第7章　常用工具类

本章知识点

- 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

- 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。

7.1 字符串处理类

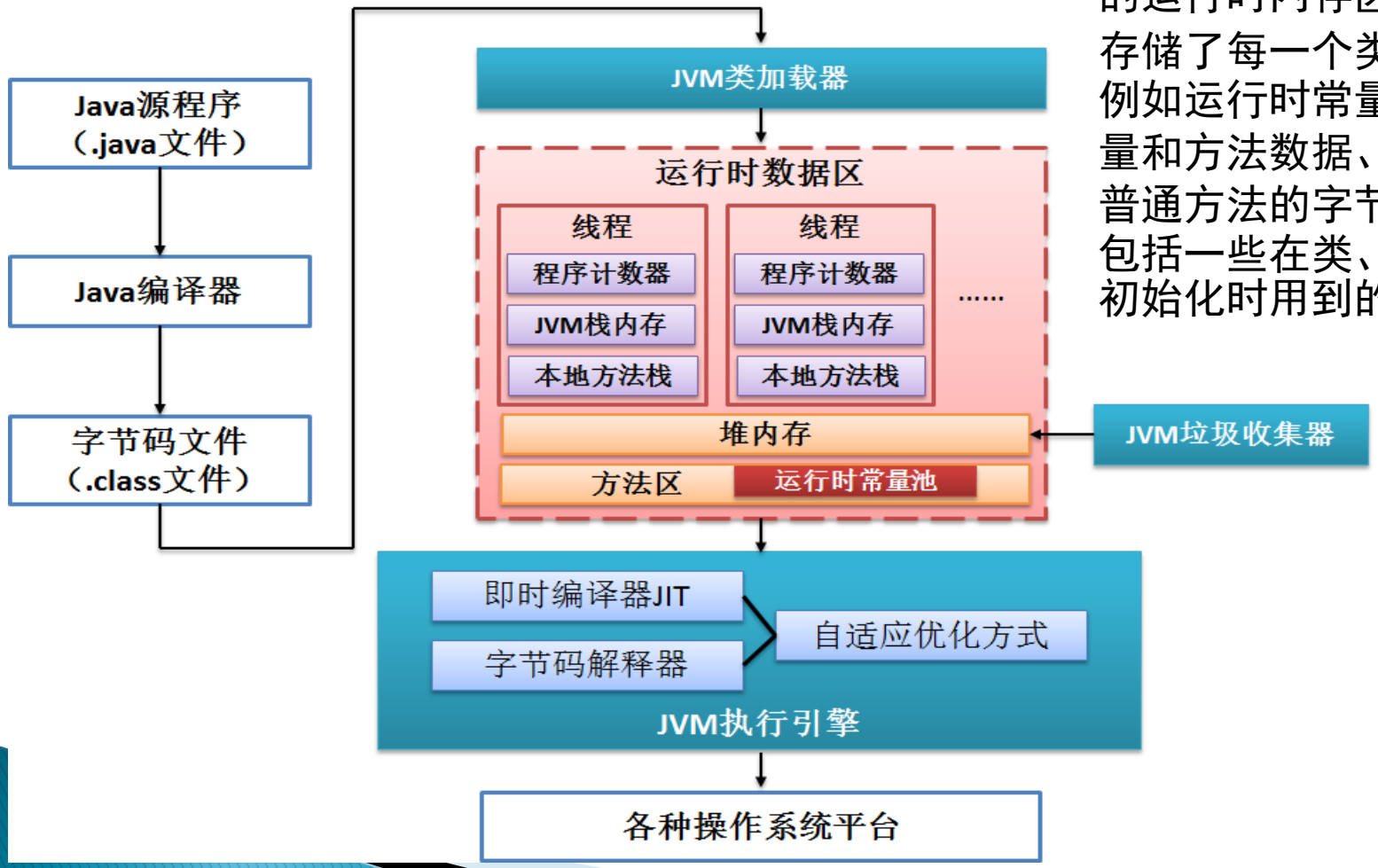
7.1.1 Java中String对象的管理

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。

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

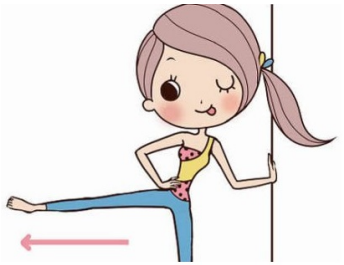
7.1.1 Java中String对象的管理



方法区是可供各条线程共享的运行时内存区域。

存储了每一个类的结构信息，例如运行时常量池、成员变量和方法数据、构造方法和普通方法的字节码内容、还包括一些在类、实例、接口初始化时用到的特殊方法。

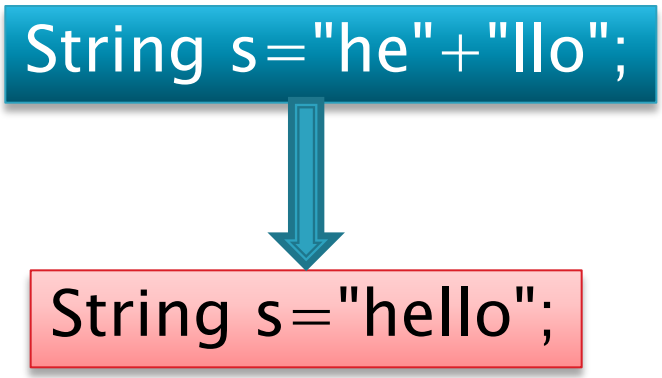
7.1.1 Java中String对象的管理



- 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。

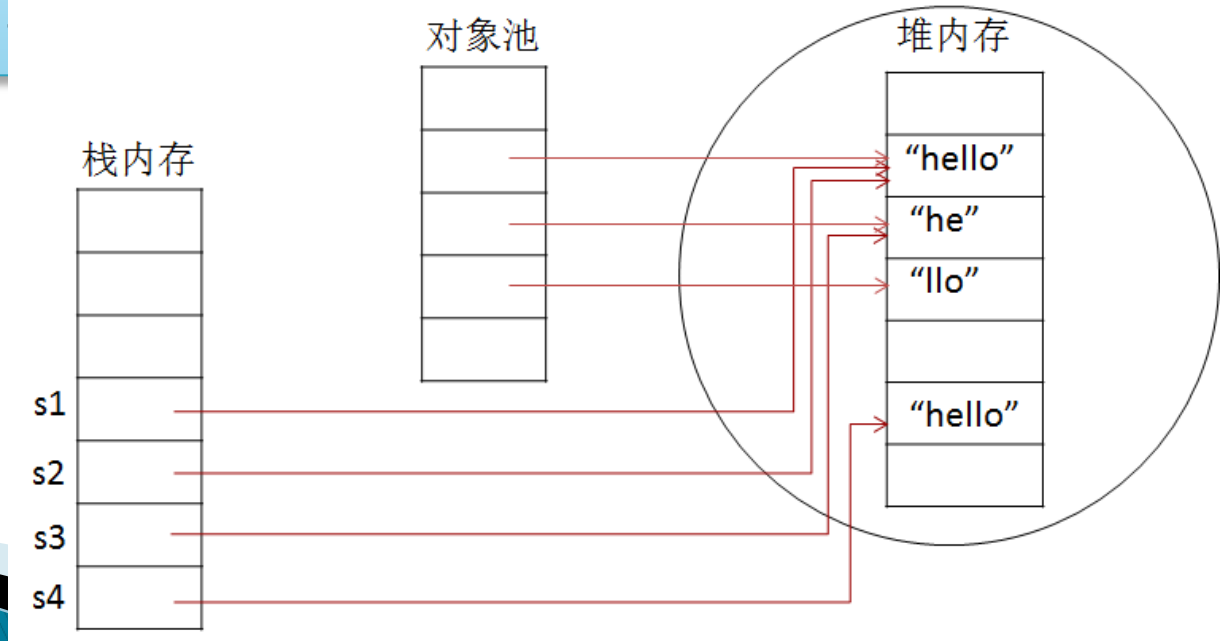


7.1.1 Java中String对象的管理

【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？



7.1.2 String类的常用方法

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'

7.1.2 String类的常用方法

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);
```

7.1.2 String类的常用方法

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
```

7.1.2 String类的常用方法

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...");
}
```

7.1.2 String类的常用方法

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

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

7.1.2 String类的常用方法

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

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!"

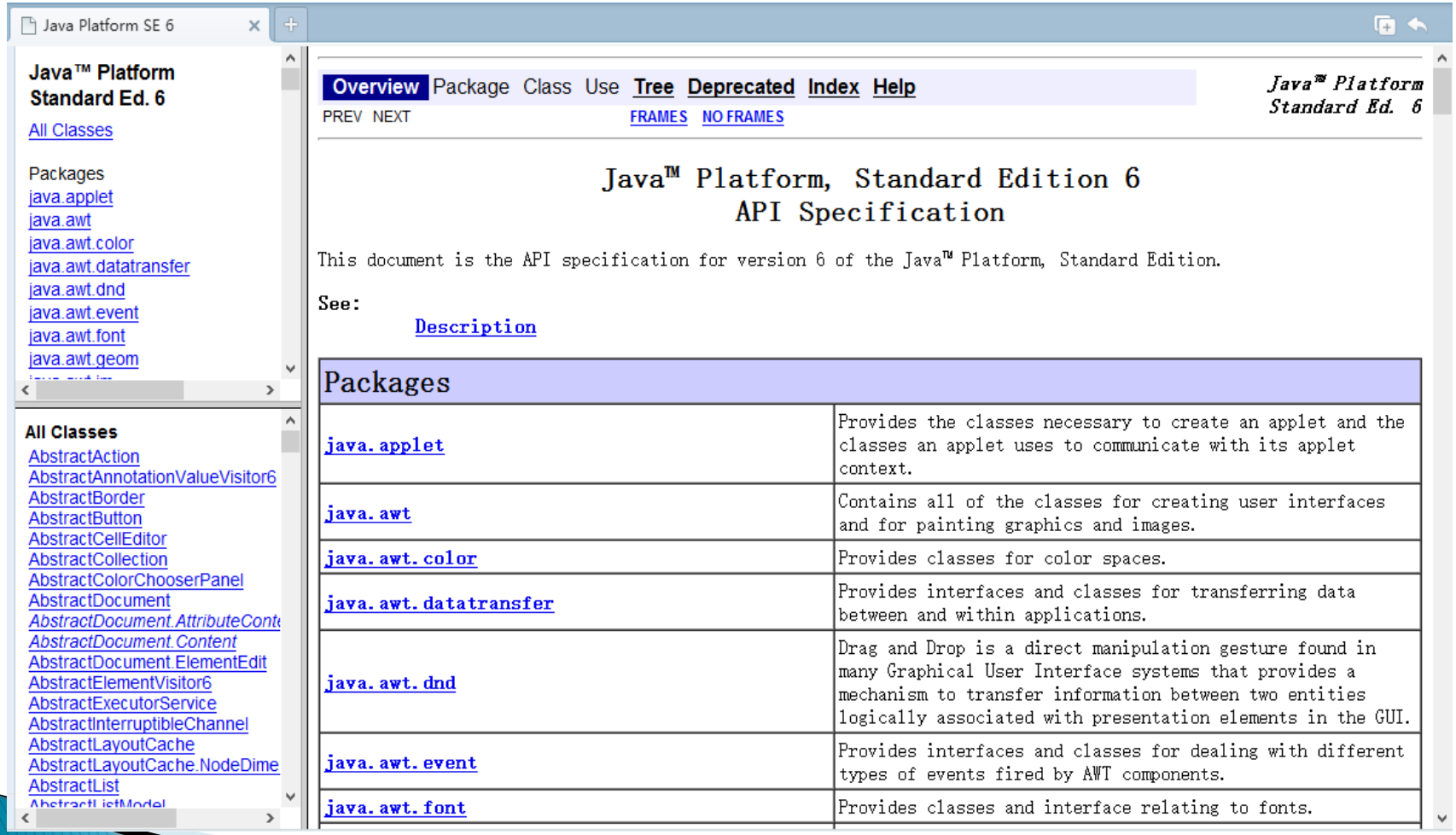
7.1.2 String类的常用方法

9. trim()

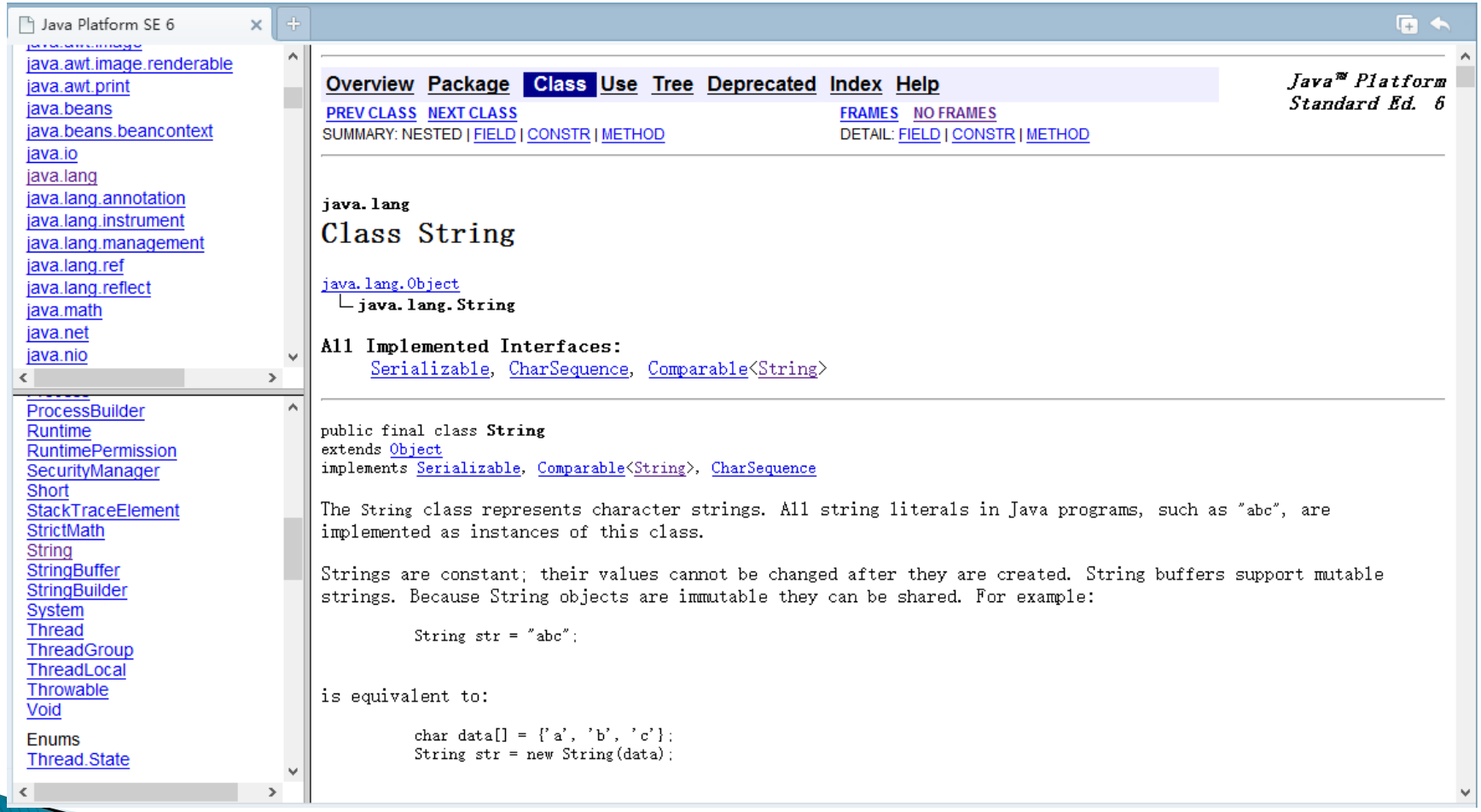
- String trim()，将调用该方法的String的前后空格都删除，返回新字符串。

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

7.1.3 阅读API文档

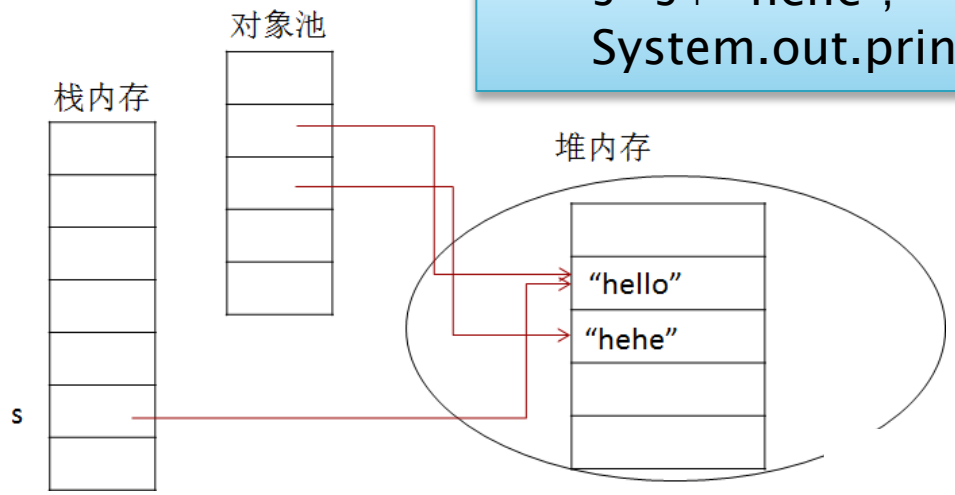

Java Platform SE 6
Java™ Platform
Standard Ed. 6
All Classes
Packages
java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font
java.awt.geom
All Classes
AbstractAction
AbstractAnnotationValueVisitor6
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractElementVisitor6
AbstractExecutorService
AbstractInterruptibleChannel
AbstractLayoutCache
AbstractLayoutCache.NodeDime
AbstractList
Overview Package Class Use Tree Deprecated Index Help
PREV NEXT
FRAMES NO FRAMES
Java™ Platform
Standard Ed. 6
Java™ Platform, Standard Edition 6
API Specification
This document is the API specification for version 6 of the Java™ Platform, Standard Edition.
See:
Description
Packages
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 found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event
Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font
Provides classes and interface relating to fonts.

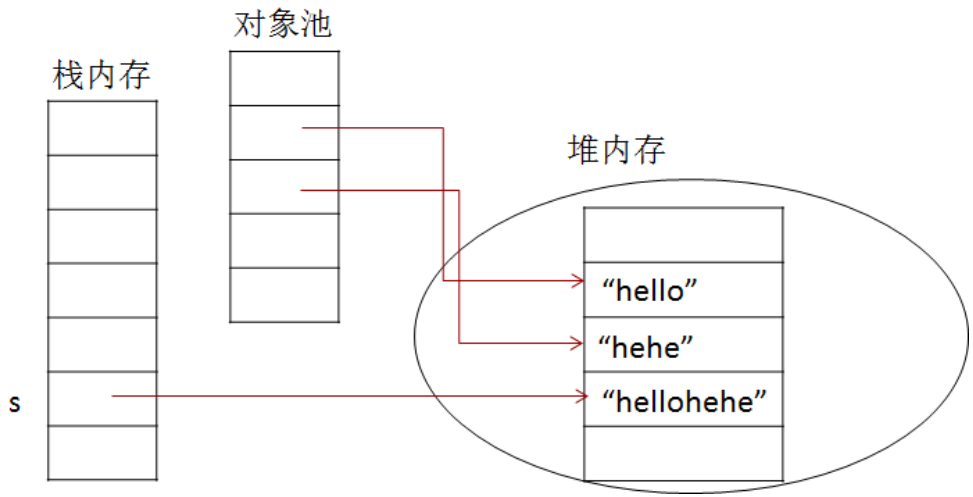
7.1.3 阅读API文档


Java Platform SE 6
java.awt.image.renderable
java.awt.print
java.beans
java.beans.beancontext
java.io
java.lang
java.lang.annotation
java.lang.instrument
java.lang.management
java.lang.ref
java.lang.reflect
java.math
java.net
java.nio
ProcessBuilder
Runtime
RuntimePermission
SecurityManager
Short
StackTraceElement
StrictMath
String
StringBuffer
StringBuilder
System
Thread
ThreadGroup
ThreadLocal
Throwable
Void
Enums
Thread.State
Overview Package Class Use Tree Deprecated Index Help
Java™ Platform Standard Ed. 6
PREV CLASS NEXT CLASS
FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD
java.lang
Class String
java.lang.Object
java.lang.String
All Implemented Interfaces:
Serializable, CharSequence, Comparable<String>
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.
Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:
String str = "abc";
is equivalent to:
char data[] = {'a', 'b', 'c'};
String str = new String(data);

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

- 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";
```

7.1.4 StringBuilder和StringBuffer类

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。

7.1.4 StringBuilder和StringBuffer类

【例7-3】对比测试String类和StringBuilder类的拼接效率。

7.1.4 StringBuilder和StringBuffer类

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));

StringBuilder s3 = new StringBuilder("hello");
StringBuilder s4 = new StringBuilder("hello");
System.out.println(s3.equals(s4));
```

输出true

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。

7.1.4 StringBuilder和StringBuffer类

【例7-4】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));}
```

7.2 正则表达式

- 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。

7.2.1 正则表达式的语法

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

7.2.1 正则表达式的语法

2、方括号

示例	说明
[abcd]	表示枚举。匹配a,b,c,d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-z&&[def]]	&&表示交集运算。匹配a~z与d，e，f的交集，即d，e或f
[a-d[m-p]]	表示并集运算。匹配a~d和m~p范围内的字符，也可直接写作[a-dm-p]

“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。

7.2.1 正则表达式的语法

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。

7.2.1 正则表达式的语法

- 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	
[a-zA-Z0-9_+\.-]+@([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	

7.2.2 String类中操作正则表达式的方法

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("[a-zA-Z0-9_+.-]+@([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“\\\\”。

7.2.2 String类中操作正则表达式的方法

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\rcc  dd eeff";
System.out.println(text.replaceAll("\\s+",""));
```

7.2.2 String类中操作正则表达式的方法

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。

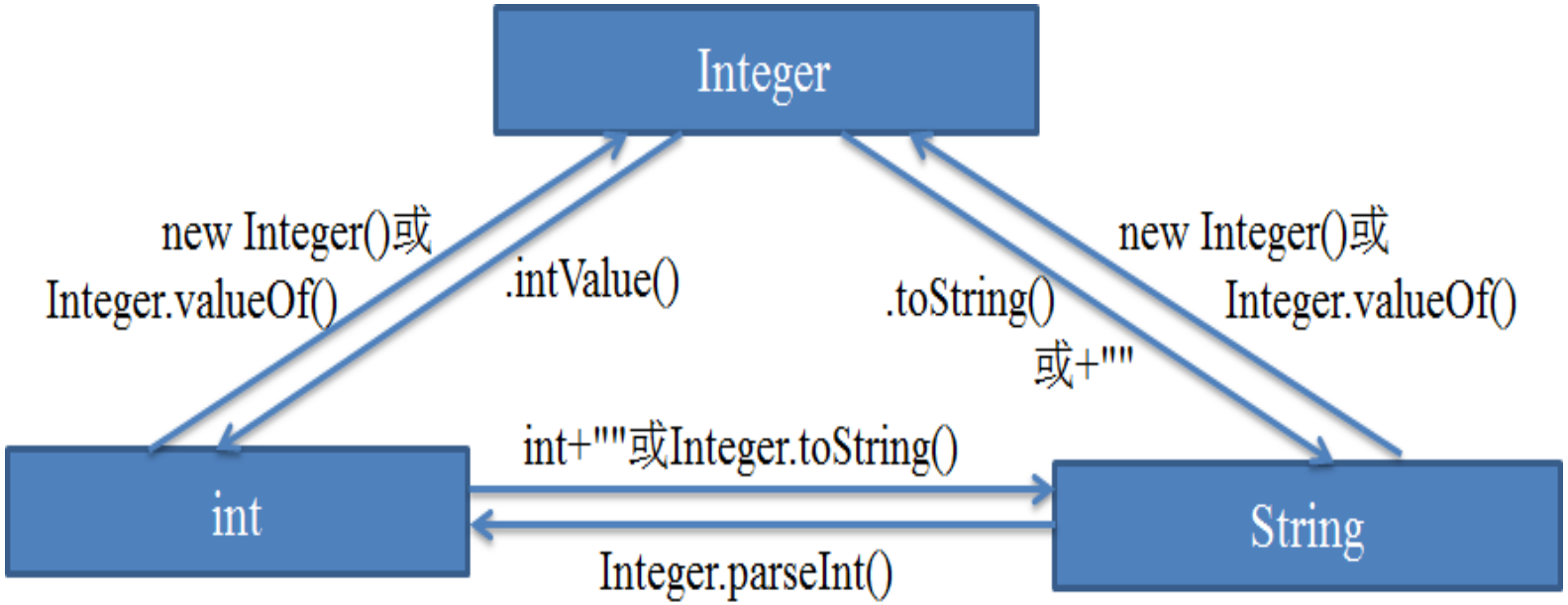
```
String text="aa\tbb\ncc  dd eeff";
String[] res = text.split("\\s+");
System.out.println(Arrays.toString(res));
```

输出[aa, bb, cc, dd, eeff]

7.3 包装类

- 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



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。

7.3.1 Integer类

【例7-5】int、Integer和String数据间的类型转换示例。

7.3.2 自动封箱、[redacted]封

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

7.3.2 自动封箱[redacted]

- 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 日期类

7.4.1 Date类

（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。

7.4.1 Date类

（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。

7.4.2 Calendar类

（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。

7.4.2 Calendar类

（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。

（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。

7.4.2 Calendar类

【例7-6】打印2016年8月的日历。

```
********************2016年8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      29      30      31
```

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。

7.4.3 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");
```

7.4.3 SimpleDateFormat类

（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");
```

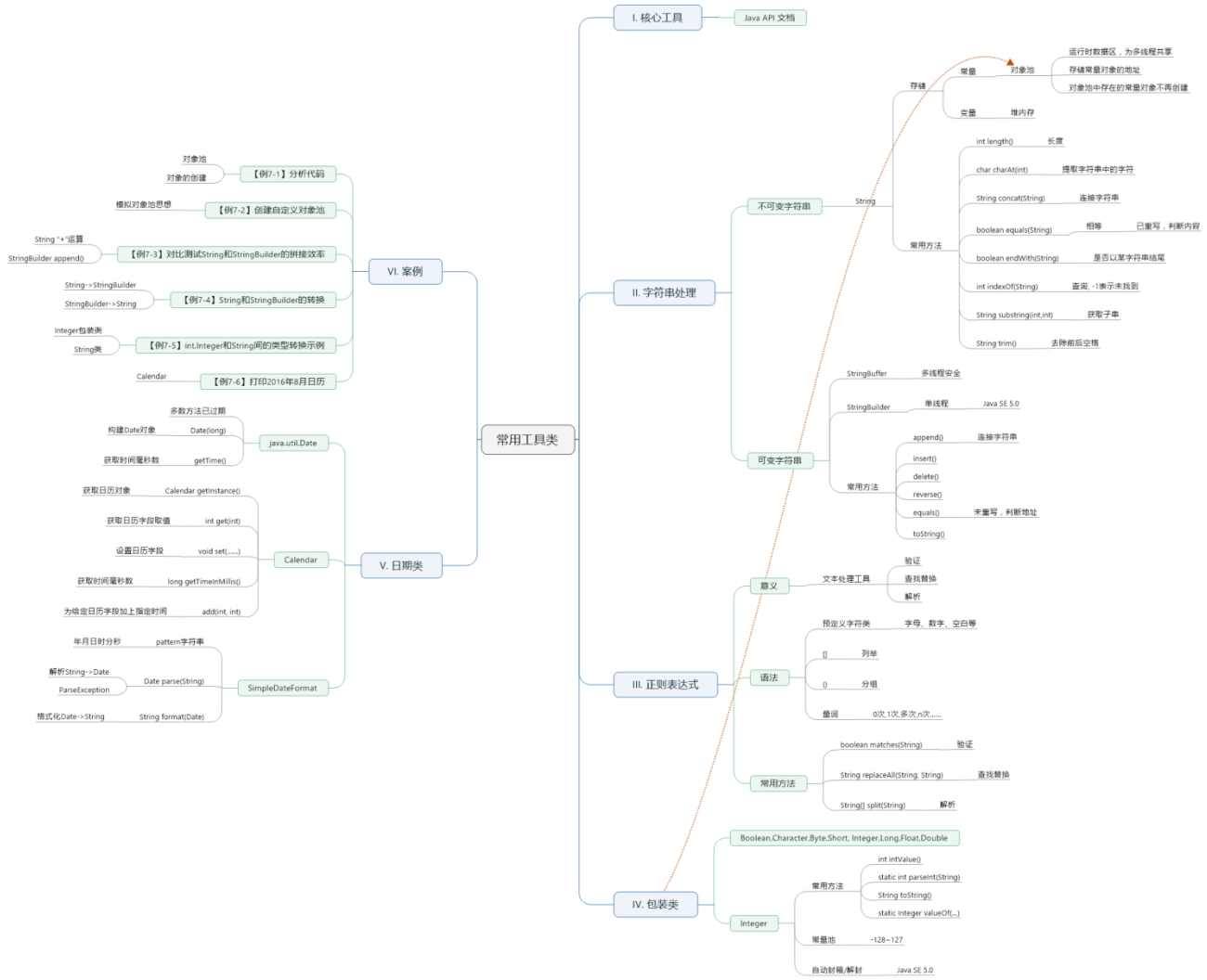
如果字符串与给定的日期模板不匹配，解析将失败，并抛出ParseException异常。

7.4.3 SimpleDateFormat类

（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。

本章思维导图


常用工具类
I. 核心工具
Java API 文档
II. 字符串处理
不可变字符串
String
存储
常量
对象池
运行时数据区，为多线程共享
存储常量对象的地址
对象池中存在的常量对象不再创建
变量
堆内存
常用方法
int length()
长度
char charAt(int)
提取字符串中的字符
String concat(String)
连接字符串
boolean equals(String)
相等
已重写，判断内容
boolean endWith(String)
是否以某字符串结尾
int indexOf(String)
查询，-1表示未找到
String substring(int,int)
获取子串
String trim()
去除前后空格
可变字符串
StringBuffer
多线程安全
StringBuilder
单线程
Java SE 5.0
常用方法
append()
连接字符串
insert()
delete()
reverse()
equals()
未重写，判断地址
toString()
III. 正则表达式
意义
文本处理工具
验证
查找替换
解析
语法
预定义字符类
字母、数字、空白等
[]
列举
()
分组
量词
0次,1次,多次,n次,.....
常用方法
boolean matches(String)
验证
String replaceAll(String, String)
查找替换
String[] split(String)
解析
IV. 包装类
Boolean,Character,Byte,Short, Integer,Long,Float,Double
Integer
常用方法
int intValue()
static int parseInt(String)
String toString()
static Integer valueOf(...)
常量池
-128~127
自动封箱/解封
Java SE 5.0
V. 日期类
java.util.Date
多数方法已过期
构建Date对象
Date(long)
获取时间毫秒数
getTime()
Calendar
获取日历对象
Calendar getInstance()
获取日历字段取值
int get(int)
设置日历字段
void set(.....)
获取时间毫秒数
long getTimeInMillis()
为给定日历字段加上指定时间
add(int, int)
SimpleDateFormat
年月日时分秒
pattern字符串
解析String->Date
Date parse(String)
ParseException
格式化Date->String
String format(Date)
VI. 案例
对象池
对象的创建
【例7-1】分析代码
模拟对象池思想
【例7-2】创建自定义对象池
String "+"运算
StringBuilder append()
【例7-3】对比测试String和StringBuilder的拼接效率
String->StringBuilder
StringBuilder->String
【例7-4】String和StringBuilder的转换
Integer包装类
String类
【例7-5】int,Integer和String间的类型转换示例
Calendar
【例7-6】打印2016年8月日历