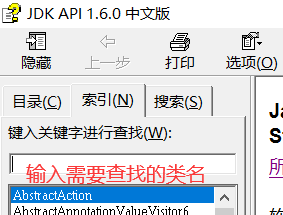
# day11

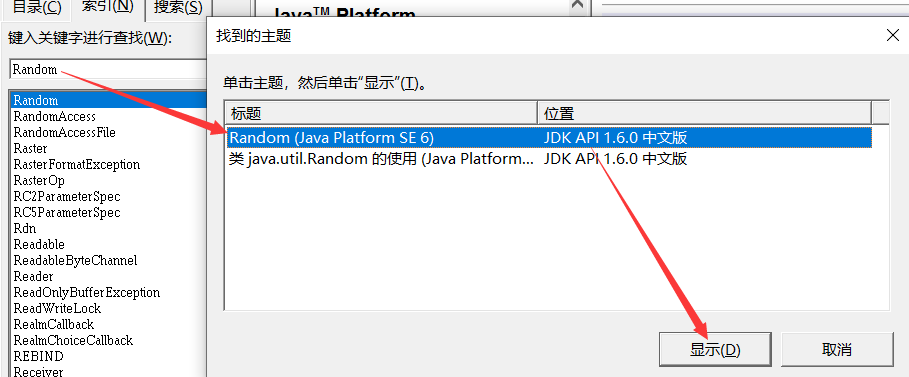
## API和类库

1. API : Application Programming Interface, 应用程序接口. API是JDK给每一个开发人员提供的使用JDK中核心类库内容操作文档, 开发人员可以通过参考API学习JDK的核心类库内容(每一个JDK版本都对应有指定版本API帮助文档)
2. API的使用步骤:

双击API帮助文档:





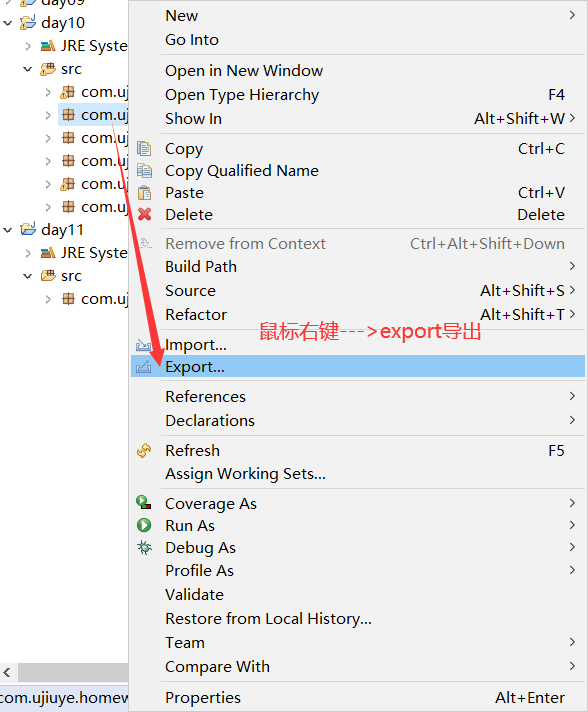


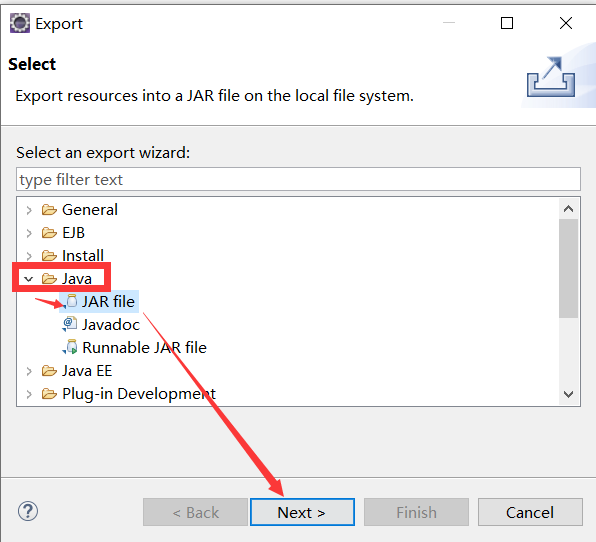
1. 类库 :
2. Java源文件是.java为结尾的文件, 源文件需要通过编译成.class字节码文件, 这个字节码文件中包含了源文件中所有功能和内容, JVM虚拟机最终运行就是.class
3. 实际开发中, 需要交付给客户还是给同事, 提供的都是.class文件, 如何提供? 需要将类打包成jar包, jar包就是Java代码的标准压缩包

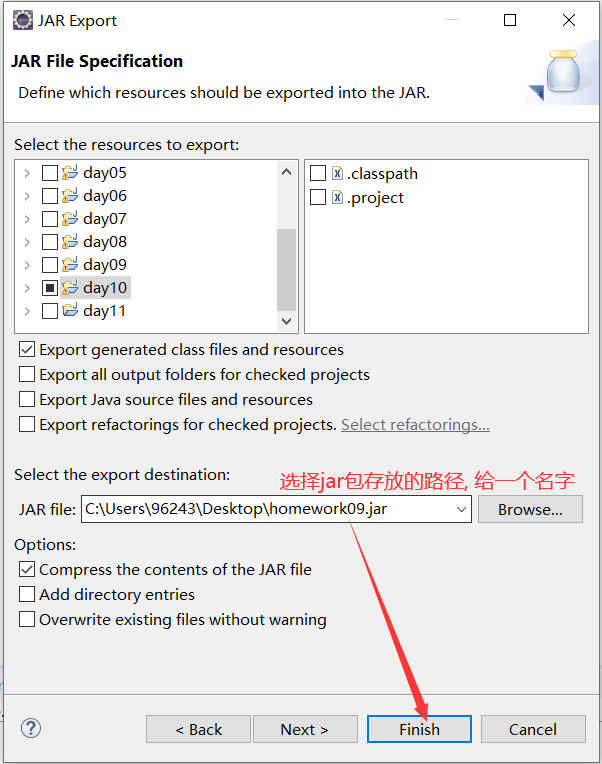
在项目工程中, JRE System Libary 中包含JDK提供核心类库中的类型, 这些类型都是以jar包为标准压缩包存在, 压缩包中包含的就是很多的.class字节码文件

1. 将自己代码打包成标准的jar包(导出jar包):

选中需要打成压缩包的package包, 或者是项目工程



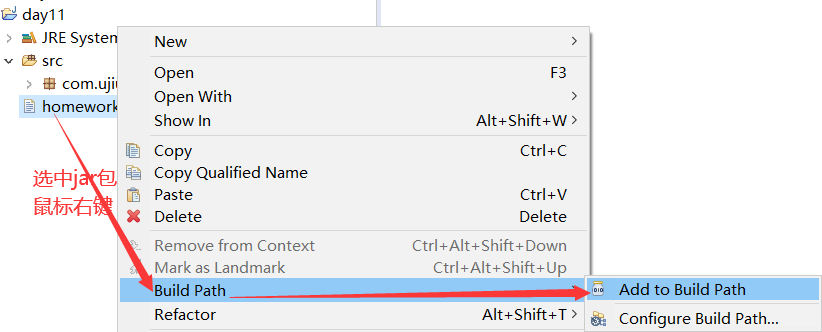




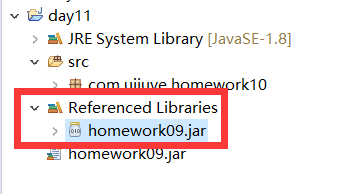
1. 如何将jar包导入到项目中使用(导入jar包):

a : 将目标jar包进行复制, 粘贴到当前需要使用的项目工程下

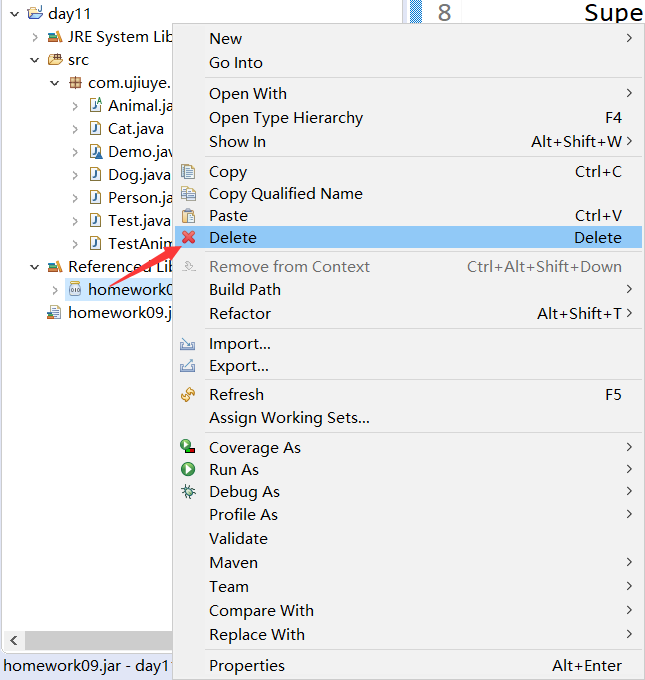
b :



导入成功后的效果, jar包前面有一个奶瓶标志:

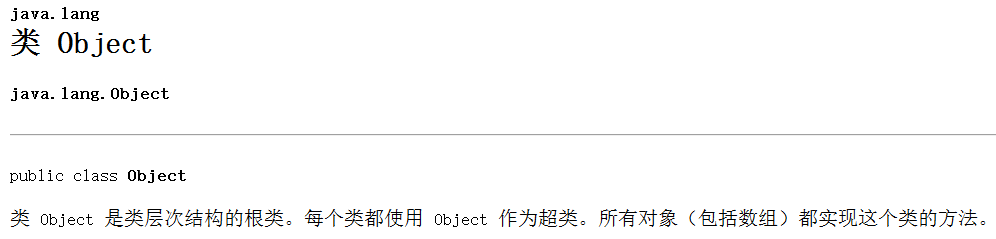


1. 删除无用的jar包:



## Object类

1. Object 类: 来自于java.lang包



1. 如果一个类没有显示的定义出其父类, 那么默认让Object类成为这个类的父类

class A{}// Object 类就是A类的父类

class B extends A{} // B类根据多层继承特性, Object间接是B类的父类

1. Object类型构造方法:
2. Object类中具有一个空参数构造方法, 但是因为Object类已经没有父类, 因此构造方法第一行没有super()
3. 因为Object是所有类的父类, 因此Object类其实最先进入到内存中

### toString方法



1. Object类中toString方法源代码实现过程

public String toString() {

带有包路径的类全名 分隔符 将对象在内存中地址转换成16进制数表示

return getClass().getName() + "@" + Integer.toHexString(hashCode());

}

1. 标准输出语句 : System.out.println(), 当输出的是一个引用数据类型时, println方法自动调用当前对象中的toString方法结果
2. 因为查看和打印一个对象的地址值, 一般来说对代码无用, 建议Object类的子类重写toString方法功能, 通过alt + shift + s , 选择toString方法重写, 重写之后的效果就是打印出的是当前对象中成员变量的数值结果

代码

|  |
| --- |
| package com.ujiuye.object;  public class Person {// 完整表现形式 : public class Person extends Object  private String name;  private int age;  @Override  public String toString() {  return "Person [name=" + name + ", age=" + age + "]";  }  } |

|  |
| --- |
| package com.ujiuye.object;  public class TestObject {  public static void main(String[] args) {  Person p = new Person();  String result = p.toString();  /\*public String toString() {  带有包路径的类的全名 分隔符 将对象在内存中地址转换成16进制数表示  return getClass().getName() + "@" + Integer.toHexString(hashCode());  }\*/  System.out.println(result);//com.ujiuye.object.Person@15db9742    System.out.println(p+"---");//com.ujiuye.object.Person@15db9742---    Object obj = new Object();  }  } |

### equals方法

1. equals(Object obj) : 验证两个对象是否相等, 两个对象分别为 : 方法的调用者对象, 方法的参数对象, 返回值类型boolean
2. equals方法源代码: 比较的是两个对象的内存地址是否相等

public boolean equals(Object obj) {

// this : 表示方法的调用者对象

return (this == obj);

}

1. 实际开发中, equals方法功能, 使用比较多, 实际场景中,通常不比较内存地址, 比较两个对象中成员变量的值是否相等, 于是需要子类重写equals方法功能, alt + shift + s, 自动生成即可

举例 : 录入咱们班级学生信息, 限定 : 如果学生姓名和学号都一致, 认为这个学生信息重复了, 不会多次录入

代码

|  |
| --- |
| package com.ujiuye.object;  public class Person {// 完整表现形式 : public class Person extends Object  private String name;  private int age;  @Override  public boolean equals(Object obj) {  if (this == obj)  return true;  if (obj == null)  return false;  if (getClass() != obj.getClass())  return false;  Person other = (Person) obj;  if (age != other.age)  return false;  if (name == null) {  if (other.name != null)  return false;  } else if (!name.equals(other.name))  return false;  return true;  }  public Person(String name, int age) {  super();  this.name = name;  this.age = age;  }  public Person() {  super();  }  } |

|  |
| --- |
| package com.ujiuye.object;  public class TestObject {  public static void main(String[] args) {  // 1. toString  Person p = new Person();  String result = p.toString();  /\*public String toString() {  带有包路径的类的全名 分隔符 将对象在内存中地址转换成16进制数表示  return getClass().getName() + "@" + Integer.toHexString(hashCode());  }\*/  System.out.println(result);//com.ujiuye.object.Person@15db9742    System.out.println(p+"---");//com.ujiuye.object.Person@15db9742---    // 2. equals  Person p1 = new Person("张三",20);  Person p2 = new Person("张三",20);  System.out.println(p1.equals(p2));// 源代码中比较两个对象内存地址 , false  }  } |

### ==和equals方法的比较

1. == : 比较运算符

a : 可以比较基本数据类型, 比较两个数据值是否相等

b : 可以比较引用数据类型, 比较两个引用类型的地址是否相同

1. equals : 是来自于Object类中方法功能

a : equals方法只能比较引用数据类型, 不能进行基本数据类型的比较

b : equals比较引用数据类型

1. Object源代码中, 比较两个对象内存地址是否相等
2. 子类重写了equals方法, 比较两个对象成员变量的值是否相等

## Scanner类

1. Scanner类 : 来自于java.util包, 本身功能就是可以扫描基本数据类型和字符串(可以接收从键盘录入的基本数据类型和字符串)
2. Scanner类型构造方法 :

Scanner(System.in) : System.in表示标准输入流资源, 默认通过键盘录入数据

1. Scanner类中方法功能:
2. nextByte() : 接收通过键盘录入的一个字节数据
3. nextShort()
4. nextInt()
5. nextLong()
6. nextFloat()
7. nextDouble()
8. nextBoolean()

Scanner类型中, 没有提供nextChar()接收键盘录入一个字符的功能, 可以使用键盘录入字符串替代nextChar功能

1. Scanner键盘录入字符串
2. next() : 表示接收通过键盘录入的一个字符串数据, 返回值类型就是输入的字符串结果, next方法功能可以通过空格进行多个数据之间区分(回车换行也可以作为一个输入方法结束标志)
3. nextLine() : 表示接收通过键盘录入的一个字符串数据, 返回值类型就是输入的字符串结果, nextLine() 方法功能只能通过回车换行进行多个数据之间的切分

1. 注意 : nextLine()方法如果读取到了回车换行, 那么认为这次数据输入结束; 如果在nextLine()方法之前, 调用next或者next基本数据类型 这些方法, 上述方法输入结束时的回车换行, 就会被nextLine接收并且作为数据内容, 导致效果上nextLine少输入一次

注意的解决方案 : 1) 多调用一次nextLine方法, 用于吸收前面方法多余的回车换行

2)创建出一个新的Scanner类型对象, 用新对象调用nextLine

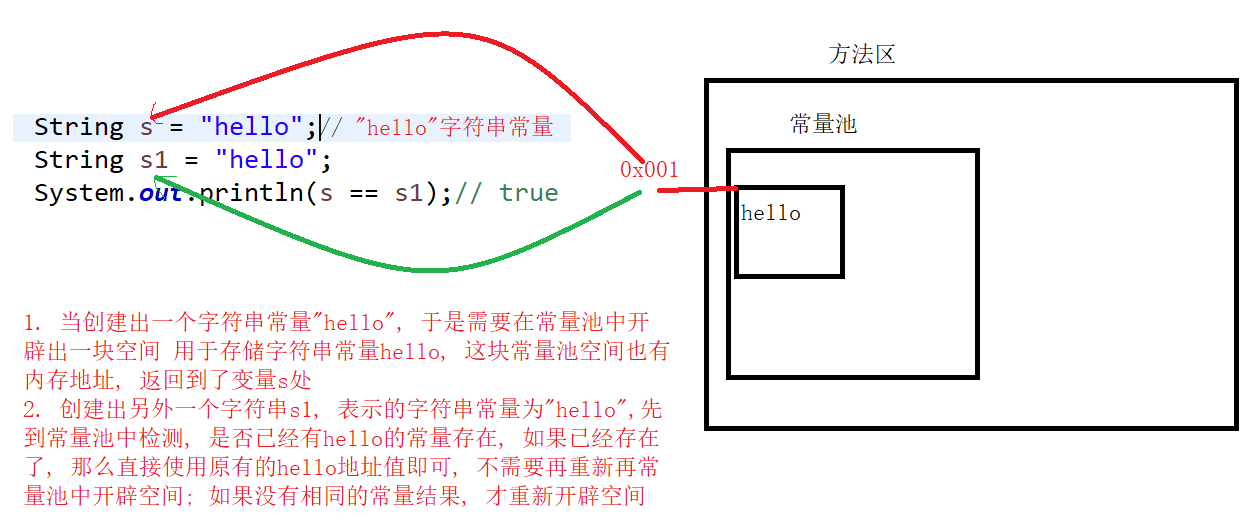
代码

|  |
| --- |
| package com.ujiuye.object;  import java.util.Scanner;  public class ScannerDemo {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  // 1. 测试通过键盘录入基本数据类型  /\*System.out.println("请输入一个字节数据, 范围-128---127");  byte b = sc.nextByte();  System.out.println(b);    System.out.println("请输入一个double类型数据:");  double d = sc.nextDouble();  System.out.println(d + "----");\*/    // 2. 测试通过键盘录入的字符串  System.out.println("请输入字符串:");  // 1)next方法功能可以通过空格进行多个数据之间区分(回车换行也可以作为一个输入方法结束标志)  String first = sc.next();  String second = sc.next();  //System.out.println(first + "---" + second);    // admin 123  System.out.println("请输入一个用户名:");    //String name1 = sc.nextLine();  Scanner sc1 = new Scanner(System.in);  // 2) nextLine() 方法功能只能通过回车换行进行多个数据之间的切分  String name = sc1.nextLine();  System.out.println("请输入一个密码:");  String password = sc1.nextLine();  System.out.println(name + "---" + password);  }  } |

## String类

### String类的概述

1. String类 : 来自于java.lang包, 表示字符串
2. 在Java代码中, 所有字符串常量, “123”, “hello”..., 都是一个字符串类型的对象(实例)
3. 字符串常量, 定义之后其值不能修改; String类型代码底层存储一个字符串, 是将这个字符串中的每一个字符放置到一个char类型数组中, 这个数组定义使用 private final char[] value; 并且在String类型中, 没有提供对外的可以访问私有成员value的方式, 因此String类型所表示的字符串, 称为常量, 因为字符串值无法修改
4. 字符串常量存储在常量池中(常量池在方法区中) : 相同的字符串常量指向同一块常量池内存空间



代码

|  |
| --- |
| package com.ujiuye.string;  public class Demo01\_String {  public static void main(String[] args) {  // 1. 所有字符串常量, “123”, “hello”..., 都是一个字符串类型的对象  String s = "hello";  String s1 = "hello";  System.out.println(s == s1);// true    String s2 = "hell";  System.out.println(s1 == s2);// false    String s3 = "Hello";  System.out.println(s3 == s1);// false  }  } |

### String类构造方法

1. String() : 创建出一个空字符串, “”
2. String(String str) : 创建出一个字符串, 字符串中的字符内容与参数str一致
3. String(byte[] b) : 将字节数组转换成一个字符串; 字节数据参考平台默认编码表, 将字节转换成对应的字符, 拼接字符形成一个字符串
4. String(byte[] b, int beginIndex, int length): 将字节数组的一部分转换成字符串

a : beginIndex表示从字节数组的某一个索引位置开始

b : length 表示需要截取的长度(个数)

1. String(char[] ch) : 将字符数组转换成一个字符串; 直接将参数ch字符数组中的每一个字符获取到, 拼接成一个字符串
2. String(char[] ch, int beginIndex, int length): 将字符数组的一部分转换成字符串;

a : beginIndex表示从字节数组的某一个索引位置开始

b : length 表示需要截取的长度(个数)

代码

|  |
| --- |
| package com.ujiuye.string;  public class Demo02\_String构造 {  public static void main(String[] args) {  // 1. String()  String s = new String();// "" String s1 = "";  System.out.println(s + "---");    // 2. String(String str)  // s1通过new创建对象, s1指向堆内存空间地址  String s1 = new String("123");  System.out.println(s1);  // s2指向常量池中内存地址, s1和s2内容上相同, 但是地址值不同的  String s2 = "123";  System.out.println(s1 == s2);// false    // 3.String(byte[] b) : 将字节数组转换成一个字符串; 将字节数据参考平台默认编码表, 将字节转换成对应的字符, 拼接字符形成一个字符串  byte[] b = {97,98,99,100}; // abcd  String s3 = new String(b);  System.out.println(s3);// abcd    /\*4.String(byte[] b, int beginIndex, int length): 将字节数组的一部分转换成字符串  a : beginIndex表示从字节数组的某一个索引位置开始  b : length 表示需要截取的长度(个数) \*/  String s4 = new String(b,1,2);  System.out.println(s4);// bc    // 5.String(char[] ch) : 将字符数组转换成一个字符串; 直接将参数ch字符数组中的每一个字符获取到, 拼接成一个字符串  char[] ch = {'1','A','>','夹','z'};  String s5 = new String(ch);  System.out.println(s5);//1A>夹z    /\*6.String(char[] ch, int beginIndex, int length): 将字符数组的一部分转换成字符串;  a : beginIndex表示从字节数组的某一个索引位置开始  b : length 表示需要截取的长度(个数)\*/  String s6 = new String(ch,0,3);  System.out.println(s6);// 1A>  }  } |

### String类型中判断功能

String类型中有很多方法用于验证字符串是否符合某种规则, 方法的返回值类型结果都是boolean类型

1. equals(Object obj) : 是从Object类中继承到的方法, String类重写, 使用方法调用者字符串与参数obj所表示的字符串进行内容的比较, 如果内容相同, 返回true, 否则返回false
2. [equalsIgnoreCase](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/lang/../../java/lang/String.html" \l "equalsIgnoreCase(java.lang.String))([String](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/lang/../../java/lang/String.html" \o "java.lang 中的类) str) : 让方法调用字符串与参数str进行内容比较, 忽略掉大小写之后, 内容如果相同, 返回true, 否则返回false(只有字母分大小写)
3. startsWith(String str) : 验证方法调用字符串是否以参数字符串str为前缀开头(str可以从前往后匹配方法调用字符串), 如果是前缀返回true, 否则返回false
4. endsWith(String str) : 验证方法调用字符串是否以参数字符串str为后缀(str可以从后往前匹配方法调用字符串), 如果是后缀返回true, 否则返回false
5. contains(String s) : 验证方法调用字符串是否完全包含参数字符串s, 完全包含返回true, 否则返回false
6. isEmpty() : 验证方法调用字符串是否是一个空字符串, “”, 如果是空字符串返回true. 否则返回false, 在使用一个字符串之前, 可以利用isEmpty验证字符串中是否包含字符内容

代码

|  |
| --- |
| package com.ujiuye.string;  public class Demo03\_String判断功能 {  public static void main(String[] args) {  // 1.equals(Object obj) : 是从Object类中继承到的方法, String类重写, 使用方法调用者字符串与参数obj所表示的字符串进行内容的比较, 如果内容相同, 返回true, 否则返回false  String s = "123";  String s1 = new String("123");  System.out.println(s == s1);// false  System.out.println(s.equals(s1));// true    String s2 = "Hello";  String s3 = "hello";  System.out.println(s2.equals(s3));// false    // 2.equalsIgnoreCase(String str):让方法调用字符串与参数str进行内容比较, 忽略掉大小写之后, 内容如果相同, 返回true, 否则返回false  System.out.println(s2.equalsIgnoreCase(s3));// true  System.out.println("一".equalsIgnoreCase("壹"));// false    // 3.startsWith(String str) : 验证方法调用字符串是否以参数字符串str为前缀开头(str可以从前往后匹配方法调用字符串),  // 如果是前缀返回true, 否则返回false  String name = "张三";  // 验证一个人的姓氏  System.out.println(name.startsWith("张三"));// true    // 4.endsWith(String str) : 验证方法调用字符串是否以参数字符串str为后缀(str可以从后往前匹配方法调用字符串), 如果是后缀返回true, 否则返回false  String file = "abc.txt";  System.out.println(file.endsWith(".txt"));//true    // 5.contains(String s) : 验证方法调用字符串是否完全包含参数字符串s, 完全包含返回true, 否则返回false  String s4 = "helloworld";  System.out.println(s4.contains("hello"));// true  System.out.println(s4.contains("llw"));// false    // 6.isEmpty() : 验证方法调用字符串是否是一个空字符串, “”, 如果是空字符串返回true. 否则返回false, 在使用一个字符串之前, 可以利用isEmpty验证字符串中是否包含字符内容  System.out.println("".isEmpty());// true  System.out.println(" ".isEmpty());//false, " "1个空格也算作是一个字符  }  } |

### String类型获取功能

String获取功能就是想要去获取到字符串中的数据

1. length() : 返回字符串中字符的个数(返回字符串长度), 返回值类型int类型; 如果字符串长度为0, 证明isEmpty结果为true
2. charAt(int index) : 将字符串中指定index索引位置对应的字符获取到, 返回值char类型
3. substring(int beginIndex) : 将方法调用字符串从beginIndex索引位置开始, 截取到剩下的全部, 返回截取出的新字符串
4. substring(int beginIndex, int endIndex) : 将方法调用字符串从beginIndex索引位置开始,截取到endIndex-1结束, 中间截取到的字符串作为方法的返回值结果

注意 : JDK提供方法功能中, 如果是同时在操作两个索引位, 通常包含起始索引而不包含结束索引

1. indexOf(String str): 获取参数字符串str在方法调用字符串中第一次出现的索引位置, 返回是int类型索引值结果; 如果没有找到目标的字符串str, 返回-1
2. indexOf(String str, int beginIndex) : 从beginIndex索引位置开始向后寻找, str第一次出现在方法调用字符串中的索引位置, 如果没有找到目标的字符串str, 返回-1

代码

|  |
| --- |
| package com.ujiuye.string;  public class Demo04\_String获取功能 {  public static void main(String[] args) {  //1.length() : 返回字符串中字符的个数(返回字符串长度), 返回值类型int类型  System.out.println("".length());// 0  System.out.println(" 1".length());// 3  // 2.charAt(int index) : 将字符串中指定index索引位置对应的字符获取到, 返回值char类型  String s = "helloabc";  char ch = s.charAt(3);  System.out.println(ch);// l    for(int index = 0; index < s.length(); index++) {  System.out.print(s.charAt(index) + " ");  }    System.out.println();    //3.substring(int beginIndex) : 将方法调用字符串从beginIndex索引位置开始, 截取到剩下的全部, 返回截取出的新字符串  String s1 = s.substring(5);  System.out.println(s1);//abc    // 4.substring(int beginIndex, int endIndex) : 将方法调用字符串从beginIndex索引位置开始,截取到endIndex-1结束, 中间截取到的字符串作为方法的返回值结果  // 注意 : JDK提供方法功能中, 如果是同时在操作两个索引位, 通常包含起始索引而不包含结束索引  String s2 = s.substring(1,6);//elloa  System.out.println(s2);    // 5.indexOf(String str): 获取参数字符串str在方法调用字符串中第一次出现的索引位置, 返回是int类型索引值结果; 如果没有找到目标的字符串str, 返回-1  String name = "海盗船长";  System.out.println(name.indexOf("123"));// -1  System.out.println(name.indexOf("船长"));// 2    // 6.indexOf(String str, int beginIndex) : 从beginIndex索引位置开始向后寻找, str第一次出现在方法调用字符串中的索引位置  System.out.println(name.indexOf("船长", 1));// 2  }  } |

### String类型转换功能

1. getBytes() : 将方法调用字符串转换成字符, 再通过平台默认编码表将字符转换成字节数据, 将转换结果存储在byte[]中
2. toCharArray() : 将方法调用字符串转换成字符数组, 将转换结果存储在char[]中
3. toUpperCase() : 将方法调用字符串中的所有字符(字母)转换成大写, 返回转换之后的新字符串结果
4. toLowerCase() : 将方法调用字符串中的所有字符(字母)转换成小写, 返回转换之后的新字符串结果
5. static valueOf(Object obj) : 可以将其他任意数据类型转换成字符串类型, 方法本身是重载方法, 也可以将基本数据类型转换成字符串类型

代码

|  |
| --- |
| package com.ujiuye.string;  import java.util.Arrays;  public class Demo05\_String转换功能 {  public static void main(String[] args) {    String s = "ABCDEF";  // 1.getBytes() : 将方法调用字符串转换成字符, 再通过平台默认编码表将字符转换成字节数据, 将转换结果存储在byte[]中  byte[] b = s.getBytes();  System.out.println(Arrays.toString(b));//[65, 66, 67, 68, 69, 70]    // 2.toCharArray() : 将方法调用字符串转换成字符数组, 将转换结果存储在char[]中  char[] ch = s.toCharArray();  System.out.println(Arrays.toString(ch));    for(int index = 0; index < ch.length; index++) {  System.out.print(ch[index] + " ");  }  System.out.println();    // 3.toUpperCase() : 将方法调用字符串中的所有字符(字母)转换成大写, 返回转换之后的新字符串结果  String s1 = "HelloWorld";  System.out.println(s1.toUpperCase());// HELLOWORLD    // 4. toLowerCase() : 将方法调用字符串中的所有字符(字母)转换成小写, 返回转换之后的新字符串结果  System.out.println(s1.toLowerCase());// helloworld    // 5.static valueOf(Object obj) : 可以将其他任意数据类型转换成字符串类型, 方法本身是重载方法, 也可以将基本数据类型转换成字符串类型  String result = String.valueOf(10) + 1;  System.out.println(result);// 101    String ss = String.valueOf(new Object());  System.out.println(ss);  }  } |

### 其他方法功能

1. replace(String old, String newStr): 将字符串中所有的old字符串内容替换成newStr字符串内容, 将替换之后的字符串结果作为方法的返回值
2. trim() : 将字符串中前后的空格去除掉, 字符串中空格无法去除, 将去掉空格的字符串结果作为方法的返回值

代码

|  |
| --- |
| package com.ujiuye.string;  public class Demo06\_String其他功能 {  public static void main(String[] args) {  // 1.replace(String old, String newStr): 将字符串中所有的old字符串内容替换成newStr字符串内容, 将替换之后的字符串结果作为方法的返回值  String s = "helloworld";  String newStr = s.replace("l", "\*");  System.out.println(newStr);//he\*\*owor\*d    // 2.trim() : 将字符串中前后的空格去除掉, 字符串中空格无法去除, 将去掉空格的字符串结果作为方法的返回值  String s1 = " 123 abc ";  System.out.println(s1.trim());// 123 abc  }  } |