第 15 章泛型

15. 1 泛型的理解和好处

15.1.1 看一个需求



|  |
| --- |
| package com.hspedu.generic;  import java.util.ArrayList;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class Generic01 {  public static void main(String[] args) { |

|  |
| --- |
| //使用传统的方法来解决  ArrayList arrayList = new ArrayList();  arrayList.add(new Dog("旺财", 10));  arrayList.add(new Dog("发财", 1));  arrayList.add(new Dog("小黄", 5));  //假如我们的程序员，不小心，添加了一只猫  arrayList.add(new Cat("招财猫", 8));  //遍历  for (Object o : arrayList) {  //向下转型 Object ->Dog  Dog dog = (Dog) o;  System.out.println(dog.getName() + "-" + dog.getAge());  }  }  }  /\*  请编写程序，在 ArrayList 中，添加 3 个 Dog 对象  Dog 对象含有 name 和 age, 并输出 name 和 age (要求使用 getXxx())  \*/  class Dog {  private String name;  private int age;  public Dog(String name, int age) { |

|  |
| --- |
| this.name = name;  this.age = age;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  }  class Cat { //Cat 类  private String name;  private int age;  public Cat(String name, int age) {  this.name = name; |

|  |
| --- |
| this.age = age;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  } |

15.1.2 使用传统方法的问题分析



15.1.3 泛型快速体验-用泛型来解决前面的问题



|  |
| --- |
| package com.hspedu.generic.improve;  import java.util.ArrayList;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class Generic02 {  public static void main(String[] args) {  //使用传统的方法来解决===> 使用泛型  //老韩解读  //1. 当我们 ArrayList<Dog> 表示存放到 ArrayList 集合中的元素是 Dog 类型 (细节后面说...)  //2. 如果编译器发现添加的类型，不满足要求，就会报错  //3. 在遍历的时候，可以直接取出 Dog 类型而不是 Object  //4. public class ArrayList<E> {} E 称为泛型,那么 Dog->E  ArrayList<Dog> arrayList = new ArrayList<Dog>();  arrayList.add(new Dog("旺财", 10));  arrayList.add(new Dog("发财", 1));  arrayList.add(new Dog("小黄", 5));  //假如我们的程序员，不小心，添加了一只猫 |

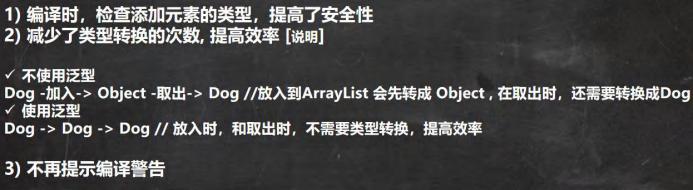
|  |
| --- |
| //arrayList.add(new Cat("招财猫", 8));  System.out.println("===使用泛型====");  for (Dog dog : arrayList) {  System.out.println(dog.getName() + "-" + dog.getAge());  }  }  }  /\*  1.请编写程序，在 ArrayList 中，添加 3 个 Dog 对象  2.Dog 对象含有 name 和 age, 并输出 name 和 age (要求使用 getXxx())  3.老师使用泛型来完成代码  \*/  class Dog {  private String name;  private int age;  public Dog(String name, int age) {  this.name = name;  this.age = age;  }  public String getName() {  return name;  } |

|  |
| --- |
| public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  }  class Cat { //Cat 类  private String name;  private int age;  public Cat(String name, int age) {  this.name = name;  this.age = age;  }  public String getName() {  return name;  } |

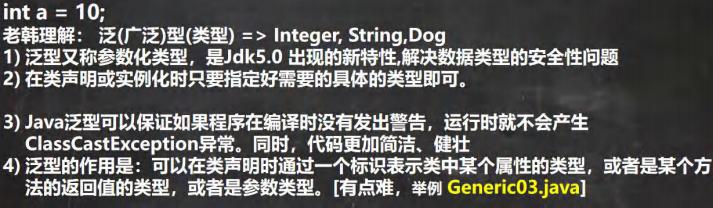
|  |
| --- |
| public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  } |

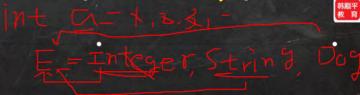
15.2 泛型的理解和好处

15.2.1 泛型的好处



15.3 泛型介绍





|  |
| --- |
| package com.hspedu.generic;  import java.util.List;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class Generic03 {  public static void main(String[] args) {  //注意，特别强调： E 具体的数据类型在定义 Person对象的时候指定, 即在编译期间，就确定 E 是什么类型  Person<String> person = new Person<String>("韩顺平教育");  person.show(); //String  /\* |

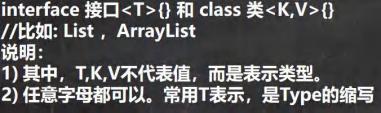
|  |
| --- |
| 你可以这样理解，上面的 Person类  class Person {  String s ;//E 表示 s 的数据类型, 该数据类型在定义 Person 对象的时候指定, 即在编译期间，就确定 E  是什么类型  public Person(String s) {//E 也可以是参数类型  this.s = s;  }  public String f() {//返回类型使用 E  return s;  }  }  \*/  Person<Integer> person2 = new Person<Integer>(100);  person2.show();//Integer  /\*  class Person {  Integer s ;//E 表示 s 的数据类型, 该数据类型在定义 Person 对象的时候指定, 即在编译期间，就确定 E  是什么类型  public Person(Integer s) {//E 也可以是参数类型  this.s = s;  } |

|  |
| --- |
| public Integer f() {//返回类型使用 E  return s;  }  }  \*/  }  }  //泛型的作用是：可以在类声明时通过一个标识表示类中某个属性的类型，  // 或者是某个方法的返回值的类型，或者是参数类型  class Person<E> {  E s ;//E 表示 s 的数据类型, 该数据类型在定义 Person对象的时候指定, 即在编译期间，就确定 E 是什么类型  public Person(E s) {//E 也可以是参数类型  this.s = s;  }  public E f() {//返回类型使用 E  return s;  }  public void show() {  System.out.println(s.getClass());//显示 s 的运行类型  } |

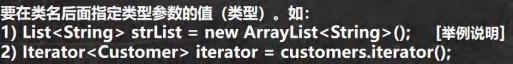
|  |
| --- |
| } |

15.4 泛型的语法

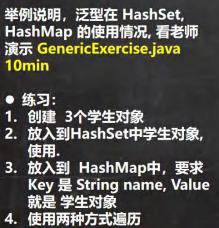
15.4.1 泛型的声明



15.4.2 泛型的实例化



15.4.3 泛型使用举例



|  |
| --- |
| package com.hspedu.generic;  import java.util.\*; |

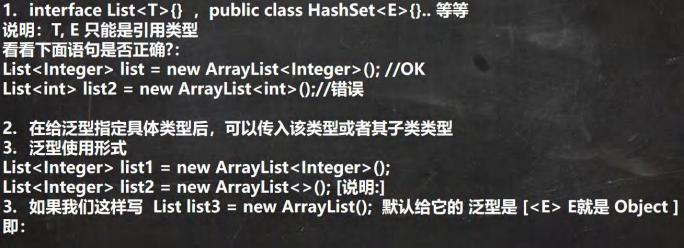
|  |
| --- |
| /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class GenericExercise {  public static void main(String[] args) {  //使用泛型方式给 HashSet 放入 3 个学生对象  HashSet<Student> students = new HashSet<Student>();  students.add(new Student("jack", 18));  students.add(new Student("tom", 28));  students.add(new Student("mary", 19));  //遍历  for (Student student : students) {  System.out.println(student);  }  //使用泛型方式给 HashMap 放入 3 个学生对象  //K -> String V->Student  HashMap<String, Student> hm = new HashMap<String, Student>();  /\*  public class HashMap<K,V> {}  \*/  hm.put("milan", new Student("milan", 38));  hm.put("smith", new Student("smith", 48)); |

|  |
| --- |
| hm.put("hsp", new Student("hsp", 28));  //迭代器 EntrySet  /\*  public Set<Map.Entry<K,V>> entrySet() {  Set<Map.Entry<K,V>> es;  return (es = entrySet) == null ? (entrySet = new EntrySet()) : es;  }  \*/  Set<Map.Entry<String, Student>> entries = hm.entrySet();  /\*  public final Iterator<Map.Entry<K,V>> iterator() {  return new EntryIterator();  }  \*/  Iterator<Map.Entry<String, Student>> iterator = entries.iterator();  System.out.println("==============================");  while (iterator.hasNext()) {  Map.Entry<String, Student> next = iterator.next();  System.out.println(next.getKey() + "-" + next.getValue());  }  }  }  /\*\* |

|  |
| --- |
| \* 创建 3 个学生对象  \* 放入到 HashSet 中学生对象, 使用.  \* 放入到 HashMap 中，要求 Key 是 String name, Value 就是 学生对象  \* 使用两种方式遍历  \*/  class Student {  private String name;  private int age;  public Student(String name, int age) {  this.name = name;  this.age = age;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  } |

|  |
| --- |
| public void setAge(int age) {  this.age = age;  }  @Override  public String toString() {  return "Student{" +  "name='" + name + '\'' +  ", age=" + age +  '}';  }  } |

15.4.4 泛型使用的注意事项和细节 GenericDetail.java



|  |
| --- |
| package com.hspedu.generic;  import java.util.ArrayList; |

|  |
| --- |
| import java.util.List;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class GenericDetail {  public static void main(String[] args) {  //1.给泛型指向数据类型是，要求是引用类型，不能是基本数据类型  List<Integer> list = new ArrayList<Integer>(); //OK  //List<int> list2 = new ArrayList<int>();//错误  //2. 说明  //因为 E 指定了 A 类型, 构造器传入了 new A()  //在给泛型指定具体类型后，可以传入该类型或者其子类类型  Pig<A> aPig = new Pig<A>(new A());  aPig.f();  Pig<A> aPig2 = new Pig<A>(new B());  aPig2.f();  //3. 泛型的使用形式  ArrayList<Integer> list1 = new ArrayList<Integer>();  List<Integer> list2 = new ArrayList<Integer>();  //在实际开发中，我们往往简写  //编译器会进行类型推断, 老师推荐使用下面写法 |

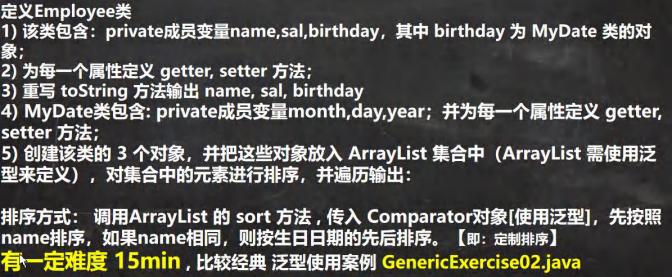
|  |
| --- |
| ArrayList<Integer> list3 = new ArrayList<>();  List<Integer> list4 = new ArrayList<>();  ArrayList<Pig> pigs = new ArrayList<>();  //4. 如果是这样写 泛型默认是 Object  ArrayList arrayList = new ArrayList();//等价 ArrayList<Object> arrayList = new ArrayList<Object>();  /\*  public boolean add(Object e) {  ensureCapacityInternal(size + 1); // Increments modCount!!  elementData[size++] = e;  return true;  }  \*/  Tiger tiger = new Tiger();  /\*  class Tiger {//类  Object e;  public Tiger() {}  public Tiger(Object e) {  this.e = e;  }  } |

|  |
| --- |
| \*/  }  }  class Tiger<E> {//类  E e;  public Tiger() {}  public Tiger(E e) {  this.e = e;  }  }  class A {}  class B extends A {}  class Pig<E> {//  E e;  public Pig(E e) {  this.e = e;  }  public void f() { |

|  |
| --- |
| System.out.println(e.getClass()); //运行类型  }  } |

15.5 泛型课堂类型

15.5.1 泛型课堂练习题



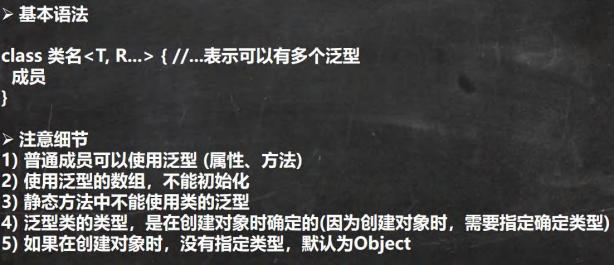
|  |
| --- |
| package com.hspedu.generic;  import java.util.ArrayList;  import java.util.Comparator;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class GenericExercise02 { |

|  |
| --- |
| public static void main(String[] args) {  ArrayList<Employee> employees = new ArrayList<>();  employees.add(new Employee("tom", 20000, new MyDate(1980, 12, 11)));  employees.add(new Employee("jack", 12000, new MyDate(2001, 12, 12)));  employees.add(new Employee("tom", 50000, new MyDate(1980, 12, 10)));  System.out.println("employees=" + employees);  employees.sort(new Comparator<Employee>() {  @Override  public int compare(Employee emp1, Employee emp2) {  //先按照 name 排序，如果 name 相同，则按生日日期的先后排序。【即：定制排序】  //先对传入的参数进行验证  if(!(emp1 instanceof Employee && emp2 instanceof Employee)) {  System.out.println("类型不正确..");  return 0;  }  //比较 name  int i = emp1.getName().compareTo(emp2.getName());  if(i != 0) {  return i;  }  //下面是对 birthday 的比较，因此，我们最好把这个比较，放在 MyDate 类完成 |

|  |
| --- |
| //封装后，将来可维护性和复用性，就大大增强.  return emp1.getBirthday().compareTo(emp2.getBirthday());  }  });  System.out.println("==对雇员进行排序==");  System.out.println(employees);  }  }  /\*\*  \* 定义 Employee 类  \* 1) 该类包含：private 成员变量 name,sal,birthday ，其中 birthday 为 MyDate 类的对象；  \* 2) 为每一个属性定义 getter, setter 方法；  \* 3) 重写 toString 方法输出 name, sal, birthday  \* 4) MyDate 类包含: private 成员变量 month,day,year；并为每一个属性定义 getter, setter 方法；  \* 5) 创建该类的 3 个对象，并把这些对象放入 ArrayList 集合中（ArrayList 需使用泛型来定义），对集合中的元素进  行排序，并遍历输出：  \*  \* 排序方式： 调用 ArrayList 的 sort 方法 ,  \* 传入 Comparator对象[使用泛型]，先按照 name 排序，如果 name 相同，则按生日日期的先后排序。【即：定制排序】  \* 有一定难度 15min , 比较经典 泛型使用案例 GenericExercise02.java  \*/ |

15.6 自定义泛型

15.6.1 自定义泛型类 (难度)





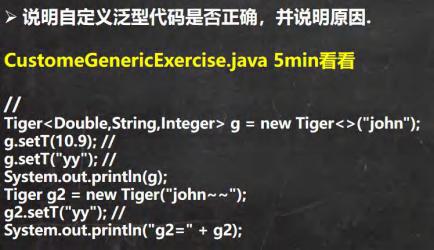
|  |
| --- |
| package com.hspedu.customgeneric;  import java.util.Arrays;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"}) |

|  |
| --- |
| public class CustomGeneric\_ {  public static void main(String[] args) {  //T=Double, R=String, M=Integer  Tiger<Double,String,Integer> g = new Tiger<>("john");  g.setT(10.9); //OK  //g.setT("yy"); //错误，类型不对  System.out.println(g);  Tiger g2 = new Tiger("john~~");//OK T=Object R=Object M=Object  g2.setT("yy"); //OK , 因为 T=Object "yy"=String 是 Object 子类  System.out.println("g2=" + g2);  }  }  //老韩解读  //1. Tiger 后面泛型，所以我们把 Tiger 就称为自定义泛型类  //2, T, R, M 泛型的标识符, 一般是单个大写字母  //3. 泛型标识符可以有多个.  //4. 普通成员可以使用泛型 (属性、方法)  //5. 使用泛型的数组，不能初始化  //6. 静态方法中不能使用类的泛型  class Tiger<T, R, M> {  String name;  R r; //属性使用到泛型  M m; |

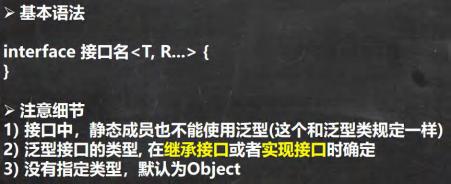
|  |
| --- |
| T t;  //因为数组在 new 不能确定 T 的类型，就无法在内存开空间  T[] ts;  public Tiger(String name) {  this.name = name;  }  public Tiger(R r, M m, T t) {//构造器使用泛型  this.r = r;  this.m = m;  this.t = t;  }  public Tiger(String name, R r, M m, T t) {//构造器使用泛型  this.name = name;  this.r = r;  this.m = m;  this.t = t;  }  //因为静态是和类相关的，在类加载时，对象还没有创建  //所以，如果静态方法和静态属性使用了泛型，JVM 就无法完成初始化  // static R r2;  // public static void m1(M m) { |

|  |
| --- |
| //  // }  //方法使用泛型  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public R getR() {  return r;  }  public void setR(R r) {//方法使用到泛型  this.r = r;  }  public M getM() {//返回类型可以使用泛型.  return m;  }  public void setM(M m) { |

|  |
| --- |
| this.m = m;  }  public T getT() {  return t;  }  public void setT(T t) {  this.t = t;  }  @Override  public String toString() {  return "Tiger{" +  "name='" + name + '\'' +  ", r=" + r +  ", m=" + m +  ", t=" + t +  ", ts=" + Arrays.toString(ts) +  '}';  }  } |



15.6.2 自定义泛型接口





|  |
| --- |
| package com.hspedu.customgeneric;  /\*\*  \* @author 韩顺平 |

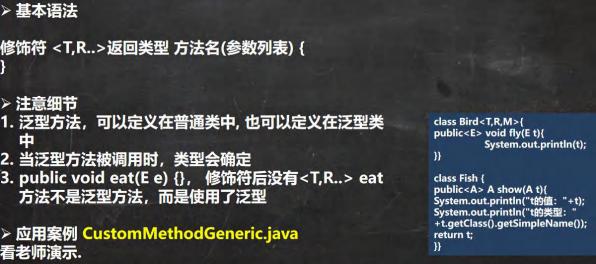
|  |
| --- |
| \* @version 1.0  \*/  public class CustomInterfaceGeneric {  public static void main(String[] args) {  }  }  /\*\*  \* 泛型接口使用的说明  \* 1. 接口中，静态成员也不能使用泛型  \* 2. 泛型接口的类型, 在继承接口或者实现接口时确定  \* 3. 没有指定类型，默认为 Object  \*/  //在继承接口 指定泛型接口的类型  interface IA extends IUsb<String, Double> {  }  //当我们去实现 IA 接口时，因为 IA 在继承 IUsu 接口时，指定了 U 为 String R 为 Double  // ，在实现 IUsu 接口的方法时，使用 String 替换 U, 是 Double 替换 R  class AA implements IA {  @Override  public Double get(String s) {  return null; |

|  |
| --- |
| }  @Override  public void hi(Double aDouble) {  }  @Override  public void run(Double r1, Double r2, String u1, String u2) {  }  }  //实现接口时，直接指定泛型接口的类型  //给 U 指定 Integer 给 R 指定了 Float  //所以，当我们实现 IUsb 方法时，会使用 Integer 替换 U, 使用 Float 替换 R  class BB implements IUsb<Integer, Float> {  @Override  public Float get(Integer integer) {  return null;  }  @Override  public void hi(Float aFloat) {  } |

|  |
| --- |
| @Override  public void run(Float r1, Float r2, Integer u1, Integer u2) {  }  }  //没有指定类型，默认为 Object  //建议直接写成 IUsb<Object,Object>  class CC implements IUsb { //等价 class CC implements IUsb<Object,Object> {  @Override  public Object get(Object o) {  return null;  }  @Override  public void hi(Object o) {  }  @Override  public void run(Object r1, Object r2, Object u1, Object u2) {  }  }  interface IUsb<U, R> {  int n = 10;  //U name; 不能这样使用 |

|  |
| --- |
| //普通方法中，可以使用接口泛型  R get(U u);  void hi(R r);  void run(R r1, R r2, U u1, U u2);  //在jdk8 中，可以在接口中，使用默认方法, 也是可以使用泛型  default R method(U u) {  return null;  }  } |

15.6.3 自定义泛型方法



|  |
| --- |
| package com.hspedu.customgeneric;  import java.util.ArrayList; |

|  |
| --- |
| /\*\*  \* @author 韩顺平  \* @version 1.0  \* 泛型方法的使用  \*/  @SuppressWarnings({"all"})  public class CustomMethodGeneric {  public static void main(String[] args) {  Car car = new Car();  car.fly("宝马", 100);//当调用方法时，传入参数，编译器，就会确定类型  System.out.println("=======");  car.fly(300, 100. 1);//当调用方法时，传入参数，编译器，就会确定类型  //测试  //T->String, R-> ArrayList  Fish<String, ArrayList> fish = new Fish<>();  fish.hello(new ArrayList(), 11.3f);  }  }  //泛型方法，可以定义在普通类中, 也可以定义在泛型类中  class Car {//普通类  public void run() {//普通方法  } |

|  |
| --- |
| //说明 泛型方法  //1. <T,R> 就是泛型  //2. 是提供给 fly 使用的  public <T, R> void fly(T t, R r) {//泛型方法  System.out.println(t.getClass());//String  System.out.println(r.getClass());//Integer  }  }  class Fish<T, R> {//泛型类  public void run() {//普通方法  }  public<U,M> void eat(U u, M m) {//泛型方法  }  //说明  //1. 下面 hi 方法不是泛型方法  //2. 是 hi 方法使用了类声明的 泛型  public void hi(T t) {  }  //泛型方法，可以使用类声明的泛型，也可以使用自己声明泛型  public<K> void hello(R r, K k) {  System.out.println(r.getClass());//ArrayList  System.out.println(k.getClass());//Float  } |

|  |
| --- |
| } |

15.6.4 自定义泛型方法

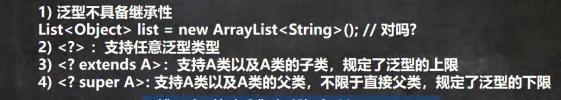


|  |
| --- |
| package com.hspedu.customgeneric;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class CustomMethodGenericExercise {  public static void main(String[] args) {  //T->String, R->Integer, M->Double  Apple<String, Integer, Double> apple = new Apple<>();  apple.fly(10);//10 会被自动装箱 Integer10, 输出 Integer  apple.fly(new Dog());//Dog |

|  |
| --- |
| }  }  class Apple<T, R, M> {//自定义泛型类  public <E> void fly(E e) { //泛型方法  System.out.println(e.getClass().getSimpleName());  }  //public void eat(U u) {}//错误，因为 U 没有声明  public void run(M m) {  } //ok  }  class Dog {  } |

15.7 泛型的继承和通配符

15.7.1 泛型的继承和通配符说明 GenericExtends.java



15.7.2 应用案例

|  |
| --- |
| package com.hspedu; |

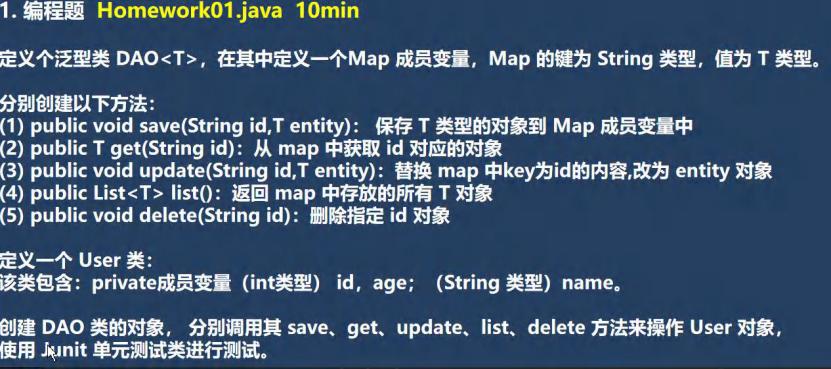
|  |
| --- |
| import java.util.ArrayList;  import java.util.List;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 泛型的继承和通配符  \*/  public class GenericExtends {  public static void main(String[] args) {  Object o = new String("xx");  //泛型没有继承性  //List<Object> list = new ArrayList<String>();  //举例说明下面三个方法的使用  List<Object> list1 = new ArrayList<>();  List<String> list2 = new ArrayList<>();  List<AA> list3 = new ArrayList<>();  List<BB> list4 = new ArrayList<>();  List<CC> list5 = new ArrayList<>();  //如果是 List<?> c ，可以接受任意的泛型类型  printCollection1(list1);  printCollection1(list2); |

|  |
| --- |
| printCollection1(list3);  printCollection1(list4);  printCollection1(list5);  //List<? extends AA> c： 表示 上限，可以接受 AA 或者 AA 子类  // printCollection2(list1);//×  // printCollection2(list2);//×  printCollection2(list3);//√  printCollection2(list4);//√  printCollection2(list5);//√  //List<? super AA> c: 支持 AA 类以及 AA 类的父类，不限于直接父类  printCollection3(list1);//√  //printCollection3(list2);//×  printCollection3(list3);//√  //printCollection3(list4);//×  //printCollection3(list5);//×  //冒泡排序  //插入排序  //.... |

|  |
| --- |
| }  // ? extends AA 表示 上限，可以接受 AA 或者 AA 子类  public static void printCollection2(List<? extends AA> c) {  for (Object object : c) {  System.out.println(object);  }  }  //说明: List<?> 表示 任意的泛型类型都可以接受  public static void printCollection1(List<?> c) {  for (Object object : c) { // 通配符，取出时，就是 Object  System.out.println(object);  }  }  // ? super 子类类名 AA:支持 AA 类以及 AA 类的父类，不限于直接父类，  //规定了泛型的下限  public static void printCollection3(List<? super AA> c) {  for (Object object : c) {  System.out.println(object);  }  }  } |

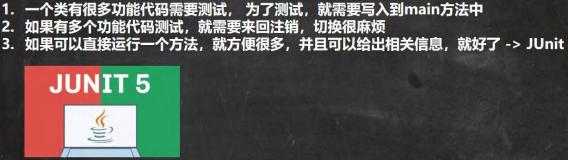
|  |
| --- |
| class AA {  }  class BB extends AA {  }  class CC extends BB {  } |

15.8 本章作业



15.9 JUnit

15.9.1 为什么需要 JUnit



15.9.2 基本介绍



15.9.3 使用步骤,看老师演示 JUnit\_.java

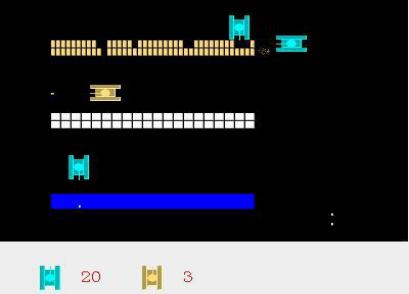
|  |
| --- |
| package com.hspedu.junit\_;  import org.junit.jupiter.api.Test;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class JUnit\_ {  public static void main(String[] args) {  //传统方式  //new JUnit\_().m1();  //new JUnit\_().m2();  } |

|  |
| --- |
| @Test  public void m1() {  System.out.println("m1 方法被调用");  }  @Test  public void m2() {  System.out.println("m2 方法被调用");  }  @Test  public void m3() {  System.out.println("m3 方法被调用");  }  } |

第 16 章坦克大战[1]

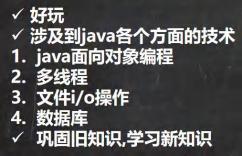
16. 1 坦克大战游戏演示

16.1.1 游戏演示

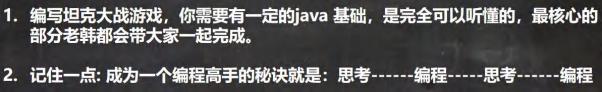


16.1.2 文档坦克游戏说明文档.doc

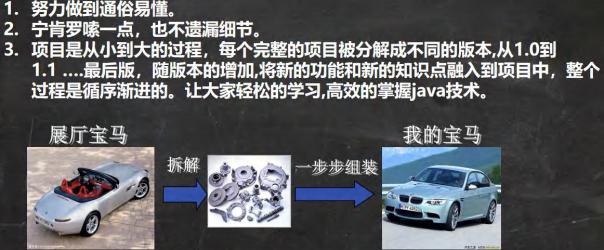
16.1.3 为什么写这个项目



16.1.4 写项目前的提醒

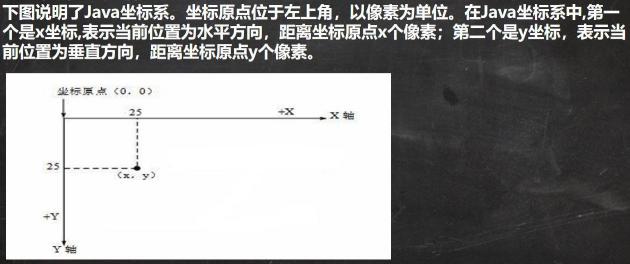


16.1.5 如何讲解这个项目,授课的原则:

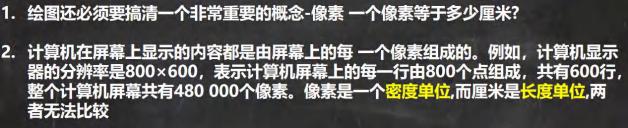


16.2 java 绘图坐标体系

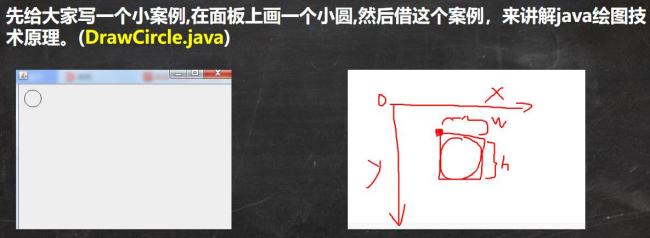
16.2.1 坐标体系-介绍



16.2.2 坐标体系-像素



16.2.3 介绍-快速入门



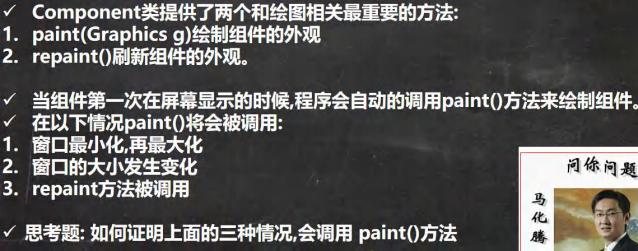
|  |
| --- |
| package com.hspedu.draw;  import javax.swing.\*;  import java.awt.\*;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示如何在面板上画出圆形  \*/  @SuppressWarnings({"all"})  public class DrawCircle extends JFrame { //JFrame 对应窗口,可以理解成是一个画框 |

|  |
| --- |
| //定义一个面板  private MyPanel mp = null;  public static void main(String[] args) {  new DrawCircle();  System.out.println("退出程序~");  }  public DrawCircle() {//构造器  //初始化面板  mp = new MyPanel();  //把面板放入到窗口(画框)  this.add(mp);  //设置窗口的大小  this.setSize(400, 300);  //当点击窗口的小× , 程序完全退出.  this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  this.setVisible(true);//可以显示  }  }  //1.先定义一个 MyPanel, 继承 JPanel 类， 画图形，就在面板上画  class MyPanel extends JPanel { |

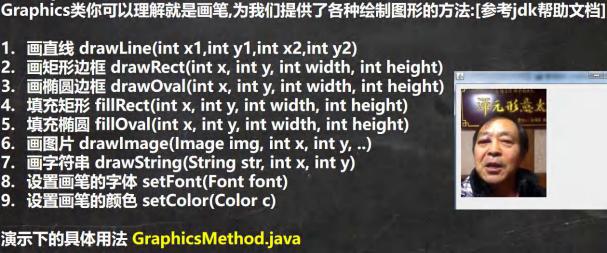
|  |
| --- |
| //说明:  //1. MyPanel 对象就是一个画板  //2. Graphics g 把 g 理解成一支画笔  //3. Graphics 提供了很多绘图的方法  //Graphics g  @Override  public void paint(Graphics g) {//绘图方法  super.paint(g);//调用父类的方法完成初始化.  System.out.println("paint 方法被调用了~");  //画出一个圆形.  //g.drawOval(10, 10, 100, 100);  //演示绘制不同的图形..  //画直线 drawLine(int x1,int y1,int x2,int y2)  //g.drawLine(10, 10, 100, 100);  //画矩形边框 drawRect(int x, int y, int width, int height)  //g.drawRect(10, 10, 100, 100);  //画椭圆边框 drawOval(int x, int y, int width, int height)  //填充矩形 fillRect(int x, int y, int width, int height)  //设置画笔的颜色  // g.setColor(Color.blue);  // g.fillRect(10, 10, 100, 100);  //填充椭圆 fillOval(int x, int y, int width, int height)  // g.setColor(Color.red); |

|  |
| --- |
| // g.fillOval(10, 10, 100, 100);  //画图片 drawImage(Image img, int x, int y, ..)  //1. 获取图片资源, /bg.png 表示在该项目的根目录去获取 bg.png 图片资源  // Image image = Toolkit.getDefaultToolkit().getImage(Panel.class.getResource("/bg.png"));  // g.drawImage(image, 10, 10, 175, 221, this);  //画字符串 drawString(String str, int x, int y)//写字  //给画笔设置颜色和字体  g.setColor(Color.red);  g.setFont(new Font("隶书", Font.BOLD, 50));  //这里设置的 100 ， 100 ， 是 "北京你好"左下角  g.drawString("北京你好", 100, 100);  //设置画笔的字体 setFont(Font font)  //设置画笔的颜色 setColor(Color c)  }  } |

16.2.4 绘图原理



16.2.5 Graphics 类



|  |
| --- |
| package com.hspedu.draw;  import javax.swing.\*;  import java.awt.\*;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示如何在面板上画出圆形  \*/  @SuppressWarnings({"all"})  public class DrawCircle extends JFrame { //JFrame 对应窗口,可以理解成是一个画框  //定义一个面板  private MyPanel mp = null; |

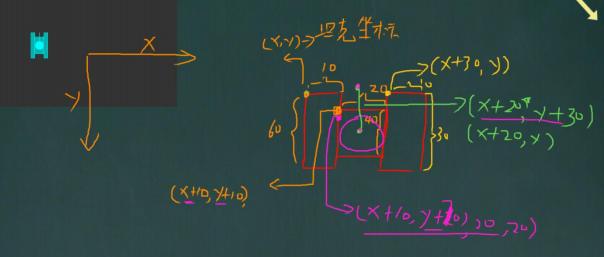
|  |
| --- |
| public static void main(String[] args) {  new DrawCircle();  System.out.println("退出程序~");  }  public DrawCircle() {//构造器  //初始化面板  mp = new MyPanel();  //把面板放入到窗口(画框)  this.add(mp);  //设置窗口的大小  this.setSize(400, 300);  //当点击窗口的小× , 程序完全退出.  this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  this.setVisible(true);//可以显示  }  }  //1.先定义一个 MyPanel, 继承 JPanel 类， 画图形，就在面板上画  class MyPanel extends JPanel {  //说明:  //1. MyPanel 对象就是一个画板  //2. Graphics g 把 g 理解成一支画笔  //3. Graphics 提供了很多绘图的方法 |

|  |
| --- |
| //Graphics g  @Override  public void paint(Graphics g) {//绘图方法  super.paint(g);//调用父类的方法完成初始化.  System.out.println("paint 方法被调用了~");  //画出一个圆形.  //g.drawOval(10, 10, 100, 100);  //演示绘制不同的图形..  //画直线 drawLine(int x1,int y1,int x2,int y2)  //g.drawLine(10, 10, 100, 100);  //画矩形边框 drawRect(int x, int y, int width, int height)  //g.drawRect(10, 10, 100, 100);  //画椭圆边框 drawOval(int x, int y, int width, int height)  //填充矩形 fillRect(int x, int y, int width, int height)  //设置画笔的颜色  // g.setColor(Color.blue);  // g.fillRect(10, 10, 100, 100);  //填充椭圆 fillOval(int x, int y, int width, int height)  // g.setColor(Color.red);  // g.fillOval(10, 10, 100, 100);  //画图片 drawImage(Image img, int x, int y, ..)  //1. 获取图片资源, /bg.png 表示在该项目的根目录去获取 bg.png 图片资源 |

|  |
| --- |
| // Image image = Toolkit.getDefaultToolkit().getImage(Panel.class.getResource("/bg.png"));  // g.drawImage(image, 10, 10, 175, 221, this);  //画字符串 drawString(String str, int x, int y)//写字  //给画笔设置颜色和字体  g.setColor(Color.red);  g.setFont(new Font("隶书", Font.BOLD, 50));  //这里设置的 100 ， 100 ， 是 "北京你好"左下角  g.drawString("北京你好", 100, 100);  //设置画笔的字体 setFont(Font font)  //设置画笔的颜色 setColor(Color c)  }  } |

16.2.6 绘出坦克





|  |
| --- |
| package com.hspedu.tankgame;  import javax.swing.\*;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class HspTankGame01 extends JFrame {  //定义 MyPanel  MyPanel mp = null;  public static void main(String[] args) {  HspTankGame01 hspTankGame01 = new HspTankGame01();  } |

|  |
| --- |
| public HspTankGame01() {  mp = new MyPanel();  this.add(mp);//把面板(就是游戏的绘图区域)  this.setSize(1000, 750);  this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  this.setVisible(true);  }  } |
| package com.hspedu.tankgame;  import javax.swing.\*;  import java.awt.\*;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 坦克大战的绘图区域  \*/  public class MyPanel extends JPanel {  //定义我的坦克  Hero hero = null;  public MyPanel() {  hero = new Hero(100, 100);//初始化自己坦克  } |

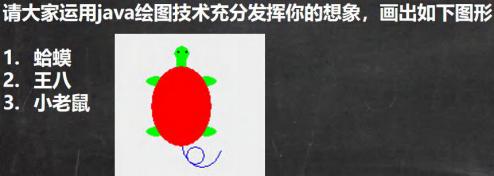
|  |
| --- |
| @Override  public void paint(Graphics g) {  super.paint(g);  g.fillRect(0, 0, 1000, 750);//填充矩形，默认黑色  //画出坦克-封装方法  drawTank(hero.getX(), hero.getY(), g, 0, 0);  }  //编写方法，画出坦克  /\*\*  \* @param x 坦克的左上角 x 坐标  \* @param y 坦克的左上角 y 坐标  \* @param g 画笔  \* @param direct 坦克方向（上下左右）  \* @param type 坦克类型  \*/  public void drawTank(int x, int y, Graphics g, int direct, int type) {  //根据不同类型坦克，设置不同颜色  switch (type) {  case 0: //我们的坦克  g.setColor(Color.cyan); |

|  |
| --- |
| break;  case 1: //敌人的坦克  g.setColor(Color.yellow);  break;  }  //根据坦克方向，来绘制坦克  switch (direct) {  case 0: //表示向上  g.fill3DRect(x, y, 10, 60, false);//画出坦克左边轮子  g.fill3DRect(x + 30, y, 10, 60, false);//画出坦克右边轮子  g.fill3DRect(x + 10, y + 10, 20, 40, false);//画出坦克盖子  g.fillOval(x + 10, y + 20, 20, 20);//画出圆形盖子  g.drawLine(x + 20, y + 30, x + 20, y);//画出炮筒  break;  default:  System.out.println("暂时没有处理");  }  }  } |
| package com.hspedu.tankgame;  /\*\*  \* @author 韩顺平 |

|  |
| --- |
| \* @version 1.0  \* 自己的坦克  \*/  public class Hero extends Tank {  public Hero(int x, int y) {  super(x, y);  }  } |
| package com.hspedu.tankgame;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class Tank {  private int x;//坦克的横坐标  private int y;//坦克的纵坐标  public Tank(int x, int y) {  this.x = x;  this.y = y;  }  public int getX() {  return x; |

|  |
| --- |
| }  public void setX(int x) {  this.x = x;  }  public int getY() {  return y;  }  public void setY(int y) {  this.y = y;  }  } |

16.2.7 绘图练习



16.3 java 事件处理机制

16.3.1 事件处理机制-看个问题



|  |
| --- |
| package com.hspedu.event\_;  import javax.swing.\*;  import java.awt.\*;  import java.awt.event.KeyEvent;  import java.awt.event.KeyListener;  import java.awt.event.MouseListener;  import java.awt.event.WindowListener;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示小球通过键盘控制上下左右的移动-> 讲解 Java 的事件控制  \*/  public class BallMove extends JFrame { //窗口  MyPanel mp = null;  public static void main(String[] args) {  BallMove ballMove = new BallMove();  } |

|  |
| --- |
| //构造器  public BallMove() {  mp = new MyPanel();  this.add(mp);  this.setSize(400, 300);  //窗口 JFrame 对象可以监听键盘事件, 即可以监听到面板发生的键盘事件  this.addKeyListener(mp);  this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  this.setVisible(true);  }  }  //面板, 可以画出小球  //KeyListener 是监听器, 可以监听键盘事件  class MyPanel extends JPanel implements KeyListener {  //为了让小球可以移动, 把他的左上角的坐标(x,y)设置变量  int x = 10;  int y = 10;  @Override  public void paint(Graphics g) {  super.paint(g);  g.fillOval(x, y, 20, 20); //默认黑色  }  //有字符输出时，该方法就会触发 |

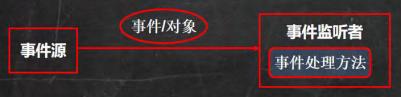
|  |
| --- |
| @Override  public void keyTyped(KeyEvent e) {  }  //当某个键按下，该方法会触发  @Override  public void keyPressed(KeyEvent e) {  //System.out.println((char)e.getKeyCode() + "被按下..");  //根据用户按下的不同键，来处理小球的移动 (上下左右的键)  //在java 中，会给每一个键，分配一个值(int)  if(e.getKeyCode() == KeyEvent.VK\_DOWN) {//KeyEvent.VK\_DOWN 就是向下的箭头对应的 code  y++;  } else if(e.getKeyCode() == KeyEvent.VK\_UP) {  y--;  } else if(e.getKeyCode() == KeyEvent.VK\_LEFT) {  x--;  } else if(e.getKeyCode() == KeyEvent.VK\_RIGHT) {  x++;  }  //让面板重绘  this.repaint();  }  //当某个键释放(松开) ，该方法会触发 |

|  |
| --- |
| @Override  public void keyReleased(KeyEvent e) {  }  } |

16.3.2 基本说明



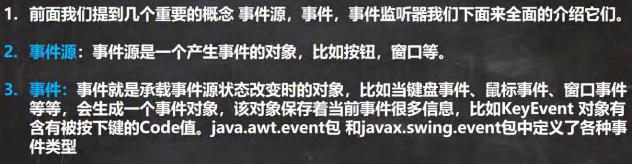
16.3.3 请大家看一个示意图



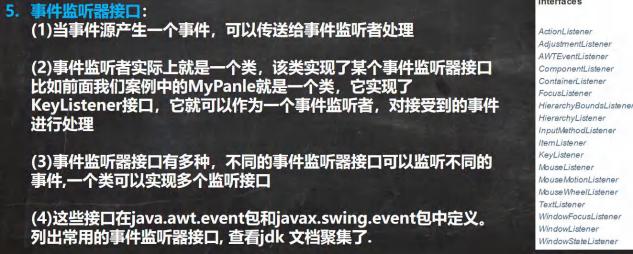
16.3.4 机制分析



16.3.5 事件处理机制深入理解



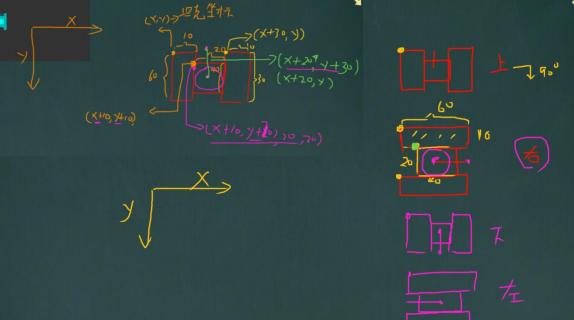




16.4 坦克大战游戏

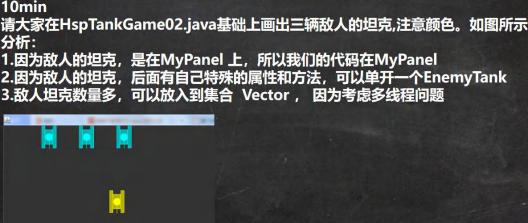
16.4.1 让你的坦克动起来





16.5 本章作业

16.5.1 本章作业



|  |
| --- |
| package com.hspedu.tankgame2; |

|  |
| --- |
| import javax.swing.\*;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class HspTankGame02 extends JFrame {  //定义 MyPanel  MyPanel mp = null;  public static void main(String[] args) {  HspTankGame02 hspTankGame01 = new HspTankGame02();  }  public HspTankGame02() {  mp = new MyPanel();  this.add(mp);//把面板(就是游戏的绘图区域)  this.setSize(1000, 750);  this.addKeyListener(mp);//让 JFrame 监听 mp 的键盘事件  this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  this.setVisible(true);  }  } |

|  |
| --- |
| package com.hspedu.tankgame2;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 敌人的坦克  \*/  public class EnemyTank extends Tank {  public EnemyTank(int x, int y) {  super(x, y);  }  } |
| package com.hspedu.tankgame2;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 自己的坦克  \*/  public class Hero extends Tank {  public Hero(int x, int y) {  super(x, y);  }  } |

|  |
| --- |
| package com.hspedu.tankgame2;  import javax.swing.\*;  import java.awt.\*;  import java.awt.event.KeyEvent;  import java.awt.event.KeyListener;  import java.util.Vector;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 坦克大战的绘图区域  \*/  //为了监听 键盘事件， 实现 KeyListener  public class MyPanel extends JPanel implements KeyListener {  //定义我的坦克  Hero hero = null;  //定义敌人坦克，放入到 Vector  Vector<EnemyTank> enemyTanks = new Vector<>();  int enemyTankSize = 3;  public MyPanel() {  hero = new Hero(100, 100);//初始化自己坦克  //初始化敌人坦克  for (int i = 0; i < enemyTankSize; i++) { |

|  |
| --- |
| //创建一个敌人的坦克  EnemyTank enemyTank = new EnemyTank((100 \* (i + 1)), 0);  //设置方向  enemyTank.setDirect(2);  //加入  enemyTanks.add(enemyTank);  }  }  @Override  public void paint(Graphics g) {  super.paint(g);  g.fillRect(0, 0, 1000, 750);//填充矩形，默认黑色  //画出自己坦克-封装方法  drawTank(hero.getX(), hero.getY(), g, hero.getDirect(), 1);  //画出敌人的坦克, 遍历 Vector  for (int i = 0; i < enemyTanks.size(); i++) {  //取出坦克  EnemyTank enemyTank = enemyTanks.get(i);  drawTank(enemyTank.getX(), enemyTank.getY(), g, enemyTank.getDirect(), 0);  }  } |

|  |
| --- |
| //编写方法，画出坦克  /\*\*  \* @param x 坦克的左上角 x 坐标  \* @param y 坦克的左上角 y 坐标  \* @param g 画笔  \* @param direct 坦克方向（上下左右）  \* @param type 坦克类型  \*/  public void drawTank(int x, int y, Graphics g, int direct, int type) {  //根据不同类型坦克，设置不同颜色  switch (type) {  case 0: //敌人的坦克  g.setColor(Color.cyan);  break;  case 1: //我的坦克  g.setColor(Color.yellow);  break;  }  //根据坦克方向，来绘制对应形状坦克  //direct 表示方向(0: 向上 1 向右 2 向下 3 向左 )  //  switch (direct) { |

|  |
| --- |
| case 0: //表示向上  g.fill3DRect(x, y, 10, 60, false);//画出坦克左边轮子  g.fill3DRect(x + 30, y, 10, 60, false);//画出坦克右边轮子  g.fill3DRect(x + 10, y + 10, 20, 40, false);//画出坦克盖子  g.fillOval(x + 10, y + 20, 20, 20);//画出圆形盖子  g.drawLine(x + 20, y + 30, x + 20, y);//画出炮筒  break;  case 1: //表示向右  g.fill3DRect(x, y, 60, 10, false);//画出坦克上边轮子  g.fill3DRect(x, y + 30, 60, 10, false);//画出坦克下边轮子  g.fill3DRect(x + 10, y + 10, 40, 20, false);//画出坦克盖子  g.fillOval(x + 20, y + 10, 20, 20);//画出圆形盖子  g.drawLine(x + 30, y + 20, x + 60, y + 20);//画出炮筒  break;  case 2: //表示向下  g.fill3DRect(x, y, 10, 60, false);//画出坦克左边轮子  g.fill3DRect(x + 30, y, 10, 60, false);//画出坦克右边轮子  g.fill3DRect(x + 10, y + 10, 20, 40, false);//画出坦克盖子  g.fillOval(x + 10, y + 20, 20, 20);//画出圆形盖子  g.drawLine(x + 20, y + 30, x + 20, y + 60);//画出炮筒  break;  case 3: //表示向左  g.fill3DRect(x, y, 60, 10, false);//画出坦克上边轮子  g.fill3DRect(x, y + 30, 60, 10, false);//画出坦克下边轮子  g.fill3DRect(x + 10, y + 10, 40, 20, false);//画出坦克盖子  g.fillOval(x + 20, y + 10, 20, 20);//画出圆形盖子 |

|  |
| --- |
| g.drawLine(x + 30, y + 20, x, y + 20);//画出炮筒  break;  default:  System.out.println("暂时没有处理");  }  }  @Override  public void keyTyped(KeyEvent e) {  }  //处理 wdsa 键按下的情况  @Override  public void keyPressed(KeyEvent e) {  System.out.println(e.getKeyCode());  if (e.getKeyCode() == KeyEvent.VK\_W) {//按下 W 键  //改变坦克的方向  hero.setDirect(0);//  //修改坦克的坐标 y -= 1  hero.moveUp();  } else if (e.getKeyCode() == KeyEvent.VK\_D) {//D 键, 向右  hero.setDirect(1);  hero.moveRight(); |

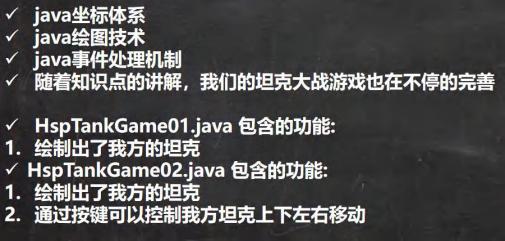
|  |
| --- |
| } else if (e.getKeyCode() == KeyEvent.VK\_S) {//S 键  hero.setDirect(2);  hero.moveDown();  } else if (e.getKeyCode() == KeyEvent.VK\_A) {//A 键  hero.setDirect(3);  hero.moveLeft();  }  //让面板重绘  this.repaint();  }  @Override  public void keyReleased(KeyEvent e) {  }  } |
| package com.hspedu.tankgame2;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class Tank {  private int x;//坦克的横坐标 |

|  |
| --- |
| private int y;//坦克的纵坐标  private int direct = 0;//坦克方向 0 上 1 右 2 下 3 左  private int speed = 1;  public int getSpeed() {  return speed;  }  public void setSpeed(int speed) {  this.speed = speed;  }  //上右下左移动方法  public void moveUp() {  y -= speed;  }  public void moveRight() {  x += speed;  }  public void moveDown() {  y += speed;  }  public void moveLeft() {  x -= speed;  } |

|  |
| --- |
| public int getDirect() {  return direct;  }  public void setDirect(int direct) {  this.direct = direct;  }  public Tank(int x, int y) {  this.x = x;  this.y = y;  }  public int getX() {  return x;  }  public void setX(int x) {  this.x = x;  }  public int getY() {  return y;  }  public void setY(int y) { |

|  |
| --- |
| this.y = y;  }  } |

16.5.2 本章内容小结



第 17 章多线程基础

17. 1 线程相关概念

17.1.1 程序(program)



17.1.2 进程

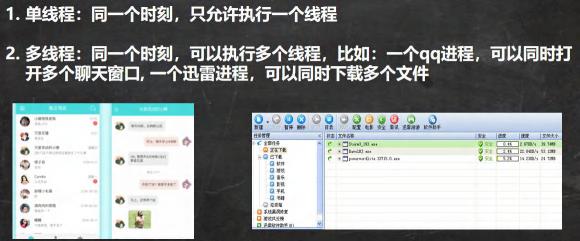


17.1.3 什么是线程





17.1.4 其他相关概念



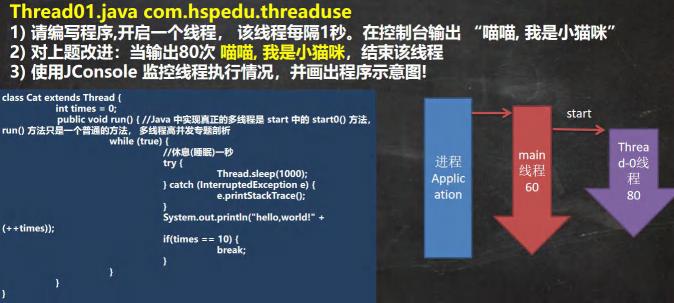


17.2 线程基本使用

17.2.1 创建线程的两种方式



17.2.2 线程应用案例 1-继承 Thread 类



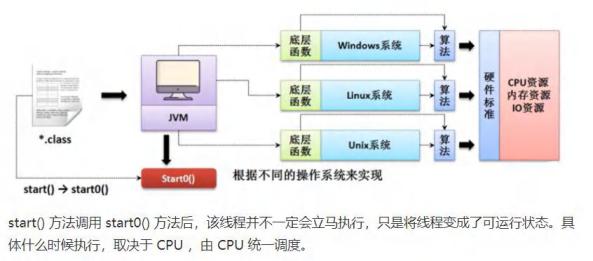
JConsole 小溪 ， 池塘

|  |
| --- |
| package com.hspedu.threaduse;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示通过继承 Thread 类创建线程  \*/  public class Thread01 {  public static void main(String[] args) throws InterruptedException {  //创建 Cat对象，可以当做线程使用  Cat cat = new Cat();  //老韩读源码  /\* |

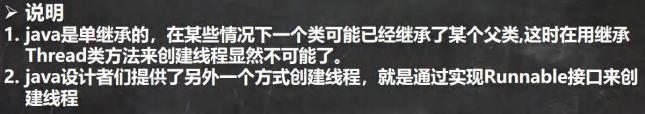
|  |
| --- |
| (1)  public synchronized void start() {  start0();  }  (2)  //start0() 是本地方法，是 JVM 调用, 底层是 c/c++实现  //真正实现多线程的效果， 是 start0(), 而不是 run  private native void start0();  \*/  cat.start();//启动线程-> 最终会执行 cat 的 run 方法  //cat.run();//run 方法就是一个普通的方法, 没有真正的启动一个线程，就会把 run 方法执行完毕，才向下执行  //说明: 当 main 线程启动一个子线程 Thread-0, 主线程不会阻塞, 会继续执行  //这时 主线程和子线程是交替执行..  System.out.println("主线程继续执行" + Thread.currentThread().getName());//名字 main  for(int i = 0; i < 60; i++) {  System.out.println("主线程 i=" + i);  //让主线程休眠  Thread.sleep(1000);  }  } |

|  |
| --- |
| }  //老韩说明  //1. 当一个类继承了 Thread 类， 该类就可以当做线程使用  //2. 我们会重写 run 方法，写上自己的业务代码  //3. run Thread 类 实现了 Runnable 接口的 run 方法  /\*  @Override  public void run() {  if (target != null) {  target.run();  }  }  \*/  class Cat extends Thread {  int times = 0;  @Override  public void run() {//重写 run 方法，写上自己的业务逻辑  while (true) {  //该线程每隔 1 秒。在控制台输出 “喵喵, 我是小猫咪 ”  System.out.println("喵喵, 我是小猫咪" + (++times) + " 线程名=" + Thread.currentThread().getName()); |

|  |
| --- |
| //让该线程休眠 1 秒 ctrl+alt+t  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  e.printStackTrace();  }  if(times == 80) {  break;//当 times 到 80, 退出 while, 这时线程也就退出..  }  }  }  } |



17.2.3 线程应用案例 2-实现 Runnable 接口





|  |
| --- |
| package com.hspedu.threaduse;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 通过实现接口 Runnable 来开发线程  \*/  public class Thread02 {  public static void main(String[] args) {  Dog dog = new Dog();  //dog.start(); 这里不能调用 start  //创建了 Thread 对象，把 dog 对象(实现 Runnable),放入 Thread  Thread thread = new Thread(dog);  thread.start();  // Tiger tiger = new Tiger();//实现了 Runnable  // ThreadProxy threadProxy = new ThreadProxy(tiger);  // threadProxy.start(); |

|  |
| --- |
| }  }  class Animal {  }  class Tiger extends Animal implements Runnable {  @Override  public void run() {  System.out.println("老虎嗷嗷叫....");  }  }  //线程代理类 , 模拟了一个极简的 Thread 类  class ThreadProxy implements Runnable {//你可以把 Proxy 类当做 ThreadProxy  private Runnable target = null;//属性，类型是 Runnable  @Override  public void run() {  if (target != null) {  target.run();//动态绑定（运行类型 Tiger）  }  } |

|  |
| --- |
| public ThreadProxy(Runnable target) {  this.target = target;  }  public void start() {  start0();//这个方法时真正实现多线程方法  }  public void start0() {  run();  }  }  class Dog implements Runnable { //通过实现 Runnable 接口，开发线程  int count = 0;  @Override  public void run() { //普通方法  while (true) {  System.out.println("小狗汪汪叫..hi" + (++count) + Thread.currentThread().getName());  //休眠 1 秒  try {  Thread.sleep(1000); |

|  |
| --- |
| } catch (InterruptedException e) {  e.printStackTrace();  }  if (count == 10) {  break;  }  }  }  } |

17.2.4 线程使用应用案例-多线程执行

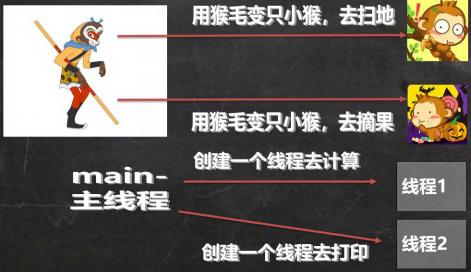


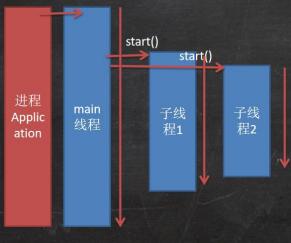
|  |
| --- |
| package com.hspedu.threaduse;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* main 线程启动两个子线程  \*/  public class Thread03 {  public static void main(String[] args) {  T1 t1 = new T1(); |

|  |
| --- |
| T2 t2 = new T2();  Thread thread1 = new Thread(t1);  Thread thread2 = new Thread(t2);  thread1.start();//启动第 1 个线程  thread2.start();//启动第 2 个线程  //...  }  }  class T1 implements Runnable {  int count = 0;  @Override  public void run() {  while (true) {  //每隔 1 秒输出 “hello,world ”,输出 10 次  System.out.println("hello,world " + (++count));  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  e.printStackTrace();  }  if(count == 60) {  break; |

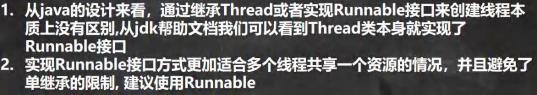
|  |
| --- |
| }  }  }  }  class T2 implements Runnable {  int count = 0;  @Override  public void run() {  //每隔 1 秒输出 “hi ”,输出 5 次  while (true) {  System.out.println("hi " + (++count));  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  e.printStackTrace();  }  if(count == 50) {  break;  }  }  }  } |

17.2.5 线程如何理解





17.3 继承 Thread vs 实现 Runnable 的区别





|  |
| --- |
| package com.hspedu.ticket;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 使用多线程，模拟三个窗口同时售票 100 张  \*/  public class SellTicket {  public static void main(String[] args) {  //测试  // SellTicket01 sellTicket01 = new SellTicket01();  // SellTicket01 sellTicket02 = new SellTicket01();  // SellTicket01 sellTicket03 = new SellTicket01();  //  // //这里我们会出现超卖..  // sellTicket01.start();//启动售票线程  // sellTicket02.start();//启动售票线程  // sellTicket03.start();//启动售票线程  System.out.println("===使用实现接口方式来售票=====");  SellTicket02 sellTicket02 = new SellTicket02();  new Thread(sellTicket02).start();//第 1 个线程-窗口  new Thread(sellTicket02).start();//第 2 个线程-窗口 |

|  |
| --- |
| new Thread(sellTicket02).start();//第 3 个线程-窗口  }  }  //使用 Thread 方式  class SellTicket01 extends Thread {  private static int ticketNum = 100;//让多个线程共享 ticketNum  @Override  public void run() {  while (true) {  if (ticketNum <= 0) {  System.out.println("售票结束...");  break;  }  //休眠 50 毫秒, 模拟  try {  Thread.sleep(50);  } catch (InterruptedException e) { |

|  |
| --- |
| e.printStackTrace();  }  System.out.println("窗口 " + Thread.currentThread().getName() + " 售出一张票"  + " 剩余票数=" + (--ticketNum));  }  }  }  //实现接口方式  class SellTicket02 implements Runnable {  private int ticketNum = 100;//让多个线程共享 ticketNum  @Override  public void run() {  while (true) {  if (ticketNum <= 0) {  System.out.println("售票结束...");  break;  }  //休眠 50 毫秒, 模拟 |

|  |
| --- |
| try {  Thread.sleep(50);  } catch (InterruptedException e) {  e.printStackTrace();  }  System.out.println("窗口 " + Thread.currentThread().getName() + " 售出一张票"  + " 剩余票数=" + (--ticketNum));//1 - 0 - - 1 - -2  }  }  } |

17.4 线程终止

17.4.1 基本说明

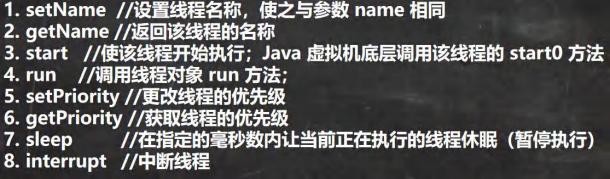


17.4.2 应用案例 ThreadExit\_.java com.hspedu.exit\_

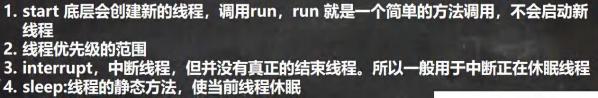


17.5 线程常用方法

17.5.1 常用方法第一组



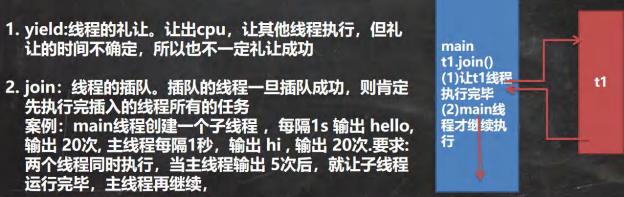
17.5.2 注意事项和细节

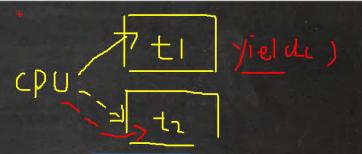


17.5.3 应用案例 ThreadMethod01.java



17.5.4 常用方法第二组

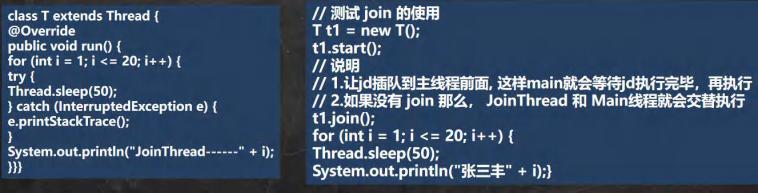




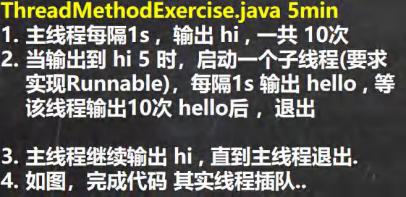
17.5.5 应用案例

ThreadMethod02.java

测试 yield 和 join 方法 ，注意体会方法的特点，看老师代码演示



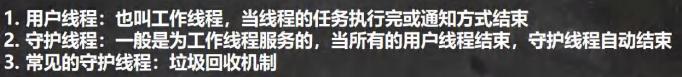
17.5.6 课堂练习



|  |
| --- |
| package com.hspedu.method;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class ThreadMethodExercise {  public static void main(String[] args) throws InterruptedException {  Thread t3 = new Thread(new T3());//创建子线程  for (int i = 1; i <= 10; i++) {  System.out.println("hi " + i);  if(i == 5) {//说明主线程输出了 5 次 hi  t3.start();//启动子线程 输出 hello...  t3.join();//立即将 t3 子线程，插入到 main 线程，让 t3 先执行  }  Thread.sleep(1000);//输出一次 hi, 让 main 线程也休眠 1s  }  }  } |

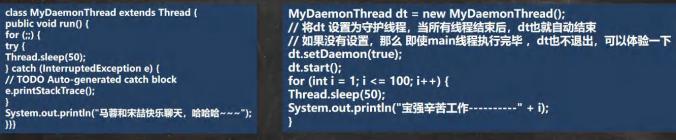
|  |
| --- |
| class T3 implements Runnable {  private int count = 0;  @Override  public void run() {  while (true) {  System.out.println("hello " + (++count));  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  e.printStackTrace();  }  if (count == 10) {  break;  }  }  }  } |

17.5.7 用户线程和守护线程



17.5.8 应用案例 ThreadMethod03.java

下面我们测试如何将一个线程设置成守护线程

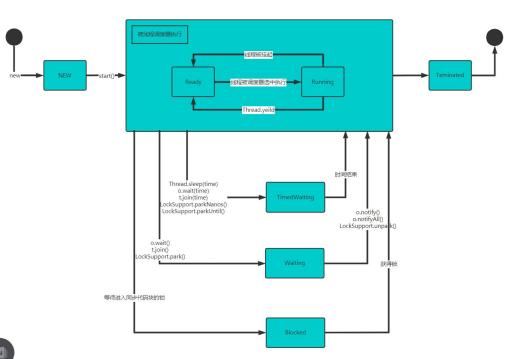


17.6 线程的生命周期

17.6.1 JDK 中用 Thread.State 枚举表示了线程的几种状态



17.6.2 线程状态转换图



17.6.3 写程序查看线程状态 ThreadState\_.java

|  |
| --- |
| package com.hspedu.state\_;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class ThreadState\_ {  public static void main(String[] args) throws InterruptedException {  T t = new T();  System.out.println(t.getName() + " 状态 " + t.getState());  t.start(); |

|  |
| --- |
| while (Thread.State.TERMINATED != t.getState()) {  System.out.println(t.getName() + " 状态 " + t.getState());  Thread.sleep(500);  }  System.out.println(t.getName() + " 状态 " + t.getState());  }  }  class T extends Thread {  @Override  public void run() {  while (true) {  for (int i = 0; i < 10; i++) {  System.out.println("hi " + i);  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  break;  }  }  } |

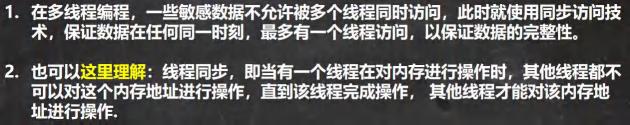
17.7 线程的同步

17.7.1 先看一个问题

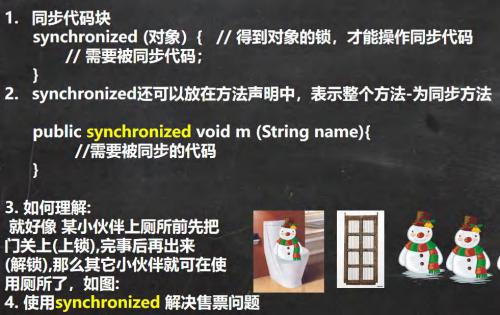


17.8 Synchronized

17.8.1 线程同步机制



17.8.2 同步具体方法-Synchronized

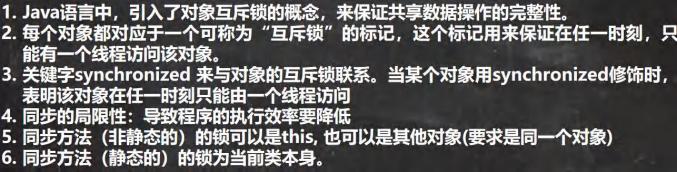


17.9 分析同步原理



17. 10互斥锁

17.10. 1 基本介绍



17.10.2 使用互斥锁来解决售票问题

看老师代码演示(两种方式都演示下， 代码块加锁，和方法上加锁)

|  |
| --- |
| package com.hspedu.syn;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 使用多线程，模拟三个窗口同时售票 100 张  \*/  public class SellTicket {  public static void main(String[] args) {  //测试  // SellTicket01 sellTicket01 = new SellTicket01();  // SellTicket01 sellTicket02 = new SellTicket01();  // SellTicket01 sellTicket03 = new SellTicket01();  //  // //这里我们会出现超卖..  // sellTicket01.start();//启动售票线程 |

|  |
| --- |
| // sellTicket02.start();//启动售票线程  // sellTicket03.start();//启动售票线程  // System.out.println("===使用实现接口方式来售票=====");  // SellTicket02 sellTicket02 = new SellTicket02();  //  // new Thread(sellTicket02).start();//第 1 个线程-窗口  // new Thread(sellTicket02).start();//第 2 个线程-窗口  // new Thread(sellTicket02).start();//第 3 个线程-窗口  //测试一把  SellTicket03 sellTicket03 = new SellTicket03();  new Thread(sellTicket03).start();//第 1 个线程-窗口  new Thread(sellTicket03).start();//第 2 个线程-窗口  new Thread(sellTicket03).start();//第 3 个线程-窗口  }  }  //实现接口方式, 使用 synchronized 实现线程同步  class SellTicket03 implements Runnable {  private int ticketNum = 100;//让多个线程共享 ticketNum  private boolean loop = true;//控制 run 方法变量  Object object = new Object(); |

|  |
| --- |
| //同步方法（静态的）的锁为当前类本身  //老韩解读  //1. public synchronized static void m1() {} 锁是加在 SellTicket03.class  //2. 如果在静态方法中，实现一个同步代码块.  /\*  synchronized (SellTicket03.class) {  System.out.println("m2");  }  \*/  public synchronized static void m1() {  }  public static void m2() {  synchronized (SellTicket03.class) {  System.out.println("m2");  }  }  //老韩说明  //1. public synchronized void sell() {} 就是一个同步方法  //2. 这时锁在 this 对象  //3. 也可以在代码块上写 synchronize , 同步代码块, 互斥锁还是在 this 对象  public /\*synchronized\*/ void sell() { //同步方法, 在同一时刻， 只能有一个线程来执行 sell 方法 |

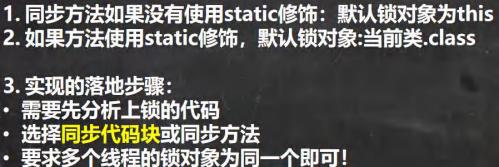
|  |
| --- |
| synchronized (/\*this\*/ object) {  if (ticketNum <= 0) {  System.out.println("售票结束...");  loop = false;  return;  }  //休眠 50 毫秒, 模拟  try {  Thread.sleep(50);  } catch (InterruptedException e) {  e.printStackTrace();  }  System.out.println("窗口 " + Thread.currentThread().getName() + " 售出一张票"  + " 剩余票数=" + (--ticketNum));//1 - 0 - - 1 - -2  }  }  @Override  public void run() {  while (loop) {  sell();//sell 方法是一共同步方法  }  } |

|  |
| --- |
| }  //使用 Thread 方式  // new SellTicket01().start()  // new SellTicket01().start();  class SellTicket01 extends Thread {  private static int ticketNum = 100;//让多个线程共享 ticketNum  // public void m1() {  // synchronized (this) {  // System.out.println("hello");  // }  // }  @Override  public void run() {  while (true) {  if (ticketNum <= 0) {  System.out.println("售票结束...");  break;  } |

|  |
| --- |
| //休眠 50 毫秒, 模拟  try {  Thread.sleep(50);  } catch (InterruptedException e) {  e.printStackTrace();  }  System.out.println("窗口 " + Thread.currentThread().getName() + " 售出一张票"  + " 剩余票数=" + (--ticketNum));  }  }  }  //实现接口方式  class SellTicket02 implements Runnable {  private int ticketNum = 100;//让多个线程共享 ticketNum  @Override  public void run() {  while (true) {  if (ticketNum <= 0) {  System.out.println("售票结束...");  break; |

|  |
| --- |
| }  //休眠 50 毫秒, 模拟  try {  Thread.sleep(50);  } catch (InterruptedException e) {  e.printStackTrace();  }  System.out.println("窗口 " + Thread.currentThread().getName() + " 售出一张票"  + " 剩余票数=" + (--ticketNum));//1 - 0 - - 1 - -2  }  }  } |

17.10.3 注意事项和细节



17. 11线程的死锁

17.11. 1 基本介绍



17.11.2 应用案例



17.11.3 应用案例 DeadLock\_.java

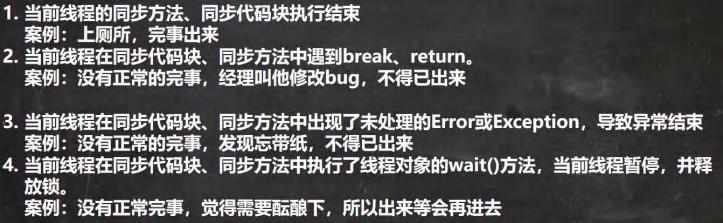
|  |
| --- |
| package com.hspedu.syn;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 模拟线程死锁  \*/  public class DeadLock\_ {  public static void main(String[] args) {  //模拟死锁现象  DeadLockDemo A = new DeadLockDemo(true);  A.setName("A 线程");  DeadLockDemo B = new DeadLockDemo(false);  B.setName("B 线程");  A.start();  B.start();  }  }  //线程 |

|  |  |
| --- | --- |
| class DeadLockDemo extends Thread {  static Object o1 = new Object();// 保证多线程，共享一个对象,这里使用 static  static Object o2 = new Object();  boolean flag;  public DeadLockDemo(boolean flag) {//构造器  this.flag = flag;  }  @Override  public void run() {  //下面业务逻辑的分析  //1. 如果 flag 为 T, 线程 A 就会先得到/持有 o1 对象锁, 然后尝试去获取 o2 对象锁 | |
| //2. 如果线程 A 得不到 o2 //3. 如果 flag 为 F, 线程 B //4. 如果线程 B 得不到 o1  if (flag) { | 对象锁，就会 Blocked  就会先得到/持有 o2 对象锁, 然后尝试去获取 o1 对象锁  对象锁，就会 Blocked |
| synchronized (o1) {//对象互斥锁, 下面就是同步代码  System.out.println(Thread.currentThread().getName() + " 进入 1");  synchronized (o2) { // 这里获得 li 对象的监视权  System.out.println(Thread.currentThread().getName() + " 进入 2");  }  }  } else { | |

|  |
| --- |
| synchronized (o2) {  System.out.println(Thread.currentThread().getName() + " 进入 3");  synchronized (o1) { // 这里获得 li 对象的监视权  System.out.println(Thread.currentThread().getName() + " 进入 4");  }  }  }  }  } |

17. 12释放锁

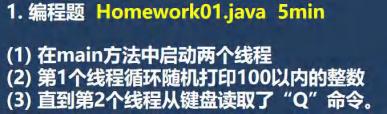
17.12. 1 下面操作会释放锁

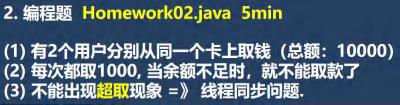


17.12.2 下面操作不会释放锁



17. 13本章作业

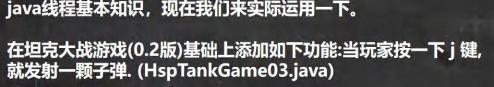


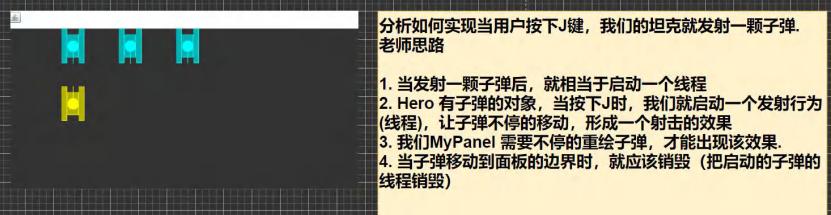


第 18 章坦克大战【2】

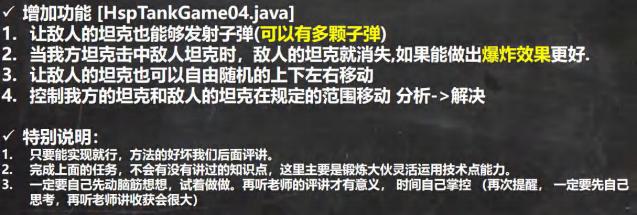
18. 1 线程-应用到坦克大战

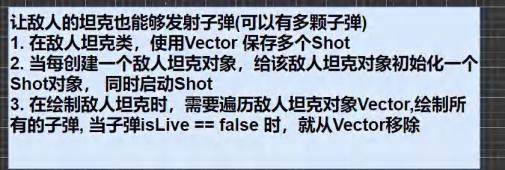
18.1.1 坦克大战 0.3

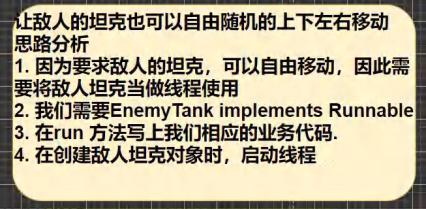


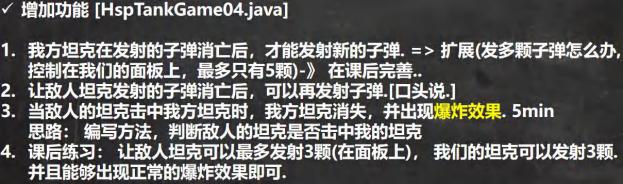


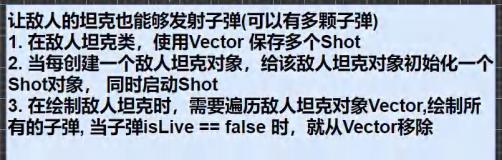
18.1.2 坦克大战 0.4 版

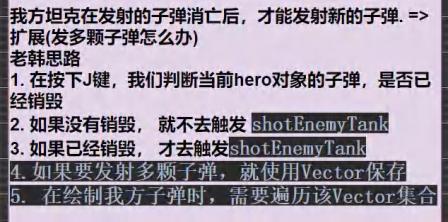












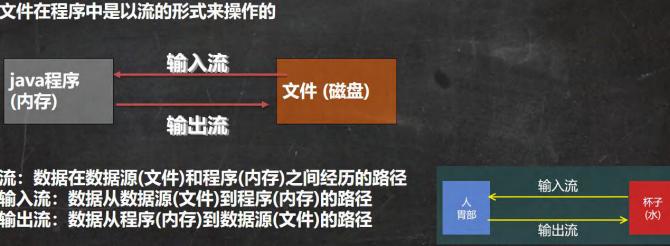
第 19 章IO 流

19. 1 文件

19.1.1 什么是文件

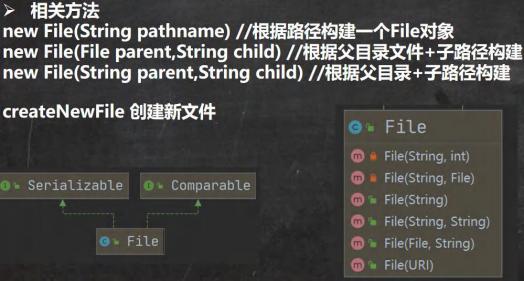


19.1.2 文件流



19.2 常用的文件操作

19.2.1 创建文件对象相关构造器和方法





|  |
| --- |
| package com.hspedu.file;  import org.junit.jupiter.api.Test;  import java.io.\*;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示创建文件  \*/  public class FileCreate {  public static void main(String[] args) { |

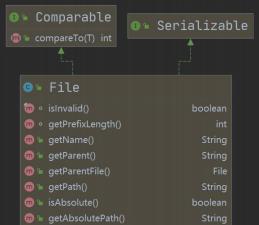
|  |
| --- |
| }  //方式 1 new File(String pathname)  @Test  public void create01() {  String filePath = "e:\\news1.txt";  File file = new File(filePath);  try {  file.createNewFile();  System.out.println("文件创建成功");  } catch (IOException e) {  e.printStackTrace();  }  }  //方式 2 new File(File parent,String child) //根据父目录文件+子路径构建  //e:\\news2.txt  @Test  public void create02() {  File parentFile = new File("e:\\");  String fileName = "news2.txt";  //这里的 file 对象，在java 程序中，只是一个对象  //只有执行了 createNewFile 方法，才会真正的，在磁盘创建该文件  File file = new File(parentFile, fileName); |

|  |
| --- |
| try {  file.createNewFile();  System.out.println("创建成功~");  } catch (IOException e) {  e.printStackTrace();  }  }  //方式 3 new File(String parent,String child) //根据父目录+子路径构建  @Test  public void create03() {  //String parentPath = "e:\\";  String parentPath = "e:\\";  String fileName = "news4.txt";  File file = new File(parentPath, fileName);  try {  file.createNewFile();  System.out.println("创建成功~");  } catch (IOException e) {  e.printStackTrace();  }  }  //下面四个都是抽象类  // |

|  |
| --- |
| //InputStream  //OutputStream  //Reader //字符输入流  //Writer //字符输出流  } |

19.2.2 获取文件的相关信息





19.2.3 应用案例演示 FileInformation.java



|  |
| --- |
| package com.hspedu.file;  import org.junit.jupiter.api.Test;  import java.io.File; |

|  |
| --- |
| /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class FileInformation {  public static void main(String[] args) {  }  //获取文件的信息  @Test  public void info() {  //先创建文件对象  File file = new File("e:\\news1.txt");  //调用相应的方法，得到对应信息  System.out.println("文件名字=" + file.getName());  //getName 、getAbsolutePath 、getParent 、length 、exists 、isFile 、isDirectory  System.out.println("文件绝对路径=" + file.getAbsolutePath());  System.out.println("文件父级目录=" + file.getParent());  System.out.println("文件大小(字节)=" + file.length());  System.out.println("文件是否存在=" + file.exists());//T  System.out.println("是不是一个文件=" + file.isFile());//T  System.out.println("是不是一个目录=" + file.isDirectory());//F |

|  |
| --- |
| }  } |

19.2.4 目录的操作和文件删除



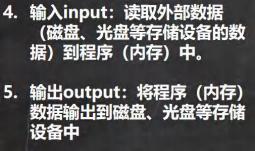
19.2.5 应用案例演示



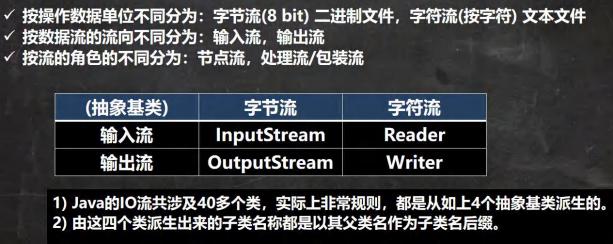
19.3 IO 流原理及流的分类

19.3.1 Java IO 流原理



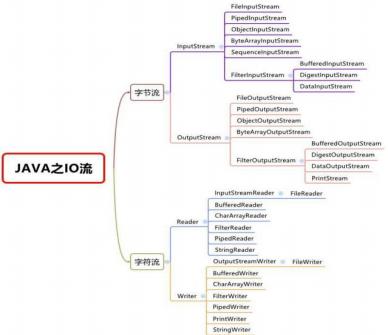


19.3.2 流的分类

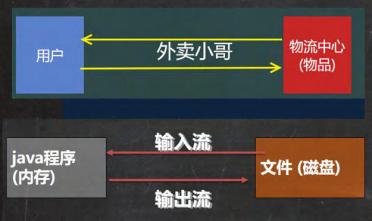


19.4 IO 流体系图-常用的类

1. IO 流体系图



2. 文件 VS 流



19.4.1 FileInputStream 介绍

19.4.2 FileInputStream 应用实例 FileInputStream\_.java

要求: 请使用 FileInputStream 读取 hello.txt 文件，并将文件内容显示到控制台. [老师代码演示]

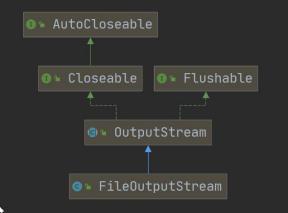
|  |
| --- |
| package com.hspedu.inputstream\_;  import org.junit.jupiter.api.Test;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  import java.io.IOException;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示 FileInputStream 的使用(字节输入流 文件--> 程序)  \*/  public class FileInputStream\_ {  public static void main(String[] args) { |

|  |
| --- |
| }  /\*\*  \* 演示读取文件...  \* 单个字节的读取，效率比较低  \* -> 使用 read(byte[] b)  \*/  @Test  public void readFile01() {  String filePath = "e:\\hello.txt";  int readData = 0;  FileInputStream fileInputStream = null;  try {  //创建 FileInputStream 对象，用于读取 文件  fileInputStream = new FileInputStream(filePath);  //从该输入流读取一个字节的数据。 如果没有输入可用，此方法将阻止。  //如果返回- 1 , 表示读取完毕  while ((readData = fileInputStream.read()) != - 1) {  System.out.print((char)readData);//转成 char 显示  }  } catch (IOException e) {  e.printStackTrace();  } finally {  //关闭文件流，释放资源. |

|  |
| --- |
| try {  fileInputStream.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  /\*\*  \* 使用 read(byte[] b) 读取文件，提高效率  \*/  @Test  public void readFile02() {  String filePath = "e:\\hello.txt";  //字节数组  byte[] buf= new byte[8]; //一次读取 8 个字节.  int readLen = 0;  FileInputStream fileInputStream = null;  try {  //创建 FileInputStream 对象，用于读取 文件  fileInputStream = new FileInputStream(filePath);  //从该输入流读取最多 b.length 字节的数据到字节数组。 此方法将阻塞，直到某些输入可用。  //如果返回- 1 , 表示读取完毕  //如果读取正常, 返回实际读取的字节数  while ((readLen = fileInputStream.read(buf)) != - 1) { |

|  |
| --- |
| System.out.print(new String(buf, 0, readLen));//显示  }  } catch (IOException e) {  e.printStackTrace();  } finally {  //关闭文件流，释放资源.  try {  fileInputStream.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  } |

19.4.3 FileOutputStream 介绍



19.4.4 FileOutputStream 应用实例 1 FileOutputStream01.java

要求: 请使用 FileOutputStream 在 a.txt 文件，中写入 “hello ，world ”. [老师代码演示], 如果文件不存在，会创建

文件(注意：前提是目录已经存在.)

|  |
| --- |
| package com.hspedu.outputstream\_;  import org.junit.jupiter.api.Test;  import java.io.FileNotFoundException;  import java.io.FileOutputStream;  import java.io.IOException;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class FileOutputStream01 {  public static void main(String[] args) {  }  /\*\*  \* 演示使用 FileOutputStream 将数据写到文件中,  \* 如果该文件不存在，则创建该文件  \*/  @Test |

|  |
| --- |
| public void writeFile() {  //创建 FileOutputStream 对象  String filePath = "e:\\a.txt";  FileOutputStream fileOutputStream = null;  try {  //得到 FileOutputStream 对象 对象  //老师说明  //1. new FileOutputStream(filePath) 创建方式，当写入内容是，会覆盖原来的内容  //2. new FileOutputStream(filePath, true) 创建方式，当写入内容是，是追加到文件后面  fileOutputStream = new FileOutputStream(filePath, true);  //写入一个字节  //fileOutputStream.write('H');//  //写入字符串  String str = "hsp,world!";  //str.getBytes() 可以把 字符串-> 字节数组  //fileOutputStream.write(str.getBytes());  /\*  write(byte[] b, int off, int len) 将 len 字节从位于偏移量 off 的指定字节数组写入此文件输出流  \*/  fileOutputStream.write(str.getBytes(), 0, 3);  } catch (IOException e) {  e.printStackTrace();  } finally {  try { |

|  |
| --- |
| fileOutputStream.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  } |

19.4.5 FileOutputStream 应用实例 2 FileCopy.java

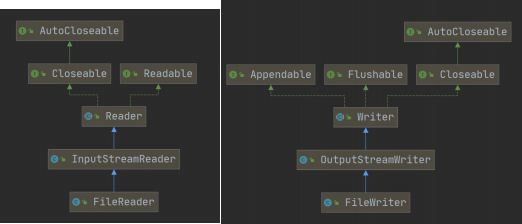
要求: 编程完成图片/音乐 的拷贝. [老师代码演示]

|  |
| --- |
| package com.hspedu.outputstream\_;  import com.hspedu.inputstream\_.FileInputStream\_;  import java.io.\*;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class FileCopy {  public static void main(String[] args) {  //完成 文件拷贝，将 e:\\Koala.jpg 拷贝 c:\\  //思路分析 |

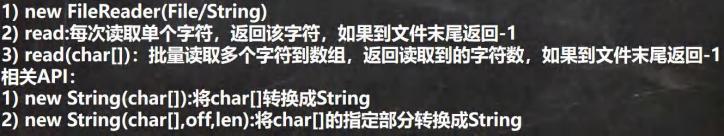
|  |
| --- |
| //1. 创建文件的输入流 , 将文件读入到程序  //2. 创建文件的输出流， 将读取到的文件数据，写入到指定的文件.  String srcFilePath = "e:\\Koala.jpg";  String destFilePath = "e:\\Koala3.jpg";  FileInputStream fileInputStream = null;  FileOutputStream fileOutputStream = null;  try {  fileInputStream = new FileInputStream(srcFilePath);  fileOutputStream = new FileOutputStream(destFilePath);  //定义一个字节数组,提高读取效果  byte[] buf= new byte[1024];  int readLen = 0;  while ((readLen = fileInputStream.read(buf)) != - 1) {  //读取到后，就写入到文件 通过 fileOutputStream  //即，是一边读，一边写  fileOutputStream.write(buf, 0, readLen);//一定要使用这个方法  }  System.out.println("拷贝 ok~");  } catch (IOException e) {  e.printStackTrace();  } finally { |

|  |
| --- |
| try {  //关闭输入流和输出流，释放资源  if (fileInputStream != null) {  fileInputStream.close();  }  if (fileOutputStream != null) {  fileOutputStream.close();  }  } catch (IOException e) {  e.printStackTrace();  }  }  }  } |

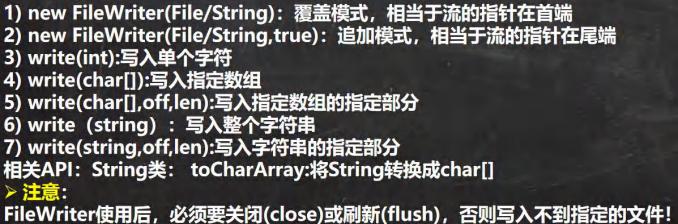
19.4.6 FileReader 和 FileWriter 介绍



19.4.7 FileReader 相关方法：



19.4.8 FileWriter 常用方法



19.4.9 FileReader 和 FileWriter 应用案例 FileReader\_.java

要求：

1) 使用 FileReader 从 story.txt 读取内容，并显示

|  |
| --- |
| package com.hspedu.reader\_;  import org.junit.jupiter.api.Test;  import java.io.FileNotFoundException;  import java.io.FileReader;  import java.io.FileWriter;  import java.io.IOException; |

|  |
| --- |
| /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class FileReader\_ {  public static void main(String[] args) {  }  /\*\*  \* 单个字符读取文件  \*/  @Test  public void readFile01() {  String filePath = "e:\\story.txt";  FileReader fileReader = null;  int data = 0;  //1. 创建 FileReader 对象  try {  fileReader = new FileReader(filePath);  //循环读取 使用 read, 单个字符读取  while ((data = fileReader.read()) != - 1) {  System.out.print((char) data);  } |

|  |
| --- |
| } catch (IOException e) {  e.printStackTrace();  } finally {  try {  if (fileReader != null) {  fileReader.close();  }  } catch (IOException e) {  e.printStackTrace();  }  }  }  /\*\*  \* 字符数组读取文件  \*/  @Test  public void readFile02() {  System.out.println("~~~readFile02 ~~~");  String filePath = "e:\\story.txt";  FileReader fileReader = null;  int readLen = 0;  char[] buf= new char[8];  //1. 创建 FileReader 对象 |

|  |
| --- |
| try {  fileReader = new FileReader(filePath);  //循环读取 使用read(buf), 返回的是实际读取到的字符数  //如果返回- 1, 说明到文件结束  while ((readLen = fileReader.read(buf)) != - 1) {  System.out.print(new String(buf, 0, readLen));  }  } catch (IOException e) {  e.printStackTrace();  } finally {  try {  if (fileReader != null) {  fileReader.close();  }  } catch (IOException e) {  e.printStackTrace();  }  }  }  } |

2) 使用 FileWriter 将 “风雨之后，定见彩虹 ” 写入到 note.txt 文件中, 注意细节.

FileWriter\_.java com.hspedu.writer\_

|  |
| --- |
| package com.hspedu.writer\_; |

|  |
| --- |
| import java.io.FileWriter;  import java.io.IOException;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class FileWriter\_ {  public static void main(String[] args) {  String filePath = "e:\\note.txt";  //创建 FileWriter 对象  FileWriter fileWriter = null;  char[] chars = {'a', 'b', 'c'};  try {  fileWriter = new FileWriter(filePath);//默认是覆盖写入  // 3) write(int):写入单个字符  fileWriter.write('H');  // 4) write(char[]):写入指定数组  fileWriter.write(chars);  // 5) write(char[],off,len):写入指定数组的指定部分  fileWriter.write("韩顺平教育".toCharArray(), 0, 3);  // 6) write（string）：写入整个字符串  fileWriter.write(" 你好北京~");  fileWriter.write("风雨之后，定见彩虹"); |

|  |
| --- |
| // 7) write(string,off,len):写入字符串的指定部分  fileWriter.write("上海天津", 0, 2);  //在数据量大的情况下，可以使用循环操作.  } catch (IOException e) {  e.printStackTrace();  } finally {  //对应 FileWriter , 一定要关闭流，或者 flush 才能真正的把数据写入到文件  //老韩看源码就知道原因.  /\*  看看代码  private void writeBytes() throws IOException {  this.bb.flip();  int var1 = this.bb.limit();  int var2 = this.bb.position();  assert var2 <= var1;  int var3 = var2 <= var1 ? var1 - var2 : 0;  if (var3 > 0) {  if (this.ch != null) {  assert this.ch.write(this.bb) == var3 : var3;  } else {  this.out.write(this.bb.array(), this.bb.arrayOffset() + var2, var3); |

|  |
| --- |
| }  }  this.bb.clear();  }  \*/  try {  //fileWriter.flush();  //关闭文件流，等价 flush() + 关闭  fileWriter.close();  } catch (IOException e) {  e.printStackTrace();  }  }  System.out.println("程序结束...");  }  } |

19.5 节点流和处理流

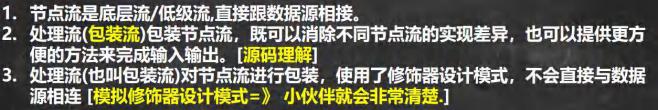
19.5.1 基本介绍



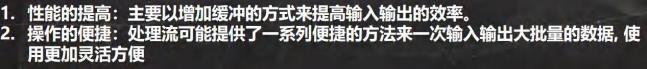
19.5.2 节点流和处理流一览图



19.5.3 节点流和处理流的区别和联系



19.5.4 处理流的功能主要体现在以下两个方面:



19.5.5 处理流-BufferedReader 和 BufferedWriter





|  |
| --- |
| package com.hspedu.reader\_;  import java.io.BufferedReader;  import java.io.FileReader;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示 bufferedReader 使用  \*/  public class BufferedReader\_ {  public static void main(String[] args) throws Exception {  String filePath = "e:\\a.java";  //创建 bufferedReader  BufferedReader bufferedReader = new BufferedReader(new FileReader(filePath)); |

|  |
| --- |
| //读取  String line; //按行读取, 效率高  //说明  //1. bufferedReader.readLine() 是按行读取文件  //2. 当返回null 时，表示文件读取完毕  while ((line = bufferedReader.readLine()) != null) {  System.out.println(line);  }  //关闭流, 这里注意，只需要关闭 BufferedReader ，因为底层会自动的去关闭 节点流  //FileReader。  /\*  public void close() throws IOException {  synchronized (lock) {  if (in == null)  return;  try {  in.close();//in 就是我们传入的 new FileReader(filePath), 关闭了.  } finally {  in = null;  cb = null;  }  }  }  \*/ |

|  |
| --- |
| bufferedReader.close();  }  } |



|  |
| --- |
| package com.hspedu.writer\_;  import java.io.BufferedWriter;  import java.io.FileWriter;  import java.io.IOException;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示 BufferedWriter 的使用  \*/  public class BufferedWriter\_ {  public static void main(String[] args) throws IOException {  String filePath = "e:\\ok.txt";  //创建 BufferedWriter  //说明:  //1. new FileWriter(filePath, true) 表示以追加的方式写入  //2. new FileWriter(filePath) , 表示以覆盖的方式写入 |

|  |
| --- |
| BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(filePath));  bufferedWriter.write("hello, 韩顺平教育!");  bufferedWriter.newLine();//插入一个和系统相关的换行  bufferedWriter.write("hello2, 韩顺平教育!");  bufferedWriter.newLine();  bufferedWriter.write("hello3, 韩顺平教育!");  bufferedWriter.newLine();  //说明：关闭外层流即可 ， 传入的 new FileWriter(filePath) ,会在底层关闭  bufferedWriter.close();  }  } |

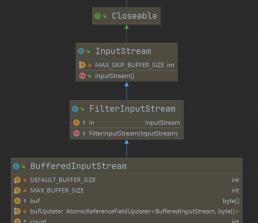


|  |
| --- |
| package com.hspedu.writer\_;  import java.io.\*;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class BufferedCopy\_ { |

|  |
| --- |
| public static void main(String[] args) {  //老韩说明  //1. BufferedReader 和 BufferedWriter 是安装字符操作  //2. 不要去操作 二进制文件[声音，视频，doc, pdf ], 可能造成文件损坏  //BufferedInputStream  //BufferedOutputStream  String srcFilePath = "e:\\a.java";  String destFilePath = "e:\\a2.java";  // String srcFilePath = "e:\\0245\_韩顺平零基础学 Java\_ 引出 this.avi";  // String destFilePath = "e:\\a2 韩顺平.avi";  BufferedReader br = null;  BufferedWriter bw = null;  String line;  try {  br = new BufferedReader(new FileReader(srcFilePath));  bw = new BufferedWriter(new FileWriter(destFilePath));  //说明: readLine 读取一行内容，但是没有换行  while ((line = br.readLine()) != null) {  //每读取一行，就写入  bw.write(line);  //插入一个换行  bw.newLine(); |

|  |
| --- |
| }  System.out.println("拷贝完毕...");  } catch (IOException e) {  e.printStackTrace();  } finally {  //关闭流  try {  if(br != null) {  br.close();  }  if(bw != null) {  bw.close();  }  } catch (IOException e) {  e.printStackTrace();  }  }  }  } |

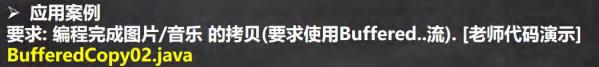
19.5.6 处理流-BufferedInputStream 和 BufferedOutputStream





19.5.7 介绍 BufferedOutputStream





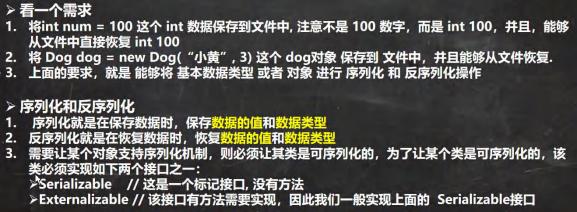
|  |
| --- |
| package com.hspedu.outputstream\_; |

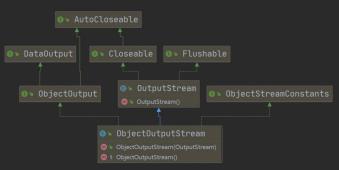
|  |
| --- |
| import java.io.\*;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示使用 BufferedOutputStream 和 BufferedInputStream 使用  \* 使用他们，可以完成二进制文件拷贝.  \* 思考：字节流可以操作二进制文件，可以操作文本文件吗？当然可以  \*/  public class BufferedCopy02 {  public static void main(String[] args) {  // String srcFilePath = "e:\\Koala.jpg";  // String destFilePath = "e:\\hsp.jpg";  // String srcFilePath = "e:\\0245\_韩顺平零基础学 Java\_ 引出 this.avi";  // String destFilePath = "e:\\hsp.avi";  String srcFilePath = "e:\\a.java";  String destFilePath = "e:\\a3.java";  //创建 BufferedOutputStream 对象 BufferedInputStream 对象  BufferedInputStream bis = null;  BufferedOutputStream bos = null;  try {  //因为 FileInputStream 是 InputStream 子类  bis = new BufferedInputStream(new FileInputStream(srcFilePath)); |

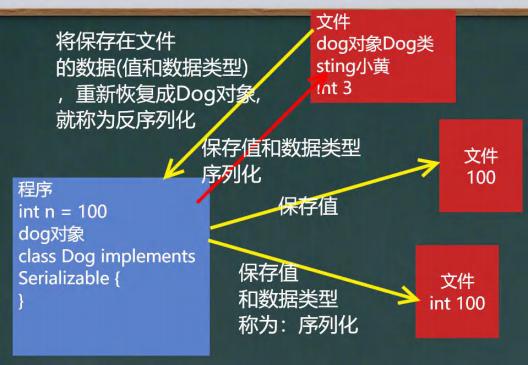
|  |
| --- |
| bos = new BufferedOutputStream(new FileOutputStream(destFilePath));  //循环的读取文件，并写入到 destFilePath  byte[] buff= new byte[1024];  int readLen = 0;  //当返回 - 1 时，就表示文件读取完毕  while ((readLen = bis.read(buff)) != - 1) {  bos.write(buff, 0, readLen);  }  System.out.println("文件拷贝完毕~~~");  } catch (IOException e) {  e.printStackTrace();  } finally {  //关闭流 , 关闭外层的处理流即可，底层会去关闭节点流  try {  if(bis != null) {  bis.close();  }  if(bos != null) {  bos.close();  }  } catch (IOException e) {  e.printStackTrace(); |

|  |
| --- |
| }  }  }  } |

19.5.8 对象流-ObjectInputStream 和 ObjectOutputStream





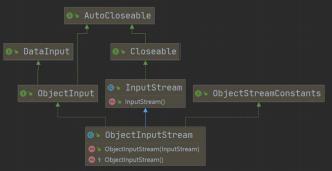


19.5.9 对象流介绍

功能：提供了对基本类型或对象类型的序列化和反序列化的方法

ObjectOutputStream 提供 序列化功能

ObjectInputStream 提供 反序列化功能





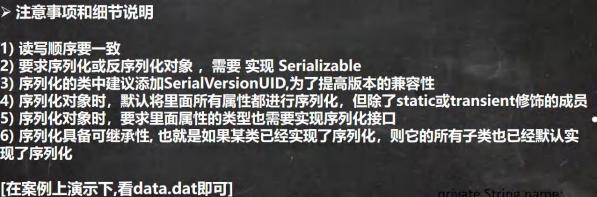
|  |
| --- |
| package com.hspedu.outputstream\_; |

|  |
| --- |
| import java.io.FileOutputStream;  import java.io.ObjectOutputStream;  import java.io.Serializable;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示 ObjectOutputStream 的使用, 完成数据的序列化  \*/  public class ObjectOutStream\_ {  public static void main(String[] args) throws Exception {  //序列化后，保存的文件格式，不是存文本，而是按照他的格式来保存  String filePath = "e:\\data.dat";  ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(filePath));  //序列化数据到 e:\data.dat  oos.writeInt(100);// int -> Integer (实现了 Serializable)  oos.writeBoolean(true);// boolean -> Boolean (实现了 Serializable)  oos.writeChar('a');// char -> Character (实现了 Serializable)  oos.writeDouble(9.5);// double -> Double (实现了 Serializable)  oos.writeUTF("韩顺平教育");//String  //保存一个 dog 对象  oos.writeObject(new Dog("旺财", 10, " 日本", " 白色"));  oos.close(); |

|  |
| --- |
| System.out.println("数据保存完毕(序列化形式)");  }  } |

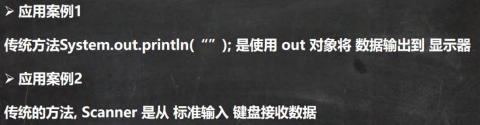


|  |
| --- |
| // 1.创建流对象  ObjectInputStream ois = new ObjectInputStream(new FileInputStream("src\\data.dat"));  // 2.读取， 注意顺序  System.out.println(ois.readInt());  System.out.println(ois.readBoolean());  System.out.println(ois.readChar());  System.out.println(ois.readDouble());  System.out.println(ois.readUTF());  System.out.println(ois.readObject());  System.out.println(ois.readObject());  System.out.println(ois.readObject());  // 3.关闭  ois.close();  System.out.println(" 以反序列化的方式读取(恢复)ok~"); |



19.5. 10 标准输入输出流

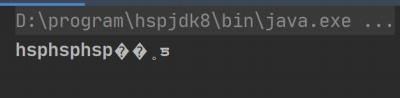




19.5. 11 转换流-InputStreamReader 和 OutputStreamWriter

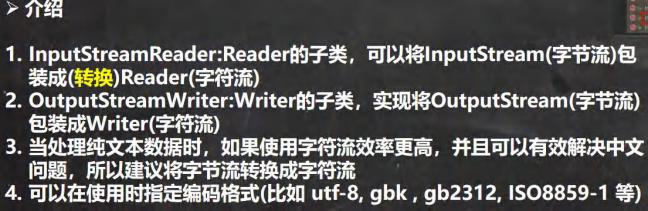
com.hspedu.transformation CodeQuestion.java

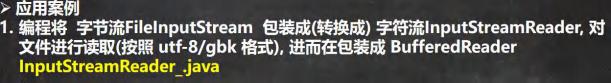
先看一个文件乱码问题，引出学习转换流必要性.



|  |
| --- |
| package com.hspedu.transformation; |

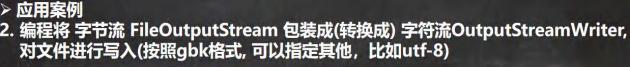
|  |
| --- |
| import java.io.\*;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 看一个中文乱码问题  \*/  public class CodeQuestion {  public static void main(String[] args) throws IOException {  //读取 e:\\a.txt 文件到程序  //思路  //1. 创建字符输入流 BufferedReader [处理流]  //2. 使用 BufferedReader 对象读取 a.txt  //3. 默认情况下，读取文件是按照 utf-8 编码  String filePath = "e:\\a.txt";  BufferedReader br = new BufferedReader(new FileReader(filePath));  String s = br.readLine();  System.out.println("读取到的内容: " + s);  br.close();  //InputStreamReader  //OutputStreamWriter  }  } |



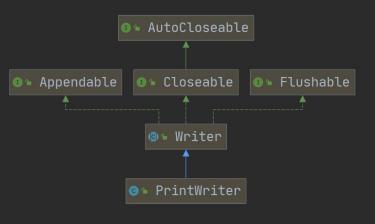


|  |
| --- |
| package com.hspedu.transformation;  import java.io.\*;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示使用 InputStreamReader 转换流解决中文乱码问题  \* 将字节流 FileInputStream 转成字符流 InputStreamReader, 指定编码 gbk/utf-8  \*/  public class InputStreamReader\_ {  public static void main(String[] args) throws IOException {  String filePath = "e:\\a.txt";  //解读  //1. 把 FileInputStream 转成 InputStreamReader |

|  |
| --- |
| //2. 指定编码 gbk  //InputStreamReader isr = new InputStreamReader(new FileInputStream(filePath), "gbk");  //3. 把 InputStreamReader 传入 BufferedReader  //BufferedReader br = new BufferedReader(isr);  //将 2 和 3 合在一起  BufferedReader br = new BufferedReader(new InputStreamReader(  new FileInputStream(filePath), "gbk"));  //4. 读取  String s = br.readLine();  System.out.println("读取内容=" + s);  //5. 关闭外层流  br.close();  }  } |



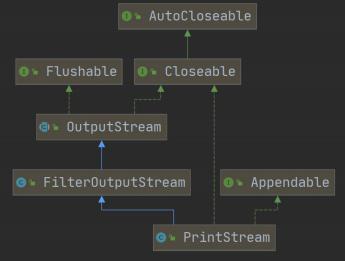
|  |
| --- |
| // 1.创建流对象  OutputStreamWriter osw =  new OutputStreamWriter(new FileOutputStream("d:\\a.txt"), "gbk"); |



|  |
| --- |
| // 2.写入  osw.write("hello,韩顺平教育~");  // 3.关闭  osw.close();  System.out.println("保存成功~"); |

19.6 打印流-PrintStream 和 PrintWriter





|  |
| --- |
| package com.hspedu.transformation;  import java.io.FileWriter;  import java.io.IOException;  import java.io.PrintWriter;  /\*\*  \* @author 韩顺平 |

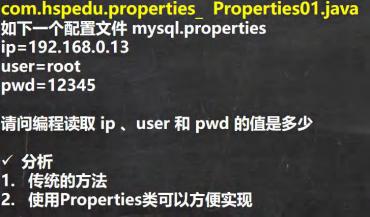
|  |
| --- |
| \* @version 1.0  \* 演示 PrintWriter 使用方式  \*/  public class PrintWriter\_ {  public static void main(String[] args) throws IOException {  //PrintWriter printWriter = new PrintWriter(System.out);  PrintWriter printWriter = new PrintWriter(new FileWriter("e:\\f2.txt"));  package com.hspedu.printstream;  import java.io.IOException;  import java.io.PrintStream;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示 PrintStream （字节打印流/输出流）  \*/  public class PrintStream\_ {  public static void main(String[] args) throws IOException {  PrintStream out = System.out;  //在默认情况下，PrintStream 输出数据的位置是 标准输出，即显示器  /\*  public void print(String s) {  if (s == null) { |

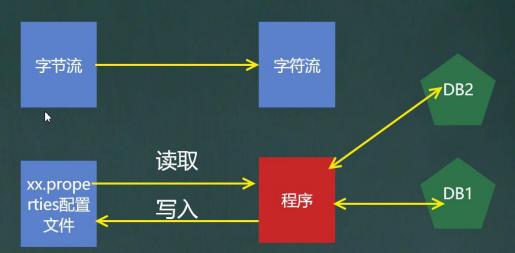
|  |
| --- |
| s = "null";  }  write(s);  }  \*/  out.print("john, hello");  //因为 print 底层使用的是 write , 所以我们可以直接调用 write 进行打印/输出  out.write("韩顺平,你好".getBytes());  out.close();  //我们可以去修改打印流输出的位置/设备  //1. 输出修改成到 "e:\\f1.txt"  //2. "hello, 韩顺平教育~" 就会输出到 e:\f1.txt  //3. public static void setOut(PrintStream out) {  // checkIO();  // setOut0(out); // native 方法，修改了 out  // }  System.setOut(new PrintStream("e:\\f1.txt"));  System.out.println("hello, 韩顺平教育~");  }  }  printWriter.print("hi, 北京你好~~~~");  printWriter.close();//flush + 关闭流, 才会将数据写入到文件.. |

|  |
| --- |
| }  } |
|  |

19.7 Properties 类

19.7.1 看一个需求



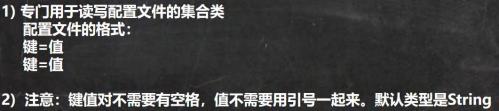


|  |
| --- |
| package com.hspedu.properties\_; |

|  |
| --- |
| import java.io.BufferedReader;  import java.io.FileNotFoundException;  import java.io.FileReader;  import java.io.IOException;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class Properties01 {  public static void main(String[] args) throws IOException {  //读取 mysql.properties 文件，并得到 ip, user 和 pwd  BufferedReader br = new BufferedReader(new FileReader("src\\mysql.properties"));  String line = "";  while ((line = br.readLine()) != null) { //循环读取  String[] split = line.split("=");  //如果我们要求指定的 ip 值  if("ip".equals(split[0])) {  System.out.println(split[0] + "值是: " + split[1]);  }  }  br.close();  } |

|  |
| --- |
| } |

19.7.2 基本介绍





19.7.3 应用案例



|  |
| --- |
| package com.hspedu.properties\_;  import java.io.FileNotFoundException;  import java.io.FileReader;  import java.io.IOException;  import java.util.Properties;  /\*\*  \* @author 韩顺平 |

|  |
| --- |
| \* @version 1.0  \*/  public class Properties02 {  public static void main(String[] args) throws IOException {  //使用 Properties 类来读取 mysql.properties 文件  //1. 创建 Properties 对象  Properties properties = new Properties();  //2. 加载指定配置文件  properties.load(new FileReader("src\\mysql.properties"));  //3. 把 k-v 显示控制台  properties.list(System.out);  //4. 根据 key 获取对应的值  String user = properties.getProperty("user");  String pwd = properties.getProperty("pwd");  System.out.println("用户名=" + user);  System.out.println("密码是=" + pwd);  }  } |
| package com.hspedu.properties\_;  import java.io.FileNotFoundException; |

|  |
| --- |
| import java.io.FileOutputStream;  import java.io.IOException;  import java.util.Properties;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class Properties03 {  public static void main(String[] args) throws IOException {  //使用 Properties 类来创建 配置文件, 修改配置文件内容  Properties properties = new Properties();  //创建  //1.如果该文件没有 key 就是创建  //2.如果该文件有 key ,就是修改  /\*  Properties 父类是 Hashtable ， 底层就是 Hashtable 核心方法  public synchronized V put(K key, V value) {  // Make sure the value is not null  if (value == null) {  throw new NullPointerException();  }  // Makes sure the key is not already in the hashtable.  Entry<?,?> tab[] = table; |

|  |
| --- |
| int hash = key.hashCode();  int index = (hash & 0x7FFFFFFF) % tab.length;  @SuppressWarnings("unchecked")  Entry<K,V> entry = (Entry<K,V>)tab[index];  for(; entry != null ; entry = entry.next) {  if ((entry.hash == hash) && entry.key.equals(key)) {  V old = entry.value;  entry.value = value;//如果 key 存在，就替换  return old;  }  }  addEntry(hash, key, value, index);//如果是新 k, 就 addEntry  return null;  }  \*/  properties.setProperty("charset", "utf8");  properties.setProperty("user", "汤姆");//注意保存时，是中文的 unicode 码值  properties.setProperty("pwd", "888888");  //将 k-v 存储文件中即可  properties.store(new FileOutputStream("src\\mysql2.properties"), null);  System.out.println("保存配置文件成功~");  } |

|  |
| --- |
| } |

19.8 本章作业



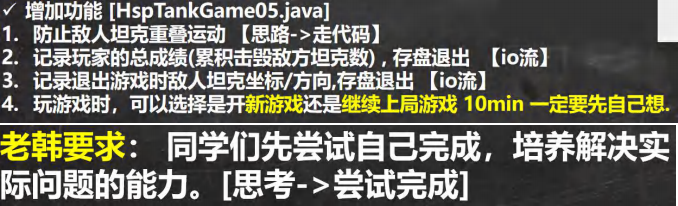


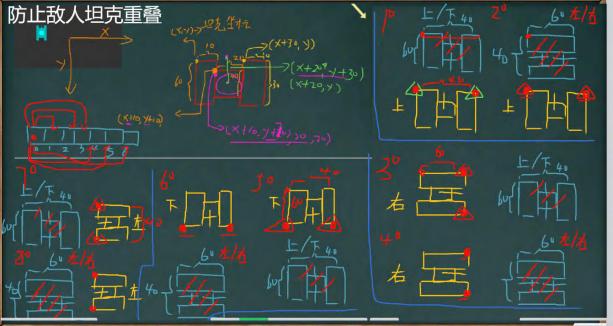


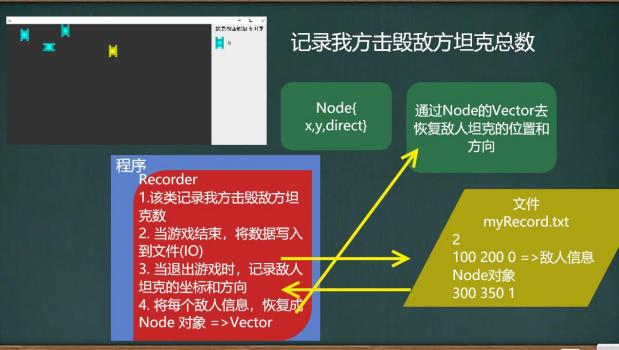
20. 1 IO 流-应用到坦克大战

老师提示: 把代码删除，重写写一遍，锻炼自己的思维.

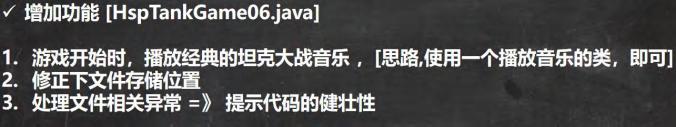
20.1.1 坦克大战 0.5 版







20.1.2 坦克大战 0.6 版



20.2 第二个阶段就到这里，大家好好总结



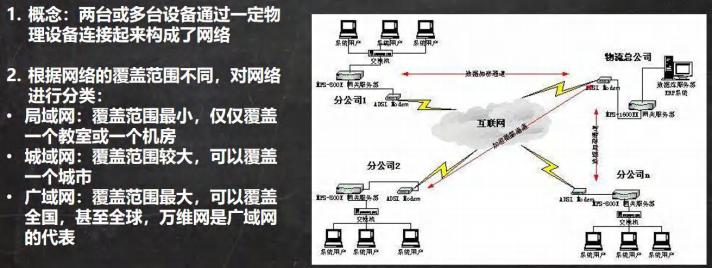
第 21 章网络编程

21. 1 网络的相关概念

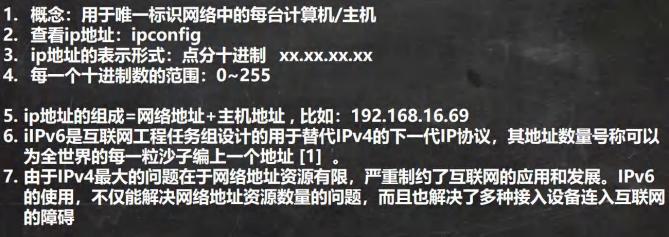
21.1.1 网络通信



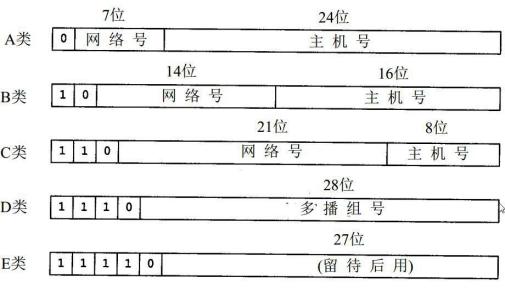
21.1.2 网络



21.1.3 ip 地址

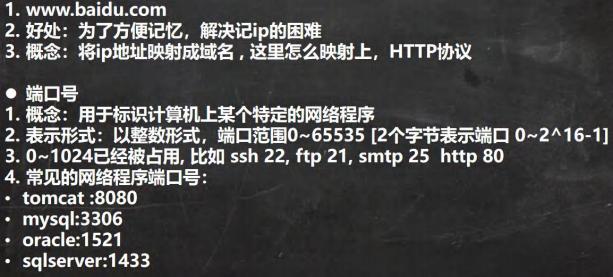


21.1.4 ipv4 地址分类

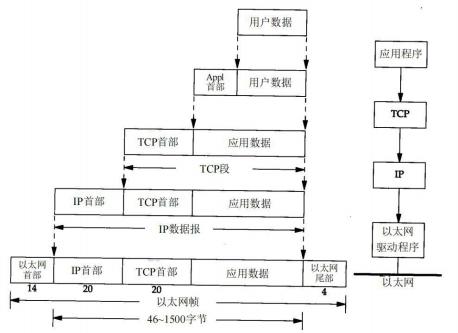


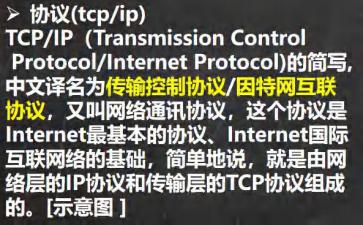


21.1.5 域名



21.1.6 网络通信协议





21.1.7 网络通信协议

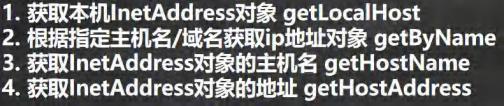


21.1.8 TCP 和 UDP



21.2 InetAddress 类

21.2.1 相关方法



21.2.2 应用案例 com.hspedu.api API\_.java

编写代码，获取计算机的主机名和 IP 地址相关 API

|  |
| --- |
| //获取本机 InetAddress 对象 getLocalHost  InetAddress localHost = InetAddress.getLocalHost();  System.out.println(localHost);  //根据指定主机名/域名获取 ip 地址对象 getByName  InetAddress host2 = InetAddress.getByName("ThinkPad-PC");  System.out.println(host2);  InetAddress host3 = InetAddress.getByName("www.hsp.com");  System.out.println(host3);  //获取 InetAddress 对象的主机名 getHostName  String host3Name = host3.getHostName();  System.out.println(host3Name);  //获取 InetAddress 对象的地址 getHostAddress  String host3Address = host3.getHostAddress();  System.out.println(host3Address); |

21.3 Socket

21.3.1 基本介绍

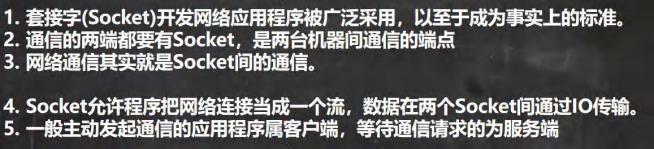
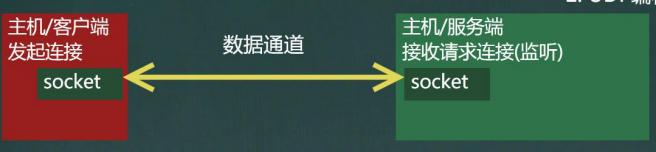
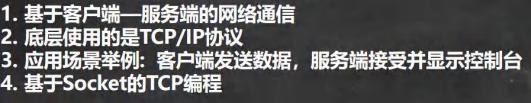


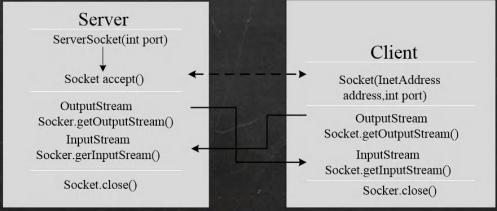
示意图:



21.4 TCP 网络通信编程

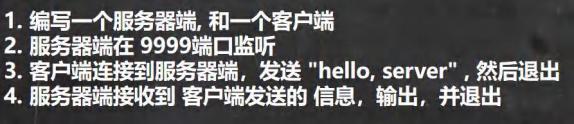
21.4.1 基本介绍





21.4.2 应用案例 1(使用字节流)

com.hspedu.socket SocketTCP01Server.java SocketTCP01Client.java



|  |
| --- |
| package com.hspedu.socket;  import java.io.IOException;  import java.io.InputStream;  import java.net.ServerSocket;  import java.net.Socket;  /\*\*  \* @author 韩顺平  \* @version 1.0 \* 服务端  \*/  public class SocketTCP01Server {  public static void main(String[] args) throws IOException {  //思路  //1. 在本机 的 9999 端口监听, 等待连接  // 细节: 要求在本机没有其它服务在监听 9999  // 细节：这个 ServerSocket 可以通过 accept() 返回多个 Socket[多个客户端连接服务器的并发]  ServerSocket serverSocket = new ServerSocket(9999); |

|  |
| --- |
| System.out.println("服务端，在 9999 端口监听，等待连接..");  //2. 当没有客户端连接 9999 端口时，程序会 阻塞, 等待连接  // 如果有客户端连接，则会返回 Socket对象，程序继续  Socket socket = serverSocket.accept();  System.out.println("服务端 socket =" + socket.getClass());  //  //3. 通过 socket.getInputStream() 读取客户端写入到数据通道的数据, 显示  InputStream inputStream = socket.getInputStream();  //4. IO 读取  byte[] buf= new byte[1024];  int readLen = 0;  while ((readLen = inputStream.read(buf)) != - 1) {  System.out.println(new String(buf, 0, readLen));//根据读取到的实际长度，显示内容.  }  //5.关闭流和 socket  inputStream.close();  socket.close();  serverSocket.close();//关闭  }  } |
| package com.hspedu.socket; |

|  |
| --- |
| import java.io.IOException;  import java.io.OutputStream;  import java.net.InetAddress;  import java.net.Socket;  import java.net.UnknownHostException;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 客户端，发送 "hello, server" 给服务端  \*/  public class SocketTCP01Client {  public static void main(String[] args) throws IOException {  //思路  //1. 连接服务端 (ip , 端口）  //解读: 连接本机的 9999 端口, 如果连接成功，返回 Socket 对象  Socket socket = new Socket(InetAddress.getLocalHost(), 9999);  System.out.println("客户端 socket 返回=" + socket.getClass());  //2. 连接上后，生成 Socket, 通过 socket.getOutputStream()  // 得到 和 socket 对象关联的输出流对象  OutputStream outputStream = socket.getOutputStream();  //3. 通过输出流，写入数据到 数据通道  outputStream.write("hello, server".getBytes());  //4. 关闭流对象和 socket, 必须关闭  outputStream.close();  socket.close(); |

|  |
| --- |
| System.out.println("客户端退出.....");  }  } |

21.4.3 应用案例 2(使用字节流) SocketTCP02.java



|  |
| --- |
| package com.hspedu.socket;  import java.io.IOException;  import java.io.InputStream;  import java.io.OutputStream;  import java.net.ServerSocket;  import java.net.Socket;  /\*\*  \* @author 韩顺平  \* @version 1.0 \* 服务端  \*/  @SuppressWarnings({"all"})  public class SocketTCP02Server {  public static void main(String[] args) throws IOException { |

|  |
| --- |
| //思路  //1. 在本机 的 9999 端口监听, 等待连接  // 细节: 要求在本机没有其它服务在监听 9999  // 细节：这个 ServerSocket 可以通过 accept() 返回多个 Socket[多个客户端连接服务器的并发]  ServerSocket serverSocket = new ServerSocket(9999);  System.out.println("服务端，在 9999 端口监听，等待连接..");  //2. 当没有客户端连接 9999 端口时，程序会 阻塞, 等待连接  // 如果有客户端连接，则会返回 Socket对象，程序继续  Socket socket = serverSocket.accept();  System.out.println("服务端 socket =" + socket.getClass());  //  //3. 通过 socket.getInputStream() 读取客户端写入到数据通道的数据, 显示  InputStream inputStream = socket.getInputStream();  //4. IO 读取  byte[] buf= new byte[1024];  int readLen = 0;  while ((readLen = inputStream.read(buf)) != - 1) {  System.out.println(new String(buf, 0, readLen));//根据读取到的实际长度，显示内容.  }  //5. 获取 socket 相关联的输出流  OutputStream outputStream = socket.getOutputStream();  outputStream.write("hello, client".getBytes());  // 设置结束标记  socket.shutdownOutput(); |

|  |
| --- |
| //6.关闭流和 socket  outputStream.close();  inputStream.close();  socket.close();  serverSocket.close();//关闭  }  } |
| package com.hspedu.socket;  import java.io.IOException;  import java.io.InputStream;  import java.io.OutputStream;  import java.net.InetAddress;  import java.net.Socket;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 客户端，发送 "hello, server" 给服务端  \*/  @SuppressWarnings({"all"})  public class SocketTCP02Client {  public static void main(String[] args) throws IOException { |

|  |
| --- |
| //思路  //1. 连接服务端 (ip , 端口）  //解读: 连接本机的 9999 端口, 如果连接成功，返回 Socket 对象  Socket socket = new Socket(InetAddress.getLocalHost(), 9999);  System.out.println("客户端 socket 返回=" + socket.getClass());  //2. 连接上后，生成 Socket, 通过 socket.getOutputStream()  // 得到 和 socket 对象关联的输出流对象  OutputStream outputStream = socket.getOutputStream();  //3. 通过输出流，写入数据到 数据通道  outputStream.write("hello, server".getBytes());  // 设置结束标记  socket.shutdownOutput();  //4. 获取和 socket 关联的输入流. 读取数据(字节) ，并显示  InputStream inputStream = socket.getInputStream();  byte[] buf= new byte[1024];  int readLen = 0;  while ((readLen = inputStream.read(buf)) != - 1) {  System.out.println(new String(buf, 0, readLen));  }  //5. 关闭流对象和 socket, 必须关闭  inputStream.close();  outputStream.close();  socket.close();  System.out.println("客户端退出....."); |

|  |
| --- |
| }  } |

21.4.4 应用案例 3(使用字符流)



|  |
| --- |
| package com.hspedu.socket;  import java.io.\*;  import java.net.ServerSocket;  import java.net.Socket;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 服务端, 使用字符流方式读写  \*/  @SuppressWarnings({"all"})  public class SocketTCP03Server {  public static void main(String[] args) throws IOException {  //思路  //1. 在本机 的 9999 端口监听, 等待连接  // 细节: 要求在本机没有其它服务在监听 9999 |

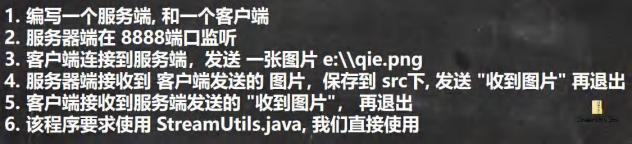
|  |
| --- |
| // 细节：这个 ServerSocket 可以通过 accept() 返回多个 Socket[多个客户端连接服务器的并发]  ServerSocket serverSocket = new ServerSocket(9999);  System.out.println("服务端，在 9999 端口监听，等待连接..");  //2. 当没有客户端连接 9999 端口时，程序会 阻塞, 等待连接  // 如果有客户端连接，则会返回 Socket对象，程序继续  Socket socket = serverSocket.accept();  System.out.println("服务端 socket =" + socket.getClass());  //  //3. 通过 socket.getInputStream() 读取客户端写入到数据通道的数据, 显示  InputStream inputStream = socket.getInputStream();  //4. IO 读取, 使用字符流, 老师使用 InputStreamReader 将 inputStream 转成字符流  BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(inputStream)); String s = bufferedReader.readLine();  System.out.println(s);//输出  //5. 获取 socket 相关联的输出流  OutputStream outputStream = socket.getOutputStream();  // 使用字符输出流的方式回复信息  BufferedWriter bufferedWriter = new BufferedWriter(new OutputStreamWriter(outputStream));  bufferedWriter.write("hello client 字符流");  bufferedWriter.newLine();// 插入一个换行符，表示回复内容的结束  bufferedWriter.flush();//注意需要手动的 flush |

|  |
| --- |
| //6.关闭流和 socket  bufferedWriter.close();  bufferedReader.close();  socket.close();  serverSocket.close();//关闭  }  } |
| package com.hspedu.socket;  import java.io.\*;  import java.net.InetAddress;  import java.net.Socket;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 客户端，发送 "hello, server" 给服务端， 使用字符流  \*/  @SuppressWarnings({"all"})  public class SocketTCP03Client {  public static void main(String[] args) throws IOException {  //思路  //1. 连接服务端 (ip , 端口）  //解读: 连接本机的 9999 端口, 如果连接成功，返回 Socket 对象 |

|  |
| --- |
| Socket socket = new Socket(InetAddress.getLocalHost(), 9999);  System.out.println("客户端 socket 返回=" + socket.getClass());  //2. 连接上后，生成 Socket, 通过 socket.getOutputStream()  // 得到 和 socket 对象关联的输出流对象  OutputStream outputStream = socket.getOutputStream();  //3. 通过输出流，写入数据到 数据通道, 使用字符流  BufferedWriter bufferedWriter = new BufferedWriter(new OutputStreamWriter(outputStream));  bufferedWriter.write("hello, server 字符流");  bufferedWriter.newLine();//插入一个换行符，表示写入的内容结束, 注意，要求对方使用 readLine()!!!!  bufferedWriter.flush();// 如果使用的字符流，需要手动刷新，否则数据不会写入数据通道  //4. 获取和 socket 关联的输入流. 读取数据(字符) ，并显示  InputStream inputStream = socket.getInputStream();  BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(inputStream)); String s = bufferedReader.readLine();  System.out.println(s);  //5. 关闭流对象和 socket, 必须关闭  bufferedReader.close();//关闭外层流  bufferedWriter.close();  socket.close();  System.out.println("客户端退出.....");  }  } |

21.4.5 应用案例 4 TCPFileUploadServer.java TCPFileUploadClient.java

com.hspedu.upload



|  |
| --- |
| package com.hspedu.upload;  import java.io.BufferedReader;  import java.io.ByteArrayOutputStream;  import java.io.IOException;  import java.io.InputStream;  import java.io.InputStreamReader;  /\*\*  \* 此类用于演示关于流的读写方法  \*  \*/  public class StreamUtils {  /\*\*  \* 功能：将输入流转换成 byte[]  \* @param is  \* @return  \* @throws Exception  \*/ |

|  |
| --- |
| public static byte[] streamToByteArray(InputStream is) throws Exception{  ByteArrayOutputStream bos = new ByteArrayOutputStream();//创建输出流对象  byte[] b = new byte[1024];  int len;  while((len=is.read(b))!=- 1){  bos.write(b, 0, len);  }  byte[] array = bos.toByteArray();  bos.close();  return array;  }  /\*\*  \* 功能：将 InputStream 转换成 String  \* @param is  \* @return  \* @throws Exception  \*/  public static String streamToString(InputStream is) throws Exception{  BufferedReader reader = new BufferedReader(new InputStreamReader(is));  StringBuilder builder= new StringBuilder();  String line;  while((line=reader.readLine())!=null){ //当读取到 null 时，就表示结束  builder.append(line+"\r\n");  }  return builder.toString(); |

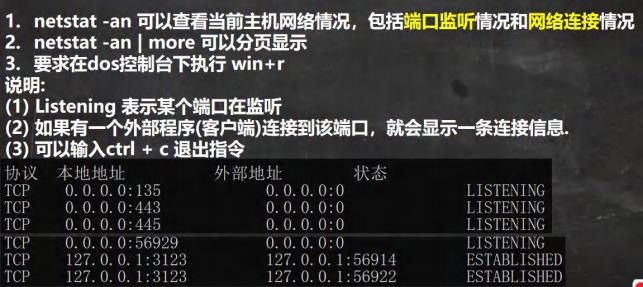
|  |
| --- |
| }  } |
| package com.hspedu.upload;  import java.io.\*;  import java.net.ServerSocket;  import java.net.Socket;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 文件上传的服务端  \*/  public class TCPFileUploadServer {  public static void main(String[] args) throws Exception {  //1. 服务端在本机监听 8888 端口  ServerSocket serverSocket = new ServerSocket(8888);  System.out.println("服务端在 8888 端口监听....");  //2. 等待连接  Socket socket = serverSocket.accept(); |

|  |
| --- |
| //3. 读取客户端发送的数据  // 通过 Socket 得到输入流  BufferedInputStream bis = new BufferedInputStream(socket.getInputStream());  byte[] bytes = StreamUtils.streamToByteArray(bis);  //4. 将得到 bytes 数组，写入到指定的路径，就得到一个文件了  String destFilePath = "src\\abc.mp4";  BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(destFilePath));  bos.write(bytes);  bos.close();  // 向客户端回复 "收到图片"  // 通过 socket 获取到输出流(字符)  BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  writer.write("收到图片");  writer.flush();//把内容刷新到数据通道  socket.shutdownOutput();//设置写入结束标记  //关闭其他资源  writer.close();  bis.close();  socket.close();  serverSocket.close();  }  } |
| package com.hspedu.upload; |

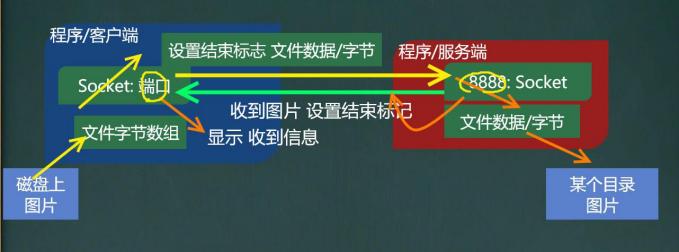
|  |
| --- |
| import java.io.\*;  import java.net.InetAddress;  import java.net.Socket;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 文件上传的客户端  \*/  public class TCPFileUploadClient {  public static void main(String[] args) throws Exception {  //客户端连接服务端 8888 ，得到 Socket 对象  Socket socket = new Socket(InetAddress.getLocalHost(), 8888);  //创建读取磁盘文件的输入流  //String filePath = "e:\\qie.png";  String filePath = "e:\\abc.mp4";  BufferedInputStream bis = new BufferedInputStream(new FileInputStream(filePath));  //bytes 就是 filePath 对应的字节数组  byte[] bytes = StreamUtils.streamToByteArray(bis);  //通过 socket 获取到输出流, 将 bytes 数据发送给服务端  BufferedOutputStream bos = new BufferedOutputStream(socket.getOutputStream()); |

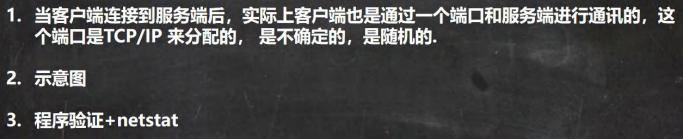
|  |
| --- |
| bos.write(bytes);//将文件对应的字节数组的内容，写入到数据通道  bis.close();  socket.shutdownOutput();//设置写入数据的结束标记  //=====接收从服务端回复的消息=====  InputStream inputStream = socket.getInputStream();  //使用 StreamUtils 的方法，直接将 inputStream 读取到的内容 转成字符串  String s = StreamUtils.streamToString(inputStream);  System.out.println(s);  //关闭相关的流  inputStream.close();  bos.close();  socket.close();  }  } |

21.4.6 netstat 指令



21.4.7 TCP 网络通讯不为人知的秘密





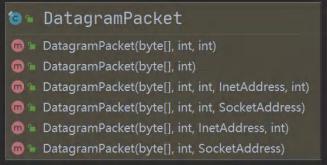
21.5 UDP 网络通信编程[了解]

21.5.1 基本介绍

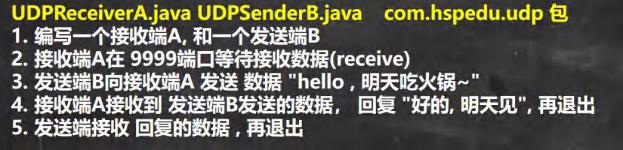


21.5.2 基本流程





21.5.3 应用案例



|  |
| --- |
| package com.hspedu.udp; |

|  |
| --- |
| import java.io.IOException;  import java.net.DatagramPacket;  import java.net.DatagramSocket;  import java.net.InetAddress;  import java.net.SocketException;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* UDP 接收端  \*/  public class UDPReceiverA {  public static void main(String[] args) throws IOException {  //1. 创建一个 DatagramSocket 对象，准备在 9999 接收数据  DatagramSocket socket = new DatagramSocket(9999);  //2. 构建一个 DatagramPacket 对象，准备接收数据  // 在前面讲解 UDP 协议时，老师说过一个数据包最大 64k  byte[] buf= new byte[1024];  DatagramPacket packet = new DatagramPacket(buf, buf.length);  //3. 调用 接收方法, 将通过网络传输的 DatagramPacket 对象  // 填充到 packet 对象  //老师提示: 当有数据包发送到 本机的 9999 端口时，就会接收到数据  // 如果没有数据包发送到 本机的 9999 端口, 就会阻塞等待.  System.out.println("接收端 A 等待接收数据..");  socket.receive(packet); |

|  |
| --- |
| //4. 可以把 packet 进行拆包，取出数据，并显示.  int length = packet.getLength();//实际接收到的数据字节长度  byte[] data = packet.getData();//接收到数据  String s = new String(data, 0, length);  System.out.println(s);  //===回复信息给 B 端  //将需要发送的数据，封装到 DatagramPacket 对象  data = "好的, 明天见".getBytes();  //说明: 封装的 DatagramPacket 对象 data 内容字节数组 , data.length , 主机(IP) , 端口  packet =  new DatagramPacket(data, data.length, InetAddress.getByName(" 192.168.12.1"), 9998);  socket.send(packet);//发送  //5. 关闭资源  socket.close();  System.out.println("A 端退出...");  }  } |
| package com.hspedu.udp; |

|  |
| --- |
| import java.io.IOException;  import java.net.\*;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 发送端 B====> 也可以接收数据  \*/  @SuppressWarnings({"all"})  public class UDPSenderB {  public static void main(String[] args) throws IOException {  //1.创建 DatagramSocket 对象，准备在 9998 端口 接收数据  DatagramSocket socket = new DatagramSocket(9998);  //2. 将需要发送的数据，封装到 DatagramPacket 对象  byte[] data = "hello 明天吃火锅~".getBytes(); //  //说明: 封装的 DatagramPacket 对象 data 内容字节数组 , data.length , 主机(IP) , 端口  DatagramPacket packet =  new DatagramPacket(data, data.length, InetAddress.getByName(" 192.168.12.1"), 9999);  socket.send(packet);  //3.=== 接收从 A 端回复的信息  //(1) 构建一个 DatagramPacket 对象，准备接收数据 |

|  |
| --- |
| // 在前面讲解 UDP 协议时，老师说过一个数据包最大 64k  byte[] buf= new byte[1024];  packet = new DatagramPacket(buf, buf.length);  //(2) 调用 接收方法, 将通过网络传输的 DatagramPacket 对象  // 填充到 packet 对象  //老师提示: 当有数据包发送到 本机的 9998 端口时，就会接收到数据  // 如果没有数据包发送到 本机的 9998 端口, 就会阻塞等待.  socket.receive(packet);  //(3) 可以把 packet 进行拆包，取出数据，并显示.  int length = packet.getLength();//实际接收到的数据字节长度  data = packet.getData();//接收到数据  String s = new String(data, 0, length);  System.out.println(s);  //关闭资源  socket.close();  System.out.println("B 端退出");  }  } |

21.6 本章作业



第 22 章多用户即时通信系统

22. 1 QQ 聊天项目演示

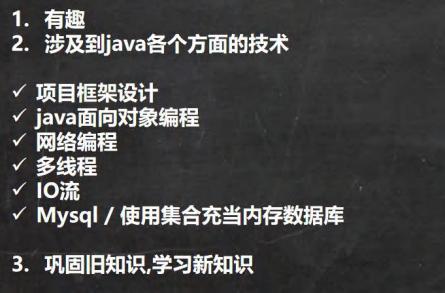
22.1.1 项目 QQ 演示







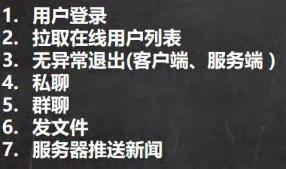
22.2 为什么选择这个项目



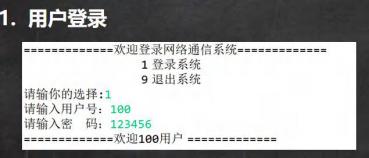
22.2.1 项目开发流程

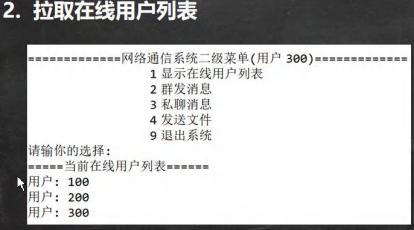


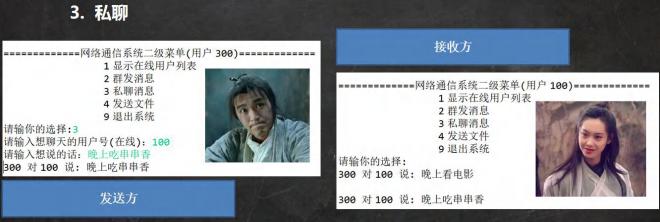
22.2.2 需求分析



22.2.3 界面设计

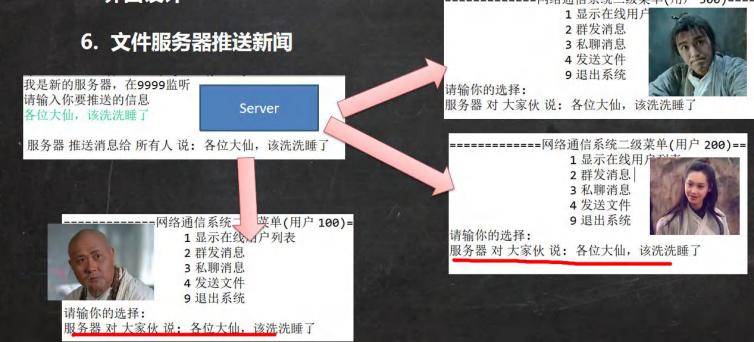




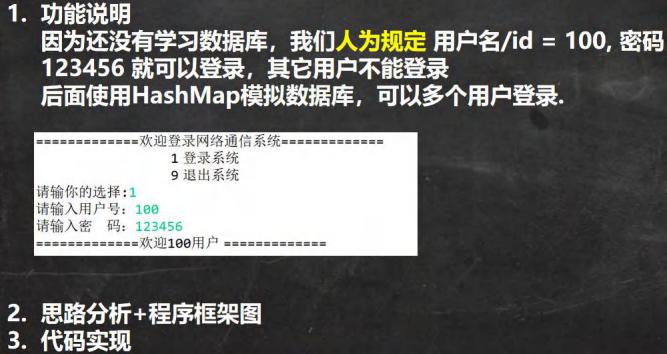


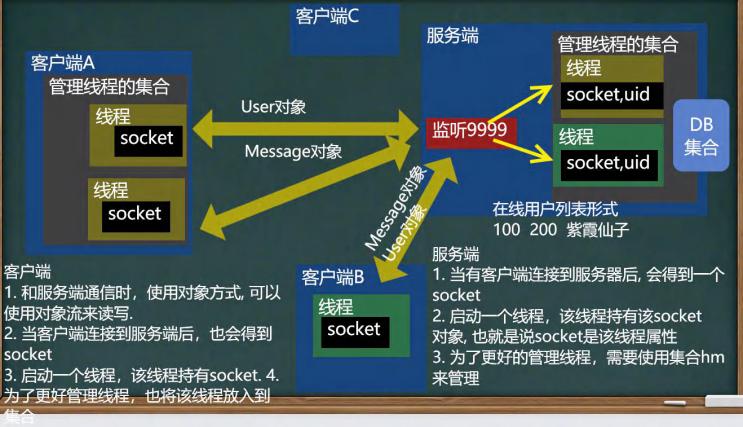




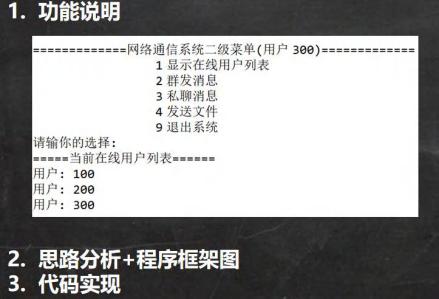


22.2.4 功能实现-用户登录

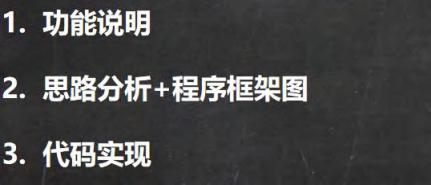




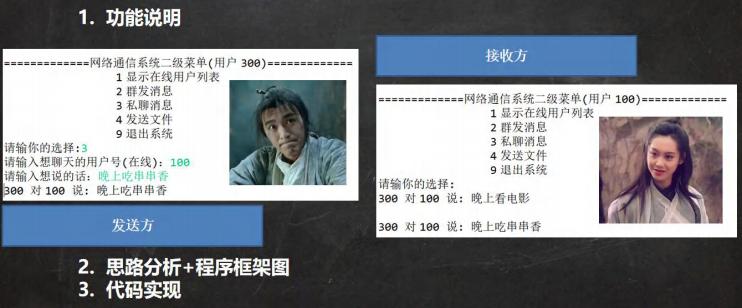
22.2.5 功能实现-拉取在线用户列表

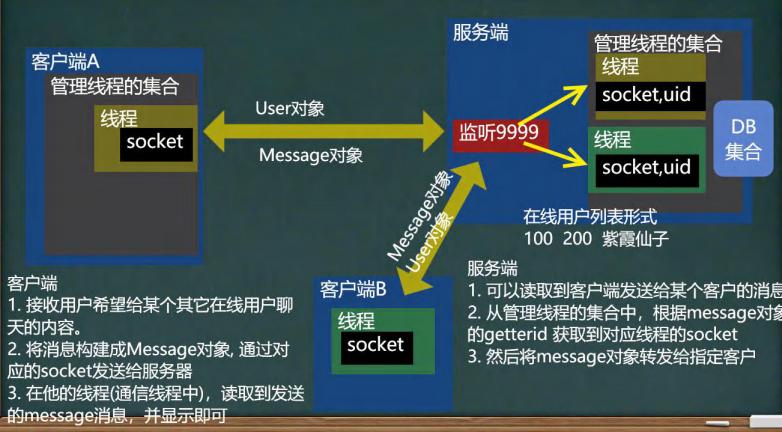


22.2.6 功能实现-无异常退出

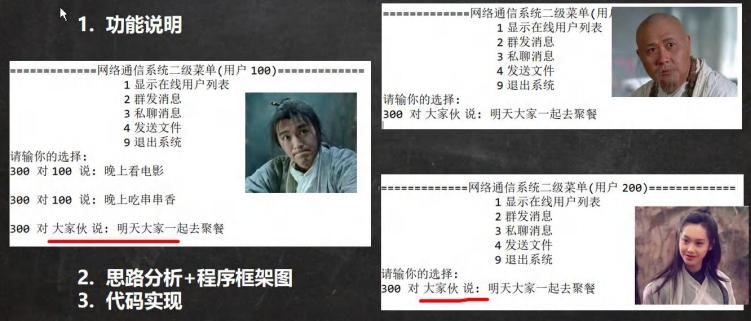


22.2.7 功能实现-私聊

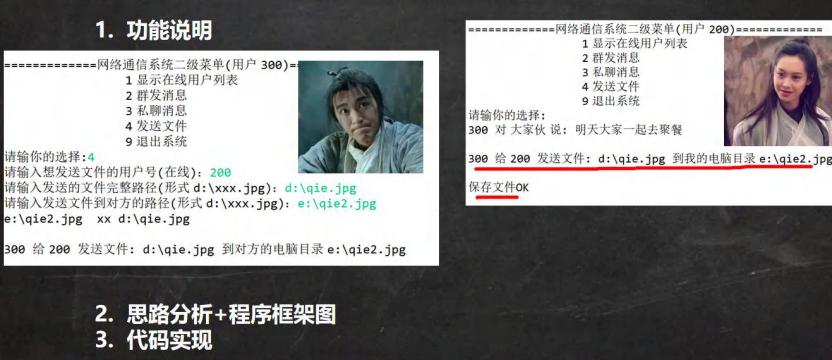


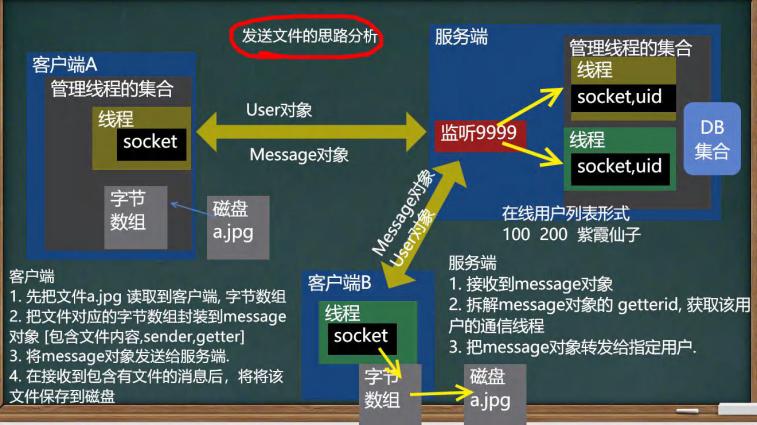


22.2.8 功能实现-群聊

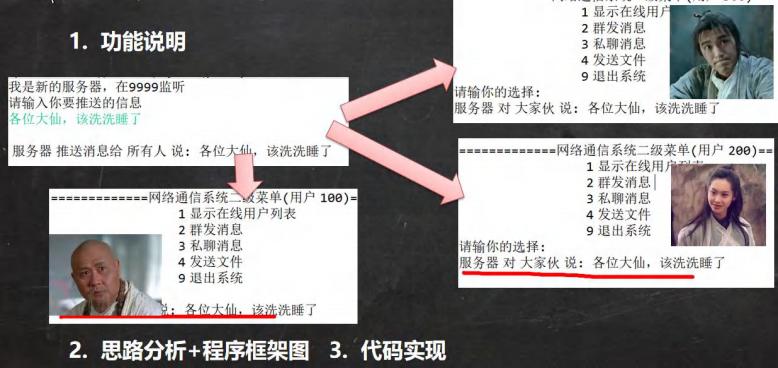


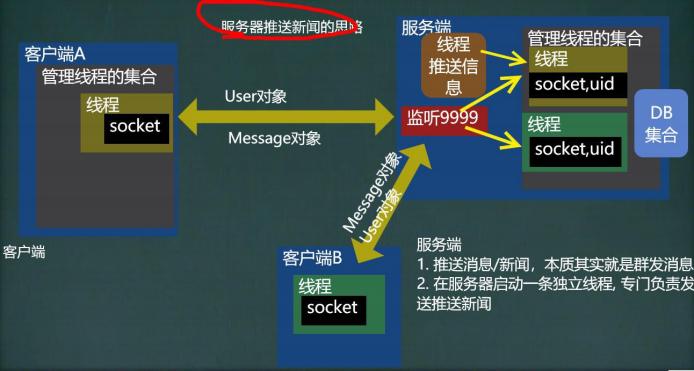
22.2.9 功能说明-发文件



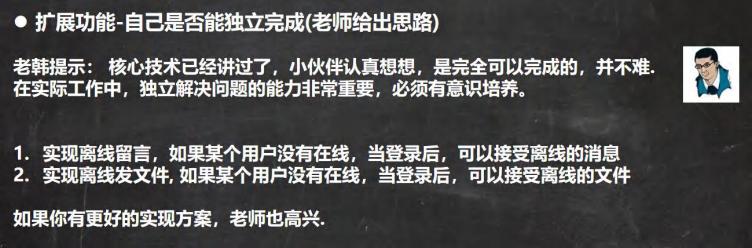


22.2. 10 功能实现-服务器推送新闻





22.2. 11 多用户即时通信系统



**老韩强烈建议**：一定要写出来 , 一定动脑筋.