Object类

```java
package com.iweb.lesson01;

/**
 * 作者: jack
 * 时间: 2021-04-24 0024 08:57
 * 描述: User
 */
public class User {

    private String name;
    private int age;

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

    @Override
    public String toString() {
        return name + "\t" + age;
    }
}
```

```java
package com.iweb.lesson01;

import java.lang.reflect.Field;

/**
 * 作者: jack
 * 时间: 2021-04-24 0024 08:52
 * 描述: Test
 * Object类： 始祖类，所有类的父类。 如果一个类没有显示的继承另外一个类，缺省继承了Object类
 * <p>
 * native: 本地的意思，调用本地方法区(C语言的一些源码)
 * JAVA->JVM->硬件(CPU内存)
 * <p>
 * registerNatives： 初始化java在执行过程中需要使用的一些API
 * <p>
 * getClass(): 回去当前类
 * <p>
 * public native int hashCode();获取对象的 hashCode值
 * <p>
 * hashCode：是一个int类型，同一个数据类型(基本数据类型、引用数据类型)的hashCode值不同，
 * 不同的类型的算法不同。
 * String存在字面量不同hashCode值相同
```

```
 * clone(): 克隆一个对象. 和原来的不是同一个对象
 * toString(): 内存地址, 对象输出的时候默认调用 toString
 *
 */
public class Test {

    public static void main(String[] args) {

        User user = new User("jack", 20);

        // 封装了类的一些成员和属性,Class 类的分类, 可以通过Class的对象获取类的 构造函数、属性、方法
        Class clazz = user.getClass();

        System.out.println(clazz.getSimpleName()); // 获取类的名称

        // Field 属性的分类
        Field[] fields = clazz.getDeclaredFields();// 获取类中的所有属性(包含私有的)

        for (int i = 0; i < fields.length; i++) {
            System.out.println(fields[i].getName());
        }

        /*hashCode*/
        // byte b = 100;
        // System.out.println(b); // 基本数据类型不能使用方法, 可以根据他对应的包装类来获取
        Byte b = 100;
        System.out.println(b.hashCode());// 100

        Long l = 900L;
        System.out.println(l.hashCode());

        Float f = 100.1F;
        System.out.println(f.hashCode()); // 1120416563

        System.out.println("".hashCode()); // 0

        System.out.println("abc".hashCode()); // 97  96354  : 存在值不同而hashcode值相同


        System.out.println(user.toString()); // 对象类型输出的时候默认调用 toString
        System.out.println(user);

    }


}
```

垃圾回收

```java
package com.iweb.lesson02;

/**
 * 作者: jack
 * 时间: 2021-04-24 0024 09:33
 * 描述: User
 */
public class User {


    @Override
    protected void finalize() throws Throwable {
        System.out.println("我被回收了");
        super.finalize();
    }
}
```

```java
package com.iweb.lesson02;

import com.sun.org.apache.xpath.internal.operations.String;

/**
 * 作者: jack
 * 时间: 2021-04-24 0024 09:28
 * 描述: Test ： 垃圾回收
 * 如何判定一个对象是垃圾?
 * 什么时候触发GC?
 */
public class Test {


    /**
     * <p>
     * 垃圾回收：Java自动垃圾回收机制(GC:Garbage Collection)，对象不可达的时候回收
     * 1. 当内存到达一定的使用空间后会触发垃圾回收
     * 2. 对象不可达的时候回收
     * 3. 主动垃圾回收 System.gc(); 不一定会立即回收
     * <p>
     * finalize() 垃圾回收
     */
    @org.junit.Test
    public void finalizeTest() throws Exception {
        User user = new User();
        user = null;
        // System.gc(); // 提示垃圾回收
        String[] sts = new String[900000000]; // 空间不够
        Thread.sleep(1000); // 给一点时间 让 JVM 触发 GC
    }

    // 对象不可达
    public void createUser() {
        User user = new User();
    }

    @org.junit.Test
```

```
    public void finalizeTest1() throws Exception {
        createUser();
        System.gc();
        Thread.sleep(1000);
    }

    // 如何判定一个对象是垃圾。
    // 1. 引用计数法  : 问题是循环引用  a 是类B 的成员 , b是类A的成员, 导致回收
    // 2. 对象不可达 (java)

    /**
     * User  user = new User();
     * User user1 = user;
     * user = null;
     *
     *
     * public void show(){
     *     B b = new B();
     *     A a = new A();
     *     a.setB(b);
     *     b.setA(a);
     * }
     * 引用计数法：上面的场景不会回收  对象不可达算法会回收
     */


}
```

最终类

最终类：final修饰的类。 不可继承。

Java的常见最终类：

```
public final class System
public final class String
public final class Long
....
```

包

java中通过包的机制来进行类的管理。

声明包

```
package com.iweb.lesson02;
```

类的全路径：包名+类名

```
com.iweb.lesson02.User
```

引用包: java.lang下的不需要引用可以直接使用。（String、System类就是java.lang）

```
import java.lang.reflect.Field;
import java.util.Sanner;
import java.util.*; // 引用包下所有的类。
```

java中常用的包：

```
java.lang:  可以直接使用不需要引用包。
java.sql: 数据库操作
java.util: 常用工具类
java.net: 网络工具
java.io: 输入输出流
...
```

命名规则：

```
1. 不要大写
2. 3级含以上 :   gongsi.业务模块.功能...
```

final、static

final（面试重点）

可以修饰类、属性、方法

修饰类： 最终类，类不能被继承

修饰属性：final int number = 100; 必须给定初始值, 值不可变量

修饰方法：public final void show()； 不能被重写。

特殊的场景：

```
public class Test {

    final int number; // 创建对象的时候初始化

    public Test() {
        number = 100; // 给定了初始值
    }

}
```

static(面设重点)

静态的：可以修饰属性和方法, 在类被加载的时候进行初始化。

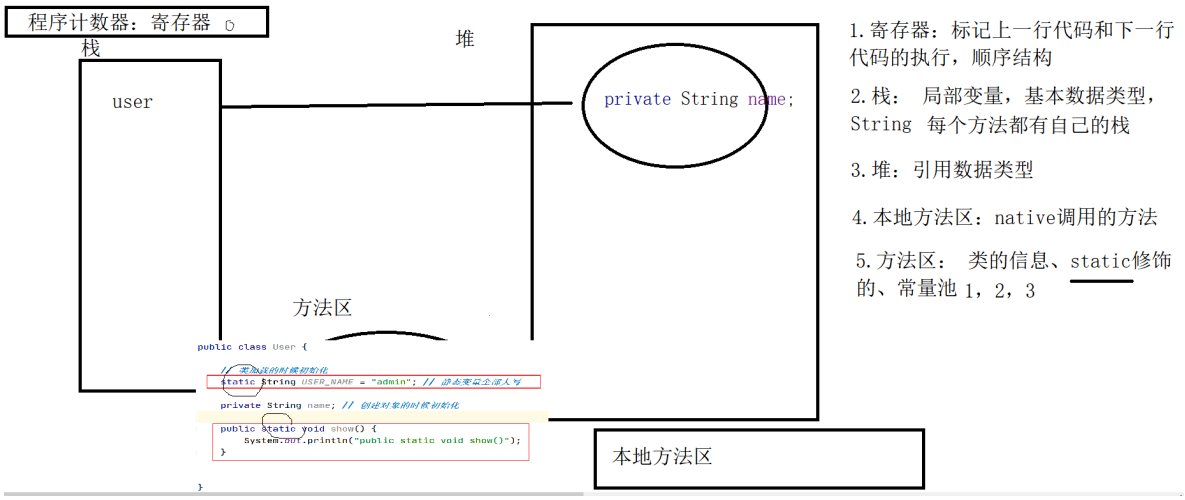
修饰属性： 静态变量也叫做类变量。

修饰方法： 静态方法也叫做类方法。

静态代码块： 类加载的时候自动调用。

代码块：调用构造函数之前自动调用。

执行顺序：1. 静态代码块 2. 代码块 3. 构造函数




```java
package com.iweb.lesson04;

/**
 * 作者: jack
 * 时间: 2021-04-24 0024 10:43
 * 描述: User
 */
public class User {

    // 类加载的时候初始化
    static String USER_NAME = "admin"; // 静态变量全部大写

    private String name; // 创建对象的时候初始化

    // 代码块： 构造函数之前自动执行
    {
        System.out.println("代码块");
    }

    // 静态代码块： 类被加载的时候自动执行
    static {
        System.out.println("静态代码块");
    }

    public User() {
        System.out.println("User()");
    }

    public static void show() {
        System.out.println("public static void show()");
    }

}
```

```

}
```

```
package com.iweb.lesson04;

/**
 * 作者: jack
 * 时间: 2021-04-24 0024 10:49
 * 描述: Test
 */
public class Test {
    public static void main(String[] args) throws Exception {

        // 加载类
        // Class.forName("com.iweb.lesson04.User"); // 没有创建对象 没有调用方法

        // 创建对象
        // new User(); // 1. 加载类(加载到方法区) 2. 创建对象(调用构造函数)
        // 执行顺序 1. 静态代码块 2. 代码块 3. 构造函数


        // 调用静态属性和静态方法: 直接通过类的名称来调用
        // System.out.println(User.USER_NAME);
        User.show();

    }
}
```

静态方法中只能直接访问静态属性和静态方法。

```
public class User {

    static String USER_NAME = "admin";

    int number = 200;


    public static void show() {
        System.out.println(USER_NAME);
        // System.out.println(number); // 不能访问
    }

}
```

初始化顺序:

当类被加载的时候首先会初始化所有 static 修饰的东西。按照书写顺序。

当创建对象的时候首先会初始化所有的成员属性和方法。按照书写顺序。

```
package com.iweb.lesson06;

/**
 * 作者: jack
 * 时间: 2021-04-24 0024 11:09
```

```
 * 描述: Test
 */
public class Test {
    static String USER_NAME = "admin"; // 如果写在static代码块之后不能访问

    static {
        System.out.println(1);
        System.out.println(USER_NAME);
        // null
        System.out.println(Test.PASSWORD); // 如果写在static代码块之后可以通过Test类名访问
    }

    static {
        System.out.println(2);
    }

    static {
        System.out.println(3);
    }


    static String PASSWORD = "admin";

    {
        System.out.println("a");
        // System.out.println(name);
        System.out.println(this.name); // null
    }

    {
        System.out.println("b");
    }

    {
        System.out.println("c");
    }

    String name = "jack";

    public static void main(String[] args) throws Exception {

        // Class.forName("com.iweb.lesson06.Test"); // 静态代码块
        // 代码块
        new Test();
    }
}
```

访问控制符

访问控制符： 标记了类、属性、方法的作用范围(全局、同包、子类、类中)

访问控制符：

public : 全局，任何地方都可以访问。 修饰 类、属性、方法

缺省的： 同包、类中，只有同一个包中可以访问。类、属性、方法

protected：同包、不同包的子类、类中可以访问。属性、方法

private：类中。属性、方法