接 口

> Author：zhangzhang·
>
> Version：1.0.0

一、接口语法

1.1 基本使用

> 概念：接口相当于特殊的抽象类，定义方式、组成部分与抽象类类似。使用interface关键字定义接口。

> 特点：
>
> - 没有构造方法，不能创建对象。
> - 只能定义：公开静态常量、公开抽象方法。

> 案例演示：自定义接口。

```
public interface MyInterface {
  //公开的静态常量
  //public static final String NAME="接口1";
  String NAME="接口1";
  //公开的抽象方法
  //public abstract void method1();
  void method();
}
```

> 实现类：

```
public class Impl implements MyInterface1{

  @Override
  public void method() {
    System.out.println("method");
  }
}
```

> 测试类：

```
public class TestInterface {
  public static void main(String[] args) {
    //new MyInterface1();
    MyInterface1 myInterface1=new Impl();
    myInterface1.method();
  }
}
```

1.2 和抽象类区别

> 相同点：
>
> - 可编译成字节码文件。
> - 不能创建对象。

- 可以作为引用类型。
- 具备Object类中所定义的方法。

不同点：

- 所有属性都是公开静态常量，隐式使用public static final修饰。
- 所有方法都是公开抽象方法，隐式使用public abstract修饰。
- 没有构造方法、动态代码块、静态代码块。

二、微观接口【重点】

微观概念：接口是一种能力和约定。

- 接口的定义：代表了某种能力。
- 方法的定义：能力的具体要求。

经验：

- Java为单继承，当父类的方法种类无法满足子类需求时，可实现接口扩充子类能力。
- 接口支持多实现，可为类扩充多种能力。

案例演示：实现类实现多个接口。

```
public interface Flyable {
  void fly();
}
```

```
public interface Fireable {
  void fire();
}
```

实现类：Person

```
public class Person implements Flyable,Fireable{
  String name;
  int age;
  public Person() {
    // TODO Auto-generated constructor stub
  }

  public Person(String name, int age) {
    super();
    this.name = name;
    this.age = age;
  }

  public void eat() {
    System.out.println(name+"吃东西...");
  }
  public void sleep() {
    System.out.println(name+"睡了...");
  }
  @Override
  public void fly() {
    System.out.println(name+"开始飞了...");
  }

  @Override
  public void fire() {
    System.out.println(name+"可以喷火了...");
  }
}
```

测试类：

```
public class TestPerson1 {
  public static void main(String[] args) {
    Person xiaoming=new Person("小明",20);
    xiaoming.fly();
    xiaoming.fire();

  }
}
```

三、接口规范

> - 任何类在实现接口时，必须实现接口中所有的抽象方法，否则此类为抽象类。
> - 实现接口中的抽象方法时，访问修饰符必须是public。
> - 同父类一样，接口也可声明为引用，并指向实现类对象。

注：

- **仅可调用接口中所声明的方法，不可调用实现类中独有的方法。**
- **可强转回实现类本身类型，进行独有方法调用。**

```
public class TestPerson2 {
  public static void main(String[] args) {

    Flyable flyable=new Person("小张",22);
    flyable.fly();


    Fireable fireable=new Person("小李", 15);
    fireable.fire();
  }
}
```

四、接口实现多态

> 接口实现多态：使用接口作为方法的参数和返回值，实际赋值实现类对象实现多态。

> 案例演示：

```
public interface Runnable{
  //跑
    void run();
}
```

```
public interface Swimable {
  //游泳
    void swim();
}
```

```
public abstract class Animal {
  public void eat() {
    System.out.println("吃");
  }
  public void sleep() {
    System.out.println("睡");
  }
}
```

```
public class Dog extends Animal implements Runnable,Swimable{

  public void shout() {
    System.out.println("狗狗开始叫...");
  }

  @Override
  public void swim() {
    System.out.println("狗狗游泳...");
  }

  @Override
  public void run() {
    System.out.println("狗狗跑步...");
  }
}
```

```
public class TestDog {
  public static void main(String[] args) {
    Dog wangcai=new Dog();
    Animal a=wangcai;
    Runnable runnable=wangcai;
    Swimable swimable=wangcai;

    //---------调用方法-----------
        wangcai.shout();
```

```
        a.eat();
        a.sleep();
    runnable.run();
    swimable.swim();

  }
}
```

五、接口常见关系

> 类与类：
>
> - 单继承
> - extends 父类名称
>
> 类与接口：
>
> - 多实现
> - implements 接口名称1 , 接口名称2 , 接口名称n
>
> 接口与接口：
>
> - 多继承
> - extends 父接口1 , 父接口2 , 父接口n

> 案例演示：接口多继承。

```
public interface Runnable extends Serializable ,Cloneable{
  //跑
    void run();
}
```

六、常量接口

> 将多个常用于表示状态或固定值的变量，以静态常量的形式定义在接口中统一管理，提高代码可读性。

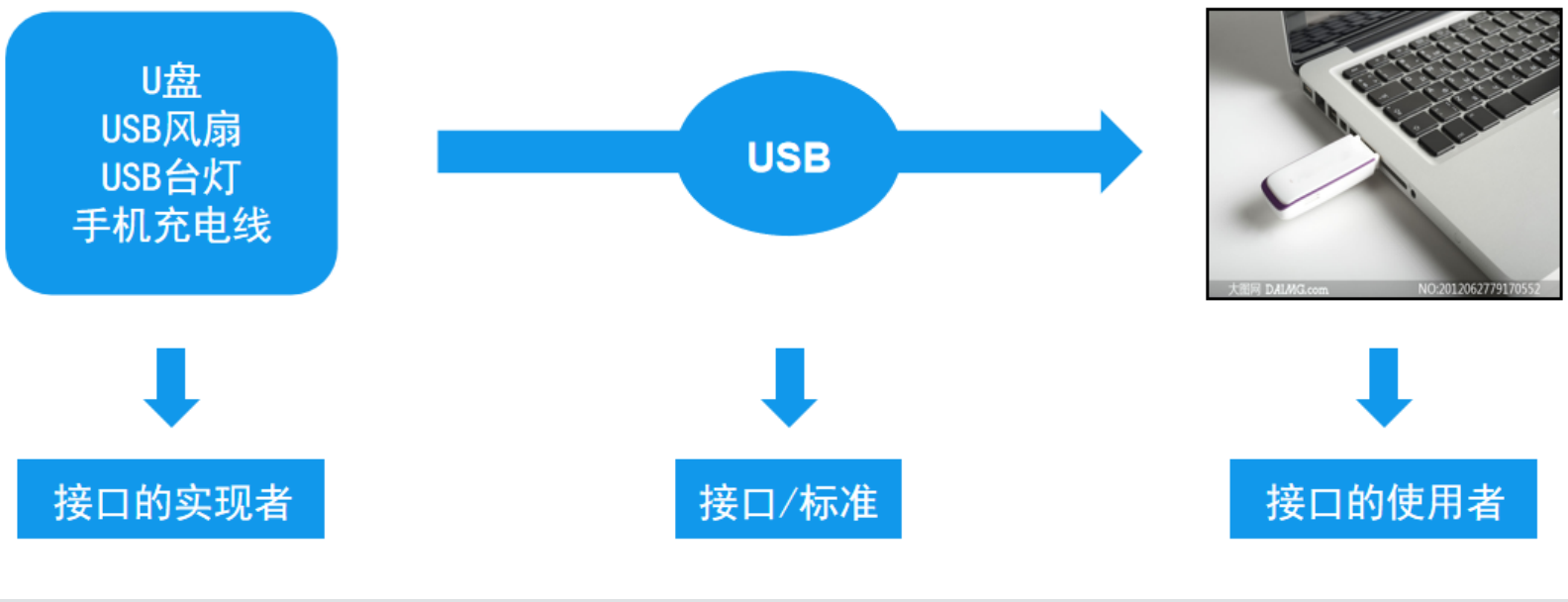
```
public interface ConstInterface {
  String CONST1 = "aaa";
  String CONST2 = "bbb";
  String CONST3 = "ccc";
}
```

```
public class TestConstInterface {
  public static void main(String[] args) {
    if(ConstInterface.CONST1.equals("aaa")) {
      System.out.println();
    }
  }
}
```

七、宏观接口【 重点 】

7.1 概念

> 宏观概念：接口是一种标准、规范。

生活中接口


> 案例演示：

```java
public interface Usb {
  void service();
}
```

```java
public class Mouse implements Usb{

  @Override
  public void service() {
    System.out.println("鼠标连接电脑成功开始工作了...");
  }

}
```

```java
public class Fan implements Usb{

  @Override
  public void service() {
    System.out.println("风扇连接电脑成功，开始工作...");
  }

}
```

```java
public class Upan implements Usb{

  @Override
  public void service() {
    System.out.println("Upan连接电脑成功，开始工作...");
  }

}
```

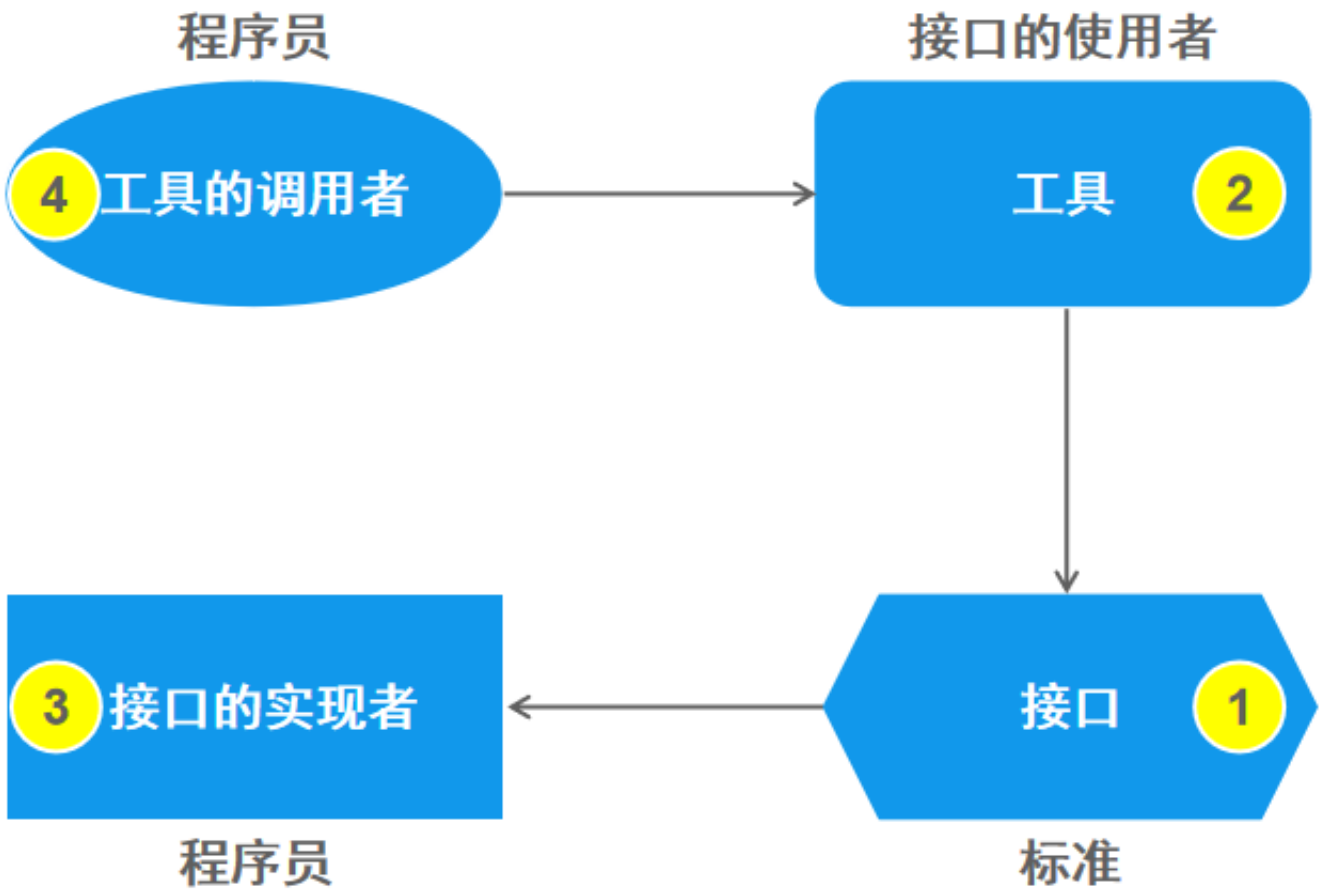
```java
public class Computer {
  Usb usb1;
  Usb usb2;
  Usb usb3;

  public void run() {
    System.out.println("电脑开始工作...");
    if(usb1!=null) {
      usb1.service();
    }
    if(usb2!=null) {
      usb2.service();
    }
    if(usb3!=null) {
      usb3.service();
    }
  }
}
```

```java
public class Test {
  public static void main(String[] args) {
    Computer lenovo=new Computer();

    Usb mouse=new Mouse();
    Usb fan=new Fan();
    Usb jinshidun=new Upan();

    //把usb设备连接到电脑上
    lenovo.usb1=fan;
    lenovo.usb2=mouse;
    lenovo.usb3=jinshidun;

    lenovo.run();
  }
}
```

7.2 回调原理

> 接口回调：先有接口的使用者，后有接口的实现者。

接口回调



7.3 接口好处

> - 程序的耦合度降低。
> - 更自然的使用多态。
> - 设计与实现完全分离。
> - 更容易搭建程序框架。
> - 更容易更换具体实现。