# 自动装箱/出箱

简单类型可以直接存在集合中

# 自动引入（静态引入）

# 可变长参数

# 格式化输入输出

引入System.out.printf()

# 枚举类型

一个类所创建的对象的个数是固定的

枚举本质上还是类

Enum关键词

枚举类型有构造方法但只能是私有的

# 注解

## @Override

## @Deprecated（忽略过时的）

## @SuppressWarnings忽略警告

注解是写给程序看的，注释是给程序员看的。

元数据:用来描述数据的数据

Java注解：是java代码的元数据，他表示了java代码无法描述的额外信息

## 元注解

描述注解的注解

|  |
| --- |
| @Retention(RetentionPolicy.RUNTIME) @Target({ElementType.METHOD}) |

Java注解可以在编译、类加载、运行时被读取，注解不会影响程序代码的运行。

## 注解的分类：

1. 内建注解（@Override）
2. 自定义注解
3. 元注解

根据是否有成员变量注解分为：

标记注解：没有定义成员变量的注解。

@Target:限定注解的使用目标。

Type 用在类或者接口上

|  |
| --- |
| **import** java.lang.annotation.ElementType; **import** java.lang.annotation.Retention; **import** java.lang.annotation.RetentionPolicy; **import** java.lang.annotation.Target;  *//当前的元注解Target描述了First注解的使用地方 ：类，接口ElementType.TYPE //@Target(value = ElementType.METHOD)//单个参数赋值 //@Target(ElementType.METHOD)//单个参数赋值 //@Target({ ElementType.METHOD, ElementType.FIELD })//多个参数赋值* @Target(value = { ElementType.***METHOD***, ElementType.***FIELD*** }) *// 多个参数赋值* @Retention(RetentionPolicy.***RUNTIME***) **public** @**interface** First {  String[] value(); } |

@Retention：指明注解信息的保存时间

Source 注解信息只保存在源程序中，在类文件中不保存。编译时读取

Runtime 定义为此时才能使用反射

# Lambda表达式

|  |
| --- |
| **package** mytest; **public class** LambdaTest{  **interface** LmTest{  **public void** m(String s);  }  **public static void** main(String[] args) { *// LmTest lt = (String s )->{System.out.println(s);}; // lt.m("ssss");* Converter<String,Integer> ct = (String s)->Integer.*parseInt*(s); //函数式表达式的用法  System.***out***.println(ct.convert(**"11111"**));      } } |

# 函数式表达式

|  |
| --- |
| **package** mytest;  @FunctionalInterface **interface** Converter<String, Integer> {  *//只有一个方法 能横好的支持lambda表达式* Integer convert(String s); } |