**类变量（静态变量）**

Java 中的静态变量是指在类中定义的一个变量，它与类相关而不是与实例相关，即无论创建多少个类实例，静态变量在内存中只有一份拷贝，被所有实例共享。

静态变量在类加载时被创建，在整个程序运行期间都存在。

定义方式

静态变量的定义方式是在类中使用 static 关键字修饰变量，通常也称为类变量。

以下实例中我们定义一个静态变量 count ，其初始值为 0：

实例

public class MyClass {

public static int count = 0;

// 其他成员变量和方法

}

访问方式

由于静态变量是与类相关的，因此可以通过类名来访问静态变量，也可以通过实例名来访问静态变量。

实例

MyClass.count = 10; // 通过类名访问

MyClass obj = new MyClass();

obj.count = 20; // 通过实例名访问

生命周期

静态变量的生命周期与程序的生命周期一样长，即它们在类加载时被创建，在整个程序运行期间都存在，直到程序结束才会被销毁。因此，静态变量可以用来存储整个程序都需要使用的数据，如配置信息、全局变量等。

初始化时机

静态变量在类加载时被初始化，其初始化顺序与定义顺序有关。

如果一个静态变量依赖于另一个静态变量，那么它必须在后面定义。

实例

public class MyClass {

public static int count1 = 0;

public static int count2 = count1 + 1;

// 其他成员变量和方法

}

上面的例子中，count1 要先于 count2 初始化，否则编译时会报错。

常量和静态变量的区别

常量也是与类相关的，但它是用 final 关键字修饰的变量，一旦被赋值就不能再修改。与静态变量不同的是，常量在编译时就已经确定了它的值，而静态变量的值可以在运行时改变。另外，常量通常用于存储一些固定的值，如数学常数、配置信息等，而静态变量通常用于存储可变的数据，如计数器、全局状态等。

总之，静态变量是与类相关的变量，具有唯一性和共享性，可以用于存储整个程序都需要使用的数据，但需要注意初始化时机和与常量的区别。

静态变量的访问修饰符

静态变量的访问修饰符可以是 public、protected、private 或者默认的访问修饰符（即不写访问修饰符）。

需要注意的是，静态变量的访问权限与实例变量不同，因为静态变量是与类相关的，不依赖于任何实例。

静态变量的线程安全性

Java 中的静态变量是属于类的，而不是对象的实例。因此，当多个线程同时访问一个包含静态变量的类时，需要考虑其线程安全性。

静态变量在内存中只有一份拷贝，被所有实例共享。因此，如果一个线程修改了静态变量的值，那么其他线程在访问该静态变量时也会看到修改后的值。这可能会导致并发访问的问题，因为多个线程可能同时修改静态变量，导致不确定的结果或数据一致性问题。

为了确保静态变量的线程安全性，需要采取适当的同步措施，如同步机制、原子类或 volatile 关键字，以便在多线程环境中正确地读取和修改静态变量的值。

静态变量的命名规范

静态变量的命名规范与实例变量相同，一般采用驼峰命名法，并且要用 static 关键字明确标识。例如：

实例

public class MyClass {

public static int MAX\_COUNT = 100;

// 其他成员变量和方法

}

静态变量的使用场景

静态变量通常用于以下场景：

存储全局状态或配置信息

计数器或统计信息

缓存数据或共享资源

工具类的常量或方法

单例模式中的实例变量

实例

以下实例定义了一个 AppConfig 类，其中包含了三个静态变量 APP\_NAME、APP\_VERSION 和 DATABASE\_URL，用于存储应用程序的名称、版本和数据库连接URL。这些变量都被声明为 final，表示它们是不可修改的常量。

在 main() 方法中，我们打印出了这些静态变量的值。

AppConfig.java 文件代码：

public class AppConfig {

public static final String APP\_NAME = "MyApp";

public static final String APP\_VERSION = "1.0.0";

public static final String DATABASE\_URL = "jdbc:mysql://localhost:3306/mydb";

public static void main(String[] args) {

System.out.println("Application name: " + AppConfig.APP\_NAME);

System.out.println("Application version: " + AppConfig.APP\_VERSION);

System.out.println("Database URL: " + AppConfig.DATABASE\_URL);

}

}

以上实例编译运行结果如下:

Application name: MyApp

Application version: 1.0.0

Database URL: jdbc:mysql://localhost:3306/mydb

可以看到，这些静态变量存储的全局配置信息可以在整个程序中使用，并且不会被修改。这个例子展示了静态变量的另一个常见应用，通过它我们可以很方便地存储全局配置信息，或者实现其他需要全局共享的数据。

以下实例定义了一个 Counter 类，其中包含了一个静态变量 count，用于记录创建了多少个 Counter 对象。

每当创建一个新的对象时，构造方法会将计数器加一。静态方法 getCount() 用于获取当前计数器的值。

在 main() 方法中，我们创建了三个 Counter 对象，并打印出了计数器的值。

Counter.java 文件代码：

public class Counter {

private static int count = 0;

public Counter() {

count++;

}

public static int getCount() {

return count;

}

public static void main(String[] args) {

Counter c1 = new Counter();

Counter c2 = new Counter();

Counter c3 = new Counter();

System.out.println("目前为止创建的对象数: " + Counter.getCount());

}

}

以上实例编译运行结果如下:

目前为止创建的对象数: 3

可以看到，计数器记录了创建了三个对象。这个例子展示了静态变量的一个简单应用，通过它我们可以很方便地统计对象的创建次数，或者记录其他需要全局共享的数据。