스프링시작하기

스프링 6강 AOP

박명회

1.1 AOP

AOP는 Aspect Oriented Programming의 약자로 여러 객체에 공통으로 적용 할 수 있는 기능을 분리해서 재사용성을 높여주는 프로그래밍 기법이다.

AOP는 핵심 기능과 공통 기능의 구현을 분리함으로써 핵심 기능을 구현한 코드의 수정 없이 공통기능을 적용 할 수 있게 만들어 준다

1.2 AOP 주요 용어

용어	설명
Joinpoint	Advice를 적용가능한 지점을 의미합니다.
Pointcut	Joinpoint의 부분 집합으로, Advice가 적용되는 Joinpoint를 나타냅니다.
Advice	Aspect를 언제 핵심 코드에 적용할 지를 정의합니다.
Weaving	Advice를 핵심 코드에 적용하는 것을 말합니다.
Aspect	여러 객체에 공통으로 적용되는 기능을 말합니다. (공통 기능)

1.3 Advice의 종류

용어	설명
Before Advice	대상 객체의 메서드 호출 전에 공통 기능을 실행합니다.
After Returning Advice	대상 객체의 메서드가 예외 없이 실행된 이후에 공통기능을 실행합니다.
After Throwing Advice	대상 객체의 메서드를 실행하는 도중 예외가 발생한 경우 공통기능을 실행합니
After Advice	대상 객체의 메서드 실행 후 공통 기능을 실행합니다.
Around Advice	대상 객체의 메서드 실행 전 / 후, 예외 발생 시점에 공통 기능을 실행합니다

1.4 @Aspect, @Pointcut, @Around를 이용한 AOP 구현

개발자는 공통 기능을 제공하는 Aspect 구현 클래스를 만들고 자바 설정을 이용해서 Aspect를 어디에 적용할지 설정하면 된다.

```java
@Aspect
public class ExeTimeAspect {
    @Pointcut("execution(public * chap07..*(..))")
    private void publicTarget() {
    }
    @Around("publicTarget()")
    public Object measure(ProceedingJoinPoint joinPoint) throws Throwable {
        long start = System.nanoTime();
        try {
            Object result = joinPoint.proceed();
            return result;
        } finally {
            long finish = System.nanoTime();
            Signature sig = joinPoint.getSignature();
            System.out.printf(format:"%s.%s(%s) 실행 시간 : %d ns\n",
                    joinPoint.getTarget().getClass().getSimpleName(),
                    sig.getName(), Arrays.toString(joinPoint.getArgs()),
                    (finish - start));
        }
    }
```

```java
@Configuration
@EnableAspectJAutoProxy
public class AppCtx {
    @Bean
    public ExeTimeAspect exeTimeAspect() {
        return new ExeTimeAspect();
    }

    @Bean
    public Calculator calculator() {
        return new RecCalculator();
    }

}
```

1.4 인터셉터

```
ngframework.context.annotation.Configuration;
ngframework.web.servlet.config.annotation.InterceptorRegistry;
ngframework.web.servlet.config.annotation.WebMvcConfigurer;


oConfig implements WebMvcConfigurer {


 addInterceptors(InterceptorRegistry registry) {
.addInterceptor(new MyInterceptor());
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

public class MyInterceptor implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse respons
            throws Exception {
        // 요청 처리 전에 실행될 코드 작성
        System.out.println("Pre Handle method is Calling");
        return true; // true를 반환하면 요청 처리를 진행하고, false를 반환하면 요청 처리를
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response,
            ModelAndView modelAndView) throws Exception {
        // 요청 처리 후에 실행될 코드 작성
        System.out.println("Post Handle method is Calling");
    }

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse resp
            throws Exception {
        // 요청 완료 후에 실행될 코드 작성
        System.out.println("Request and Response is completed");
    }
}
```

1.4 필터

```
trationBean<MyFilter> myFilter() {
tionBean<MyFilter> registration = new FilterRegistrationBean<
etFilter(new MyFilter());
ddUrlPatterns("/api/*"); // 필터를 적용할 URL 패턴 지정
ation;
```

```
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import java.io.IOException;

public class MyFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // 필터 초기화 작업
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterCh
            throws IOException, ServletException {
        // 요청 전 처리 작업

        // 다음 필터로 요청 전달 (만약 다음 필터가 없다면 실제 요청을 처리하는 서블릿이 호출
        chain.doFilter(request, response);

        // 응답 후 처리 작업
    }

    @Override
    public void destroy() {
        // 필터 종료 작업
    }
}
```

고맙습니다.