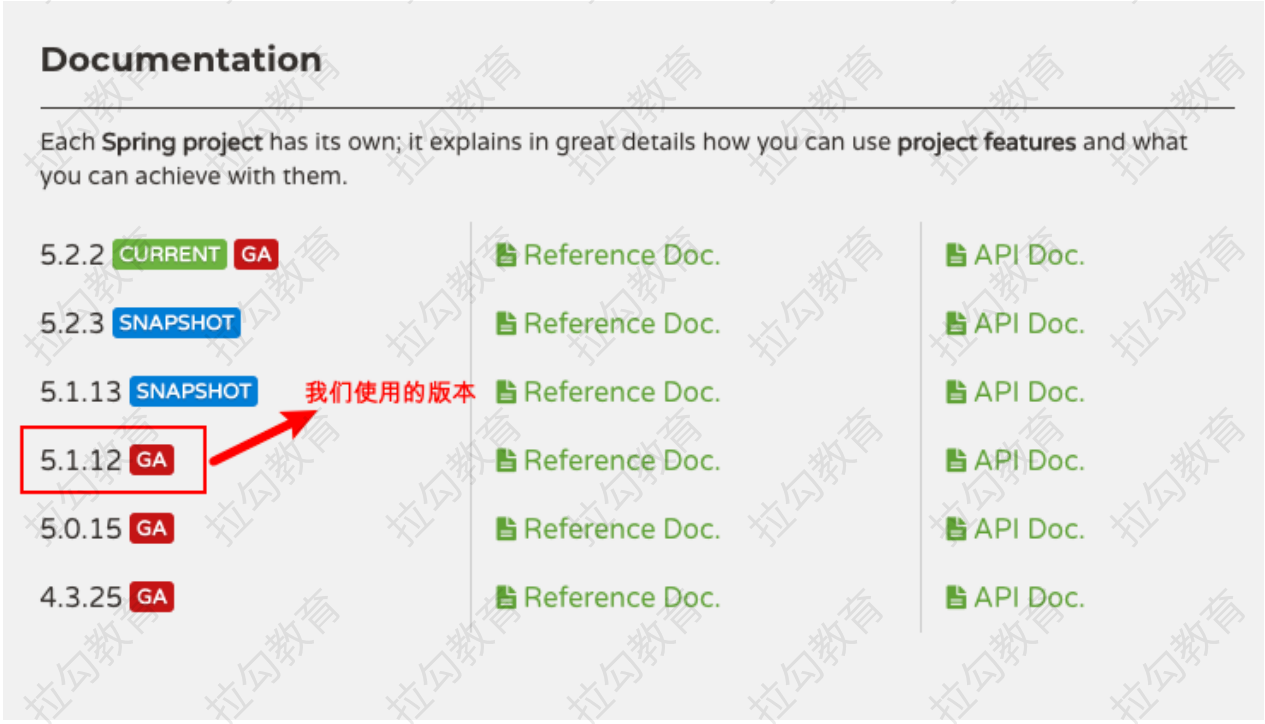
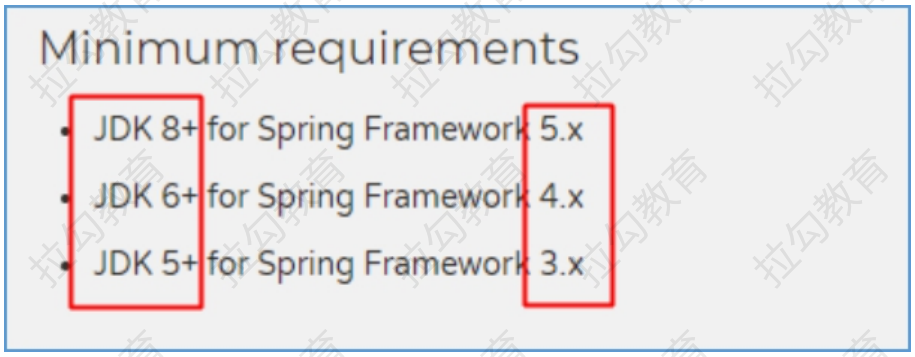


Spring Framework不同版本对 Jdk 的要求



JDK 11.0.5

IDE idea 2019

Maven 3.6.x

第二部分 核心思想

注意：IOC和AOP不是spring提出的，在spring之前就已经存在，只不过更偏向于理论化，spring在技术层次把这两个思想做了非常好的实现（Java）

第1节 IoC

1.1 什么是IoC?

IoC Inversion of Control (控制反转/反转控制)，注意它是一个技术思想，不是一个技术实现

描述的事情：Java开发领域对象的创建，管理的问题

传统开发方式：比如类A依赖于类B，往往会在类A中new一个B的对象

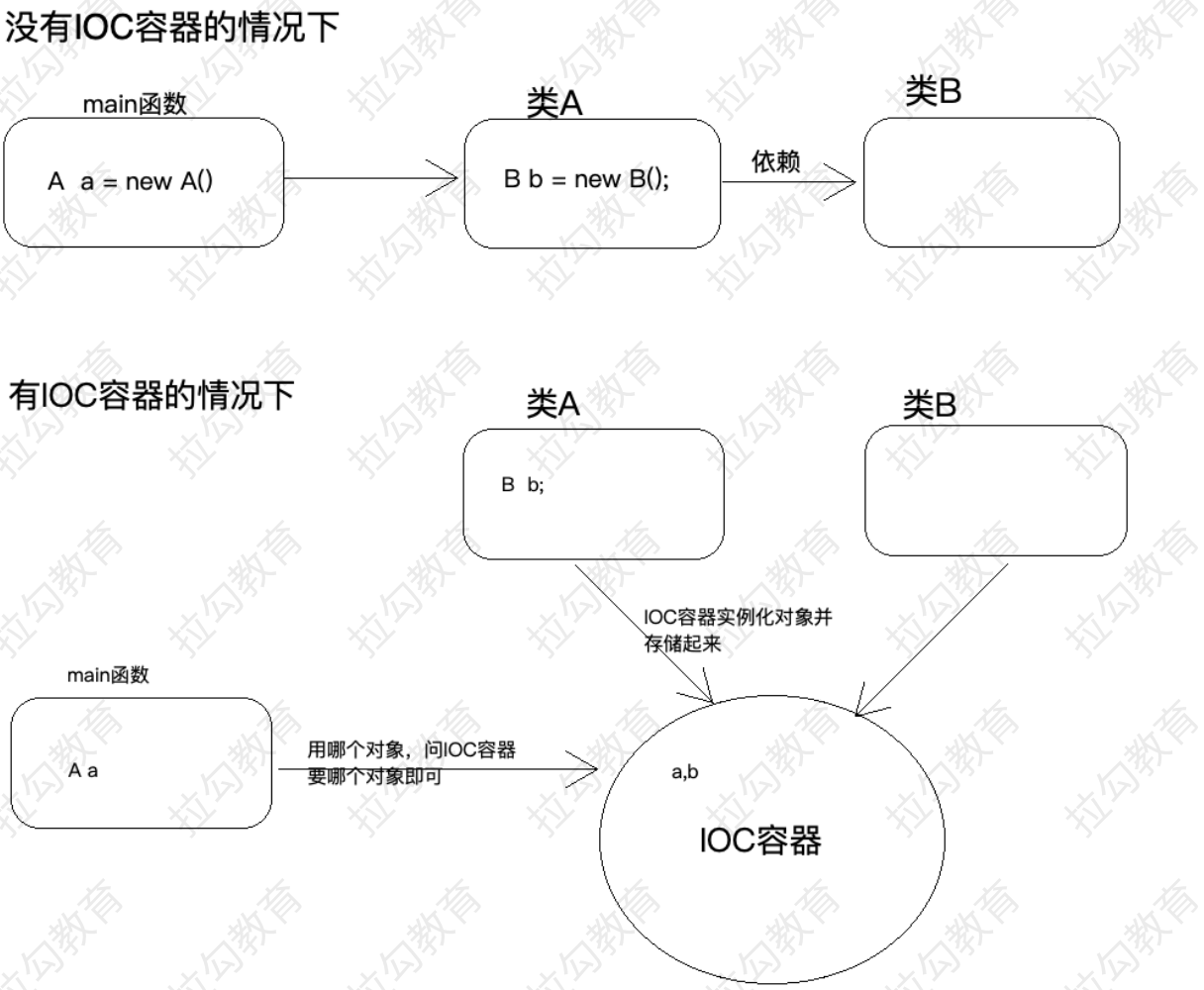
IoC思想下开发方式：我们不用自己去new对象了，而是由IoC容器（Spring框架）去帮助我们实例化对象并且管理它，我们需要使用哪个对象，去问IoC容器要即可

我们丧失了一个权利（创建、管理对象的权利）,得到了一个福利（不用考虑对象的创建、管理等一系列事情）

为什么叫做控制反转?

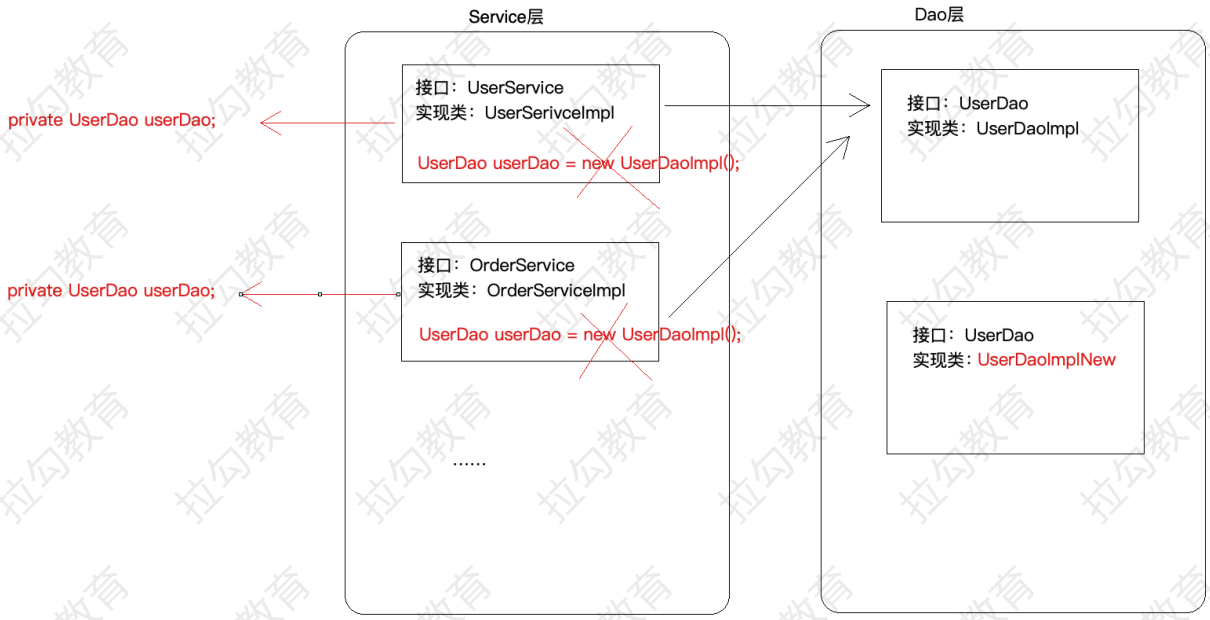
控制：指的是对象创建（实例化、管理）的权利

反转：控制权交给外部环境了（spring框架、IoC容器）



1.2 IoC解决了什么问题

IoC解决对象之间的耦合问题

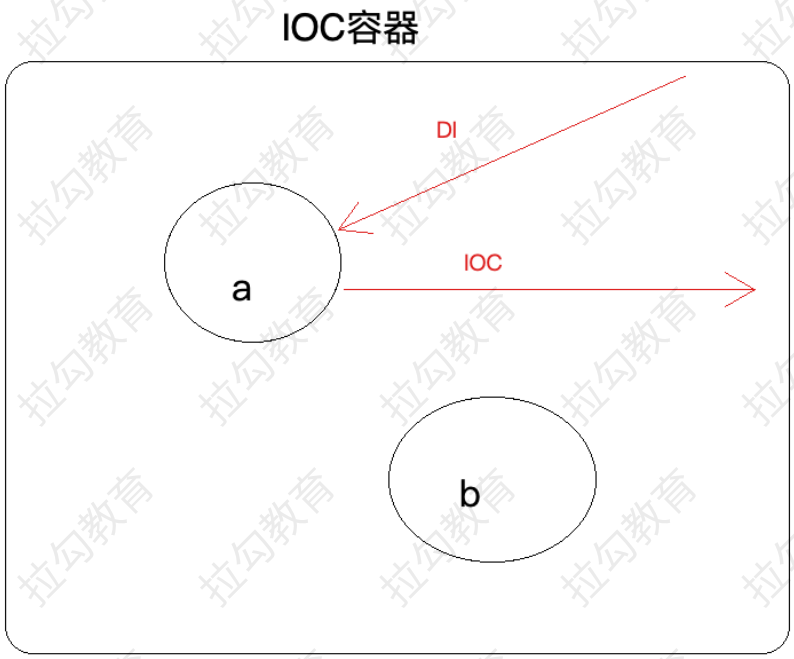


1.3 IoC和DI的区别

DI：Dependancy Injection（依赖注入）

怎么理解：

IOC和DI描述的是同一件事情，只不过角度不一样罢了



IOC和DI描述的是同一件事情（对象实例化及依赖关系维护这件事情），只不过角度不同罢了

IOC是站在对象的角度，对象实例化及其管理的权利交给了（反转）给了容器

DI是站在容器的角度，

容器会把对象依赖的其他对象注入（送进去），比如A对象实例化过程中因为声明了一个B类型的属性，那么就需要容器把B对象注入给A

第2节 AOP

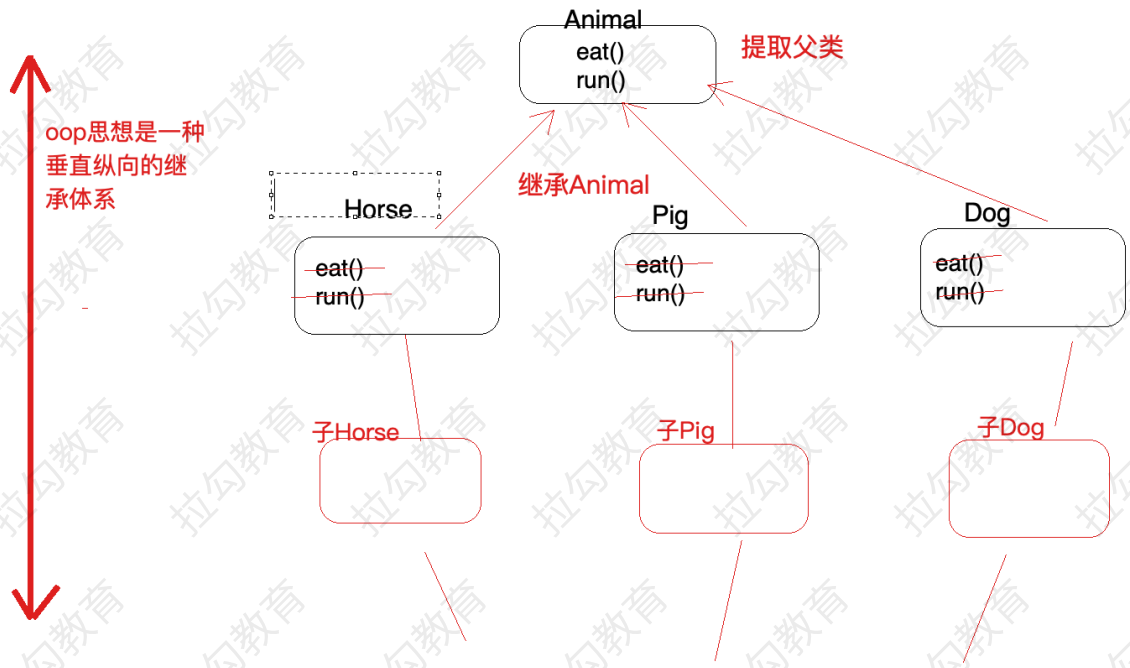
2.1 什么是AOP

AOP: Aspect oriented Programming 面向切面编程/面向方面编程

AOP是OOP的延续，从OOP说起

OOP三大特征：封装、继承和多态

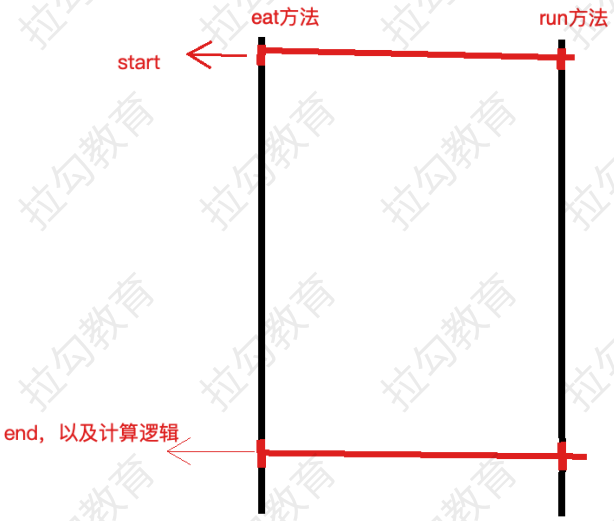
oop是一种垂直继承体系



OOP编程思想可以解决大多数的代码重复问题，但是有一些情况是处理不了的，比如下面的在顶级父类Animal中的多个方法中相同位置出现了重复代码，OOP就解决不了

```
 * @author 应癫
 */
public class Animal {

    private String height; // 高度
    private float weight;  // 体重

    public void eat(){

        // 性能监控代码
        long start = System.currentTimeMillis();

        // 业务逻辑代码
        System.out.println("I can eat...");

        // 性能监控代码
        long end = System.currentTimeMillis();
        System.out.println("执行时长: " + (end-start)/1000f + "s");
    }

    public void run(){
        // 性能监控代码
        long start = System.currentTimeMillis();

        // 业务逻辑代码
        System.out.println("I can run...");

        // 性能监控代码
        long end = System.currentTimeMillis();
        System.out.println("执行时长: " + (end-start)/1000f + "s");
    }

}
```

横切逻辑代码



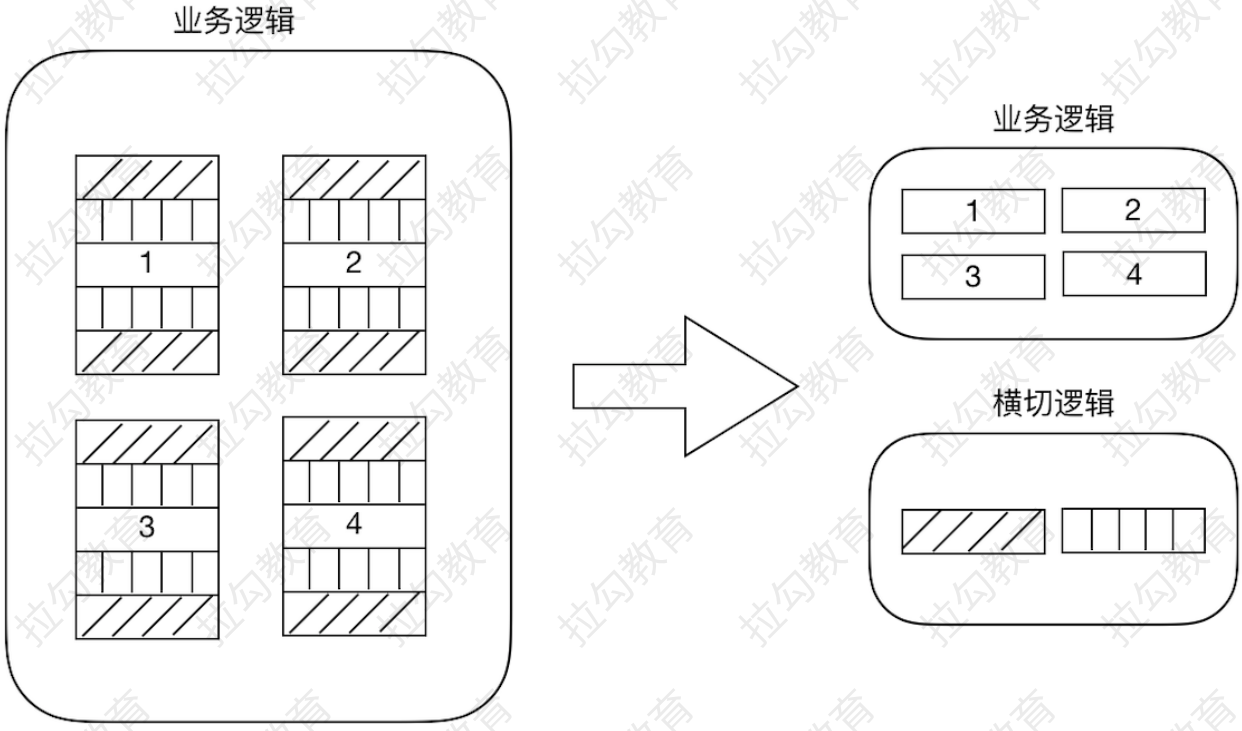
在多个纵向（顺序）流程中出现的相同子流程代码，我们称之为横切逻辑代码

横切逻辑代码的使用场景很有限：一般是事务控制、权限校验、日志

横切逻辑代码存在什么问题：

- 横切代码重复问题
- 横切逻辑代码和业务代码混杂在一起，代码臃肿，维护不方便

AOP出场，AOP独辟蹊径提出横向抽取机制，将横切逻辑代码和业务逻辑代码分析



代码拆分容易，那么如何在不改变原有业务逻辑的情况下，悄无声息的把横切逻辑代码应用到原有的业务逻辑中，达到和原来一样的效果，这个是比较难的

2.2 AOP在解决什么问题

在不改变原有业务逻辑情况下，增强横切逻辑代码，根本上解耦合，避免横切逻辑代码重复