Chap 13
MVC 3 : 세션, 인터셉터, 쿠키

로그인 처리 과정

- 로그인 폼을 보여주는 뷰 – login/loginForm.jsp
- 로그인 성공 결과를 보여주는 뷰 – login/loginSuccess.jsp

- 폼에 입력된 값을 전달하는 커맨드 객체 – LoginCommand
- LoginCommand 커맨드 객체의 값 검증과 에러 처리 – LoginCommandValidator

- 이메일과 비밀번호가 일치하는지 확인하고 AuthInfo 객체를 생성 – AuthService
- 암호 일치 여부를 확인 – Member#matchPassword()

- 로그인 성공 후 인증 상태 정보를 세션에 보관함 – AuthInfo
- 로그인 요청을 처리 – LoginController

로그인 상태 유지하기

- 1) HttpSession을 이용하는 방법
- 2) 쿠키를 이용하는 방법

컨트롤러에서 HttpSession 사용하기

- 1) 요청 매핑 애노테이션 적용 메서드에 HttpSession 파라미터를 추가한다.

```
@PostMapping
public String form(LoginCommand loginCommand, Errors errors, HttpSession session) {
```

- 2) 요청 매핑 애노테이션 적용 메서드에 HttpServletRequest 파라미터를 추가하고 HttpServletRequest를 이용해서 HttpSession을 구한다.

```
@PostMapping
public String submit(
        LoginCommand loginCommand, Errors errors, HttpServletRequest req) {
    HttpSession session = req.getSession();
```

LoginController.java

```java
@PostMapping
public String submit(
        LoginCommand loginCommand, Errors errors, HttpSession session,
        HttpServletResponse response) {
    new LoginCommandValidator().validate(loginCommand, errors);
    if (errors.hasErrors()) {
        return "login/loginForm";
    }
    try {
        AuthInfo authInfo = authService.authenticate(
                loginCommand.getEmail(),
                loginCommand.getPassword());

        session.setAttribute("authInfo", authInfo);
```

LogoutController.java

```java
@Controller
public class LogoutController {

    @RequestMapping("/logout")
    public String logout(HttpSession session) {
        session.invalidate();
```

AuthService.java: 이메일과 비밀번호가 일치하는지 확인하고 AuthInfo 객체를 생성

```java
public AuthInfo authenticate(String email, String password) {
    Member member = memberDao.selectByEmail(email);
    if (member == null) {
        throw new WrongIdPasswordException();
    }
    if (!member.matchPassword(password)) {
        throw new WrongIdPasswordException();
    }
    return new AuthInfo(member.getId(),
            member.getEmail(),
            member.getName());
}
```

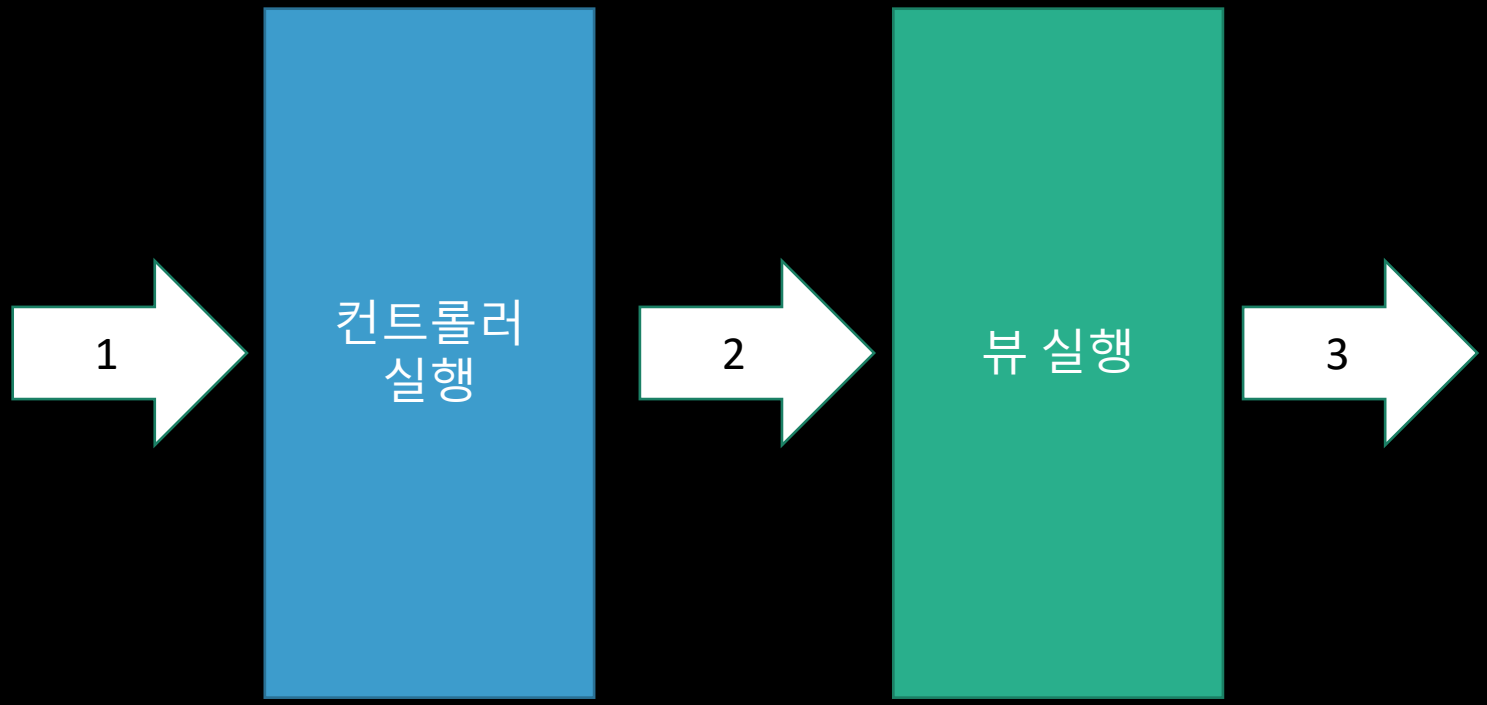
Interceptor

- 로그인하지 않은 상태에서 비밀번호 변경 폼을 요청하면 로그인 화면으로 이동시키려고 한다.
- 이 때 HttpSession에 authInfo 객체가 존재하는지 검사하고 존재하지 않으면 로그인 경로로 리다이렉트한다.
- 로그인 여부를 확인해야하는 경우가 다수 존재한다.

-> 다수의 컨트롤러에 대해 동일한 기능을 적용해야 할 때 인터셉터 사용

HandlerInterceptor 인터페이스

- org.springframework.web.HandlerInterceptor



HandlerInterceptor 인터페이스

```
default boolean preHandle(HttpServletRequest request,
                          HttpServletResponse response,
                          Object handler)
                   throws Exception
```

컨트롤러 실행 전

```
default void postHandle(HttpServletRequest request,
                        HttpServletResponse response,
                        Object handler,
                        @Nullable
                        ModelAndView modelAndView)
                 throws Exception
```

컨트롤러 실행 후

```
default void afterCompletion(HttpServletRequest request,
                             HttpServletResponse response,
                             Object handler,
                             @Nullable
                             Exception ex)
                      throws Exception
```

뷰 렌더링 후

HandlerInterceptor 인터페이스 구현

```java
import org.springframework.web.servlet.HandlerInterceptor;

public class AuthCheckInterceptor implements HandlerInterceptor {

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

}
```

- 로그인 상태가 아니면 컨트롤러 실행 전에 로그인 페이지로 리다이렉트 시킨다.
- preHandle()에서 true를 리턴하면 컨트롤러를 실행하고, false를 리턴하면 리다이렉트된다.

HandlerInterceptor 설정 추가

```java
@Configuration
@EnableWebMvc
public class MvcConfig implements WebMvcConfigurer {
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(authCheckInterceptor())
            .addPathPatterns("/edit/**")
            .excludePathPatterns("/edit/help/**");
    }
```

- WebMvcConfigurer#addInterceptors() 메소드는 인터셉터를 설정하는 메소드이다.
- AuthCheckInterceptor 객체를 인터셉터로 설정하고, 이를 적용할 경로를 지정한다. 이때 Ant 경로 패턴을 사용한다.

AsyncHandlerInterceptor

- HandlerInterceptor를 상속하는 인터페이스
- public interface AsyncHandlerInterceptor extends HandlerInterceptor
- 스프링에서 제공하는 비동기 처리용 인터셉터 인터페이스이다.

```
default void afterConcurrentHandlingStarted(HttpServletRequest request,
                                            HttpServletResponse response,
                                            Object handler)
                                    throws Exception
```

- afterConcurrentHandlingStarted() 메소드는 비동기 처리를 시작하는 시점에 postHandle()이나 afterCompletion() 대신 호출된다.

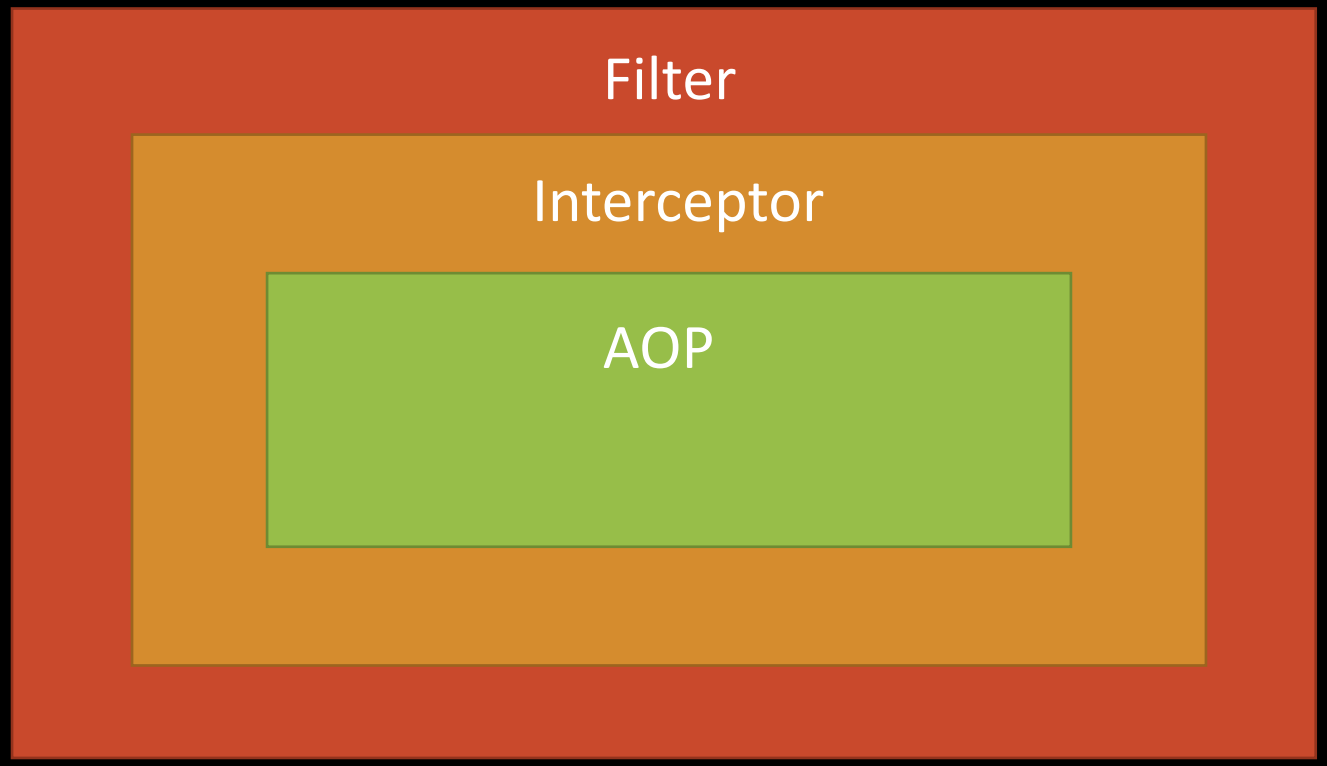
HandlerInterceptorAdapter

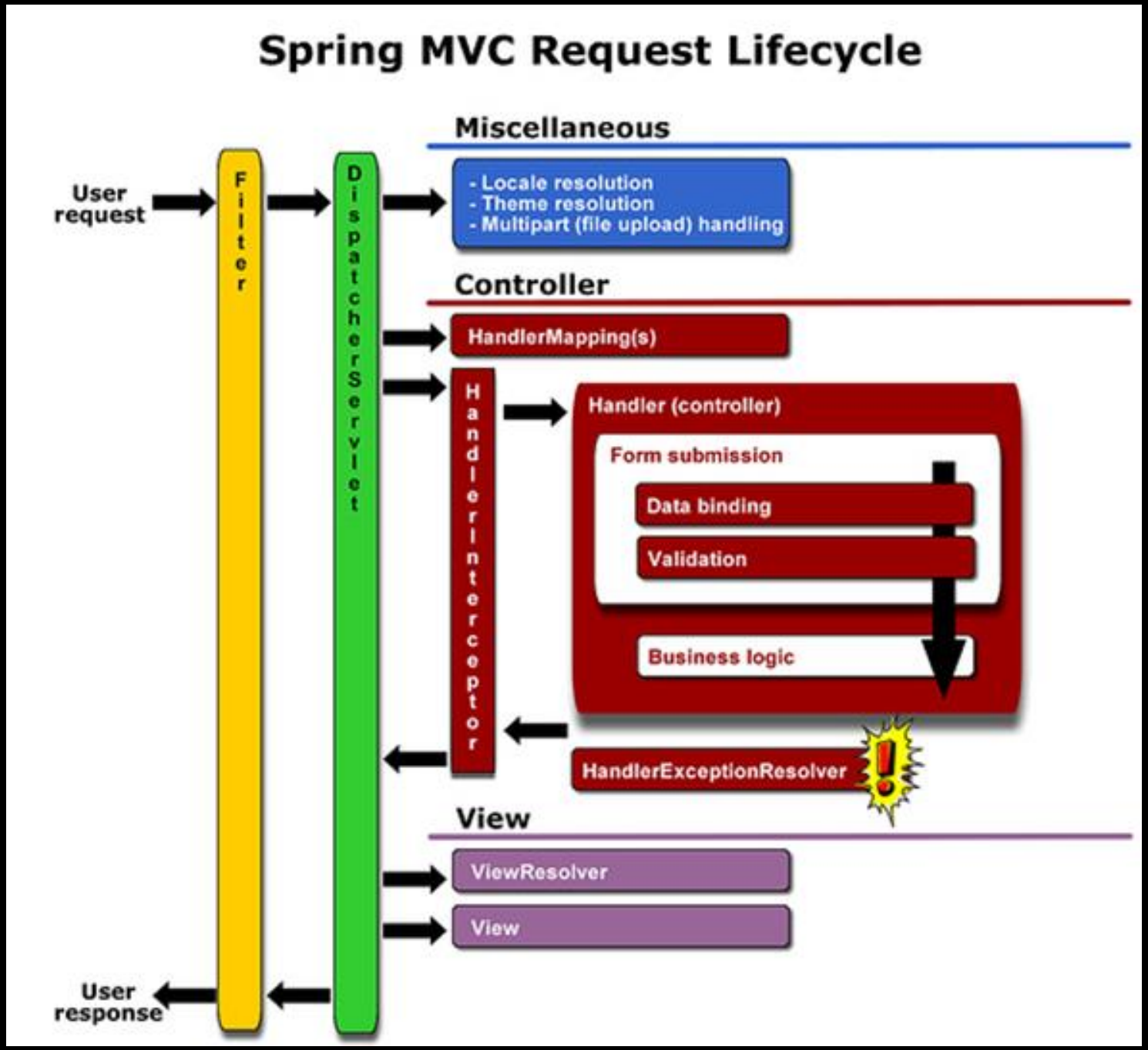
- AsyncHanlderInterceptor를 구현한 추상 어댑터 클래스
- public abstract class HandlerInterceptorAdapter extends Object implements AsyncHandlerInterceptor

- Adapter: 특정 인터페이스를 미리 구현해서 사용하기 쉽게 하는 용도

Filter vs Interceptor vs AOP

- 공통점: 공통 관심사를 해결하기 위한 목적을 가진다.




Spring MVC Request Lifecycle
Miscellaneous
User request
Filter
DispatcherServlet
- Locale resolution
- Theme resolution
- Multipart (file upload) handling
Controller
HandlerMapping(s)
HandlerInterceptor
Handler (controller)
Form submission
Data binding
Validation
Business logic
HandlerExceptionResolver
View
ViewResolver
View
User response

Filter

- Request와 Response를 거르고 정제하는 역할

- web.xml에 등록하여 사용한다.
- Servlet Filter는 DispatcherServlet 이전에 실행되어 Request 내용을 변경하거나 확인할 수 있다.
- 또한, 자원의 처리가 끝난 후 Response 내용을 변경할 수 있다.

- 인코딩, 압축, 변환 등

Interceptor

- 컨트롤러에 관한 요청을 처리하는 역할

- DispatcherServlet이 컨트롤러를 호출하기 전, 후에 동작할 수 있다.
- 스프링 컨텍스트 내부에 위치하기 때문에 스프링의 모든 빈 객체에 접근할 수 있다.
- 매개변수로 HttpServletRequest, HttpServletResponse를 가진다.

- 로그인 체크, 권한 체크, 쿠키 등

AOP

- Aspect-oriented programming, 관점지향 프로그래밍

- OOP를 보완하는 개념이다.
- 비즈니스단의 공통 관심사를 처리한다.

- 로깅, 트랜잭션, 에러 처리 등

Filter vs Interceptor vs AOP

- Filter와 Interceptor: 실행 시점에 속하는 영역이 다르다.

- 필터는 스프링 컨텍스트 바깥, 웹 어플리케이션에 위치하며 스프링과 무관하게 적용할 수 있지만, 인터셉터는 스프링 컨텍스트 내부에서 DispatcherServlet 뒤의 Handler 영역에서 적용된다.

Filter vs Interceptor vs AOP

- Interceptor와 AOP: 매개변수가 다르다.

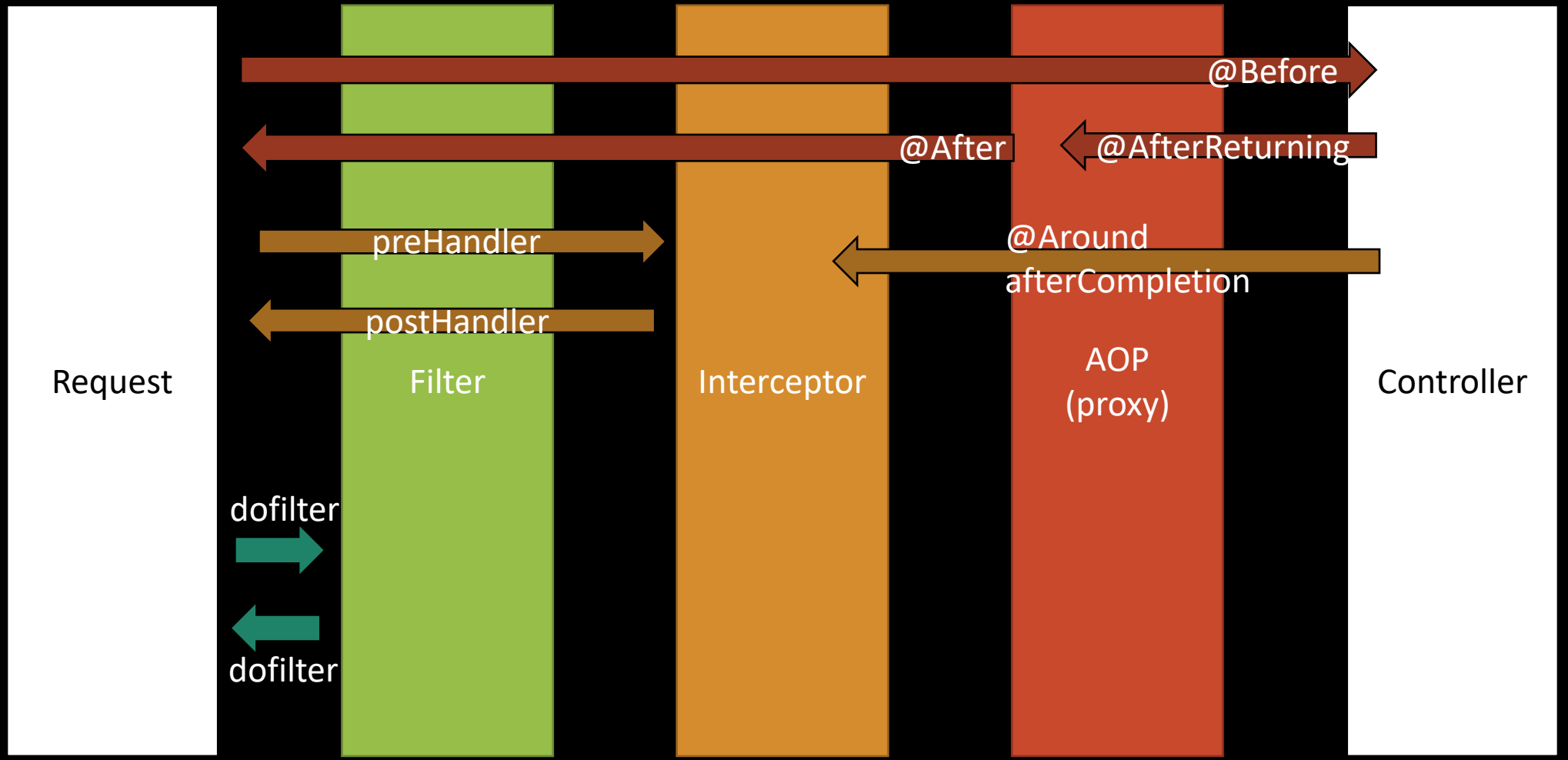
- AOP Advice는 JoinPoint나 ProceedingJoinPoint 등을 활용하여 호출하지만, 인터셉터는 HttpServletRequest, HttpServletResponse를 매개변수로 사용한다.
- 따라서 인터셉터가 웹에 보다 특화되어 있으며 일반적으로 컨트롤러에서 AOP Advice보다 인터셉터를 활용한다.

Filter vs Interceptor vs AOP

- Filter, Interceptor와 AOP: AOP는 인터셉터와 필터보다 실행 시점을 자유롭게 설정할 수 있다.

- Before Advice, Around Advice, After Advice, After Returning Advice, After Throwing Advice

Filter vs Interceptor vs AOP



컨트롤러에서 쿠키 사용하기

- @CookieValue 애노테이션을 요청 매핑 애노테이션 적용 메소드의 Cookie 타입 파라미터에 적용하여 쿠키를 사용할 수 있다.
- 쿠키가 존재하면 폼의 email에 쿠키 값이 채워진다.

```
@GetMapping
public String form(LoginCommand loginCommand,
        @CookieValue(value = "REMEMBER", required = false) Cookie rCookie) {
    if (rCookie != null) {
        loginCommand.setEmail(rCookie.getValue());
        loginCommand.setRememberEmail(true);
    }
    return "login/loginForm";
}
```

커맨드 객체의 setter 메소드를 이용하여 쿠키의 값이 들어간다.

컨트롤러에서 쿠키 생성하기

- 로그인을 처리할 때 '이메일 기억하기' 체크박스를 선택했을 경우 쿠키를 생성하도록 구현한다.

```
@PostMapping
public String submit(
        LoginCommand loginCommand, Errors errors, HttpSession session,
        HttpServletResponse response) {
```

쿠키를 생성하려면 HttpServletResponse 객체가 필요하다.

```
Cookie rememberCookie =
        new Cookie("REMEMBER", loginCommand.getEmail());
rememberCookie.setPath("/");
if (loginCommand.isRememberEmail()) {
    rememberCookie.setMaxAge(60 * 60 * 24 * 30);
} else {
    rememberCookie.setMaxAge(0);
}
response.addCookie(rememberCookie);
```

이메일 기억하기를 선택했다면 30일동안 유지되는 쿠키를 생성한다.