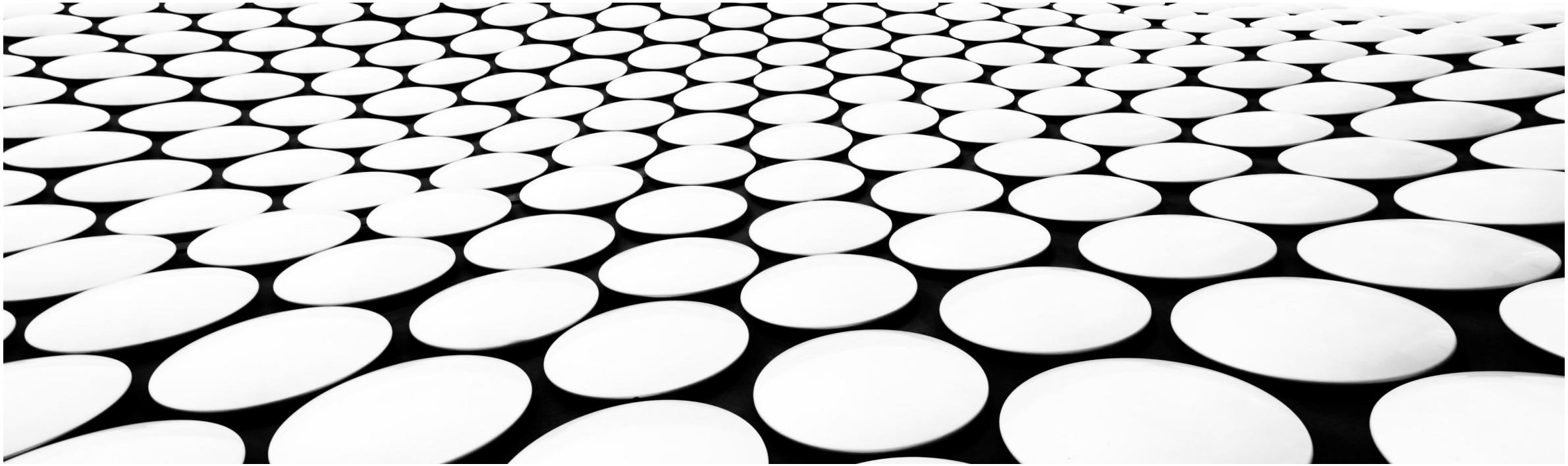


服务端开发-面向切面编程（AOP）

回顾-告诉容器创建Bean，以及如何组装它们

1、自动化配置

```java
@Component
public class CDPlayer implements MediaPlayer {
    private CompactDisc cd;

    @Autowired
    public CDPlayer(CompactDisc cd) {
        this.cd = cd;
    }

    public void play() {
        cd.play();
    }
```

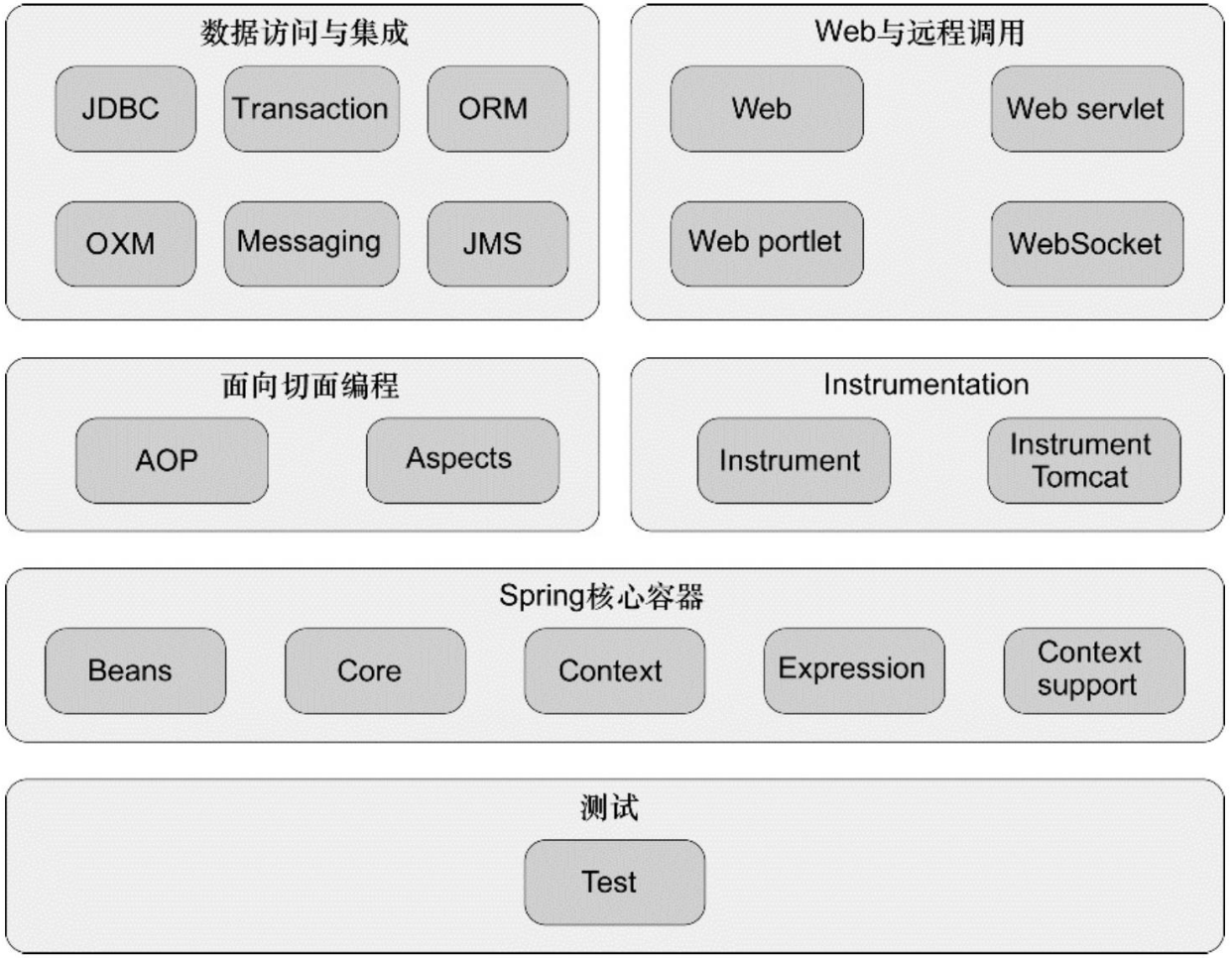
2、 JavaConfig

```java
@Configuration
public class CDPlayerConfig {

    @Bean
    public CompactDisc compactDisc() {
        return new SgtPeppers();
    }

    @Bean
    public CDPlayer cdPlayer(CompactDisc cd) {
        return new CDPlayer(cd);
    }
```

3、XML配置

```xml
<bean id="compactDisc" class="soundsystem.SgtPeppers" />

<bean id="cdPlayer" class="soundsystem.CDPlayer">
  <constructor-arg ref="compactDisc" />
</bean>
```

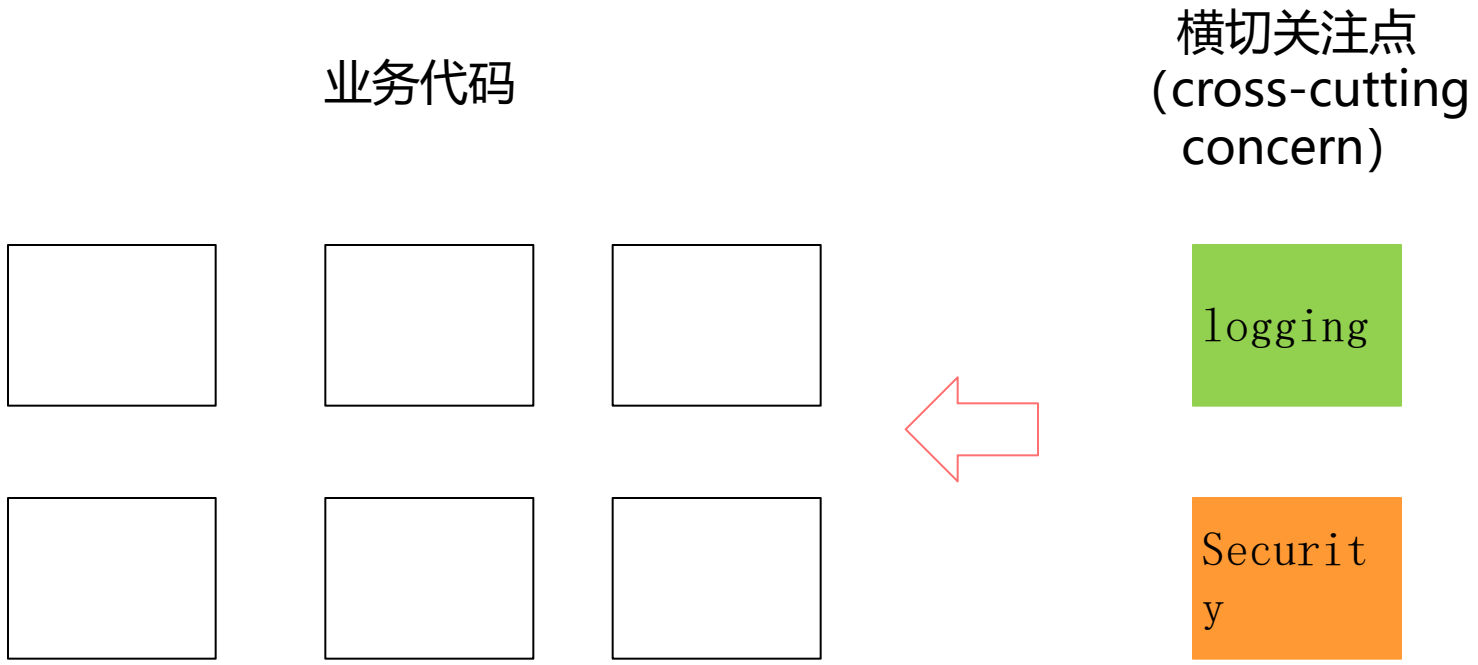
Spring的模块组成



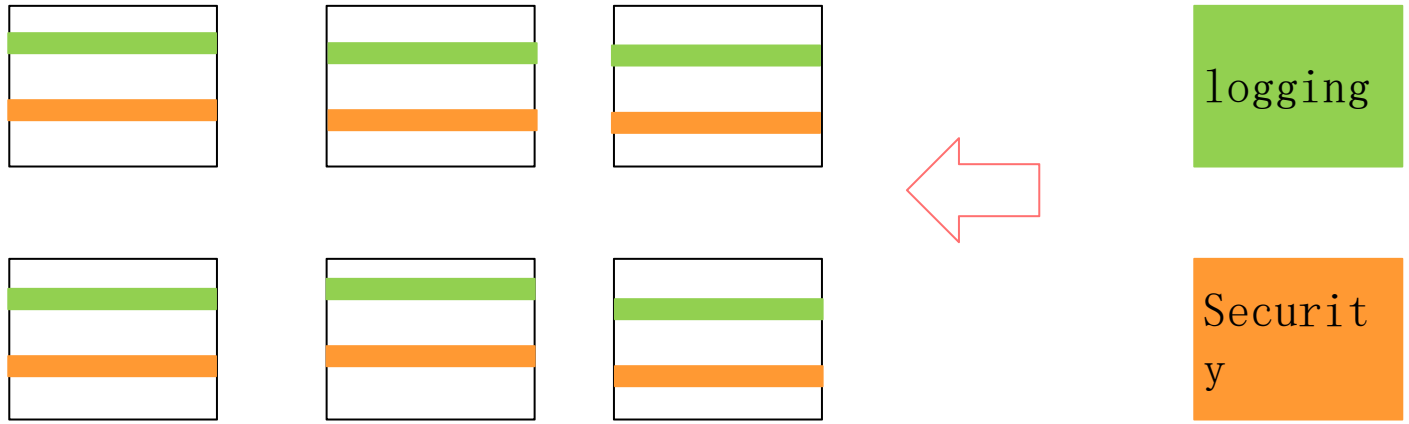
软件编程方法的发展

- 面向过程编程（POP，Procedure Oriented Programming)
- 面向对象编程（OOP，Object Oriented Programming）
- 面向切面编程（AOP，Aspect Oriented Programming）
- 函数式编程（FP， Functional Programming）
- 反应式编程（RP，Reactive Programming）

AOP：Aspect Oriented Programming



切入后



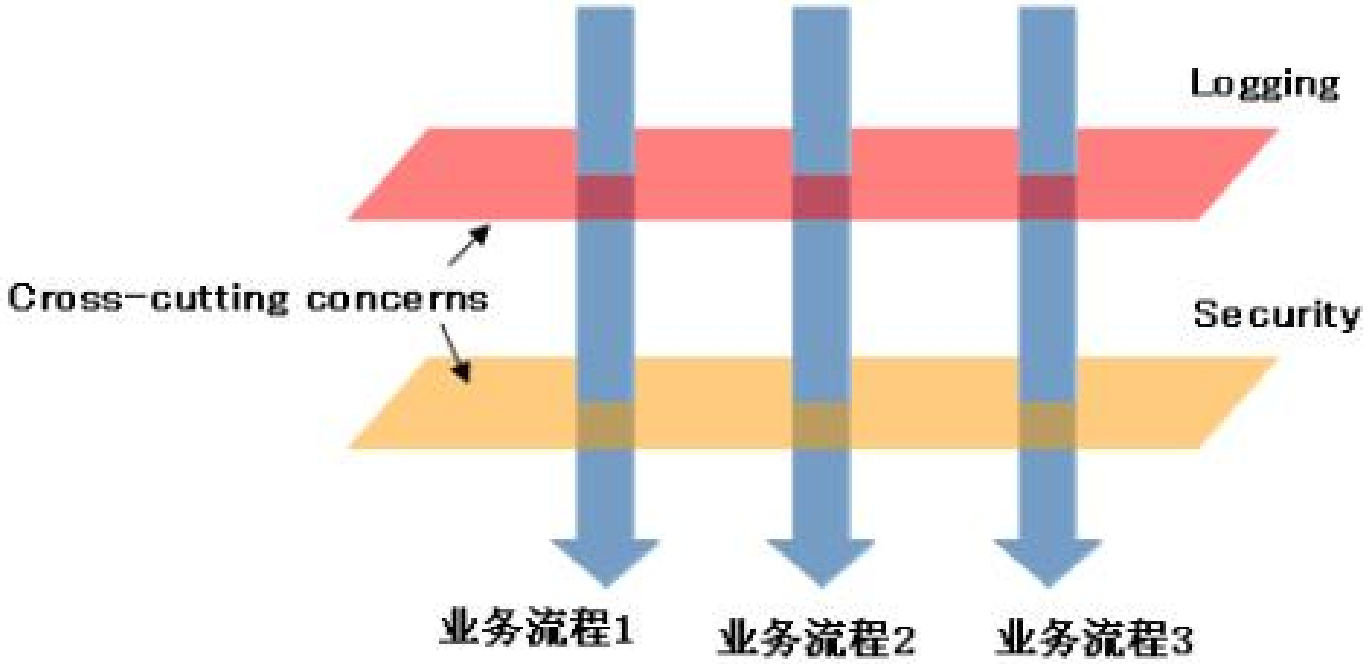
横切关注点（cross-cutting concern）

- 日志
- 安全
- 事务
- 缓存

可选

- 继承（inheritance）
- 委托（delegation）

AOP图解


Logging
Cross-cutting concerns
Security
业务流程1
业务流程2
业务流程3

AOP图解

AOP术语

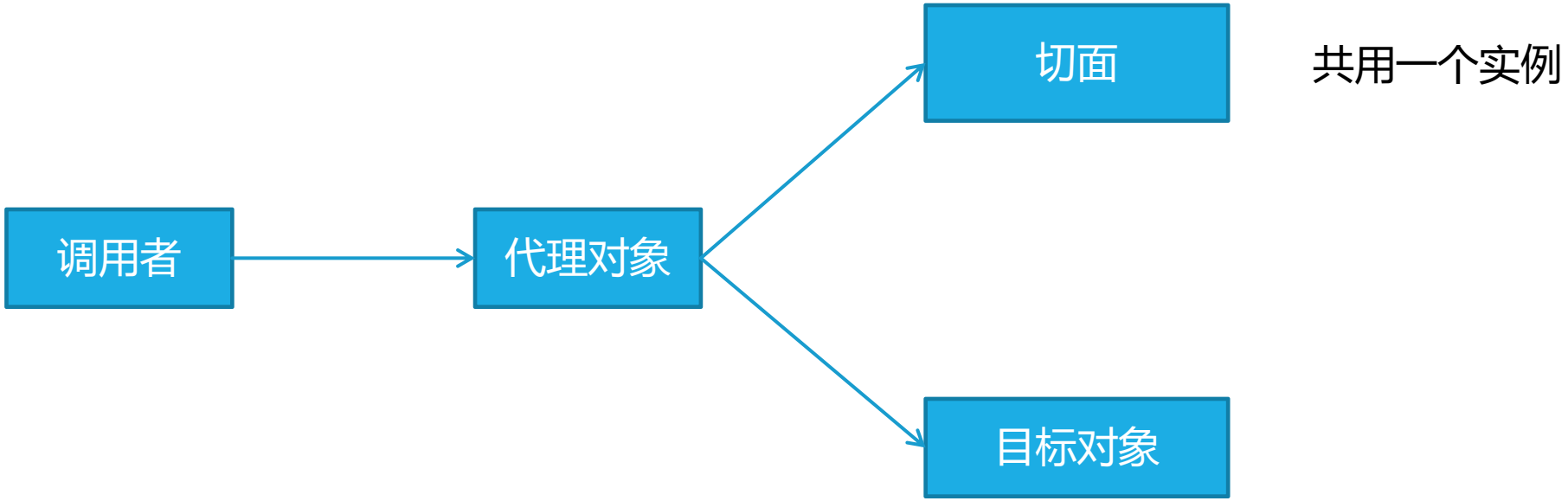
- 通知（Advice）：切面做什么以及何时做
- 切点（Pointcut）：何处
- 切面（Aspect）：Advice和Pointcut的结合
- 连接点（Join point）：方法、字段修改、构造方法
- 引入（introduction）：引入新的行为和状态
- 织入（Weaving）：切面应用到目标对象的过程

通知（Advice）类型

- @Before
- @After
- @AfterReturning
- @AfterThrowing
- @Around

织入时机

- 编译期，需要特殊的编译器
- 类加载期，需要类加载器的处理
- 运行期： Spring所采纳的方式，使用代理对象、只支持方法级别的连接点



共用一个实例

JDK提供的代理对象

- Proxy.newProxyInstance

Spring AOP

- @AspectJ注解驱动的切面
- @EnableAspectJAutoProxy //开启AspectJ的自动代理机制

定义切面（@Aspect）

- 加注解的普通POJO
- 定义可重用的切点
- Around通知
- 定义参数（CD），测试

AspectJ 切点指示器（pointcut designator）

- 例子

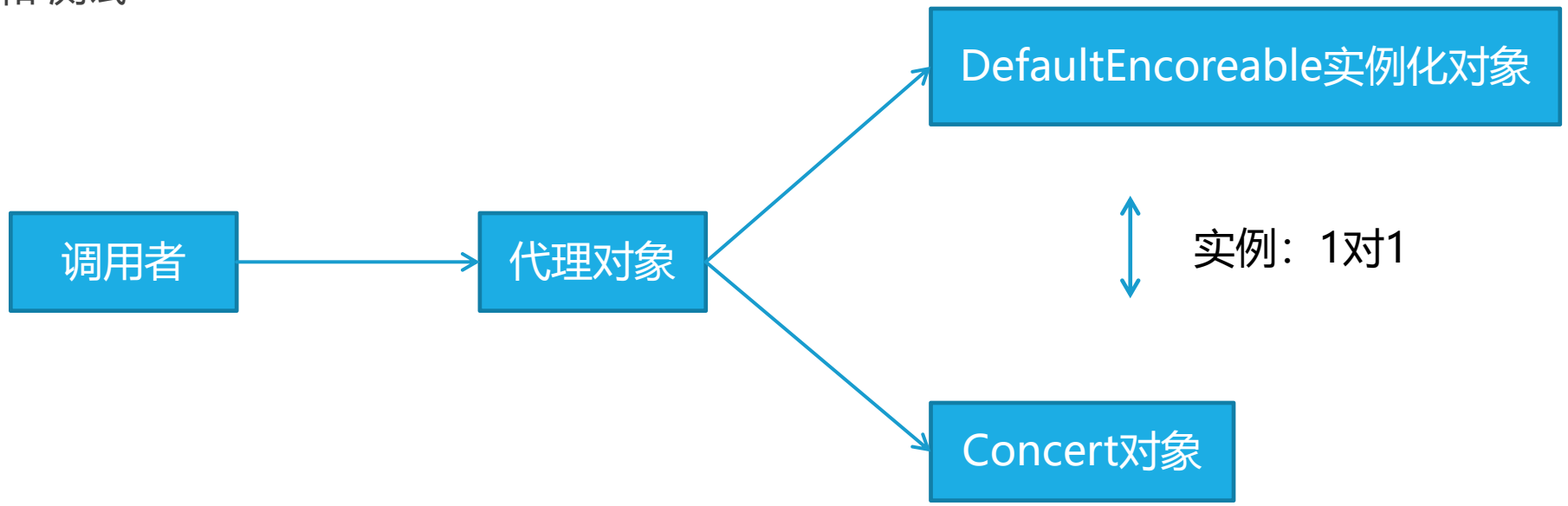
```
@Pointcut(
        "execution(* soundsystem.CompactDisc.playTrack( int )) " +
            "&& args(trackNumber) //获取参数
&& within(soundsystem.*)    //限定包路径
&& bean(sgtPeppers) ")    //限定bean名称，或者：  && !bean(sgtPeppers)
```

- 另一个例子

```
@Around("@annotation(innerAuth)")    //限定注解
 public Object innerAround(ProceedingJoinPoint point, InnerAuth innerAuth) { ... }

@InnerAuth
public R<Boolean> register(@RequestBody SysUser sysUser) { ... }
```

引入接口(introduction)

- @DeclareParents
- main 和 测试

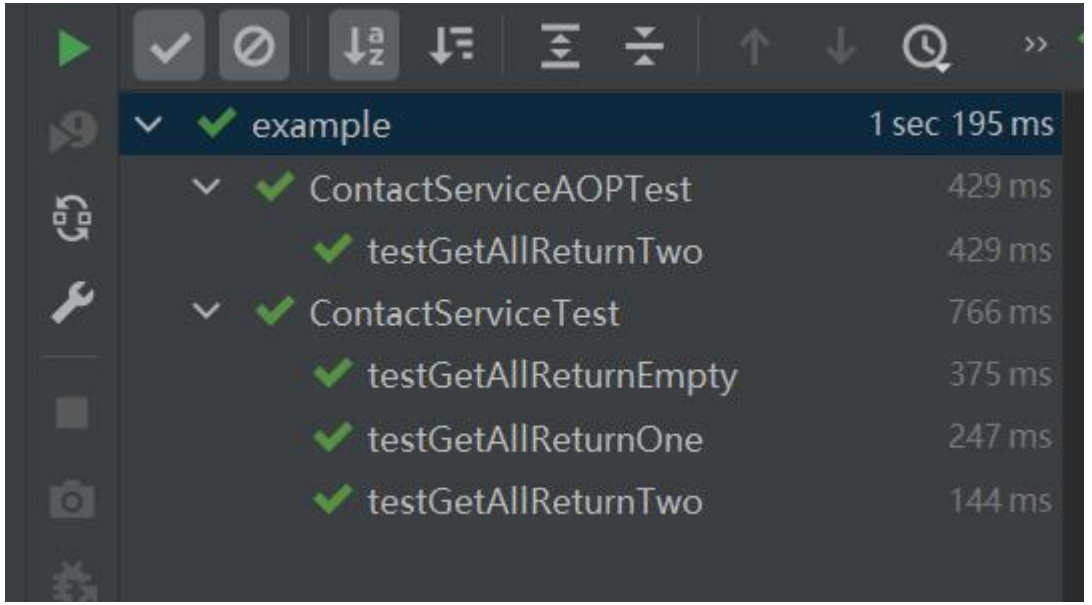


在XML中声明切面

- 前置和后置通知
- <aop:pointcut>
- 声明环绕通知
- 传参数，测试
- 引入新功能

作业

- 目标：在前一次课的作业基础上使用AOP织入的方式给ContactService.getAll增加缓存处理，使的每次对getAll方法的调用都返回固定的两条记录（参看提供的测试代码ContactServiceAOPTest）
- 要求：前一次课作业的测试代码（ContactServiceTest）、本次课提供的测试代码（ContactServiceAOPTest）都能同时测试通过（如右图）



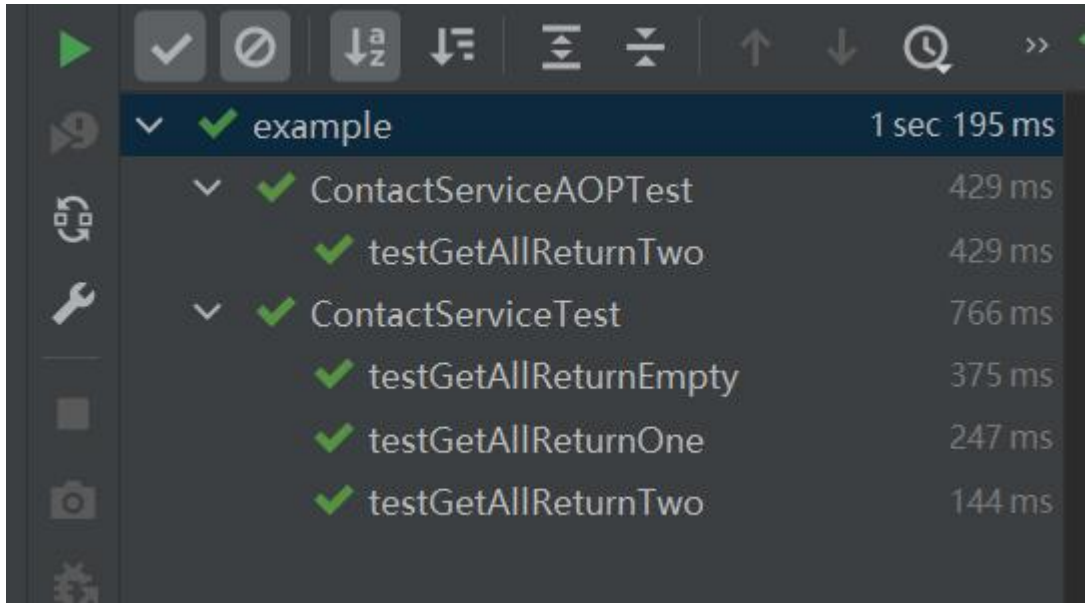
已提供的文件

- Contact.java，领域类，表示一个人的联系信息，上次已提供
- ContactService.java，业务类接口，上次已提供
- ContactRepository.java，数据访问层接口，上次已提供
- ContactServiceTest.java，测试代码，上次已提供
- ContactServiceAOPTest.java，本次新增的测试代码

提交

- 所有源代码，压缩成一个文件。不包含编译后的class文件（删除target目录）

- 测试成功的截图



谢谢观看！