Java API

一、包

```
package r;

import java.util.Random;

import e.demo3;



public class test1 {
    public static void main(String[] args) {
        //1.用一个包可直接访问
        demo2 d=new demo2();
        d.print();
        //2.访问其它包下的程序，必须导包
        demo3 d3=new demo3();
        d3.print();
//3.除了Java.lang,其它用Java的包都要导包
        Random r=new Random();
//4.访问多个同名的包，必须写明路径
        t.demo3 d4=new t.demo3();
        d4.print();
    }
}
```

二、String类

和长度有关的方法

返回类型　方法名　　作用

int　length()　得到一个字符串的字符个数（一个中文是一个字符，一个英文是一个字符，一个转义字符是一个字符）

和数组有关的方法

返回类型　方法名　　作用

byte[]　getBytes()　将一个字符串转换成字节数组

char[]　toCharArray()　将一个字符串转换成字符数组

String[]　split(String)　将一个字符串按照指定内容劈开

和判断有关的方法

返回类型　方法名　　作用

boolean　equals(String)　　判断两个字符串的内容是否一模一样

boolean　equalsIgnoreCase(String)　忽略大小写的比较两个字符串的内容是否一模一样

boolean　contains(String)　　判断一个字符串里面是否包含指定的内容

boolean　startsWith(String)　　判断一个字符串是否以指定的内容开头

boolean　endsWith(String)　　判断一个字符串是否以指定的内容结尾

和改变内容有关的方法

和改变内容有关的方法，都不会直接操作原本的字符串

而是将符合条件的字符串返回给我们，所以注意接收

返回类型	方法名	作用
String	toUpperCase()	将一个字符串全部转换成大写
String	toLowerCase()	将一个字符串全部转换成小写
String	replace(String,String)	将某个内容全部替换成指定内容
String	replaceAll(String,String)	将某个内容全部替换成指定内容，支持正则
String	repalceFirst(String,String)	将第一次出现的某个内容替换成指定的内容
String	substring(int)	从指定下标开始一直截取到字符串的最后
String	substring(int,int)	从下标x截取到下标y-1对应的元素
String	trim()	去除一个字符串的前后空格

和位置有关的方法

返回类型	方法名	作用
char	charAt(int)	得到指定下标位置对应的字符
int	indexOf(String)	得到指定内容第一次出现的下标
int	lastIndexOf(String)	得到指定内容最后一次出现的下标

三、StringBuffer

要先创建StringBuffer缓冲区

·String是不可改变的字符序列

·StringBuffer是一个可改变的字符序列

使用StringBuffer类的构造方法初始化字符串对象

```
StringBuffer 变量名=new String(字符串);
```

借助String类来创建StringBuffer类对象

StringBuffer s="abc";(错误方法)

String 类和 StringBuffer 类是两个不相同的类，这里的，但是我们可以采用下面的方法进行转换：

String s="abc";
StringBuffer s5=new StringBuffer(s);

方法声明	功能描述
StringBuffer append(char c)	添加字符到StringBuffer对象中末尾
StringBuffer insert(int offset,String str)	在StringBuffer对象中的offset位置插入字符串str
StringBuffer deleteCharAt(int index)	移除StringBuffer对象中指定位置的字符
StringBuffer delete(int start,int end)	删除StringBuffer对象中指定范围的字符或字符串
StringBuffer replace(int start,int end,String s)	将StringBuffer对象中指定范围的字符或字符串用新的字符串s进行替换
void setCharAt(int index, char ch)	修改指定位置index处的字符
String toString()	返回StringBuffer缓冲区中的字符串对象
StringBuffer reverse()	将此StringBuffer对象用其反转形式取代

四、System类

类别　名称　描述
方法　arraycopy(Object src, int srcPos, Object dest, int destPos, int length)　复制数组中的元素到另一个数组
currentTimeMillis()　返回当前时间的毫秒数
exit(int status)　退出Java虚拟机，可使用指定的状态码作为程序的结束代码
gc()　强制运行垃圾回收器
getenv(String name)　获取指定环境变量的值
getProperties() 获取当前系统属性的集合。 如获取操作系统版本、文件分隔符等
getProperty(String key)　根据键名获取系统属性的值
getProperty(String key, String defaultValue)　根据键名获取系统属性的值，如果没有找到则返回默认值

五、Runtime类

要先创建Runtime对象

public static Runtime getRuntime()　返回单例的Runtime实例
public void exit(int status)　终止当前的虚拟机
public void addShutdownHook(Thread hook)　增加一个JVM关闭后的钩子
public Process exec(String command)throws IOException　执行command指令，启动一个新的进程
public int availableProcessors()获得JVM可用的处理器数量（一般为CPU核心数）
public long freeMemory()　获得JVM已经从系统中获取到的总共的内存数【byte】
public long totalMemory()　获得JVM中剩余的内存数【byte】
public long maxMemory()　获得JVM中可以从系统中获取的最大的内存数【byte】

六、Math类

1. Math.abs(a)　取a的绝对值

2. Math.sqrt(a) 取a的平方根
3. Math.cbrt(a) 取a的立方根
4. Math.max(a,b) 取a、b之间的最大值
5. Math.min(a,b) 取a、b之间的最小值
6. Math.pow(a,b) 取a的b平方
7. Math.ceil(a) 向上取整
8. Math.floor(a) 向下取整
9. Math.round(a) 四舍五入
10. Math.PI 取圆周率

七、Ramdon类

要先创建random对象

1.Random() 使用当前系统时间创建一个Ramdom对象

2.Random(long seed) 使用参数seed指定的种子创建一个Random对象

`nextInt(100)` 是 `Random` 类的一个方法，用于生成一个随机整数，该整数的取值范围是 0 到指定的参数值减 1（即 `0` 到 `n - 1`）。

八、bigInteger类

它是处理大数的类

要先创建bigInteger对象

```
public void testBasic() {
        BigInteger a = new BigInteger("13");
        BigInteger b = new BigInteger("4");
        int n = 3;

        //1.加(a与b)
        BigInteger bigNum1 = a.add(b);          //17
        //2.减
        BigInteger bigNum2 = a.subtract(b);     //9
        //3.乘
        BigInteger bigNum3 = a.multiply(b);     //52
        //4.除
        BigInteger bigNum4 = a.divide(b);       //3
        //5.取余，第一个返回数值为商，第二个返回为余数
        BigInteger[] intgers = a.divideAndRemainder(b);

    }
```

九、BigDecimal类

解决小数运算失真的问题

要先创建BigDecimal对象

BigDecimal(double val)	**创建一个具有参数所指定双精度值的对象。** 不推荐使用，因为存在精度丢失问题
BigDecimal(String val)	创建一个具有参数所指定以字符串表示的数值的对象。 推荐使用
add(BigDecimal)	BigDecimal对象中的值相加，返回BigDecimal对象
subtract(BigDecimal)	BigDecimal对象中的值相减，返回BigDecimal对象
multiply(BigDecimal)	BigDecimal对象中的值相乘，返回BigDecimal对象
divide(BigDecimal)	BigDecimal对象中的值相除，返回BigDecimal对象。
max(BigDecimal val)	两值比较，返回最大值
min(BigDecimal val)	两值比较，返回最小值

```
import java.math.BigDecimal;

public class Test {
    public static void main(String[] args){
        BigDecimal b1 = new BigDecimal("1");
        BigDecimal b2 = new BigDecimal("2");
        BigDecimal b3 = new BigDecimal("4");
        System.out.println("相加："+b1.add(b2));
        System.out.println("相减："+b1.subtract(b2));
        System.out.println("相乘："+b2.multiply(b3));
        System.out.println("相除："+b2.divide(b3));
    }
}
```

十、Data类

Date date1=new Date();//用于创建当前日期和时间对象

Date date2=new Date(long date);//创建指定的日期和时间对象

十一、Calender类

Calender是抽象类

要先创建Calender类对象

```
Calendar calendar = Calendar.getInstance();
int year = calendar.get(Calendar.YEAR);
int month = calendar.get(Calendar.MONTH);
int day = calendar.get(Calendar.DAY_OF_MONTH);
int hour = calendar.get(Calendar.HOUR_OF_DAY);
int minute = calendar.get(Calendar.MINUTE);
int second = calendar.get(Calendar.SECOND);

System.out.println("Current Date: " + year + "-" + (month + 1) + "-" + day);
System.out.println("Current Time: " + hour + ":" + minute + ":" + second);
```

set(field, value)：设置指定字段的值。例如，设置月份为5（六月）：

```
Calendar calendar = Calendar.getInstance();
calendar.set(Calendar.MONTH, 5);
```

add(field, amount)：在指定字段上添加或减去指定的数量。例如，增加一天：

```
Calendar calendar = Calendar.getInstance();
calendar.add(Calendar.DAY_OF_MONTH, 1);
```

十二、Instant 类

可以精确到纳秒数

要先创建Instant类对象

方法声明 功能描述
now（） 从系统时钟获取当前时刻
now(Clock clock) 从指定时钟获取当前时刻
ofEpochSecond(long epochSecond) 使用自标准 Java 历元开始的秒数获取 Istant 类的实例
ofEpochMilli(long epochMilli) 使用自标准 Java 历元开始的毫秒数获取 Instant 类的实例
getEpochSecond() 根据标准 Java 历元获取秒数
getNano() 获取 Instant 实例时间到当前时间的纳秒数
parse(CharSequence text) 从一个时间文本字符串(如 2023-04-25T16:18:33.44Z)获取 Instant的实例
from(TemporalAccessor tenporal) 从时间对象获取 Instant 类的实例

EXP:

```
import java.time.Instant;

public class InstantMain {
    public static void main(String[] args) {
        Instant now = Instant.now();
        System.out.println("从系统获取的当前时刻为："+now);
        Instant instant1= Instant.ofEpochMilli(0);
        System.out.println("计算机元年为："+instant1);
        Instant instant2= Instant.ofEpochSecond(60*60*24);
        System.out.println("计算机元年增加60*60*24秒后为："+instant2);
```

```
        Instant instant3= Instant.ofEpochMilli(1000*60*60*24);
        System.out.println("计算机元年增加1000*60*60*24毫秒后为："+instant3);
        System.out.println("获取秒值为："+Instant.parse("2023-04-
25T16:18:33.44Z").getEpochSecond());
        System.out.println("获取纳秒值为："+Instant.parse("2023-04-
25T16:18:33.44Z").getNano());
        System.out.println("从时间对象获取的Instant实例为："+Instant.from(now));
    }
}
```

date是可变对象，而instant类是不可变对象

十三、LocalDate类

获取本地日期对象(不可变对象)

public static Xxxx now();获取系统当前时间对应对象

1、LocalDate

年月日

```
//1）创建方法
LocalDate now1 = LocalDate.now();                           // 当前日期
LocalDate now2 = LocalDate.now(ZoneId.of("Asia/Shanghai")); // 当前日期（指定时区）
LocalDate now3 = LocalDate.now(Clock.systemDefaultZone());  // 当前日期（指定时钟）
LocalDate localDate = LocalDate.of(2023, 1, 1);             // 指定日期 2023-01-01

//2）获取方法
LocalDate now = LocalDate.now();
int year = now.getYear();                       // 获取年份
int month = now.getMonthValue();                // 获取月份（1-12）
Month monthEnum = now.getMonth();               // 获取月份的枚举值
int dayOfMonth = now.getDayOfMonth();           // 获取月份中的第几天（1-31）
int dayOfYear = now.getDayOfYear();             // 获取一年中的第几天（1-366）
DayOfWeek dayOfWeek = now.getDayOfWeek();       // 获取现在是星期几
int dayOfYear = now.getDayOfYear();             // 获取一年中的第几天（1-366）
int lengthOfYear = now.lengthOfYear();          // 获得当年总天数
int lengthOfMonth = now.lengthOfMonth();        // 获得当月总天数
long epochDay = now.toEpochDay();               // 与时间纪元（1970年1月1日）相差的天
数

//3）运算方法
LocalDate now = LocalDate.now();
LocalDate localDate1 = now.plusDays(1);         // 给当前时间加一天
LocalDate localDate2 = now.plusDays(1);         // 给当前时间加一周
LocalDate localDate3 = now.plusMonths(1);       // 给当前时间加一月
LocalDate localDate4 = now.plusYears(1);        // 给当前时间加一年
LocalDate localDate5 = now.minusDays(1);        // 给当前时间减一天
LocalDate localDate6 = now.minusWeeks(1);       // 给当前时间减一周
LocalDate localDate7 = now.minusMonths(1);      // 给当前时间减一月
LocalDate localDate8 = now.minusYears(1);       // 给当前时间减一年
```

```
//4）修改方法
LocalDate now = LocalDate.now();
LocalDate localDate1 = now.withYear(2020);      // 修改日期对象年份为2020
LocalDate localDate2 = now.withMonth(1);        // 修改日期对象月份为1
LocalDate localDate3 = now.withDayOfMonth(1);   // 修改日期对象的日期(一月中的第几天)
LocalDate localDate4 = now.withDayOfYear(1);    // 修改日期对象的日期(一年中的第几天)

//5）比较方法
LocalDate now = LocalDate.now();
LocalDate localDate = LocalDate.of(2023, 1, 1)

boolean isBefore = localDate.isBefore(now);     // localDate是否在当天之前
boolean isAfter = localDate.isAfter(now);       // localDate是否在当天之后
boolean isEqual = localDate.isEqual(now);       // localDate是否在当天
boolean isLeapYear = localDate.isLeapYear();    // localDate是否是闰年
```

2、Localtime

时分秒纳秒

```
//1）创建方法
LocalTime now1 = LocalTime.now();                               // 当前时间
LocalTime now2 = LocalTime.now(ZoneId.of("Asia/Shanghai")); // 当前时间 (指定时区)
LocalTime now3 = LocalTime.now(Clock.systemDefaultZone());  // 当前时间 (指定时钟)
LocalTime localTime = LocalTime.of(9, 9, 9);                    // 指定时间 09:09:09

//2）获取方法
LocalTime now = LocalTime.now();
int hour = now.getHour();                       // 获取小时
int minute = now.getMinute();                   // 获取小时
int second = now.getSecond();                   // 获取秒
int nano = now.getNano();                       // 获取纳秒

//3）运算方法
LocalTime now = LocalTime.now();
LocalTime localTime1 = now.plusHours(1);        // 给当前时间加一小时
LocalTime localTime2 = now.plusMinutes(1);      // 给当前时间加一分钟
LocalTime localTime3 = now.plusSeconds(1);      // 给当前时间加一秒
LocalTime localTime4 = now.plusNanos(1);        // 给当前时间加一纳秒
LocalTime localTime1 = now.minusHours(1);       // 给当前时间减一小时
LocalTime localTime2 = now.minusMinutes(1);     // 给当前时间减一分钟
LocalTime localTime3 = now.minusSeconds(1);     // 给当前时间减一秒
LocalTime localTime4 = now.minusNanos(1);       // 给当前时间减一纳秒

//4）修改方法
LocalTime now = LocalTime.now();
LocalDate localTime1 = now.withHour(1);         // 修改时间对象小时为1
LocalDate localTime2 = now.withMinute(1);       // 修改时间对象分钟为1
LocalDate localTime3 = now.withSecond(1);       // 修改时间对象秒钟为1
LocalDate localTime4 = now.withNano(1);         // 修改时间对象纳秒为1
```

```
//5）比较方法
LocalTime now = LocalTime.now();
LocalTime localTime = LocalTime.of(9, 9, 9);
int compare = localTime.compareTo(now);      // localTime和当前时间比较--0(相等)正数(大)负数(小)
boolean after = localTime.isAfter(now);      // localTime是否在当前时间之后
boolean before = localTime.isBefore(now);    // localTime是否在当前时间之前
```

3、LocalDateTime

```
//1）创建方法
// 1.当前日期
LocalDateTime now1 = LocalDateTime.now();
// 2.当前日期 (指定时区)
LocalDateTime now2 = LocalDateTime.now(ZoneId.of("Asia/Shanghai"));
// 3.当前日期 (指定时钟)
LocalDateTime now3 = LocalDateTime.now(Clock.systemDefaultZone());
// 4.指定日期 2023-01-01 01:01:00
LocalDateTime localDateTime1 = LocalDateTime.of(2023, 1, 1, 1, 1);
// 4.使用LocalDate和LocalTime对象创建LocalDateTime对象
LocalDate localDate = LocalDate.now();
LocalTime localTime = LocalTime.now();
LocalDateTime localDateTime2 = LocalDateTime.of(localDate, localTime);

//2）获取方法
LocalDateTime now = LocalDateTime.now();
int year = now.getYear();                          // 获取年份
Month month = now.getMonth();                      // 使用月份枚举值
int dayOfMonth = now.getDayOfMonth();              // 获取日期在该月是第几天
DayOfWeek dayOfWeek = now.getDayOfWeek();          // 获取日期是星期几
int dayOfYear = now.getDayOfYear();                // 获取日期在该年是第几天
int hour = now.getHour();                          // 获取小时
int minute = now.getMinute();                      // 获取分钟
int second = now.getSecond();                      // 获取秒钟
int nano = now.getNano();                          // 获取纳秒

//3）运算方法
LocalDateTime now = LocalDateTime.now();
LocalDateTime localDateTime1 = now.plusDays(1);          // 给当前时间加一天
LocalDateTime localDateTime2 = now.plusDays(1);          // 给当前时间加一周
LocalDateTime localDateTime3 = now.plusMonths(1);        // 给当前时间加一月
LocalDateTime localDateTime4 = now.plusYears(1);         // 给当前时间加一年
LocalDateTime localDateTime5 = now.plusHours(1);         // 给当前时间加一小时
LocalDateTime localDateTime6 = now.plusMinutes(1);       // 给当前时间加一分钟
LocalDateTime localDateTime7 = now.plusSeconds(1);       // 给当前时间加一秒
LocalDateTime localDateTime8 = now.plusNanos(1);         // 给当前时间加一纳秒
LocalDateTime localDateTime9 = now.minusDays(1);         // 给当前时间减一天
LocalDateTime localDateTime10 = now.minusWeeks(1);       // 给当前时间减一周
LocalDateTime localDateTime11 = now.minusMonths(1);      // 给当前时间减一月
LocalDateTime localDateTime12 = now.minusYears(1);       // 给当前时间减一年
LocalDateTime localDateTime13 = now.minusHours(1);       // 给当前时间减一小时
LocalDateTime localDateTime14 = now.minusMinutes(1);     // 给当前时间减一分钟
```

```
LocalDateTime localDateTime15 = now.minusSeconds(1);     // 给当前时间减一秒
LocalDateTime localDateTime16 = now.minusNanos(1);       // 给当前时间减一纳秒


//4）修改方法
LocalDateTime now = LocalDateTime.now();
LocalDateTime localDate1 = now.withYear(2020);           // 修改日期对象年份为2020
LocalDateTime localDate2 = now.withMonth(1);             // 修改日期对象月份为1
LocalDateTime localDate3 = now.withDayOfMonth(1);        // 修改日期对象的日期(一月中
的第几天)
LocalDateTime localDate4 = now.withDayOfYear(1);         // 修改日期对象的日期(一年中
的第几天)
LocalDateTime localTime1 = now.withHour(1);              // 修改时间对象小时为1
LocalDateTime localTime2 = now.withMinute(1);            // 修改时间对象分钟为1
LocalDateTime localTime3 = now.withSecond(1);            // 修改时间对象秒钟为1
LocalDateTime localTime4 = now.withNano(1);              // 修改时间对象纳秒为1

//5）比较方法
LocalDateTime now = LocalDateTime.now();
LocalDateTime localDateTime = LocalDateTime.of(2023, 1, 1, 1, 1);
boolean isBefore = localDateTime.isBefore(now);          // localDateTime是否在当天
之前
boolean isAfter = localDateTime.isAfter(now);            // localDateTime是否在当天
之后
boolean isEqual = localDateTime.isEqual(now);            // localDateTime是否在当天

//6）从LocalDateTime获取LocalDate或者LocalTime
LocalDateTime localDateTime = LocalDateTime.now();
LocalDate localDate = localDateTime.toLocalDate();    // 从LocalDateTime获取
LocalDate对象
LocalTime localTime = localDateTime.toLocalTime();    // 从LocalDateTime获取
LocalTime对象
```

十四、Period类

Period 类表示一段时间的年、月、日，开使用between()方法获取两个日期之间的差作为Period 对象返回：

```
LocalDate startDate = LocalDate.of(2015, 2, 20);
LocalDate endDate = LocalDate.of(2017, 1, 15);

Period period = Period.between(startDate, endDate);
```

十五、Duration类

Duration类表示秒或纳秒时间间隔，适合处理较短的时间，需要更高的精确性。我们能使用between()方法比较两个瞬间的差：

```
Instant start = Instant.parse("2017-10-03T10:15:30.00Z");
Instant end = Instant.parse("2017-10-03T10:16:30.00Z");

Duration duration = Duration.between(start, end);
```

```
String timeString = duration.toString(); //此持续时间的字符串表示形式,使用基于ISO-8601秒*的表示形式,例如 PT8H6M12.345S
System.out.println("相差的天数="+duration.toDays());
System.out.println("相差的小时="+ duration.toHours());
System.out.println("相差的分钟="+duration.toMinutes());
System.out.println("相差的秒数="+duration.getSeconds());
System.out.println("相差的毫秒="+duration.toMillis());
System.out.println("相差的纳秒="+duration.toNanos());
System.out.println("timeString时间="+timeString);
```

十六、DateFormat类

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）允许进行格式化（也就是日期→文本）、解析（文本→日期）和标准化日期。

DateFormat是抽象类

在创建 DateFormat 对象时不能使用 new 关键字，而应该使用 DateFormat 类中的静态方法 getDateInstance()，示例代码如下：

```
DateFormat df = DateFormat.getDatelnstance();
```

从SHORT到MEDIUM到LONG到FULL到DEFAULT。确切的结果取决于区域设置，但通常：

SHORT:	2019/1/28	下午5:23
MEDIUM:	2019年1月28日	下午5:23:12
LONG:	2019年1月28日	CST 下午5:23:12
FULL:	2019年1月28日星期一	中国标准时间 下午5:23:12
DEFAULT:	2019年1月28日	下午5:23:12

常用方法：

（1）getInstance()方法，使用SHORT样式的日期和时间获取格式化的日期和时间程序，输出样式：2019/1/28 下午5:23

（2）getDateInstance()方法，获取具有默认语言环境的默认格式样式的日期格式化程序，输出样式：2019/1/28

getDateInstance(int style)方法　//style为下面的日期长度参数，如SHORT

getDateInstance(int style,Locale alocale)方法　//alocale为语言环境，如CHINA

（3）getDateTimeInstance()方法，获取具有默认语言环境的默认格式样式的日期和时间格式化程序,输出样式：2019/1/28 下午5:23:12

getDateTimeInstance(int datastyle,int timestyle)方法 //datastyle为下面的日期长度参数，timestyle为时间长度参数

getDateTimeInstance(int datastyle,int timestyle,Locale alocale)方法

（4）getTimeInstance()方法，获取具有默认语言环境的默认格式样式的时间格式化程序，输出样式：下午5:23:12

getTimeInstance(int style)方法 //style为下面的时间长度参数

getTimeInstance(int style,Locale alocale)方法 //alocale为语言环境，如CHINA

（5）getCalendar()方法，获取与此日期/时间格式化程序关联的日历。

（6）getNumberFormat()方法，获取此日期/时间格式化程序用于格式化和解析时间的数字格式化程序。

```
package codeFormat;
import java.text.*;
import java.util.*;
public class ceshi {

    public static void main(String[] args) throws Exception {
        Date date = new Date();
        DateFormat dateFormat1 = DateFormat.getDateInstance();//DateFormat为抽象类所以创建实例的时候不能加new
        DateFormat dateFormat2 = DateFormat.getTimeInstance(DateFormat.SHORT, Locale.CHINA);
        DateFormat dateFormat3 = DateFormat.getDateTimeInstance(DateFormat.FULL, DateFormat.DEFAULT, Locale.CHINA);
        System.out.println(dateFormat1.format(date));
        System.out.println(dateFormat2.format(date));
        System.out.println(dateFormat3.format(date));

        DateFormat dateFormat4 = DateFormat.getDateInstance(DateFormat.MEDIUM, Locale.CHINA);
        DateFormat dateFormat5 = DateFormat.getTimeInstance(DateFormat.MEDIUM, Locale.CHINA);
        DateFormat dateFormat6 = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.LONG, Locale.CHINA);
        DateFormat dateFormat7 = DateFormat.getInstance();
        System.out.println(dateFormat4.format(date));
        System.out.println(dateFormat5.format(date));
        System.out.println(dateFormat6.format(date));
        System.out.println(dateFormat7.format(date));
        try {
            DateFormat dateFormat8 = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,DateFormat.MEDIUM, Locale.CHINA);
            System.out.println(dateFormat8.parse("2019年1月29日 下午2:23:45"));
        }
        catch (ParseException e) {
            System.out.println(e.getMessage());
        }
```

```
    }


}
/*运行结果：
2019年1月28日
下午6:02
2019年1月28日星期一 下午6:02:49
2019年1月28日
下午6:02:49
2019年1月28日 CST 下午6:02:49
2019/1/28 下午6:02
Tue Jan 29 14:23:45 CST 2019*/
```

十七、SimpleDataFormat类

有两种构造器：

```
    // 空参构造器，采用默认格式
    SimpleDateFormat sdf = new SimpleDateFormat();
    // 带参指定格式的构造器
    SimpleDateFormat sdf1 = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
```

参数中对应字符意义如下：
y：年
M：月
d：日
h：时
m：分
s：秒

方法的使用：

```
    Date date = new Date();             // 新建一个util.Date对象
    System.out.println(date);           // Wed Jun 24 11:48:10 CST 2020

    String format = sdf.format(date);   // 格式化：按默认的格式
    System.out.println(format);         // 20-6-24 上午11:48

    String format1 = sdf1.format(date); // 格式化，按给定的格式
    System.out.println(format1);        // 2020-06-24 12:09:58
```

值得一提的是，MMM将以中文形式输出月份：

```
    SimpleDateFormat sdf2 = new SimpleDateFormat("yyyy-MMM-dd hh:mm:ss");// 3*M
    String format1 = sdf2.format(date);
    System.out.println(format1);                // 2020-六月-24 12:16:05
```

解析字符串时间

```java
String datestr="2022-12-12 12:12:11";
//1、创建单日期格式化对象，指定时间格式必须与被解析的时间格式一致，否则会出bug
SimpleDateFormat sdf2=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
Date d2 =sdf2.parse(datestr);

System.out.println(d2);
```

十八、包装类

父类Number方法
方法名　作用
byte byteValue()　以 byte 形式返回指定的数值
abstract double doubleValue()　以 double 形式返回指定的数值
abstract float floatValue()　以 float 形式返回指定的数值
abstract int intValue()　以 int形式返回指定的数值
abstract long longValue()　以 long形式返回指定的数值
short shortValue()　以 short形式返回指定的数值
注意：　这些方法所有的数字包装类的子类都有，这些方法是负责 拆箱 的。

```java
//自动装箱
//Integer i3=Inyeger.valueOf(100);
Integer i3=100;
//自动拆箱
//int num=i3.intValue(); 将Integer类型以int返回
int num=13;
```

十九、正则表达式

```java
// 只能是 a  b  c
System.out.println("-----------1-------------");
System.out.println("a".matches( regex: "[abc]")); // true
System.out.println("z".matches( regex: "[abc]")); // false
System.out.println("ab".matches( regex: "[abc]")); // false


// 不能出现a  b  c
System.out.println("-----------2-------------");
System.out.println("a".matches( regex: "[^abc]")); // false
System.out.println("z".matches( regex: "[^abc]")); // true
System.out.println("zz".matches( regex: "[^abc]")); //false
System.out.println("zz".matches( regex: "[^abc][^abc]")); //true


//  a到z A到Z（包括头尾的范围）
System.out.println("-----------3-------------");
System.out.println("a".matches( regex: "[a-zA-Z]")); // true
System.out.println("z".matches( regex: "[a-zA-Z]")); // true
System.out.println("aa".matches( regex: "[a-zA-Z]")); //false
System.out.println("zz".matches( regex: "[a-zA-Z]")); //false
System.out.println("0".matches( regex: "[a-zA-Z]")); //false
```

它是从左到右一个一个去匹配的

如果要判断两个字符，要用两个字符类

如果要求两个范围的交集，那么要写符号&&

如果写成了一个&，那么此时&表示就不是交集了，而是一个简简单单的&符号

正则表达式

字符类(只匹配一个字符)

[abc]	只能是a, b, 或c
[^abc]	除了a, b, c之外的任何字符
[a-zA-Z]	a到z A到Z，包括（范围）
[a-d[m-p]]	a到d，或m到p
[a-z&&[def]]	a-z和def的交集。为：d, e, f
[a-z&&[^bc]]	a-z和非bc的交集。（等同于[ad-z]）
[a-z&&[^m-p]]	a到z和除了m到p的交集。（等同于[a-lq-z]）

预定义字符(只匹配一个字符)

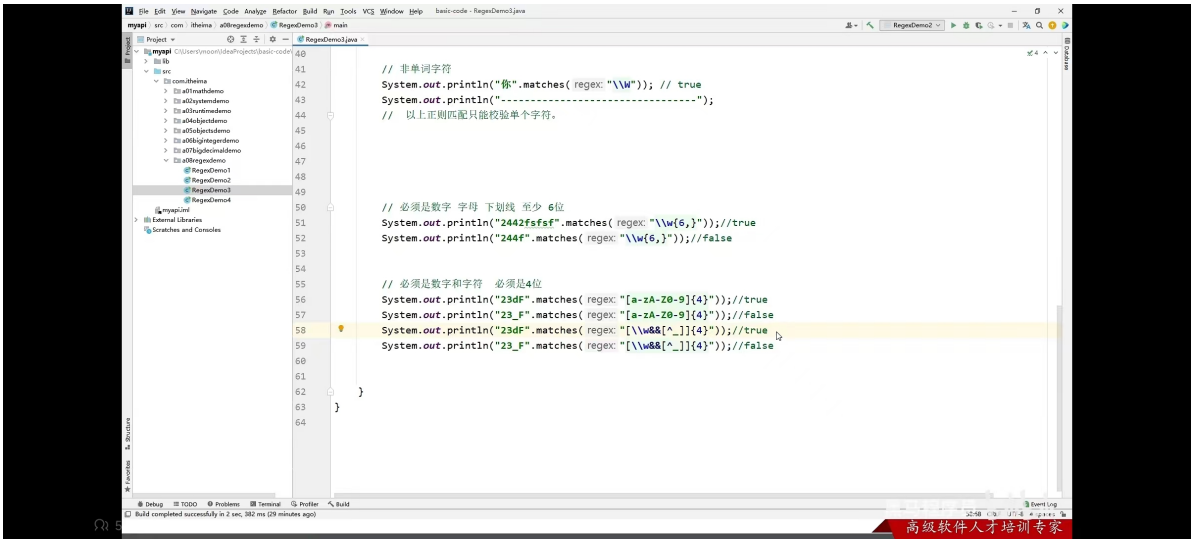
.	任何字符
\d	一个数字：[0-9]
\D	非数字：[^0-9]
\s	一个空白字符：[\t\n\x0B\f\r]
\S	非空白字符：[^\s]
\w	[a-zA-Z_0-9] 英文、数字、下划线
\W	[^\w] 一个非单词字符

```
// 只能是 a  b  c
System.out.println("-----------1-------------");
System.out.println("a".matches( regex: "[abc]")); // true
System.out.println("z".matches( regex: "[abc]")); // false
System.out.println("ab".matches( regex: "[abc]")); // false
```

```
// 不能出现a  b  c
System.out.println("-----------2-------------");
System.out.println("a".matches( regex: "[^abc]")); // false
System.out.println("z".matches( regex: "[^abc]")); // true
System.out.println("zz".matches( regex: "[^abc]")); //false
System.out.println("zz".matches( regex: "[^abc][^abc]")); //true
```

```
//  a到z A到Z（包括头尾的范围）
System.out.println("-----------3-------------");
System.out.println("a".matches( regex: "[a-zA-Z]")); // true
System.out.println("z".matches( regex: "[a-zA-Z]")); // true
System.out.println("aa".matches( regex: "[a-zA-Z]")); //false
System.out.println("zz".matches( regex: "[a-zA-Z]")); //false
System.out.println("0".matches( regex: "[a-zA-Z]")); //false
```

X?	X，一次或0次
X*	X，零次或多次
X+	X，一次或多次
X{n}	X，正好n次
X{n,}	X，至少n次
X{n,m}	X，至少n但不超过m次


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
basic-code - RegexDemo3.java
myapi > src > com > itheima > a08regexdemo > RegexDemo3 > main
RegexDemo2
Project
RegexDemo3.java
myapi E:\Users\noon\IdeaProjects\basic-code\
lib
src
com.itheima
a01mathdemo
a02systemdemo
a03runtimedemo
a04objectdemo
a05objectsdemo
a06bigintegerdemo
a07bigdecimaldemo
a08regexdemo
RegexDemo1
RegexDemo2
RegexDemo3
RegexDemo4
myapi.iml
External Libraries
Scratches and Consoles
40
41 // 非单词字符
42 System.out.println("你".matches(regex: "\\W")); // true
43 System.out.println("--------------------------------");
44 // 以上正则匹配只能校验单个字符。
45
46
47
48
49
50 // 必须是数字 字母 下划线 至少 6位
51 System.out.println("2442fsfsf".matches(regex: "\\w{6,}"));//true
52 System.out.println("244f".matches(regex: "\\w{6,}"));//false
53
54
55 // 必须是数字和字符 必须是4位
56 System.out.println("23dF".matches(regex: "[a-zA-Z0-9]{4}"));//true
57 System.out.println("23_F".matches(regex: "[a-zA-Z0-9]{4}"));//false
58 System.out.println("23dF".matches(regex: "[\\w&&[^_]]{4}"));//true
59 System.out.println("23_F".matches(regex: "[\\w&&[^_]]{4}"));//false
60
61
62 }
63 }
64
Debug TODO Problems Terminal Profiler Build
Event Log
Build completed successfully in 2 sec, 382 ms (29 minutes ago)
高级软件人才培训专家