

Java 常规教学

计算机协会

上海财经大学

23 会计学院 ACCA 张华轩



Contents

① Java 生态

② Java 基础

③ 面向对象程序设计

① Java 生态

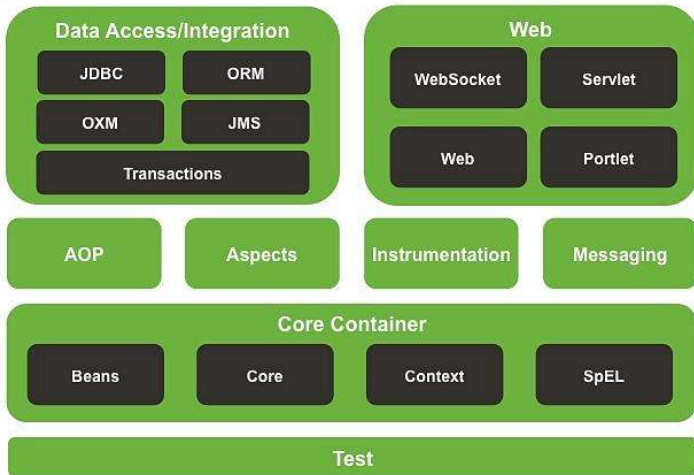
② Java 基础

③ 面向对象程序设计

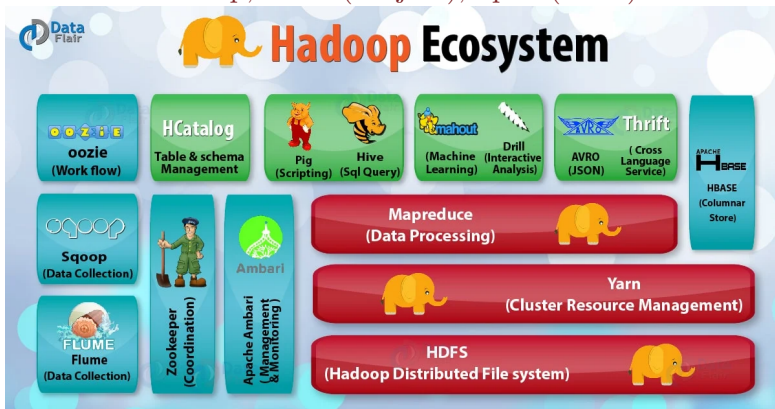
Spring & Spring Boot



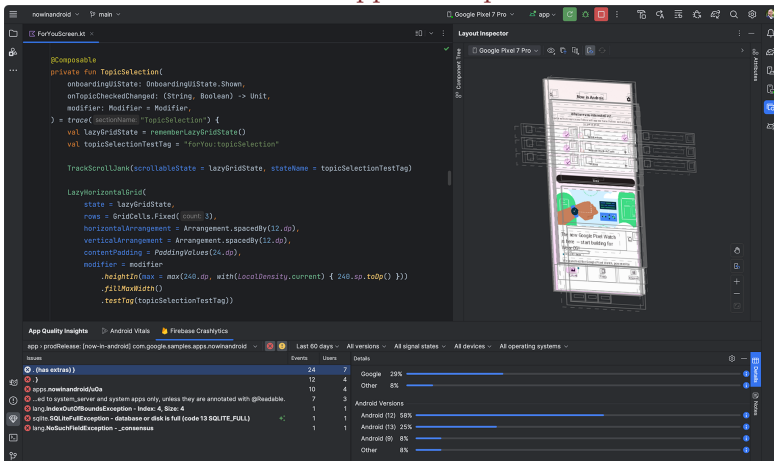
Spring Framework Runtime



Hadoop, Storm(Clojure), Spark(Scala)



Android App Development



① Java 生态

② Java 基础

③ 面向对象程序设计

环境配置

？有必要吗

谁还用命令行编译 Java

- JRE(Java Runtime Environment): Java 运行时环境
- JDK(Java Development Kit): Java 开发工具包
- Java IDE: Eclipse, IntelliJ IDEA, VS Code(new!)

<https://www.jetbrains.com/zh-cn/idea>

IntelliJ 启动!

Java 版本演进

Java SE 8 (LTS)	52	2014 年 3 月 18 日	OpenJDK 目前由 Red Hat 维护 Oracle 于 2022 年 3 月停止更新（商用） Oracle 于 2030 年 12 月停止更新（非商用） Azul 于 2030 年 12 月停止更新 IBM Semeru 于 2026 年 5 月停止更新 Eclipse Adoptium 于 2026 年 5 月或之后停止更新 Amazon Corretto 于 2026 年 5 月或之后停止更新	Oracle 于 2030 年 12 月停止更新 Red Hat 于 2026 年 11 月停止更新
Java SE 9	53	2017 年 9 月 21 日	OpenJDK 于 2018 年 3 月停止更新	不适用
Java SE 10	54	2018 年 3 月 20 日	OpenJDK 于 2018 年 9 月停止更新	不适用
Java SE 11 (LTS)	55	2018 年 9 月 25 日	OpenJDK 目前由 Red Hat 维护 Azul 于 2026 年 9 月停止更新 IBM Semeru 于 2024 年 10 月停止更新 Eclipse Adoptium 于 2024 年 10 月或之后停止更新 Amazon Corretto 于 2027 年 9 月或之后停止更新 微软于 2024 年 10 月或之后停止更新	Oracle 于 2026 年 9 月停止更新 Azul 于 2026 年 9 月停止更新 Red Hat 于 2024 年 10 月停止更新

Latest: Java SE 25(LTS)

JDK

主流 JDK 分支

- OpenJDK: 由社区支持、开源
- Oracle JDK: Oracle 主导、商用 (最后的免费商用版本是 8u202)
- Adoptium Eclipse Temurin: 由 Eclipse 基金会支持、开源
- Amazon Corretto: 由 Amazon 主导、免费商用

<https://adoptium.net/zh-CN/temurin/releases>

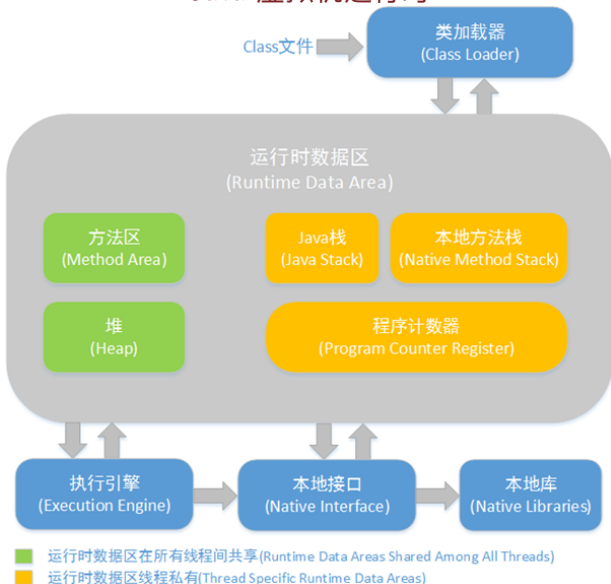
JVM

Java 虚拟机 (Java Virtual Machine)

- 是什么：用于执行 Java 字节码的虚拟机，拥有独立的运行机制，所有 Java 程序都运行在 JVM 内部，但其运行的 Java 字节码未必由 Java 编译而成
- 为什么：一次编译，到处运行；自动内存管理；自动垃圾回收功能（GC）
- 怎么样：汇编-> 字节码，本地运行时->JVM 运行时，只需要针对特定平台提供 JVM 实现

JVM 对内提供统一接口，对外对接本地接口

Java 虚拟机运行时



基本结构

最简单的类

```
1  /**
2  *  可以用来自动创建文档的注释
3  **/
4  public class Solution {
5      public static void main(String[] args) {
6          // 向屏幕输出文本:
7          System.out.println("Hello , world!");
8          /* 多行注释开始
9             注释内容
10            注释结束 */
11      }
12 } // class 定义结束
```

基本结构

Annotation for Cpp

- **入口方法**: main 方法为入口方法, 是程序的唯一入口
- **外壳类** (shell class): Solution 类在这里没有任何作用
- **退出码** (exit code): main 方法不返回任何值

任何 Java 源文件都是类/类的集合!

Dive-in

入口方法

- **public 修饰符**: main 方法是唯一的程序入口, 为了让 JVM 可以调用, 必须使用 public 修饰符暴露出来
- **static 静态方法**: 入口方法需要在类创建之前调用
- **void 返回值**: JVM 不接受返回值, 所以返回 void

Java 访问修饰符

访问修饰符

- public：公共访问，允许该类或成员被任何其他类访问
- protected：受保护访问，仅限于同包内的类和子类
- **包访问**：不加修饰符时，默认包访问，仅限同包内访问
- private：私有访问，仅限类内部访问。适用于不希望外部直接访问的属性或方法

Example

protected 子类访问

```
1 class Parent {  
2     protected String message = "Hello from Parent";  
3  
4     protected void displayMessage() {  
5         System.out.println(message);  
6     }  
7 }  
8  
9 class Child extends Parent {  
10     public void show() {  
11         displayMessage(); // 子类可以访问父类的受保  
12         }  
13 }
```

Example(Continue)

protected 子类访问

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Child child = new Child();  
4         child.show(); // 输出: Hello from Parent  
5     }  
6 }
```

Example

protected 包内访问

```
1 package mypackage;
2 class Parent {
3     protected String message = "Hello from Parent";
4
5     protected void displayMessage() {
6         System.out.println(message);
7     }
8 }
9
10 class NonSubclass {
11     public void accessParent() {
12         Parent parent = new Parent();
13         parent.displayMessage(); // 可访问包内的受
14     }
15 }
```

保护方法

Example(Continue)

protected 包内访问

```
1 public class Main {  
2     public static void main(String[] args) {  
3         NonSubclass example = new NonSubclass();  
4         example.accessParent(); // 输出: Hello from  
5         Parent  
6     }  
}
```

Why protected?

Annotation for Cpp

- Java: 同包内的类和子类均可以访问
- C++: 仅限子类访问

Java 访问修饰符的设计大体参考 C++

为什么允许包内访问?

Java 包机制

Java Package

- **包的定义：**Java 包 (Package) 是一种命名空间机制，用于将相关的类和接口分组
- **包的作用：**
 - **命名空间：**提供独立的命名空间，防止类和接口命名冲突
 - **访问控制：**包允许定义类和成员的访问级别 (public、protected、default 和 private)，实现不同的封装级别。
 - **模块化：**将相关的类按功能分组，建立模块化的项目结构

Example

Java Package

```
1 // 在 mypackage 包中定义一个类
2 package mypackage;
3
4 public class MyClass {
5     public void display() {
6         System.out.println("Hello from MyClass in
7 mypackage");
8     }
9 }
10 // 在其他类中导入该包
11 import mypackage.MyClass;
```

设计模式：组合

组合而非继承

组合模型中，一个对象（称为复合对象）可以包含另一个对象（称为组件对象），复合对象可以使用组件对象的行为

降低继承可能带来的耦合性

① Java 生态

② Java 基础

③ 面向对象程序设计

设计模式

设计模式 (Design Pattern)

是对软件设计中普遍存在（反复出现）的各种问题，所提出的解决方案

一题多解

同一输出可以由很多种程序结构来实现

设计模式要解决的问题是

- 实用性：解决实际需求
- 可复用性：减少代码重复
- 易维护性：方便代码维护

在 OOP 之前

- 尼克劳斯·维尔特 (Niklaus Wirth): 《算法 + 数据结构 = 程序》
- 传统设计模式 (C-Style): 结构化、过程式程序设计

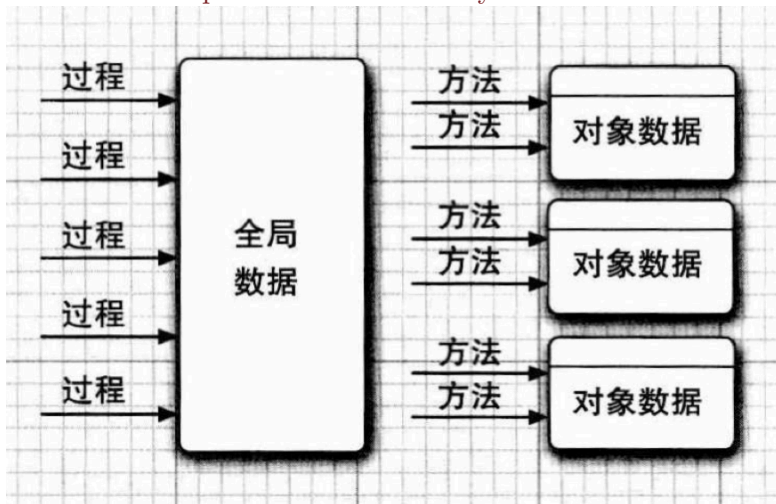
将算法放在第一位, 数据结构放在第二位

OOP: 面向对象程序设计

- 阿兰·凯 (Alan Kay): 面向对象编程的创始人之一
- 封装 (Encapsulation): 数据与操作封装在对象中
- 继承 (Inheritance): 通过继承重用代码
- 多态 (Polymorphism): 为实现同一接口的不同对象提供同一操作, 产生不同行为, 但该行为在逻辑上是相同的

将对象放在第一位, 强调数据与行为的结合

Comparison Between C-Style and OOP



对象

对象的主要特性

- **对象的行为** (behavior): 可以对对象完成哪些操作, 或者可以对对象应用哪些方法?
- **对象的状态** (state): 当调用那些方法时, 对象会如何响应?
- **对象的标识** (identity): 如何区分具有相同行为与状态的不同对象?

对象的状态与标识是每个对象独有的, 即便在同一类的实例中, 也存在差异

一些术语

- 行为 \approx 方法
- 状态 \approx 属性 \approx 字段

相对来讲，行为和状态更为抽象

方法、属性和字段通常可以特指，而行为和状态则可以理解是它们的集合

Example

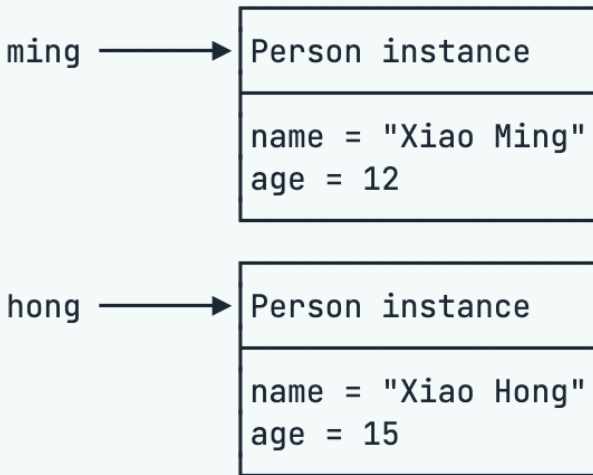
现实世界	计算机模型	Java 代码
人	类 / class	class Person { }
小明	实例 / ming	Person ming = new Person();
小红	实例 / hong	Person hong = new Person();
小军	实例 / jun	Person jun = new Person();

Example(Continue)

Person 类及其实例

```
1  class Person {  
2      public String name;  
3      public int age;  
4  }  
5  
6  Person ming = new Person();  
7  ming.name = "Xiao Ming"; // 对字段name赋值  
8  ming.age = 12; // 对字段age赋值  
9  System.out.println(ming.name); // 访问字段name  
10  
11 Person hong = new Person();  
12 hong.name = "Xiao Hong";  
13 hong.age = 15;
```

Person 类及其实例



封装性

public 带来的逻辑混乱

```
1 Person ming = new Person();  
2 ming.name = "Xiao Ming";  
3 ming.age = -99; // age 设置为负数
```

封装性

试试 private?

```
1 public class Main {
2     public static void main(String[] args) {
3         Person ming = new Person();
4         ming.name = "Xiao Ming"; // 对字段name赋值
5         ming.age = 12; // 对字段age赋值
6         // Error: not visible!!!
7     }
8 }
9
10 class Person {
11     private String name;
12     private int age;
13 }
```

this 变量

回忆

前面我们在外部代码中使用形似 `ming.name` 的语法修改 `ming` 实例的字段

如何在类的内部代码中修改当前实例的字段？

this 变量 (Continue)

变量 this 是方法内部的隐含变量，它始终指向当前实例

方法

方法允许外部代码间接修改对象状态

```
1 class Person {  
2     private String name;  
3     private int age;  
4  
5     public String getName() {  
6         return this.name;  
7     }  
8  
9     public void setName(String name) {  
10        this.name = name;  
11    }  
12    ...  
13 }
```


方法 (Continue)

```
1  class Person {  
2      ...  
3      public int getAge() {  
4          return this.age;  
5      }  
6  
7      public void setAge(int age) {  
8          if (age < 0 || age > 100) {  
9              throw new IllegalArgumentException();  
10         }  
11         this.age = age;  
12     }  
13 }
```

方法 (Continue)

现在我们可以通过方法来修改对象属性

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Person ming = new Person();  
4         ming.setName("Xiao Ming"); // 设置name  
5         ming.setAge(12); // 设置age  
6         System.out.println(ming.getName() + ", " +  
ming.getAge());  
7     }  
8 }
```

封装

封装 (Encapsulation)

是面向对象编程的核心特性之一，它指的是将对象的状态与行为封装在一起，对外部只暴露必要的接口

为什么封装？

- ① **隐藏内部实现**：外部无法直接访问和修改对象的内部数据，只能通过对象提供的方法进行操作，可减少错误和不当使用
- ② **增强安全性和可靠性**：封装性确保对象的内部状态不会因外部干扰而处于不一致的状态
- ③ **易于维护和扩展**：封装可以更改对象的内部实现而不影响使用对象的代码，从而提高代码的灵活性和可维护性
- ④ **提高代码的模块化**：封装使得每个对象可以被视为一个独立的模块，便于开发者更好地理解和管理复杂的系统

Thanks!

Reference

- [1] Cay S. Horstmann. Core Java: Fundamentals, Volume 1. Addison-Wesley Professional, 2018.
- [2] 廖雪峰. 廖雪峰 Java 教程.