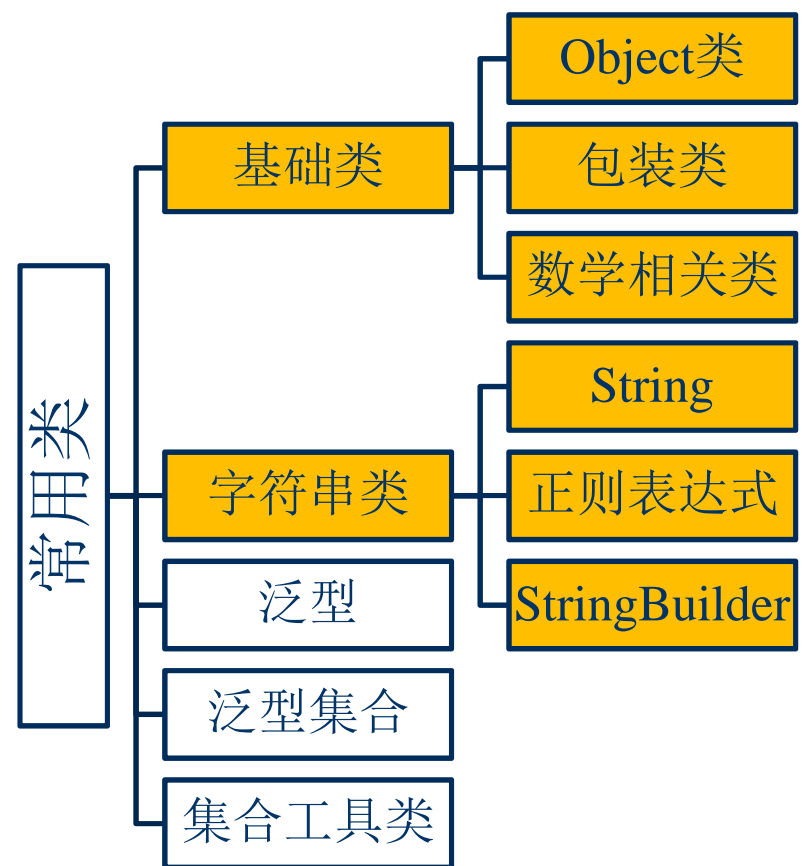




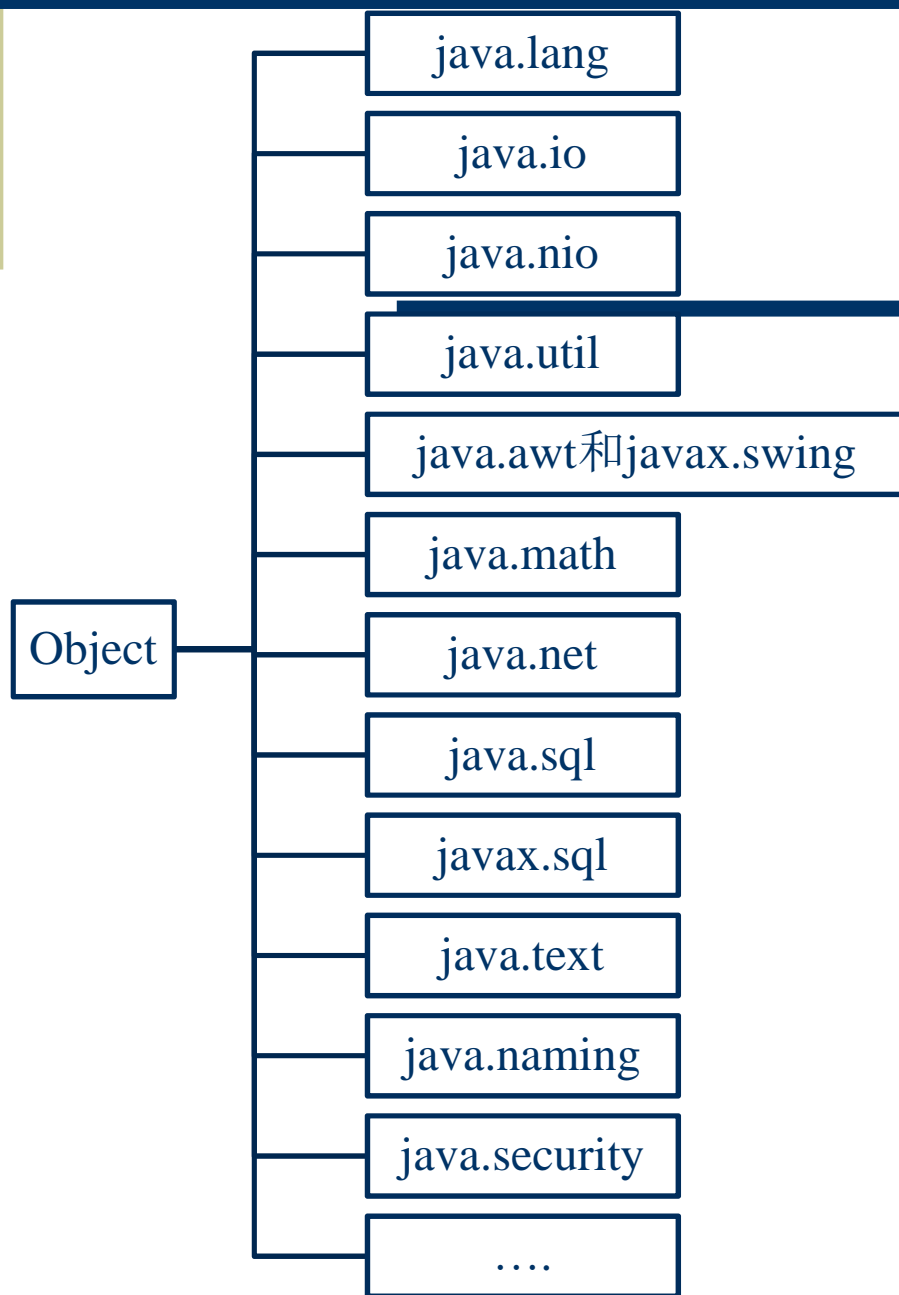
第七章 常用类(1)



第七章 常用类



Java 常用API：根据需要选择学习



7.1.2 Object类

◆ Object类： 顶级父类

- `protected Object clone()`
- `boolean equals(Object obj)` //用于比较两个对象是否相同
- `public String toString()` //用于返回对象的字符串信息。
- `public final Class<?> getClass()` // 返回对象的类型信息
- `public void notify()`
- `public void notifyAll()`
- `public wait()`
- `public int hashCode()`

7.1.2 Object类

◆ 1、equals方法

equals方法来判断两个对象是否相等，如果两个对象相等，是指他们类型相同，属性值相同。超类Object的equals提供的默认方法只是判断两个对象是否是同一对象，因此，子类需要重写 equals方法。

◆ 2、toString方法

该方法的初衷是输出对象各成员真正值，Object类里该方法不知子类的具体成员，因此返回的是“类名@哈希码”。在子类里必须覆盖该方法。

字符串连接符“+”如果连接的是对象，也会自动调用调用toString()方法。

```
package chapter7;
class Employee{
    private String name;
    private int age;
    private double salary;
    public Employee(String n, int a, double s){
        name=n;
        salary=s;
        age=a;
    }
    public String getName(){ return name; }
    public double getSalary(){return salary; }
    public int getAge(){ return age; }
```

待续

```
public boolean equals(Object obj) {
    if(obj instanceof Employee) {
        Employee other=(Employee) obj; //如果两个对象指向同一
        对象地址, 或者两者的属性值分别相等, 则两个对象相等。
        if((this==other) || (name.equals(other.name)
            &&age==other.age&&salary==other.salary))
            return true;
        }
    return false;
}

public String toString() {
    return getClass().getName()+
        "[name="+name+", salary="+salary+", age="+age+"]";
}
}
```

```
public class Equals_toStringTest {  
    public static void main(String[] args) {  
        Employee alice1=new Employee("Alice", 5000, 25);  
        Employee alice2=alice1;  
        Employee alice3=new Employee("Alice", 5000, 25);  
        Employee bob=new Employee("Bob", 10000, 30);  
        System.out.println("alice1==alice2: "+(alice1==alice2));  
        System.out.println("alice1==alice3: "+(alice1==alice3));  
        System.out.println("alice1.equals(alice3): "+alice1.equals(alice3));  
        System.out.println("alice1.equals(bob): "+alice1.equals(bob));  
        System.out.println("bob.toString(): "+bob);  
    }  
}
```

【运行结果】

```
alice1==alice2: true  
alice1==alice3: false  
alice1.equals(alice3): true  
alice1.equals(bob): false  
bob.toString(): chapter7.Employee[name=Bob, salary=30.0, age=10000]
```


7.1.3 包装类

- ◆ Java为其8个基本数据类型设计了对应的类，称为包装类（`Wrapped Class`）。包装类封装了基本数据类型的属性值和转换方法，赋予基本数据类型面向对象的特征。

7.1.3 包装类

基本数据类型的封装类：

封装类基本类型数据的属性值和转换方法

- byte的封装类是Byte
- short的封装类是Short
- int的封装类是Integer
- long的封装类型是Long
- float的封装类型是Float
- double的封装类型是Double
- char的封装类型是Character
- boolean的封装类型是Boolean

直接父类：
Number

直接父类：
Object

表7-1 Integer类常用属性

常用属性和方法	功能
static int MAX_VALUE, MIN_VALUE	int型的最大/小常量，值为 $2^{31}-1/-2^{31}$
static Integer valueOf(int i)	构造Integer实例， 返回一个值为i的Integer实例
static Integer valueOf(String s)	返回一个数值串s表示值的Integer实例
static Integer valueOf(String s, int radix)	返回一个radix进制的数值串s表示的Integer实例
static int compare(int x, int y)	比较两个int值的大小
int intValue()	返回Integer实例的基本类型值。对于其他类，这个方法可以用xxxValue()统一表示。
static String toString(int i, int radix)	用radix进制，表示值i的字符串形式。
String toString()	返回当前整数的信息
String toBinaryString(int i)	以二进制无符号整数形式返回i的字符串形式。

7.1.3 包装类及自动装箱和自动拆箱

- ◆ 包装类不提倡用构造方法创建对象，可以用valueOf方法构造对象，
- ◆ 也可以分别用Java提供的自动装箱（Autoboxing）和自动拆箱（Unboxing）方法实现构造对象、取值操作。
 - 装箱是指基本数据类型转换成对象的封装类，
 - 拆箱是指将封装的对象转换成基本类型数据值。

7.1.3 包装类及自动装箱和自动拆箱

1、利用方法进行操作

```
Integer obj=Integer.valueOf(10); //构造Integer实例  
int i=obj.intValue();           //取值
```

2、利用自动拆装箱进行操作

```
Integer obj2=10; //装箱，构造包装类对象  
int a=obj2;      //拆箱，从包装类对象取值。
```

- ◆ 这样，基本类型和包装类可以混用。例如：

```
obj2*10 //结果为100
```

7.1.4 数学相关类

（有需要时进行查询）

- ◆ Java提供了几个与数学运算密切相关的类，比如
 - `java.lang.Math`（数学类）、
 - `java.util.Random`（随机数类）、
 - `java.math.BigInteger`、`BigDecimal`
//（大整型数类）和（大数值精确运算类）。

java.lang.Math (了解)

分类	方法	功能描述
常量 字段	(1) double E: (2) double PI:	数学常量e 圆周率常量 π
三角 函数	(1) double sin(double a) (2) double cos(double a) (3) double tan(double a)	正弦: 返回角度a的sin值; 余弦: 返回角度a的cos值; 正切: 返回角度a的tan值;
反三角 函数	(1) double asin(double r) (2) double acos(double r) (3) double atan(double r)	反正弦: 返回 sin值为r的角度值; 反余弦: 返回 cos值为r的角度值; 反正切: 返回 tan值为r的角度值。
乘方	(1) double pow(double y,double x) (2) double exp(double x) (3) double log(double x) (4) double sqrt(double x)	返回y的x次方; 返回 e^x ; 返回x的自然对数; 返回x的平方根。
取整	(1) double ceil(double a) (2) double floor(double a) (3) int round(float a) (4) round(double a) (5) double rint(double a)	返回大于或等于a的最小整数, 类型为浮点型; 返回小于或等于a的最大整数, 类型为浮点型; 返回a四舍五入后的值, 结果为int整型数 返回a四舍五入后的值, 结果为长整型数 返回舍入尾数后接近a的整数值; 如果存在两个这样的整数, 则返回其中的偶数, 如 Math.rint(2.5) 结果为2.0 Math.rint(1.5)结果为2.0
其他	(1) abs(a) (2) max(a,b) (3) min(a,b) (4) random()	返回a的绝对值; 返回a和b的最大值; 返回a和b的最小值。 返回一个[0.0, 1.0) 之间的随机数

2、Random类

1) Random用于产生随机数，可以通过实例化Random对象创建一个随机数生成器

◆ 例如：

`Random r=new Random();`//以当前系统时间作为随机数生成器的种子

`Random r2=new Random(12345);` //输入种子值。

2) 主要方法有setSeed()、nextBoolean()、nextBytes()、nextDouble()、nextGaussian()、nextInt()、nextInt(int n)、nextLong()
其中：

nextInt(int n)方法生成[0, n)之间的随机整数。

nextDouble()生成[0, 1) 的浮点数，

nextGaussian()返回一个概率密度为高斯分布的双精度浮点数

例：

- ◆ `r.nextInt()`;
- ◆ `r2.nextInt(400)`;
- ◆ `r.nextDouble()`;
- ◆ `r2.nextGaussian()`;
- ◆ 其每次运行结果都不一样，给出一次的运行结果如下：
 - ◆ 286159253
 - ◆ 251
 - ◆ 0.28383408896284257
 - ◆ 0.026914003016245847

3、BigInteger和BigDecimal类

- **BigInteger**可提供任意精度的整数，表示的数字范围比**Integer**要大的多。在 **BigInteger** 上执行的操作不产生溢出，也不会丢失精度。除标准算法操作外，**BigInteger** 还提供模算法、GCD（求最大公约数） 计算、及判断是否为质数、位处理等操作。
- **BigDecimal** 用来对超过16位有效位的数进行浮点精确运算。比如在商业计算中要求数字精度比较高，就需要用**BigDecimal**。

3、BigInteger和BigDecimal类

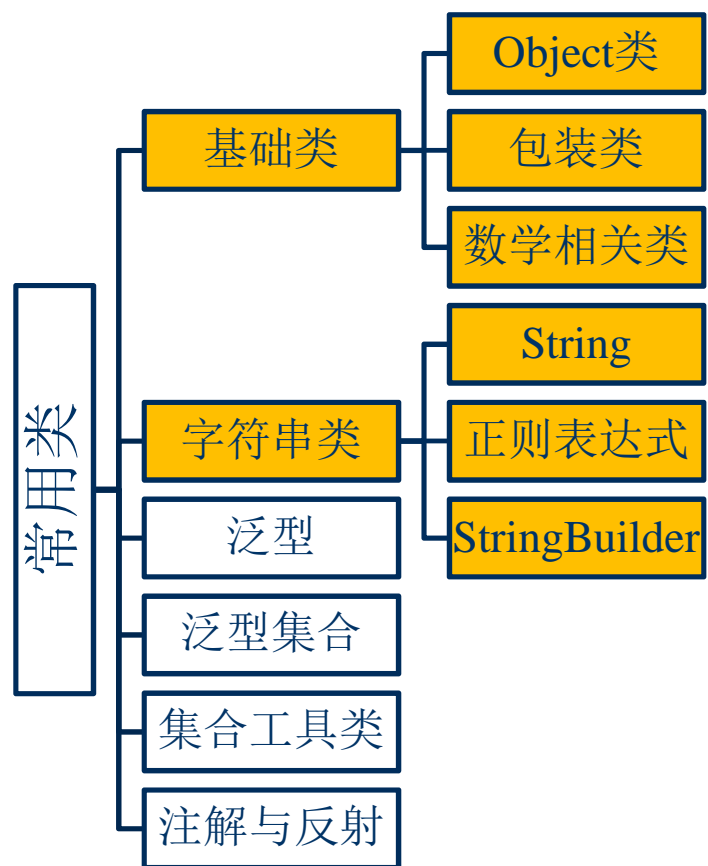
BigInteger实例:

```
BigInteger b1=new BigInteger("98765432100");
BigInteger b2=new BigInteger("123456789");
b1.multiply(b2); // 结果为 12193263111263526900
b1.divideAndRemainder(b2)[0]; //取商, 结果为800
//为divideAndRemainder结果的第一个元素
b1.divideAndRemainder(b2)[1]; //取余数, 结果为900
//为divideAndRemainder结果的第二个元素
```

BigDecimal 实例:

```
BigDecimal bd1=new BigDecimal("0.00987654321");
BigDecimal bd2=new BigDecimal(0.0001234567);
System.out.println(bd1.multiply(bd2));
System.out.println(bd1.divide(bd2, 15, RoundingMode.HALF_UP));
```

第七章 常用类



String 和StringBuilder

处理字符串的类分为两大类：

- **String**类创建之后内容不会做修改和变动的字符串类，如：用于比较两个字符串，查找，提取字符串中的子串，实现字符串类型与其他数据类型的相互转换
- **StringBuffer/StringBuilder**类：创建后允许更改和变化的字符串类

7.2.1 String

1、创建String对象

可以通过构造方法，传入字符串的内容，也可以直接赋予一个字符串字面常量。

- (1) `public String(char[] value, [int offset, int length])`
`public String(byte[] value, [int offset, int length])`

例如：

```
char[] a={'J','A','V','A'};
```

```
String str2=new String(a,1,3); //等价于"AVA"
```

- (2) `public String(StringBuffer buffer)` 后续介绍

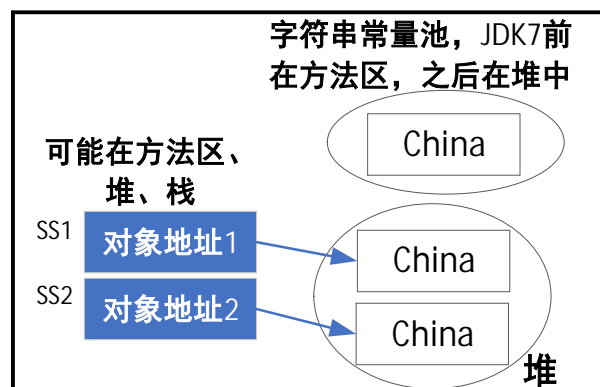
7.2.1 String

1、创建String对象

可以通过构造方法，传入字符串的内容，也可以直接赋予一个字符串字面常量。

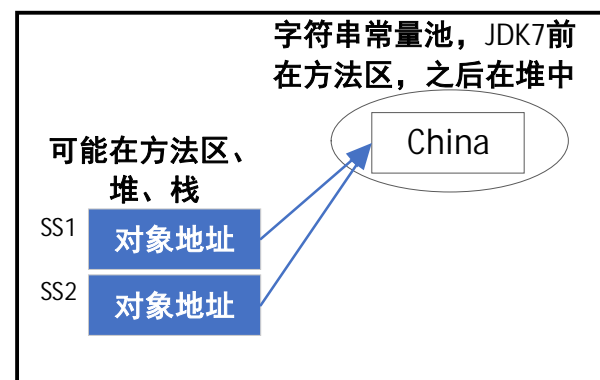
(3) public String(String value)

例如: `String ss1 = new String("china");`
`String ss2 = new String("china");`
`ss1==ss2? true:false ->false`
`ss1.equals(ss2)?true:false->true`



(4) 直接引用字符串常量

例如: `String s1 = "china";`
`String s2 = "china";`
`s1==s2? true:false //true`
`s1.equals(s2)?true:false //true`



String类型的字符串一旦创建，其值就不会再改变。

```
public class StringDemo{
    public static void change(String str, char ch[]){
        str="Changed";
        ch[0]='C';
    }
    public static void main(String arg[]){
        String s=new String("world");
        char c[]={'H','e','l','l','o'};
        change(s,c);
        System.out.println(" s= "+s);
        System.out.print(" c= ");
        System.out.println(c);
    }
}
```

【运行结果】

s= world
c= Cello

- ◆ String构造的字符串一旦创建,对象内容不能修改

- ◆ 例:

传入字符串做参数后, 如果字符串String 对象发生改变实际上是生成了一个新的字符对象。

2、字符串的常见处理方法

(1) 查找字符或子串

`int indexOf(String str, [int fromIndex])` `int lastIndexOf(String str, [int fromIndex])`

(2) 比较两个字符串

`int compareTo(String anotherString)` `int compareToIgnoreCase(String str)`
`boolean equals(Object o)`

(3) 求字符或子串

`char charAt(int index)` `String substring(int beginIndex, [int endIndex])`

(4) 判断字符串的前后缀

`Boolean endsWith(String suffix)` `Boolean startsWith(String prefix)`

(5) 连接字符串

`public String concat(String str)` 用连接符" + "连接多个类型值

(6) 替换字符

`String replaceAll(String regex, String replace)` `replaceFirst(String regex, String replace)`

(7) 格式化

`static String format(String format, Object... args)`

(8) 分割串

`String[] split(String regex)` `String[] split(String regex, int limit)`

(9) 其他操作

`String trim()`: 去掉字符串前后的空格。 `int length()`: 字符串的长度。

`String toUpperCase()`: 转成大写。 `String toLowerCase()`: 转成小写。

```
public class UseString {  
    public static void main(String arg[]) {  
        String str1="we are Chinese,you are Koreans";  
        System.out.println(str1.length());  
        System.out.println(str1.indexOf('a'));  
        System.out.println(str1.startsWith("we"));  
        System.out.println(str1.compareTo("hello"));  
        System.out.println(str1.substring(0, 6));  
        System.out.println(str1.replaceAll("are", "were"));  
        String str2[]=str1.split("[,\\s]", 4); //用空白符或者, 做分隔符  
        for(String s:str2)  
            System.out.println(s);  
    }  
}
```

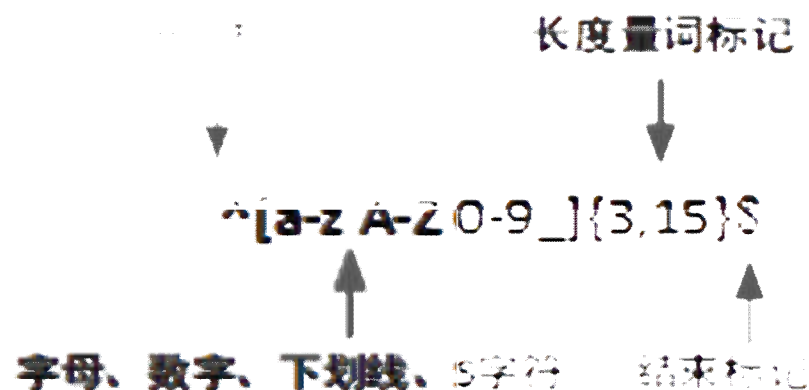
```
30  
3  
true  
15  
we are  
we were Chinese,you were Koreans  
we  
are  
Chinese  
you are Koreans
```

正则表达式

1. 正则表达式的概念

正则表达式使用一个字符串来描述、匹配符合某个句法规则的文本模式。

正则表达式使用普通字符（例如，a 到 z 之间的字母）和特殊字符（称为"元字符"）来描述字符串的组成。



(2) 字符类元字符

表 7-4 字符类元字符

字符	描述
\d	数字，等价[0-9]
\D	非数字，等价[^0-9]
\w	单字符，可用于标识符的字符，不包括“\$”
\W	非单字符，不可用于标识符的字符
\s	空白(空格符、换行符、回车符、制表符)，也可表示成 [\f\n\r\t]
\S	非空白
.	匹配除换行符\n之外的任何单字符，例 t.n，可匹配 tan,ten,tin
\p{Lower}	小写字母 a~z
\p{Upper}	大写字母 A~Z
\p{ASCII}	ASCII 字符
\p{Alpha}	字母字符
\p{Digit}	十进制数字，即 0~9
\p{Alnum}	数字或字母字符
\p{Punct}	标点符号!"#\$%&'()*+,-./:;<=>?@[]^_{}~
\p{Graph}	可见字符[\p{Alnum} \p{Punct}]
\p{Print}	可打印字符[\p{Graph} \x20]
\p{Blank}	空格或制表符[\t]
\p{Cntrl}	控制字符[\x00-\x1F \x7F]

(2) 量词元字符

匹配次数的 6 种元字符：*、?、+、{n}、{n,}、{n,m}

表 7-5 量词元字符

元字符	表达的意义	举例
X*	匹配零个或多个 X。	zo*能匹配"z"、"zoo"。* 等价于{0,}。
X?	匹配零个或一个 X。	"do(es)?" 可以匹配 "do"、"does" 。? 等价于 {0,1}。
X+	匹配一个或多个 X。	“zo+” 能匹配 "zo" 以及 "zoo"等，但不能匹配 "z"。 + 等价于 {1, }。
X{n}	匹配 n 个 X。	“o{2}”，能匹配 "oo"。
X{n, }	至少匹配 n 个 X。	'o{2,}',能匹配"oo"、"ooo"等。
X{n,m}	至少匹配 n 个至多匹配 m 个 X。注意逗号和两个数之间没有空格。	"o{1,3}" 能匹配"o"、 "oo"、"ooo"。'o{0,1}'

(3) 运算符元字符

表 7-6 运算符元字符

运算符	含义	举例
[]	表示中括号[]中任意一个字符	“[aeio]”表示 a、e、i、o 任意一个字符
	指明两项之间的一个选择。	“(a e i o)” 表示 a、ei、io 中的一组
^	1、匹配输入字符串的开始位置， 2、在方括号表达式中使用，当该符号在方括号表达式中使用，表示不接受该方括号表达式中的字符集合。	“^spring.*” “[^bc]” 非 b、c
\$	匹配输入字符串的结尾位置。如果设置了 <code>RegExp</code> 对象的 <code>Multiline</code> 属性，则 <code>\$</code> 也匹配 <code>\n</code> 或 <code>\r</code> 。	“.*App\$” 匹配 <code>IOApp</code> ，不匹配 <code>Apps</code>
\	用来将其后的字符当作普通字符而非元字符。	比如：序列 <code>\\</code> 匹配 <code>"\"</code> 。
()	标记一个子表达式的开始和结束位置。可以获取供以后使用。分组号从 1 编号，一组内容用“\$编号”表示，如\$1。注意： <code>\$0</code> 表示整个模式。	“\\D+(a e i o)”
&&	交运算	<code>[a-e&&[def]]</code> 代表字母 d、e

- ◆ 注意在正则表达式中特殊字符需要用“\”进行转义成普通字符。

举例：

[abc]表示a、b、c中的任何一个字符；

[^abc]表示除a、b、c之外的任何字符；

[a-c]表示a至c的任何一个字符；

[a-j&&[i-k]]表示i、j之中的任何一个字符；

“m(o+)n” :表示mn中间的o可以重复一次或多次如 “moon” “mon”


“\\w+\\d+” 表示以多个单字符开头，多个数字结尾的字符串，如：a100, df10000


```

public class RegexDemo {
    public static void main(String arg[]) {
        String str1="10元1000人民币1000元10000RMB, 单位有元、RMB、人民币等";
        str1=str1.replaceAll("(\\d+)(元|人民币|RMB)", "$1块");
        System.out.println(str1);
        String str2="薪水, 职位 姓名 ; 年龄 性别";
        String[] dataArr=str2.split("(\\s*, \\s*) | (\\s)*; \\s* | (\\s+)");
        for(String strTmp: dataArr){
            System.out.println(strTmp);
        }
        boolean temp="1983-07-27".matches("\\d{4}-\\d{2}-\\d{2}");
        System.out.println(temp);
    }
}

```



10块1000块1000块10000块, 单位有元、RMB、人民币等
 薪水
 职位
 姓名
 年龄
 性别
 True



String 中融合了java中两个对正则进行处理类的字符串验证、拆分、替换功能功能：

java.util.regex.Pattern

java.util.regex.Matcher;

- 
- 
- ◆ 练习4（1）：输入三个字符串，分别
 - 1 必须满足密码复杂性要求（认证需求）
 - 2 满足身份证号码规范（15/18）
 - 3 满足电子邮件规范

StringBuilder/StringBuffer

- ◆ String构造的字符串一旦创建不能修改
- ◆ StringBuffer, StringBuilder的内容可以修改
 - append方法
 - insert方法
 - delete方法
 - deleteCharAt方法
- ◆ StringBuffer: 线程安全
- ◆ StringBuilder: 线程非安全

表 7-8 StringBuilder 的常用方法

类型	方法	功能
创建对象	StringBuilder()	默认初始容量为 16 个字符
	StringBuilder(int capacity)	初始容量为 capacity
	StringBuilder(String str)	利用已存在的字符串 str 初始化
添加	StringBuilder append(XXX b)	向字符串追加 XXX 类型的数据内容。
插入	StringBuilder insert(int offset, XXX s)	在指定位置插入内容 s, XXX 是任意类型
删除	StringBuilder delete(int start, int end)	删除指定位置的字符
	StringBuilder deleteCharAt(int index)	删除指定区间的子串内容
反转	StringBuilder reverse()	反转字符串内容
获取 String	String substring(int start)	返回从 start 开始的 String 类型子串
	String substring(int start, int end)	返回从 start 到 end 的子串
	String toString()	返回所有内容的 String 字符串。

```
import java.io.*;
public class StringBufferToString {
public static void main(String args[]) {
char ch;
try
{int length =20; //置StringBuffer的初试大小为20
StringBuffer strb=new StringBuffer(length);
while ((ch=(char)System.in.read())!='\n')
{   strb.append(ch) ;
}
String str=strb.toString();
System.out.println(str);
} catch (IOException e) { }
} }
```