

# JAVA 程序设计

潘微科

感谢：教材《Java大学实用教程》的作者和其他老师提供PowerPoint讲义等资料！  
说明：本课程所使用的所有讲义，都是在以上资料上修改的。

# Outline

- 6.1 String类
- 6.2 StringBuffer类
- 6.3 StringTokenizer类
- 6.5 Scanner类
- 6.6 模式匹配
- 6.4 正则表达式及字符串的替换与分解

注：调整顺序，6.4节移到最后讲解

## 6.1 String类

- Java使用java.lang包中的String类来创建一个字符串变量，因此字符串变量是类类型的变量，是一个对象（object）。
- 字符串类String表示一个UTF-16格式的字符串，其代码单元是char。

## 6.1 String类

- 1.创建字符串对象
  - 使用String类的**构造方法**创建字符串对象

```
String s = new String("we are students");
```

- 也可以用一个**已创建的字符串**创建另一个字符串

```
String s2 = new String(s);
```

## 6.1 String类

- String类还有两个比较常用的构造方法：
  - **String(char a[])**: 用一个**字符数组**a创建一个String对象

```
char[] a = {'b','o','y'};  
String s = new String(a);
```

- **String(char a[], int startIndex, int count)**: 提取**字符数组**a中的一部分字符创建一个String对象，参数startIndex和count分别指定在a中提取字符的起始位置和从该位置开始截取的字符个数

```
char[] a = {'s','t','b','u','s','n'};  
String s = new String(a,2,3);
```

# 6.1 String类

- 2. 引用字符串常量对象
  - 字符串常量 (**string literal**) 被当作是**String对象**，因此可以把字符串常量的引用赋值给一个字符串变量 (**String variable**)

```
String s1, s2;  
s1 = "How are you";  
s2 = "How are you";
```

- s1, s2具有**相同的引用 (reference)**，因而具有**相同的实体 (string value or content)**。

Since strings are **immutable** and are ubiquitous in programming, the JVM uses a **unique instance for string literals with the same character sequence** in order to improve efficiency and save memory.

A **String variable** holds a reference to a **String object** that stores a **string value**. 大多数情况下，三者之间的差异可以忽略。

# 6.1 String类

- 3.String类的常用方法
  - public int **length**()
    - 获取一个字符串的**长度**
  - public boolean **equals**(String s)
    - 比较当前字符串对象的**实体**是否与参数指定的字符串s的**实体**相同

## 6.1 String类

- public boolean **startsWith**(String s)
  - 判断当前字符串对象的前缀是否是参数指定的字符串s
- public boolean **endsWith**(String s)
  - 判断当前字符串对象的后缀是否是参数指定的字符串s
- public int **compareTo**(String s)
  - 按字典序与参数s指定的字符串比较大小。如果当前字符串与s相同，该方法返回值0；如果当前字符串对象大于s，该方法返回正值；如果小于s，该方法返回负值。



## 6.1 String类

- 【例子】

```
public class Example6_1
{
    public static void main(String args[])
    {
        String s1,s2;
        s1 = new String("we are students");
        s2 = new String("we are students");
        System.out.println(s1.equals(s2)); // same content?
        System.out.println(s1==s2); // same reference?
        System.out.println(s1.compareTo(s2));

        String s3,s4;
        s3 = "how are you";
        s4 = "how are you";
        System.out.println(s3.equals(s4)); // same content?
        System.out.println(s3==s4); // same reference?
        System.out.println(s3.compareTo(s4));
    }
}
```

```
true
false
0
true
true
0
```

## 6.1 String类

- public int **indexOf**(String s)
  - 从当前字符串的头开始**检索**字符串s，并返回首次出现s的位置。如果没有检索到字符串s，该方法返回的值是-1。
- public String **substring**(int startpoint)
  - 获得一个当前字符串的**子串**，该子串是从当前字符串的startpoint处截取到最后所得到的字符串。

## 6.1 String类

- public String **replaceAll**(String s1, String s2)
  - 获得一个新的字符串对象，该字符串对象是通过用参数**s2指定**的字符串**替换**原字符串中**由s1指定的所有字符串**而得到的字符串。
- public String **trim**()
  - 获得一个新的字符串对象，该字符串对象是**去掉前后空格**后的字符串。

## 6.1 String类

- 4.字符串与基本数据的相互转化

- java.lang包中的Integer类调用其静态方法

`public static int parseInt(String s)`

可以将“数字”格式的字符串，如“12387”，转化为int型数据。

- 类似地：

`public static byte parseByte(String s)`

`public static short parseShort(String s)`

`public static long parseLong(String s)`

`public static double parseFloat(String s)`

`public static double parseDouble(String s)`

## 6.1 String类

- 我们也可以将数字转化为字符串，可以使用String类的静态方法

public String **valueOf**(byte b)

public String **valueOf**(short s)

public String **valueOf**(int i)

public String **valueOf**(long l)

public String **valueOf**(float f)

public String **valueOf**(double d)

## 6.1 String类

- 【例子】

```
public class Example6_2
{
    public static void main(String args[])
    {
        System.out.println(Double.parseDouble("99.99"));

        System.out.println(Integer.toBinaryString(64)); // or toString(64,2)
        System.out.println(Integer.toOctalString(64)); // or toString(64,8)
        System.out.println(Integer.toHexString(64)); // or toString(64,16)
    }
}
```

```
99.99
1000000
100
40
```

## 6.1 String类

- 5.对象的字符串表示
- 所有的类都默认是java.lang包中Object类的子类或间接子类。Object类有一个public方法**toString()**，一个对象通过调用该方法可以获得**该对象的字符串表示**。


## 6.1 String类

- 【例子】

```
import java.util.Date;
public class Example6_3
{
    public static void main(String args[])
    {
        Date date = new Date();
        Student stu = new Student("Tom", 89);
        TV tv = new TV("Samsung", 8776);
        System.out.println(date.toString());
        System.out.println(stu.toString());
        System.out.println(tv.toString());
    }
}
```

```
class TV
{
    String name;
    double price;
    TV(String name, double price)
    {
        this.name = name;
        this.price = price;
    }
}
```

```
class Student
{
    String name;
    double score;
    Student(String name, double score)
    {
        this.name = name;
        this.score = score;
    }
    public String toString()
    {
        return name+": "+score;
    }
}
```



```
Sun Oct 19 18:29:48 CST 2014
Tom: 89.0
TV@52cd32e5
```



# 6.1 String类

- 6.字符串与字符数组、字节数组
- (1)字符串与字符数组
  - String类提供了将字符串存放到数组中的方法

`public void getChars(int start, int end, char c[], int offset )`

字符串调用该方法将当前字符串中的一部分字符**复制**到参数c指定的数组中，将字符串中**从位置start到end-1位置上的字符**复制到数组c中，并从数组c的offset处开始存放这些字符。

需要注意的是，必须保证数组c能容纳下要被复制的字符。

– `public char[] toCharArray()`

字符串对象调用该方法可以初始化一个字符数组，该数组的长度与字符串的长度相等，并**将字符串对象的全部字符复制到该数组中**。

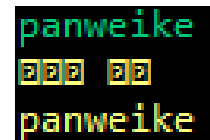
## 6.1 String类

- 【例子】

```
import java.util.Scanner;

public class Example6_4
{
    public static void main(String args[])
    {
        Scanner reader = new Scanner(System.in);
        String s = reader.nextLine();
        char a[] = s.toCharArray();
        for(int i=0; i<a.length; i++)
        {
            a[i] = (char)(a[i]^'w');
        }
        String secret = new String(a);
        System.out.println(secret);

        for(int i=0;i<a.length;i++)
        {
            a[i]=(char)(a[i]^'w');
        }
        String code = new String(a);
        System.out.println(code);
    }
}
```



```
panweike
███ ██
panweike
```

## 6.1 String类

- (2)字符串与字节数组
  - `String(byte[])`: 用指定的字节数组构造一个字符串对象。
  - `String(byte[], int offset, int length)`: 用指定的字节数组的一部分，即从数组起始位置`offset`开始取`length`个字节构造一个字符串对象。
  - `public byte[] getBytes()`: 使用平台默认的字符编码，将当前字符串转化为一个字节数组。

## 6.1 String类

- 【例子】

```
public class Example6_5
{
    public static void main(String args[])
    {
        byte d[] = "ShenzhenUniversity".getBytes();
        System.out.println(d.length);
        String s = new String(d,8,10);
        System.out.println(s);
    }
}
```

```
18
University
```

# Outline

- 6.1 String类
- 6.2 StringBuffer类
- 6.3 StringTokenizer类
- 6.5 Scanner类
- 6.6 模式匹配
- 6.4 正则表达式及字符串的替换与分解

## 6.2 StringBuffer类

- String类创建的字符串对象是**不可修改**的（不能修改、删除或替换字符串中的某个字符），即String对象一旦创建，那么**实体**是不可以再发生变化的。
- StringBuffer类：能创建**可修改**的**字符串序列**，也就是说，该类的对象的实体的内存空间可以自动的改变大小，便于存放一个**可变的字符串**。

## 6.2 StringBuffer类

- 1.StringBuffer类的构造方法
- StringBuffer类的构造方法
  - StringBuffer(): 分配给该对象的实体的**初始容量 (capacity)** 可以容纳**16个字符**，当该对象的实体存放的字符序列的长度大于16时，实体的容量自动增加，以便存放所增加的字符。
  - StringBuffer(int size): 指定分配给该对象的实体的**初始容量**为**参数size指定的字符个数**，当该对象的实体存放的字符序列的长度大于size个字符时，实体的容量自动增加，以便存放所增加的字符。
  - StringBuffer(String s): 指定分配给该对象的实体的**初始容量**为**参数字符串s的长度额外再加16个字符**。

## 6.2 StringBuffer类

- StringBuffer对象可以通过
  - length()方法获取实体中存放的字符序列的**长度**（length）
  - capacity()方法获取当前实体的实际**容量**（capacity）



## 6.2 StringBuffer类

- 2.StringBuffer类的常用方法
  - **append**方法：可以将其它Java类型数据转化为字符串后，再追加到StringBuffer对象中。
  - **char charAt(int index)**：得到参数index指定的位置上的单个字符。当前对象实体中的字符串序列的第一个位置为0，第二个位置为1，依次类推。index的值必须是非负的，并且小于当前对象实体中字符串序列的长度。
  - **void setCharAt(int index, char ch)**：将当前StringBuffer对象实体中的字符串位置index处的字符用参数ch指定的字符**替换**。index的值必须是非负的，并且小于当前对象实体中字符串序列的长度。

## 6.2 StringBuffer类

- StringBuffer **insert**(int index, String str): 将一个字符串**插入**另一个字符串中，并返回当前对象的引用。
- public StringBuffer **reverse**(): 将该对象实体中的字符串**翻转**，并返回当前对象的引用。
- StringBuffer **delete**(int startIndex, int endIndex): 从当前StringBuffer对象实体中的字符串中**删除**一个子字符串，并返回当前对象的引用。这里startIndex指定了需删除的第一个字符的下标，而endIndex指定了需删除的最后一个字符的前一个字符的下标。因此要删除的子字符串从startIndex到endIndex-1。

## 6.2 StringBuffer类

- StringBuffer **replace**(int startIndex, int endIndex, String str): 将当前StringBuffer对象实体中的字符串的一个子字符串用参数str指定的字符串**替换**。被替换的子字符串由下标startIndex和endIndex指定, 即**从startIndex到endIndex-1**的字符串被替换。该方法返回当前StringBuffer对象的引用。

## 6.2 StringBuffer类

- 【例子】

```
public class Example6_6
{
    public static void main(String args[])
    {
        StringBuffer str = new StringBuffer("0123456789");
        str.setCharAt(0, 'a');
        str.setCharAt(1, 'b');
        System.out.println(str);

        str.insert(2, "***");
        System.out.println(str);

        str.delete(6,8);
        System.out.println(str);
    }
}
```

```
ab23456789
ab**23456789
ab**236789
```

## 6.2 StringBuffer类

- StringBuffer与StringBuilder
  - 功能几乎完全相同
  - **StringBuffer是线程安全的，StringBuilder不是线程安全的**
    - **线程安全是指多个线程操作同一个对象不会出现问题。**
  - 如果字符串缓冲区被**单个线程**使用（这种情况很普遍），建议优先采用**StringBuilder**，因为效率高（因为**线程同步**需要时间开支）
  - 如果需要**多线程同步**，则建议使用**StringBuffer**

# Outline

- 6.1 String类
- 6.2 StringBuffer类
- 6.3 StringTokenizer类
- 6.5 Scanner类
- 6.6 模式匹配
- 6.4 正则表达式及字符串的替换与分解

## 6.3 StringTokenizer类

- 当我们需要分析一个字符串并将字符串**分解**成可被独立使用的单词时，可以使用java.util包中的StringTokenizer类，该类有两个常用的构造方法：
  - **StringTokenizer**(String s): 为字符串s构造一个分析器。使用默认的分隔符集合，即空格符（多个空格被看做一个空格）、换行符'\n'、回车符'\r'、tab符'\t'、进纸符'\f'
  - **StringTokenizer**(String s, String **delim**): 为字符串s构造一个分析器，参数delim中的字符被作为**分隔符**

## 6.3 StringTokenizer类

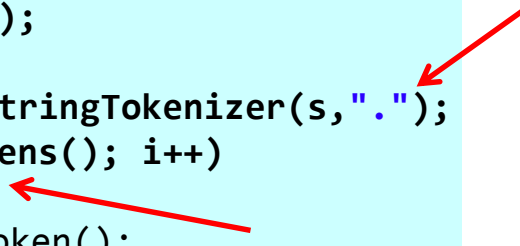
- 我们把一个StringTokenizer对象称作一个**字符串分析器**，字符串分析器封装着语言符号和对其进行操作的方法。
- **字符串分析器**可以使用nextToken()方法逐个获取**字符串分析器**中的语言符号（单词），每当获取到一个语言符号，**字符串分析器**中的负责计数的变量的值就自动减一，该计数变量的初始值等于字符串中的单词数目，**字符串分析器**调用countTokens()方法可以得到计数变量的值。
- **字符串分析器**通常用while循环来逐个获取语言符号，为了控制循环，我们可以使用StringTokenizer类中的hasMoreTokens()方法，只要计数的变量的值大于0，该方法就返回true，否则返回false。



## 6.3 StringTokenizer类

- 【例子】

```
import java.util.*;
public class Example6_7
{
    public static void main(String args[])
    {
        String [] mess = {"integer part", "decimal part"};
        Scanner reader = new Scanner(System.in);
        double x = reader.nextDouble();
        String s = String.valueOf(x);
        StringTokenizer fenxi = new StringTokenizer(s, ".");
        for(int i=0; fenxi.hasMoreTokens(); i++)
        {
            String str = fenxi.nextToken();
            System.out.println(mess[i] + ":" + str);
        }
    }
}
```



```
99.9999
integer part:99
decimal part:9999
```

# Outline

- 6.1 String类
- 6.2 StringBuffer类
- 6.3 StringTokenizer类
- 6.5 Scanner类
- 6.6 模式匹配
- 6.4 正则表达式及字符串的替换与分解

## 6.5 Scanner类

- Scanner类不仅可以创建出用于读取用户从键盘输入的数据的对象，而且还可以创建出用于解析字符串的对象。
- 1. 使用默认分隔标记解析字符串
  - 以“空白”作为分隔符

## 6.5 Scanner类

- 【例子】

```
import java.util.*;

public class Example_Scanner1
{
    public static void main (String args[])
    {
        String cost = " TV cost 877 dollar, Computer cost 2398";
        Scanner scanner = new Scanner(cost);
        double sum = 0;
        while(scanner.hasNext())
        {
            try{
                double price = scanner.nextDouble();
                sum = sum + price;
                System.out.println(price);
            }
            catch(InputMismatchException exp)
            {
                String t = scanner.next();
            }
        }
        System.out.println("Sum: " + sum);
    }
}
```

```
877.0
2398.0
Sum: 3275.0
```


## 6.5 Scanner类

- 2. 使用正则表达式作为分隔标记解析字符串
  - Scanner对象可以调用**useDelimiter()**方法将一个**正则表达式**作为分隔标记，即和正则表达式匹配的字符串都是分隔标记。

## 6.5 Scanner类

```
import java.util.*;

public class Example_Scanner2
{
    public static void main (String args[])
    {
        String cost = "市话费：176.89元，长途费：187.98元，网络费：928.66元";
        Scanner scanner = new Scanner(cost);
        scanner.useDelimiter("[^0123456789.]+");
        while(scanner.hasNext())
        {
            try{
                double price = scanner.nextDouble();
                System.out.println(price);
            }
            catch(InputMismatchException exp)
            {
                String t = scanner.next();
            }
        }
    }
}
```



```
176.89
187.98
928.66
```

# Outline

- 6.1 String类
- 6.2 StringBuffer类
- 6.3 StringTokenizer类
- 6.5 Scanner类
- 6.6 模式匹配
- 6.4 正则表达式及字符串的替换与分解

## 6.6 模式匹配

- 模式匹配就是检索和指定模式匹配的字符串。Java提供了专门用来进行模式匹配的类，这些类在`java.util.regex`包中。
- (1) 建立模式对象
- 进行模式匹配的第一步就是使用Pattern类创建一个对象，称作**模式对象**。Pattern类调用**静态方法**`compile(String pattern)`来完成这一任务，其中的参数`pattern`是一个正则表达式，称作模式对象使用的模式。
- 例如，我们使用正则表达式“A\\d”建立一个**模式对象p**  

```
Pattern p = Pattern.compile("A\\d");
```

**\\d代表0到9中的任何一个**
- 如果参数`pattern`指定的正则表达式有错，`compile`方法将抛出异常`PatternSyntaxException`。



## 6.6 模式匹配

- Pattern类也可以调用静态方法compile(String regex, int flags)返回一个Pattern对象，参数flags可以取下列有效值
  - Pattern.CASE\_INSENSITIVE
    - 表示模式匹配时将忽略大小写
  - ...

## 6.6 模式匹配

- (2) 建立匹配对象
- **模式对象p**调用**matcher**(CharSequence input)方法返回一个**Matcher对象m**（称作**匹配对象**），参数input可以是任何一个实现了CharSequence接口的类创建的对象，我们前面学习的String类和StringBuffer类都实现了CharSequence接口。
- 一个**Matcher对象m**可以使用下列3个方法寻找参数input指定的字符序列中是否有和**pattern**匹配的子序列（**pattern**是创建模式对象p时使用的正则表达式）
  - public boolean **find()**：在input中寻找和pattern匹配的**下一子序列**
  - public boolean **matches()**：判断input是否**完全**和pattern匹配
  - public boolean **lookingAt()**：判断从input的**开始位置**是否有和pattern匹配的子序列

## 6.6 模式匹配

- 下列几个方法也是**Matcher对象m**常用的方法
  - `public boolean find(int start)`: 判断input从参数start指定位置开始是否有和pattern匹配的子序列，参数start取值0时，该方法和`lookingAt()`的功能相同。
  - `public String replaceAll(String replacement)`: Matcher对象m调用该方法可以返回一个字符串对象，该字符串是通过把input中与pattern匹配的子字符串全部替换为参数replacement指定的字符串得到的（**input本身没有发生变化**）。
  - `public String replaceFirst(String replacement)`: Matcher对象m调用该方法可以返回一个字符串对象，该字符串是通过把input中**第一个**与pattern匹配的子字符串替换为参数replacement指定的字符串得到的（**input本身没有发生变化**）。

## 6.6 模式匹配

- 【例子】

```
From 0 To 3: 0A1
From 4 To 7: 2A3
From 8 To 11: 4A5
From 12 To 15: 6A7
From 16 To 19: 8A9
```

```
import java.util.regex.*;

public class Example6_8
{
    public static void main(String args[])
    {
        Pattern p;
        Matcher m;
        String input = "0A1A2A3A4A5A6A7A8A9";
        p = Pattern.compile("\\dA\\d");
        m = p.matcher(input);

        while(m.find())
        {
            String str = m.group();
            System.out.print("From " + m.start() + " To " + m.end() + ": ");
            System.out.println(str);
        }
    }
}
```

返回匹配的字符串

## 6.6 模式匹配

- 【例子】

```
import java.util.regex.*;

public class Example6_8
{
    public static void main(String args[])
    {
        Pattern p;
        Matcher m;
        String input = "0A1A2A3A4A5A6A7A8A9";
        p = Pattern.compile("\\dA\\d");
        m = p.matcher(input);

        String temp = m.replaceAll("***");
        System.out.println(temp);
        System.out.println(input);
    }
}
```

```
***A***A***A***A***
0A1A2A3A4A5A6A7A8A9
```

## 6.6 模式匹配

- 【例子】

```
import java.util.regex.*;
public class Example6_8
{
    public static void main(String args[])
    {
        Pattern p;
        Matcher m;
        String input = "9A00A3";
        p = Pattern.compile("\\dA\\d");
        m = p.matcher(input);

        if(!m.matches())
        {
            System.out.println("Not exact match");
        }
        if( m.lookingAt() )
        {
            String str = m.group();
            System.out.println(str);
        }
    }
}
```

```
Not exact match
9A0
```

# Outline

- 6.1 String类
- 6.2 StringBuffer类
- 6.3 StringTokenizer类
- 6.5 Scanner类
- 6.6 模式匹配
- 6.4 正则表达式及字符串的替换与分解

## 6.4 正则表达式及字符串的替换与分解

- 1. 正则表达式
- 一个正则表达式是一些含有特殊意义字符的字符串，这些特殊字符称作正则表达式中的**元字符**。比如，“`\\dok`”中的**`\\d`**就是有特殊意义的元字符，**代表0到9中的任何一个**。
- 一个正则表达式也称作一个**模式**，字符串“9ok”和“1ok”都是和模式“`\\dok`”匹配的字符串之一。
- 和一个模式匹配的字符串称作**匹配模式字符串**，也称作**模式匹配字符串**。



## 6.4 正则表达式及字符串的替换与分解

- 表6.1 元字符

元字符	在正则表达式中的写法	意义
.	.	代表任何一个字符
\d	\\d	代表0~9的任何一个数字
\D	\\D	代表任何一个非数字字符
\s	\\s	代表空格类字符，'\t', '\n', '\x0B', '\f', '\r'
\S	\\S	代表非空格类字符
\w	\\w	代表可用于标识符的字符（不包括美元符号）
\W	\\W	代表不能用于标识符的字符

## 6.4 正则表达式及字符串的替换与分解

- 表6.2 限定修饰符

带限定符号的模式	意义
$X?$	X出现0次或1次
$X^*$	X出现0次或多次
$X^+$	X出现1次或多次
$X\{n\}$	X恰好出现n次
$X\{n, \}$	X至少出现n次
$X\{n, m\}$	X出现n次至m次

## 6.4 正则表达式及字符串的替换与分解

- 在正则表达式（模式）中可以使用一对**方括号**括起若干个字符，代表方括号中的**任何**一个字符。例如

```
pattern = "[159]ABC"
```

- “**1**ABC”、 “**5**ABC” 和 “**9**ABC” 都是和模式pattern匹配的字符序列。

## 6.4 正则表达式及字符串的替换与分解

- [abc]: 代表a, b, c中的任何一个
- [^abc]: 代表除了a, b, c以外的任何字符
- [a-d]: 代表 a至d中的任何一个
- 另外, 方括号里允许嵌套方括号, 可以进行并、交、差运算
  - [a-d[m-p]]: 代表a至d, 或m至p中的任何字符 (并)
  - [a-z&&[def]]: 代表d, e或f中的任何一个 (交)
  - [a-f&&[^bc]]: 代表a, d, e, f (差)

## 6.4 正则表达式及字符串的替换与分解

- 用X代表正则表达式中的一个元字符或普通字符，那么“**X?**”就表示X出现0次或1次。

```
pattern = "A[1359]?"
```

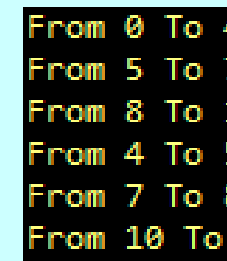
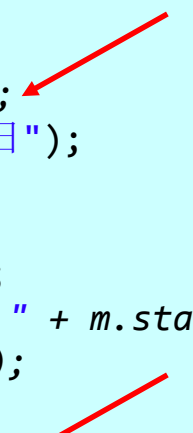
- X是“A[1359]”，那么“A”，“A1”，“A3”，“A5”，“A9”是匹配模式pattern的**全部**字符串。

```
pattern = "@\\w{4}"
```

- X是“\\w”，那么“@abcd”，“@girl”，“@moon”，“@flag”**都是**匹配模式pattern的字符串之一。

## 6.4 正则表达式及字符串的替换与分解

```
import java.util.regex.*;
public class Example6_9
{
    public static void main(String args[])
    {
        Pattern p;
        Matcher m;
        p = Pattern.compile("\\d+");
        m = p.matcher("2008年08月08日");
        while(m.find())
        {
            String str = m.group();
            System.out.print("From " + m.start() + " To " + m.end() + ": ");
            System.out.println(str);
        }
        p = Pattern.compile("\\D+");
        m = p.matcher("2008年08月08日");
        while(m.find())
        {
            String str=m.group();
            System.out.print("From " + m.start() + " To " + m.end() + ": ");
            System.out.println(str);
        }
    }
}
```



From 0 To 4: 2008  
From 5 To 7: 08  
From 8 To 10: 08  
From 4 To 5: 年  
From 7 To 8: 月  
From 10 To 11: 日

## 6.4 正则表达式及字符串的替换与分解

- 模式可以使用“|”位运算符进行逻辑“或”运算得到一个新模式。例如，pattern1、pattern2是两个模式，即两个正则表达式。那么，

```
pattern=pattern1|pattern2;
```

- 就是两个模式的“或”。一个字符串如果匹配模式pattern1 或 匹配模式pattern2，那么就匹配模式pattern。

## 6.4 正则表达式及字符串的替换与分解

- 【例子】

```
import java.util.regex.*;
public class Example6_10
{
    public static void main(String args[])
    {
        Pattern p;
        Matcher m;
        String s1 = "likeKFChateMDlike123jkjhate999like888";
        p = Pattern.compile("Like\\w{3}/hate\\w{2}");
        m = p.matcher(s1);
        while(m.find())
        {
            String str = m.group();
            System.out.print("From " + m.start() + " To " + m.end() + ": ");
            System.out.println(str);
        }
    }
}
```

```
From 0 To 7: likeKFC
From 7 To 13: hateMD
From 13 To 20: like123
From 23 To 29: hate99
From 30 To 37: like888
```



## 6.4 正则表达式及字符串的替换与分解

- 2.字符串的替换
- `public String replaceAll(String regex, String replacement)`方法返回一个字符串，该字符串是当前字符串中所有与参数`regex`指定的正则表达式匹配的字符串被参数`replacement`制定的字符串替换后的字符串。
- 【例子】

```
String result = "12hello567".replaceAll("[a-zA-Z]+", "***");
```

```
12***567
```

## 6.4 正则表达式及字符串的替换与分解

- 【例子】

```
public class Example_replaceAll
{
    public static void main (String args[])
    {
        String str = "Please logon :http://www.cctv.cn Watch TV";
        String regex = "(http://|www)[.]?\\w+[.]{1}\\w+[.]{1}\\p{Alpha}+";

        String newStr = str.replaceAll(regex, "");
        System.out.println(str);
        System.out.println(newStr);
    }
}
```

`\\p{Alpha}`表示字母字符

```
Please logon :http://www.cctv.cn Watch TV
Please logon : Watch TV
```

`p{Alpha}`: 字母

## 6.4 正则表达式及字符串的替换与分解

- 3.字符串的分解
- `public String[] split(String regex):` 使用参数指定的正则表达式`regex`做为分隔标记**分解**出其中的单词，并将分解出的单词存放在字符串数组中。
- 注：我在处理数据的时候，为了分隔不同的字段，常用`split(...)`方法

## 6.4 正则表达式及字符串的替换与分解

- 【例子】

```
import java.util.Scanner;
public class Example6_11
{
    public static void main (String args[])
    {
        Scanner reader = new Scanner(System.in);
        String str = reader.nextLine();
        //空格字符、数字和符号(!"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~)组成的正则表达式
        String regex = "[\\s\\d\\p{Punct}]+";
        String words[] = str.split(regex);
        for(int i=0; i<words.length; i++)
        {
            int m = i+1;
            System.out.println("Word" + m + ":" + words[i]);
        }
    }
}
```

```
shenzhen university @china
Word1:shenzhen
Word2:university
Word3:china
```

`\\p{Punct}`表示标点符号 `!"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~`

# 小结

- 6.1 String类: process **fixed** strings
- 6.2 StringBuffer类: process **flexible** strings
- 6.3 StringTokenizer类
- 6.5 Scanner类
- 6.6 模式匹配
- 6.4 正则表达式及字符串的替换与分解