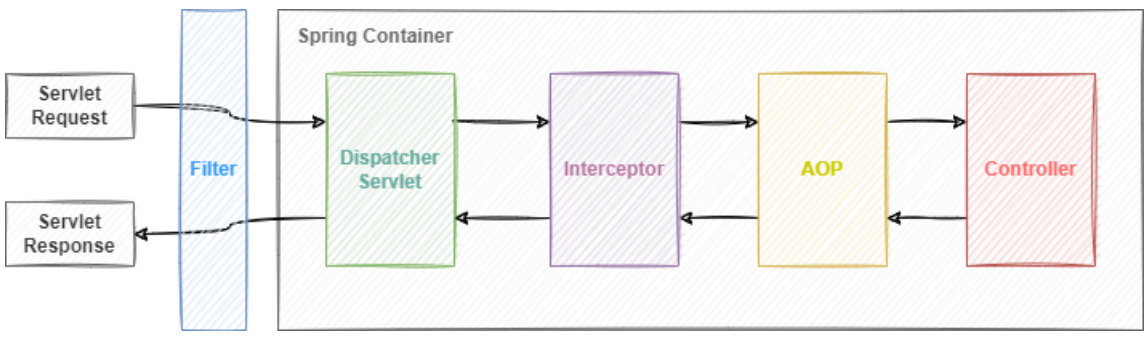
공통관심사(Cross-cutting Concerns)

이전에 스프링의 특징으로 AOP 에 대해서 설명한 적이 있습니다. 객체 지향 프로그래밍(Object Oriented Programming, OOP) 을 하다보면 단일 책임 원칙(Single Responsibility Principle, SRP) 에 따라 클래스를 분리하게 되지만, 클래스를 설계하다보면 공통적인 부분이 존재하게 됩니다. 예를 로그인 기능을 구현한다면 컨트롤러마다 로그인 기능을 작성하게 됩니다. 이렇게 중복되는 코드를 공통 관심사(Cross-cutting Concerns) 라고 합니다.

스프링에는 이러한 공통 관심사를 처리하기 위한 AOP 를 제공하지만, 웹어플리케이션의 경우 AOP 보다는 Filter 또는 Interceptor 에서 공통 관심사를 처리합니다.



3개의 인터페이스는 공통관심사를 처리한다는 점에서 같은 역할을 하지만, 약간의 차이점이 존재합니다.

Filter 는 Servlet Container 에 존재하며 DispatcherServlet 의 호출 전후로 실행됩니다.

Interceptor 는 Spring Container 에 존재하며, Controller 의 호출 전후로 실행됩니다.

AOP 는 Interceptor와 마찬가지로 Spring Container 에서 실행되지만 Servlet 에서 실행되는 Filter, Interceptor와 달리 AOP는 Proxy 를 통해 실행됩니다.

정리하면 클라이언트 요청이 들어오면 Filter -> Interceptor -> AOP -> Interceptor -> Filter 순서로 실행됩니다.

서블릿 필터(Servlet Filter)

필터는 Filter 인터페이스를 상속받아 구현하고 web.xml 파일에 등록하여 적용할 수 있습니다.

Filter 인터페이스는 다음과 같습니다.

javax.servlet.Filter

```
public interface Filter {

    public void init(FilterConfig filterConfig) throws ServletException;


    public void doFilter ( ServletRequest request, ServletResponse response, FilterChain chain ) throws IOException, ServletException;


    public void destroy();
}
```

`init` 는 필터를 초기화하는 메소드 입니다. 서블릿 컨테이너가 생성될 때 호출됩니다. `FilterConfig` 객체를 통해 초기 파라미터를 설정할 수 있습니다.

`doFilter` 는 클라이언트의 요청마다 호출됩니다. ServletRequest와 ServletResponse를 매개변수로 받으며, HTTP 요청에 처리가 필요하다면 `HttpServletRequest` 와 `HttpServletResponse` 로 캐스팅하여 처리하면 됩니다.

`destroy` 는 서블릿 컨테이너가 종료될 때 호출됩니다.

로그인 필터를 다음과 같이 구현해보겠습니다.

/kro/rubisco/filter/LoginCheckFilter.java

```
package kro.rubisco.filter;

import java.io.IOException;

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

public class LoginCheckFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {}

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        HttpSession session = httpRequest.getSession(false);

        if(session == null || session.getAttribute("member") == null) {
            httpResponse.sendRedirect("/login");
            return;
        }

        chain.doFilter(request, response);

}

@Override
```

```
        public void destroy() {}
    }
```

클라이언트 요청(request)에서 세션을 조회하여 member 객체가 있는지 확인하고 없다면 HttpServletResponse 객체의 sendRedirect 메소드를 통해 /login 경로로 리다이렉트 하도록 응답(response)을 보냅니다. 인증된 상태라면 세션에 member 객체가 있어서 필터를 통과하게 될 것이고, FilterChain 객체의 doFilter 메소드를 통해 다음 필터를 실행할 수 있도록 합니다.

구현한 로그인 필터를 web.xml 파일에 등록하도록 하겠습니다. 이전에 컨트롤러를 작성할 때, 인코딩과 관련하여 필터를 등록한 적이 있습니다.

src/main/webapp/WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee https://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

        <!-- The definition of the Root Spring Container shared by all Servlets and Filters -->
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/root-context.xml</param-value>
</context-param>

<!-- Creates the Spring Container shared by all Servlets and Filters -->
<listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<!-- Processes application requests -->
<servlet>
        <servlet-name>appServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
        <servlet-name>appServlet</servlet-name>
        <url-pattern>/</url-pattern>
</servlet-mapping>

<filter>
        <filter-name>encodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
        <init-param>
                <param-name>encoding</param-name>
                <param-value>UTF-8</param-value>
        </init-param>
</filter>
```

```
<filter>
        <filter-name>httpMethodFilter</filter-name>
        <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
</filter>
<filter>
        <filter-name>loginCheckFilter</filter-name>
        <filter-class>kro.rubisco.filter.LoginCheckFilter</filter-class>
</filter>
<filter-mapping>
        <filter-name>encodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
        <filter-name>httpMethodFilter</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
        <filter-mapping>
                <filter-name>loginCheckFilter</filter-name>
                <url-pattern>/board/*</url-pattern>
        </filter-mapping>
    </web-app>
```

`loginCheckFilter` 를 추가하고 `/board/*` 패턴에 매핑했습니다.

`localhost:8080/board` 에 접속하면 필터를 통해 로그인 페이지로 이동하는 것을 확인할 수 있습니다. 로그인을 한 후 다시 /board에 접속하면 정상적으로 게시판이 출력됩니다.

스프링 인터셉터(Spring Interceptor)

이번에는 인터셉터를 통해 로그인 인증을 해보겠습니다. `web.xml` 에 등록한 필터를 지워서 원래 상태로 되돌려주세요.

`Interceptor` 인터페이스는 다음과 같습니다.

org.springframework.web.servlet.HandlerInterceptor

```
public interface HandlerInterceptor {

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


            return true;
        }

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

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

서블릿 필터와 달리 스프링 인터셉터는 호출시점이 3개로 분리되어 있습니다.

`preHandle` 메소드는 컨트롤러 호출전에 호출되고, `postHandle` 메소드는 컨트롤러 호출후에 호출됩니다.

`afterCompletion` 메소드는 뷰가 렌더링 된 이후에 호출됩니다.

이제 `HandlerInterceptor` 인터페이스를 상속받은 `LoginInterceptor` 를 구현해 보겠습니다.

/kro/rubisco/interceptor/LoginInterceptor.java

```
package kro.rubisco.interceptor;

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

import org.springframework.stereotype.Component;
import org.springframework.web.servlet.HandlerInterceptor;

@Component
public class LoginInterceptor implements HandlerInterceptor {

    @Override
    public boolean preHandle(
        HttpServletRequest request,
        HttpServletResponse response,
        Object handler
    ) throws Exception {

        HttpSession session = request.getSession(false);

        if(session == null || session.getAttribute("member") == null) {
            response.sendRedirect("/login");
            return false;
        }

        return true;
    }
}
```

`@Component` 어노테이션을 통해 빈으로 등록될 수 있도록 하고, `preHandle` 메소드를 Override하여 로그인 인증을 구현합니다. 필터와 동일하게 클라이언트의 요청으로부터 세션을 조회하여 member 객체가 있는지 확인한 후 없다면 `HttpServletResponse` 객체의 `sendRedirect` 메소드를 통해 `/login` 경로로 리다이렉트 하도록 응답(response)을 작성합니다. 또한 `false` 를 리턴하여 컨트롤러를 호출하지 않고 클라이언트로 응답을 보내어 `/login` 경로로 리다이렉트되도록 합니다.

이제 `servlet-context.xml` 파일에 interceptor 패키지를 스캔할 수 있도록 등록하고, `@Component` 어노테이션을 통해 등록된 `loginInterceptor` 객체를 `/board/**` 경로에 매핑해주도록 합니다.

/src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:beans="http://www.springframework.org/schema/beans"
	xmlns:context="http://www.springframework.org/schema/context"
	xsi:schemaLocation="http://www.springframework.org/schema/mvc https://www.springframework.org/schema/mvc/spring-mvc.xsd
		http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd">

	<!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->

	<!-- Enables the Spring MVC @Controller programming model -->
	<annotation-driven />

	<!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources in the ${webappRoot}/resources directory -->
	<resources mapping="/resources/**" location="/resources/" />

	<!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->
	<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
		<beans:property name="prefix" value="/WEB-INF/views/" />
		<beans:property name="suffix" value=".jsp" />
	</beans:bean>

	<beans:bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
		<beans:property name="exceptionMappings">
			<beans:props>
				<beans:prop key="TestException">error/testException</beans:prop>
			</beans:props>
		</beans:property>
		<beans:property name="defaultErrorView" value="error/error" />
	</beans:bean>

	<context:component-scan base-package="kro.rubisco.controller" />
	<context:component-scan base-package="kro.rubisco.interceptor" />

	<interceptors>
		<interceptor>
			<mapping path="/board/**" />
			<beans:ref bean="loginInterceptor" />
		</interceptor>
	</interceptors>

</beans:beans>
```

localhost:8080/board 에 접속하면 인터셉터를 통해 로그인 페이지로 이동하는 것을 확인할 수 있습니다. 로그인을 한 후 다시 /board에 접속하면 정상적으로 게시판이 출력됩니다.