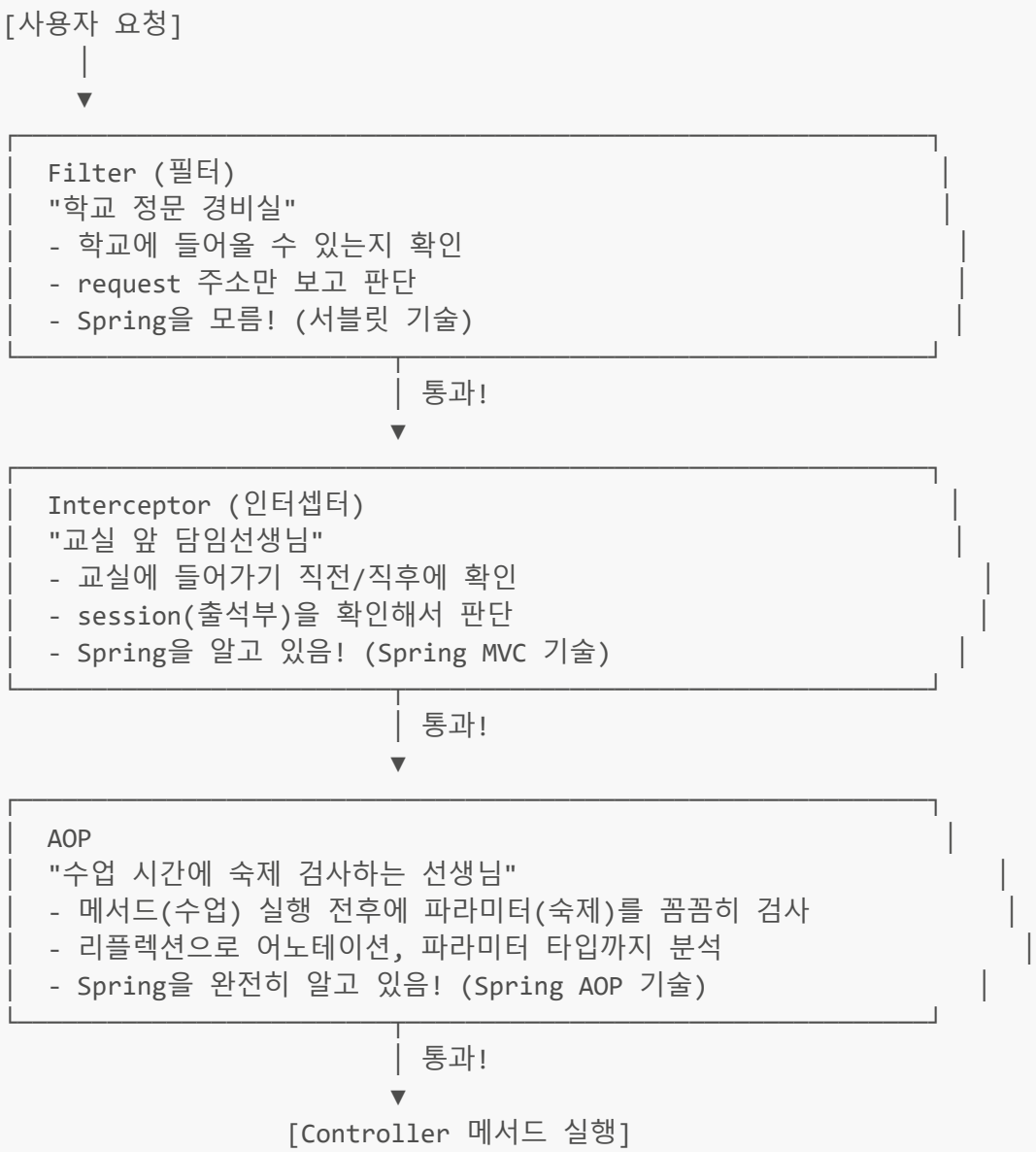
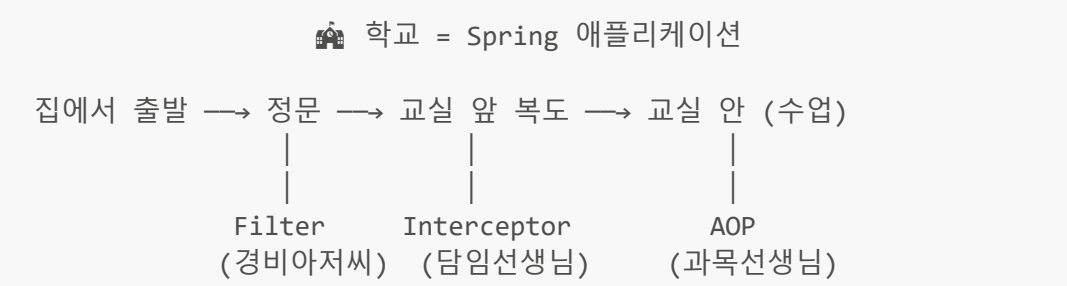
AOP vs Filter vs Interceptor - 스프링의 3가지 관문

한눈에 보기



비유: 학교에 가는 과정



Filter = 정문 경비아저씨

```
👮 "학생증 보여줘!"

- 학교 밖에서 검사 (Spring 밖!)
- 학교 안 상황을 모름
- 학생증(request 주소)만 보고 판단
- "넌 이 학교 학생 아니지? 돌아가!" (차단)
- "학생증 확인! 들어가!" (통과)
```

Interceptor = 교실 앞 담임선생님

```
👩‍🏫 "출석부 확인할게!"

- 교실 문 앞에서 검사 (Controller 직전!)
- 학교 안 상황을 알고 있음 (Spring 알음!)
- 출석부(session)를 확인해서 판단
- "결석 처리된 학생이네? 교무실로!" (차단)
- "출석 확인! 들어와!" (통과)
- 수업 끝나고 나올 때도 확인 가능 (postHandle)
```

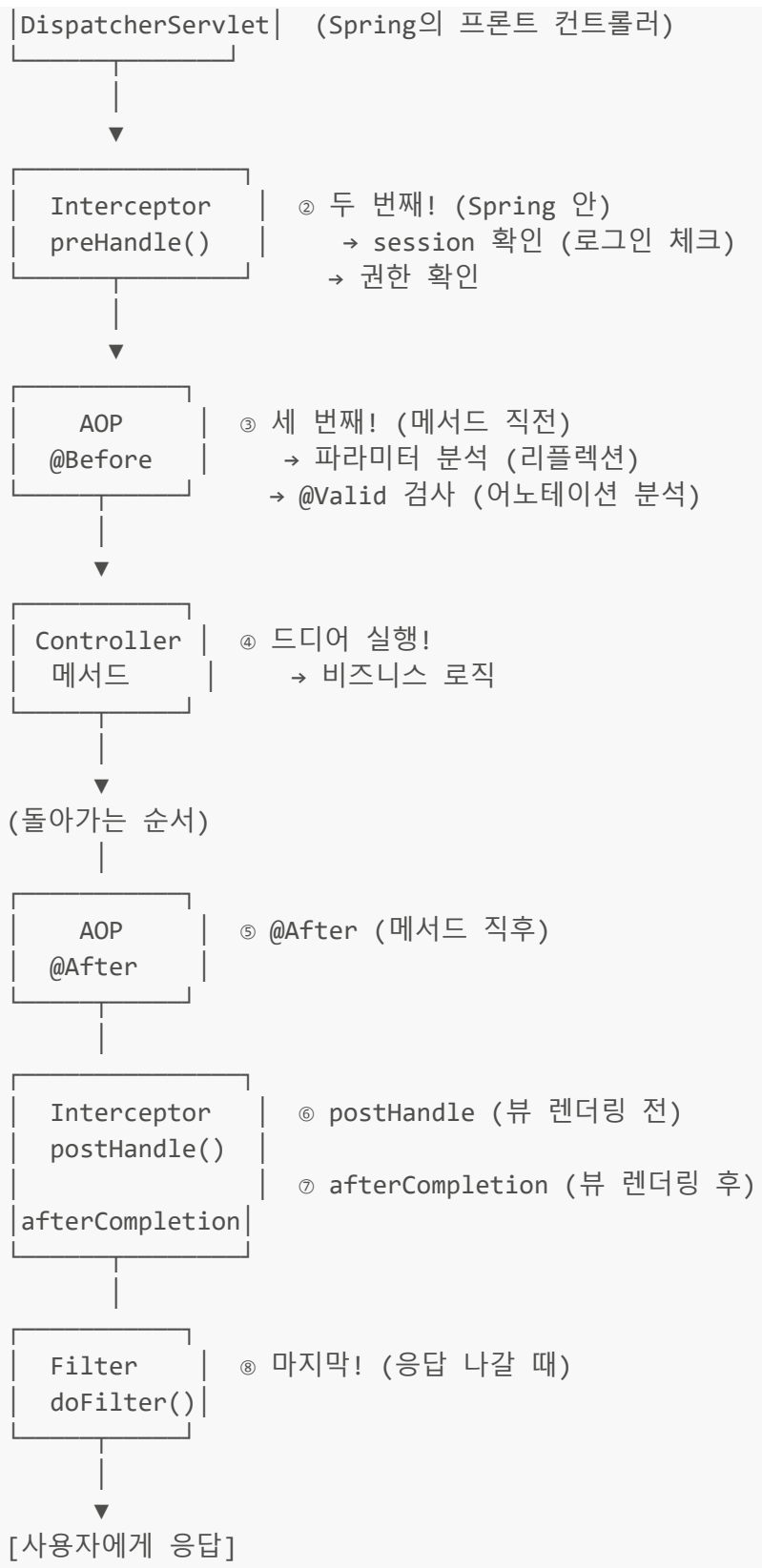
AOP = 수업 시간 과목 선생님

```
📝 "숙제 검사! 가방 열어봐!"

- 교실 안에서 검사 (메서드 실행 전후!)
- 가방(파라미터)을 하나씩 열어봄 (리플렉션!)
- 숙제 종류(@어노테이션)까지 확인
- "수학 숙제 안 했네? 벌점!" (Exception 발생)
- "숙제 다 했구나! 수업 시작!" (통과)
```

실행 순서 다이어그램

```
[사용자의 HTTP 요청]
        │
        ▼
  ┌─────────────┐
  │   Filter    │  ① 가장 먼저! (Spring 밖)
  │  doFilter() │     → request 주소 확인
  └─────────────┘     → 인코딩 설정, CORS 처리
        │
        ▼
  ┌─────────────┐
```


DispatcherServlet (Spring의 프론트 컨트롤러)
Interceptor preHandle()
② 두 번째! (Spring 안)
→ session 확인 (로그인 체크)
→ 권한 확인
AOP @Before
③ 세 번째! (메서드 직전)
→ 파라미터 분석 (리플렉션)
→ @Valid 검사 (어노테이션 분석)
Controller 메서드
④ 드디어 실행!
→ 비즈니스 로직
(돌아가는 순서)
AOP @After
⑤ @After (메서드 직후)
Interceptor postHandle()
afterCompletion
⑥ postHandle (뷰 렌더링 전)
⑦ afterCompletion (뷰 렌더링 후)
Filter doFilter()
⑧ 마지막! (응답 나갈 때)
[사용자에게 응답]

3줄 핵심 비교


Filter (필터)
Interceptor (인터셉터)
AOP

비유	정문 경비실	교실 앞 담임	수업 중 숙제 검사
위치	Spring 밖 (서블릿)	Spring 안 (MVC)	Spring 안 (프록시)
실행 시점	가장 먼저	Controller 전후	메서드 전후
확인할 수 있는 것	request response URL	request response session handler	파라미터 타입 어노테이션 리턴값 메서드 정보
못 하는 것	파라미터 분석 어노테이션 확인	파라미터 분석 어노테이션 확인	request 직접 접근 (가능은 하지만 목적이 다름)
주 용도	인코딩 CORS 보안 필터	로그인 체크 권한 체크 로깅	유효성 검사 로깅 트랜잭션
기술	Servlet API (javax/ jakarta)	Spring MVC (HandlerInter- ceptor)	Spring AOP (@Aspect, @Before)
Spring 필요?	❌ 필요 없음	☑ 필요	☑ 필요

우리 프로젝트에서의 사용 예시

Filter - 사용 안 함 (Spring Boot가 기본 제공)

```
Spring Boot가 알아서 해주는 것들:
- 문자 인코딩 (CharacterEncodingFilter)
- 등등...
→ 우리가 직접 만들 필요 없음!
```

Interceptor - 로그인 체크

```java
// LoginInterceptor.java
public boolean preHandle(HttpServletRequest request, ...) {
    HttpSession session = request.getSession();
    User sessionUser = (User) session.getAttribute("sessionUser");

    if (sessionUser == null) {
        throw new Exception401("인증되지 않았습니다");
    }
    return true;
}
```

```
// WebMvcConfig.java - 어떤 주소에 적용할지 설정
registry.addInterceptor(new LoginInterceptor())
        .addPathPatterns("/boards/**")       // boards 전체 적용
        .addPathPatterns("/replies/**")      // replies 전체 적용
        .excludePathPatterns("/boards/[0-9]+"); // 상세보기는 예외!
```

```
역할: "로그인 안 한 사람은 글쓰기/수정/삭제 못해!"
방법: session에서 사용자 정보 확인
위치: Controller 진입 직전
```

AOP - 유효성 검사 자동화

```
// ValidationHandler.java
@Aspect
@Component
public class ValidationHandler {

    @Before("@annotation(org.springframework.web.bind.annotation.PostMapping)")
    public void validationCheck(JoinPoint jp) {
        Object[] args = jp.getArgs();      // 파라미터 꺼내기 (리플렉션!)
        for (Object arg : args) {
            if (arg instanceof Errors) {   // Errors 타입 찾기 (타입 분석!)
                Errors errors = (Errors) arg;
                if (errors.hasErrors()) {
                    throw new Exception400(
                        errors.getAllErrors().get(0).getDefaultMessage()
                    );
                }
            }
        }
    }
}
```

```
역할: "제목 안 적었네? 다시 작성해!"
방법: 리플렉션으로 파라미터 분석 + @PostMapping 어노테이션 확인
위치: @PostMapping 메서드 실행 직전
```

왜 3개가 다 필요할까?

```
Filter만 쓰면?
→ ❌ session을 편하게 쓸 수 없음
→ ❌ 파라미터 분석 불가능
→ ❌ 어노테이션 확인 불가능

Interceptor만 쓰면?
→ ☑ session 확인 가능
→ ❌ 파라미터 분석 불가능
→ ❌ 어노테이션 확인 불가능

AOP만 쓰면?
→ ☑ 파라미터 분석 가능
→ ☑ 어노테이션 확인 가능
→ ❌ URL 패턴 매칭이 불편
→ ❌ session 접근이 번거로움
```

```
🏫 학교 비유로 정리:

경비아저씨(Filter)만 있으면?
→ 학교 안에서 무슨 일이 일어나는지 모름!

담임선생님(Interceptor)만 있으면?
→ 출석은 체크하지만, 숙제 내용까지 못 봄!

과목선생님(AOP)만 있으면?
→ 숙제는 검사하지만, 출석 체크를 매 수업마다 해야 함!

셋 다 있으면?
→ ☑ 경비아저씨가 정문에서 걸러내고
→ ☑ 담임선생님이 출석 체크하고
→ ☑ 과목선생님이 숙제 검사!
→ 완벽한 분업! 각자 자기 역할에만 집중!
```

한 문장 정리

기술	한 문장
Filter	request 주소를 보고 **입구에서 걸러내는** 정문 경비실
Interceptor	session을 확인해서 **Controller 직전에 막는** 담임선생님
AOP	리플렉션으로 파라미터와 어노테이션을 **분석하는** 과목선생님