面向对象进阶(三)

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

- **内部类**
  - 内部类概述、成员内部类[了解]
  - 静态内部类[了解]
  - 局部内部类[了解]
  - 匿名内部类[重点]
- **枚举**
- **泛型**
- **java.lang包下的常用API**

内部类

- 是类中的五大成分之一（成员变量、方法、构造器、内部类、代码块），如果一个类定义在另一个类的内部，这个类就是内部类。
- 场景：当一个类的内部，包含了一个完整的事物，且这个事物没有必要单独设计时，就可以把这个事物设计成内部类。

```
public class Car{
    // 内部类
    public class Engine{
    }
}
```

内部类有四种形式

01 成员内部类[了解]

02 静态内部类[了解]

03 局部内部类[了解]

04 匿名内部类[重点]

成员内部类

- 就是类中的一个普通成员，类似前面我们学过的普通的成员变量、成员方法。

```
public class Outer {
    // 成员内部类
    public class Inner {

    }
}
```

注意：JDK16之前，成员内部类中不能定义静态成员，JDK 16开始也可以定义静态成员了

创建对象的格式：

```
外部类名.内部类名 对象名 = new 外部类(...).new 内部类(...);

Outer.Inner in =  new Outer().new  Inner();
```

成员内部类中访问其他成员的特点：

1、和前面学过的实例方法一样，成员内部类的实例方法中，同样可以直接访问外部类的实例成员、静态成员。

2、可以在成员内部类的实例方法中，拿到当前外部类对象，格式是：外部类名.this 。

总结

1. 成员内部类是什么？ 如何创建其对象？

   - 就是类中的一个普通成员，类似前面我们学过的普通成员变量、成员方法
   - 外部类名.内部类名 对象名 = new 外部类(...).new 内部类(…);

2. 成员内部类的实例方法中，访问其他成员有啥特点？

   - 可以直接访问外部类的实例成员、静态成员
   - 可以通过：外部类名.this，获取当前外部类对象，格式是：外部类名.this。

- **内部类**
  - 内部类概述、成员内部类[了解]
  - 静态内部类[了解]
  - 局部内部类[了解]
  - 匿名内部类[重点]
- **枚举**
- **泛型**
- **java.lang包下的常用API**

什么是静态内部类?

- 有static修饰的内部类，属于外部类自己持有。

```
public class Outer{
    // 静态内部类
    public static class Inner{

    }
}
```

创建对象的格式:

```
外部类名.内部类名 对象名 = new 外部类.内部类(…);

Outer.Inner in =  new Outer.Inner();
```

静态内部类中访问外部类成员的特点

- 可以直接访问外部类的静态成员，不可以直接访问外部类的实例成员。

总结

1. 什么是静态内部类？如何创建对象？ 有啥特点？

- 有static修饰的内部类。
- 外部类名.内部类名 对象名 = new 外部类.内部类(…);
- 可以直接访问外部类的静态成员，不能直接访问外部类的实例成员

。

- **内部类**
  - 内部类概述、成员内部类[了解]
  - 静态内部类[了解]
  - 局部内部类[了解]
  - 匿名内部类[重点]
- **枚举**
- **泛型**
- **java.lang包下的常用API**

局部内部类

- 局部内部类是定义在在方法中、代码块中、构造器等执行体中。

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

    }

    public static void go(){
        class A{

        }

        abstract class B{

        }

        interface C{

        }
    }
}
```

鸡肋语法，看看就好

- **内部类**
  - 内部类概述、成员内部类[了解]
  - 静态内部类[了解]
  - 局部内部类[了解]
  - 匿名内部类[重点]
- **枚举**
- **泛型**
- **java.lang包下的常用API**

匿名内部类的学习路径

01

认识匿名内部类

02

匿名内部类的
常见使用场景

小结

匿名内部类

- 就是一种特殊的局部内部类；所谓匿名：指的是程序员不需要为这个类声明名字。

```
new 类或接口(参数值…) {
    类体(一般是方法重写);
};
```

```
new Animal(){
    @Override
    public void cry() {
    }
};
```

- **特点：**匿名内部类本质就是一个子类，并会立即创建出一个子类对象。
- **作用：**用于更方便的创建一个子类对象。

匿名内部类的学习路径

01

认识匿名内部类

02

匿名内部类的
常见使用场景

小结

匿名内部类在开发中的使用场景

- 通常作为一个参数传输给方法。

需求：猫、狗参加游泳比赛。

```
public interface Swimming{
    void swim();
}
```

```
public class Test {
    public static void main(String[] args) {
        // 目标：认识匿名内部类，并掌握其作用。
        Swimming s1 = new Swimming() {
            @Override
            public void swim() {
                System.out.println("狗    飞快~~~");
            }
        };
        go(s1);

        go(new Swimming() {
            @Override
            public void swim() {
                System.out.println("猫    也还行~~~");
            }
        });
    }

    public static void go(Swimming s){
        System.out.println("开始============");
        s.swim();
    }
}
```

总结

1、匿名内部类的书写格式是什么样的？

```
new 类或接口(参数值…) {
    类体(一般是方法重写);
};
```

```
new Animal(){
    @Override
    public void cry() {
    }
};
```

2、匿名内部类有啥特点？

- 匿名内部类本质就是一个子类，并会立即创建出一个子类对象

3、匿名内部类有啥作用、应用场景?

- 可以更方便的创建出一个子类对象。

匿名内部类通常作为一个参数传输给方法。

- 内部类
- 枚举
  - 认识枚举
  - 枚举的常见应用场景
- 泛型
- java.lang包下的常用API

枚举

- 枚举是一种特殊类。

枚举类的格式：

```
修饰符 enum 枚举类名{
    名称1 , 名称2, ... ;
    其他成员…
}
```

枚举的概述

- 枚举是一种特殊类。

枚举类的格式：

```
修饰符 enum 枚举类名{
    名称1，名称2, ... ;
    其他成员…
}
```

```
public enum A{
    X , Y , Z;
    ...
}
```

注意：

- 枚举类中的第一行，只能写一些合法的标识符(名称)，多个名称用逗号隔开。
- 这些名称，本质是常量，每个常量都会记住枚举类的一个对象。

枚举类的特点：

```
public enum A{
    X , Y , Z;
}
```

```
Compiled from “A.java"
public final class A extends java.lang.Enum<A> {
    public static final A X = new A();
    public static final A Y = new A();
    public static final A Z = new A();

    public static A[] values();
    public static A valueOf(java.lang.String);
}
```

- 枚举类的第一行只能罗列一些名称，这些名称都是常量，并且每个常量记住的都是枚举类的一个对象。
- 枚举类的构造器都是私有的（写不写都只能是私有的），因此，枚举类对外不能创建对象。
- 枚举都是最终类，不可以被继承。
- 枚举类中，从第二行开始，可以定义类的其他各种成员。
- 编译器为枚举类新增了几个方法，并且枚举类都是继承：java.lang.Enum类的，从enum类也会继承到一些方法。

多学一招

使用枚举类实现单例设计模式

- 内部类
- 枚举
  - 认识枚举
  - 枚举的常见应用场景
- 泛型
- java.lang、java.util包下的常用API

枚举的常见应用场景：

- 用来表示一组信息，然后作为参数进行传输。

选 信息，并作为参数传输

-

选 为参数传输

- ，对使用者更友好，建议使用!

- 内部类
- 枚举
- 泛型
  - 认识泛型
  - 泛型类
  - 泛型接口
  - 泛型方法、泛型通配符、上下限
  - 泛型的注意事项：擦除问题、基本数据类型问题
- 常用API

泛型

- 定义类、接口、方法时，同时声明了一个或者多个类型变量（如：<E>），称为泛型类、泛型接口，泛型方法、它们统称为泛型。

```
public class ArrayList<E>{
   ...
}
```

- 作用：泛型提供了在编译阶段约束所能操作的数据类型，并自动进行检查的能力！这样可以避免强制类型转换，及其可能出现的异常。

- 泛型的本质：把具体的数据类型作为参数传给类型变量。

自定义泛型类　自定义泛型接口　自定义泛型方法

- 内部类
- 枚举
- 泛型
  - 认识泛型
  - 泛型类
  - 泛型接口
  - 泛型方法、泛型通配符、上下限
  - 泛型的注意事项：擦除问题、基本数据类型问题
- 常用API

泛型类

```
修饰符 class 类名<类型变量，类型变量，…> {


}
```

```
public class ArrayList<E>{
    ...
}
```

- 注意：类型变量建议用大写的英文字母，常用的有：**E、T、K、V** 等

- 内部类
- 枚举
- 泛型
  - 认识泛型
  - 泛型类
  - 泛型接口
  - 泛型方法、泛型通配符、上下限
  - 泛型的注意事项：擦除问题、基本数据类型问题
- 常用API

泛型接口

```
修饰符 interface 接口名<类型变量，类型变量，…> {


}
```

```
public interface A<E>{
    ...
}
```

- 注意：类型变量建议用大写的英文字母，常用的有：**E、T、K、V** 等

- 内部类
- 枚举
- 泛型
  - 认识泛型
  - 泛型类
  - 泛型接口
  - 泛型方法、泛型通配符、上下限
  - 泛型的注意事项：擦除问题、基本数据类型问题
- 常用API

泛型方法

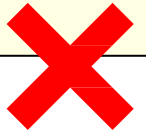
```
修饰符 <类型变量，类型变量，…> 返回值类型 方法名(形参列表)
{

}
```

```
public static <T> void test(T t){


}
```

```
public E get(int index){
    return (E) arr[index];
}
```



通配符

- 就是“?”，可以在“使用泛型”的时候代表一切类型； ＥＴＫＶ是在定义泛型的时候使用。

泛型的上下限：

- 泛型上限：？**extends Car**：？能接收的必须是Car或者其子类 。
- 泛型下限： ？ **super Car** ： ？ 能接收的必须是Car或者其父类。

- 内部类
- 枚举
- 泛型
  - 认识泛型
  - 泛型类
  - 泛型接口
  - 泛型方法、泛型通配符、上下限
  - 泛型的注意事项：擦除问题、基本数据类型问题
- 常用API

泛型的擦除问题和注意事项

- 泛型是工作在编译阶段的，一旦程序编译成class文件，class文件中就不存在泛型了，这就是泛型擦除。
- 泛型不支持基本数据类型，只能支持对象类型（引用数据类型）。

- 内部类
- 枚举
- 泛型
- 常用API
  - API概述、接下来课程介绍
  - Object类
  - Objects
  - 包装类
  - StringBuilder、StringBuffer
  - StringJoiner
  - BigDecimal

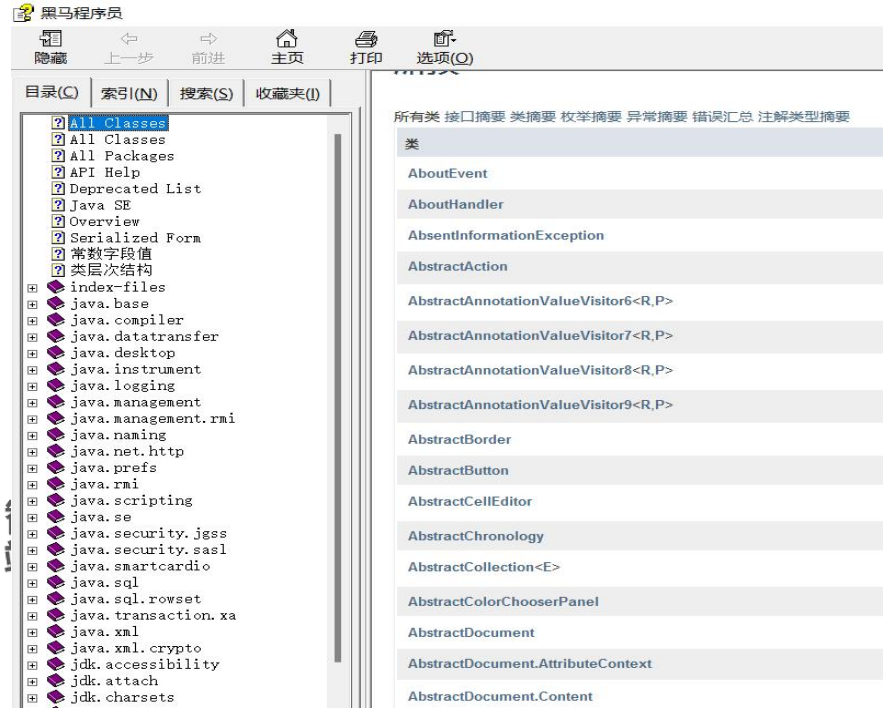
什么是API?

- API(Application Programming interface) ： 应用程序编程接口
- **就是Java帮我们已经写好一些程序，如：类、方法等，我们直接拿过来用就可以解决一些问题。**

为什么要学别人写好的程序？

开发效率高！

API文档



常用API

Object	LocalTime	Set	FileOutputStream	Properties
objects	LocalDateTime	HashSet	Reader	Thread
Integer	Duration	LinkedHashSet	FileReader	Runnable
StringBuilder	Period	TreeSet	Writer	Callable
StringBuffer	ZoneId	Map	FileWriter	ExecutorService
Math	ZonedDateTime	HashMap	BufferdInputStream	ThreadPoolExecutor
System	Arrays	LinkedHashMap	BufferdOutputStream	Socket
Runtime	Comparable	TreeMap	BufferedReader	ServerSocket
BigDecimal	Comparator	Iterator	BufferedWriter	Class
Date	Collection	Stream	PrintStream	Method
SimpleDateFormat	List	InputStream	PrintWriter	Constructor
Calendar	ArrayList	FileInputStream	ObjectInputStream	Field
LocalDate	LinkedList	OutputStream	ObjectOutputStream	Proxy ...

一听就会，就是记不住！

千里之行始于足下

多记，多查，多写代码，孰能生巧！

- 内部类
- 枚举
- 泛型
- 常用API
  - API概述、接下来课程介绍
  - Object
  - Objects
  - 包装类
  - StringBuilder、StringBuffer
  - StringJoiner

Object类的作用：

- Object类是Java中所有类的祖宗类，因此，Java中所有类的对象都可以直接使用Object类中提供的一些方法。

Object类的常见方法

方法名	说明
public String toString()	返回对象的字符串表示形式。
public boolean equals(Object o)	判断两个对象是否相等。
protected Object clone()	对象克隆

方法名	说明
public String toString()	返回对象的字符串表示形式。
public boolean equals(Object o)	判断两个对象是否相等。
protected Object clone()	对象克隆

toString存在的意义：toString()方法存在的意义就是为了被子类重写，以便返回对象具体的内容。

equals存在的意义：直接比较两个对象的地址是否相同完全可以用“==”替代equals，equals存在的意义就是为了被子类重写，以便子类自己来定制比较规则（比如比较对象内容）。

总结

1、Object中toString方法的作用是什么？存在的意义是什么？

- 基本作用：返回对象的字符串形式。
- 存在的意义：让子类重写，以便返回子类对象的内容。

2、Object中equals方法的作用是什么？存在的意义是什么？

- 基本作用：默认是比较两个对象的地址是否相等。
- 存在的意义：让子类重写，以便用于比较对象的内容是否相同。

Object类提供的对象克隆方法

方法名	说明
protected Object clone()	对象克隆

当某个对象调用这个方法时，这个方法会复制一个一模一样的新对象返回。

浅克隆

深克隆

浅克隆

拷贝出的新对象，与原对象中的数据一模一样（引用类型拷贝的只是地址）

堆内存

int	id	1
String	username	"zhangsan"
String	password	"wo666"
double[]	scores	99.0, 99.5

字符串常量池

浅克隆

拷贝出的新对象，与原对象中的数据一模一样（引用类型拷贝的只是地址）

深克隆

对象中基本类型的数据直接拷贝。
对象中的字符串数据拷贝的还是地址。
对象中还包含的其他对象，不会拷贝地址，会创建新对象。

堆内存

深克隆

对象中基本类型的数据直接拷贝。
对象中的字符串数据拷贝的还是地址。
对象中还包含的其他对象，不会拷贝地址，会创建新对象。

总结

1、什么是对象克隆？

- 复制一个一模一样的新对象出来。

2、什么是浅克隆？

- 拷贝出的新对象，与原对象中的数据一模一样（引用类型拷贝的只是地址）

3、什么是深克隆？

- 对象中基本类型的数据直接拷贝。
- 对象中的字符串数据拷贝的还是地址。
- 对象中的其他引用类型的数据，不会拷贝地址，会创建新对象。

- 内部类
- 枚举
- 泛型
- 常用API
  - API概述、接下来课程介绍
  - Object
  - Objects
  - 包装类
  - StringBuilder、StringBuffer
  - StringJoiner

```java
@Override
public boolean equals(Object o) {
    // 1、判断是否是同一个对象比较，如果是返回true。
    if (this == o) return true;
    // 2、如果o是null返回false  如果o不是学生类型返回false  ...Student !=  ..Pig
    if (o == null || this.getClass() != o.getClass()) return false;
    // 3、说明o一定是学生类型而且不为null
    Student student = (Student) o;
    return sex == student.sex && age == student.age && Objects.equals(name, student.name);
}
```

为啥比较两个对象是否相等，要用Objects的equals方法，而不是用对象自己的equals方法来比较呢？？？

Objects

- Objects是一个工具类，提供了很多操作对象的静态方法给我们使用。

Objects类的常见方法

方法名	说明
public static boolean equals(Object a, Object b)	先做非空判断，再比较两个对象
public static boolean isNull(Object obj)	判断对象是否为null，为null返回true ,反之
public static boolean nonNull(Object obj)	判断对象是否不为null，不为null则返回true, 反之

源码分析

```
public static boolean equals(Object a, Object b) {
    return (a == b) || (a != null && a.equals(b));
}
```

总结

1. 为什么要使用Objects类提供的equals方法来比较两个对象？

●更安全。

- 内部类
- 枚举
- 泛型
- 常用API
  - API概述、接下来课程介绍
  - Object
  - Objects
  - 包装类
  - StringBuilder、StringBuffer
  - StringJoiner

为什么要有包装类?

基本类型的数据
byte
short
int
long
char
float
double
boolean

万物皆对象

我们又不是对象啊！

包装类

- 包装类就是把基本类型的数据包装成对象。

基本数据类型	对应的包装类（引用数据类型）
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double
boolean	Boolean

基本类型的数据包装成对象的方案
public Integer(int value)： 已过时
public static Integer valueOf(int i)

自动装箱：基本数据类型可以自动转换为包装类型。 **自动拆箱**：包装类型可以自动转换为基本数据类型。

包装类的其他常见操作

- 可以把基本类型的数据转换成字符串类型。

public static String toString(double d)
public String toString()

- 可以把字符串类型的数值转换成数值本身对应的数据类型。

public static int parseInt(String s)
public static Integer valueOf(String s)

总结

1. 为什么要有包装类，包装类有哪些？

   - 为了万物皆对象，并且泛型和集合都不支持基本类型，支持包装类
   - 8种，int -> Integer ， char -> Character，其他的都是首字母大写

2. 包装类提供了哪些常用的功能？

   - 可以把基本类型的数据转换成字符串类型。

     ```
     public static String toString(double d)
     public String toString()
     ```

   - 可以 类型。

     ```
     public static int parseInt(String s)
     public static Integer valueOf(String s)
     ```

- 内部类
- 枚举
- 泛型
- 常用API
  - API概述、接下来课程介绍
  - Object
  - Objects
  - 包装类
  - StringBuilder、StringBuffer

StringBuilder

- StringBuilder代表可变字符串对象，相当于是一个容器，它里面装的字符串是可以改变的，就是用来操作字符串的。
- 好处：StringBuilder比String更适合做字符串的修改操作，效率会更高，代码也会更简洁。

构造器	说明
public StringBuilder()	创建一个空白的可变的字符串对象，不包含任何内容
public StringBuilder(String str)	创建一个指定字符串内容的可变字符串对象

方法名称	说明
public StringBuilder append(任意类型)	添加数据并返回StringBuilder对象本身
public StringBuilder reverse()	将对象的内容反转
public int length()	返回对象内容长度
public String toString()	通过toString()就可以实现把StringBuilder转换为String

为啥操作字符串建议使用StringBuilder，而不用原来学过的String??

```
public class Test {
    public static void main(String[] args) {
        String s = "";
        for (int i = 0; i < 1000000; i++) {
            s = s + "abc";
        }
        System.out.println(s);
    }
}
```


```
public class Test {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();
        for(int i = 0; i < 1000000; i++) {
                sb.append("abc");
        }
        System.out.println(sb);
    }
}
```

- 对于字符串相关的操作，如频繁的拼接、修改等，建议用StringBuidler，效率更高!
- 注意：如果操作字符串较少，或者不需要操作，以及定义字符串变量，还是建议用String。

StringBuffer与StringBuilder

注意：

- StringBuffer的用法与StringBuilder是一模一样的
- 但 StringBuilder是线程不安全的 StringBuffer是线程安全的

总结

1、为什么对于字符串相关的操作，如频繁的拼接、修改等，建议用StringBuidler?

- **String 是不可变字符串、频繁操作字符串会产生很多无用对象，性能差。**
- **StringBuilder：是内容可变的字符串、拼接字符串性能好、代码优雅。**
- **注意：如果操作字符串较少，或者不需要操作，以及定义字符串时，则用String。**

案例

返回任意整型数组的内容

需求

设计一个方法，用于返回任意整型数组的内容，要求返回的数组内容格式如：[11, 22, 33]

分析：

- 方法是否需要接收数据？ 需要接收整型数组
- 方法是否需要返回数据？ 需要返回拼接后的结果
- 方法内部：遍历数组的数据，把遍历到的数据都拼接起来，此时使用StringBuilder来完成拼接。

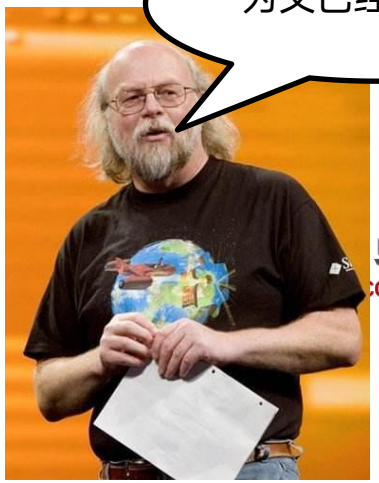
- 内部类
- 枚举
- 泛型
- 常用API
  - API概述、接下来课程介绍
  - Object
  - Objects
  - 包装类
  - StringBuilder、StringBuffer
  - StringJoiner

为什么学StringJoiner?

```
public class Test {
    public static void main(String[] args) {
        String s = "";
        for (int i = 0; i < 1000000; i++) {
            s += "abc";
        }
        System.out.println(s);
    }
}
```

速度忒慢

既能高效，又能实现更方便的拼接？




```
public static String getArrayData(int[] arr){
    // 1、判断arr是否为null
    if(arr == null){
        return null;
    }
    // 2、arr数组对象存在。 arr = [11, 22, 33]
    StringBuilder sb = new StringBuilder();
    sb.append("[");
    for (int i = 0; i < arr.length; i++) {
        if(i == arr.length - 1){
            sb.append(arr[i]);
        }else {
            sb.append(arr[i]).append(", ");
        }
    }
    sb.append("]");

    return sb.toString();
}
```

速度快了

StringJoiner

- JDK8开始才有的，跟StringBuilder一样，也是用来操作字符串的，也可以看成是一个容器，创建之后里面的内容是可变的。
- 好处：不仅能提高字符串的操作效率，并且在有些场景下使用它操作字符串，代码会更简洁

构造器	说明
public StringJoiner (间隔符号)	创建一个StringJoiner对象，指定拼接时的间隔符号
public StringJoiner (间隔符号，开始符号，结束符号)	创建一个StringJoiner对象，指定拼接时的间隔符号、开始符号、结束符号

方法名称	说明
public StringJoiner add (添加的内容)	添加数据，并返回对象本身
public int length()	返回长度（字符出现的个数）
public String toString()	返回一个字符串（该字符串就是拼接之后的结果）

总结

1. StringJoiner是啥？使用它有什么好处？

- JDK8出现的一个可变的、操作字符串的容器。
- 使用它拼接字符串的时候，不仅高效，而且代码的编写更加的方便。


黑马程序员
www.itheima.com
传智教育旗下
高端IT教育品牌