1. 枚举
2. 单例设计模式
3. 概述

一个类只能实例化出一个对象。

1. 实现步骤

* 第一步：声明私有静态字段，字段类型与当前类相同
* 第二步：将构造函数私有化(避免外部直接通过构造函数实例化对象)
* 第三步：定义公共静态方法作为接口来实例化对象(使用静态字段)

1. 示例

|  |
| --- |
| /\*\*  \* 单例设计模式  \* **@author** X230  \*  \*/  **public** **class** SingleInstance {  //1、定义私有静态成员对象。对象的类型与类相同  **private** **static** SingleInstance *obj* = **null**;    /\*\*  \* 构造函数  \* 2、将构造函数私有化（避免直接调用构造函数）  \*/  **private** SingleInstance() {  System.*out*.println("构造函数被执行了....." + System.*currentTimeMillis*());  }    /\*\*  \* 创建一个SingleInstance类型的对象  \* 3、提供一个静态公共的方法作为访问的接口  \* 方法中我们对对象进行判断，如果对象是null则调用构造函数，否则将直接返回对象  \* **@return**  \*/  **public** **static** SingleInstance create(){  //判断对象是否为null，如果是则实例化对象  **if**(*obj* == **null**)  *obj* = **new** SingleInstance();    //返回对象的实例  **return** *obj*;  }  } |

1. 枚举
2. 概述

当一个类可以实例化出有限个对象的时候，就可以使用枚举。枚举可以和switch进行连用

1. JDK1.5以前

Jdk1.5以前是通过自定义类进行实现的。

* 私有化构造函数
* 声明静态公共对象常量

|  |
| --- |
| /\*\*  \* 自定义类实现枚举  \* **@author** X230  \*  \*/  **public** **class** OS {  **public** **static** **final** OS *WINDOWS* = **new** OS("Windows","操作简单的操作系统");  **public** **static** **final** OS *LINUX* = **new** OS("Linux","比较安全的服务器系统");  **public** **static** **final** OS *UNIX* = **new** OS("Unix","较早的操作系统");  **public** **static** **final** OS *MAC* = **new** OS("Mac","苹果的操作系统");    //属性  **private** String name;  **private** String description;    //构造函数  **private** OS(String name, String description) {  **this**.name = name;  **this**.description = description;  }  //get/set访问器  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** String getDescription() {  **return** description;  }  **public** **void** setDescription(String description) {  **this**.description = description;  }  @Override  **public** String toString() {  **return** "OS [name=" + name + ", description=" + description + "]";  }      } |

1. JDK1.5以后

提供了enum关键字来声明枚举类型。

|  |
| --- |
| /\*\*  \* Enum  \* JDK1.5之后提供的关键字  \* **@author** X230  \*  \*/  **public** **enum** OSOfEnum {  *WINDOWS*,  *LINUX*,  *UNIX*,  *MAC*;  } |

枚举的元素其实就是用static final修饰的对象。

1. 枚举与swith

|  |
| --- |
| //枚举与switch语句  OSOfEnum ofe = OSOfEnum.*MAC*;    **switch**(ofe){  **case** *MAC*:  System.*out*.println("苹果的系统");  **break**;  **case** *WINDOWS*:  System.*out*.println("微软的系统");  **break**;  **case** *LINUX*:  System.*out*.println("Sun的系统");  **break**;    } |

1. 枚举与继承

枚举无法继承于类，但是枚举可以实现接口。

|  |
| --- |
| /\*\*  \* 父接口  \* **@author** X230  \*  \*/  **public** **interface** MyEnum {  **public** **abstract** **void** show();  } |
| /\*\*  \* 声明了枚举类型  \* **@author** X230  \*  \*/  **public** **enum** Os2 **implements** MyEnum{  *WINDOWS* {  @Override  **public** **void** show() {  System.*out*.println("show()方法被调用了。。。。。1");  }  },  *LINUX*{  @Override  **public** **void** show() {  System.*out*.println("show()方法被调用了。。。。。2");  }  },  *UNIX*{  @Override  **public** **void** show() {  System.*out*.println("show()方法被调用了。。。。。3");  }  },  *MAC*{  @Override  **public** **void** show() {  System.*out*.println("show()方法被调用了。。。。。4");  }  };    } |

1. Enum的常用成员

* Values ：获取枚举类型的成员数组
* ValueOf：根据字符串的内容获取枚举常量（如果不包含该常量会产生异常）

1. 应用

* 对象的个数已知
* 需要限制对象的命名及个数的情况下使用