Spring의 AOP, 인터셉터, 필터

CS study 2주차
조은지

Spring AOP

스프링 AOP (Aspect Oriented Programming)

- 관점 지향 프로그래밍, 프록시 기반의 AOP 구현체

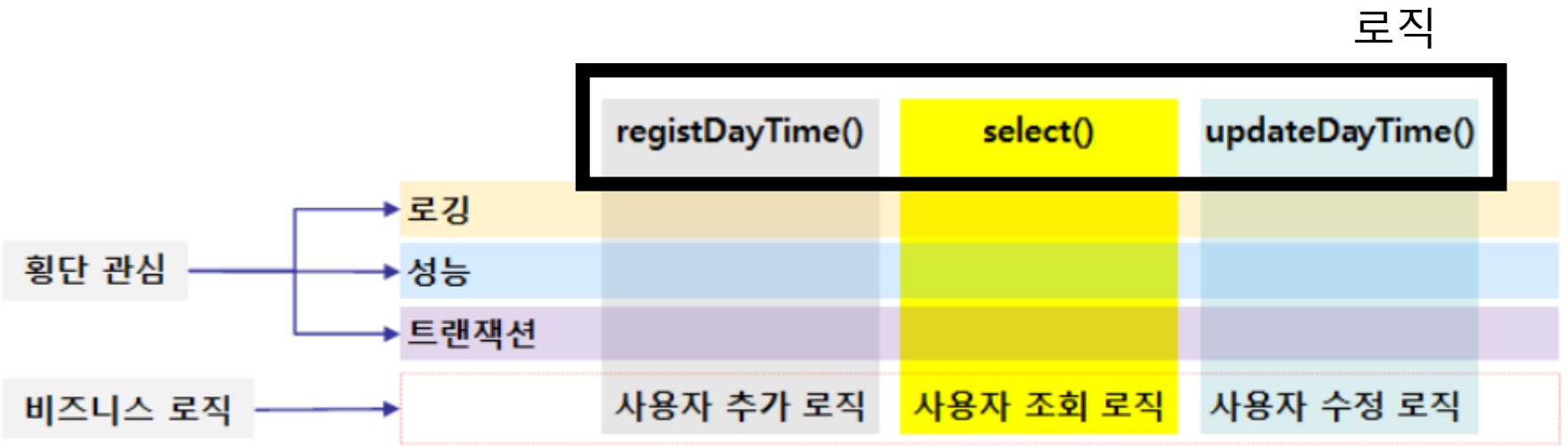
```
@Override
  public int insert(Book book) throws SQLException{
    Connection con=util.getConnection();  리소스 관리 – DB 접속
    int result=-1;
    try {
      result=repo.insert(con, book);  자료 저장 – 비즈니스 로직
    }finally {
      util.close(con);  리소스 관리 – 접속 반환
    }

    return result;
  }
```

Spring에서 각각의 로직들은
1) 비즈니스 로직 2) 부가기능으로 구성됨.

AOP는 이 로직을 기준으로 비즈니스 로직과 부가기능이라는 관점을 나누어 각각을 모듈화하고, 부가기능을 로직에서 분리하여 재사용할 수 있게 만든다는 것!

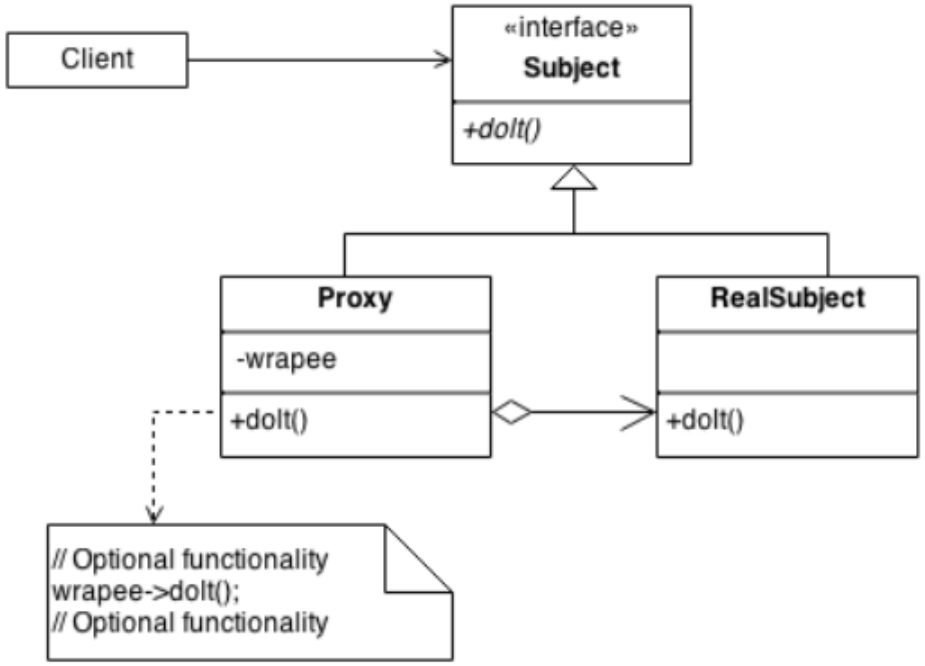
AOP는 웹과 무관하게 비즈니스 로직과 연결되어 사용됨

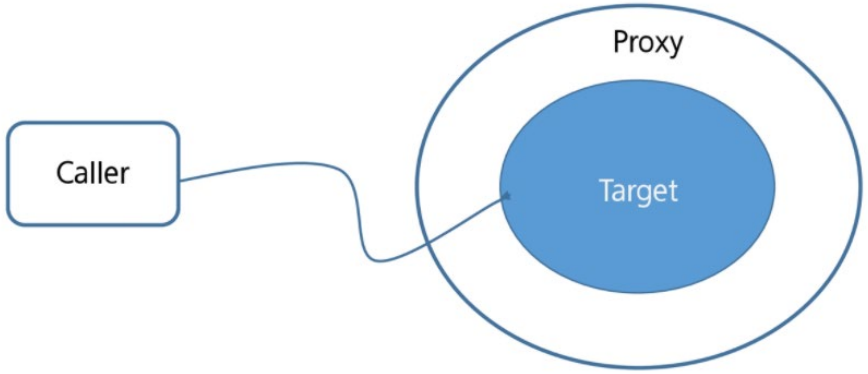
왜 모듈화 할까?



비즈니스 로직(=핵심 로직)을 실행시키 위한 부가적인 기능들은 각각의 로직들에서 계속 반복되어 사용된다.
이러한 부가적인 기능들을 횡단 관심사라고 한다.
횡단 관심사들을 각각의 모듈로 만들어 놓으면 **개발 속도가 빨라지고, 유지보수성이 향상**된다.

Proxy pattern?





- 원래 객체를 감싸고 있는 객체.
- 프록시 객체가 원래 객체를 감싸서, client의 요청을 처리하게 하는 패턴
- 특정한 interface를 노출시키지 않고, 외부로부터 감추고 싶을 때 사용하는 패턴
- 부가적인 기능은 프록시 클래스에 작성, 실제 비즈니스 로직은 원래 서비스 객체에 코딩
- 접근을 제어하고 싶거나, 부가 기능을 추가할 때 원래 코드에 손을 대지 않고 기능을 추가할 수 있음.

Proxy pattern의 장단점

장점

- 사이즈가 큰 객체가 로딩되기 전에 proxy를 통해 참조할 수 있다.
- 실제 객체의 public, protected 메서드들은 숨기고 인터페이스를 통해 노출시킬 수 있다.
- 로컬에 있지 않고 떨어져 있는 객체를 사용할 수 있다. (분산처리)
- 원래 객체의 접근에 대해 사전 처리가 가능하다.

단점

- 객체를 생성할 때 한단계를 거치게 되므로, 빈번한 객체 생성이 필요한 경우 성능이 저하될 수 있다.

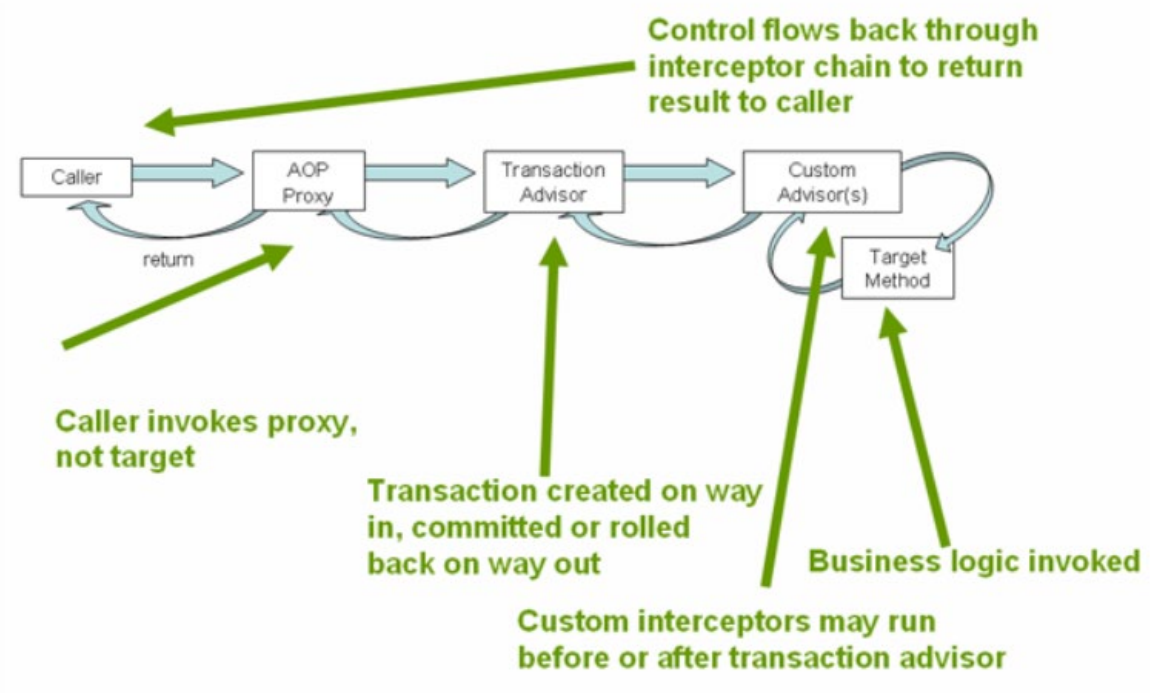
왜 proxy를 이용해 AOP 구현?

- OCP를 지키기 위해
- OCP(개방 폐쇄 원칙) : 소프트웨어 확장에 대해서는 열려 있어야 하고, 수정에 대해서는 닫혀 있어야 하는 원칙
- Proxy 패턴도 인터페이스를 사용해 변화는 허용하지 않게 해주는 패턴 (객체를 외부로부터 지킴)
- OCP는 객체지향 설계 원칙 중 하나이고, JAVA는 객체지향 언어이며 Spring은 객체지향 언어인 JAVA를 기반으로 한 웹 프레임워크이다.

OCP에 대한 더 자세한 정보 https://shxrecord.tistory.com/131

Spring에서의 proxy

- 실제 target의 기능을 대신 수행하면서, 기능을 확장하거나 추가하는 실제 객체
- 메서드의 호출 전/후 요청을 가로채 부가적인 역할을 해주는 객체

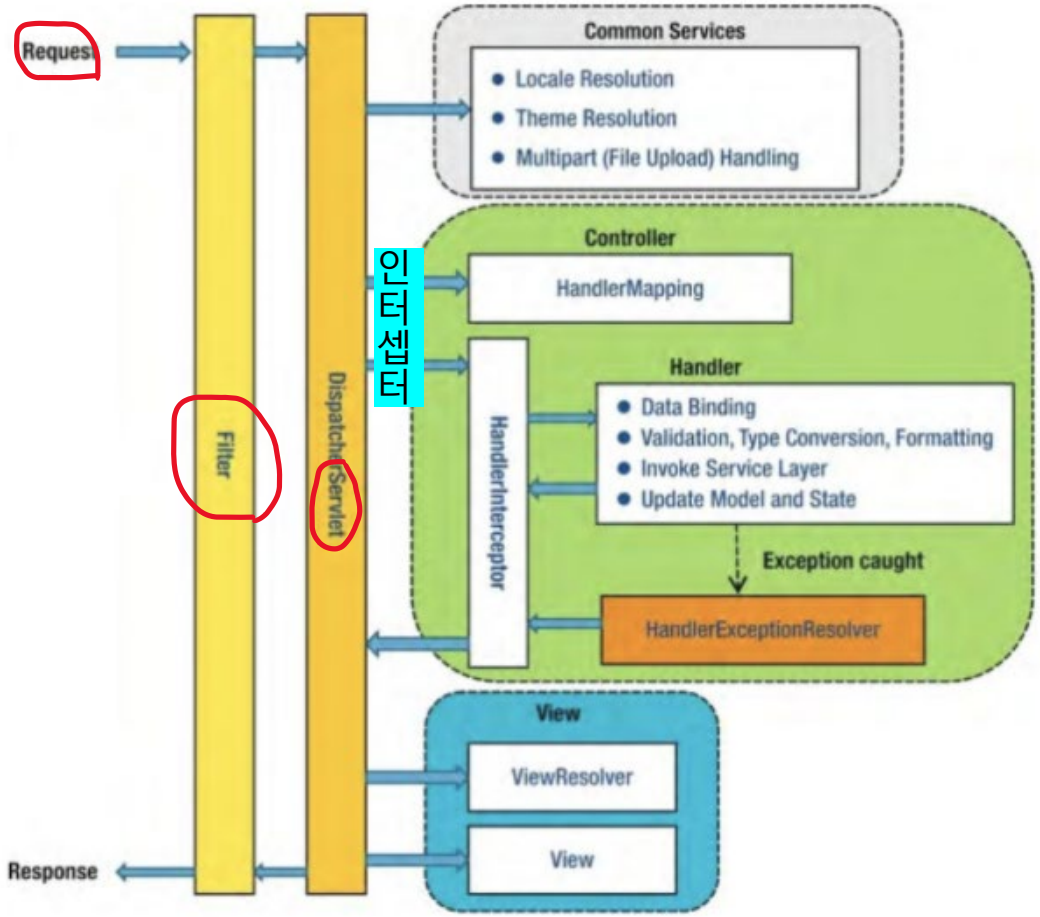


Ex) DB를 사용할 때 사용하는 @Transactional

- DB Transaction에 대한 코딩을 간단하게 해결해주며, 비즈니스 로직에만 집중할 수 있도록 도와줌

필터

필터



- Spring의 독자적인 기능이 아닌 servlet에서 제공하는 기능
- 클라이언트로부터 오는 요청과 servlet 사이에 위치한다.
- 클라이언트의 요청 정보와, 요청 결과를 알맞게 변경할 수 있다.
- servletResponse / servletRequset에 대한 사전/사후 처리

필터의 특징

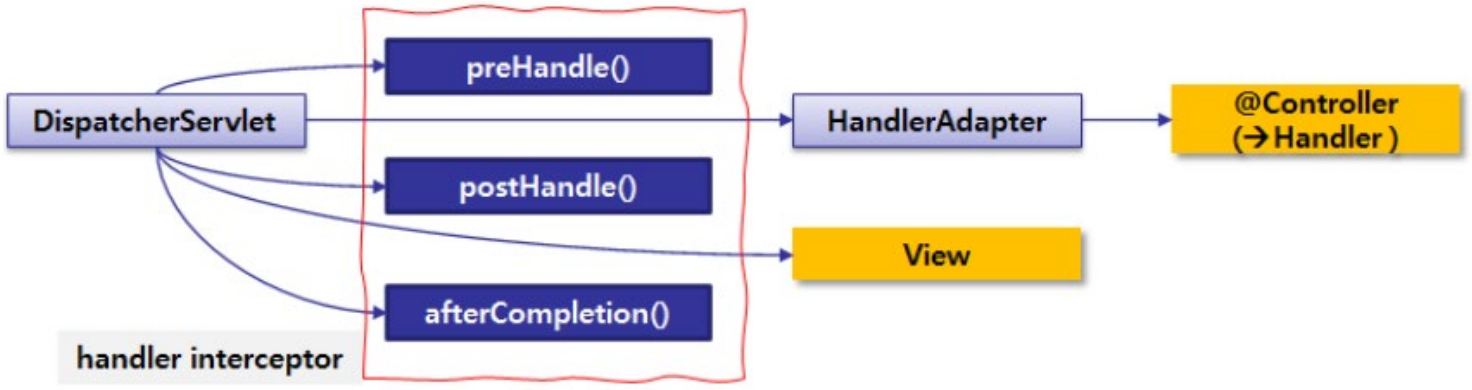
- Filter chain을 통해 여러 필터가 연쇄적으로 동작하게 할 수 있음
- 주로 요청에 대한 인증, 권한 체크 등을 하는데 쓰임
- 들어온 요청이 servlet으로 전달되기 전에 헤더를 검사해 인증 토큰이 있는지 없는지, 올바른지 올바르지 않은지 등을 검사함
- Post로 요청을 받을 때 인코딩 설정

인터셉터

Handler Interceptor

- Client에서 server로 들어온 controller로 가는 요청을 가로채는 것

  여러 컨트롤러에서 공통적으로 사용되는 기능을 정의한다.
  (request에 대한 검사, response header 설정 등)
  - 주 목적은 컨트롤러의 수정 없이 컨트롤러 수행 전/후처리 동작을 추가해 컨트롤러의 반복적인 코드를 제거하기 위함



핸들러 인터셉터의 주요 메서드

- preHandle : handler 동작 전
- postHandle : handler 작 후
- afterCompletion : view의 response까지 완전히 끝난 시점에 동작 (예외가 발생해도 실행)

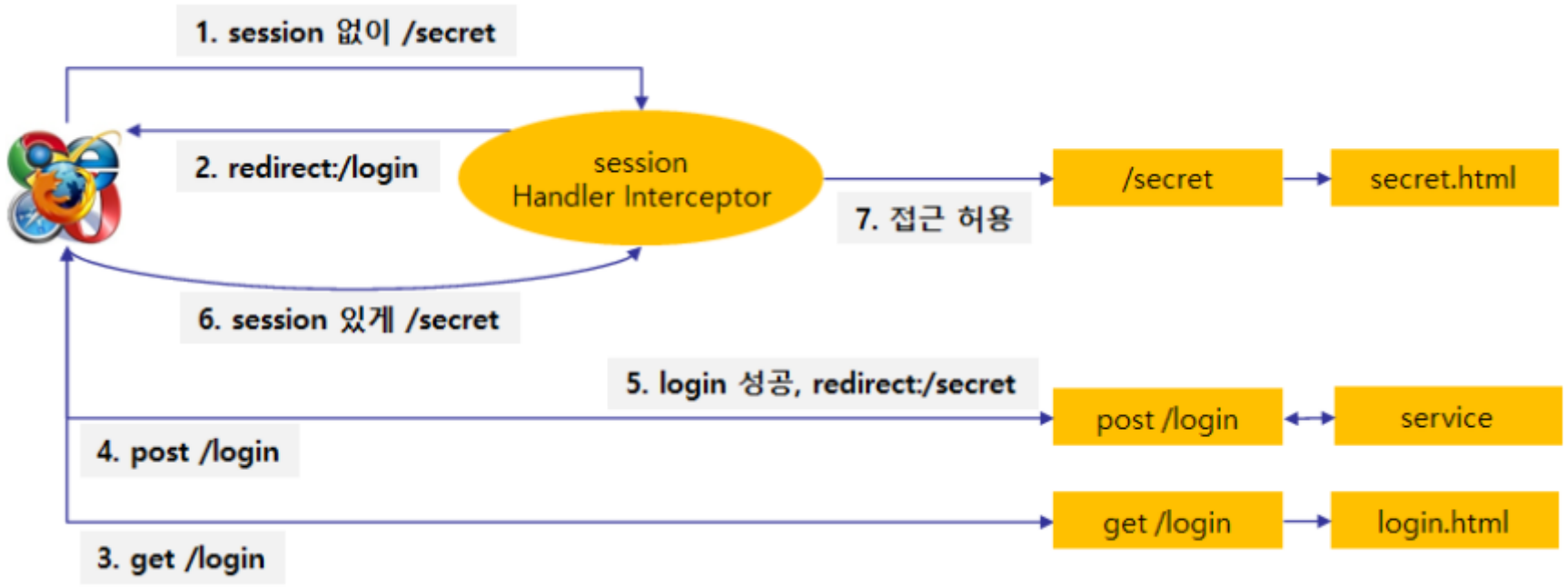
Handler Interceptor

필터와 매우 유사하지만 필터의 기능 중
1) 핸들러 자체의 실행을 금지하는 옵션
2) 후처리 기능
이 두가지만 허용하기 때문에 실제로는 필터가 더 강력한 기능임

인터셉터는 조금 더 세분화된, 컨트롤러의 반복 코드 제거나 권한 확인시에 사용하는 것을 권장함
필터는 모든 요청과 응답에 관한처리, 특정 컨텐츠에 필터를 매핑하는 경우 사용됨

인터셉터 사용 예시

- 특정 웹 페이지에 접근할 때, 이미 로그인한 사용자인지 확인하고 로그인 하지 않은 경우는 로그인 페이지로 유도



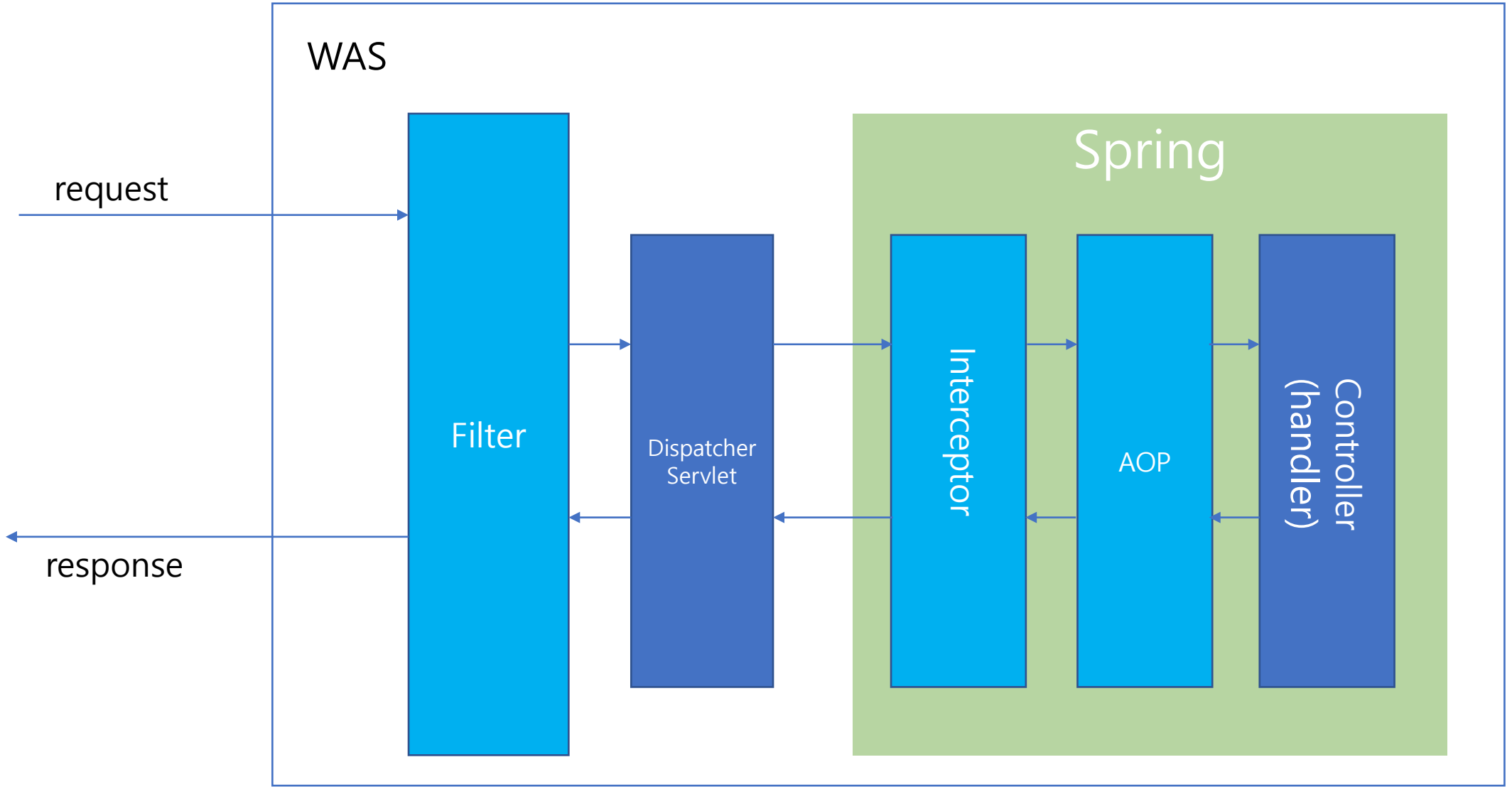
```java
@SuppressWarnings("deprecation")
public class ConfirmInterceptor extends HandlerInterceptorAdapter {

  @Override
  public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object
      throws Exception {
    HttpSession session = request.getSession();
    MemberDto memberDto = (MemberDto) session.getAttribute("userinfo");
    if(memberDto == null) {
      response.sendRedirect(request.getContextPath() + "/user/login");
      return false;
    }
    return true;
  }

}
```

AOP vs Interceptor vs Filter

AOP vs Interceptor vs Filter의 공통점

- 공통된 부분을 모듈화 함
- 어떤 행동을 하기 전 먼저 실행하거나, 실행한 후에 추가적인 행동을 할 때 사용되는 기능들

AOP vs Interceptor vs Filter



AOP vs Interceptor vs Filter

- Filter : spring과 무관한 servlet의 기술, url 기반으로 동작
- Interceptor : spring의 기술, url 기반으로 동작, 빈 객체 접근 o
- AOP : spring의 기술, pointcut 기반으로 동작, 웹과 무관하게 business logic과 연결해서 사용됨, 빈 객체 접근 o

- Pointcut? 어디에 aop를 삽입할지에 대한 내용을 정의한 것

자료 출처

https://sourcemaking.com/design_patterns/proxy
https://sabarada.tistory.com/20
https://goodteacher.tistory.com/