# 第6天常用API

今日内容介绍

* 正则表达式
* Date
* DateFormat
* Calendar
* 基本类型包装类
* System
* Math

今日学习目标

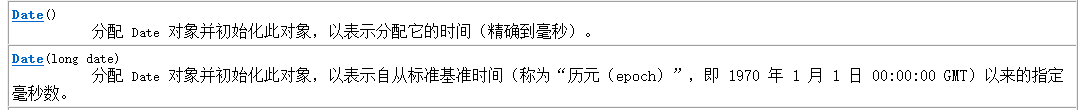
* 能够理解正则表达式验证11位手机号码
* 能够理解正则表达式验证QQ号码
* 能够使用日期类输出当前的日期
* 能够说出将日期格式化成字符串的方法
* 说出将字符串转换成日期的方法
* 写出基本数据类型对应的八种包装类
* 写出字符串转换成基本数据类型的方法
* 写出基本数据类型转换成字符串方式
* 能够说出拆箱装箱概念
* 能够掌握System类常见方法的使用
* 能够使用Math类进行数学运算

# Date

## 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。



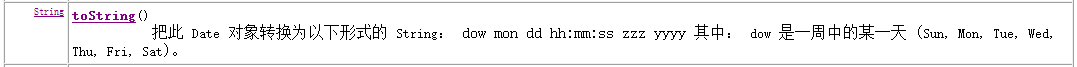
//创建日期对象，把当前的毫秒值转成日期对象

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、getMouth等方法，可以他们已经过时，继续往下查阅，看到了toString方法。



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.

## Date类常用方法



* 把日期对象转换成对应的时间毫秒值
* 毫秒值是不断变化的,所以每次打印的不一样,金钱买不了时间

# DateFormat

## 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。

## 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



* DateFormat类方法



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);

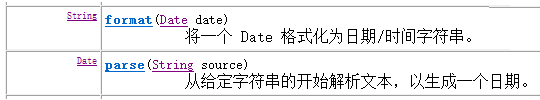
System.*out*.println(str\_time);//2020年12月11日

* 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

## DateFormat类常用方法



* 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

# Calendar

## 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



Calendar c = Calendar.getInstance(); //返回当前日历对象

Syso(c);//我们可以打印出来 看一下,东西很多不必要挨个研究

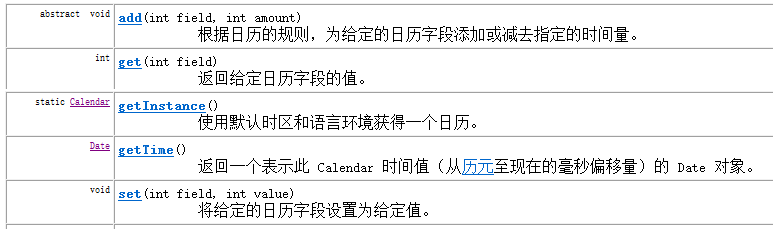
public static Calendar getInstance(){

Calendar c = new GregorianCalendar();

return c;

}

## Calendar类常用方法



* 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);

//做一个小练习 打印当前时间:xxxx年xx月xx日(注意月份的时间???)

* 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();

## 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。

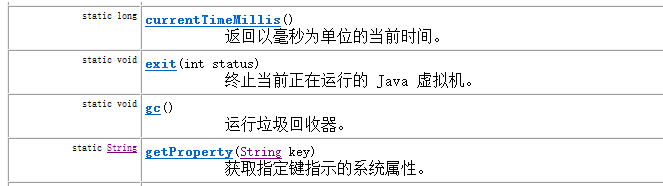
# System类

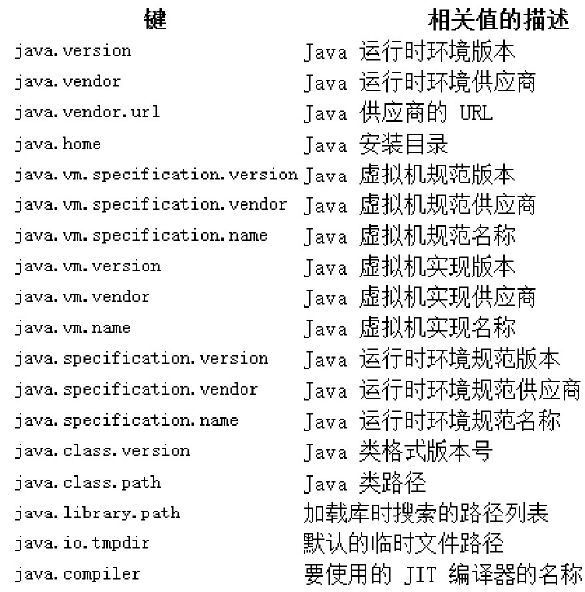
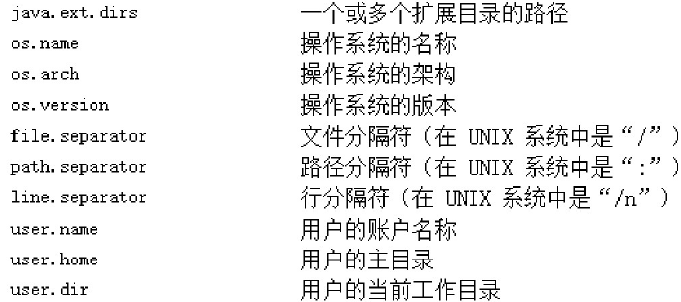
## 概念

在API中System类介绍的比较简单，我们给出定义，System中代表程序所在系统，提供了对应的一些系统属性信息，和系统操作。

查看文档System类不能手动创建对象，因为构造方法被private修饰，阻止外界创建对象。System类中的都是static方法，类名访问即可。在JDK中，有许多这样的类。

## 常用方法(arraycopy)



* **currentTimeMillis**() 获取当前系统时间与1970年01月01日00:00点之间的毫秒差值
* **exit(int status)** 用来结束正在运行的Java程序。参数传入一个数字即可。通常传入0记为正常状态，其他为异常状态
* **gc()** 用来运行JVM中的垃圾回收器，完成内存中垃圾的清除。
* **getProperty(String key)** 用来获取指定**键**(字符串名称)中所记录的系统属性信息
* 
* 

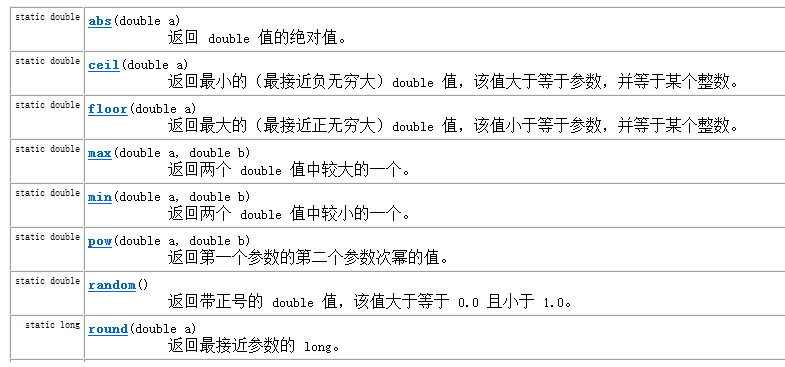
# Math类

## 概念

Math 类是包含用于执行基本数学运算的方法的数学工具类，如初等指数、对数、平方根和三角函数。

类似这样的工具类，其所有方法均为静态方法，并且一般不会创建对象。如System类

## 常用方法



* abs方法,结果都为正数

double d1 = Math.abs(-5); // d1的值为5

double d2 = Math.abs(5); // d2的值为5

* ceil方法，结果为比参数值大的最小整数的double值

double d1 = Math.ceil(3.3); //d1的值为 4.0

double d2 = Math.ceil(-3.3); //d2的值为 -3.0

double d3 = Math.ceil(5.1); // d3的值为 6.0

* floor方法，结果为比参数值小的最大整数的double值

double d1 = Math.floor(3.3); //d1的值为3.0

double d2 = Math.floor(-3.3); //d2的值为-4.0

double d3 = Math.floor(5.1); //d3的值为 5.0

* max方法，返回两个参数值中较大的值

double d1 = Math.max(3.3, 5.5); //d1的值为5.5

double d2 = Math.max(-3.3, -5.5); //d2的值为-3.3

* min方法，返回两个参数值中较小的值

double d1 = Math.min(3.3, 5.5); //d1的值为3.3

double d2 = Math.max(-3.3, -5.5); //d2的值为-5.5

* pow方法，返回第一个参数的第二个参数次幂的值

double d1 = Math.pow(2.0, 3.0); //d1的值为 8.0

double d2 = Math.pow(3.0, 3.0); //d2的值为27.0

* round方法，返回参数值四舍五入的结果

double d1 = Math.round(5.5); //d1的值为6.0

double d2 = Math.round(5.4); //d2的值为5.0

* random方法，产生一个大于等于0.0且小于1.0的double小数

double d1 = Math.random();

# 基本类型包装类

大家回想下，在第二天我们学习Java中的基本数据类型时，说Java中有8种基本的数据类型，可是这些数据是基本数据，想对其进行复杂操作，变的很难。怎么办呢？

## 基本类型包装类概述

在实际程序使用中，程序界面上用户输入的数据都是以字符串类型进行存储的。而程序开发中，我们需要把字符串数据，根据需求转换成指定的基本数据类型，如年龄需要转换成int类型，考试成绩需要转换成double类型等。那么，想实现字符串与基本数据之间转换怎么办呢？

Java中提供了相应的对象来解决该问题，基本数据类型对象包装类：java将基本数据类型值封装成了对象。封装成对象有什么好处？可以提供更多的操作基本数值的功能。

8种基本类型对应的包装类如下：

其中需要注意int对应的是Integer，char对应的Character，其他6个都是基本类型首字母大写即可。

基本数据类型对象包装类特点：用于在基本数据和字符串之间进行转换。

* 将字符串转成基本类型：



parseXXX(String s);其中XXX表示基本类型，参数为可以转成基本类型的字符串，如果字符串无法转成基本类型，将会发生数字转换的问题[NumberFormatException](mk://../../java/lang/NumberFormatException.html)

System.out.println(Integer.parseInt("123") + 2);

//打印结果为 125

基本类型变成String

基本类型+”” 就可以

## 自动装箱拆箱

在需要的情况下，基本类型与包装类型可以通用。有些时候我们必须使用引用数据类型时，可以传入基本数据类型。

比如：

基本类型可以使用运算符直接进行计算，但是引用类型不可以。而基本类型包装类作为引用类型的一种却可以计算，原因在于，Java”偷偷地”自动地进行了对象向基本数据类型的转换。

相对应的，引用数据类型变量的值必须是new出来的内存空间地址值，而我们可以将一个基本类型的值赋值给一个基本类型包装类的引用。原因同样在于Java又”偷偷地”自动地进行了基本数据类型向对象的转换。

* 自动拆箱：对象转成基本数值
* 自动装箱：基本数值转成对象

Integer i = 4;//自动装箱。相当于Integer i = Integer.valueOf(4);

i = i + 5;//等号右边：将i对象转成基本数值(自动拆箱) i.intValue() + 5; 加法运算完成后，再次装箱，把基本数值转成对象。

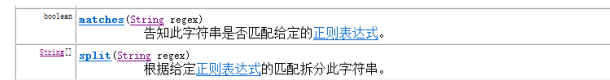
# 正则表达式

## 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。

## 字符串类中涉及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

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：分割出电话号码字符串中的的数字

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);

## 附录(这部分内容大家知道有就可以了):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滑线

**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

s

**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]**)

String regex = ".\*\\b[abc]\\b.\*";

String str = "abc a bbc";

**boolean**flag = str.matches(regex);

**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

**逻辑运算符：XY**

含义：代表的是X后跟Y

例如：匹配规则为**"ab"**，那么需要匹配的字符串内容就是 ”ab”

**逻辑运算符：X|Y**

含义：代表的是*X* 或 *Y*

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

**逻辑运算符：**(*X*)

含义：代表的是（）括号内的数据作为一组数据出现，(*X*)的方式称为正则表达式中的组

例如：匹配规则为**"(hello)+"**，那么需要匹配的内容是组内数据要出现多次，如”hellohellohello”；或者组内数据出现一次，如”hello”。

* 注意实现：
  + 在定义正则表达式的匹配规则字符串的里面，想再次使用组中的内容，可通过\\1来进行使用

例如：正则表达式的匹配规则为**"(a) == \\1"**；

使用数据"a == a"进行匹配结果为true；使用数据"a == b"进行匹配结果为false。