CS544 EA

Spring

AOP: Working with Data Inside Advice

Data and Advice Methods

- There are several ways you can receive data in an advice method
  - Through the JoinPoint (args, target, this)
  - Return value / Thrown exceptions
  - Injected into the Aspect object (eg. DAOs)

Injected Objects

- An Aspect class is **just another bean**
  - Can have objects injected just like any other bean
  - Useful: you can inject DAOs to retrieve additional data from DB

```
package cs544.spring43.aop.data;
...

@Component
@Aspect
public class InjectAspect {
	@Autowired
	private PersonDao personDao;

	@Before("execution(* cs544.spring43.aop.data.CustomerService.setName(String))")
	public void argsBefore(JoinPoint jp) {
		Object[] args = jp.getArgs();
		String name = (String)args[0];
		Person p = personDao.byName(name);
		if (p.getAge() > 18) {
			System.out.println("adult");
		}
	}
}
```

Arguments

- `jp.getArgs()` returns an `Object[]`
  - Spring does not know the types of the args
  - **You have to cast** them yourself

```
package cs544.spring43.aop.data;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class TestAspect {
	@Before("execution(* cs544.spring43.aop.data.CustomerService.setName(String))")
	public void argsBefore(JoinPoint jp) {
		Object[] args = jp.getArgs();
		String name = (String)args[0];
		System.out.println("Argument value: " + name);
	}
}
```

Pointcut args() Designator

- It is also possible to receive incoming args directly **into the advice method**
  - Use args() pointcut to specify names instead of types
  - A bit slower (more CPU) than using JoinPoint

```
package cs544.spring43.aop.data;

...

@Aspect
@Component
public class TestAspect {
	@Before("execution(* cs544.spring43.aop.data.CustomerService.setName(String)) "
			+ " && args(name))")
	public void argsBefore(JoinPoint jp, String name) {
		System.out.println("Argument value: " + name);
	}
}
```

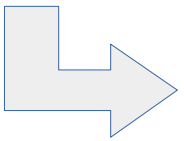
Changing Args

- The @Around advice has the additional possibility of **changing the argument** values
  - Before giving them to the real method

```
package cs544.spring43.aop.data;

...

@Aspect
@Component
public class TestAspect {

	@Around("execution(* cs544.spring43.aop.data.CustomerService.setName(String))")
	public Object aroundSetName(ProceedingJoinPoint pjp) throws Throwable {
		Object[] args = pjp.getArgs();
		System.out.println("Argument value: " + args[0]);
		args[0] = "James";
		return pjp.proceed(args);
	}

}
```

Changing Args Demo

```
package cs544.spring43.aop.data;

import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {
  public static void main(String[] args) {
    ConfigurableApplicationContext context;
    //context = new ClassPathXmlApplicationContext("cs544/spring43/aop/data/springconfig.xml");
    context = new AnnotationConfigApplicationContext(Config.class);
    ICustomerService cs = context.getBean("customerService", ICustomerService.class);
    cs.setName("John");
    System.out.println("Inside cs: " + cs.getName());


    context.close();
  }
}
```



```
Argument value: John
Inside cs: James
```

Return Value

- @AfterReturning can receive the return

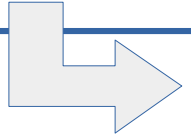
```
package cs544.spring41.aop.advices;

...

@Aspect
@Component
public class TestAspect {
	@AfterReturning(pointcut="execution(* cs544.spring41.aop.advices.CustomerService.getName())", returning="ret")
	public void afterRet(JoinPoint jp, String ret) {
		System.out.println(jp.getSignature().getName() + " returned: " + ret);
	}
}
```

Changing return value

- @Around can also **change the return** value

```
package cs544.spring43.aop.data;

...

@Aspect
@Component
public class TestAspect {
	@Around("execution(* cs544.spring43.aop.data.CustomerService.getName())")
	public Object aroundGetName(ProceedingJoinPoint pjp) throws Throwable {
		Object name = pjp.proceed();
		return "Chris";
	}
}
```

```
public class App {
  public static void main(String[] args) {
    ConfigurableApplicationContext context;
    context = new AnnotationConfigApplicationContext(Config.class);
    ICustomerService cs = context.getBean("customerService", ICustomerService.class);
    cs.setName("John");
    System.out.println("From cs: " + cs.getName());
  }
}
```



```
From cs: Chris
```

Exception

- @AfterThrowing can receive the exception
  - Cannot stop or alter it!

```
package cs544.spring41.aop.advices;

...

@Aspect
@Component
public class TestAspect {
	@AfterThrowing(pointcut="execution(* cs544.spring41.aop.advices.CustomerService.getAge(..))", throwing="ex")
	public void afterThrow(JoinPoint jp, MyException ex) {
		System.out.println(jp.getSignature().getName() + " threw a: " + ex.getClass().getName());
	}
}
```

Changing the Exception

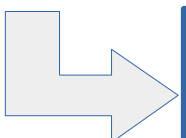
- @Around can **catch the exception** and choose:
  - Re-throw the same exception
  - Throw another exception
  - Don’t throw anything (stop the exception)

```
package cs544.spring43.aop.data;

...

@Aspect
@Component
public class TestAspect {
	@Around("execution(* cs544.spring43.aop.data.CustomerService.exception())")
	public Object aroundException(ProceedingJoinPoint pjp) {
		try {
			return pjp.proceed();
		} catch (Throwable e) {
			throw new OtherException();
		}
	}
}
```

Other Exception Demo

```java
package cs544.spring43.aop.data;

import org.springframework.stereotype.Service;

@Service
public class CustomerService implements ICustomerService {
	@Override
	public void exception() {
		throw new MyException();
	}
}
```

```java
public class App {
  public static void main(String[] args) {
    ConfigurableApplicationContext context;
    context = new AnnotationConfigApplicationContext(Config.class);
    ICustomerService cs = context.getBean("customerService", ICustomerService.class);
    cs.exception();
  }
}
```



```
Exception in thread "main" cs544.spring43.aop.data.OtherException
	at cs544.spring43.aop.data.TestAspect.aroundException(TestAspect.java:32)
	at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
	at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
	at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
	at java.lang.reflect.Method.invoke(Method.java:498)
```

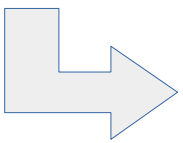
Full Power of @Around 1/2

```java
package cs544.spring43.aop.data;
...
@Component
public class Calculator implements ICalculator {
	public int add(int x, int y) {
		System.out.println("Calculator.add receiving: x= " + x + " and y= " + y);
		return x + y;
	}
}
```

```java
package cs544.spring43.aop.data;
...
@Aspect
@Component
public class CalcAspect {
	@Around("execution(* cs544.spring43.aop.data.Calculator.add(..))")
	public Object changeNumbers(ProceedingJoinPoint pjp) {
		Object[] args = pjp.getArgs();
		int x = (Integer) args[0];
		int y = (Integer) args[1];
		System.out.println("CalcAdvice.changeNumbers: x= " + x + " and y= " + y);

		args[0] = 5;
		args[1] = 9;
		Object object = null;
		try { object = pjp.proceed(args);
		} catch (Throwable e) { /* do nothing*/ }
		System.out.println("CalcAdvice.changeNumbers: call.proceed returns " + object);
		return 26;
	}
}
```

Full Power of @Around 2/2

```
package cs544.spring43.aop.data;
...
public class App {
	public static void main(String[] args) {
		ConfigurableApplicationContext context;

		ICalculator calc = context.getBean("calculator", ICalculator.class);
		int result = calc.add(3, 4);
		System.out.println("The result of 3 + 4 = " + result);

		context.close();
	}
}
```



```
CalcAspect.changeNumbers: x= 3 and y= 4
Calculator.add receiving: x= 5 and y= 9
CalcAdvice.changeNumbers: call.proceed returns 14
The result of 3 + 4 = 26
```

jp.getTarget() and jp.getThis()

- You can ask the JoinPoint for the target object
  - Or the calling object (provided by jp.getThis)
  - Sometimes these have **useful data or DAOs**

- Be aware though:
  - Calling methods on these objects will be without AOP!
  - See next section (video) for more info