BIT

第7章　常用工具类

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

德以明理　学以精工

第7章　常用工具类

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- String类
  - Java管理String的方法(创建、equals相等策略)
  - String类常用方法(字符串的连接、查找、比较、截取子串)
- StringBuffer/StringBuilder类
  - StringBuffer与String的区别(能否变化、对象的连接、equals比较)
  - StringBuffer与String的转换
- 正则表达式
- 包装类
- 日期类

应用程序接口API

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- Application Program Interface：类库，它们分别存放在Java核心包（包名以java开头）和扩展包（包名以javax开头）中。

- Java的类库非常庞大，本章通过实例介绍一些使用频率非常高的工具类，更重要的是读者要学会使用Java的API文档，能够随时随地浏览要使用的资源。

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

7.1 字符串处理类

7.1.1 Java中String对象的管理

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

1．对象池

- Java对象的生命周期大致包括三个阶段：对象的创建、对象的使用和对象的释放。

- 对象的创建：Java对象是通过构造方法来创建的，在这一过程中，该构造方法链中的所有构造方法也都会被自动调用。

7.1.1 Java中String对象的管理

操作	示例	标准化时间
本地赋值	i = n	1.0
实例赋值	this.i = n	1.2
方法调用	fun()	5.9
新建对象	new Object()	980
新建数组	new int[10]	3100

7.1.1 Java中String对象的管理

- 对象的释放：垃圾收集器开销。首先是对象管理开销，垃圾收集器为了能够正确释放对象，它必须监控每一个对象的运行状态，包括对象的申请、引用、被引用、赋值等；其次，在垃圾收集器开始回收垃圾对象时，系统会暂停应用程序的执行，独自占用CPU。

- 结论：要改善应用程序的性能，应尽量减少对象的创建和对象的销毁时间，这些可以通过对象池技术来实现。

7.1.1 Java中String对象的管理

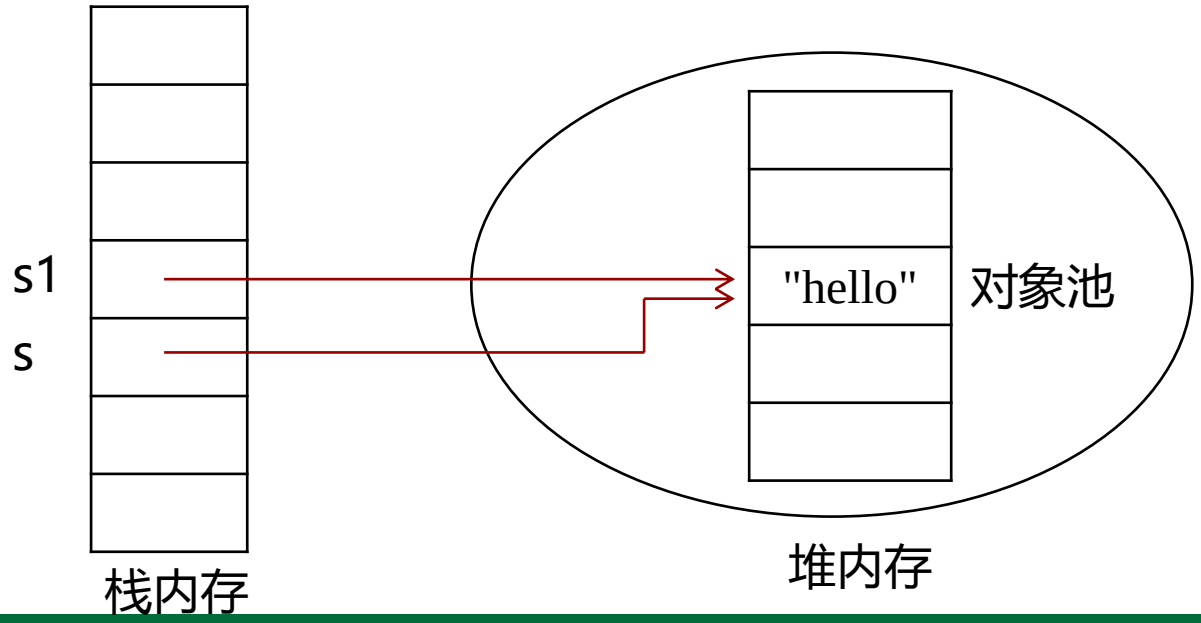
- 对象池技术的基本原理：**缓存和共享**。对于那些被频繁使用的对象，在使用完后，不立即将它们释放，而是将它们缓存起来，以供后续的应用程序重复使用，从而减少创建对象和释放对象的次数，改善应用程序的性能。

2．String对象的常量池

- 字符串操作是计算机程序设计中最常见的行为，而String常量对象的频繁出现占用了大量内存。
- Java管理String常量对象时在JVM运行时数据区开辟出一个称为“对象池”的存储空间。
- String对象池的使用方式：当编译器遇到String常量时，首先检查该池中是否已存在相同的String常量，如果已存在，则不再创建。

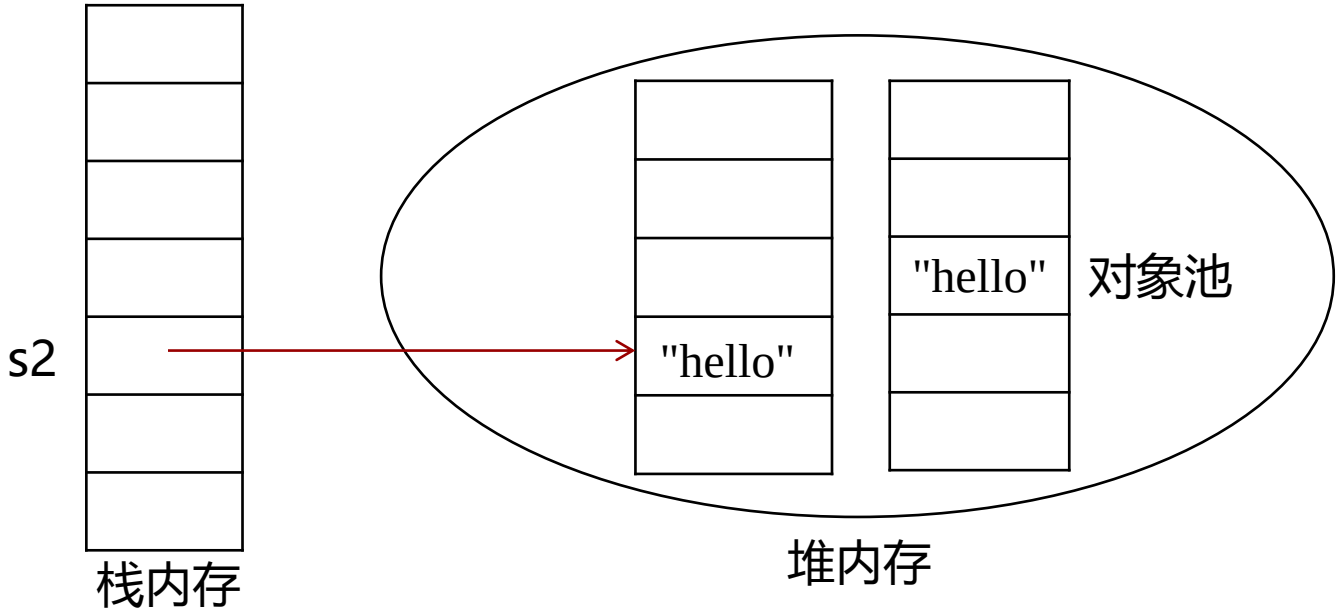
7.1.1 Java中String对象的管理

```
String s1 = "hello";
String s ="hello";
System.out.println(s==s1);
```



7.1.1 Java中String对象的管理

```
String s1 = "hello";
String s2 = new String("hello") ;
```



7.1.1 Java中String对象的管理

- String s = new String("hi"); 语句创建了几个String对象?

7.1.1 Java中String对象的管理

- 当“+”运算两侧都是String常量时，编译器会对字符串常量的运算进行优化。

String s="he"+"llo";

↓

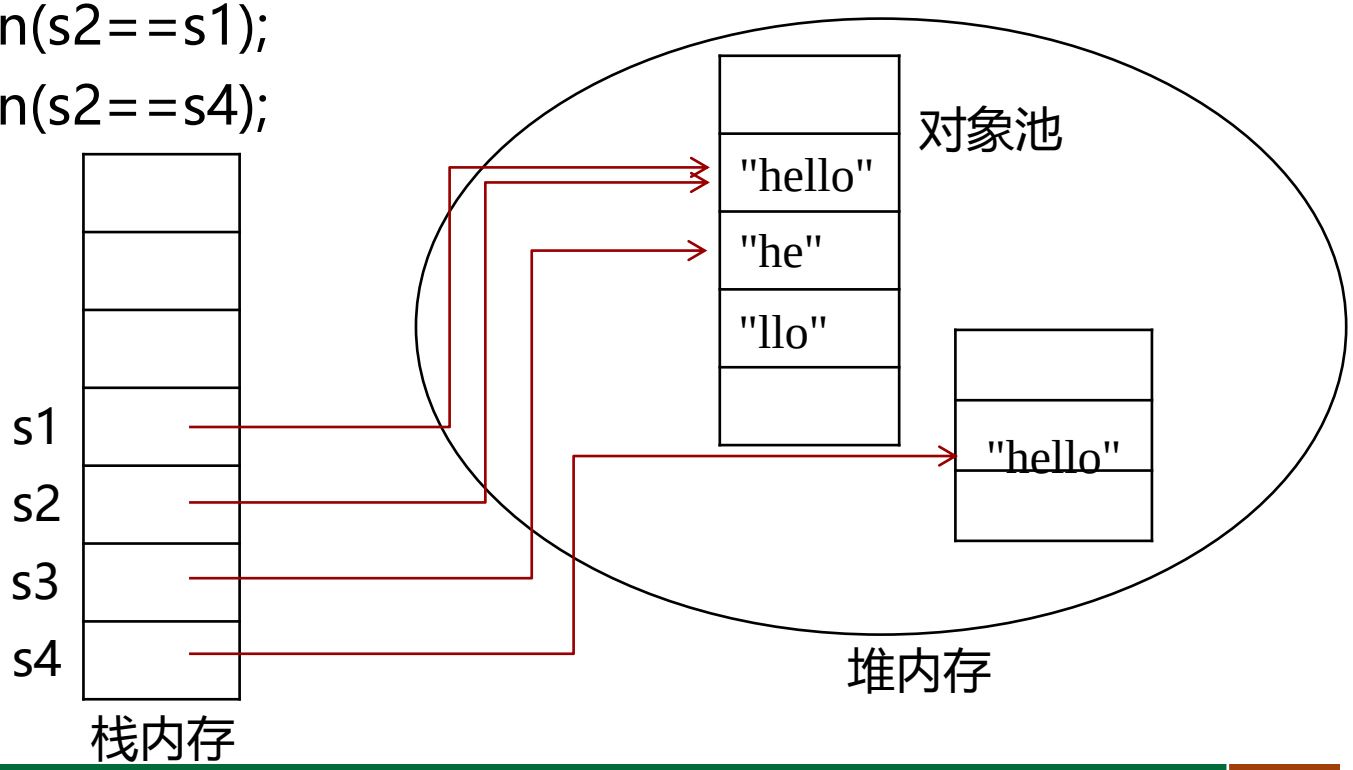
String s="hello";

【例7-1】分析下面的代码段。

```
String s1 = "hello";
String s2="he"+"llo"
String s3="he";
String s4=s3+"llo";
System.out.println(s2==s1);
System.out.println(s2==s4);
```

的运行结果是什么？该代码段一共创建了几个对象？

```
String s1 = "hello";
String s2="he"+"llo"
String s3="he";
String s4=s3+"llo";
System.out.println(s2==s1);
System.out.println(s2==s4);
```



北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

1．charAt()

- char charAt(int index)，返回调用该方法的字符串位于指定位置index处的字符。需要注意，字符串的索引值是从0开始计算的。

```
String x = "good";
System.out.println(x.charAt(3));
```

输出'd'

2．concat()

- String concat(String s)，该方法将参数字符串s追加至调用该方法的String的尾部，返回新字符串。

```
String x = "good";
System.out.println(x.concat(" idea!"));
```

"good idea!"

concat()与“+=”运算符的区别：“+=”运算是一个赋值运算。

```
String x = "good";
x += " idea!";
System.out.println(x);
```

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

3．equals()和equalsIgnoreCase()

- boolean equals(String s)，该方法将参数字符串s的值与调用该方法的String的值进行比较，返回true或false。
- String类重写了Object的equals()方法，不再比较引用地址，而是比较字符串的内容。
- boolean equalsIgnoreCase(String s)，该方法将参数字符串s的值与调用该方法的String的值进行忽略大小写的比较，返回true或false。

```
String x = "quit";
System.out.println(x.equalsIgnoreCase("QUIT"));
```

```
true
```

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

4．endsWith()和startsWith()

- boolean endsWith(String s)，判断调用该方法的String是否以指定的参数字符串结尾，返回true或false。
- boolean startsWith(String s)，判断调用该方法的String是否以指定的参数字符串开始，返回true或false。

以文件的扩展名区分文件类型：

```
String file = "photo.png";
boolean isImageFile = file.endsWith(".png")|| file.endsWith(".jpg");
System.out.println(isImageFile);
```

```
String command = "get photo.png";
if(command.startsWith("get")){
        System.out.println("开始下载文件...");
}
```

5．indexOf()和lastIndexOf()

- int indexOf(String s)， int indexOf(String s, int fromIndex)，查找参数字符串在调用该方法的String中首次出现的起始位置，如果参数字符串不存在，则返回-1，可以使用参数fromIndex指定查找的起点。
- int lastIndexOf (String s)，int lastIndexOf (String s, int fromIndex)，查找参数字符串在调用该方法的String中最后一次出现的起始位置。

7.1.2 String类的常用方法

```
String email = "computer_dite@126.com";
System.out.println(email.indexOf("@"));
System.out.println(email.indexOf("/"));
int index = email.indexOf("e");
System.out.println(email.indexOf("e", index+1));
System.out.println(email.lastIndexOf("e"));
```

输出13

输出-1

6

输出12

输出12

6．length()

- int length()，该方法返回调用该方法的String的长度，即其所包含字符的个数。

```
String x = "0123456789";
System.out.println(x.length());
```

```
10
```

7．substring()

- String substring(int begin)，String substring(int begin, int end)
- substring()方法用于获取调用该方法的String的一个子串。
- 参数begin指定截取的起始索引位置
- 参数end指定截取的结束索引位置，如果不指定end则截取至String的末尾。
- 需要注意截取子串的范围是[begin,end)，包括begin位置，但不包括end位置，至end-1位置截取结束。

7.1.2 String类的常用方法

```
String x =  "0123456789";
System.out.println(x.substring(3));
System.out.println(x.substring(3,8));
```

输出"3456789"

输出"34567"

```
String email = "computer_dite@126.com";
String name = email.substring(0, email.indexOf("@"));
String host = email.substring(email.indexOf("@")+1);
```

"computer_dite"

"126.com"

8．toLowerCase()和toUpperCase()

- String toLowerCase()，将调用该方法的String的所有大写字母都转换为小写字母，即新字符串全部由小写字母组成，String toUpperCase()方法与之相反。

```
String e = "A Good Idea!";
System.out.println(e.toLowerCase());
System.out.println(e.toUpperCase());
```

输出"a good idea!"

输出"A GOOD IDEA!"

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

9．trim()

- String trim()，将调用该方法的String的前后空格都删除，返回新字符串。

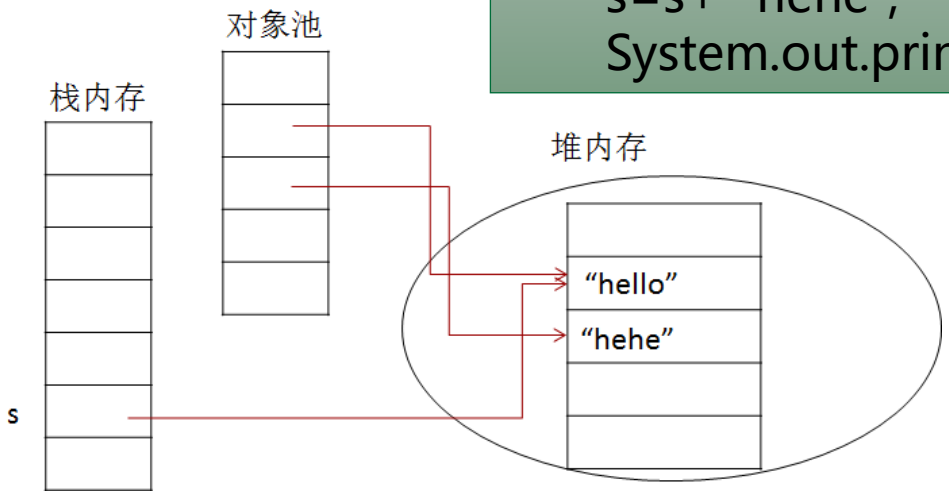
```
Scanner scn = new Scanner(System.in);
String command = scn.next();
if(command.trim().equalsIgnoreCase("dir")){
        //对该命令进行处理
}
```

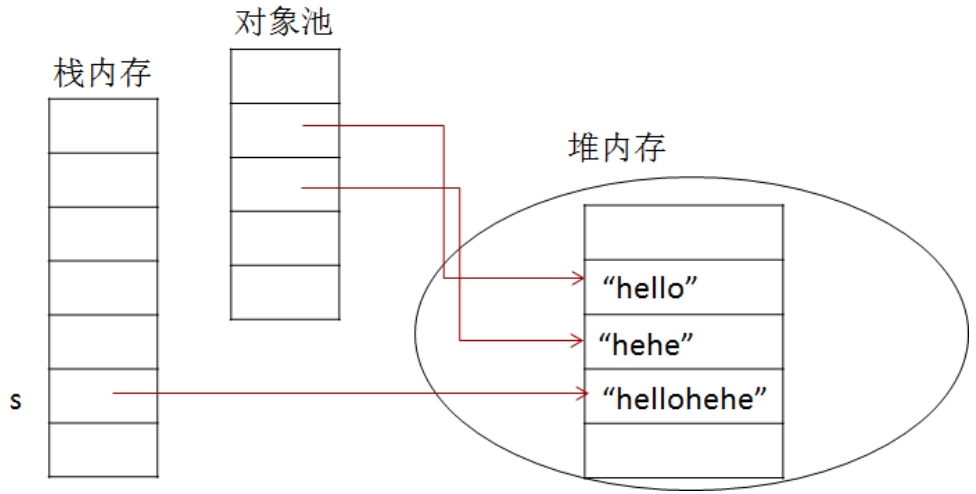
1．String的不可变性

- Java中String对象是不可变的，回顾前面介绍过的String类方法就会发现，String类中每一个看似会修改String值的方法，实际上都是创建了一个全新的String对象存储修改后的字符串内容，最初的String对象丝毫无改。

7.1.4 StringBuilder和StringBuffer类

```
String s= "hello";
s=s+ "hehe";
System.out.println(s);  //输出" hellohehe"
```





7.1.4 StringBuilder和StringBuffer类

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- String对象的不可变性的弊病：如果大量拼接字符串（如在循环中拼接），则每次拼接都会产生垃圾，由此消耗了大量内存，且降低了执行效率。

```
String s="1"+"2"+"3"+"4"+"5";
```

2．StringBuilder类和StringBuffer类

- StringBuilder是在Java SE 5.0引入的，在此之前Java使用的是StringBuffer类，二者的区别就是StringBuilder类是单线程的，StringBuffer是多线程安全的，支持并发访问（线程详见第11章），因此StringBuffer的开销会大些。
- 在非多线程的情况下，应该优先选择StringBuilder类。

2．StringBuilder类和StringBuffer类

- 与String的不可变性不同，StringBuilder对象代表一个可变的字符串，当一个StringBuilder对象被创建后，通过它的append()、insert()、delete()、reverse()等方法可以改变这个字符串对象的字符序列。

- StringBuilder对象默认长度为16，也可以在创建对象时指定初始长度，如果附加的字符超出可容纳长度，StringBuilder对象会自动扩容。

【例7-2】对比测试String类和StringBuilder类的拼接效率。

```
public static void main(String[] args) {
        //String对象的拼接
        String text="";
        long beginTime=System.currentTimeMillis();  //起始时间

        for(int i=0; i<20000; i++){ //循环20000次拼接字符串
                text=text+i;
        }

        long endTime=System.currentTimeMillis();     //终止时间
        System.out.println("String的执行时间：  "+(endTime-beginTime));
```

【例7-2】对比测试String类和StringBuilder类的拼接效率。

```
        //StringBuffer对象的拼接
         StringBuffer buffer = new StringBuffer("");
         beginTime=System.currentTimeMillis();
         for(int i=0; i<20000; i++){ //用StringBuffer类的append()方法拼接字符串
                    buffer.append(i);
          }
         endTime=System.currentTimeMillis();

         System.out.println("StringBuffer的执行时间：  "+(endTime-beginTime));
    }//main
}//class
```

7.1.4 StringBuilder和StringBuffer类

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- StringBuilder类中的常用方法：

（1）append()

- StringBuilder append(boolean b)
- StringBuilder append(int i)
- StringBuilder append(String s)
- ……
- append()方法有很多重载形式，允许各种类型数据的追加操作。append()将参数指定的数据转换为String后（如果需要转换的话），追加到调用该方法的StringBuilder字符串对象的末尾。

```
StringBuilder builder = new StringBuilder("This ");
builder.append("is ").append("a ").append("good ").append("idea!");  //没有产生新字符串
System.out.println(builder);  // builder对象值更新为"This is a good idea!"
```

(2) insert()

- StringBuilder insert(int offset, String s)
- 该方法将参数字符串s插入在调用该方法的StringBuilder字符串中，位置由参数offset指定。它同样有很多重载形式，允许各种类型数据的插入操作。

```
StringBuilder builder = new StringBuilder("01068911626");
builder.insert(3, "-");
System.out.println(builder);
```

输出"010-68911626"

(3) delete()

- StringBuilder delete(int start, int end)，删除调用该方法的字符串的从索引值start开始至索引值end前的子串。

```
StringBuilder builder = new StringBuilder("0123456789");
builder.delete(3, 6);
System.out.println(builder);
```

输出"0126789"

(4) reverse()

- StringBuilder reverse()
- 将调用该方法的字符串逆置。

```
StringBuilder builder = new StringBuilder("0123456789");
builder.reverse();
System.out.println(builder);
```

输出"9876543210"

3．String类与StringBuilder类的比较

（1）String类的对象具有不可变性，StringBuilder类的对象是可变的。

（2）String类重写了Object类的equals()方法，StringBuilder类没有。

```
String s1 = new String("hello");
String s2 = new String("hello");
System.out.println(s1.equals(s2)); //字符串是否相同
```

输出true

```
StringBuilder s3 = new StringBuilder("hello");
StringBuilder s4 = new StringBuilder("hello");
System.out.println(s3.equals(s4)); //引用地址是否相同
```

输出false

（3）String对象的拼接使用“+”运算符，StringBuilder类使用append()方法，效率比String类高很多。

（4）String类比StringBuilder类多了很多关于字符串处理的方法，如字符串的解析、比较大小，与其他数据类型之间的数据转换方法等。

- 结论：
  - 如果一个字符串有频繁的插入、删除、修改等操作，使用StringBuilder类。
  - 如果一个字符串需要进行丰富的串运算，则使用String类。

【例7-3】String和StringBuilder的转换。

```
public static void main(String[] args) {
        String s = "";
        //包装为StringBuilder类型
        StringBuilder builder = new StringBuilder(s);

        //利用append()提高拼接的效率
        builder.append("This ").append("is ").append("a ").append("good
").append("idea!");

        s = builder.toString();      //转换为String类型

        //利用String类的解析方法将字符串用空格分解为若干个单词
        String[] words = s.split(" ");

        //解析结果,数组[This, is, a, good, idea!]
        System.out.println(Arrays.toString(words));}
```

- 正则表达式（regular expression）
  - 1956年诞生
  - 一种强大而灵活的文本处理工具
  - 一门学科，独立存在，并不依附于某种计算机语言，而是由每种语言对其提供语法上的支持。
  - 在文本的编辑器和搜索工具中占据着非常重要的地位。
  - 正则表达式主要用于文本处理，对字符串进行查找、解析和验证，比如单词的查找替换，电子邮件、身份证号格式的校验等。这些操作都涉及字符串的模式匹配问题，模式匹配是一种复杂的字符串运算算法。正则表达式赋予一些字符特殊的含义，用它们描述字符串模式，功能化了模式匹配的过程。

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

1、预定义字符类

预定义字符	说明	预定义字符	说明
\d	匹配0~9任意一个数字	\D	匹配非0~9数字
\s	匹配\t、\n、\r等空白符	\S	匹配非空白符
\w	匹配数0~9，字母a~z，A~Z，下划线	\W	匹配非数字和字母
.	匹配任意一个字符		

“c\wt”	cat、cbt、c0t、c_t等一批以c开头、以t结尾、中间是任意字符
“.bc”	设有字符串“abcdaaebcaddbc”，“.bc”可以从中匹配出“abc”、“ebc”、“dbc”三个字符串

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

2、方括号

示例	说明
[aeiou]	表示枚举。匹配a,e,i,o,u中的任意一个字符
[a-fx-z]	-表示范围。匹配a~f和x~z范围内的任意一个字符
[^abcd]	^表示求反。匹配a,b,c,d之外的任意一个字符
[a-e&&[dfg]]	&&表示交集运算。匹配a~e与d，f，g的交集，即d

“a[xyz]c” 可以匹配axc、ayc、azc这些字符串。

“a[^\d]c” 可以匹配a0c~a9c之外的以a开头以c结尾的3个字符组成的字符串。

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

3、量词

示例	说明
X?	1≥X表达式出现的次数≥0，即0次或1次
X*	X表达式出现的次数≥0，即0次或多次
X+	X表达式出现的次数≥1，即1次或多次
X{n}	X表达式出现的次数=n次，即n次
X{,n}	X表达式出现的次数≥n次，即最少出现n次
X{n,m}	m≥X表达式出现的次数≥n，即最少出现n次，最多出现m次

“a[a-z]{3}c”

可以匹配以a开头，以c结尾，中间由a~z中的任意3个字符组成的字符串，从“abzycaadecaab?ca+abcadddc”中匹配出“abzyc”、“aadec”、“adddc”，而“aab?c”、“a+abc”则与模式不匹配。

- 常见正则表达式举例

正则表达式	匹配的内容	说明
[\u4e00-\u9fa5]	匹配一个汉字	
[\w.-]+@([a-zA-Z0-9-]+\.)+[a-zA-Z0-9]{2,4}	匹配Email地址	( )表示分组，定义满足指定规则的一个单位 .在正则中代表匹配任意一个字符，表示.本身时使用\.
((13[0-9])|(14[5,7])|(15([0-3,5-9]))|(18[0,5-9]))\d{8}	匹配手机号码	| 表示或者
[1-9][0-9]{4,11}	匹配QQ号码	
[1-9]\d{14}|[1-9]\d{17}|[1-9]\d{16}x	匹配身份证号	

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

1．matches()

- boolean matches(String regex)
- 检测调用该方法的字符串是否与给定的正则表达式匹配。

```
String email="song.yan@gc.ustb.edu.cn";
System.out.println(email.matches("[\\w.-]+@([a-zA-Z0-9-]+\\.)+[a-zA-Z0-9]{2,4}"));
```

true

7.2.2 String类中操作正则表达式的方法

- 在Java中“\”是转义字符的起始字符，具有特殊的含义，所以在正则中出现“\”的地方都需要用“\\”表示。如果要表示“\”本身的话，则需要“\\\\”。

2．replaceAll()

- String replaceAll(String regex, String replacement)。
- 该方法使用给定的replacement字符串替换调用此方法的字符串中与给定的正则表达式匹配的每个子字符串。

```
String text="aa\tbb\ncc  dd eeff";
System.out.println(text.replaceAll("\\s+",""));
```

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

3．split()

- String[] split(String regex)
- 利用给定的正则表达式将调用此方法的字符串拆分为字符串数组。

```
String text="aa\tbb\ncc  dd eeff";
String[] res = text.split("\\s+");
System.out.println(Arrays.toString(res));
```

输出[aa, bb, cc, dd, eeff]

7.2.3 Pattern和Matcher类

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- Java语言使用java.util.regex包中的Pattern和Matcher类处理正则表达式的模式匹配问题。String中之所以可以使用正则表达式，实际上就是在底层调用了这两个类中的方法。比如，当调用String的matches()方法时，底层实际是调用Pattern类的静态方法matches()。String中其他方法对正则表达式的支持也都是建立在底层Pattern和Matcher类之上的。

- 相比String而言，Pattern和Matcher类提供了更为丰富的功能运用正则表达式。

- Pattern类
  - 用于包装正则表达式，创建模式对象
    - public static Pattern compile(String regex)：将给定的正则表达式编译为模式对象。
    - public static Pattern compile(String regex, int flags)：结合给定的标志编译正则表达式，指明模式匹配时需要遵守的特殊规则，例如Pattern.CASE_INSENSITIVE表示忽略大小写。

```
String teleRegex = "((13[0-9])|(14[5,7])|(15([0-3,5-9]))|(18[0,5-9]))\\d{8}";
Pattern pattern = Pattern.compile(teleRegex);
```

- Pattern类
  - 获取Matcher匹配器对象
    - public Matcher matcher(CharSequence input)：由模式对象创建一个匹配器Matcher 对象，该匹配器将根据此模式匹配给定的input输入。

```
String teleRegex = "((13[0-9])|(14[5,7])|(15([0-3,5-9]))|(18[0,5-9]))\\d{8}";
Pattern pattern = Pattern.compile(teleRegex);
Matcher matcher = pattern.matcher("13641052992");
```

- Matcher类通过解释模式对字符序列执行匹配操作
  - 对目标字符串是否匹配进行判断
    - public boolean find()：尝试查找与模式相匹配的输入序列的下一个子序列。可以使用循环组织find()在字符序列中进行多次匹配计算。

```
String teleRegex = "((13[0-9])|(14[5,7])|(15([0-3,5-9]))|(18[0,5-9]))\\d{8}";
Pattern pattern = Pattern.compile(teleRegex);
Matcher matcher = pattern.matcher("13641052992");
System.out.println(matcher.find());  // true
```

- Matcher类通过解释模式对字符序列执行匹配操作
  - 从目标字符串中提取与正则表达式相匹配的子串，提取子串的操作通常与正则表达式的“分组”(group)概念相结合。分组是正则表达式中用“()”括起来的部分，它形成了一个局部的整体，即子串的匹配规则。
    - public String group()：返回分组匹配结果的全部信息。
    - public String group(int groupIndex)：返回分组匹配结果中的指定序号的分组信息。groupIndex等于0表示全部信息（即group(0)），实际提取到的分组索引从1开始。

【例7-5】利用正则表达式提取城市（地区）名称和编码。

北京:101010100朝阳:101010300顺义:101010400怀柔:101010500
通州:101010600昌平:101010700延庆:101010800丰台:101010900
石景山:101011000大兴:101011100房山:101011200密
云:101011300门头沟:101011400平谷:101011500八达
岭:101011600乌兰浩特:101081101

```
([\u4e00-\u9fa5]+)：(\d{9})
```

【例7-5】利用正则表达式提取城市（地区）名称和编码。

北京:101010100朝阳:101010300顺义:101010400怀柔:101010500
通州:101010600昌平:101010700延庆:101010800丰台:101010900
石景山:101011000大兴:101011100房山:101011200密
云:101011300门头沟:101011400平谷:101011500八达
岭:101011600乌兰浩特:101081101

```
String Regex = "([\u4e00-\u9fa5]+):(\\d{9})";
Pattern pattern = Pattern.compile(Regex);    //1. 包装正则表达式
Matcher matcher = pattern.matcher(districts);    //2.获取匹配器
while (matcher.find()) {  // 3.判断是否匹配
    // 4.利用group()方法提取匹配的分组信息
    System.out.println(matcher.group(1)+":\t"+matcher.group(2));
}
```

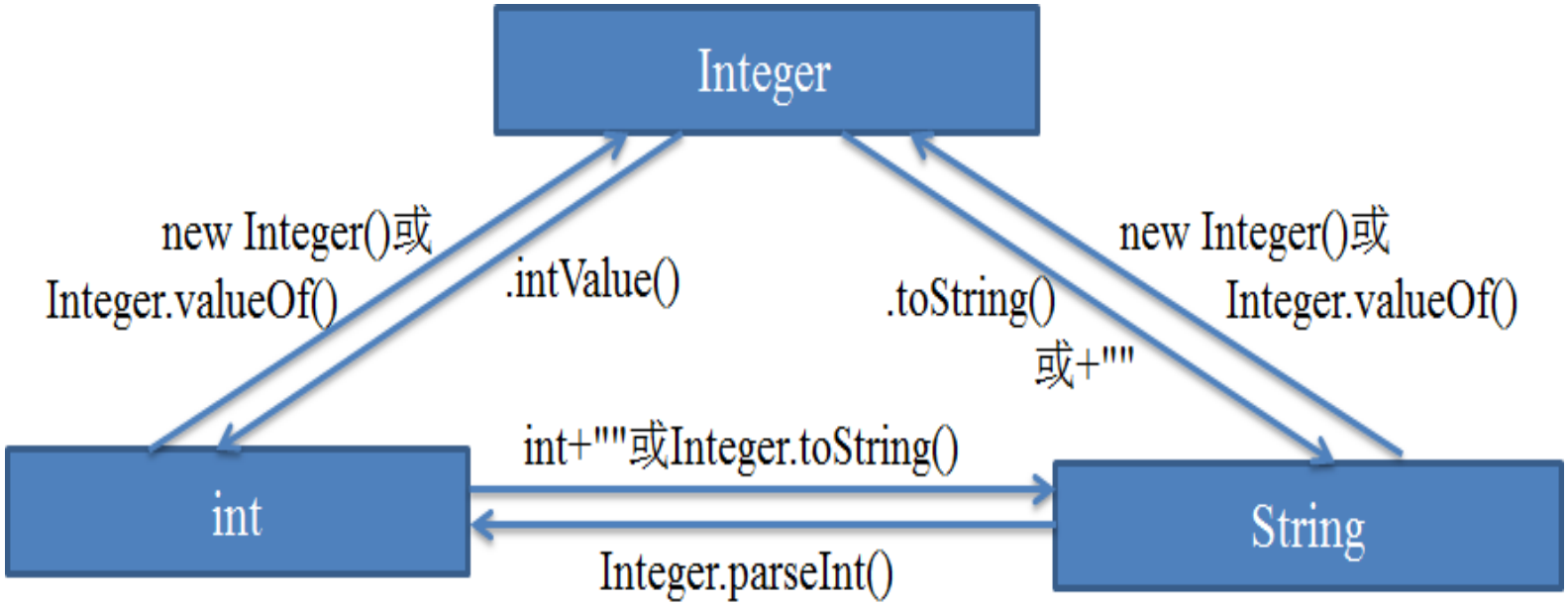
- 一切皆为对象
- 保留基本类型的好处是可以提高运算的效率
- 为了建立基本类型与引用类型之间的通信，Java为每个基本类型设计了包装类，这些包装类继承自Object，在java.lang包下，可以直接使用，包括Boolean、Character、Byte、Short、Integer、Long、Float和Double。

- 包装类使程序员可以像操作对象一样操作基本类型，通过包装类定义的方法使基本类型具有了更丰富的功能，可以实现将基本类型数据值传递给Object类型。
- 每个包装类均声明为final，因此它们的方法隐式地成为final方法，程序员不能重写这些方法。而且，基本类型包装类中的很多方法被声明为静态方法，程序员可以直接通过类名来调用这些静态方法。

7.3.1 Integer类

- Integer类是int类型的包装类
- 常量
  - MAX_VALUE和MIN_VALUE表示int类型的最大、最小值……
- 常用方法
  - int intValue()，返回调用该方法的Integer对象对应的int类型值。
  - static int parseInt(String s)，这是Integer中最常用的方法，它可以将一个数字字符组成的字符串解析为10进制整数。
  - String toString()，将调用该方法的Integer对象转换为字符串。
  - static Integer valueOf(int i)，static Integer valueOf(String s)，它们返回参数对应的一个十进制整数对象。

7.3.1 Integer类



7.3.1 Integer类

- 当使用图形用户界面与用户交互时，用户输入的数据通常都是以字符串的形式存在（在文本框中完成输入）。即便用户输入的是一个纯数字，也是一个数字字符组成的字符串，所以将字符串解析还原为用户交给程序的原始数据是经常会遇到的运算。类似的，parseDouble()、parseBoolean()等方法在每个包装类中都存在。

【例7-6】int、Integer和String数据间的类型转换示例。

```
Integer i=5;
```

自动封箱

- 这样的表述在Java SE 5.0之前是错误。

```
Integer i= Integer.valueOf(5);
```

- 从Java SE 5.0开始，编译器对基本类型进行自动封箱（AutoBoxing）和自动解封(Auto-unBoxing)。

```
Integer i=5;
int a=i;
```

```
int a=i.intValue();
```

自动解封

- 与String的对象池类似，在Java SE 5.0中对包装类（除了Float和Double）对象的常量也在JVM的运行时数据区维护了它们的常量池

Integer i1=5；向对象池写入Integer对象(5)
Integer i2=Integer.valueof(5)；将对象池中的Integer对象(5)引用赋值给i2
Integer i3=new Integer(5)；创建新的对象

```
public static void main(String[] args) {
        Integer i1=5;
        Integer i2=Integer.valueOf(5);
        Integer i3=new Integer(5);

        System.out.println(i2==i1);
        System.out.println(i2==i3);
}
```

true
false

- Integer的常量池中的数据的范围仅仅是-128~127。

```
public static void main(String[] args) {
        Integer i1=250;
        Integer i2=Integer.valueOf(250);
        Integer i3=new Integer(250);

        System.out.println(i2==i1);
        System.out.println(i2==i3);
}
```

```
false
false
```

7.4 传统日期类

7.4.1 Date类

- java.util.Date类用来处理日期及时间。
- Date类出现于JDK 1.0，因为历史太悠久，所以大部分构造方法、方法都已经过时（deprecated），不再推荐使用，取而代之的是Calendar类。

7.4.1 Date类

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

(1) Date()

- 构造方法，生成一个代表当前日期的Date对象，通过调用System.currentTimeMillis()方法获得long类型整数代表日期。这个整数是距离格林威治时间1970年1月1日0点的毫秒数，这个时间点是为了纪念Unix系统诞生。

(2) Date(long date)

- 构造方法，利用一个距离1970年1月1日0点的毫秒数生成一个Date对象。

(3) getTime()

- long getTime()，返回调用该方法的时间对象所对应的long型整数。

(4) compareTo()

- int compareTo(Date anotherDate)，比较调用该方法的日期和参数日期的大小，前者比后者大时返回1，比后者小时返回-1。

(5) before()

- boolean before(Date when)，判断调用该方法的日期是否在参数日期之前。

(6) after()

- boolean after(Date when)，判断调用该方法的日期是否在参数日期之后。

7.4.2 Calendar类

- Calendar类在java.util包中，用于表示日历，取代java.util.Date更好地处理日期和时间。
- Calendar是一个抽象类，不能用构造方法创建Calendar对象，但它提供了静态方法getInstance()来获取Calendar实例。
  - static Calendar getInstance()，使用默认时区和语言环境，根据当前时间返回一个日历。Calendar支持国际化，所以可以对日历基于的时区和语言环境进行设置。默认的时区和语言环境来自于操作系统。

7.4.2 Calendar类

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- Calendar类提供了大量访问、修改日期的方法。

（1）get()

- int get(int field)，返回调用该方法的Calendar对象的指定日历字段的取值。Field为Calendar类中的常量，例如：
- get(Calendar.DAY_OF_MONTH)返回一个代表本月第几天的整数。
- get(Calendar.MONTH)返回一个代表月的整数，范围为0~11。
- get(Calendar.Year) 返回一个代表年的整数。
- get(Calendar.DAY_OF_Year) 返回一个代表本年内第几天的整数。
- get(Calendar.DAY_OF_WEEK) 返回一个代表星期几的整数，范围为1~7，1表示星期日，2表示星期一，其他类推。

关于field取值具体可以查看API文档。

(2) set()

- void set(int field, int value)，根据给定的日历字段设置给定值，字段同上。
- void set(int year, int month, int date)
- void set(int year, int month, int date, int hour, int minute)
- void set(int year, int month, int date, int hour, int minute, int second)

7.4.2 Calendar类

- 在Calendar中，月份的取值是从0开始的，比如Calendar对象表示的日历是6月份，从Calendar对象中取出的是5；要设置日历表示12月份，送给Calendar对象的值应该是11。

（3）add()

- void add(int field, int amount)，该方法根据日历的规则，为给定的日历字段加上指定的时间量。
- add()方法的功能非常强大，当日历字段的取值超出日历规则允许的范围时，会自动进行进位或退位处理。

```
Calendar c1 = Calendar.getInstance();
c1.set(2022,1,4);        //2022-2-4
c1.add(Calendar.DAY_OF_MONTH, 30);          //2022-3-6
c1.add(Calendar.DAY_OF_MONTH, -7);          //2022-2-27
```

在2月4日的基础上加上30天，会对上一级字段Calendar.MONTH进行进位，变成3月；在3月6日的基础上减去7天会对Calendar.MONTH字段退位。add()方法自动依据日历规则进行计算。

(4) getTime()

- Date getTime()，返回一个表示调用此方法的Calendar对象的Date对象。

(5) getTimeInMillis()

- long getTimeInMillis()，返回表示调用此方法的Calendar对象的毫秒数。

(6) getActualMaximum()

- int getActualMaximum(int field)，返回指定日历字段可能的最大值。如日历字段为MONTH时，返回11；日历字段为DAY_OF_MONTH时，返回调用该方法的日历对象的月份的最大天数。

【例7-7】打印2022年2月的日历。

```
**********2022年2月日历**********
日      一      二      三      四      五      六
                1       2       3       4       5
6       7       8       9       10      11      12
13      14      15      16      17      18      19
20      21      22      23      24      25      26
27      28
```

分析：此处利用Calendar类可以打印任何日期的日历，并且，关于该月第一天是星期几、该月有多少天等运算都可以调用Calendar的方法获取。

【例7-7】打印2022年2月的日历。

```
**********2022年2月日历**********
日      一      二      三      四      五      六
                1       2       3       4       5
6       7       8       9       10      11      12
13      14      15      16      17      18      19
20      21      22      23      24      25      26
27      28
```

分析：此处利用Calendar类可以打印任何日期的日历，并且，关于该月第一天是星期几、该月有多少天等运算都可以调用Calendar的方法获取。

7.4.2 Calendar类

```
**********2022年2月日历**********
一      二      三      四      五      六      日
                1       2       3       4       5       6
7       8       9       10      11      12      13
14      15      16      17      18      19      20
21      22      23      24      25      26      27
28
```

1 2 3 4 5 6 7 day_of_week

```
**********2022年5月日历**********
一      二      三      四      五      六      日
                                                1
2       3       4       5       6       7       8
9       10      11      12      13      14      15
16      17      18      19      20      21      22
23      24      25      26      27      28      29
30      31
```

```
for(int s=1; s<=day_of_week-1; s++){
        System.out.print("\t");
}
```

7.4.2 Calendar类

1	SUNDAY
2	MONDAY
3	TUESDAY
4	WEDNESDAY
5	THURSDAY
6	FRIDAY
7	SATURDAY

Calendar.DAY_OF_WEEK

```
int day_of_week = cal.get(Calendar.DAY_OF_WEEK)-1;
if(day_of_week==0){   //7表示星期日
        day_of_week=7;
}
```


```
**********2022年2月日历**********
一      二      三      四      五      六      日
        1       2       3       4       5       6
7       8       9       10      11      12      13
14      15      16      17      18      19      20
21      22      23      24      25      26      27
28
```

```
//输出该月所有天
for(int day=1; day<=maxDay; day++){
        System.out.print(day+"\t");
        if((day+day_of_week-1)%7==0){
                System.out.println();
        }
}
```

该月最大天数

Calendar

```
int maxDay = cal.getActualMaximum(Calendar.DAY_OF_MONTH);
```

7.4.2 Calendar类

```
public static void main(String[] args) {
        Calendar cal = Calendar.getInstance();
        cal.set(Calendar.YEAR, 2022);
        cal.set(Calendar.MONTH, 1);
        cal.set(Calendar.DAY_OF_MONTH,1); //1日，依此计算该月第一天是星期几

        int year = cal.get(Calendar.YEAR);
        int month = cal.get(Calendar.MONTH)+1;
        int maxDay = cal.getActualMaximum(Calendar.DAY_OF_MONTH);  //该月最大天数
        int day_of_week = cal.get(Calendar.DAY_OF_WEEK)-1;

        //输出标题行
        System.out.println("******************"+year+"年"+month+"月日历******************");

        System.out.println("日\t一\t二\t三\t四\t五\t六");
```

```
        //计算星期，并输出之前的空白
        for(int s=1; s<day_of_week; s++){
                System.out.print("\t");
        }

        //输出该月所有天
        for(int day=1; day<=maxDay;; day++){
                System.out.print(day+"\t");
                if((day+day_of_week-1)%7==0){
                        System.out.println();
                }
        }
}
```

7.4.3 SimpleDateFormat类

- SimpleDateFormat类位于java.text包下（java.text包下还有很多关于各种格式控制的类），用于格式化日期和解析日期字符串。

- 它通过特定的pattern字符串处理日期格式。

7.4.3 SimpleDateFormat类

模板字符	日期或时间元素	模板字符	日期或时间元素
y	年	a	Am/pm 标记
M	年中的月份	h	一天中的小时数（0-23）
d	月份中的天数	k	一天中的小时数（1-24）
w	年中的周数	K	am/pm 中的小时数（0-11）
W	月份中的周数	H	am/pm 中的小时数（1-12）
D	年中的天数	m	小时中的分钟数
F	月份中的星期	s	分钟中的秒数
E	星期中的天数	S	分钟中的毫秒数

yyyy-MM-dd”表示按2016-08-05这样的格式表示一个日期。

7.4.3 SimpleDateFormat类

- 模板字符串通常在构建SimpleDateFormat对象时作为初始化参数传入。

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
```

（1）parse()

- Date parse(String text)，该方法对参数字符串text进行解析，如果按照指定的日期模板解析成功，返回得到的日期对象。

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
Date date = sdf.parse("2016-8-5");
```

如果字符串与给定的日期模板不匹配，解析将失败，并抛出ParseException异常。

(2) format()

- String format(Date date)，该方法按照调用此方法的SimpleDateFormat对象所设定的模式格式化日期型参数date，返回一个字符串。实现从Date到String类型的转换。

- Java SE 8提供了日期/时间的处理的新的API
  - 修正了过去的缺陷，API设计更为便捷。
  - 新的API在java.time包中，LocalDate和LocalTime两个类分别处理本地日期和时间（“本地”即不包含时区信息），也可以使用LocalDateTime类同时处理日期时间。

- LocalDate是带有年、月、日的日期。

（1）创建LocalDate实例

1）now()

- public static LocalDate now()
- 该方法从默认时区的系统时钟获取当前日期。例如：
- LocalDate today = LocalDate.now();

2）of()

- public static LocalDate of(int year，int month, int dayOfMonth)
  - of()方法按照指定的年、月和日创建LocalDate的实例。
  - 参数month月份的取值从1月到12，dayOfMonth表示月份的第1天到第31天，如果日期参数无效，of()方法会抛出DateTimeException异常。

```
LocalDate day1 = LocalDate.of(2022, 2,28);
LocalDate day2 = LocalDate.of(2022, 2,29); //抛出异常，Invalid date 'February 29' as '2022' is not a leap year
```

3）ofYearDay()

- public static LocalDate ofYearDay(int year, int dayOfYear)
  - ofYearDay()方法按照一年中的第dayOfYear天创建LocalDate的实例，dayOfYear取值从1到366，如果超出范围ofYearDay()方法同样抛出DateTimeException异常。

```
LocalDate day3 = LocalDate.ofYearDay(2022, 100);
LocalDate day4 = LocalDate.ofYearDay(2022, 400);  //Invalid value for DayOfYear (valid values 1 - 365/366): 400
```

（2）获取日期信息的方法

- LocalDate提供了大量便捷的获取日期信息的方法，如getYear()，getMonthValue()，getDayOfMonth()，getDayOfWeek()，getDayOfYear()等。
- public DayOfWeek getDayOfWeek()
- getDayOfWeek()获取当前日期是星期几时，返回数据为枚举类型DayOfWeek。
- 枚举是从避免混淆int值的含义、提高程序可读性的角度，采用符号名称代替数字的数据类型，DayOfWeek中用MONDAY表示1，TUESDAY表示2……，SUNDAY表示7。如果需要访问原语int值，使用DayOfWeek中的getValue()方法。

7.5.1 LocalDate类

```
date.getDayOfWeek().getValue()
```

1	MONDAY
2	TUESDAY
3	WEDNESDAY
4	THURSDAY
5	FRIDAY
6	SATURDAY
7	SUNDAY

LocalDate：DayOfWeek

（3）用绝对值调整日期的方法

- LocalDate提供了with()方法用于调整日期，并返回该日期的调整后副本。
- public LocalDate with(TemporalAdjuster adjuster)
- 调整根据参数TemporalAdjuster对象进行，java.time.temporal包下的TemporalAdjusters类提供了返回各种TemporalAdjuster对象的静态方法，如TemporalAdjusters中的lastDayOfMonth()方法返回月内最后一天的调整器。

- public static TemporalAdjuster lastDayOfMonth()
  - 使用如下语句即可将一个LocalDate对象（下面用**表示）的时间调整该改月的最后一天，从而获取该月有多少天。

**.with(TemporalAdjusters.lastDayOfMonth()).getDayOfMonth()

（4）用相对值调整日期的方法

- LocalDate提供了相对调整日期的方法，在当前日期的基础上加（减）一个日期量，如plusDays()，plusMonths()，plusWeeks()，plusYears()等。
- public LocalDate plusWeeks(long weeksToAdd)
- 在日期对象上添加指定的周数，返回LocalDate对象的副本。

7.5.1 LocalDate类

- 假设开学日期为2022-2-28日，教学运行16周，16周后开始期末考试，考试从哪天开始呢?

```
LocalDate termBeginsDate = LocalDate.of(2022, 2, 28);
//加上16周
LocalDate termEndsDate = termBeginsDate.plusWeeks(16);
System.out.println(termEndsDate);  //2022-06-20
```

7.5.2 DateTimeFormatter

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- DateTimeFormatter 类位于java.time.format包下，用于格式化和解析日期/时间对象

模板字符	日期或时间元素	模板字符	日期或时间元素
y	year-of-era	H	hour-of-day (0-23)
M	month-of-year	m	minute-of-hour
d	day-of-month	s	second-of-minute
E	day-of-week	a	am-pm-of-day

7.5.2 DateTimeFormatter

- 通过ofPattern()方法指定模式来定制日期格式
  - DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

- 通过format()方法用于对日期时间对象进行格式化。
  - public String format(TemporalAccessor temporal)
    - 参数为要格式化的日期时间对象，如果格式化过程中发生错误会抛出DateTimeException异常。

7.5.2 DateTimeFormatter

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

【例7-8】使用DateTimeFormatter 格式化输出日期时间对象。

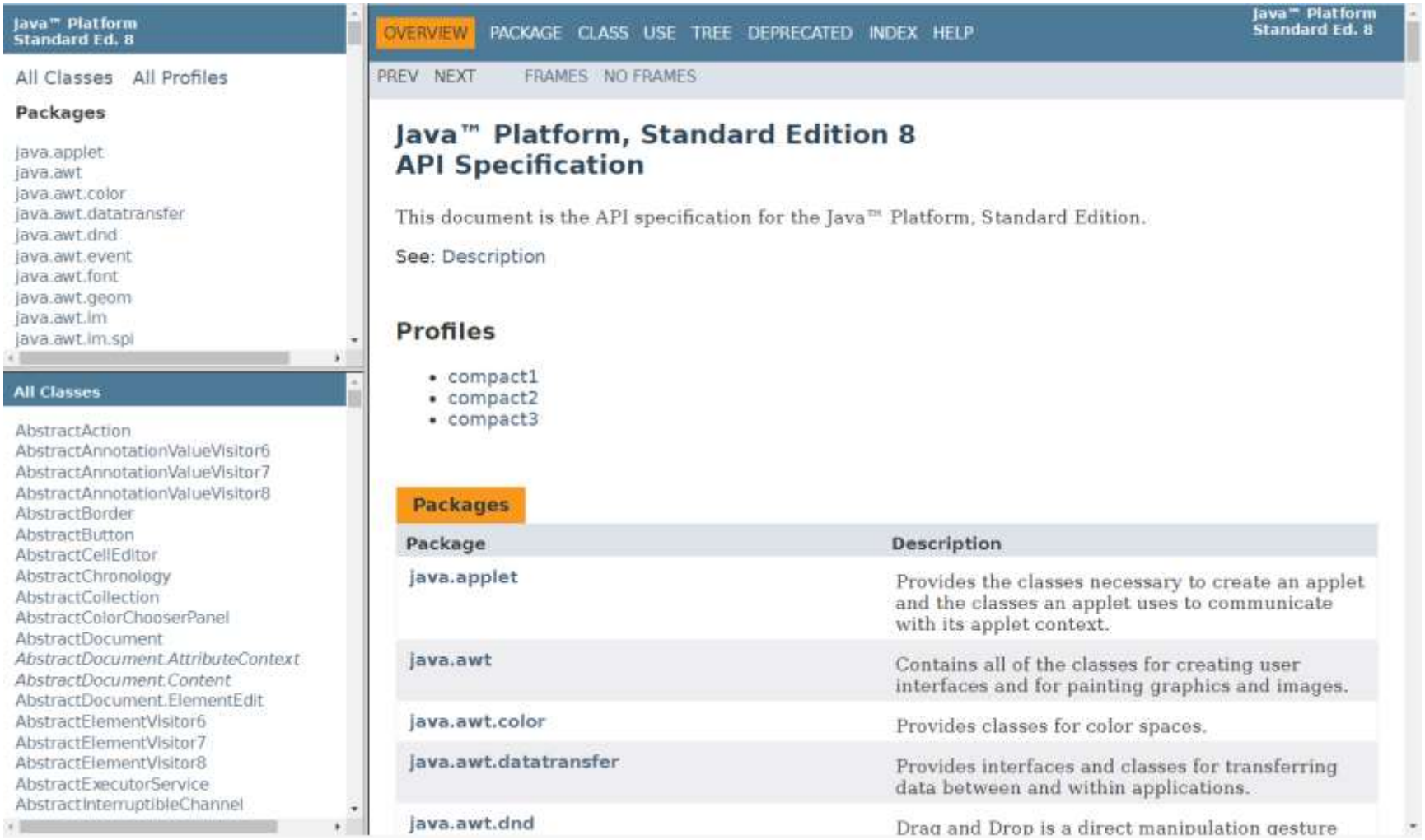
7.5.3 与java.sql的类型转换

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

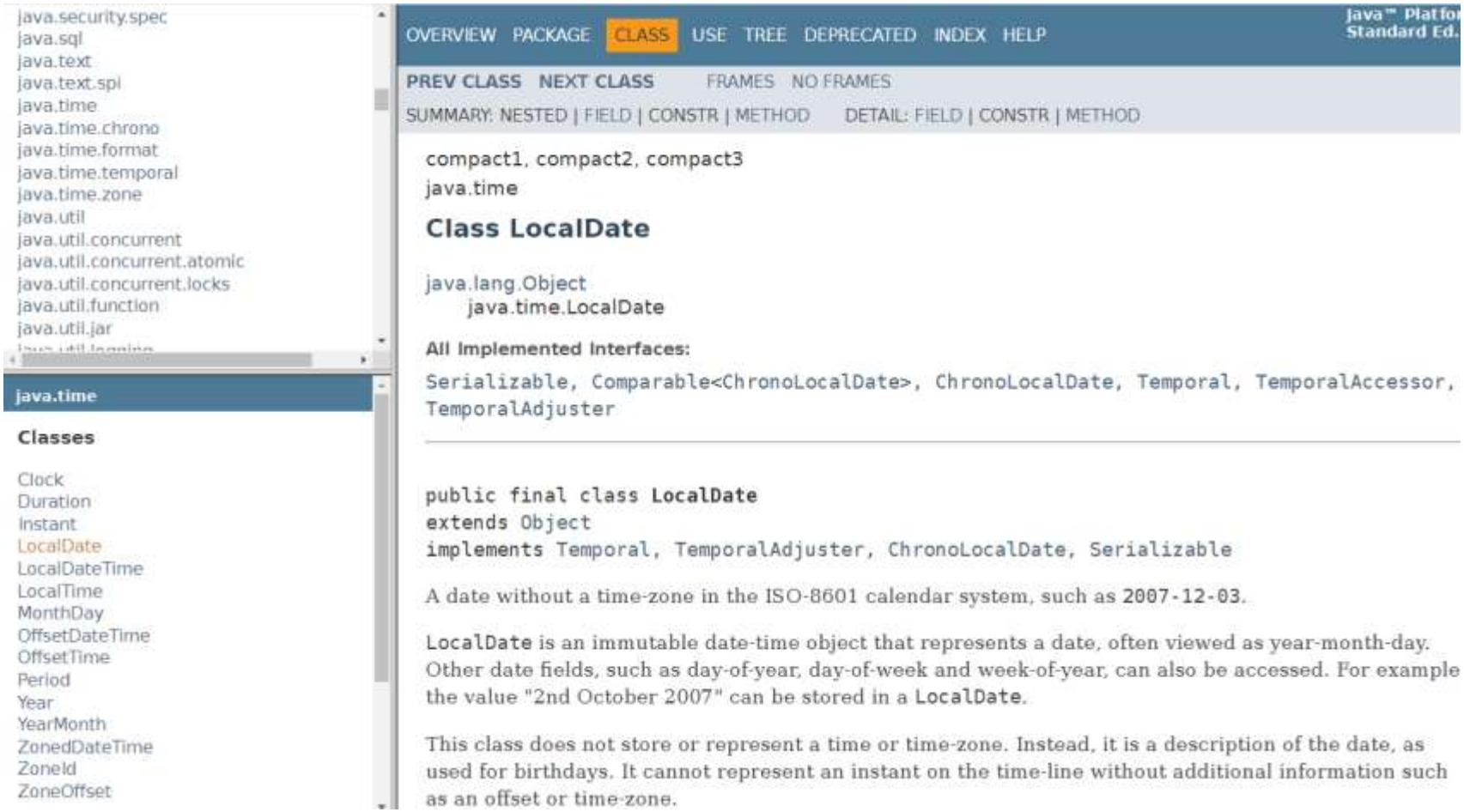
- java.sql包下的日期/时间类是数据库访问时日期/时间数据的存储类型，Java SE 8在java.sql中的日期/时间类中增加了与java.time包进行类型转换的方法。
- java.sql.Date对象转换为LocalDate类型
  - public LocalDate toLocalDate()
- LocalDate对象转换为java.sql.Date类型
  - public static Date valueOf(LocalDate date)

```
LocalDate day = LocalDate.of(2022,  2,  4);
java.sql.Date dayOfSql = java.sql.Date.valueOf(day);   //LocalDate->java.sql.Date
LocalDate dayOfLocateDate = dayOfSql.toLocalDate();  //java.sql.Date->LocalDate
```

7.6 阅读API文档


Java™ Platform
Standard Ed. 8
All Classes All Profiles
Packages
java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font
java.awt.geom
java.awt.im
java.awt.im.spi
All Classes
AbstractAction
AbstractAnnotationValueVisitor6
AbstractAnnotationValueVisitor7
AbstractAnnotationValueVisitor8
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractChronology
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeContext
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractElementVisitor6
AbstractElementVisitor7
AbstractElementVisitor8
AbstractExecutorService
AbstractInterruptibleChannel
OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
Java™ Platform
Standard Ed. 8
PREV NEXT FRAMES NO FRAMES
Java™ Platform, Standard Edition 8
API Specification
This document is the API specification for the Java™ Platform, Standard Edition.
See: Description
Profiles
compact1
compact2
compact3
Packages
Package
Description
java.applet
Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt
Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color
Provides classes for color spaces.
java.awt.datatransfer
Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd
Drag and Drop is a direct manipulation gesture

7.6 阅读API文档


java.security.spec
java.sql
java.text
java.text.spi
java.time
java.time.chrono
java.time.format
java.time.temporal
java.time.zone
java.util
java.util.concurrent
java.util.concurrent.atomic
java.util.concurrent.locks
java.util.function
java.util.jar
java.time
Classes
Clock
Duration
Instant
LocalDate
LocalDateTime
LocalTime
MonthDay
OffsetDateTime
OffsetTime
Period
Year
YearMonth
ZonedDateTime
ZoneId
ZoneOffset
OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
compact1, compact2, compact3
java.time
Class LocalDate
java.lang.Object
java.time.LocalDate
All Implemented Interfaces:
Serializable, Comparable<ChronoLocalDate>, ChronoLocalDate, Temporal, TemporalAccessor, TemporalAdjuster
public final class LocalDate
extends Object
implements Temporal, TemporalAdjuster, ChronoLocalDate, Serializable
A date without a time-zone in the ISO-8601 calendar system, such as 2007-12-03.
LocalDate is an immutable date-time object that represents a date, often viewed as year-month-day. Other date fields, such as day-of-year, day-of-week and week-of-year, can also be accessed. For example the value "2nd October 2007" can be stored in a LocalDate.
This class does not store or represent a time or time-zone. Instead, it is a description of the date, as used for birthdays. It cannot represent an instant on the time-line without additional information such as an offset or time-zone.

- http://www.weather.com.cn/weather/101010100.shtml

```
<ul class="t clearfix">
      <li class="sky skyid lv3 on">
        <h1>13日（今天）</h1>
        <big class="png40 d15"></big><big class="png40 n14"></big>
        <p title="中雪转小雪" class="wea">中雪转小雪</p>
        <p class="tem"><span>-2</span>/<i>-6℃</i></p>
        <p class="win">
          <em><span title="东风" class="E"></span><span title="北风" class="N"></span></em>
          <i><3级</i>
        </p>
      </li>
      <li class="sky skyid lv3">
        <h1>14日（明天）</h1>
       <big class="png40 d01"></big><big class="png40 n01"></big>
       <p title="多云" class="wea">多云</p>
        <p class="tem"><span>0</span>/<i>-7℃</i></p>
        <p class="win">
          <em><span title="北风" class="N"></span><span title="东北风" class="NE"></span></em>
          <i><3级</i>
        </p>
      </li>
      ……
</ul>
```

```
<li class="sky skyid lv3 on">
    <h1>13 日（今天）</h1>
    <big class="png40 d15"></big><big class="png40 n14"></big>
    <p title="中雪转小雪" class="wea">中雪转小雪</p>
    <p class="tem"><span>-2</span>/<i>-6℃</i></p>
    <p class="win">
        <em><span title="东风" class="E"></span><span title="北风" class="N"></span></em>
        <i>3 级</i>
    </p>
</li>
```

```
<li class=\"sky.*?
<p.*?class=\"wea\">(.+?)</p>
<p.*?<span>(.+?)</span>.*?
<i>(.+?)</i>.*?"
<span title=\"(.+?)\".*?
<span title=\"(.+?)\".*?
<i>(.+?)</i>.*?
</li>
```

.* 表示大于等于0个任意字符
.+ 表示大于等于1个任意字符

在量词+和*的后面加上问号“？”是禁止量词的贪婪特性，使该部分尽可能少的进行匹配，防止错过有效数据的提取。

```
public static void main(String[] args) {
    String reg1 = "aa(\\d+)";
    Pattern pattern = Pattern.compile(reg1);
    Matcher matcher = pattern.matcher("aa2343ddd");
    matcher.find();
    System.out.println(matcher.group(1));

    String reg2 = "aa(\\d+?)";
    pattern = Pattern.compile(reg2);
    matcher = pattern.matcher("aa2343ddd");
    matcher.find();
    System.out.println(matcher.group(1));
}
```

2343

2

7.7 综合实践—天气预报信息提取

```
<li class="sky skyid lv3 on">
    <h1>13 日（今天）</h1>
    <big class="png40 d15"></big><big class="png40 n14"></big>
    <p title="中雪转小雪" class="wea">中雪转小雪</p>
    <p class="tem"><span>-2</span>/<i>-6℃</i></p>
    <p class="win">
        <em><span title="东风" class="E"></span><span title="北风" class="N"></span></em>
        <i>3 级</i>
    </p>
</li>
```

```
<li class=\"sky.*?
<p.*?class=\"wea\">(.+?)</p>
<p.*?<span>(.+?)</span>.*?
<i>(.+?)</i>.*?"
<span title=\"(.+?)\".*?
<span title=\"(.+?)\".*?
<i>(.+?)</i>.*?
</li>
```

分组1：天气情况
分组2：最高气温
分组3：最低气温
分组4：风向1
分组5：风向2
分组6：风力

7.7 综合实践—天气预报信息提取


```
<li class=\"sky.*?
<p.*?class=\"wea\">(.+?)</p>
<p.*?<span>(.+?)</span>.*?
<i>(.+?)</i>.*?"
<span title=\"(.+?)\".*?
<span title=\"(.+?)\".*?
<i>(.+?)</i>.*?
</li>
```

分组1：天气情况

分组2：最高气温

分组3：最低气温

分组4：风向1

分组5：风向2

分组6：风力

```
String dayReg =
        "<li class=\"sky.*?" +
        "<p.*?class=\"wea\">(.+?)</p>" +      // 分组1：天气情况
        "<p.*?<span>(.+?)</span>.*?" +        // 分组2：最高气温
        "<i>(.+?)</i>.*?" +                   // 分组3：最低气温
        "<span title=\"(.+?)\".*?" +          // 分组4：风向1
        "<span title=\"(.+?)\".*?" +          // 分组5：风向2
        "<i>(.+?)</i>.*?" +                   // 分组6：风力
        "</li>";
```

```
public class WeatherOfDay{
      private LocalDate day;      //日期
      private String info;        //天气情况
      private int maxTemp;        //最高气温
      private int minTemp;        //最低气温
      private String windDirection;    //风向
      private int windForce;           //风力
      ……
}
```

本章思维导图