# AbstractApplicationContext.refresh()

```
public void refresh() throws BeansException, IllegalStateException {
    synchronized (this.startupShutdownMonitor) {
        // Prepare this context for refreshing.
        prepareRefresh();

        // Tell the subclass to refresh the internal bean factory.
        ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();

        // Prepare the bean factory for use in this context.
        prepareBeanFactory(beanFactory);

        try {
            // Allows post-processing of the bean factory in context subclasses.
            postProcessBeanFactory(beanFactory);

            // Invoke factory processors registered as beans in the context.
            invokeBeanFactoryPostProcessors(beanFactory);
```

AbstractApplicationContext.invokeBeanFactoryPostProcessors

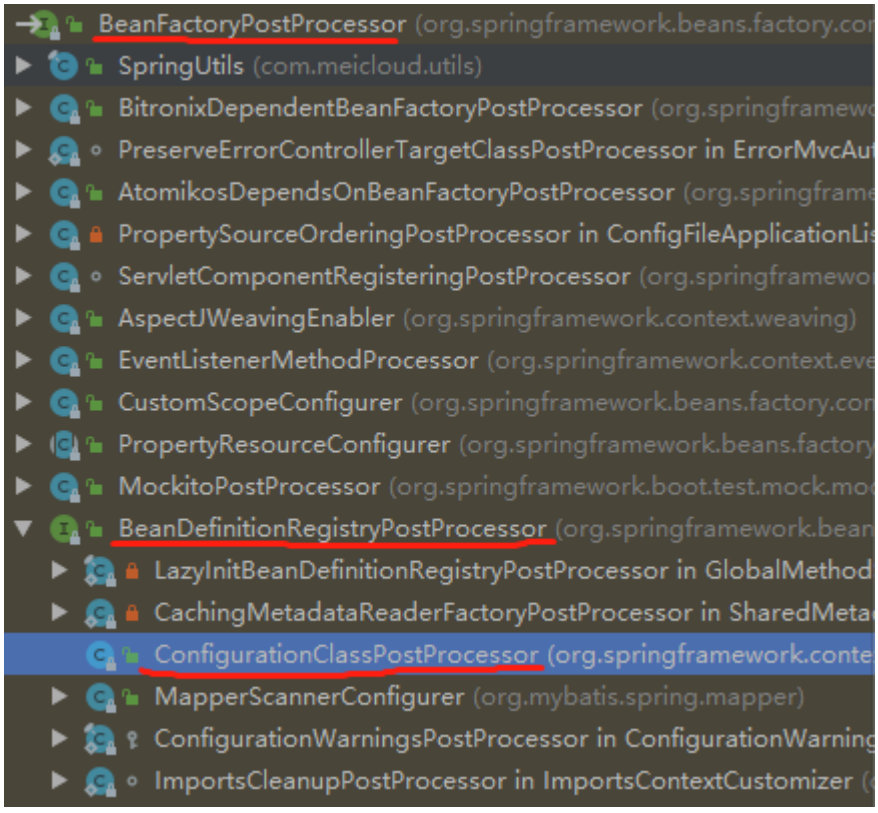
()

```
protected void invokeBeanFactoryPostProcessors(ConfigurableListableBeanFactory beanFactory) {
    PostProcessorRegistrationDelegate.invokeBeanFactoryPostProcessors(beanFactory, getBeanFactoryPostProcessors());

    // Detect a LoadTimeWeaver and prepare for weaving, if found in the meantime
    // (e.g. through an @Bean method registered by ConfigurationClassPostProcessor)
    if (beanFactory.getTempClassLoader() == null && beanFactory.containsBean(LOAD_TIME_WEAVER_BEAN_NAME)) {
        beanFactory.addBeanPostProcessor(new LoadTimeWeaverAwareProcessor(beanFactory));
        beanFactory.setTempClassLoader(new ContextTypeMatchClassLoader(beanFactory.getBeanClassLoader()));
    }
}
```

# PostProcessorRegistrationDelegate.invokeBeanFactoryPostProcessors（）

```
public static void invokeBeanFactoryPostProcessors(
        ConfigurableListableBeanFactory beanFactory, List<BeanFactoryPostProcessor> beanFactoryPostProcessors) {

    // Invoke BeanDefinitionRegistryPostProcessors first, if any.
    Set<String> processedBeans = new HashSet<>();

    if (beanFactory instanceof BeanDefinitionRegistry) {
        BeanDefinitionRegistry registry = (BeanDefinitionRegistry) beanFactory;
        List<BeanFactoryPostProcessor> regularPostProcessors = new ArrayList<>();
        List<BeanDefinitionRegistryPostProcessor> registryProcessors = new ArrayList<>();
```

```
    for (BeanFactoryPostProcessor postProcessor : beanFactoryPostProcessors) {
        if (postProcessor instanceof BeanDefinitionRegistryPostProcessor) {
            BeanDefinitionRegistryPostProcessor registryProcessor =
                    (BeanDefinitionRegistryPostProcessor) postProcessor;
            registryProcessor.postProcessBeanDefinitionRegistry(registry);
            registryProcessors.add(registryProcessor);
        }
        else {
            regularPostProcessors.add(postProcessor);
        }
    }
```



BeanDefinitionRegistryPostProcessor.postProcessBeanDefinitionRegistry

```
public void postProcessBeanDefinitionRegistry(BeanDefinitionRegistry registry) {
    int registryId = System.identityHashCode(registry);
    if (this.registriesPostProcessed.contains(registryId)) {
        throw new IllegalStateException(
                "postProcessBeanDefinitionRegistry already called on this post-processor against " +
    }
    if (this.factoriesPostProcessed.contains(registryId)) {
        throw new IllegalStateException(
                "postProcessBeanFactory already called on this post-processor against " + registry);
    }
    this.registriesPostProcessed.add(registryId);

    processConfigBeanDefinitions(registry);
}
```

## BeanDefinitionRegistryPostProcessor.processConfigBeanDefinitions

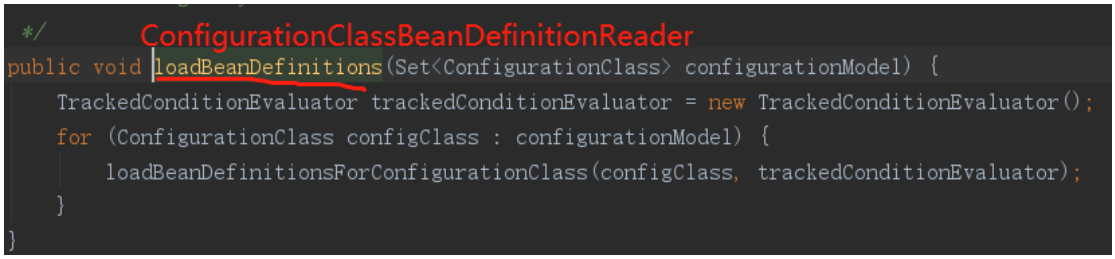
解释每一个@Configuration 的类

```
// Parse each @Configuration class
ConfigurationClassParser parser = new ConfigurationClassParser(
        this.metadataReaderFactory, this.problemReporter, this.environment,
        this.resourceLoader, this.componentScanBeanNameGenerator, registry);

Set<BeanDefinitionHolder> candidates = new LinkedHashSet<>(configCandidates);
Set<ConfigurationClass> alreadyParsed = new HashSet<>(configCandidates.size());
do {
    parser.parse(candidates);
    parser.validate();

    Set<ConfigurationClass> configClasses = new LinkedHashSet<>(parser.getConfigurationClasses());
    configClasses.removeAll(alreadyParsed);
```

```
// Read the model and create bean definitions based on its content
if (this.reader == null) {
    this.reader = new ConfigurationClassBeanDefinitionReader(
            registry, this.sourceExtractor, this.resourceLoader, this.environment,
            this.importBeanNameGenerator, parser.getImportRegistry());
}
this.reader.loadBeanDefinitions(configClasses);
alreadyParsed.addAll(configClasses);
```

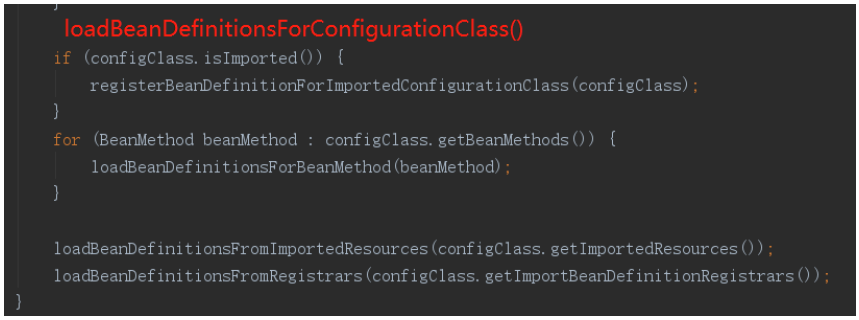
processConfigBeanDefinitions ()

ConfigurationClassBeanDefinitionReader.loadBeanDefinitions

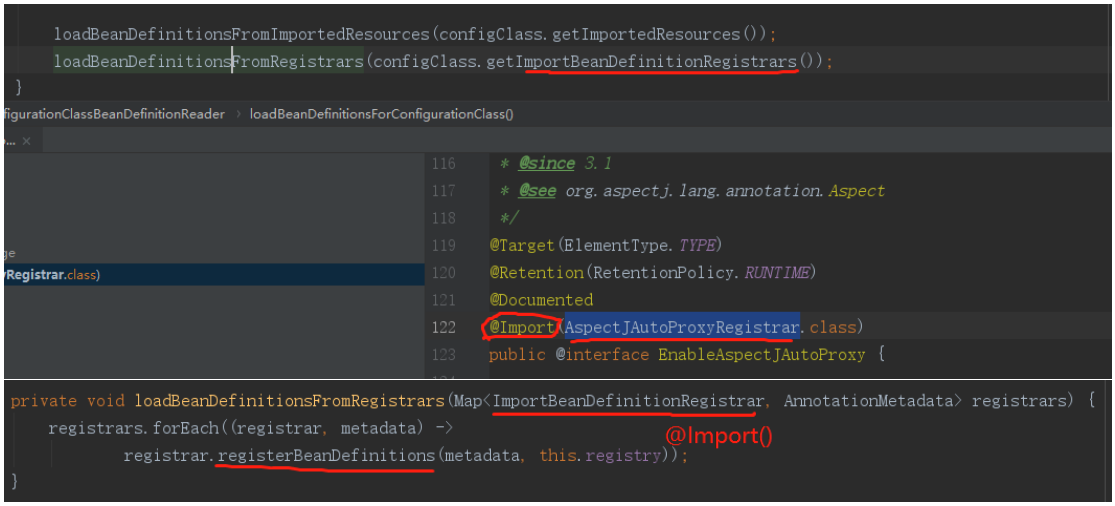



```
*/
public void loadBeanDefinitions(Set<ConfigurationClass> configurationModel) {
    TrackedConditionEvaluator trackedConditionEvaluator = new TrackedConditionEvaluator();
    for (ConfigurationClass configClass : configurationModel) {
        loadBeanDefinitionsForConfigurationClass(configClass, trackedConditionEvaluator);
    }
}
```

**ConfigurationClassBeanDefinitionReader.loadBeanDefinitionsForConfigurationClass**




```
    if (configClass.isImported()) {
        registerBeanDefinitionForImportedConfigurationClass(configClass);
    }
    for (BeanMethod beanMethod : configClass.getBeanMethods()) {
        loadBeanDefinitionsForBeanMethod(beanMethod);
    }

    loadBeanDefinitionsFromImportedResources(configClass.getImportedResources());
    loadBeanDefinitionsFromRegistrars(configClass.getImportBeanDefinitionRegistrars());
}
```



```
public void registerBeanDefinitions(
        AnnotationMetadata importingClassMetadata, BeanDefinitionRegistry registry) {

    AopConfigUtils.registerAspectJAnnotationAutoProxyCreatorIfNecessary(registry);

    AnnotationAttributes enableAspectJAutoProxy =
            AnnotationConfigUtils.attributesFor(importingClassMetadata, EnableAspectJAutoProxy.class);
    if (enableAspectJAutoProxy != null) {
        if (enableAspectJAutoProxy.getBoolean( attributeName: "proxyTargetClass")) {
            AopConfigUtils.forceAutoProxyCreatorToUseClassProxying(registry);
        }
        if (enableAspectJAutoProxy.getBoolean( attributeName: "exposeProxy")) {
            AopConfigUtils.forceAutoProxyCreatorToExposeProxy(registry);
        }
    }
}
```

AspectJAutoProxyRegistrar

```
public static BeanDefinition registerAspectJAnnotationAutoProxyCreatorIfNecessary(
        BeanDefinitionRegistry registry, @Nullable Object source) {

    return registerOrEscalateApcAsRequired(AnnotationAwareAspectJAutoProxyCreator.class, registry, source);
}
```

beanClass

AspectJAutoProxyRegistrar.registerBeanDefinitions()

ImportBeanDefinitionRegistrar

处理@Configuration 注解的类的附加 bean 定义的注册器需要实现该接口

```
@Override
public void registerBeanDefinitions(
        AnnotationMetadata importingClassMetadata, BeanDefinitionRegistry registry) {

    AopConfigUtils.registerAspectJAnnotationAutoProxyCreatorIfNecessary(registry);

    AnnotationAttributes enableAspectJAutoProxy =
            AnnotationConfigUtils.attributesFor(importingClassMetadata, EnableAspectJAutoProxy.class);
    if (enableAspectJAutoProxy != null) {
        if (enableAspectJAutoProxy.getBoolean( attributeName: "proxyTargetClass")) {
            AopConfigUtils.forceAutoProxyCreatorToUseClassProxying(registry);
        }
        if (enableAspectJAutoProxy.getBoolean( attributeName: "exposeProxy")) {
            AopConfigUtils.forceAutoProxyCreatorToExposeProxy(registry);
        }
    }
}
```

```
public static void forceAutoProxyCreatorToUseClassProxying(BeanDefinitionRegistry registry) {
    if (registry.containsBeanDefinition(AUTO_PROXY_CREATOR_BEAN_NAME)) {
        BeanDefinition definition = registry.getBeanDefinition(AUTO_PROXY_CREATOR_BEAN_NAME);
        definition.getPropertyValues().add( propertyName: "proxyTargetClass", Boolean.TRUE);
    }
}
```

org.springframework.aop.config.internalAutoProxyCreator

```java
private static BeanDefinition registerOrEscalateApcAsRequired(
        Class<?> cls, BeanDefinitionRegistry registry, @Nullable Object source) {

    Assert.notNull(registry, message: "BeanDefinitionRegistry must not be null");

    if (registry.containsBeanDefinition(AUTO_PROXY_CREATOR_BEAN_NAME)) {
        BeanDefinition apcDefinition = registry.getBeanDefinition(AUTO_PROXY_CREATOR_BEAN_NAME);
        if (!cls.getName().equals(apcDefinition.getBeanClassName())) {
            int currentPriority = findPriorityForClass(apcDefinition.getBeanClassName());
            int requiredPriority = findPriorityForClass(cls);
            if (currentPriority < requiredPriority) {
                apcDefinition.setBeanClassName(cls.getName());
            }
        }
        return null;
    }
                                    AnnotationAwareAspectJAutoProxyCreator.class
    RootBeanDefinition beanDefinition = new RootBeanDefinition(cls);
    beanDefinition.setSource(source);
    beanDefinition.getPropertyValues().add( propertyName: "order", Ordered.HIGHEST_PRECEDENCE);
    beanDefinition.setRole(BeanDefinition.ROLE_INFRASTRUCTURE);
    registry.registerBeanDefinition(AUTO_PROXY_CREATOR_BEAN_NAME, beanDefinition);
    return beanDefinition;
}
```

点评:

1、利用 beanFactory 的 BeanFactoryPostProcessor 的机制,外加一些 bean 的 register 操作;

2、在解释@Configuration 的类的过程中,读取@Import 注解的类,将 AnnotationAwareAspectJAutoProxyCreator.class 注册到 BeanFatory 中。

这是第一个与 IOC 结合的地方;

3、感知接口在 createBean 的时候注入被感知对象

## AnnotationAwareAspectJAutoProxyCreator

AbstractAutoProxyCreator-method.pdf

AbstractAutoProxyCreator-field.pdf

# postProcessBeforeInstantiation()

```java
@Override
public Object postProcessBeforeInstantiation(Class<?> beanClass, String beanName) {
    Object cacheKey = getCacheKey(beanClass, beanName);

    if (!StringUtils.hasLength(beanName) || !this.targetSourcedBeans.contains(beanName)) {
        if (this.advisedBeans.containsKey(cacheKey)) {
            return null;
        }
        if (isInfrastructureClass(beanClass) || shouldSkip(beanClass, beanName)) {
            this.advisedBeans.put(cacheKey, Boolean.FALSE);
            return null;
        }
    }

    // Create proxy here if we have a custom TargetSource.
    // Suppresses unnecessary default instantiation of the target bean:
    // The TargetSource will handle target instances in a custom fashion.
    TargetSource targetSource = getCustomTargetSource(beanClass, beanName);
    if (targetSource != null) {
        if (StringUtils.hasLength(beanName)) {
            this.targetSourcedBeans.add(beanName);
        }
        Object[] specificInterceptors = getAdvicesAndAdvisorsForBean(beanClass, beanName, targetSource);
        Object proxy = createProxy(beanClass, beanName, specificInterceptors, targetSource);
        this.proxyTypes.put(cacheKey, proxy.getClass());
        return proxy;
    }

    return null;
}
```

这个涉及 targetSource;

TargetSource 用于获取 AOP 调用的当前“target”，如果没有 around advice 选择终止拦截器链本身，则将通过反射调用该调用。

如果 TargetSource 是“静态”的，它将始终返回相同的目标，从而允许在 AOP 框架中进行优化。动态目标源可以支持池、热交换等。

## postProcessAfterInitialization()

```
/**
 * Create a proxy with the configured interceptors if the bean is
 * identified as one to proxy by the subclass.
 * @see #getAdvicesAndAdvisorsForBean
 */
@Override
public Object postProcessAfterInitialization(@Nullable Object bean, String beanName) {
    if (bean != null) {
        Object cacheKey = getCacheKey(bean.getClass(), beanName);
        if (!this.earlyProxyReferences.contains(cacheKey)) {
            return wrapIfNecessary(bean, beanName, cacheKey);
        }
    }
    return bean;
}
```

wrapIfNecessary()

```
protected Object wrapIfNecessary(Object bean, String beanName, Object cacheKey) {
    if (StringUtils.hasLength(beanName) && this.targetSourcedBeans.contains(beanName)) {
        return bean;
    }
    if (Boolean.FALSE.equals(this.advisedBeans.get(cacheKey))) {
        return bean;
    }
    if (isInfrastructureClass(bean.getClass()) || shouldSkip(bean.getClass(), beanName)) {
        this.advisedBeans.put(cacheKey, Boolean.FALSE);
        return bean;
    }

    // Create proxy if we have advice. 匹配这个bean的拦截器
    Object[] specificInterceptors = getAdvicesAndAdvisorsForBean(bean.getClass(), beanName, customTargetSource: null);
    if (specificInterceptors != DO_NOT_PROXY) {
        this.advisedBeans.put(cacheKey, Boolean.TRUE);
        Object proxy = createProxy(
                bean.getClass(), beanName, specificInterceptors, new SingletonTargetSource(bean));
        this.proxyTypes.put(cacheKey, proxy.getClass());
        return proxy;
    }

    this.advisedBeans.put(cacheKey, Boolean.FALSE);
    return bean;
}
```

# 1\getAdvicesAndAdvisorsForBean(class, beanName, targetSource)

getAdvicesAndAdvisorsForBean

```java
@Override
@Nullable
protected Object[] getAdvicesAndAdvisorsForBean(
        Class<?> beanClass, String beanName, @Nullable TargetSource targetSource) {

    List<Advisor> advisors = findEligibleAdvisors(beanClass, beanName);
    if (advisors.isEmpty()) {
        return DO_NOT_PROXY;
    }
    return advisors.toArray();
}
```

```java
protected List<Advisor> findEligibleAdvisors(Class<?> beanClass, String beanName) {
    List<Advisor> candidateAdvisors = findCandidateAdvisors();
    List<Advisor> eligibleAdvisors = findAdvisorsThatCanApply(candidateAdvisors, beanClass, beanName);
    extendAdvisors(eligibleAdvisors);
    if (!eligibleAdvisors.isEmpty()) {
        eligibleAdvisors = sortAdvisors(eligibleAdvisors);
    }
    return eligibleAdvisors;
}
```

## 1.1 findCandiateAdvisors()

```java
@Override   AspectJAwareAdvisorAutoProxyCreator
protected List<Advisor> findCandidateAdvisors() {
    // Add all the Spring advisors found according to superclass rules.
    List<Advisor> advisors = super.findCandidateAdvisors();
    // Build Advisors for all AspectJ aspects in the bean factory.
    if (this.aspectJAdvisorsBuilder != null) {
        advisors.addAll(this.aspectJAdvisorsBuilder.buildAspectJAdvisors());
    }
    return advisors;
}
```

子类重写

## 1.1.1AspectJAdvisorBuilder

```
public List<Advisor> buildAspectJAdvisors() {
    List<String> aspectNames = this.aspectBeanNames;

    if (aspectNames == null) {
        synchronized (this) {
            aspectNames = this.aspectBeanNames;
            if (aspectNames == null) {
                List<Advisor> advisors = new ArrayList<>();
                aspectNames = new ArrayList<>();
                String[] beanNames = BeanFactoryUtils.beanNamesForTypeIncludingAncestors(
                        this.beanFactory, Object.class, includeNonSingletons: true, allowEagerInit: false);
                for (String beanName : beanNames) {
                    if (!isEligibleBean(beanName)) {
                        continue;
                    }
                    // We must be careful not to instantiate beans eagerly as in this case they
                    // would be cached by the Spring container but would not have been weaved.
                    Class<?> beanType = this.beanFactory.getType(beanName);
                    if (beanType == null) {
                        continue;
                    }
```

BeanFactoryAspectJAdvisorsBuilder

获取beanFactory的所有bean

点评：isAspect(beanClass):判断 bean 是否是 Aspect

```
private boolean hasAspectAnnotation(Class<?> clazz) {
    return (AnnotationUtils.findAnnotation(clazz, Aspect.class) != null);
}
```

```
        if (this.advisorFactory.isAspect(beanType)) {
            aspectNames.add(beanName);
            AspectMetadata amd = new AspectMetadata(beanType, beanName);
            if (amd.getAjType().getPerClause().getKind() == PerClauseKind.SINGLETON) {
                MetadataAwareAspectInstanceFactory factory =
                        new BeanFactoryAspectInstanceFactory(this.beanFactory, beanName);
                List<Advisor> classAdvisors = this.advisorFactory.getAdvisors(factory);
                if (this.beanFactory.isSingleton(beanName)) {
                    this.advisorsCache.put(beanName, classAdvisors);
                }
                else {
                    this.aspectFactoryCache.put(beanName, factory);
                }
                advisors.addAll(classAdvisors);
            }
            else {
                // Per target or per this.
                if (this.beanFactory.isSingleton(beanName)) {
                    throw new IllegalArgumentException("Bean with name '" + beanName +
                            "' is a singleton, but aspect instantiation model is not singleton");
                }
                MetadataAwareAspectInstanceFactory factory =
                        new PrototypeAspectInstanceFactory(this.beanFactory, beanName);
                this.aspectFactoryCache.put(beanName, factory);
                advisors.addAll(this.advisorFactory.getAdvisors(factory));
            }
        }
    }
```

点评：获取了@Aspect 的类，然后交给 BeanFactoryAspectInstanceFactory 生成 Advisor。

## 1.1.2 BeanFactoryAspectInstanceFactory

## getAdvisors()

```
@Override
public List<Advisor> getAdvisors(MetadataAwareAspectInstanceFactory aspectInstanceFactory) {
    Class<?> aspectClass = aspectInstanceFactory.getAspectMetadata().getAspectClass();beanType
    String aspectName = aspectInstanceFactory.getAspectMetadata().getAspectName();  beanName
    validate(aspectClass);  1

    // We need to wrap the MetadataAwareAspectInstanceFactory with a decorator
    // so that it will only instantiate once.
    MetadataAwareAspectInstanceFactory lazySingletonAspectInstanceFactory =
            new LazySingletonAspectInstanceFactoryDecorator(aspectInstanceFactory);

    List<Advisor> advisors = new ArrayList<>();
    for (Method method : getAdvisorMethods(aspectClass)) {  2
        Advisor advisor = getAdvisor(method, lazySingletonAspectInstanceFactory, advisors.size(), aspectName);
        if (advisor != null) {
            advisors.add(advisor);  3
        }
    }

    // If it's a per target aspect, emit the dummy instantiating aspect.
    if (!advisors.isEmpty() && lazySingletonAspectInstanceFactory.getAspectMetadata().isLazilyInstantiated()) {
        Advisor instantiationAdvisor = new SyntheticInstantiationAdvisor(lazySingletonAspectInstanceFactory);
        advisors.add( index: 0, instantiationAdvisor);
    }
```

点评：Aspect 类里面一个带 PointCut 的方法就是一个 Advisor(编程式注解配置)

### 1.1.2.1 validate()

```
public void validate(Class<?> aspectClass) throws AopConfigException {
    // If the parent has the annotation and isn't abstract it's an error
    if (aspectClass.getSuperclass().getAnnotation(Aspect.class) != null &&
            !Modifier.isAbstract(aspectClass.getSuperclass().getModifiers())) {
        throw new AopConfigException("[" + aspectClass.getName() + "] cannot extend concrete aspect [" +
                aspectClass.getSuperclass().getName() + "]"); 抽象方法不能拦截
    }

    AjType<?> ajType = AjTypeSystem.getAjType(aspectClass);
    if (!ajType.isAspect()) {
        throw new NotAnAtAspectException(aspectClass);
    }
    if (ajType.getPerClause().getKind() == PerClauseKind.PERCFLOW) {
        throw new AopConfigException(aspectClass.getName() + " uses percflow instantiation model: " +
                "This is not supported in Spring AOP.");
    }
    if (ajType.getPerClause().getKind() == PerClauseKind.PERCFLOWBELOW) {
        throw new AopConfigException(aspectClass.getName() + " uses percflowbelow instantiation model: " +
                "This is not supported in Spring AOP.");
    }
}
```

### 1.1.2.2 getAdvisorMethods()

```
private List<Method> getAdvisorMethods(Class<?> aspectClass) {
    final List<Method> methods = new ArrayList<>();
    ReflectionUtils.doWithMethods(aspectClass, method -> {
        // Exclude pointcuts
        if (AnnotationUtils.getAnnotation(method, Pointcut.class) == null) {
            methods.add(method);
        }
    });
    methods.sort(METHOD_COMPARATOR);
    return methods;
}
```

剔除PointCut，不要看反了

点评：获取所有@Aspect 类中的非@PointCut 注解的方法

### getAdvisor()【ReflectiveAspectJAdvisorFactory】

```
public Advisor getAdvisor(Method candidateAdviceMethod, MetadataAwareAspectInstanceFactory aspectInstanceFactory,
        int declarationOrderInAspect, String aspectName) {

    validate(aspectInstanceFactory.getAspectMetadata().getAspectClass());

    AspectJExpressionPointcut expressionPointcut = getPointcut(
            candidateAdviceMethod, aspectInstanceFactory.getAspectMetadata().getAspectClass());
    if (expressionPointcut == null) {
        return null;
    }

    return new InstantiationModelAwarePointcutAdvisorImpl(expressionPointcut, candidateAdviceMethod,
            aspectJAdvisorFactory: this, aspectInstanceFactory, declarationOrderInAspect, aspectName);
}
```

PointCut表达式对象化

切入点　　被拦截方法

拦截顺序

点评：非@PointCut 注解的方法，获取对应的 PointCut 表达式

```
@Nullable
private AspectJExpressionPointcut getPointcut(Method candidateAdviceMethod, Class<?> candidateAspectClass) {
    AspectJAnnotation<?> aspectJAnnotation =
            AbstractAspectJAdvisorFactory.findAspectJAnnotationOnMethod(candidateAdviceMethod);
    if (aspectJAnnotation == null) {
        return null;
    }

    AspectJExpressionPointcut ajexp =
            new AspectJExpressionPointcut(candidateAspectClass, new String[0], new Class<?>[0]);
    ajexp.setExpression(aspectJAnnotation.getPointcutExpression());
    if (this.beanFactory != null) {
        ajexp.setBeanFactory(this.beanFactory);
    }
    return ajexp;
}
```

设置pointCut的表达式

expressionPointCut:

```
private static final Class<?>[] ASPECTJ_ANNOTATION_CLASSES = new Class<?>[] {
        Pointcut.class, Around.class, Before.class, After.class, AfterReturning.class, AfterThrowing.class};
```

点评：只取其一

```
@Nullable
private static <A extends Annotation> AspectJAnnotation<A> findAnnotation(Method method, Class<A> toLookFor) {
    A result = AnnotationUtils.findAnnotation(method, toLookFor);
    if (result != null) {
        return new AspectJAnnotation<>(result);
    }
    else {
        return null;
    }
}
```

点评：根据非@PointCut 方法生成 Advisor（含注解元数据，PointCut 方法等）

## InstantiationModelAwarePointcutAdvisorImpl

InstantiationModelAwarePointcutAdvisorImpl.pdf

```
public InstantiationModelAwarePointcutAdvisorImpl(AspectJExpressionPointcut declaredPointcut,
        Method aspectJAdviceMethod, AspectJAdvisorFactory aspectJAdvisorFactory,
        MetadataAwareAspectInstanceFactory aspectInstanceFactory, int declarationOrder, String aspectName) {

    this.declaredPointcut = declaredPointcut;
    this.declaringClass = aspectJAdviceMethod.getDeclaringClass();
    this.methodName = aspectJAdviceMethod.getName();
    this.parameterTypes = aspectJAdviceMethod.getParameterTypes();
    this.aspectJAdviceMethod = aspectJAdviceMethod;
    this.aspectJAdvisorFactory = aspectJAdvisorFactory;
    this.aspectInstanceFactory = aspectInstanceFactory;
    this.declarationOrder = declarationOrder;
    this.aspectName = aspectName;

    if (aspectInstanceFactory.getAspectMetadata().isLazilyInstantiated()) {
        // Static part of the pointcut is a lazy type.
        Pointcut preInstantiationPointcut = Pointcuts.union(
                aspectInstanceFactory.getAspectMetadata().getPerClausePointcut(), this.declaredPointcut);

        // Make it dynamic: must mutate from pre-instantiation to post-instantiation state.
        // If it's not a dynamic pointcut, it may be optimized out
        // by the Spring AOP infrastructure after the first evaluation.
        this.pointcut = new PerTargetInstantiationModelPointcut(
                this.declaredPointcut, preInstantiationPointcut, aspectInstanceFactory);
        this.lazy = true;
    }
    else {
        // A singleton aspect.
        this.pointcut = this.declaredPointcut;
        this.lazy = false;
        this.instantiatedAdvice = instantiateAdvice(this.declaredPointcut);
    }
```

**instantiatedAdvice**:实例化的 Advice（MethodInterceptor）

instantiateAdvice()

```
private Advice instantiateAdvice(AspectJExpressionPointcut pointcut) {
    Advice advice = this.aspectJAdvisorFactory.getAdvice(this.aspectJAdviceMethod, pointcut,
            this.aspectInstanceFactory, this.declarationOrder, this.aspectName);
    return (advice != null ? advice : EMPTY_ADVICE);
}
```

ReflectiveAspectJAdvisorFactory

```
@Nullable
public Advice getAdvice(Method candidateAdviceMethod, AspectJExpressionPointcut expressionPointcut,
        MetadataAwareAspectInstanceFactory aspectInstanceFactory, int declarationOrder, String aspectName) {

    Class<?> candidateAspectClass = aspectInstanceFactory.getAspectMetadata().getAspectClass();
    validate(candidateAspectClass);

    AspectJAnnotation<?> aspectJAnnotation = 获取方法上的注解
            AbstractAspectJAdvisorFactory.findAspectJAnnotationOnMethod(candidateAdviceMethod);
    if (aspectJAnnotation == null) {
        return null;
    }

    // If we get here, we know we have an AspectJ method.
    // Check that it's an AspectJ-annotated class
    if (!isAspect(candidateAspectClass)) {
        throw new AopConfigException("Advice must be declared inside an aspect type: " +
                "Offending method '" + candidateAdviceMethod + "' in class [" +
                candidateAspectClass.getName() + "]");
    }

    if (logger.isDebugEnabled()) {
        logger.debug( o: "Found AspectJ method: " + candidateAdviceMethod);
    }

    AbstractAspectJAdvice springAdvice;
```

AbstractAspectJAdvice

AspectJAroundAdvice.pdf

```
switch (aspectJAnnotation.getAnnotationType()) {
    case AtPointcut:
        if (logger.isDebugEnabled()) {
            logger.debug( o: "Processing pointcut '" + candidateAdviceMethod.getName() + "'");
        }
        return null;
    case AtAround:
        springAdvice = new AspectJAroundAdvice(
                candidateAdviceMethod, expressionPointcut, aspectInstanceFactory);
        break;
    case AtBefore:
        springAdvice = new AspectJMethodBeforeAdvice(
                candidateAdviceMethod, expressionPointcut, aspectInstanceFactory);
        break;
    case AtAfter:
        springAdvice = new AspectJAfterAdvice(
                candidateAdviceMethod, expressionPointcut, aspectInstanceFactory);
        break;
    case AtAfterReturning:
        springAdvice = new AspectJAfterReturningAdvice(
                candidateAdviceMethod, expressionPointcut, aspectInstanceFactory);
        AfterReturning afterReturningAnnotation = (AfterReturning) aspectJAnnotation.getAnnotation();
        if (StringUtils.hasText(afterReturningAnnotation.returning())) {
            springAdvice.setReturningName(afterReturningAnnotation.returning());
        }
        break;
    case AtAfterThrowing:
        springAdvice = new AspectJAfterThrowingAdvice(
                candidateAdviceMethod, expressionPointcut, aspectInstanceFactory);
        AfterThrowing afterThrowingAnnotation = (AfterThrowing) aspectJAnnotation.getAnnotation();
        if (StringUtils.hasText(afterThrowingAnnotation.throwing())) {
            springAdvice.setThrowingName(afterThrowingAnnotation.throwing());
```

```
// Now to configure the advice...
springAdvice.setAspectName(aspectName);
springAdvice.setDeclarationOrder(declarationOrder);
String[] argNames = this.parameterNameDiscoverer.getParameterNames(candidateAdviceMethod);
if (argNames != null) {
    springAdvice.setArgumentNamesFromStringArray(argNames);
}
springAdvice.calculateArgumentBindings();

return springAdvice;
```

获取注解上的参数属性

点评：Advisor 对象包含一个 PointCut 对象和一个 Advice 对象（实际执行拦截方法的对象）。

## 1.2 findAdvisorsThatCanApply()

```
/**
 * Search the given candidate Advisors to find all Advisors that
 * can apply to the specified bean.
 * @param candidateAdvisors the candidate Advisors
 * @param beanClass the target's bean class
 * @param beanName the target's bean name
 * @return the List of applicable Advisors
 * @see ProxyCreationContext#getCurrentProxiedBeanName()
 */
protected List<Advisor> findAdvisorsThatCanApply(
        List<Advisor> candidateAdvisors, Class<?> beanClass, String beanName) {

    ProxyCreationContext.setCurrentProxiedBeanName(beanName);
    try {
        return AopUtils.findAdvisorsThatCanApply(candidateAdvisors, beanClass);
    }
    finally {
        ProxyCreationContext.setCurrentProxiedBeanName(null);
    }
}
```

# 2\createProxy()

```
protected Object createProxy(Class<?> beanClass, @Nullable String beanName,
        @Nullable Object[] specificInterceptors, TargetSource targetSource) {

    if (this.beanFactory instanceof ConfigurableListableBeanFactory) {
        AutoProxyUtils.exposeTargetClass((ConfigurableListableBeanFactory) this.beanFactory, beanName, beanClass);
    }

    ProxyFactory proxyFactory = new ProxyFactory();
    proxyFactory.copyFrom( other: this);

    if (!proxyFactory.isProxyTargetClass()) {
        if (shouldProxyTargetClass(beanClass, beanName)) {
            proxyFactory.setProxyTargetClass(true);        指定目标类代理
        }
        else {
            evaluateProxyInterfaces(beanClass, proxyFactory);接口代理
        }
    }

    Advisor[] advisors = buildAdvisors(beanName, specificInterceptors);
    proxyFactory.addAdvisors(advisors);
    proxyFactory.setTargetSource(targetSource);
    customizeProxyFactory(proxyFactory);

    proxyFactory.setFrozen(this.freezeProxy);
    if (advisorsPreFiltered()) {
        proxyFactory.setPreFiltered(true);
    }

    return proxyFactory.getProxy(getProxyClassLoader());
}
```

## 2.1 ProxyFactory

ProxyFactory-field.pdf

ProxyFactory-method.pdf

## getProxy()

```
/**
 * Create a new proxy according to the settings in this factory.
 * <p>Can be called repeatedly. Effect will vary if we've added
 * or removed interfaces. Can add and remove interceptors.
 * <p>Uses the given class loader (if necessary for proxy creation).
 * @param classLoader the class loader to create the proxy with
 * (or {@code null} for the low-level proxy facility's default)
 * @return the proxy object
 */
public Object getProxy(@Nullable ClassLoader classLoader) {
    return createAopProxy().getProxy(classLoader);
}
```

AopProxy → JdkDynamicAopPorxy

AopProxy → CglibAopProxy

```
protected final synchronized AopProxy createAopProxy() {
    if (!this.active) {
        activate();
    }
    return getAopProxyFactory().createAopProxy( config: this);
}
```

DefaultAopProxyFactory

```
public class DefaultAopProxyFactory implements AopProxyFactory, Serializable {

    @Override
    public AopProxy createAopProxy(AdvisedSupport config) throws AopConfigException {
        if (config.isOptimize() || config.isProxyTargetClass() || hasNoUserSuppliedProxyInterfaces(config)) {
            Class<?> targetClass = config.getTargetClass();
            if (targetClass == null) {
                throw new AopConfigException("TargetSource cannot determine target class: " +
                        "Either an interface or a target is required for proxy creation.");
            }
            if (targetClass.isInterface() || Proxy.isProxyClass(targetClass)) {
                return new JdkDynamicAopProxy(config);
            }
            return new ObjenesisCglibAopProxy(config);非接口
        }
        else {
            return new JdkDynamicAopProxy(config); 接口
        }
    }
```

# AopProxy

AopProxy.pdf

## JdkDynamicAopProxy

## getProxy(classLoader)

```
@Override
public Object getProxy(@Nullable ClassLoader classLoader) {
    if (logger.isTraceEnabled()) {
        logger.trace( o: "Creating JDK dynamic proxy: " + this.advised.getTargetSource());
    }
    Class<?>[] proxiedInterfaces = AopProxyUtils.completeProxiedInterfaces(this.advised,  decoratingProxy: true);
    findDefinedEqualsAndHashCodeMethods(proxiedInterfaces);
    return Proxy.newProxyInstance(classLoader, proxiedInterfaces,  h: this);
}
```

获取代理要实现的接口

标记接口是否有equal、hashCode方法

AopProxy

生成代理对象

```
static Class<?>[] completeProxiedInterfaces(AdvisedSupport advised, boolean decoratingProxy) {
    Class<?>[] specifiedInterfaces = advised.getProxiedInterfaces();
    if (specifiedInterfaces.length == 0) {
        // No user-specified interfaces: check whether target class is an interface.
        Class<?> targetClass = advised.getTargetClass();
        if (targetClass != null) {
            if (targetClass.isInterface()) {
                advised.setInterfaces(targetClass);
            }
            else if (Proxy.isProxyClass(targetClass)) {
                advised.setInterfaces(targetClass.getInterfaces());
            }
            specifiedInterfaces = advised.getProxiedInterfaces();
        }
    }
    boolean addSpringProxy = !advised.isInterfaceProxied(SpringProxy.class);
    boolean addAdvised = !advised.isOpaque() && !advised.isInterfaceProxied(Advised.class);
    boolean addDecoratingProxy = (decoratingProxy && !advised.isInterfaceProxied(DecoratingProxy.class));
    int nonUserIfcCount = 0;
    if (addSpringProxy) {
        nonUserIfcCount++;
    }
    if (addAdvised) {
        nonUserIfcCount++;
    }
    if (addDecoratingProxy) {
        nonUserIfcCount++;
    }
```

点评：获取代理需要实现的接口

# invoke()

```
@Override
@Nullable
public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
    MethodInvocation invocation;
    Object oldProxy = null;
    boolean setProxyContext = false;

    TargetSource targetSource = this.advised.targetSource;
    Object target = null;

    try {
        if (!this.equalsDefined && AopUtils.isEqualsMethod(method)) {
            // The target does not implement the equals(Object) method itself.
            return equals(args[0]);
        }
        else if (!this.hashCodeDefined && AopUtils.isHashCodeMethod(method)) {
            // The target does not implement the hashCode() method itself.
            return hashCode();
        }
        else if (method.getDeclaringClass() == DecoratingProxy.class) {
            // There is only getDecoratedClass() declared -> dispatch to proxy config.
            return AopProxyUtils.ultimateTargetClass(this.advised);
        }
        else if (!this.advised.opaque && method.getDeclaringClass().isInterface() &&
                method.getDeclaringClass().isAssignableFrom(Advised.class)) {
            // Service invocations on ProxyConfig with the proxy config...
            return AopUtils.invokeJoinpointUsingReflection(this.advised, method, args);
        }
```

```
Object retVal;

if (this.advised.exposeProxy) {
    // Make invocation available if necessary.
    oldProxy = AopContext.setCurrentProxy(proxy);
    setProxyContext = true;
}

// Get as late as possible to minimize the time we "own" the target,
// in case it comes from a pool.
target = targetSource.getTarget();
Class<?> targetClass = (target != null ? target.getClass() : null);

// Get the interception chain for this method. 1、创建拦截器链
List<Object> chain = this.advised.getInterceptorsAndDynamicInterceptionAdvice(method, targetClass);

// Check whether we have any advice. If we don't, we can fallback on direct
// reflective invocation of the target, and avoid creating a MethodInvocation.
if (chain.isEmpty()) {
    // We can skip creating a MethodInvocation: just invoke the target directly
    // Note that the final invoker must be an InvokerInterceptor so we know it does
    // nothing but a reflective operation on the target, and no hot swapping or fancy proxying.
    Object[] argsToUse = AopProxyUtils.adaptArgumentsIfNecessary(method, args);
    retVal = AopUtils.invokeJoinpointUsingReflection(target, method, argsToUse);
}               如果拦截器链为空，直接调用
else {
    // We need to create a method invocation...
    invocation = new ReflectiveMethodInvocation(proxy, target, method, args, targetClass, chain);
    // Proceed to the joinpoint through the interceptor chain.
    retVal = invocation.proceed();
}       生成带拦截器调用的方法
```

点评：chain 是 Advice/MethodInterceptor

## getInterceptorsAndDynamicInterceptionAdvice()

```
public List<Object> getInterceptorsAndDynamicInterceptionAdvice(Method method, @Nullable Class<?> targetClass) {
    MethodCacheKey cacheKey = new MethodCacheKey(method);
    List<Object> cached = this.methodCache.get(cacheKey);
    if (cached == null) {
        cached = this.advisorChainFactory.getInterceptorsAndDynamicInterceptionAdvice(
                config: this, method, targetClass);
        this.methodCache.put(cacheKey, cached);
    }
    return cached;
}
```

DefaultAdvisorChainFactory

## getInterceptorsAndDynamicInterceptionAdvice（）

```
public List<Object> getInterceptorsAndDynamicInterceptionAdvice(
        Advised config, Method method, @Nullable Class<?> targetClass) {

    // This is somewhat tricky... We have to process introductions first,
    // but we need to preserve order in the ultimate list.
    AdvisorAdapterRegistry registry = GlobalAdvisorAdapterRegistry.getInstance();
    Advisor[] advisors = config.getAdvisors();
    List<Object> interceptorList = new ArrayList<>(advisors.length);
    Class<?> actualClass = (targetClass != null ? targetClass : method.getDeclaringClass());
    Boolean hasIntroductions = null;

    for (Advisor advisor : advisors) {
        if (advisor instanceof PointcutAdvisor) {
            // Add it conditionally.
            PointcutAdvisor pointcutAdvisor = (PointcutAdvisor) advisor;
            if (config.isPreFiltered() || pointcutAdvisor.getPointcut().getClassFilter().matches(actualClass)) {
                MethodMatcher mm = pointcutAdvisor.getPointcut().getMethodMatcher();
                boolean match;
                if (mm instanceof IntroductionAwareMethodMatcher) {
                    if (hasIntroductions == null) {
                        hasIntroductions = hasMatchingIntroductions(advisors, actualClass);
                    }
                    match = ((IntroductionAwareMethodMatcher) mm).matches(method, actualClass, hasIntroductions);
                }
                else {
                    match = mm.matches(method, actualClass);
                }
```

根据PointCut表达式匹配类

根据PointCut表达式匹配方法

```
            if (match) {
                MethodInterceptor[] interceptors = registry.getInterceptors(advisor);
                if (mm.isRuntime()) {                    获取拦截器Advice
                    // Creating a new object instance in the getInterceptors() method
                    // isn't a problem as we normally cache created chains.
                    for (MethodInterceptor interceptor : interceptors) {
                        interceptorList.add(new InterceptorAndDynamicMethodMatcher(interceptor, mm));
                    }
                }
                else {
                    interceptorList.addAll(Arrays.asList(interceptors));
                }
            }
        }
    }
```

```
@Override
public MethodInterceptor[] getInterceptors(Advisor advisor) throws UnknownAdviceTypeException {
    List<MethodInterceptor> interceptors = new ArrayList<>( initialCapacity: 3);
    Advice advice = advisor.getAdvice();
    if (advice instanceof MethodInterceptor) {
        interceptors.add((MethodInterceptor) advice);
    }
    for (AdvisorAdapter adapter : this.adapters) {
        if (adapter.supportsAdvice(advice)) {
            interceptors.add(adapter.getInterceptor(advisor));
        }
    }
    if (interceptors.isEmpty()) {
        throw new UnknownAdviceTypeException(advisor.getAdvice());
    }
    return interceptors.toArray(new MethodInterceptor[0]);
}
```

## ReflectiveMethodInvocation

```
protected ReflectiveMethodInvocation(
        Object proxy, @Nullable Object target, Method method, @Nullable Object[] arguments,
        @Nullable Class<?> targetClass, List<Object> interceptorsAndDynamicMethodMatchers) {

    this.proxy = proxy;
    this.target = target;
    this.targetClass = targetClass;                    代理接口
    this.method = BridgeMethodResolver.findBridgedMethod(method);          拦截的方法
    this.arguments = AopProxyUtils.adaptArgumentsIfNecessary(method, arguments);    argments
    this.interceptorsAndDynamicMethodMatchers = interceptorsAndDynamicMethodMatchers;    advice
}
```

## proceed()

```
@Override
@Nullable
public Object proceed() throws Throwable {
    //  We start with an index of -1 and increment early.
    if (this.currentInterceptorIndex == this.interceptorsAndDynamicMethodMatchers.size() - 1) {
        return invokeJoinpoint();
    }

    Object interceptorOrInterceptionAdvice =
            this.interceptorsAndDynamicMethodMatchers.get(++this.currentInterceptorIndex);
    if (interceptorOrInterceptionAdvice instanceof InterceptorAndDynamicMethodMatcher) {
        // Evaluate dynamic method matcher here: static part will already have
        // been evaluated and found to match.
        InterceptorAndDynamicMethodMatcher dm =
                (InterceptorAndDynamicMethodMatcher) interceptorOrInterceptionAdvice;
        Class<?> targetClass = (this.targetClass != null ? this.targetClass : this.method.getDeclaringClass());
        if (dm.methodMatcher.matches(this.method, targetClass, this.arguments)) {
            return dm.interceptor.invoke( invocation: this);
        }
        else {
            // Dynamic matching failed.
            // Skip this interceptor and invoke the next in the chain.
            return proceed(); 递归
        }
    }
    else {
        // It's an interceptor, so we just invoke it: The pointcut will have
        // been evaluated statically before this object was constructed.
        return ((MethodInterceptor) interceptorOrInterceptionAdvice).invoke( invocation: this);
    }
}
```

## invokeJoinPoint()

```
/**
 * Invoke the joinpoint using reflection.
 * Subclasses can override this to use custom invocation.
 * @return the return value of the joinpoint
 * @throws Throwable if invoking the joinpoint resulted in an exception
 */
@Nullable
protected Object invokeJoinpoint() throws Throwable {
    return AopUtils.invokeJoinpointUsingReflection(this.target, this.method, this.arguments);
}
```

```java
public static Object invokeJoinpointUsingReflection(@Nullable Object target, Method method, Object[] args)
        throws Throwable {

    // Use reflection to invoke the method.
    try {
        ReflectionUtils.makeAccessible(method);
        return method.invoke(target, args);
    }
    catch (InvocationTargetException ex) {
        // Invoked method threw a checked exception.
        // We must rethrow it. The client won't see the interceptor.
        throw ex.getTargetException();
    }
    catch (IllegalArgumentException ex) {
        throw new AopInvocationException("AOP configuration seems to be invalid: tried calling method [" +
                method + "] on target [" + target + "]", ex);
    }
    catch (IllegalAccessException ex) {
        throw new AopInvocationException("Could not access method [" + method + "]", ex);
    }
}
```

反射调用

CglibAopProxy

# Advice

Advice.pdf

AbstractAspectJAdvice.pdf

invoke()

# ReflectiveAspectJAdvisorFactory

ReflectiveAspectJAdvisorFactory.pdf

getAdvice()

getAdvisor()

# Advisor


Advisor.pdf

## PointCutAdvisor


PointcutAdvisor-field.pdf


PointcutAdvisor-method.pdf

# PointCut


Pointcut-field.pdf


Pointcut-method.pdf

## match(targetClass)

```java
@Override
public boolean matches(Class<?> targetClass) {
    PointcutExpression pointcutExpression = obtainPointcutExpression();
    try {
        try {
            return pointcutExpression.couldMatchJoinPointsInType(targetClass);
        }
        catch (ReflectionWorldException ex) {
            logger.debug( o: "PointcutExpression matching rejected target class - trying fallback expression", ex);
            // Actually this is still a "maybe" - treat the pointcut as dynamic if we don't know enough yet
            PointcutExpression fallbackExpression = getFallbackPointcutExpression(targetClass);
            if (fallbackExpression != null) {
                return fallbackExpression.couldMatchJoinPointsInType(targetClass);
            }
        }
    }
    catch (Throwable ex) {
        logger.debug( o: "PointcutExpression matching rejected target class", ex);
    }
    return false;
}
```

```java
@Override
public FuzzyBoolean fastMatch(FastMatchInfo info) {
    if (info.getKind() == Shadow.StaticInitialization) {
        return annotationTypePattern.fastMatches(info.getType());
    } else {
        return FuzzyBoolean.MAYBE;
    }
}
```

## PointcutExpression

PointcutExpression.pdf

# Advice

Advice.pdf