

# 第 16 天常用 API

## 今日内容介绍

◆ 正则表达式

◆ Date

◆ DateFormat

◆ Calendar

## 第 1 章 正则表达式

### 1.1 正则表达式的概念

**正则表达式**（英语：Regular Expression，在代码中常简写为 regex）。

正则表达式是一个字符串，使用单个字符串来描述、用来定义匹配规则，匹配一系列符合某个句法规则的字符串。在开发中，正则表达式通常被用来检索、替换那些符合某个规则的文本。

### 1.2 正则表达式的匹配规则

参照帮助文档，在 Pattern 类中有正则表达式的规则定义，正则表达式中明确区分大小写字母。我们来学习语法规则。

正则表达式的语法规则：

**字符：x**

含义：代表的是字符 x

例如：匹配规则为 **"a"**，那么需要匹配的字符串内容就是 **"a"**

**字符：\\**

**含义：**代表的是反斜线字符'

例如：匹配规则为**"\\"**，那么需要匹配的字符串内容就是 **"\"**

**字符：\t**

**含义：**制表符

例如：匹配规则为**"\t"**，那么对应的效果就是产生一个制表符的空间

**字符：\n**

**含义：**换行符

例如：匹配规则为**"\n"**，那么对应的效果就是换行,光标在原有位置的下一行

**字符：\r**

**含义：**回车符

例如：匹配规则为**"\r"**，那么对应的效果就是回车后的效果,光标来到下一行行首

**字符类：[abc]**

**含义：**代表的是字符 a、b 或 c

例如：匹配规则为**"[abc]"**，那么需要匹配的内容就是字符 a，或者字符 b，或字符 c 的

一个

**字符类：[^abc]**

**含义：**代表的是除了 a、b 或 c 以外的任何字符

例如：匹配规则为"**[^abc]**"，那么需要匹配的内容就是不是字符 a，或者不是字符 b，或不是字符 c 的任意一个字符

### 字符类：**[a-zA-Z]**

含义：代表的是 a 到 z 或 A 到 Z，两头的字母包括在内

例如：匹配规则为"**[a-zA-Z]**"，那么需要匹配的是一个大写或者小写字母

### 字符类：**[0-9]**

含义：代表的是 0 到 9 数字，两头的数字包括在内

例如：匹配规则为"**[0-9]**"，那么需要匹配的是一个数字

### 字符类：**[a-zA-Z\_0-9]**

含义：代表的字母或者数字或者下划线(即单词字符)

例如：匹配规则为"**[a-zA-Z\_0-9]**"，那么需要匹配的是一个字母或者是一个数字或一个下划线

### 预定义字符类：**.**

含义：代表的是任何字符

例如：匹配规则为"**.**"，那么需要匹配的是一个任意字符。如果，就想使用 **.** 的话，使用匹配规则"**\\.**"来实现

### 预定义字符类：**\d**

含义：代表的是 0 到 9 数字，两头的数字包括在内，相当于**[0-9]**

例如：匹配规则为"**\d**"，那么需要匹配的是一个数字

### 预定义字符类：**\w**

含义：代表的字母或者数字或者下划线(即单词字符)，相当于**[a-zA-Z\_0-9]**

例如：匹配规则为"**\w**"，那么需要匹配的是一个字母或者是一个数字或一个下滑线

### 边界匹配器：**^**

含义：代表的是行的开头

例如：匹配规则为**^[abc][0-9]\$**，那么需要匹配的内容从[abc]这个位置开始，相当于左双引号

### 边界匹配器：**\$**

含义：代表的是行的结尾

例如：匹配规则为**^[abc][0-9]\$**，那么需要匹配的内容以[0-9]这个结束，相当于右双引号

### 边界匹配器：**\b**

含义：代表的是单词边界

例如：匹配规则为"**\b[abc]\b**"，那么代表的是字母 a 或 b 或 c 的左右两边需要的是非单词字符(**[a-zA-Z\_0-9]**)

### 数量词：**X?**

含义：代表的是 X 出现一次或一次也没有

例如：匹配规则为"**a?**"，那么需要匹配的内容是一个字符 a，或者一个 a 都没有

### 数量词：**X\***

含义：代表的是 X 出现零次或多次

例如：匹配规则为"**a\***"，那么需要匹配的内容是多个字符 a，或者一个 a 都没有

**数量词：X+**

含义：代表的是 X 出现一次或多次

例如：匹配规则为"**a+**"，那么需要匹配的内容是多个字符 a，或者一个 a

**数量词：X{n}**

含义：代表的是 X 出现恰好 n 次

例如：匹配规则为"**a{5}**"，那么需要匹配的内容是 5 个字符 a

**数量词：X{n,}**

含义：代表的是 X 出现至少 n 次

例如：匹配规则为"**a{5, }**"，那么需要匹配的内容是最少有 5 个字符 a

**数量词：X{n,m}**

含义：代表的是 X 出现至少 n 次，但是不超过 m 次

例如：匹配规则为"**a{5,8}**"，那么需要匹配的内容是有 5 个字符 a 到 8 个字符 a 之间

## 1.3 正则表达式规则匹配练习

请写出满足如下匹配规则的字符串：

规则："**[0-9]{6,12}**"

该规则需要匹配的内容是：长度为 6 位到 12 位的数字。

如：使用数据"123456789"进行匹配结果为 true；

使用数据"12345"进行匹配结果为 false。

规则："**1[34578][0-9]{9}**"

该规则需要匹配的内容是：**11 位的手机号码，第 1 位为 1，第 2 位为 3、4、5、7、8 中的一个，后面 9 位为 0 到 9 之间的任意数字。**

如：使用数据"12345678901"进行匹配结果为 false；  
使用数据"13312345678"进行匹配结果为 true。

规则："a\*b"

该规则需要匹配的内容是：**在多个 a 或零个 a 后面有个 b；b 必须为最后一个字符。**

如：使用数据"aaaaab"进行匹配结果为 true；  
使用数据"abc"进行匹配结果为 false。

## 1.4 字符串类中涉及正则表达式的常用方法

boolean	<code>matches(String regex)</code> 告知此字符串是否匹配给定的 <a href="#">正则表达式</a> 。
String[]	<code>split(String regex)</code> 根据给定 <a href="#">正则表达式</a> 的匹配拆分此字符串。
String	<code>replaceAll(String regex, String replacement)</code> 使用给定的 replacement 替换此字符串所有匹配给定的 <a href="#">正则表达式</a> 的子字符串。

- public boolean **matches**(String regex) //判断字符串是否匹配给定的规则

举例：校验 qq 号码。

- 1: 要求必须是 5-15 位数字
- 2: 0 不能开头

代码演示：

```
String qq = "604154942";  
String regex = "[1-9][0-9]{4,14}";  
boolean flag2 = qq.matches(regex);
```

举例：校验手机号码

- 1：要求为 11 位数字
- 2：第 1 位为 1，第 2 位为 3、4、5、7、8 中的一个，后面 9 位为 0 到 9 之间的任意数字。

代码演示：

```
String phone = "18800022116";  
String regex = "1[34578][0-9]{9}";  
boolean flag = phone.matches(regex);
```

- public String[] **split**(String regex) //根据给定正则表达式的匹配规则，拆分此字符串

举例：分割出字符串中的数字

代码演示：

```
String s = "18-22-40-65";
```

```
String regex = "-";  
String[] result = s.split(regex);
```

代码演示：

```
String s = "18 22 40 65";  
String regex = " ";  
String[] result = s.split(regex);
```

- `public String replaceAll(String regex,String replacement)` //将符合规则的字符串内容 ,全部替换为新字符串

举例：把文字中的数字替换成\*

代码演示：

```
String s = "Hello12345World6789012";  
String regex = "[0-9]";  
String result = s.replaceAll(regex, "*");
```

## 1.5 正则表达式练习

- 匹配正确的数字

匹配规则：

```
匹配正整数："\\d+"
```

```
匹配正小数："\\d+\\.\\d+"
```

```
匹配负整数："-\\d+"
```

```
匹配负小数："-\\d+\\.\\d+"
```

```
匹配保留两位小数的正数："\\d+\\.\\d{2}"
```

```
匹配保留 1-3 位小数的正数："\\d+\\.\\d{1,3}"
```

- 匹配合法的邮箱

匹配规则：

```
"[a-zA-Z_0-9]+@[a-zA-Z_0-9]+(\\.[a-zA-Z_0-9]+)+"
```

```
"\\w+@\\w+(\\.\\w+)+"
```

- 获取 IP 地址(192.168.1.100)中的每段数字

匹配规则：

```
"\\."
```

## 第 2 章 Date

### 2.1 Date 类概述

类 Date 表示特定的瞬间，精确到毫秒。

继续查阅 Date 类的描述，发现 Date 拥有多个构造函数，只是部分已经过时，但是其中有未过时的构造函数可以把毫秒值转成日期对象。

<code>Date()</code>	分配 Date 对象并初始化此对象，以表示分配它的时间（精确到毫秒）。
<code>Date(long date)</code>	分配 Date 对象并初始化此对象，以表示自从标准基准时间（称为“历元（epoch）”，即 1970 年 1 月 1 日 00:00:00 GMT）以来的指定毫秒数。

```
//创建日期对象，把当前的毫秒值转成日期对象
Date date = new Date(1607616000000L);
System.out.println(date);
//打印结果：Fri Dec 11 00:00:00 CST 2020
```

可是将毫秒值转成日期后，输出的格式不利于我们阅读，继续查阅 API，Date 中有 `getYear`、`getMonth` 等方法，可以他们已经过时，继续往下查阅，看到了 `toString` 方法。

<code>String</code>	<code>toString()</code> 把此 Date 对象转换为以下形式的 String: dow mon dd hh:mm:ss zzz yyyy 其中: dow 是一周中的某一天 (Sun, Mon, Tue, Wed, Thu, Fri, Sat)。
---------------------	--

点开 `toString()` 方法查阅，原来上面打印的 date 对象就是默认调用了这个 `toString` 方法，并且在这个方法下面还有让我们参见 `toLocaleString` 方法，点进去，这个方法又过时了，从 JDK 1.1 开始，由 `DateFormat.format(Date date)` 取代。

既然这个方法被 `DateFormat.format(Date date)` 取代，那么就要去查阅 `DateFormat` 类。

### 2.2 Date 类常用方法

<code>long</code>	<code>getTime()</code> 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。
-------------------	--

- 把日期对象转换成对应的时间毫秒值



## 第 3 章 DateFormat

### 3.1 DateFormat 类概述

DateFormat 是日期/时间格式化子类的抽象类，它以与语言无关的方式格式化并解析日期或时间。日期/时间格式化子类（如 **SimpleDateFormat** 类）允许进行格式化（也就是日期 -> 文本）、解析（文本-> 日期）和标准化。

我们通过这个类可以帮我们完成日期和文本之间的转换。

继续阅读 API，DateFormat 可帮助进行格式化并解析任何语言环境的日期。对于月、星期，甚至日历格式（阴历和阳历），其代码可完全与语言环境的约定无关。

### 3.2 日期格式

要格式化一个当前语言环境下的日期也就是日期 -> 文本），要通过下面的方法来完成。

DateFormat 是抽象类，我们需要使用其子类 SimpleDateFormat 来创建对象。

- 构造方法

<code>SimpleDateFormat(String pattern)</code> 用给定的模式和默认语言环境的日期格式符号构造 SimpleDateFormat。
---

- DateFormat 类方法

<code>String format(Date date)</code> 将一个 Date 格式化为日期/时间字符串。
---

代码演示：

//创建日期格式化对象,在获取格式化对象时可以指定风格

DateFormat df= new SimpleDateFormat("yyyy-MM-dd");//对日期进行格式化

Date date = new Date(1607616000000L);

String str\_time = df.format(date);

System.out.println(str\_time);//2020 年 12 月 11 日

- DateFormat 类的作用：即可以将一个 Date 对象转换为一个符合指定格式的字符串，也可以

将一个符合指定格式的字符串转为一个 Date 对象。

指定格式的具体规则我们可参照 SimpleDateFormat 类的说明，这里做简单介绍，规则是在一个字符串中，会将以下字母替换成对应时间组成部分，剩余内容原样输出：

- 当出现 y 时，会将 y 替换成年
- 当出现 M 时，会将 M 替换成月
- 当出现 d 时，会将 d 替换成日
- 当出现 H 时，会将 H 替换成时
- 当出现 m 时，会将 m 替换成分
- 当出现 s 时，会将 s 替换成秒

### 3.3 SimpleDateFormat 类常用方法

String	<code>format(Date date)</code> 将一个 Date 格式化为日期/时间字符串。
Date	<code>parse(String source)</code> 从给定字符串的开始解析文本，以生成一个日期。

- format 方法，用来将 Date 对象转换成 String
- parse 方法，用来将 String 转换成 Date(转换时，该 String 要符合指定格式，否则不能转换)。

代码演示：

练习一：把 Date 对象转换成 String

```
Date date = new Date(1607616000000L); // Fri Dec 11 00:00:00 CST 2020
DateFormat df = new SimpleDateFormat("yyyy 年 MM 月 dd 日");
String str = df.format(date);
//str 中的内容为 2020 年 12 月 11 日
```

练习二：把 String 转换成 Date 对象

```
String str = "2020 年 12 月 11 日";
DateFormat df = new SimpleDateFormat("yyyy 年 MM 月 dd 日");
Date date = df.parse(str);
//Date 对象中的内容为 Fri Dec 11 00:00:00 CST 2020
```

## 第 4 章 Calendar

### 4.1 Calendar 类概念

Calendar 是日历类，在 Date 后出现，替换掉了许多 Date 的方法。该类将所有可能用到的时间信息封装为静态成员变量，方便获取。

Calendar 为抽象类，由于语言敏感性，Calendar 类在创建对象时并非直接创建，而是通过静态方法创建，将语言敏感内容处理好，再返回子类对象，如下：

- Calendar 类静态方法

static Calendar	<code>getInstance()</code> 使用默认时区和语言环境获得一个日历。
-----------------	--

```
Calendar c = Calendar.getInstance(); //返回当前时间
```

### 4.2 Calendar 类常用方法

abstract void	<code>add(int field, int amount)</code> 根据日历的规则，为给定的日历字段添加或减去指定的时间量。
int	<code>get(int field)</code> 返回给定日历字段的值。
static Calendar	<code>getInstance()</code> 使用默认时区和语言环境获得一个日历。
Date	<code>getTime()</code> 返回一个表示此 Calendar 时间值（从 <a href="#">历元</a> 至现在的毫秒偏移量）的 Date 对象。
void	<code>set(int field, int value)</code> 将给定的日历字段设置为给定值。

- public static Calendar `getInstance()` //获取日期对象
- public int `get(int field)` //获取时间字段值，字段参见帮助文档
  - YEAR 年
  - MONTH 月，从 0 开始算起，最大 11；0 代表 1 月，11 代表 12 月。

- DATE 天
- HOUR 时
- MINUTE 分
- SECOND 秒

代码演示：

```
Calendar c = Calendar.getInstance();  
int year = c.get(Calendar.YEAR);
```

- public void **add**(int field,int amount) //指定字段增加某值

代码演示：

```
Calendar c = Calendar.getInstance();  
//修改当前时间为 3 天后  
c.add(Calendar.DATE, 3);  
//修改当前时间为 5 小时后  
c.add(Calendar.HOUR, 5);
```

- public final void **set**(int field,int value)//设置指定字段的值

代码演示：

```
Calendar c = Calendar.getInstance();  
//设置时间为 2020 年 5 月 20 日  
c.set(Calendar.YEAR, 2020);  
c.set(Calendar.MONTH, 4);  
c.set(Calendar.DATE, 20);
```

- public final Date **getTime**() //获取该日历对象转成的日期对象

代码演示：

```
Calendar c = Calendar.getInstance();  
Date d = c.getTime();
```

## 4.3 注意事项

西方星期的开始为周日，中国为周一。

在 Calendar 类中，月份的表示是以 0-11 代表 1-12 月。

日期是有大小关系的，时间靠后，时间越大。

## 第 5 章 日期相关类练习

### 5.1 求出自己已经出生多少天

思路：

1. 获取当前时间对应的天数
2. 获取自己出生日期对应的天数
3. 两个时间相减（当前时间天数 - 出生日期天数）

代码实现：

```
public static void main(String[] args){
    Calendar my = Calendar.getInstance();
    Calendar c = Calendar.getInstance();
    //设置出生年月日 1995-05-10
    my.set(Calendar.YEAR, 1995);
    my.set(Calendar.MONTH, 4);
    my.set(Calendar.DATE, 10);
    //获取时间中的天数
    int day = c.get(Calendar.DATE);
    int myDay = my.get(Calendar.DATE);
    System.out.println(day - myDay);
}
```

### 5.2 求出今天距离 2020 年 1 月 1 日还有多少天

思路：

1. 获取当前时间对应的天数
2. 获取 2020 年 1 月 1 日对应的天数
3. 两个时间相减（2020 年 1 月 1 日的天数 - 当前时间天数）

代码实现：

```
public static void main(String[] args){
    Calendar my = Calendar.getInstance();
```

```
Calendar c = Calendar.getInstance();
//设置年月日 2020-01-01
my.set(Calendar.YEAR, 2020);
my.set(Calendar.MONTH, 0);
my.set(Calendar.DATE, 1);
//获取时间中的天数
int day = c.get(Calendar.DATE);
int myDay = my.get(Calendar.DATE);
System.out.println(myDay - day);
}
```

## 第 6 章 总结

### 6.1 知识点总结

- 正则表达式：用来定义匹配规则，匹配一系列符合某个句法规则的字符串。

正则表达式的匹配规则

请参见 1.2 正则表达式的匹配规则

正则表达式的常用方法：

`public boolean matches(String regex)` //判断字符串是否匹配给定的规则

`public String[] split(String regex)` //根据给定正则表达式的匹配规则，拆分此字符串

`public String replaceAll(String regex,String replacement)` //将符合规则的字符串内容 ,全部替换为新字符串

- Date: 日期/时间类

构造方法：

`public Date()` // 系统当前日期时间

`public Date(long date)` 得到一个 1970 年 1 月 1 日 0 点这个时间基础上，加上参数 `date` 毫秒值对应的日期时间

方法：

`public long getTime()` 获取日期所对应的毫秒值

- `DateFormat`:是日期/时间格式化子类的抽象类，使用其子类 `SimpleDateFormat` 实例化

构造方法：

`public SimpleDateFormat()` 默认的格式化操作

`public SimpleDateFormat(String pattern)` 按照指定的格式，进行日期格式化

日期和时间模式

y 年

M 年中的月份

d 月份中的天数

H 一天中的小时数（0-23）

m 小时中的分钟数

s 分钟中的秒数

S 毫秒数

方法：

`public final String format(Date date)` 把日期 格式化成字符串

`public Date parse(String source)` 把日期字符串 转换成 日期对象

- `Calendar`:日历类，可获取日期中指定字段的值

方法：

```
public static Calendar getInstance() //获取日期对象
```

```
public int get(int field) //获取时间字段值
```

```
public void add(int field,int amount)    //指定字段增加某值
```

```
public final void set(int field,int value)//设置指定字段的值
```

```
public final Date getTime()//获取该日历对象转成的日期对象
```