# 综合案例

此前我们已经练习了根据集合当中的字符串对象读写文件，而本综合案例主要练习根据集合当中的**自定义对象**来读写文件。

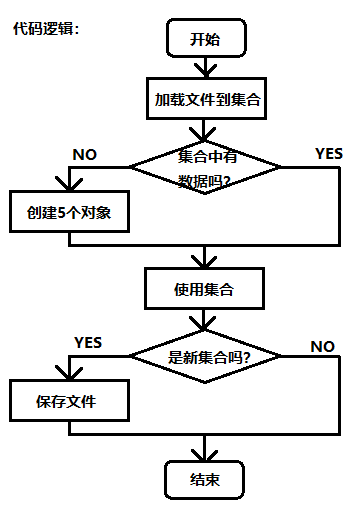
## 场景介绍

很多网络游戏当中都有组队模式，例如魔兽世界、DotA、英雄联盟（LOL）、王者荣耀等，均为5人组队进行游戏。即使在现实生活中，打麻将、斗地主也都是多人进行的游戏。那么对于多人组队的游戏，典型的场景为：

1. 有5个人都在玩游戏；
2. 这5个人组成一个队伍；
3. 这支队伍进行游戏；
4. 队伍成员彼此欣赏，决定以后还要组队一起玩；
5. 一段时间后大家再次组队游戏。

## 整体思路

1. 看看有没有已经认识的老朋友；
2. 如果没有，则找到5个新朋友，组成队伍；如果有，则队伍中直接为5个老朋友；
3. 开始游戏，例如阵容展示、战斗力计算等；
4. 如果这是新队伍，则保存5个新朋友的信息；如果这是老队伍，则无需操作；
5. 结束游戏。



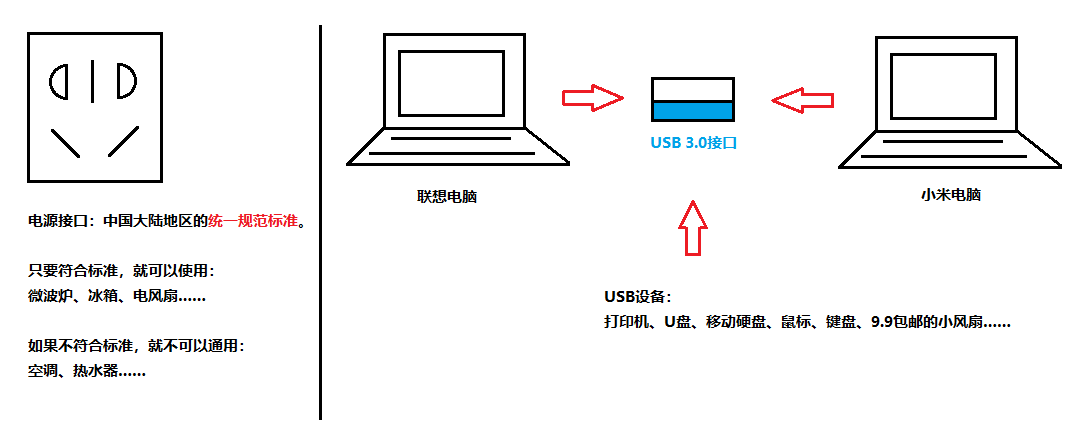
## 代码实现

具体的实现请参考相关源代码。但请注意，一定要首先把握整体流程，编写框架代码之后，再逐步编写具体的逻辑实现。

# 接口

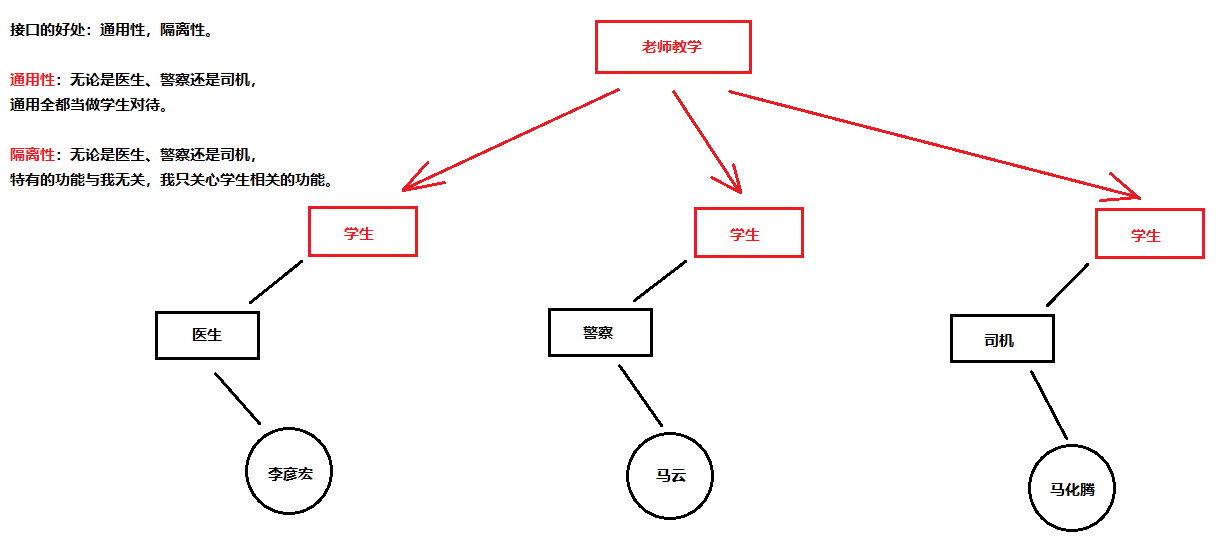
## 现实生活中的接口举例

电源接口、USB接口，都具有一定范围之内的通用性。只要符合接口的标准，那么所有的设备都无需互相商量，可以享受通用性带来的遍历。



## 面向对象中的接口思想

接口是一种针对对象的公共约束力，可以将对象中的成员方法“剥离”出来，于是通过接口就可以只关心特定的一部分方法，而不关心接口之外的其他方法。



## 接口的格式与组成部分

在Java 9版本中，接口的定义格式为：

**public interface 接口名称 {**

**// ...**

**}**

其中可以包含的组成部分有：

1. 抽象方法
2. 常量
3. 默认方法（Java 8）
4. 静态方法（Java 8）
5. 私有方法（Java 9）

今天我们主要学习接口中的**抽象方法**，其他的组成部分今后学习。

## 接口的抽象方法定义

抽象方法的定义格式：

**public abstract 返回值类型 方法名称(参数类型 参数名称);**

注意：

1. 接口中的抽象方法，修饰如果写必须是：public abstract
2. 接口中的抽象方法，修饰符可以省略不写，默认就是：public abstract
3. 抽象方法只有方法头，不能有方法体大括号。

## 接口实现类的定义

格式：

**public class 实现类名称 implements 接口名称 {**

**//覆盖重写所有抽象方法**

**}**

覆盖重写抽象方法的步骤：

1. 将接口当中的抽象方法抄写过来
2. 去掉abstract关键字
3. 写上大括号方法体

## 接口与实现类的使用

创建格式：

**接口名称 引用名 = new 实现类名称();**

调用格式：

**引用名.抽象方法名(参数);**

注意事项：

1. 左边是接口类型，那么只能调用接口当中定义好的内容，不能调用右侧实现类当中特有的内容。（接口隔离）
2. 当调用接口当中的抽象方法时，真正进行运行的是右侧new的时候类的具体方法内容。
3. 总结记住一句话：**调用的时候看左边，运行的时候看右边。**

# Lambda

## 冗余的接口实现类

接口在Java开发当中的作用非常重要。但有的时候为了使用接口而不得不去创建一个实现类对象，这种操作其实没有必要。

例如接口：

public interface Calculator {

int sum(int a, int b);

}

对应的实现类：

**public** **class** CalculatorImpl **implements** Calculator {

**public** **int** sum(**int** a, **int** b) {

**int** result = a + b;

**return** result;

}

}

使用接口的时候需要创建实现类对象才能使用：

**public** **class** Demo01Calc {

**public** **static** **void** main(String[] args) {

// 首先使用接口的格式来创建了一个计算器对象

Calculator calculator = **new** CalculatorImpl();

// 将计算器对象交给method方法去使用

*method*(calculator);

}

// 参数是接口类型：计算器接口

**public** **static** **void** method(Calculator calculator) {

**int** result = calculator.sum(10, 20);

System.***out***.println("结果是：" + result);

}

}

## 体验Lambda表达式

使用Lambda，在接口不变的情况下，不需要实现类，直接就可以使用接口，而且写法非常简单：

**public** **class** Demo02CalcLambda {

**public** **static** **void** main(String[] args) {

*method*((x, y) -> x + y);

}

**public** **static** **void** method(Calculator calculator) {

**int** result = calculator.sum(100, 200);

System.***out***.println("结果是：" + result);

}

}

## 理解Lambda的语义

Lambda表达式：**(x, y) -> x + y**。方法需要一个Calculator接口类型的参数，而Lambda表达式就是充当了Calculator接口类型的参数。理解：

1. Lambda表达式前面的小括号，其实就是接口抽象方法的小括号。
2. 箭头代表拿着小括号的数据做什么事情，是一个指向的动作。
3. 箭头后面就代表拿到了参数之后做什么事情。

## 函数式接口

Java当中使用Lambda表达式的前提是：必须有“函数式接口”。

**概念**：有且仅有一个抽象方法的接口，叫做函数式接口。

可以使用可选的**@FunctionalInterface**来检测接口是否为函数式接口，但是有没有这个注解并不影响函数式接口的定义，这只是一种**可选的检测手段**。

@FunctionalInterface

**public** **interface** Calculator {

**int** sum(**int** a, **int** b);

}

## Lambda的标准格式

Lambda的格式就是为了将抽象方法，翻译成为以下**三点**：

1. 一些参数（方法参数）
2. 一个箭头
3. 一些代码（方法体，大括号）

例如将抽象方法：public abstract int sum(int a, int b);

翻译成为：(int a, int b) -> { return a + b; }

## Lambda的上下文推断

Lambda表达式要想使用，一定要有函数式接口的推断环境：

1. 要么通过方法的参数类型来确定是哪个函数式接口
2. 要么通过赋值操作来确定是哪个函数式接口

**public** **class** Demo03Lambda {

**public** **static** **void** main(String[] args) {

// 调用方法的时候，参数类型是函数式接口，所以Lambda可以推断出来是哪个接口

*method*( (**int** a, **int** b) -> { **return** a + b;} );

System.***out***.println("======================");

// 也可以根据赋值语句左侧的类型来进行Lambda上下文推断

Calculator param = (**int** a, **int** b) -> { **return** a + b; };

*method*(param);

// 错误写法！没有上下文环境，Lambda就无法推断是哪个函数式接口

// (int a, int b) -> { return a + b; };

}

**public** **static** **void** method(Calculator calculator) {

**int** result = calculator.sum(100, 200);

System.***out***.println("结果是：" + result);

}

}

## Lambda的简便格式

在Lambda表达式当中，凡是可以推导的，都是可以省略的。

1. Lambda表达式当中的参数类型可以省略不写。
2. 如果参数有且只有一个，那么小括号可以省略。
3. 如果语句只有一个，那么大括号和return也可以省略。

**public** **class** DemoLambdaSimple {

**public** **static** **void** main(String[] args) {

// 标准格式：

*method*( (**int** x) -> { **return** ++x; } );

// 省略参数类型：

*method*( (x) -> { **return** ++x; } );

// 省略参数小括号：

*method*( x -> { **return** ++x; } );

// 省略大括号和return关键字：

*method*( x -> ++x );

}

**public** **static** **void** method(MyInter inter) {

**int** result = inter.singlePlus(10);

System.***out***.println("结果：" + result);

}

}

# 知识总结

1. 使用接口的好处：通用性、隔离性。
2. 定义接口的格式：  
   public interface 接口名 {  
    // ...  
   }
3. 接口的组成部分：
   1. 抽象方法【核心内容】
   2. 常量
   3. 默认方法（Java 8）
   4. 静态方法（Java 8）
   5. 私有方法（Java 9）
4. 定义一个抽象方法，格式：  
   public abstract 返回值类型 方法名(参数类型 参数名);
   1. 修饰符public abstract也可以省略不写
   2. 抽象方法不可以写方法体大括号
5. 接口要想使用，一定要有一个实现类。如何实现？  
   public class 类名称 implements 接口名 {  
    // 覆盖重写所有的抽象方法  
   }
6. 如何覆盖重写（Override）抽象方法呢？一般步骤：
   1. 将抽象方法抄写过来
   2. 去掉abstract关键字
   3. 写上方法体
7. 使用接口和实现类的一般格式：
   1. 创建：接口名称 引用名 = new 实现类名称();
   2. 调用：引用名.抽象方法名(参数);
   3. 注意：
      1. 只能调用左侧接口当中定义好的方法，无法调用右侧类当中特有的方法。
      2. 运行程序的时候，方法的执行者其实是右边的类对象。
      3. “调用的时候看左边，但是运行的时候看右边。”
8. 从Java 8开始，没有接口的实现类，也可以直接使用接口：Lambda表达式。Lambda表达式就替代了实现类。
9. Lambda使用前提：一定要有函数式接口才能用，没有函数式接口就不能用Lambda表达式。
10. 函数式接口：有且仅有一个抽象方法的接口。无所谓有没有@FunctionalInterface，这是一个可选的检测手段而已。
11. 使用Lambda必须有推断环境：
    1. 要么根据参数类型传参来推断函数式接口
    2. 要么就根据赋值语句左侧类型来推断函数式接口
12. Lambda表达式的标准格式：(int a, int b) -> { return a + b;}
    1. 一些参数（方法的参数）
    2. 一个箭头
    3. 一些代码（方法体）
13. Lambda表达式的简便格式：  
    (int num) -> {return ++num;}  
    (num) -> {return ++num;}  
    num -> {return ++num;}  
    num -> ++num;
    1. 参数的类型可以省略
    2. 如果有且仅有一个参数，那么小括号可以省略
    3. 如果有且仅有一个语句，那么大括号和return也可以省略