JavaEE平台技术
面向切面编程

邱明 博士

厦门大学信息学院

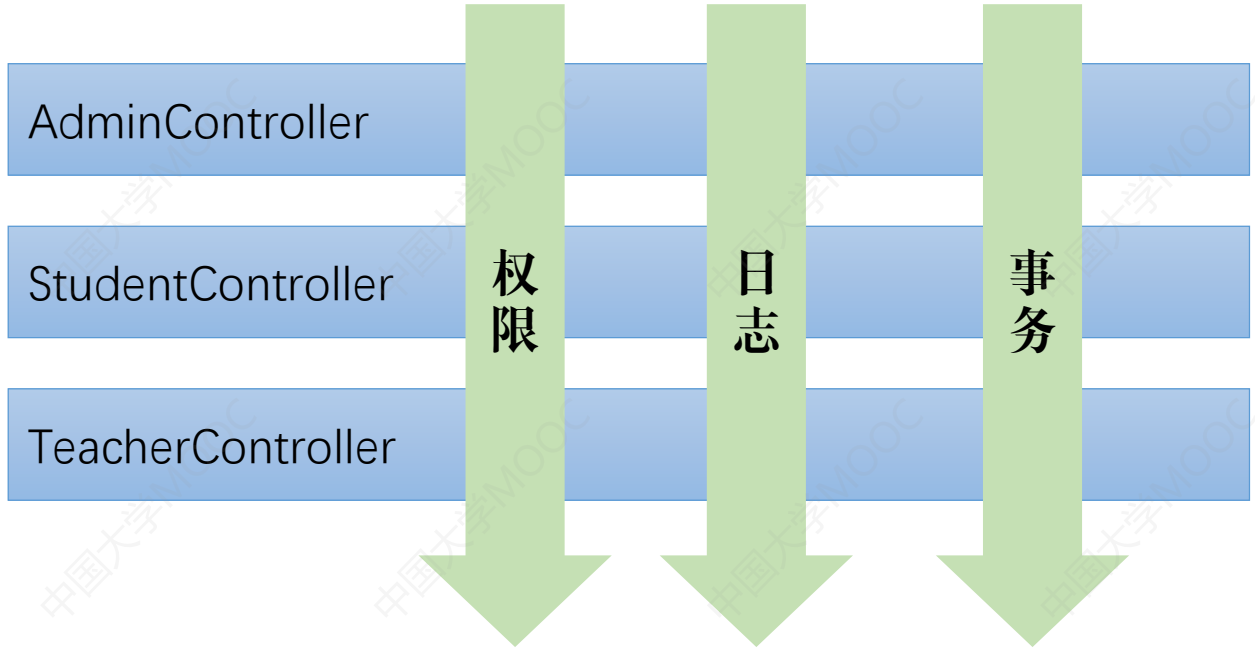
mingqiu@xmu.edu.cn

提纲

- AOP是什么
- Spring的AOP实现机制
- 定义Pointcut
- 定义Aspect
- AOP的例子-JWT认证

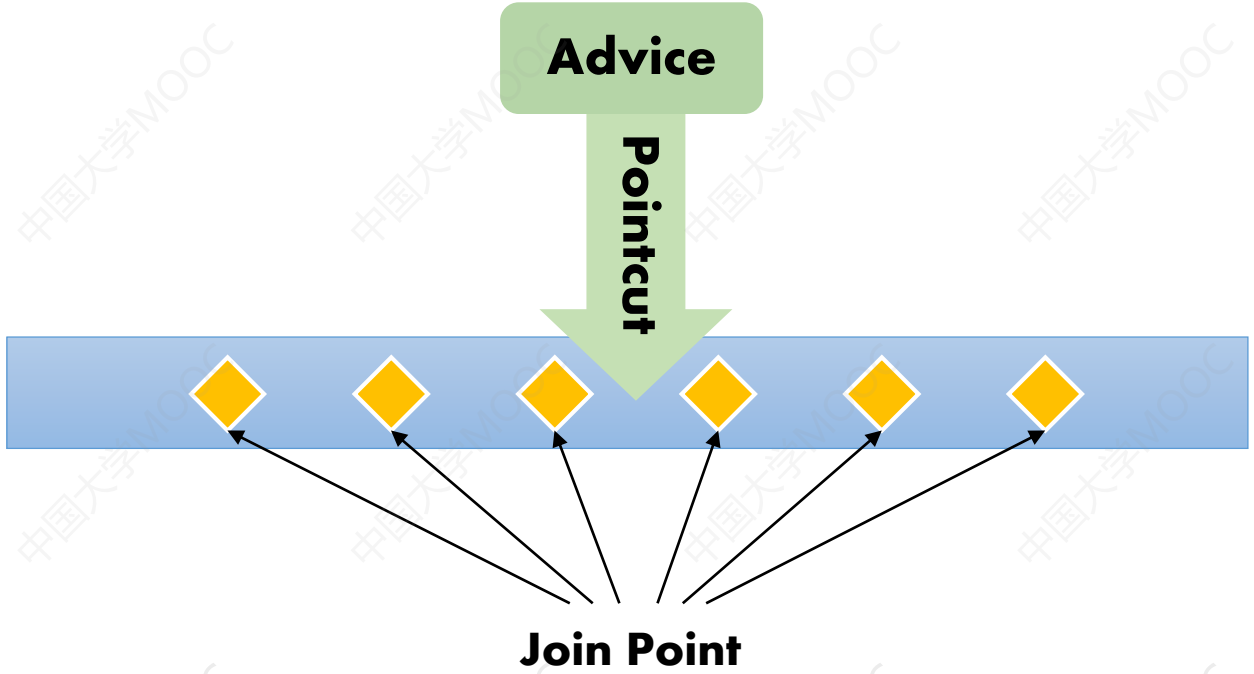
1. AOP是什么

- 面向方面编程



1. AOP是什么

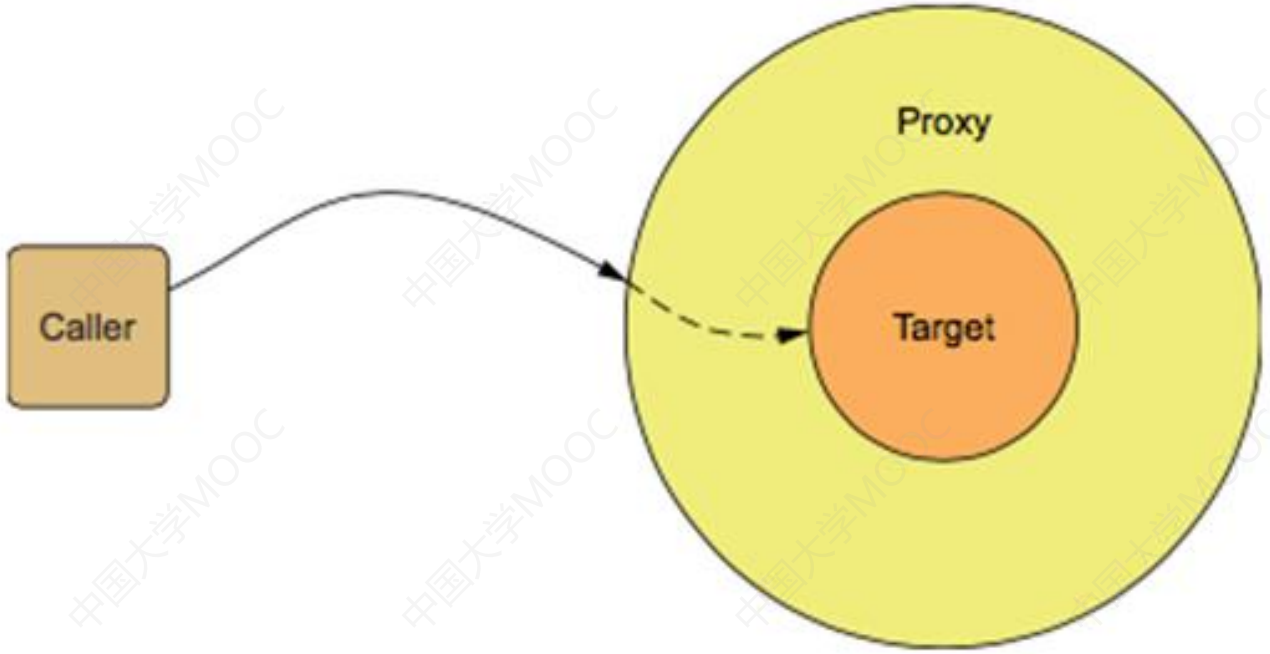
- 专有名词



1. AOP是什么

- 五种Advice
  - @Before – 在Joint Point执行之前
  - @After – 在Joint Point执行之后，无论是否成功执行
  - @AfterReturning – 在Joint Point成功执行之后
  - @AfterThrowing – 在Joint Point抛出异常之后
  - @Around – 在Joint Point运行之前和之后

2. Spring的AOP的实现机制



3.定义Pointcut

- Spring采用AspectJ的定义符定义Pointcut

定义符	描述
args()	Join point是方法，且参数是某些特定类型
@args()	Join point是方法，且其参数用特定注解标注
execution()	Join point是方法，其方法名称满足特定条件
@annotation	Joint point用特定注解标注

3.定义Pointcut

```
package concert;

public interface Performance {
  public void perform();
}
```

返回类型　包和类名　方法名　任意参数

execution(* concert.Performance.perform(..))

方法的调用　方法需满足的条件

execution(* concert.Performance.perform(..))&&within(concert.*)

在concert包内调用

4. 定义Aspect

```
package concert;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class Audience {

  @Before("execution(** concert.Performance.perform(..))")
  public void silenceCellPhones() {
    System.out.println("Silencing cell phones");
  }

  @Before("execution(** concert.Performance.perform(..))")
  public void takeSeats() {
    System.out.println("Taking seats");
  }

  @AfterReturning("execution(** concert.Performance.perform(..))")
   public void applause() {
    System.out.println("CLAP CLAP CLAP!!!");
  }

  @AfterThrowing("execution(** concert.Performance.perform(..))")
   public void demandRefund() {
    System.out.println("Demanding a refund");
  }

}
```

Before performance

Before performance

After performance

After bad performance

5. AOP的例子-JWT认证