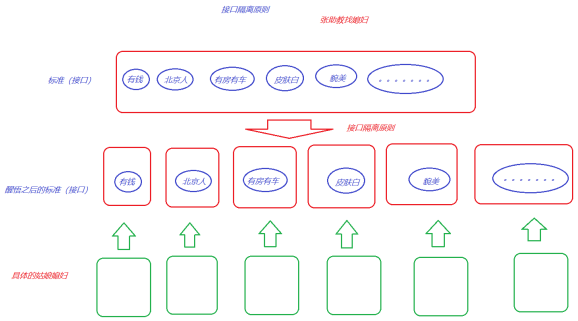
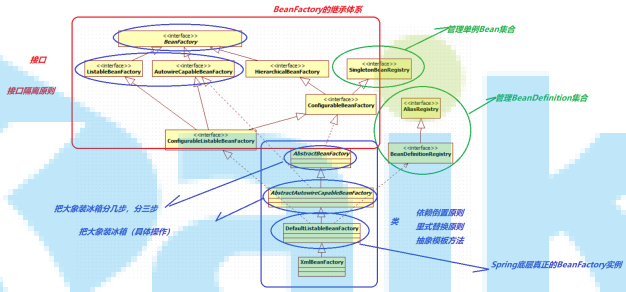
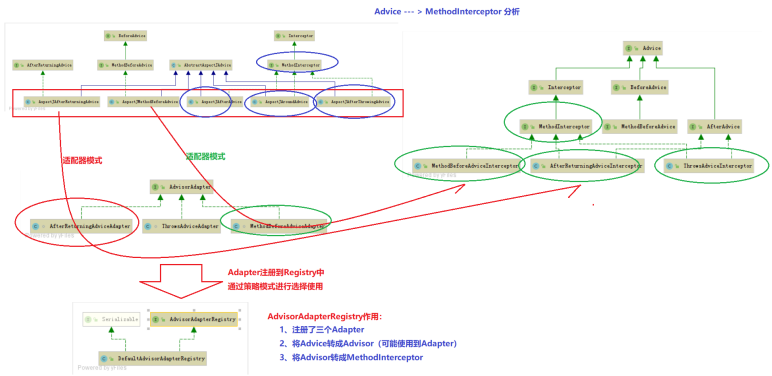
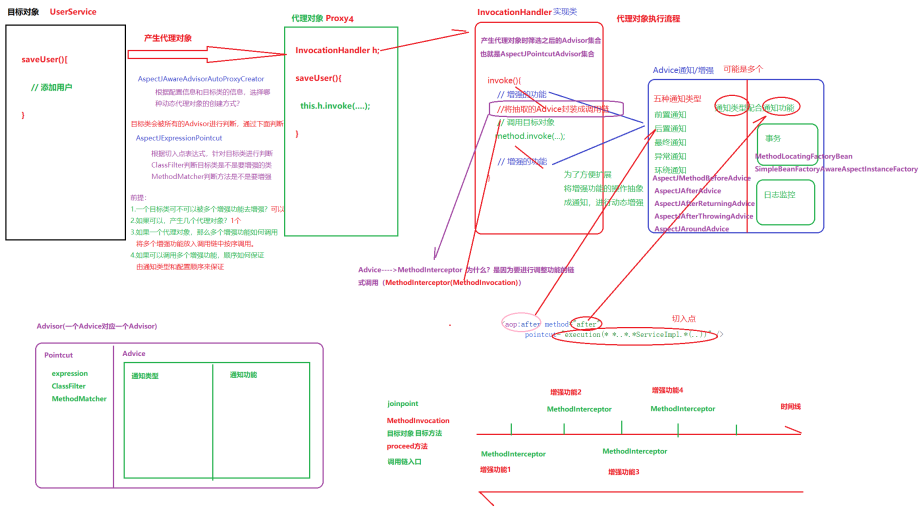
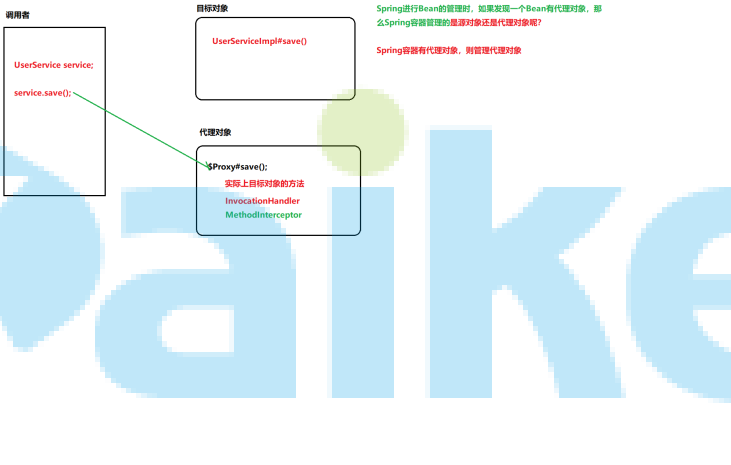


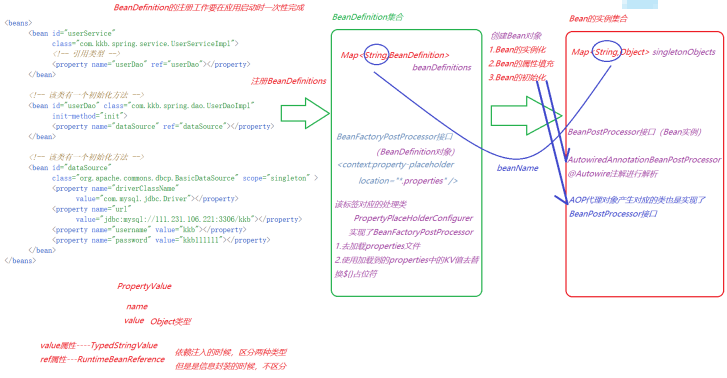
结论：即使这样，助教还单身（思考题）

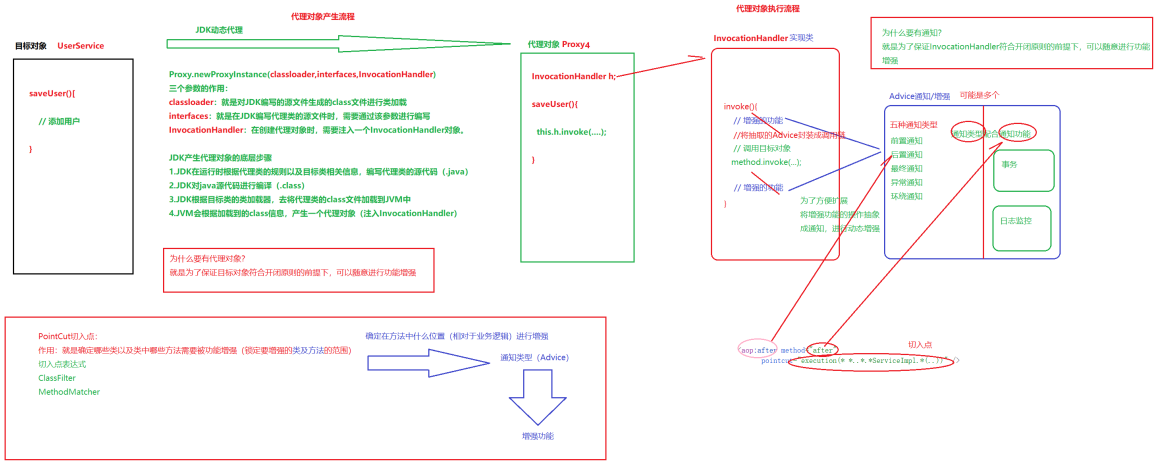





目标对象 UserService
saveUser(){
// 添加用户
}
产生代理对象
代理对象 Proxy4
InvocationHandler h;
saveUser(){
this.h.invoke(....);
}
AspectJAwareAdvisorAutoProxyCreator
根据配置信息和目标类的信息，选择哪种动态代理对象的创建方式?
目标类会被所有的Advisor进行判断，通过下面判断
AspectJExpressionPointcut
根据切入点表达式，针对目标类进行判断
ClassFilter判断目标类是不是要增强的类
MethodMatcher判断方法是不是要增强
前提：
1.一个目标类可不可以被多个增强功能去增强？可以
2.如果可以，产生几个代理对象？1个
3.如果一个代理对象，那么多个增强功能如何调用
将多个增强功能放入调用链中按序调用。
4.如果可以调用多个增强功能，顺序如何保证
由通知类型和配置顺序来保证
InvocationHandler实现类
代理对象执行流程
产生代理对象时筛选之后的Advisor集合
也就是AspectJPointcutAdvisor集合
invoke(){
// 增强前功能
//将抽取的Advice封装成调用链
// 调用目标对象
method.invoke(...);
// 增强后功能
}
为了方便扩展
将增强功能的动作抽象
成通知，进行动态增强
Advice通知/增强 可能是多个
五种通知类型
前置通知
后置通知
最终通知
异常通知
环绕通知
AspectJMethodBeforeAdvice
AspectJAfterAdvice
AspectJAfterReturningAdvice
AspectJAfterThrowingAdvice
AspectJAroundAdvice
通知类型决定增强功能
事务
MethodLocatingFactoryBean
SimpleBeanFactoryAwareAspectInstanceFactory
日志监控
Advice---->MethodInterceptor 为什么? 是因为要进行调整功能的链式调用（MethodInterceptor(MethodInvocation)）
aop:after method="after"
pointcut="execution(* *..*.*ServiceImpl.*(..))"
切入点
Advisor(一个Advice对应一个Advisor)
Pointcut
expression
ClassFilter
MethodMatcher
Advice
通知类型
通知功能
joinpoint
MethodInvocation
目标对象 目标方法
proceed方法
调用链入口
增强功能2
MethodInterceptor
增强功能4
MethodInterceptor
时间线
MethodInterceptor
增强功能1
MethodInterceptor
增强功能3


调用者
UserService service;
service.save();
目标对象
UserServiceImpl#save()
代理对象
$Proxy#save();
实际上目标对象的方法
InvocationHandler
MethodInterceptor
Spring进行Bean的管理时，如果发现一个Bean有代理对象，那么Spring容器管理的是源对象还是代理对象呢?
Spring容器有代理对象，则管理代理对象


BeanDefinition的注册工作要在应用启动时一次性完成
<beans>
<bean id="userService"
class="com.kkb.spring.service.UserServiceImpl">
<!-- 引用类型 -->
<property name="userDao" ref="userDao"></property>
</bean>
注册BeanDefinitions
<!-- 该类有一个初始化方法 -->
<bean id="userDao" class="com.kkb.spring.dao.UserDaoImpl"
init-method="init">
<property name="dataSource" ref="dataSource"></property>
</bean>
<!-- 该类有一个初始化方法 -->
<bean id="dataSource"
class="org.apache.commons.dbcp.BasicDataSource" scope="singleton" >
<property name="driverClassName"
value="com.mysql.jdbc.Driver"></property>
<property name="url"
value="jdbc:mysql://111.231.106.221:3306/kkb"></property>
<property name="username" value="kkb"></property>
<property name="password" value="kkb111111"></property>
</bean>
</beans>
PropertyValue
name
value Object类型
value属性----TypedStringValue
ref属性----RuntimeBeanReference
依赖注入的时候，区分两种类型
但是最后也是封装的时候，不区分
BeanDefinition集合
Map<String,BeanDefinition>
beanDefinitions
BeanFactoryPostProcessor接口
(BeanDefinition对象)
<context:property-placeholder
location="*.properties"/>
该标签对应的处理类
PropertyPlaceHolderConfigurer
实现了BeanFactoryPostProcessor
1.去加载properties文件
2.使用加载到的properties中的KV值去替换${}占位符
创建Bean对象
1.Bean的实例化
2.Bean的属性填充
3.Bean的初始化
beanName
Bean的实例集合
Map<String,Object> singletonObjects
BeanPostProcessor接口（Bean实例）
AutowiredAnnotationBeanPostProcessor
@Autowire注解进行解析
AOP代理对象产生对应的类也是实现了
BeanPostProcessor接口



业务逻辑：和功能性需求（CRUD）业务有关

系统逻辑：和非功能性需求业务（安全、事务、数据一致性、并发支持能力、RT）有关

一般来说，业务逻辑和系统逻辑应该是两类人去进行开发和维护。

```
class UserService{
  void saveUser(){
    // 开启事务
    业务代码编写
    // 关闭事务
  }
}
```

系统逻辑

业务逻辑

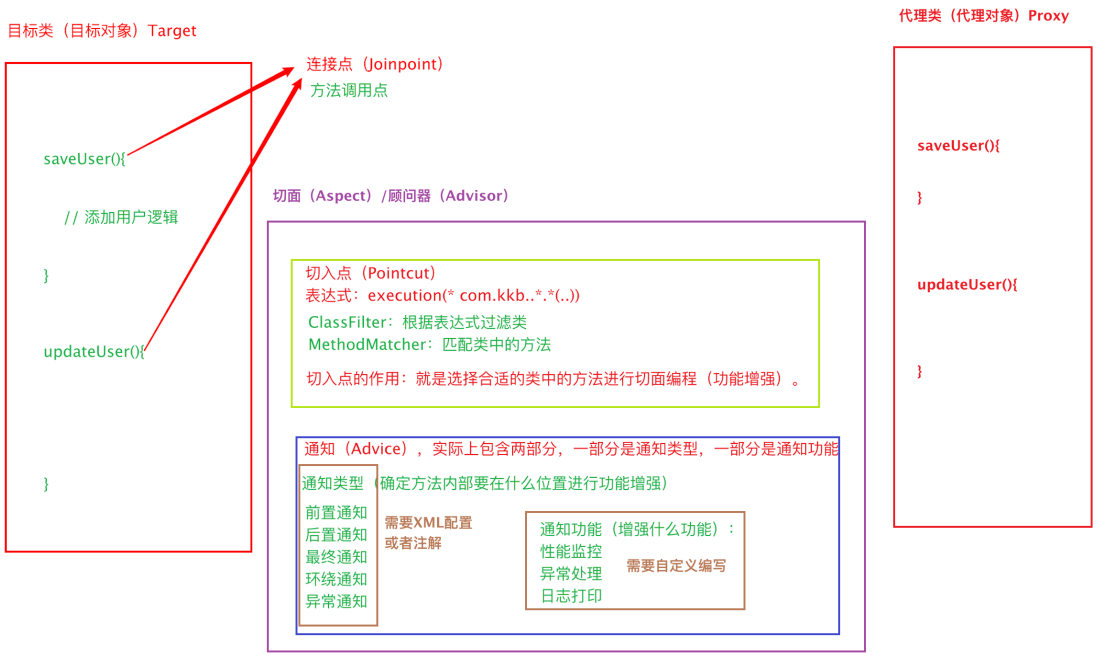
扩展点：微服务框架就需要程序员既要对服务实现进行掌握
还要对服务治理精通。

服务网格架构　　服务实现和服务治理分离

istio

AOP可以实现将系统逻辑和业务逻辑分离

AOP可以实现将业务逻辑和系统逻辑有两人进行隔离编写，最终可以无感知的去进行整合。



JDK可以按照固定逻辑写出来代理类的源代码（java），然后进行编译，及类加载

```
Proxy4 implement 接口{
  InvocationHandler h;
  // 构造方法注入InvocationHandler
  saveUser(){
    this.h.invoke(this,方法对象,args);
  }
}
```

源码阅读：入口1—解析xml中的aop标签

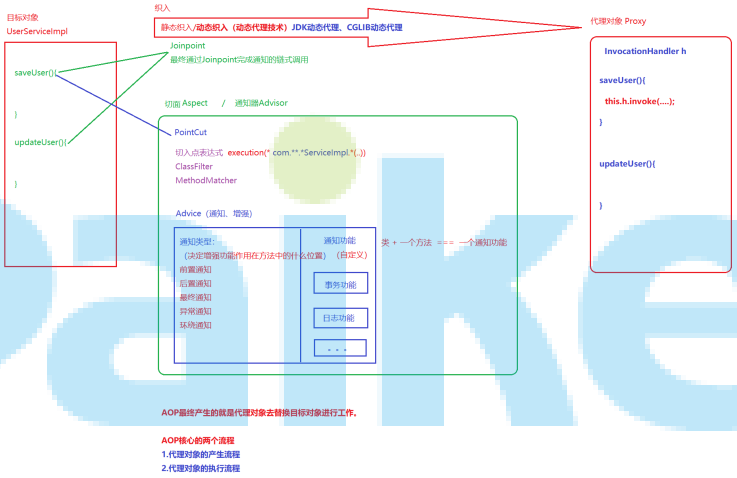
如何执行代理对象

```
// MethodLocatingFactoryBean
// SimpleBeanFactoryAwareAspectInstanceFactory

// AspectJPointcutAdvisor
// AspectJMethodBeforeAdvice
// AspectJAfterAdvice
// AspectJAfterReturningAdvice
// AspectJAfterThrowingAdvice
// AspectJAroundAdvice
// DefaultBeanFactoryPointcutAdvisor
// AspectJExpressionPointcut
```

AspectJAwareAdvisorAutoProxyCreator：产生代理对象的类

AOP核心概念：
连接点（Joinpoint）
切入点(Pointcut)
目标对象(Target)
代理对象(Proxy)
织入(Weave)
通知（增强）(Advice)
切面(Aspect)
通知器/顾问器(Advisor)



AOP最终产生的就是代理对象去替换目标对象进行工作。

AOP核心的两个流程
1.代理对象的产生流程
2.代理对象的执行流程