Interceptor

Interceptor

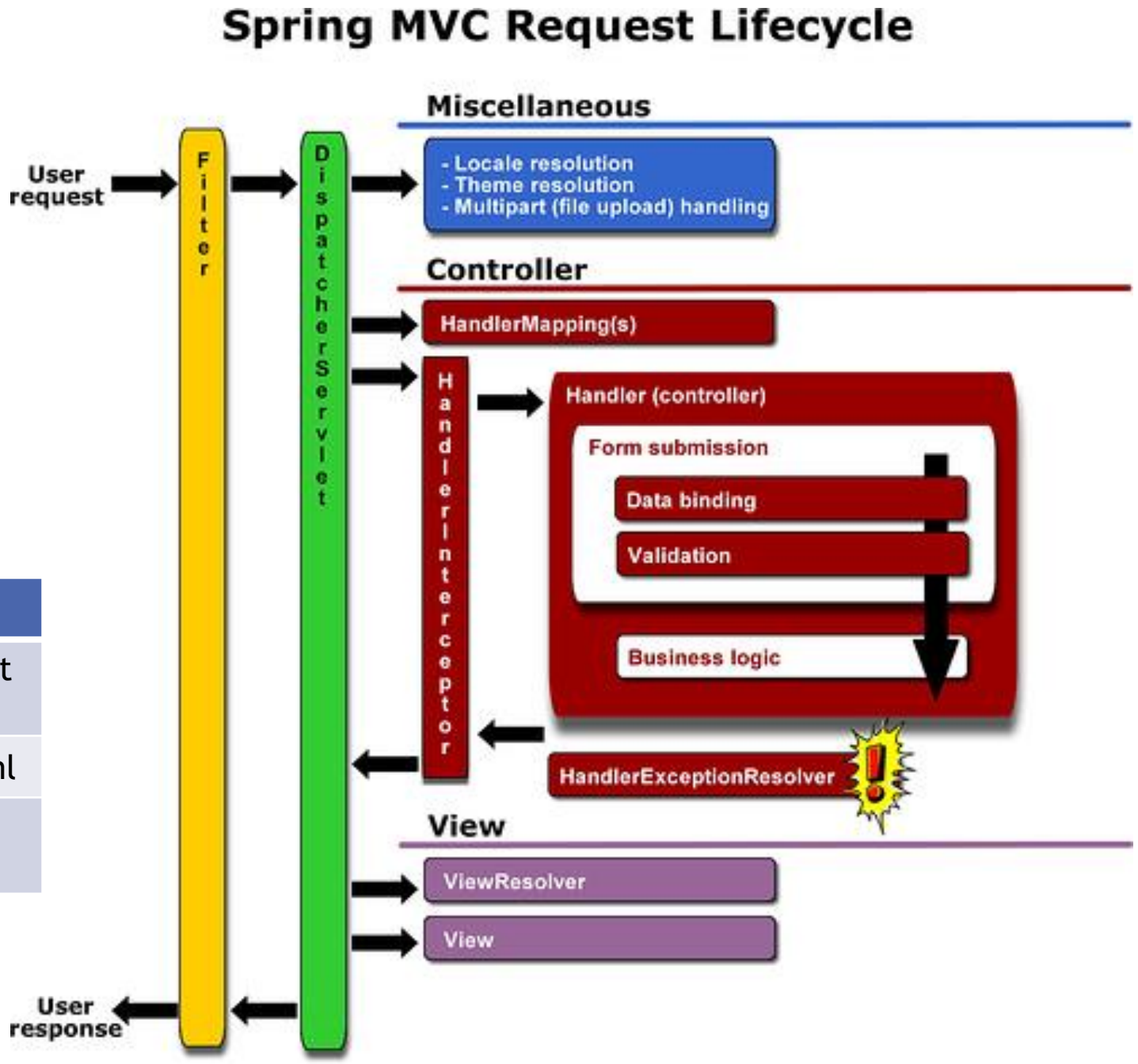
- 인터셉터란?
  - Spring에서 HTTP Request와 HTTP Response를 Controller 앞과 뒤에서 가로채는 역할을 수행
  - Servlet의 앞과 뒤에서 HTTP Request와 HTTP Response를 가로채는 필터와 유사함
  - Interceptor를 구현하기 위해서는 HandleInterceptor 인터페이스를 구현해야 한다

Interceptor

- 서블릿 필터와의 차이

1. 호출되는 시점
2. 설정 위치
3. 구현방식

	필터	인터셉터
호출 시점	DispatcherServlet 실행 이전	DispatcherServlet 실행 이후
설정 위치	web.xml	spring-servlet.xml
구현 방식	web.xml에 설정	설정 후 메서드 구현 필요



Interceptor
: 구현 예제

- Custom Interceptor 구현

  HandlerInterceptor 인터페이스의 3개의 메서드를 구현한다

```
public interface HandlerInterceptor {
    boolean preHandle( HttpServletRequest request, HttpServletResponse response,
        Object handler) throws Exception;

    void postHandle( HttpServletRequest request, HttpServletResponse response,
        Object handler, ModelAndView modelAndView) throws Exception;

    void afterCompletion( HttpServletRequest request, HttpServletResponse response,
        Object handler, Exception ex) throws Exception;
}
```

1. preHandle() 메서드는 컨트롤러가 호출되기 전에 실행된다. handler 파라미터는 HandlerMapping이 찾아준 컨트롤러의 메서드를 참조할 수 있는 HandlerMethod 오브젝트이다
   반환값이 true이면 핸들러의 다음 체인이 실행되지만 false이면 중단하고 남은 Interceptor와 컨트롤러가 실행되지 않는다
2. postHandler() 메서드는 컨트롤러가 실행된 후에 호출된다
3. afterCompletion()은 뷰에서 최종 결과가 생성되는 일을 포함한 모든 일이 완료되었을 때 실행된다

Interceptor
: 구현 예제

- Custom Interceptor 등록 (in spring-servlet.xml)

```
<!-- Interceptors -->
<mvc:interceptors>
    <mvc:interceptor>
    <mvc:mapping path="/board/**" />
    <bean class="com.example.mysite.interceptor.MyInterceptor" />
    </mvc:interceptor>
</mvc:interceptors>
```

  - 핸들러 인터셉터는 하나 이상 등록할 수 있으며 등록 순서대로 preHandle() 이 실행된다
  - postHandle()과 afterCompletion()은 그 반대로 실행된다

- 테스트
  1. mapping path 변경해 보기
  2. mapping path 여러 개 추가해 보기
  3. MyInterceptor의 preHandle 리턴 값 변경해보기

Interceptor
: 구현 예제

- HandlerInterceptorAdapter 추상 클래스를 상속하여 구현할 수 있다
  - HandlerInterceptorAdapter는 HandlerInterceptor 인터페이스 기본 메서드를 구현하여 놓았기 때문에 상속한 인터셉터가 필요한 메서드만 오버라이딩하여 다시 구현하면 된다

- HandlerInterceptorAdapter를 상속하여 MyInterceptor2를 작성한 후, 설정하고 로그를 확인해 봅시다

AuthInterceptor 구현하기

- 인증 여부에 따른 특정 URL 접근 제한
- /user/login URL 처리
  - Interceptor에서는 ApplicationContext를 구해서 직접 저장된 Bean을 가져와야 한다

```
ApplicationContext applicationContext =
    WebApplicationContextUtils.getWebApplicationContext( request.getServletContext() );
UserService userService = applicationContext.getBean( UserService.class );
```

- /user/logout URL 처리

Annotation 활용

- Annotation
  - Java 5부터 지원
  - Class, Method, Parameter에 메타 정보를 삽입한다
  - 컴파일할 때, 혹은 실행할 때 해당 타깃에 코드를 추가적으로 실행할 수 있게 한다
  - 코드의 가독성을 향상시키고 체계적인 코드를 구성할 수 있다

```
@Auth
@RequestMapping( "/modifyform" )
public String modifyForm( @AuthUser UserVo authUser, Model model ) {

}
```

Annotation 활용

- Annotation 정의

```
@Target( ElemenType.METHOD )
@Retention( RetentionPolicy.RUNTIME )
public @interface Auth {

}
```

  - @Target : 어노테이션의 타겟을 지정 (FIELD, METHOD, PARAMETER, TYPE)
  - @Retention : 어노테이션의 지속(보존) 기간을 지정 (RUNTIME, SOURCE)

Annotation 활용

- 인터셉터의 PATH mapping을 Handler 어노테이션으로 대체하는 예
  1. 접근 제한에 해당하는 URL로 매핑된 메서드에 @Auth를 추가
  2. AuthInterceptor에 다음 코드를 쿠가

```
Auth auth = ( ( HandlerMethod ) handler ).getMethodAnnotation( Auth.class );
if( auth == null ) {
    return true;
}
```

  3. Interceptor 설정을 변경

```
<mvc:interceptor>
    <mvc:mapping path="/**" />
    <mvc:exclude-mapping path="/user/login" />
    <mvc:exclude-mapping path="/assets/**" />
    <bean class="com.example.mysite.interceptor.AuthInterceptor" />
</mvc:interceptor>
```

Annotation 활용

- @AuthUser 적용하기
  1. 어노테이션 정의

```
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
public @interface AuthUser {
}
```

Annotation 활용

- @AuthUser 적용하기

  2. HandlerMethodArgumentResolver 인터페이스 구현

```
public class AuthUserHandlerMethodArgumentResolver
     implements HandlerMethodArgumentResolver{

     @Override
     public Object resolveArgument(MethodParameter parameter,
          ModelAndViewContainer mavContainer,
          NativeWebRequest webRequest,
          WebDataBinderFactory binderFactory) throws Exception {
          if( this.supportsParameter(parameter) == false ) {
               return WebArgumentResolver.UNRESOLVED;
          }

          HttpServletRequest httpServletRequest =
                         webRequest. getNativeRequest( HttpServletRequest.class );
          HttpSession session = httpServletRequest.getSession();

          UserVo authUser = (UserVo) session.getAttribute( "authUser" );
          return authUser;
     }

     //   ...
```

Annotation 활용

- @AuthUser 적용하기

  2. HandlerMethodArgumentResolver 인터페이스 구현

```
    //   ...

    @Override
    public boolean supportsParameter(MethodParameter parameter) {
        return parameter.getParameterAnnotation( AuthUser.class ) != null &&
               parameter.getParameterType().equals( UserVo.class );
    }
}
```

Annotation 활용

- @AuthUser 적용하기

  3. AuthUserHandlerMethodArgumentResolver 등록

```
<mvc:annotation-driven>
    <!-- argument resolver -->
    <mvc:argument-resolvers>
        <bean class="com.example.mysite.annotation.AuthUserHandlerMethodArgumentResolver"/>
    </mvc:argument-resolvers>
</mvc:annotation-driven>
```