AOP (Aspect Oriented Programming)

- In simple term, It helps to Intercept the method invocation. And we can perform some task before and after the method.
- AOP allow us to focus on business logic by handling boilerplate and repetitive code like logging, transaction management etc.
- So, Aspect is a module which handle this repetitive or boilerplate code.
- Helps in achieving reusability, maintainability of the code.

Used during:
- Logging
- Transaction Management
- Security etc..

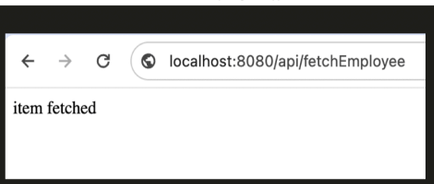
Dependency you need to add in pom.xml:

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

```java
@RestController
@RequestMapping(value = "/api/")
public class Employee {

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        return "item fetched";
    }
}
```

```java
@Component
@Aspect
public class LoggingAspect {

    @Before("execution(public String com.conceptandcoding.learningspringboot.Employee.fetchEmployee())")
    public void beforeMethod(){
        System.out.println("inside beforeMethod Aspect");
    }
}
```

```
2024-06-16T23:03:38.117+05:30  INFO 18766 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.4.4.Final
2024-06-16T23:03:38.137+05:30  INFO 18766 --- [           main] o.h.c.internal.RegionFactoryInitiator    : HHH000026: Second-level cache disabled
2024-06-16T23:03:38.237+05:30  INFO 18766 --- [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
2024-06-16T23:03:38.398+05:30  INFO 18766 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform'
2024-06-16T23:03:38.400+05:30  INFO 18766 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-06-16T23:03:38.440+05:30  WARN 18766 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may
2024-06-16T23:03:38.632+05:30  INFO 18766 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-06-16T23:03:38.637+05:30  INFO 18766 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 1.517 seconds (process running for 1.727)
```




```
2024-06-16T23:03:38.237+05:30  INFO 18766 --- [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
2024-06-16T23:03:38.398+05:30  INFO 18766 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.tra
2024-06-16T23:03:38.400+05:30  INFO 18766 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence uni
2024-06-16T23:03:38.440+05:30  WARN 18766 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore
2024-06-16T23:03:38.632+05:30  INFO 18766 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-06-16T23:03:38.637+05:30  INFO 18766 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 1.517 seconds (process
2024-06-16T23:05:49.486+05:30  INFO 18766 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-06-16T23:05:49.486+05:30  INFO 18766 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-06-16T23:05:49.487+05:30  INFO 18766 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
inside beforeMethod Aspect
```

Some important AOP concepts :

```
@Component
@Aspect
public class LoggingAspect {

    @Before("execution(public String com.conceptandcoding.learningspringboot.Employee.fetchEmployee())")
    public void beforeMethod(){
        System.out.println("inside beforeMethod Aspect");
    }
}
```

Access-modifiers: optional and can be omitted

Return type: optional but can not be omitted

This is a pointcut

This @Before and method together is called Advice

Pointcut:

Its an Expression, which tells where an ADVICE should be applied.

Types of Pointcut:

1. Execution: matches a particular method in a particular class.
   `@Before("execution(public String com.conceptandcoding.learningspringboot.Employee.fetchEmployee())")`

   → (*) wildcard : matches any single item

   Matches any return type

```
@Before("execution(* com.conceptandcoding.learningspringboot.Employee.fetchEmployee())")
public void beforeMethod(){
    System.out.println("inside beforeMethod Aspect");
}
```

   Matches any method with single parameter String

```
@Before("execution(* com.conceptandcoding.learningspringboot.Employee.*(String))")
public void beforeMethod(){
    System.out.println("inside beforeMethod Aspect");
}
```

   Matches fetchEmployee method that take any single parameter

```
@Before("execution(String com.conceptandcoding.learningspringboot.Employee.fetchEmployee(*))")
public void beforeMethod(){
    System.out.println("inside beforeMethod Aspect");
}
```

(..) wildcard : matches 0 or More item

Matches fetchEmployee method that take any 0 or More parameters

```
@Before("execution(String com.conceptandcoding.learningspringboot.Employee.fetchEmployee(..))")
public void beforeMethod(){
    System.out.println("inside beforeMethod Aspect");
}
```

Matches fetchEmployee method in *'com.conceptandcoding'* package and subpackage classes

```
@Before("execution(String com.conceptandcoding..fetchEmployee())")
public void beforeMethod(){
    System.out.println("inside beforeMethod Aspect");
}
```

Matches any method in *'com.conceptandcoding'* package and subpackage classes

```
@Before("execution(String com.conceptandcoding..*())")
public void beforeMethod(){
    System.out.println("inside beforeMethod Aspect");
}
```

2. Within: matches all method within any class or package.

This pointcut will run for each method in the class Employee

```
@Before("within(com.conceptandcoding.learningspringboot.Employee)")
```

This pointcut will run for each method in this package and subpackage

```
@Before("within(com.conceptandcoding.learningspringboot..*)")
```

3. @within: matches any method in a class which has this annotation.

```java
@RestController
@RequestMapping(value = "/api/")
public class Employee {

    @Autowired
    EmployeeUtil employeeUtil;

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        employeeUtil.employeeHelperMethod();
        return "item fetched";
    }
}
```

```java
@Aspect
@Component
public class LoggingAspect {

    @Before("@within(org.springframework.stereotype.Service)")
    public void beforeMethod() {
        System.out.println("inside beforeMethod aspect");
    }
}
```

```java
@Service
public class EmployeeUtil {

    public void employeeHelperMethod() {
        System.out.println("employee helper method called");
    }
}
```

```
2024-06-22T11:54:02.459+05:30  INFO 66918 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 1.646 s
2024-06-22T11:54:04.070+05:30  INFO 66918 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet '
2024-06-22T11:54:04.070+05:30  INFO 66918 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-06-22T11:54:04.070+05:30  INFO 66918 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 0 ms
inside beforeMethod aspect
employee helper method called
```

4. @annotation: matches any method that is annotated with given annotation

```java
@RestController
@RequestMapping(value = "/api/")
public class Employee {

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        return "item fetched";
    }
}
```

```java
@Component
@Aspect
public class LoggingAspect {

    @Before("@annotation(org.springframework.web.bind.annotation.GetMapping)")
    public void beforeMethod(){
        System.out.println("inside beforeMethod Aspect");
    }
}
```

5. Args: matches any method with particular arguments (or parameters)

@Before("args(String,int)")

```java
@RestController
@RequestMapping(value = "/api/")
public class Employee {

    @Autowired
    EmployeeUtil employeeUtil;

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        employeeUtil.employeeHelperMethod( str: "xyz",  val: 123);
        return "item fetched";
    }
}
```

```java
@Aspect
@Component
public class LoggingAspect {

    @Before("args(String, int)")
    public void beforeMethod() {
        System.out.println("inside beforeMethod aspect");
    }
}
```

```java
@Service
public class EmployeeUtil {

    public void employeeHelperMethod(String str, int val) {
        System.out.println("employee helper method called");
    }
}
```

```
2024-06-22T11:13:27.258+05:30  INFO 64094 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing Persist
2024-06-22T11:13:27.290+05:30  INFO 64094 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core
2024-06-22T11:13:27.309+05:30  INFO 64094 --- [           main] o.h.c.internal.RegionFactoryInitiator    : HHH000026: Second-level cache
2024-06-22T11:13:27.414+05:30  INFO 64094 --- [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: igno
2024-06-22T11:13:27.583+05:30  INFO 64094 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform av
2024-06-22T11:13:27.585+05:30  INFO 64094 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManager
2024-06-22T11:13:27.636+05:30  WARN 64094 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is en
2024-06-22T11:13:27.868+05:30  INFO 64094 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (
2024-06-22T11:13:27.874+05:30  INFO 64094 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication
2024-06-22T11:13:30.408+05:30  INFO 64094 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring Dispatche
2024-06-22T11:13:30.408+05:30  INFO 64094 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatc
2024-06-22T11:13:30.409+05:30  INFO 64094 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1
inside beforeMethod aspect
employee helper method called
```

If instead of primitive type, we need object, then we can give like this

@Before("args(com.conceptandcoding.learningspringboot.Employee)")

6. @args: matches any method with particular parameters and that parameter class is annotated with particular annotation.

@Before("@args(org.springframework.stereotype.Service)")

```java
@Service
public class EmployeeUtil {

    public void employeeHelperMethod(EmployeeDAO employeeDAO) {
        System.out.println("employee helper method called");
    }
}
```

```java
@Aspect
@Component
public class LoggingAspect {

    @Before("@args(org.springframework.stereotype.Service)")
    public void beforeMethod() {
        System.out.println("inside beforeMethod aspect");
    }
}
```

```java
@Service
public class EmployeeDAO {

}
```

7. target: matches any method on a particular instance of a class.

```java
@RestController
@RequestMapping(value = "/api/")
public class Employee {

    @Autowired
    EmployeeUtil employeeUtil;

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        employeeUtil.employeeHelperMethod();
        return "item fetched";
    }
}
```

```java
@Aspect
@Component
public class LoggingAspect {

    @Before("target(com.conceptandcoding.learningspringboot.EmployeeUtil)")
    public void beforeMethod() {
        System.out.println("inside beforeMethod aspect");
    }
}
```

```java
@Component
public class EmployeeUtil {

    public void employeeHelperMethod() {
        System.out.println("employee helper method called");
    }
}
```

Interface

@Before("target(com.conceptandcoding.learningspringboot.IEmployee)")

```java
@RestController
@RequestMapping(value = "/api/")
public class EmployeeController {

    @Autowired
    @Qualifier("tempEmployee")
    IEmployee employeeObj;

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        employeeObj.fetchEmployeeMethod();
        return "item fetched";
    }
}
```

```java
@Aspect
@Component
public class LoggingAspect {

    @Before("target(com.conceptandcoding.learningspringboot.IEmployee)")
    public void beforeMethod() {
        System.out.println("inside beforeMethod aspect");
    }
}
```

```java
public interface IEmployee {

    public void fetchEmployeeMethod();
}
```

```java
@Component
@Qualifier("tempEmployee")
public class TempEmployee implements IEmployee{
    @Override
    public void fetchEmployeeMethod() {
        System.out.println("in temp Employee fetch method");
    }
}
```

```java
@Component
@Qualifier("permaEmployee")
public class PermanentEmployee implements IEmployee{
    @Override
    public void fetchEmployeeMethod() {
        System.out.println("inside permanent fetch employee method");
    }
}
```

```
2024-06-22T12:12:42.485+05:30  INFO 67695 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication
2024-06-22T12:12:48.192+05:30  INFO 67695 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring Dispatcher
2024-06-22T12:12:48.192+05:30  INFO 67695 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatch
2024-06-22T12:12:48.193+05:30  INFO 67695 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1
inside beforeMethod aspect
in temp Employee fetch method
```

Combining two pointcuts using:

&& (boolean and)
|| (boolean or)

```java
@RestController
@RequestMapping(value = "/api/")
public class EmployeeController {

    @Autowired
    EmployeeUtil employeeUtil;

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        return "item fetched";
    }
}
```

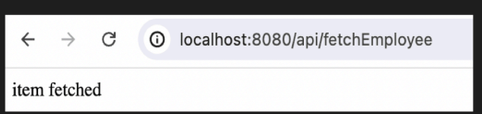
```java
@Aspect
@Component
public class LoggingAspect {

    @Before("execution(* com.conceptandcoding.learningspringboot.EmployeeController.*())"
            +
            "&& @within(org.springframework.web.bind.annotation.RestController)")
    public void beforeAndMethod() {
        System.out.println("inside beforeAndMethod aspect");
    }

    @Before("execution(* com.conceptandcoding.learningspringboot.EmployeeController.*())"
            +
            "|| @within(org.springframework.stereotype.Component)")
    public void beforeOrMethod() {
        System.out.println("inside beforeOrMethod aspect");
    }

}
```

```
2024-06-22T13:02:39.698+05:30  INFO 70500 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispat
2024-06-22T13:02:39.698+05:30  INFO 70500 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-06-22T13:02:39.699+05:30  INFO 70500 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
inside beforeAndMethod aspect
inside beforeOrMethod aspect
```

Named Pointcuts

```java
@RestController
@RequestMapping(value = "/api/")
public class EmployeeController {

    @Autowired
    EmployeeUtil employeeUtil;

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        return "item fetched";
    }
}
```

```java
@Aspect
@Component
public class LoggingAspect {

    @Pointcut("execution(* com.conceptandcoding.learningspringboot.EmployeeController.*())")
    public void customPointcutName() {
        //always stays empty
    }

    @Before("customPointcutName()")
    public void beforeMethod() {
        System.out.println("inside beforeMethod aspect");
    }

}
```




```
2024-06-22T13:07:37.055+05:30  WARN 70647 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Th
2024-06-22T13:07:37.254+05:30  INFO 70647 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context p
2024-06-22T13:07:37.260+05:30  INFO 70647 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication in 1.526 seconds (p
2024-06-22T13:07:41.084+05:30  INFO 70647 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcher
2024-06-22T13:07:41.085+05:30  INFO 70647 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-06-22T13:07:41.085+05:30  INFO 70647 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 0 ms
inside beforeMethod aspect
```

Advice:

Its an action, which is taken @Before or @After or @Around the method execution.

```
@Component
@Aspect
public class LoggingAspect {

    @Before("execution(public String com.conceptandcoding.learningspringboot.Employee.fetchEmployee())")
    public void beforeMethod(){
        System.out.println("inside beforeMethod Aspect");
    }
}
```

Advice

@Before or @After: its simple and we have already seen

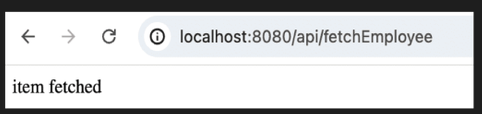
@Around:

As the name says, it surrounds the method execution (before and after both).

```
@RestController
@RequestMapping(value = "/api/")
public class EmployeeController {

    @Autowired
    EmployeeUtil employeeUtil;

    @GetMapping (path = "/fetchEmployee")
    public String fetchEmployee(){
        return "item fetched";
    }
}
```

```
@Aspect
@Component
public class LoggingAspect {

    @Around("execution(* com.conceptandcoding.learningspringboot.EmployeeUtil.*())")
    public void aroundMethod(ProceedingJoinPoint joinPoint) throws Throwable {
        System.out.println("inside before Method aspect");
        joinPoint.proceed();
        System.out.println("inside after Method aspect");
    }

}
```

```
@Component
public class EmployeeUtil {

    public void employeeHelperMethod() {
        System.out.println("employee helper method called");
    }
}
```



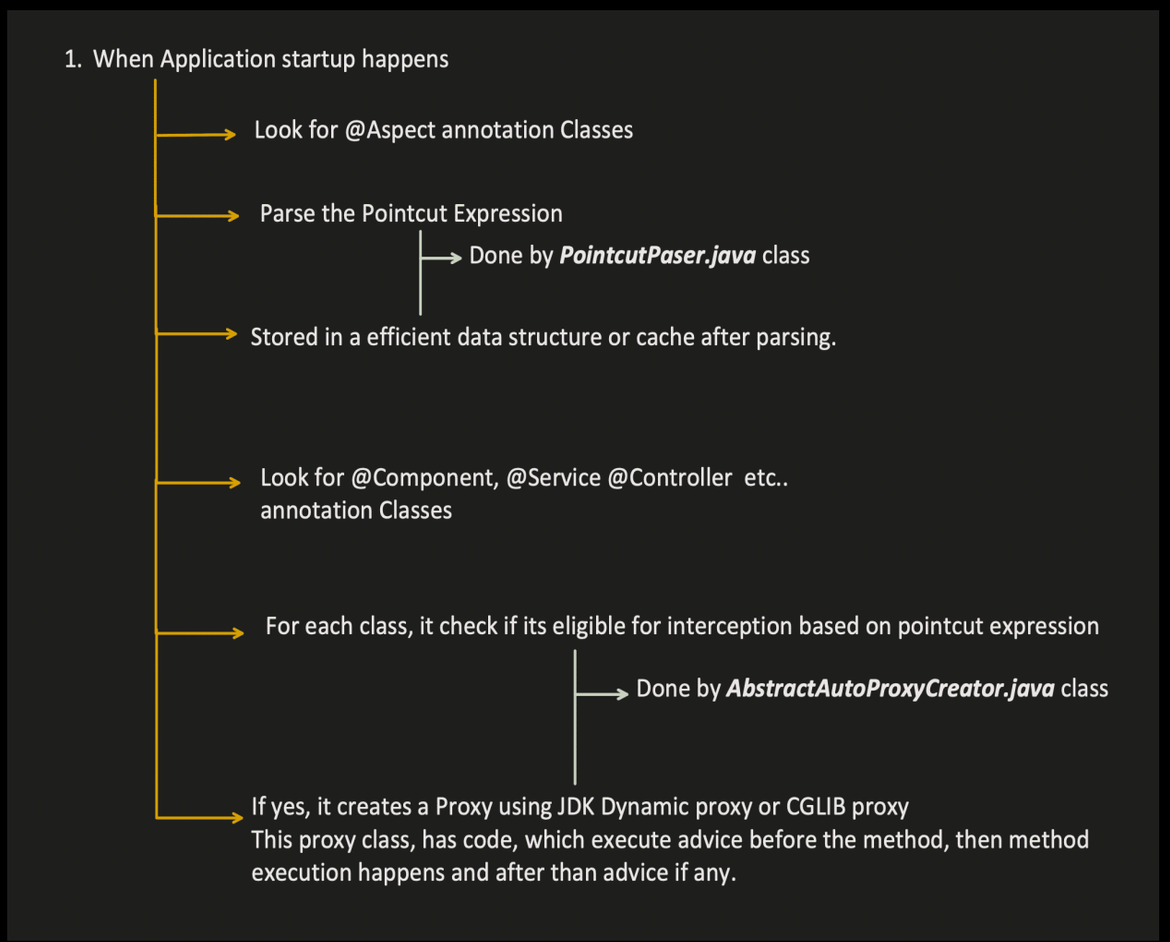

```
2024-06-22T16:25:39.014+05:30  INFO 79523 --- [           main] c.c.l.SpringbootApplication              : Started SpringbootApplication i
2024-06-22T16:25:41.031+05:30  INFO 79523 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherS
2024-06-22T16:25:41.031+05:30  INFO 79523 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatche
2024-06-22T16:25:41.032+05:30  INFO 79523 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 m
inside before Method aspect
fetching employee details
inside after Method aspect
```

Join Point: Its generally considered a point, where actual method invocation happens.

By now, few questions we all should have:

1. How this interception works?
2. What if we have 1000s of pointcut, so whenever I invoke a method, does matching happens with 1000s of pointcuts?

Let's understand the AOP flow, to get an answer of above doubts:


1. When Application startup happens
Look for @Aspect annotation Classes
Parse the Pointcut Expression
Done by PointcutPaser.java class
Stored in a efficient data structure or cache after parsing.
Look for @Component, @Service @Controller etc..
annotation Classes
For each class, it check if its eligible for interception based on pointcut expression
Done by AbstractAutoProxyCreator.java class
If yes, it creates a Proxy using JDK Dynamic proxy or CGLIB proxy
This proxy class, has code, which execute advice before the method, then method
execution happens and after than advice if any.

Now, when we initiate the Request after application startup, this Proxy class is actually invoked.

1. Parse the Pointcut Expression

PointcutParser.class

Decompiled .class file, bytecode version: 52.0 (Java 8) Download Sources

Search 0 results

```
171
172     public PointcutExpression parsePointcutExpression(String expression, Class<?> inScope, PointcutParameter[] formalParameters)
173         PointcutExpressionImpl pcExpr = null;
174
175         try {
176             Pointcut pc = this.resolvePointcutExpression(expression, inScope, formalParameters);
177             pc = this.concretizePointcutExpression(pc, inScope, formalParameters);
178             this.validateAgainstSupportedPrimitives(pc, expression);
179             pcExpr = new PointcutExpressionImpl(pc, expression, formalParameters, this.getWorld());
180             return pcExpr;
181         } catch (ParserException var6) {
182             throw new IllegalArgumentException(this.buildUserMessageFromParserException(expression, var6));
183         } catch (ReflectionWorld.ReflectionWorldException var7) {
184             throw new IllegalArgumentException(var7.getMessage());
185         }
186     }
187
```

2. Create Proxy class (either JDK Dynamic proxy or CGLIB proxy) if required for the class

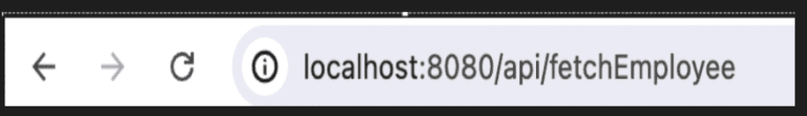
AbstractAutoProxyCreator.class

Decompiled .class file, bytecode version: 61.0 (Java 17) Download Sources Choose Sources...

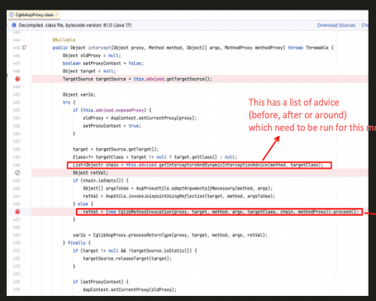
```
191         }
192     }
193
194     protected Object wrapIfNecessary(Object bean, String beanName, Object cacheKey) {  bean: EmployeeUtil@7593   beanName: "employeeUtil"
195         if (StringUtils.hasLength(beanName) && this.targetSourcedBeans.contains(beanName)) {  targetSourcedBeans:  size = 0
196             return bean;
197         } else if (Boolean.FALSE.equals(this.advisedBeans.get(cacheKey))) {
198             return bean;
199         } else if (!this.isInfrastructureClass(bean.getClass()) && !this.shouldSkip(bean.getClass(), beanName)) {
200             Object[] specificInterceptors = this.getAdvicesAndAdvisorsForBean(bean.getClass(), beanName, (TargetSource) null);  specificInterce
201             if (specificInterceptors != DO_NOT_PROXY) {
202                 this.advisedBeans.put(cacheKey, Boolean.TRUE);  cacheKey: "employeeUtil"   advisedBeans:  size = 56
203                 Object proxy = this.createProxy(bean.getClass(), beanName, specificInterceptors, new SingletonTargetSource(bean));  specificInt
204                 this.proxyTypes.put(cacheKey, proxy.getClass());
205                 return proxy;
206             } else {
207                 this.advisedBeans.put(cacheKey, Boolean.FALSE);
208                 return bean;
209             }
210         } else {
211             this.advisedBeans.put(cacheKey, Boolean.FALSE);
212             return bean;
213         }
214     }
215
```

Application startup success

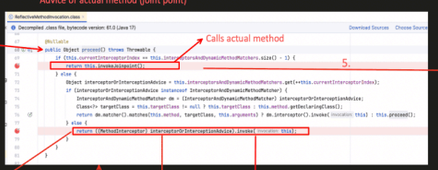
3. Invoke the request, which need Advice to be run.



4. Now, request actually tries to call Proxy Class, so it invokes either CGLIB proxy class or JDK proxy class

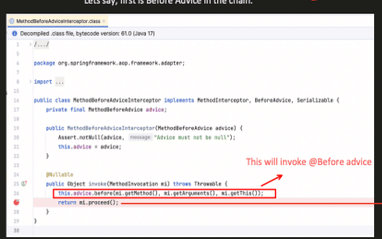


Invokes this, which decide either invoke specific Advice or actual method (joint point)



1.

Lets say, first is Before Advice in the chain.



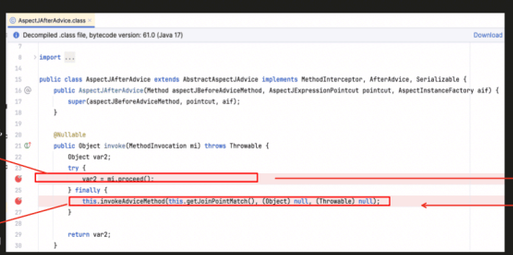
2.

3.

4.

6.

Lets say, second is After Advice in the chain (as counter ++ happens in above method right)



First proceed with other calls, either Before advice or Method execution

Then calls After Advice method