
- 正则表达式
  - Pattern和Matcher类
  - 元字符
- 大数字(java.math包)
  - BigInteger
  - BigDecimal