

16 API

• 1 要求

- 一组预先定义的类、接口和方法，用于与 Java 程序库、框架或者外部系统进行交互。它为开发者提供了调用和实现特定功能的接口，简化了软件开发。
- 记一下类名和类的作用
- 养成查阅API帮助文档的习惯

• 2 Math类

- 定义
 - 是一个帮助我们用于进行数学计算的工具类
 - 私有化构造方法，所有的方法都是静态的

Math类的常用方法

方法名	说明
public static int abs(int a)	获取参数绝对值
public static double ceil(double a)	向上取整
public static double floor(double a)	向下取整
public static int round(float a)	四舍五入
public static int max(int a,int b)	获取两个int值中的较大值
public static double pow(double a,double b)	返回a的b次幂的值
public static double random()	返回值为double的随机值，范围[0.0,1.0]

• 代码实现

```
//abs 获取参数绝对值
System.out.println(Math.abs(88));//88
System.out.println(Math.abs(-88));//88
//bug:
//以int类型为例，取值范围： -2147483648 ~ 2147483647
//如果没有正数与负数对应，那么传递负数结果有误
// -2147483648 没有正数与之对应，所以abs结果产生bug
//System.out.println(Math.abs(-2147483647));//2147483647
//System.out.println(Math.absExact(-2147483648));

//进一法，往数轴的正方向进一
System.out.println(Math.ceil(12.34));//13.0
System.out.println(Math.ceil(12.54));//13.0
System.out.println(Math.ceil(-12.34));//-12.0
System.out.println(Math.ceil(-12.54));//-12.0

//去尾法,
System.out.println(Math.floor(12.34));//12.0
System.out.println(Math.floor(12.54));//12.0
System.out.println(Math.floor(-12.34));//-13.0
System.out.println(Math.floor(-12.54));//-13.0
```

```

// 四舍五入
System.out.println(Math.round(12.34)); //12
System.out.println(Math.round(12.54)); //13
System.out.println(Math.round(-12.34)); //-12
System.out.println(Math.round(-12.54)); //-13

// 获取两个整数的较大值
System.out.println(Math.max(20, 30)); //30
// 获取两个整数的较小值
System.out.println(Math.min(20, 30)); //20

// 获取a的b次幂
System.out.println(Math.pow(2, 3)); //8
// 细节:
// 如果第二个参数 0 ~ 1之间的小数
// System.out.println(Math.pow(4,0.5)); //2.0
// System.out.println(Math.pow(2,-2)); //0.25
// 建议:
// 第二个参数: 一般传递大于等于1的正整数。
System.out.println(Math.sqrt(4)); //2.0
System.out.println(Math.cbrt(8)); //2.0

for (int i = 0; i < 10; i++) {
    System.out.println(Math.random());
}

/*for (int i = 0; i < 10; i++) {
    System.out.println(Math.floor(Math.random() * 100) + 1);
    //Math.random() [0.0 1.0)
    /* 100      [0.0 100.0)
    //floor      去掉了后面的小数
    //+1        [1 100.0]
}*/ I

```

- 判断一个数是否为质数：看这个数对从2到本身取余是否为0。但是由于因子都是成对出现的。所以只需判断到这个数的平方根即可
- 3 System

- 计算机中的时间原点

1970年1月1日 08:00:00

1秒 = 1000 毫秒

1毫秒 = 1000 微秒

1微秒 = 1000 纳秒

时间原点：1970年1月1日 0:0:0，我国在东八区，有8小时时差。

- 相关方法

System也是一个工具类，提供了一些与系统相关的方法

方法名	说明
public static void exit(int status)	终止当前运行的 Java 虚拟机
public static long currentTimeMillis()	返回当前系统的时间毫秒值形式
public static void arraycopy (数据源数组, 起始索引, 目的地数组, 起始索引, 拷贝个数)	数组拷贝

- 代码实现

```
//方法的形参:  
//状态码:  
//0: 表示当前虚拟机是正常停止  
//非0: 表示当前虚拟机异常停止  
System.exit( status: 0 );  
  
long l = System.currentTimeMillis();  
System.out.println(l);  
  
long start = System.currentTimeMillis();  
  
for (int i = 1; i <= 10000; i++) {  
    boolean flag = isPrime1(i);  
    if(flag){  
        System.out.println(i);  
    }  
}  
  
long end = System.currentTimeMillis();  
//获取程序运行的总时间  
System.out.println(end - start);  
  
//拷贝数组  
int[] arr1 = {1,2,3,4,5,6,7,8,9,10};  
int[] arr2 = new int[10];  
//把arr1数组中的数据拷贝到arr2中  
//参数一：数据源，要拷贝的数据从哪个数组而来  
//参数二：从数据源数组中的第几个索引开始拷贝  
//参数三：目的地，我要把数据拷贝到哪个数组中  
//参数四：目的地数组的索引。  
//参数五：拷贝的个数  
System.arraycopy(arr1, srcPos: 0, arr2, destPos: 0, length: 10);
```

- 方法三的细节

```
//细节：  
//1.如果数据源数组和目的地数组都是基本数据类型，那么两者的类型必须保持一致，否则会报错  
//2.在拷贝的时候需要考虑数组的长度，如果超出范围也会报错  
//3.如果数据源数组和目的地数组都是引用数据类型，那么子类类型可以赋值给父类类型
```

- 4 Runtime(方法不是静态)

- 定义

Runtime表示当前虚拟机的运行环境

方法名	说明
public static Runtime getRuntime()	当前系统的运行环境对象
public void exit(int status)	停止虚拟机
public int availableProcessors()	获得CPU的线程数
public long maxMemory()	JVM能从系统中获取总内存大小 (单位byte)
public long totalMemory()	JVM已经从系统中获取总内存大小 (单位byte)
public long freeMemory()	JVM剩余内存大小 (单位byte)
public Process exec(String command)	运行cmd命令

● 代码实现

```
//1.获取Runtime的对象
//Runtime r1 = Runtime.getRuntime();

//2.exit 停止虚拟机
//Runtime.getRuntime().exit(0);

//3.获得CPU的线程数
System.out.println(Runtime.getRuntime().availableProcessors());//8

//4.总内存大小,单位byte字节
System.out.println(Runtime.getRuntime().maxMemory() / 1024 / 1024); //4064

//5.已经获取的总内存大小,单位byte字节
System.out.println(Runtime.getRuntime().totalMemory() / 1024 / 1024); //254

//6.剩余内存大小
System.out.println(Runtime.getRuntime().freeMemory() / 1024 / 1024); //251

//7.运行cmd命令
Runtime.getRuntime().exec( command: "notepad"); ↴

//7.运行cmd命令
//shutdown : 关机
//加上参数才能执行
// -s : 默认在1分钟之后关机
// -s -t 指定时间 : 指定关机时间
// -a : 取消关机操作
// -r: 关机并重启
Runtime.getRuntime().exec( command: "shutdown -s -t 3600");
```

● 5 Object

- Object是Java中的顶级父类。所有的类都直接或间接的继承于Object类。
- Object类中的方法可以被所有子类访问，所以我们要学习Object类和其中的方法。

Object的构造方法

方法名	说明
public Object()	空参构造
class Person { private String name; private int age; public Person() { super(); } public Person(String name, int age) { super(); this.name = name; this.age = age; } }	顶级父类中只有无参构造方法

● 成员方法

方法名	说明
public String toString()	返回对象的字符串表示形式
public boolean equals(Object obj)	比较两个对象是否相等
protected Object clone(int a)	对象克隆

- 代码实现

- **toString** 方法的结论

```
//1.toString 返回对象的字符串表示形式
Object obj = new Object();
String str1 = obj.toString();
System.out.println(str1); //java.lang.Object@119d7047
```

```
Student stu = new Student();
String str2 = stu.toString();
System.out.println(str2); //com.itheima.a04objectdemo.Student@4eec7777
```

//细节：
//System: 类名
//out: 静态变量
//System.out: 获取打印的对象
//println(): 方法
//参数: 表示打印的内容
//核心逻辑：
//当我们打印一个对象的时候，底层会调用对象的**toString**方法，把对象变成字符串。
//然后再打印在控制台上，打印完毕换行处理。

//思考：默认情况下，因为**Object**类中的**toString**方法返回的是地址值
//所以，默认情况下，打印一个对象打印的就是地址值
//但是地址值对于我们是没什么意义的？
//我想要看到对象内部的属性值？我们该怎么办？
//处理方案：重写父类**Object**类中的**toString**方法
System.out.println(stu); //com.itheima.a04objectdemo.Student@4eec7777

//**toString**方法的结论：
//如果我们打印一个对象，想要看到属性值的话，那么就重写**toString**方法就可以了。
//在重写的方法中，把对象的属性值进行拼接。

- 如果我们打印一个对象，想要看到属性值的话，那么就重写**toString**方法就可以了。
- 在重写的方法中，把对象的属性值进行拼接。（用**ptg**重写）
- **equals**方法的结论

```

String s = "abc";
StringBuilder sb = new StringBuilder("abc");

System.out.println(s.equals(sb)); // false
//因为equals方法是被s调用的，而s是字符串
//所以equals要看String类中的
//字符串中的equals方法，先判断参数是否为字符串
//如果是字符串，再比较内部的属性
//但是如果参数不是字符串，直接返回false

System.out.println(sb.equals(s)); // false
//因为equals方法是被sb调用的，而sb是StringBuilder
//所以这里的equals方法要看StringBuilder中的equals方法
//那么在StringBuilder当中，没有重写equals方法
//使用的是Object中的
//在Object当中默认是使用==号比较两个对象的地址值
//而这里的s和sb记录的地址值是不一样的，所以结果返回false

```

- 如果没有重写equals方法，那么默认使用object中的方法进行比较，比较的是地址值是否相等
- 一般来讲地址值对于我们意义不大，所以我们会重写，重写之后比较的就是对象内部的属性值了。

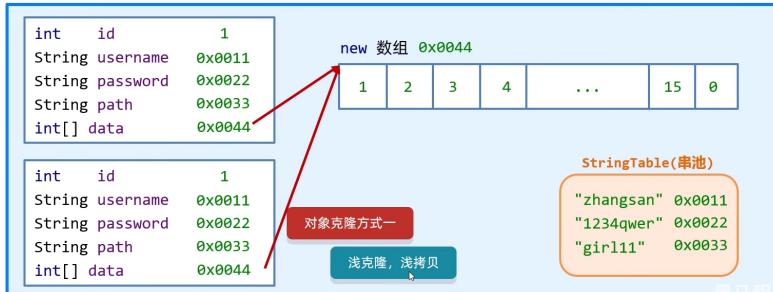
• 对象克隆

- 方法在底层会帮我们创建一个对象，并把原对象中的数据拷贝过去。
- 1.重写object中的clone方法
- 2.让javabean类实现cloneable接口
- 3.创建原对象并调用clone就可以了。

对象克隆

把A对象的属性值完全拷贝给B对象，也叫对象拷贝,对象复制

堆内存



对象克隆



对象克隆

把A对象的属性值完全拷贝给B对象，也叫对象拷贝,对象复制

浅克隆

不管对象内部的属性是基本数据类型还是引用数据类型，都完全拷贝过来

深克隆

基本数据类型拷贝过来
字符串复用
引用数据类型会重新创建新的

1. Object是Java中的顶级父类。

所有的类都直接或间接的继承于Object类。

2. `toString()`: 一般会重写，打印对象时打印属性

3. `equals()`: 比较对象时会重写，比较对象属性值是否相同

4. `clone()`: 默认浅克隆。

如果需要深克隆需要重写方法或者使用第三方工具类。

```
//第三方的工具
//1. 第三方写的代码导入到项目中
//2. 编写代码
Gson gson = new Gson();
//把对象变成一个字符串
String s = gson.toJson(u1);
//再把字符串变回对象就可以了
User user = gson.fromJson(s, User.class);
//打印对象
System.out.println(user);
```

- 6 Objects

- 成员方法

方法名	说明
public static boolean equals(Object a, Object b)	先做非空判断，比较两个对象
public static boolean isNull(Object obj)	判断对象是否为null，为null返回true，反之
public static boolean nonNull(Object obj)	判断对象是否为null，跟isNull的结果相反

- **equals方法**

```
boolean result = Objects.equals(s1, s2);
System.out.println(result);
//细节:
//1. 方法的底层会判断s1是否为null，如果为null，直接返回false
//2. 如果s1不为null，那么就利用s1再次调用equals方法
//3. 此时s1是Student类型，所以最终还是会调用Student中的equals方法。
// 如果没有重写，比较地址值，如果重写了，就比较属性值。
```

- **isNull和nonNull方法**

```
//public static boolean isNull(Object obj) 判断对象是否为null，为null返回true，反之
Student s3 = new Student();
Student s4 = null;
System.out.println(Objects.isNull(s3));//false
System.out.println(Objects.isNull(s4));//true

System.out.println(Objects.nonNull(s3));//true
System.out.println(Objects.nonNull(s4));//false
```

- **总结**

1. Objects是一个对象工具类，提供了一些操作对象的方法
2. **equals(对象1, 对象2):**先做非空判断，比较两个对象
3. **isNull(对象):**判断对象是否为空
4. **nonNull(对象):**判断对象是否不是空

- **7. BigInteger 和 BigDecimal**

- **BigInteger:** 处理任意精度整数的类，超越了int和long的数值范围
- **构造方法和一个静态构造方法**

方法名	说明
public BigInteger(int num, Random rnd)	获取随机大整数，范围：[0 ~ 2的num次方-1]
public BigInteger(String val)	获取指定的大整数
public BigInteger(String val, int radix)	获取指定进制的大整数
public static BigInteger valueOf(long val)	静态方法获取BigInteger的对象，内部有优化

对象一旦创建，内部记录的值不能发生改变

- **代码实现**

```

//1. 获取一个随机的大整数
/* Random r = new Random();
for (int i = 0; i < 100; i++) {
    BigInteger bd1 = new BigInteger(4,r);
    System.out.println(bd1); // [0 ~ 15]
}*/



//2. 获取一个指定的大整数
//细节：字符串中必须是整数，否则会报错
/* BigInteger bd2 = new BigInteger("1.1");
System.out.println(bd2);*/



/* BigInteger bd3 = new BigInteger("abc");
System.out.println(bd3);*/



//3. 获取指定进制的大整数
//细节：
//1. 字符串中的数字必须是整数
//2. 字符串中的数字必须要跟进制吻合。
//比如二进制中，那么只能写0和1，写其他的就报错。
/*BigInteger bd4 = new BigInteger("123",2);
System.out.println(bd4);*/



//4. 静态方法获取BigInteger的对象，内部有优化
//细节：
//1. 能表示范围比较小，只能在long的取值范围之内，如果超出long的范围就不行了。
//2. 在内部对常用的数字： -16 ~ 16 进行了优化。
// 提前把 -16 ~ 16 先创建好BigInteger的对象，如果多次获取不会重新创建新的。
BigInteger bd5 = BigInteger.valueOf(16);
BigInteger bd6 = BigInteger.valueOf(16);

System.out.println(bd5 == bd6); // true



BigInteger bd7 = BigInteger.valueOf(17);
BigInteger bd8 = BigInteger.valueOf(17);

System.out.println(bd7 == bd8); // false



//5. 对象一旦创建内部的数据不能发生改变
BigInteger bd9 = BigInteger.valueOf(1);
BigInteger bd10 = BigInteger.valueOf(2);
BigInteger result = bd9.add(bd10);
System.out.println(result); // 3
// 此时，不会修改参与计算的BigInteger对象中的值，而是产生了一个新的BigInteger对象记录3

System.out.println(bd9 == result); // false
System.out.println(bd10 == result); // false

```

- 总结

BigInteger构造方法小结

- ① 如果BigInteger表示的数字**没有超出**long的范围，可以用静态方法获取。
- ② 如果BigInteger表示的**超出**long的范围，可以用构造方法获取。
- ③ 对象一旦创建，BigInteger内部记录的值不能发生改变。
- ④ 只要进行计算都会产生一个新的BigInteger对象

- 成员方法

方法名	说明
public BigInteger add(BigInteger val)	加法
public BigInteger subtract(BigInteger val)	减法
public BigInteger multiply(BigInteger val)	乘法
public BigInteger divide(BigInteger val)	除法, 获取商
public BigInteger[] divideAndRemainder(BigInteger val)	除法, 获取商和余数
public boolean equals(Object x)	比较是否相同
public BigInteger pow(int exponent)	次幂
public BigInteger max/min(BigInteger val)	返回较大值/较小值
public int intValue(BigInteger val)	转为int类型整数, 超出范围数据有误

- 代码实现

```
//1. 创建两个BigInteger对象
BigInteger bd1 = BigInteger.valueOf(10);
BigInteger bd2 = BigInteger.valueOf(2);

//2. 加法
BigInteger bd3 = bd1.add(bd2);
System.out.println(bd3);

//3. 除法, 获取商和余数
BigInteger[] arr = bd1.divideAndRemainder(bd2);
System.out.println(arr[0]);
System.out.println(arr[1]);

//4. 比较是否相同
boolean result = bd1.equals(bd2);
System.out.println(result);

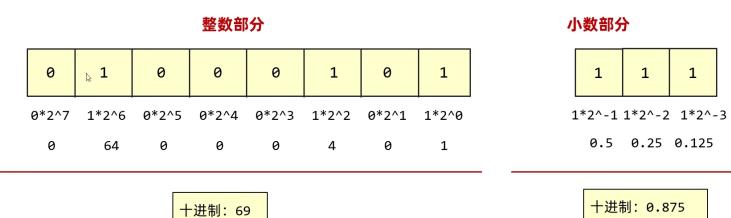
//5. 次幂
BigInteger bd4 = bd1.pow(2);
System.out.println(bd4);

//6. max
BigInteger bd5 = bd1.max(bd2);

//7. 转为int类型整数, 超出范围数据有误
/* BigInteger bd6 = BigInteger.valueOf(2147483647L);
int i = bd6.intValue();
System.out.println(i); */

BigInteger bd6 = BigInteger.valueOf(200);
double v = bd6.doubleValue();
System.out.println(v);
```

- 8 BigDecimal



类型	占用字节数	总bit位数	小数部分bit位数
float	4个字节	32个bit位	23个bit位
double	8个字节	64个bit位	52个bit位

- 用于小数的精确计算

- 用来表示很大的小数

- 构造方法

构造方法获取`BigDecimal`对象

```
public BigDecimal(double val)
public BigDecimal(String val)
```

静态方法获取`BigDecimal`对象

```
public static BigDecimal valueOf(double val)
```

- 代码实现

```
//1.通过传递double类型的小数来创建对象
//细节:
//这种方式有可能是不精确的, 所以不建议使用
BigDecimal bd1 = new BigDecimal(val: 0.01);
BigDecimal bd2 = new BigDecimal(val: 0.09);

// System.out.println(bd1);
// System.out.println(bd2);

//2.通过传递字符串表示的小数来创建对象
BigDecimal bd3 = new BigDecimal(val: "0.01");
BigDecimal bd4 = new BigDecimal(val: "0.09");
BigDecimal bd5 = bd3.add(bd4);
System.out.println(bd3);
System.out.println(bd4);
System.out.println(bd5);

//3.通过静态方法获取对象
BigDecimal bd6 = BigDecimal.valueOf(10);
System.out.println(bd6);

//细节:
//1.如果要表示的数字不大, 没有超出double的取值范围, 建议使用静态方法
//2.如果要表示的数字比较大, 超出了double的取值范围, 建议使用构造方法
//3.如果我们传递的是0~10之间的整数, 包含0, 包含10, 那么方法会返回已经创建好的对象, 不会重新new
```

- 如果是10.0, 那么, 其地址值不一样。因为只有0~10的整数, 才会用创建好的对象

- 成员方法

`BigDecimal`的使用

方法名	说明
public static BigDecimal valueOf(double val)	获取对象
public BigDecimal add(BigDecimal val)	加法
public BigDecimal subtract(BigDecimal val)	减法
public BigDecimal multiply(BigDecimal val)	乘法
public BigDecimal divide(BigDecimal val)	除法
public BigDecimal divide(BigDecimal val, 精确几位, 舍入模式)	除法

- 代码实现

```

//1.加法
BigDecimal bd1 = BigDecimal.valueOf(10.0);
BigDecimal bd2 = BigDecimal.valueOf(3.0);
BigDecimal bd3 = bd1.add(bd2);
//System.out.println(bd3);//12.0

//2.减法
BigDecimal bd4 = bd1.subtract(bd2);
//System.out.println(bd4);//8.0

//3.乘法
BigDecimal bd5 = bd1.multiply(bd2);
// System.out.println(bd5);//20.00

//4.除法
BigDecimal bd6 = bd1.divide(bd2, scale: 2, RoundingMode.HALF_UP);
System.out.println(bd6);//3.33

//1.加法
BigDecimal bd1 = BigDecimal.valueOf(10.0);
BigDecimal bd2 = BigDecimal.valueOf(3.0);
BigDecimal bd3 = bd1.add(bd2);
//System.out.println(bd3);//12.0

//2.减法
BigDecimal bd4 = bd1.subtract(bd2);
//System.out.println(bd4)//8.0

//3.乘法
BigDecimal bd5 = bd1.multiply(bd2);
// System.out.println(bd5)//20.00

//4.除法
BigDecimal bd6 = bd1.divide(bd2, scale: 2, RoundingMode.HALF_UP);
System.out.println(bd6)//3.33

```

- 底层存储方法（存每一位对应的ASCII码）

BigDecimal底层存储方式

```
BigDecimal bd = new BigDecimal("0.226");
```

```
[ 48, 46, 50, 50, 54]
```

```
BigDecimal bd = new BigDecimal("123.226");
```

```
[ 49, 50, 51, 46, 50, 50, 54]
```

```
BigDecimal bd = new BigDecimal("1.5");
```

```
[ 45, 49, 46, 53]
```

- 总结

1. BigDecimal的作用是什么?

- 表示较大的小数和解决小数运算精度失真问题。

2. BigDecimal的对象如何获取?

- `BigDecimal bd1 = new BigDecimal("较大的小数");`
- `BigDecimal bd2 = BigDecimal.valueOf(0.1);`

3. 常见操作

加: `add`

减: `subtract`

乘: `multiply`

除: `divide` (四舍五入: `RoundingMode.HALF_UP`)

• 9 正则表达式

• 使用

- API帮助文档搜Pattern类

• 作用

作用一: 校验字符串是否满足规则

作用二: 在一段文本中查找满足要求的内容

• 定义

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

- []表示范围, 字符串出现的字符一定在这个范围内, 且大括号里面的内容只能出现一个。就是一个大括号去匹配一个字符。

```
System.out.println("ab".matches(regex: "[abc]")); // false
System.out.println("ab".matches(regex: "[abc][abc]")); // true
```

- ^表示取反的意思。
- 要求两个范围的交集, 符号为&&。仅写一个&代表是一个简单的&符号。
- \为转义字符, 改变后面那个字符原本的含义。
- 如果要表示字符\" , 那么应该打印\\, 前一个\为转义字符
- .表示任意何字符 ..表示两个任意字符

```

System.out.println("你".matches( regex: "...")); //false
System.out.println("你".matches( regex: ".") ); //true
System.out.println("你a".matches( regex: "\u202a" ));//true
// \d 表示任意的一个数字
// \d只能是任意的一位数字
// 简单来记：两个\表示一个\
System.out.println("a".matches( regex: "\d" )); // false
System.out.println("3".matches( regex: "\d" )); // true
System.out.println("333".matches( regex: "\d" )); // false
System.out.println("333".matches( regex: "\d\d\d" )); // true

```

- 数量词

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

```

// 必须是数字 字母 下划线 至少 6位
System.out.println("2442fsfsf".matches( regex: "\w{6,}" ));//true
System.out.println("244f".matches( regex: "\w{6,}" ));//false

```

```

// 必须是数字和字符 必须是4位
System.out.println("23dF".matches( regex: "[a-zA-Z0-9]{4}" ));//true
System.out.println("23_F".matches( regex: "[a-zA-Z0-9]{4}" ));//false
System.out.println("23dF".matches( regex: "[\w&[^_]]{4}" ));//true
System.out.println("23_F".matches( regex: "[\w&[^_]]{4}" ));//false

```

- 插件: any-rule

- 忽略大小写

```

//忽略大小写的书写方式
//在匹配的时候忽略abc的大小写
String regex4 = "(?i)abc";
System.out.println("-----");
System.out.println("abc".matches(regex4));
System.out.println("ABC".matches(regex4));
System.out.println("aBC".matches(regex4));

//忽略大小写的书写方式
//在匹配的时候忽略abc的大小写
String regex4 = "a(?i)bc";
System.out.println("-----");
System.out.println("abc".matches(regex4));//true
System.out.println("ABC".matches(regex4));//false
System.out.println("aBC".matches(regex4));//true

```

- 心得

```
//编写正则的小心得:  
//第一步: 按照正确的数据进行拆分  
//第二步: 找每一部分的规律, 并编写正则表达式  
//第三步: 把每一部分的正则拼接在一起, 就是最终的结果  
//书写的时候: 从左到右去书写。
```

- 总结

符号	含义	举例
[]	里面的内容出现一次	[0-9] [a-zA-Z0-9]
()	分组	a(bc)+
^	取反	[^abc]
&&	交集, 不能写单个的&	[a-z&&m-p]
	写在方括号外面表示并集	[a-zA-Z0-9] x X
.	任意字符	\n 回车符号不匹配
\	转义字符	\d
\d	0-9	\d+
\D	非0-9	\D+
\s	空白字符	[\t\n\x0B\f\r]
\S	非空白字符	[^\s]
\w	单词字符	[a-zA-Z_0-9]
\W	非单词字符	[^\w]

符号	含义	举例
?	0次或1次	\d?
*	0次或多次	\d* (abc)*
+	1次或多次	\d+ (abc)+
{}	具体次数	a{7} \d{7,19}
(?i)	忽略后面字符的大小写	(?i)abc
a((?i)b)c	只忽略b的大小写	a((?i)b)c

- 10 爬虫

```
//Pattern: 表示正则表达式  
//Matcher: 文本匹配器, 作用按照正则表达式的规则去读取字符串, 从头开始读取。  
// 在大串中去找符合匹配规则的子串。
```

```
//1. 获取正则表达式的对象  
Pattern p = Pattern.compile("Java\\d{0,2}");  
//2. 获取文本匹配器的对象  
//m就是Matcher对象, m在大串str中找到p规则的小串。  
Matcher m = p.matcher(str);  
//3. 利用循环获取  
while (m.find()) {  
    String s = m.group();  
    System.out.println();  
}
```

- 1: 有条件的爬取

```
//1. 定义正则表达式  
//?理解为前面的数据Java  
//=表示在Java后面要跟随的数据  
//但是在获取的时候，只获取前半部分  
String regex = "Java(?=8|11|17)";
```

- 正向先行断言 (`(?=...)`) 检查后面的内容是否满足某个条件，但不把它包括在匹配结果中
- 负向先行断言 (`(?!...)`) 检查后面的内容是否不满足某个条件，但不把它包括在匹配结果中。
- 非捕获组 (`(?:...)`) 将括号内的内容视为一个整体进行匹配，但不捕获。
- 具体例子看正则表达式Test6

• 2: 贪婪爬取和非贪婪爬取

贪婪爬取：在爬取数据的时候尽可能的多获取数据

非贪婪爬取：在爬取数据的时候尽可能的少获取数据

Java当中，默认的就是贪婪爬取

如果我们在数量词`+ *` 的后面加上问号，那么此时就是非贪婪爬取

`ab+:`

贪婪爬取：aaaaaaaaaaaa

非贪婪爬取：`ab`

• 3: 在字符串方法中的使用

方法名	说明
<code>public String[] matches(String regex)</code>	判断字符串是否满足正则表达式的规则
<code>public String replaceAll(String regex, String newStr)</code>	按照正则表达式的规则进行替换
<code>public String[] split(String regex)</code>	按照正则表达式的规则切割字符串

- 代码实现

```

String s = "小诗诗dqwefqwfqwfq12312小丹丹dqwefqwfqwfq12312小惠惠";
//细节:
//方法在底层跟之前一样也会创建文本解析器的对象
//然后从头开始去读取字符串中的内容，只要有满足的，那么就用第二个参数去替换。
//String resut1 = s.replaceAll("[\\w&&[^_]]+", "vs");
//System.out.println(resut1);

String[] arr = s.split( regex: "[\\w&&[^_]]+" );
for (int i = 0; i < arr.length; i++) {
    System.out.println(arr[i]);
}

```

- 4: 分组

- 每一个小括号就是一组

每组是有组号的，也就是序号。

规则1：从1开始，连续不间断。

规则2：以左括号为基准，最左边的是第一组，其次为第二组，以此类推。

$\backslash(\backslash\backslash d+)(\backslash\backslash d+)(\backslash\backslash d+)$

- 捕获分组

捕获分组就是把这一组的数据捕获出来，再用一次。

后续还要继续使用本组的数据。

正则内部使用：\组号

正则外部使用：\$组号

```
String str = "我要学学编编编程程程程程程";
```

//需求：把重复的内容 替换为 单个的

//学学 学

//编编编 编

//程程程程程 程

//(.) 表示把重复内容的第一个字符看做一组

//\1 表示第一字符再次出现

//+ 至少一次

//\$1 表示把正则表达式中第一组的内容，再拿出来用

```
String result = str.replaceAll( regex: "(.)\\1+", replacement: "$1" );
```

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

- 非捕获分组：仅仅把数据括起来，但是不占用组号

分组之后不再需要再用本组数据，仅仅是把数据括起来。

↳

符号	含义	举例
(?: 正则)	获取所有	Java(?:8 11 17)
(?= 正则)	获取前面部分	Java(?=8 11 17)
(?! 正则)	获取不是指定内容的前面部分	Java(?!=8 11 17)

- 总结

1. 正则表达式中分组有两种：
捕获分组、非捕获分组
2. 捕获分组（默认）：
可以获取每组中的内容反复使用。
3. 组号的特点：
从1开始，连续不间断
以左括号为基准，最左边的是第一组
4. 非捕获分组：
分组之后不再需要再用本组数据，仅仅把数据括起来，不占组号。
(?:) (?=) (?!)

- 12 JDK8 新增的时间相关类

为什么要学JDK8新增时间相关类呢？

代码层面

安全层面

JDK7：代码麻烦

日期对象  计算
毫秒值
比较

JDK7：多线程环境下会导致数据安全的问题

JDK8：简单

判断的方法
计算时间间隔的方法

JDK8：时间日期对象都是不可变的，解决了这个问题



- 12.1 ZoneID 时区

方法名	说明
static Set<String> getAvailableZoneIds()	获取Java中支持的所有时区
static ZoneId systemDefault()	获取系统默认时区
static ZoneId of(String zoneId)	获取一个指定时区

- 代码实现

```
//1. 获取所有的时区名称
Set<String> zoneIds = ZoneId.getAvailableZoneIds();
System.out.println(zoneIds.size()); //600
System.out.println(zoneIds); // Asia/Shanghai
```

```
//2. 获取当前系统的默认时区
ZoneId zoneId = ZoneId.systemDefault();
System.out.println(zoneId); //Asia/Shanghai
```

```
//3. 获取指定的时区
ZoneId zoneId1 = ZoneId.of("Asia/Pontianak");
System.out.println(zoneId1);
```

- 12.2 Instant 时间戳

方法名	说明
static Instant now()	获取当前时间的Instant对象（标准时间）
static Instant ofXXX(long epochMilli)	根据（秒/毫秒/纳秒）获取Instant对象
ZonedDateTime atZone(ZoneId zone)	指定时区
boolean isXXX(Instant otherInstant)	判断系列的方法
Instant minusXXX(long millisToSubtract)	减少时间系列的方法
Instant plusXXX(long millisToSubtract)	增加时间系列的方法

- 代码实现

```

//1. 获取当前时间的Instant对象（标准时间）
//Instant now = Instant.now();
//System.out.println(now);

//2. 根据（秒/毫秒/纳秒）获取Instant对象
Instant instant1 = Instant.ofEpochMilli(0L);
System.out.println(instant1); //1970-01-01T00:00:00Z

Instant instant2 = Instant.ofEpochSecond(1L);
System.out.println(instant2); //1970-01-01T00:00:01Z

Instant instant3 = Instant.ofEpochSecond( epochSecond: 1L, nanoAdjustment: 1000000000L );
System.out.println(instant3); //1970-01-01T00:00:02Z

//3. 指定时区
ZonedDateTime time = Instant.now().atZone(ZoneId.of("Asia/Shanghai"));
System.out.println(time);

//4.isXXX 判断
Instant instant4 = Instant.ofEpochMilli(0L);
Instant instant5 = Instant.ofEpochMilli(1000L);

//5. 用于时间的判断
//isBefore: 判断调用者代表的时间是否在参数表示时间的前面
boolean result1 = instant4.isBefore(instant5);
System.out.println(result1); //true

//isAfter: 判断调用者代表的时间是否在参数表示时间的后面
boolean result2 = instant4.isAfter(instant5);
System.out.println(result2); //false

//6. Instant minusXXX(long millisToSubtract)    减少时间系列的方法
Instant instant6 = Instant.ofEpochMilli(3000L);
System.out.println(instant6); //1970-01-01T00:00:03Z

Instant instant7 = instant6.minusSeconds(1);
System.out.println(instant7); //1970-01-01T00:00:02Z

```

- 12.3 ZoneDateTime带时间区的时间

方法名	说明
static ZonedDateTime now()	获取当前时间的ZonedDateTime对象
static ZonedDateTime ofXxx(。。。)	获取指定时间的ZonedDateTime对象
ZonedDateTime withXxx(时间)	修改时间系列的方法
ZonedDateTime minusXxx(时间)	减少时间系列的方法
ZonedDateTime plusXxx(时间)	增加时间系列的方法

- 代码实现

```

//1. 获取当前时间对象（带时区）
ZonedDateTime now = ZonedDateTime.now();
System.out.println(now);

//2. 获取指定的时间对象（带时区）
//年月日时分秒纳秒方式指定
ZonedDateTime time1 = ZonedDateTime.of( year: 2023, month: 10, dayOfMonth: 1,
                                         hour: 11, minute: 12, second: 12, nanoOfSecond: 0,
                                         ZoneId.of("Asia/Shanghai"));
System.out.println(time1);

//通过Instant + 时区的方式指定获取时间对象
Instant instant = Instant.ofEpochMilli(0L);
ZoneId zoneId = ZoneId.of("Asia/Shanghai");
ZonedDateTime time2 = ZonedDateTime.ofInstant(instant, zoneId);
System.out.println(time2);

//3.withXxx 修改时间系列的方法
ZonedDateTime time3 = time2.withYear(2000);
System.out.println(time3);

//4. 减少时间
ZonedDateTime time4 = time3.minusYears(1);
System.out.println(time4);

//5. 增加时间
ZonedDateTime time5 = time4.plusYears(1);
System.out.println(time5);

```

```
//细节：  
//JDK8新增的时间对象都是不可变的  
//如果我们修改了，减少了，增加了时间  
//那么调用者是不会发生改变的，产生一个新的时间。
```

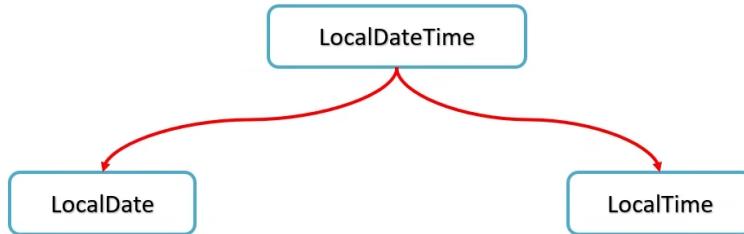
- 12.4 DateTimeFormatter 用于时间的格式化和解析

方法名	说明
static DateTimeFormatter ofPattern (格式)	获取格式对象
String format (时间对象)	按照指定方式格式化

```
//获取时间对象  
ZonedDateTime time = Instant.now().atZone(ZoneId.of("Asia/Shanghai"));  
  
// 解析/格式化器  
DateTimeFormatter dtf1 = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss EE a");  
// 格式化  
System.out.println(dtf1.format(time));
```

- 12.5 LocalDate、LocalTime、LocalDateTime

方法名	说明
static XXX now()	获取当前时间的对象
static XXX of(...)	获取指定时间的对象
get 开头的方法	获取日历中的年、月、日、时、分、秒等信息
isBefore, isAfter	比较两个 LocalDate
with 开头的	修改时间系列的方法
minus 开头的	减少时间系列的方法
plus 开头的	增加时间系列的方法



方法名	说明
public LocalDate toLocalDate()	LocalDateTime转换成一个LocalDate对象
public LocalTime toLocalTime()	LocalDateTime转换成一个LocalTime对象

```

// 判断今天是否是你的生日
LocalDate birDate = LocalDate.of( year: 2000, month: 1, dayOfMonth: 1);
LocalDate nowDate1 = LocalDate.now();

MonthDay birMd = MonthDay.of(birDate.getMonthValue(), birDate.getDayOfMonth());
MonthDay nowMd = MonthDay.from(nowDate1);

System.out.println("今天是你的生日吗? " + birMd.equals(nowMd)); //今天是你的生日吗? | ↵
  
```

- 12.6 Duration、Period、ChronoUnit

```

// 当前时间
LocalDateTime today = LocalDateTime.now();
System.out.println(today);

// 生日时间
LocalDateTime birthDate = LocalDateTime.of( year: 2000, month: 1, dayOfMonth: 1,
                                             hour: 0, minute: 0, second: 0);
System.out.println(birthDate);

System.out.println("相差的年数: " + ChronoUnit.YEARS.between(birthDate, today));
System.out.println("相差的月数: " + ChronoUnit.MONTHS.between(birthDate, today));
System.out.println("相差的周数: " + ChronoUnit.WEEKS.between(birthDate, today));
System.out.println("相差的天数: " + ChronoUnit.DAYS.between(birthDate, today));
System.out.println("相差的时数: " + ChronoUnit.HOURS.between(birthDate, today));
System.out.println("相差的分数: " + ChronoUnit.MINUTES.between(birthDate, today));
System.out.println("相差的秒数: " + ChronoUnit.SECONDS.between(birthDate, today));
System.out.println("相差的毫秒数: " + ChronoUnit.MILLISECONDS.between(birthDate, today));
System.out.println("相差的微秒数: " + ChronoUnit.MICROS.between(birthDate, today));
System.out.println("相差的纳秒数: " + ChronoUnit.NANOS.between(birthDate, today));
System.out.println("相差的半天数: " + ChronoUnit.HALF_DAYS.between(birthDate, today));
System.out.println("相差的十年数: " + ChronoUnit.DECADES.between(birthDate, today));
System.out.println("相差的世纪(百年)数: " + ChronoUnit.CENTURIES.between(birthDate, today));
System.out.println("相差的千年数: " + ChronoUnit.MILLENNIA.between(birthDate, today));
System.out.println("相差的纪元数: " + ChronoUnit.ERAS.between(birthDate, today));
  
```

- 11 JDK 7 前时间相关类

Date	时间
SimpleDateFormat	格式化时间
Calendar	日历

世界标准时间：

格林尼治时间 / 格林威治时间 (Greenwich Mean Time) 简称 GMT。

目前世界标准时间 (UTC) 已经替换为：原子钟

中国标准时间：

世界标准时间 **+ 8小时**

时间单位换算：

1秒 = 1000毫秒

1毫秒 = 1000微秒

1微秒 = 1000纳秒

● 11.1 Date时间类

Date 类是一个JDK写好的JavaBean类，用来描述时间，精确到毫秒。

利用空参构造创建的对象，默认表示系统当前时间。

利用有参构造创建的对象，表示指定的时间。

1、如何创建日期对象？

- **Date date = new Date();**
- **Date date = new Date(指定毫秒值);**

2、如何修改时间对象中的毫秒值

- **setTime(毫秒值);**

3、如何获取时间对象中的毫秒值

- **getTime();**

● 代码实现

```
//1. 创建对象表示一个时间  
Date d1 = new Date();  
//System.out.println(d1);  
  
//2. 创建对象表示一个指定的时间  
Date d2 = new Date(0L);  
System.out.println(d2);  
  
//3.setTime 修改时间  
//1000毫秒 = 1 秒  
d2.setTime(1000L);  
System.out.println(d2);  
  
//4.getTime 获取当前时间的毫秒值  
long time = d2.getTime();  
System.out.println(time);
```

● 11.2 SimpleDateFormat类作用

● **格式化：**把时间变成我们喜欢的格式。

● **解析：**把字符串表示的时间变成Date对象。

● 构造方法

构造方法	说明
public SimpleDateFormat()	构造一个SimpleDateFormat，使用默认格式
public SimpleDateFormat(String pattern)	构造一个SimpleDateFormat，使用指定的格式

//1. 利用空参构造创建SimpleDateFormat对象， 默认格式

```
SimpleDateFormat sdf1 = new SimpleDateFormat();
Date d1 = new Date(0L);
String str1 = sdf1.format(d1);
System.out.println(str1); //1970/1/1 上午8:00
```

//2. 利用带参构造创建SimpleDateFormat对象， 指定格式

```
SimpleDateFormat sdf2 = new SimpleDateFormat(pattern: "yyyy年MM月dd日 HH:mm:ss");
String str2 = sdf2.format(d1);
System.out.println(str2); //1970年01月01日 08:00:00
```

- 常用方法

常用方法	说明
public final String format(Date date)	格式化 (日期对象 -> 字符串)
public Date parse(String source)	解析 (字符串 -> 日期对象)

//1. 定义一个字符串表示时间

```
String str = "2023-11-11 11:11:11";
//2. 利用空参构造创建SimpleDateFormat对象
//细节:
//创建对象的格式要跟字符串的格式完全一致
SimpleDateFormat sdf = new SimpleDateFormat(pattern: "yyyy-MM-dd HH:mm:ss");
Date date = sdf.parse(str);
//3. 打印结果
System.out.println(date);
```

- 常用的模式对应关系

y	年		
M	月	2023-11-11 13:27:06	2023年11月11日 13:27:06
d	日		
H	时	yyyy-MM-dd HH:mm:ss	yyyy年MM月dd日 HH:mm:ss
m	分		
s	秒		

- 总结

- SimpleDateFormat的两个作用

格式化

解析

- 如何指定格式

yyyy年MM月dd日 HH: mm: ss

- 11.3 Calendar

- Calendar代表了系统当前时间的日历对象，可以单独修改、获取时间中的年，月，日
- **细节：**Calendar是一个抽象类，不能直接创建对象。

获取Calendar日历类对象的方法

方法名	说明
public static Calendar getInstance()	获取当前时间的日历对象

- 常用方法

方法名	说明
public final Date getTime()	获取日期对象
public final setTime(Date date)	给日历设置日期对象
public long getTimeInMillis()	拿到时间毫秒值
public void setTimeInMillis(long millis)	给日历设置时间毫秒值
public int get(int field)	取日历中的某个字段信息
public void set(int field,int value)	修改日历的某个字段信息
public void add(int field,int amount)	为某个字段增加/减少指定的值

- 代码实现

```
//1. 获取日历对象
//细:1: Calendar是一个抽象类，不能直接new，而是通过一个静态方法获取到子类对象
//底层原理:
//会根据系统的不同时区来获取不同的日历对象，默认表示当前时间。
//会把时间中的纪元，年，月，日，时，分，秒，星期，等等的都放到一个数组当中
//细节2:
//月份: 范围0~11 如果获取出来的是0.那么实际上是1月。
//星期: 在老外的眼里，星期日是一周中的第一天
//      1 (星期日)    2 (星期一)    3 (星期二)    4 (星期三)    5 (星期四)    6 (星期五)    7 (星期六)

Calendar c = Calendar.getInstance();

//2. 修改一下日历代表的时间
Date d = new Date(0L);
c.setTime(d);

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

```

//java在Calendar类中，把索引对应的数字都定义成常量
int year = c.get(Calendar.YEAR);
int month = c.get(Calendar.MONTH) + 1;
int date = c.get(Calendar.DAY_OF_MONTH);
int week = c.get(Calendar.DAY_OF_WEEK);
System.out.println(year + ", " + month + ", " + date + ", " + getWeek(week)); //1970, 1, 1, 星期四

//查表法：
//表：容器
//让数据跟索引产生对应的关系

//传入对应的数字： 1 ~7
//返回对应的星期
public static String getWeek(int index){
    //定义一个数组，让汉字星期几 跟1~7产生对应关系
    String[] arr = {"", "星期日", "星期一", "星期二", "星期三", "星期四", "星期五", "星期六"};
    //根据索引返回对应的星期
    return arr[index];
}

c.set(Calendar.YEAR, 2000);
c.set(Calendar.MONTH, 11); //调用方法在这个基础上增加一个月
c.add(Calendar.MONTH, amount: 1);

```

- 13 包装类

包装类：基本数据类型对应的引用类型

byte	Byte
short	Short
char	Character
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

方法名	说明
public Integer(int value)	根据传递的整数创建一个Integer对象
public Integer(String s)	根据传递的字符串创建一个Integer对象
public static Integer valueOf(int i)	根据传递的整数创建一个Integer对象
public static Integer valueOf(String s)	根据传递的字符串创建一个Integer对象
public static Integer valueOf(String s, int radix)	根据传递的字符串和进制创建一个Integer对象

```
//3.这两种方式获取对象的区别（掌握）|           ↗  
//因为在实际开发中，-128~127之间的数据，用的比较多。  
//如果每次使用都是new对象，那么太浪费内存了  
//所以，提前把这个范围之内的每一个数据都创建好对象  
//如果要用到了不会创建新的，而是返回已经创建好的对象。  
Integer i6 = Integer.valueOf(127);  
Integer i7 = Integer.valueOf(127);  
System.out.println(i6 == i7); //true  
  
Integer i8 = Integer.valueOf(128);  
Integer i9 = Integer.valueOf(128);  
System.out.println(i8 == i9); //false  
  
//在JDK5的时候提出了一个机制：自动装箱和自动拆箱  
//自动装箱：把基本数据类型会自动的变成其对应的包装类  
//自动拆箱：把包装类自动的变成其对象的基本数据类型  
  
//在底层，此时还会去自动调用静态方法valueOf得到一个Integer对象，只不过这个动作不需要我们自己去操作了  
//自动装箱的动作  
Integer i1 = 10;  
  
Integer i2 = new Integer( value: 10 );  
//自动拆箱的动作  
int i = i2;  
  
//在JDK5以后，int和Integer可以看做是同一个东西，因为在内部可以自动转化。|
```

1. 什么是包装类？

基本数据类型对应的对象

2. JDK5以后对包装类新增了什么特性？

自动装箱、自动拆箱

3. 我们以后如何获取包装类对象？

不需要new，不需要调用方法，直接赋值即可

```
Integer i = 10;  
  
Integer i1 = 10;  
Integer i2 = 10;  
Integer i3 = i1 + i2;
```

- **Integer成员方法**

方法名	说明
public static String toBinaryString(int i)	得到二进制
public static String toOctalString(int i)	得到八进制
public static String toHexString(int i)	得到十六进制
public static int parseInt(String s)	将字符串类型的整数转成int类型的整数

- 代码实现

```
//1.把整数转成二进制，十六进制
String str1 = Integer.toBinaryString( i: 100);
System.out.println(str1); //1100100

//2.把整数转成八进制
String str2 = Integer.toOctalString( i: 100);
System.out.println(str2); //144

//3.把整数转成十六进制
String str3 = Integer.toHexString( i: 100);
System.out.println(str3); //64

//4.将字符串类型的整数转成int类型的整数
//强类型语言：每种数据在java中都有各自的数据类型
//在计算的时候，如果不是同一种数据类型，是无法直接计算的。
```

```
int i = Integer.parseInt( s: "123");
System.out.println(i);

//细节1：
//在类型转换的时候，括号中的参数只能是数字不能是其他，否则代码会报错
//细节2：
//8种包装类当中，除了Character都有对应的parseXXX的方法，进行类型转换
String str = "true";
boolean b = Boolean.parseBoolean(str);
System.out.println(b);
```

- 键盘录入改进（无论什么类型都用nextLine输入，之后再用包装类强转）

```
//键盘录入
Scanner sc = new Scanner(System.in);
System.out.println("请输入一个字符串");
/* String str = sc.next();
System.out.println(str);*/
//弊端:
//当我们在使用next, nextInt, nextDouble在接收数据的时候, 遇到空格, 回车, 制表符的时候就停止了
//键盘录入的是123 123 那么此时只能接收到空格前面的数据
//我想要的是接收一整行数据
//约定:
//以后我们如果想要键盘录入, 不管什么类型, 统一使用nextLine
//特点遇到回车才停止
String line = sc.nextLine();      ↴
System.out.println(line);
```