Spring Core
Spring AOP

LUXOFT TRAINING

LXFT LISTED NYSE

AOP :: Пример

- Рассмотрим метод получения пользователя по id:

```
public class UserService {
  public UserDTO getUser(Integer id) {
     return userDAO.getUser(id);
  }
}
```

AOP :: Пример

- Рассмотрим метод получения пользователя по id:

```
public class UserService {
   public UserDTO getUser(Integer id) {
       return userDAO.getUser(id);
   }
}
```

- Добавим логгирование:

```
public UserDTO getUser(Integer id) {
   log.debug("Call method getUser with id " + id);
   UserDTO user = userDAO.getUser(id);
   log.debug("User info is: " + user.toString());
   return user;
}
```

AOP :: Пример

- Добавим обработку исключений:

```
public UserDTO getUser(Integer id) throws ServiceException{
 log.debug("Call method getUser with id " + id);
 UserDTO user = null;
   UserDTO user = userDAO.getUser(id);
 try {
       user = userDAO.getUser(id);
   } catch(SQLException e) {
   throw new ServiceException(e);
   }

 log.debug("User info is: " + user.toString());
 return user;
}
```

AOP :: Пример

- Добавим проверку прав пользователя:

```
public UserDTOgetUser(Integer id) throwsServiceException, AuthException{
 if (!SecurityContext.getUser().hasRight("getUser")) {
  thrownewAuthException("Permission Denied");
    }

 log.debug("Call method getUser with id " + id);
 UserDTO user = null;
    UserDTO user = userDAO.getUser(id);

 try {
        user = userDAO.getUser(id);
    } catch(SQLException e) {
  thrownewServiceException(e);
    }

 log.debug("User info is: " + user.toString());
 return user;
}
```

AOP :: Пример

- Добавляем кэширование результатов работы:

```
public UserDTO getUser(Integer id) throws ServiceException, AuthException {
 …
 try {
  if (cache.contains(cacheKey)) {
             user = (UserDTO) cache.get(cacheKey);
        } else {
             user = userDAO.getUser(id);
   cache.put(cacheKey, user);
        }
   } catch(SQLException e) {
  throw new ServiceException(e);
   }
 log.debug("User info is: " + user.toString());
 return user;
}
```

AOP :: Пример

- **Что мы получаем:**
  - Большой объем сервисного кода
  - Вместо одной строки мы получили 16. И это количество продолжает расти…

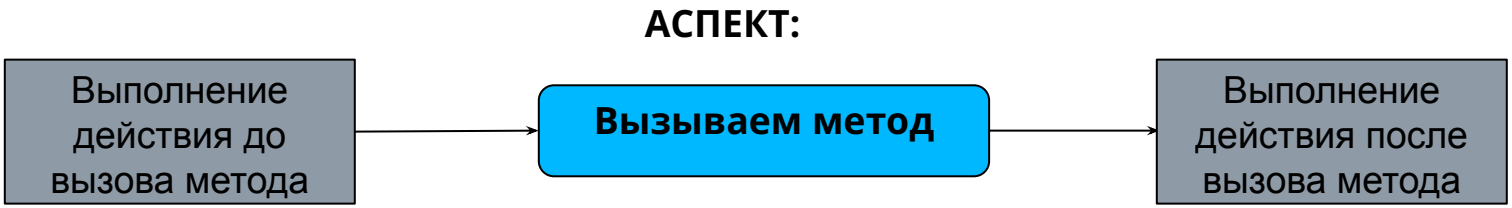
- **Виды ортогональной функциональности**
  - Логгирование
  - Обработка исключений
  - Транзакции
  - Кэширование
  - Проверка прав пользователя
  - и многое другое…

- **Минусы сервисного кода в основном коде:**
  - Растет объем кода
  - Сложнее поддерживать
  - Дублирование кода

- **Решение:** использовать аспекты

  ⇒ Вынести ортогональную функциональность в отдельные классы – аспекты

Как работают аспекты



Добавляем логгирование:

```
public UserDTO getUser(Integer id) {
   log.debug("Call method getUser with id " + id);
   UserDTO user = userDAO.getUser(id);
   log.debug("User info is: " + user.toString());
   return user;
}
```

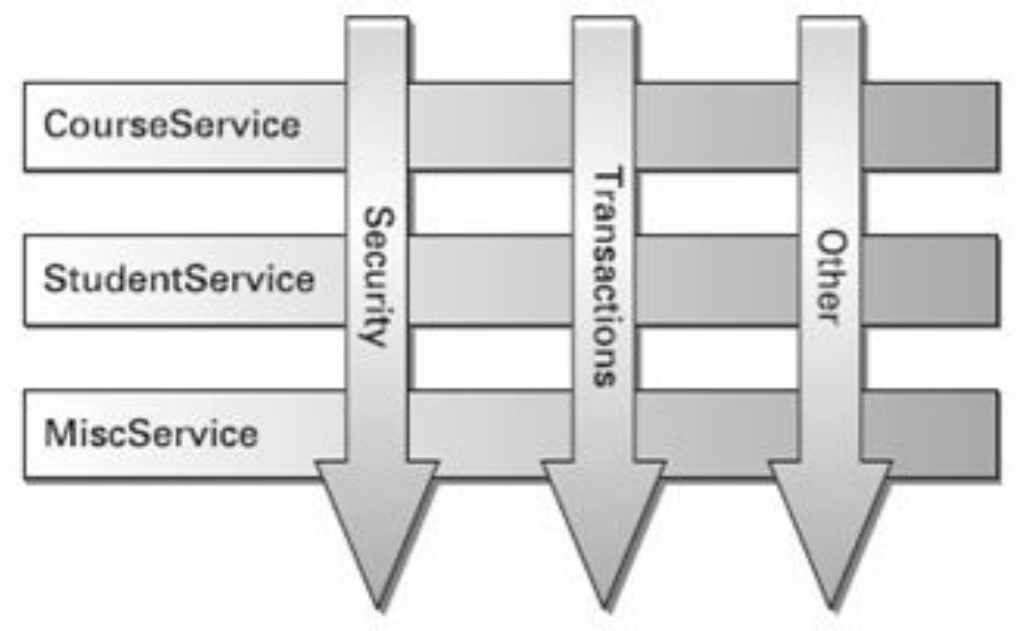
→ @Before advice

→ @After advice

Логгирование

AOP :: Введение

- Aspect Oriented Programming (AOP) – аспектно-ориентированное программирование
- АОП предоставляет средства для реализации ортогональной (crosscutting) функциональности



AOP :: Введение

- Как «ортогональную» бизнес-логику можно реализовать в СУРБД?

AOP :: Введение

- Пример ортогонального логгирования с использованием триггеров СУРБД:

```
/* Триггеры на уровне таблицы */
CREATE OR REPLACE TRIGGER DistrictUpdatedTrigger
 AFTER UPDATE ON district
 BEGIN
  INSERT INTO info VALUES ('table "district" has changed');
 END;
```

AOP :: Пример адвайса логгирования

```
@Aspect
public class LoggingAspect {
    private Logger logger = Logger.getLogger(LoggingAspect.class.getName());

   @Around("execution(* *.*User(..))")
   public Object log (ProceedingJoinPoint thisJoinPoint) throws Throwable {
           String methodName = thisJoinPoint.getSignature().getName();
           Object[] methodArgs = thisJoinPoint.getArgs();
        logger.info("Call method " + methodName + " with args " + methodArgs);
           Object result = thisJoinPoint.proceed();
        logger.info("Method " + methodName + " returns " + result);
        return result;
   }

}
```

AOP :: Пример адвайса логгирования

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:aop="http://www.springframework.org/schema/aop"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

  <aop:aspectj-autoproxy/>

  <bean id="userDao" class="UserDaoImpl"/>
  <bean id="loggingAspect" class = "LoggingAspect"/>
</beans>
```

AOP :: Пример адвайса логгирования

```
public interface UserDao {

  UserDTO getUser(int id);
}
```

```
public class UserDaoImpl implements UserDao {

   public UserDTO getUser(int id) {
       if (null != userDaoMap.get(id)) {
           return userDaoMap.get(id);
       }

       UserDTO user = new UserDTO(id);
       userDaoMap.put(id, user);
       return user;
   }

}
```

С помощью аспектов можно автоматически добавлять:

- Логгирование
- Обработку исключений
- Транзакции
- Кэширование
- Проверку прав пользователя
- и многое другое...

ex.1

AOP :: Пример

```
@Aspect
public class LoggingAspect {
  @Pointcut("execution(* *.*User(..))")
  public void userMethod() { }

  @Around("userMethod() ")
  public Object log (
  ProceedingJoinPoint thisJoinPoint) {
    String methodName =
     thisJoinPoint.getSignature().getName();
    Object[] methodArgs =
     thisJoinPoint.getArgs();
    logger.debug("Call method " + methodName
         + " with args " + methodArgs);

    Object result = thisJoinPoint.proceed();
    logger.debug("Method " + methodName
         + " returns " + result);
    return result;
  }
```

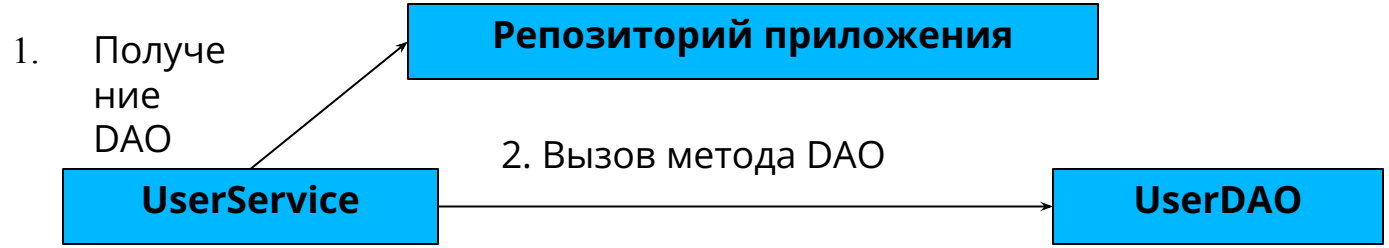
```
class UserDaoProxy implements UserDao {

 public UserDTO getUser(final Integer id)
 {
     Aspect logger = new LoggingAspect();
   ProceedingJoinPoint joinpoint =
         new ProceedingJoinPoint() {
      Object proceed() {
         return userDao.getUser(id);
      }
      };
   return logger.log(joinpoint);
  }
}
```

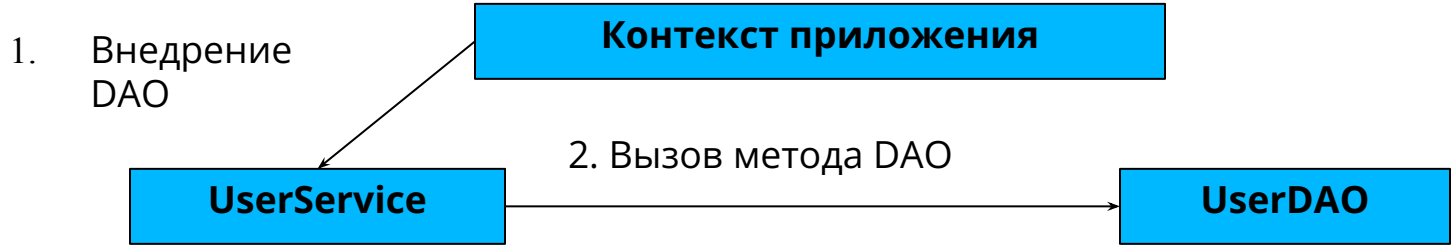
```
class UserDaoImpl implements UserDao {
 public UserDTO getUser(Integer id) {
     return userDAO.getUser(id);
 }
}
```

AOP :: Введение

Работа с DAO без IoC и AOP

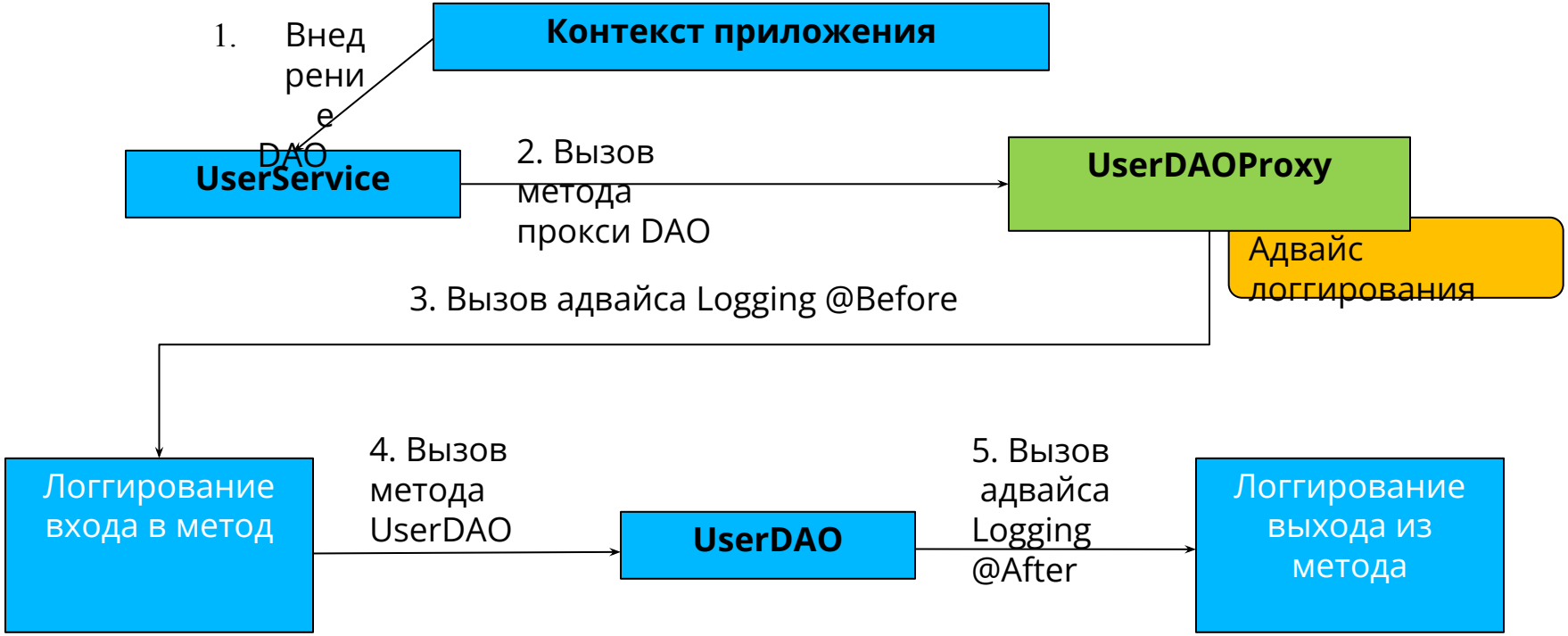


Работа с DAO с IoC, но без AOP



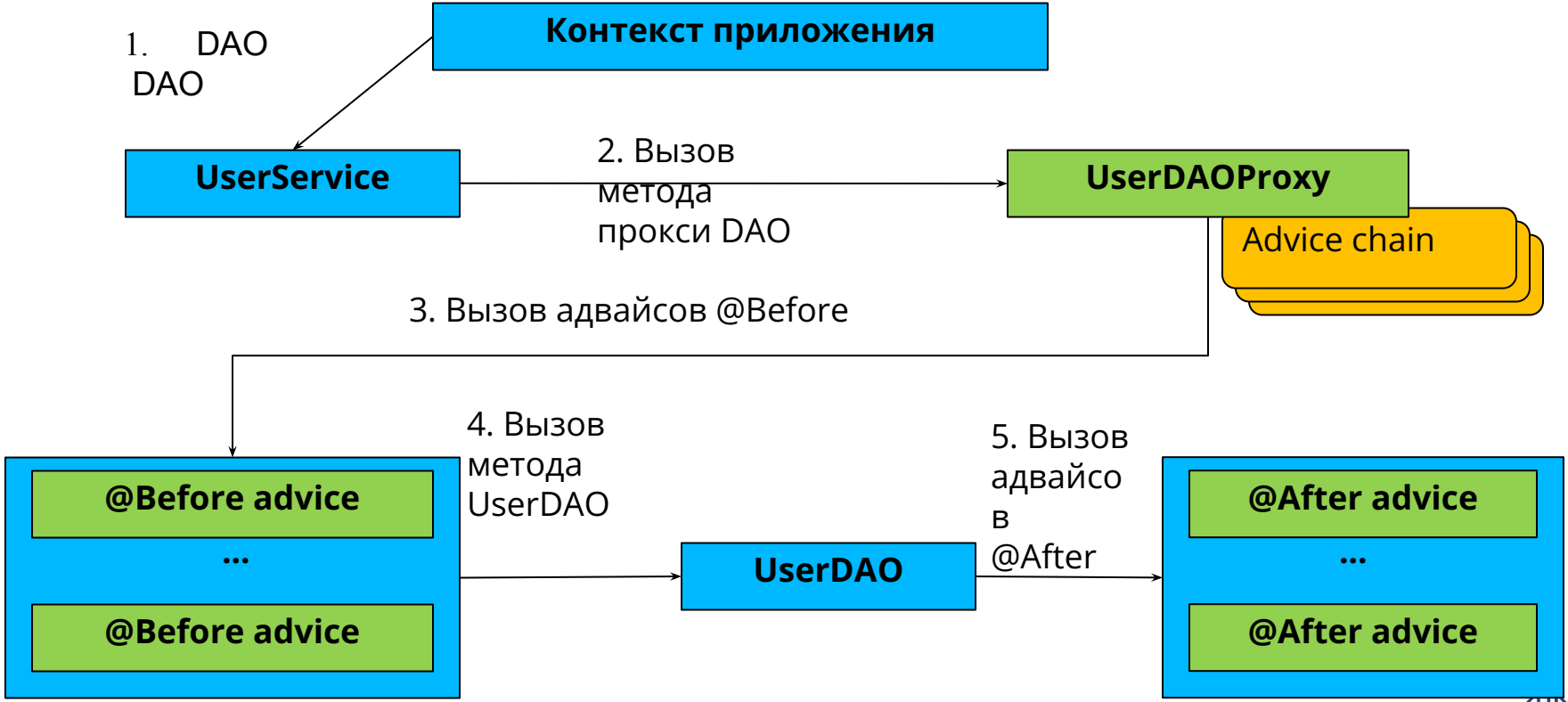
AOP :: Введение

Работа с DAO с IoC и AOP



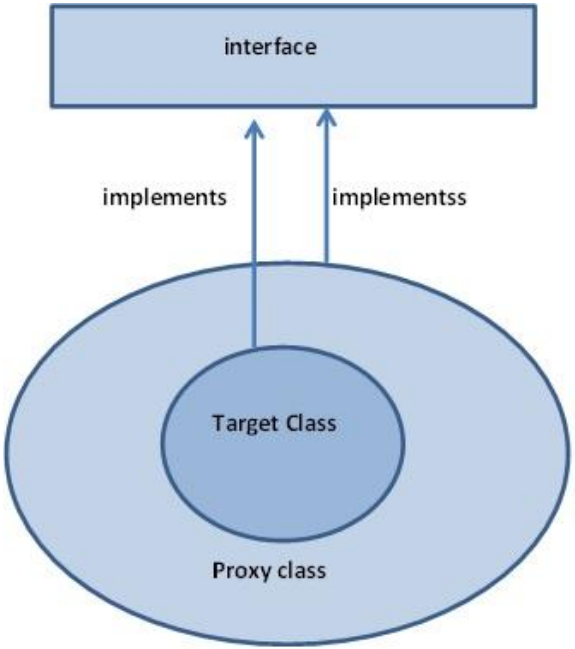
AOP :: Введение

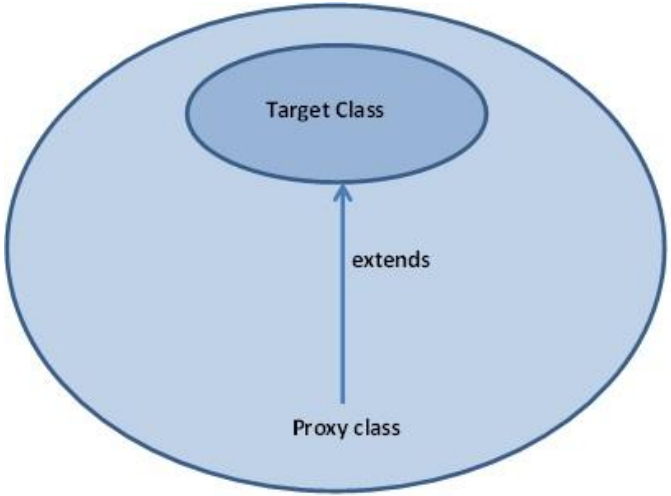
Работа с DAO с IoC и AOP



AOP :: Введение

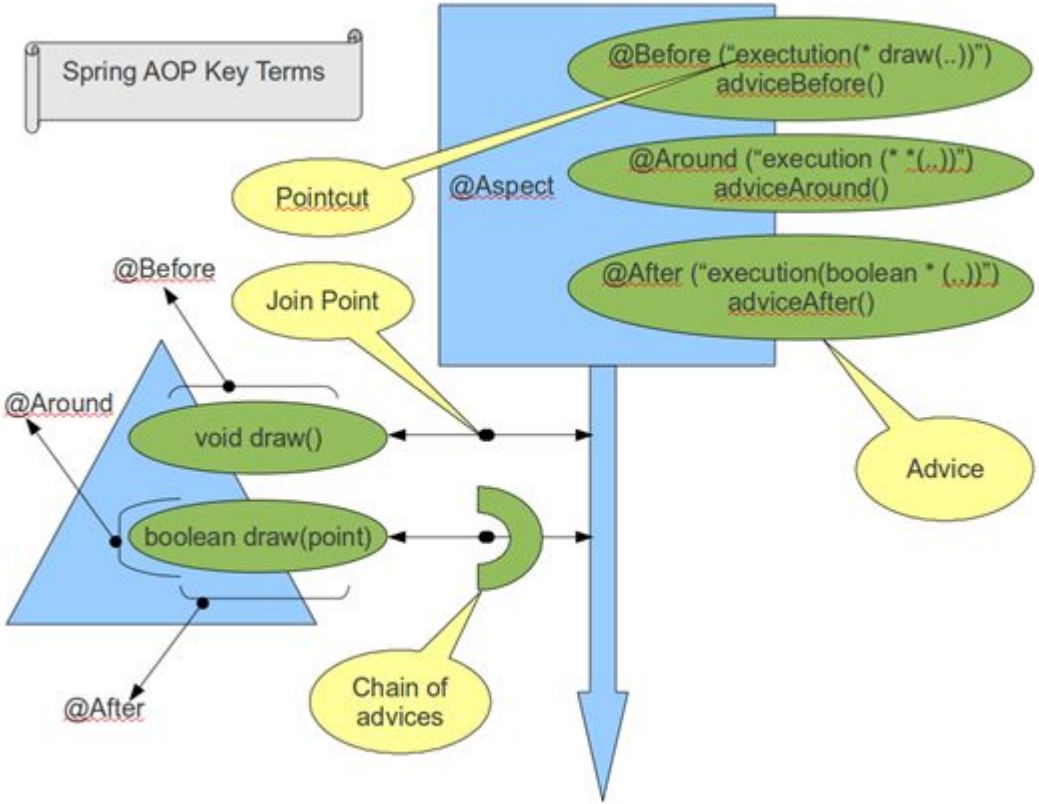
В Spring Framework AOP реализуется с помощью создания прокси-объекта на интересующий вас сервис.





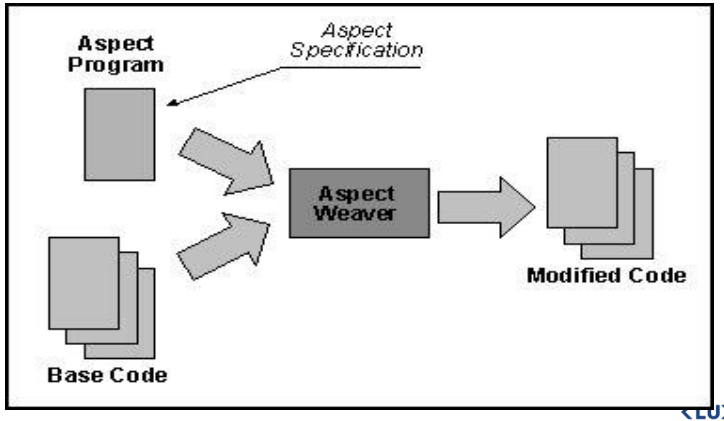
Стандартный механизм создания CGLIB прокси из JSE (динамические прокси JDK)

AOP :: Основные понятия



Активация AOP

- Weaving (связывание) – процесс применения аспекта к целевому объекту для создания нового прокси-объекта.
- Для осуществления связывания использует две дополнительные зависимости:
  - aspectjrt.jar
  - aspectjweaver.jar
- Также необходимо инициировать создание динамических прокси в файле конфигурации: `<aop:aspectj-autoproxy />`



AOP :: Язык срезов (pointcut)

execution – определяет срез на основе сигнатуры метода

execution(@CustomAnnotation? modifiers-pattern? **ret-type-pattern**

declaring-type-pattern?.**name-pattern(param-pattern)** throws-pattern?)

? – дополнительный параметр

declaring-type-pattern – шаблон для метода и имени класса

Примеры:

- **execution (* *(..))** – связывание с любым методом с любой сигнатурой;
- **execution (int *(..))** – связывание с любым методом, возвращающим int;
- **execution(* com.package.subpackage.Classname.*(..))** – связывание с любым методом com.package.subpackage.Classname class;

AOP :: Язык срезов (pointcut)

- **execution (void Test.foo(int, String))** – связывание с методом foo, класса Test, принимающим в качестве параметров int и String;
- **execution (* foo.bar.*.dao.*.update*(..))** – связывание с любым методом, начинающимся на «update», в пакете, начинающимся с foo.bar и заканчивающимся на dao;
- **bean** – связывание с точками соединения определенного Spring бина (или набора бинов)
  - **bean(“*Bean”)** – определяет точки соединения для всех бинов с идентификатором, заканчивающимся на Bean
- **within** – связывание с любым методом соответствующего класса
  - **within(com.package.subpackage.*)** – определяет любые точки сединения (выполнение метода только в Spring AOP) в рамках пакета com.package.subpackage
- **this** – связывание с точками соединения (выполнение методов при использовании Spring AOP) в случае, если ссылка на бин (Spring AOP Proxy) является объектом заданного типа

AOP :: Срез

- **this(com.package.InterfaceName)** – определяет точки соединения для всех методов в классах, реализующих интерфейс com.package.InterfaceName
- **target** – связывание с точками соединения (выполнение методов при использовании Spring AOP), когда целевой объект (т.е. объект приложения, который обернут прокси) является экземпляром заданного типа
  - **target(com.package.InterfaceName)** – определяет все методы объекта, целевой объект которого реализует com.package.InterfaceName
- **args** – связывание с точками соединения в случае, когда аргументами являются экземпляры заданных типов
  - **args(String)** – определяет методы, у которых определен один строковый аргумент
- **@annotation** - связывание с точками соединения в случае, когда метод точки соединения (выполнение метода при использовании Spring AOP) имеет данную аннотацию
  - **@annotation(com.package.annotation.Annotation)** – все методы, помеченные аннотацией @Annotation
  - **@annotation(org.springframework.stereotype.Repository)** – все методы, в

AOP :: типы адвайсов

- @Around advice – выполняется перед и после joinpoint
- Самый мощный из всех адвайсов

```
@Around("@annotation(com.luxoft.springaop.example2.Log)")
public Object log (ProceedingJoinPoint thisJoinPoint) throws Throwable {
        String methodName = thisJoinPoint.getSignature().getName();
        Object[] methodArgs = thisJoinPoint.getArgs();
     logger.info("Call method " + methodName + " with args " +
             methodArgs);
        Object result = thisJoinPoint.proceed();
     logger.info("Method " + methodName + " returns " + result);
     return result;
}
```

Примеры использования AOP

- Логгирование
- Проверки безопасности
- Управление транзакциями
- Обработка исключений
- Проверка прав пользователя
- Профилирование

AOP :: группировка аспектов

```
@Aspect
public class SystemArchitecture {

    @Pointcut("within(com.xyz.someapp.web..*)")
    public void inWebLayer() {}

    @Pointcut("within(com.xyz.someapp.service..*)")
    public void inServiceLayer() {}

    @Pointcut("within(com.xyz.someapp.dao..*)")
    public void inDataAccessLayer() {}

  @Pointcut("execution(* com.xyz.someapp.dao.*.*(..))")
  public void dataAccessOperation() {}

}
```

AOP :: Комбинирование срезов

Комбинирование pointcut выражений:

```
@Pointcut("execution(public * *(..))")
private void anyPublicOperation() {}

@Pointcut("within(com.xyz.someapp.trading..*")
private void inTrading() {}

@Pointcut("anyPublicOperation() && inTrading()")
private void tradingOperation() {}
```

AOP :: Типы адвайсов

- Может решать, исполнять ли joinpoint или вернуть собственное значение:

```
@Around("com.luxoft.example.SystemArchitecture.businessService()")
public Object accessRightsCheck(ProceedingJoinPoint pjp) throws Throwable
{

   if (currentUser.hasRights()) {
      return pjp.proceed();
   } else {
      throw new AuthorizationException();
   }

   return null;

}
```

AOP :: Использование @AfterThrowing

```
@Aspect
public class AfterThrowingExample {

    @AfterThrowing(
     pointcut="com.luxoft.example.SystemArchitecture.dataAccessOperation()",
        throwing="ex")
   public void doRecoveryActions(DataAccessException ex) {
   // ...
   }
}
```

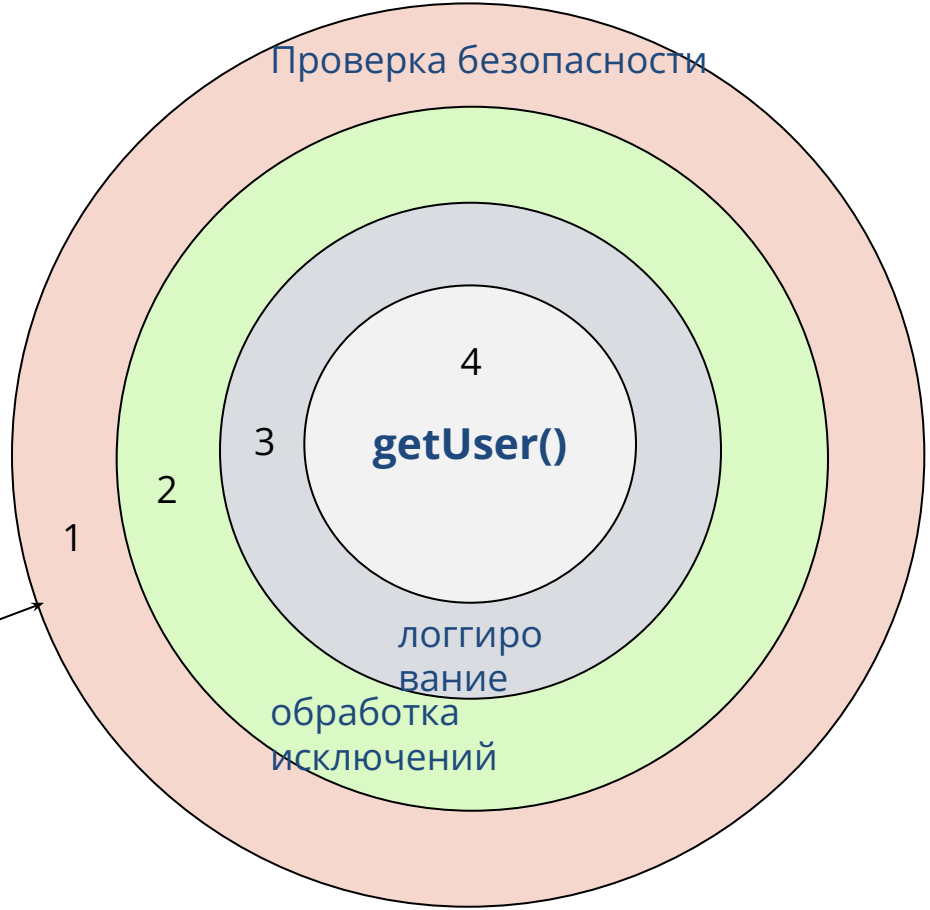
- Нет возможности вернуться к вызываемому методу или продолжить обработку на следующей строке
- Если здесь обрабатывается исключение, это не помешает ему «всплыть» в цепи

AOP :: Обзор типов адвайсов

@Before – выполняется перед joinpoint
Вызов joinpoint можно отменить, только выдав исключение
@Around – выполняется перед и после joinpoint

@AfterReturning – после успешного выполнения joinpoint, например, когда метод выполнился, не выдав исключение
@AfterThrowing – в случае выдачи исключения в joinpoint
@After – в любом случае после выполнения joinpoint

AOP :: Выстраивание цепочки аспектов



Матрешка

AOP :: @Order

Порядок выполнения аспектов можно задать с помощью аннотации `@Order`:

```
@Aspect
@Order(1)
public class AspectA
{
  @Before(".............")
  public void doIt() {}
}

@Aspect
@Order(2)
public class AspectB
{
  @Before(".............")
  public void doIt() {}
}
```

Упражнение

Практическое руководство

- Упражнение 4