15. Spring boot: Custom Interceptors

Interceptor:

it's a mediator, wich get invoked before or after your actual code.

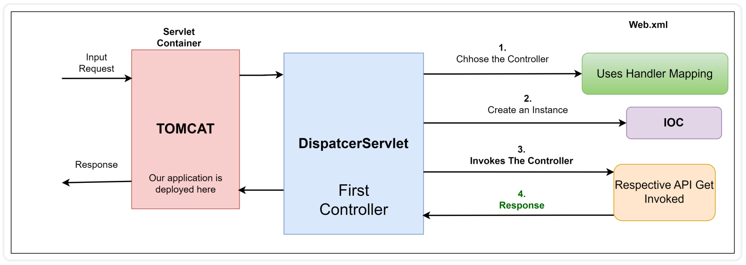
In future topics below, we might need to write our custom inceptors:

- Springboot Caching
- Springboot logging
- Spring Authentication etc.

Pre-requiste to understand custom interceptors better:

- Annotations in java
- AOP [Aspect Oriented Programming]

Custom Interceptors for Requests before even reachin to specific Controller class



```
@RestController
@RequestMapping(value = "/api/")
public class UserController {


    @Autowired
    UserServices userServices;

    @Autowired
    User user;
```

```
     @GetMapping(path = ”/getUser”)
    public String getUserDetails(){
        System.out.println(”Inside getUserDetails Current thread name:
”+Thread.currentThread().getName());
       // userServices.asyncMethodTest();
        asyncMethodTest();
        return user.getUserDetails();
    }
}
```

```
package com.springProject.SpringCourseProject.Interceptor;

import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

@Component
public class MyCustomInterceptor implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
throws Exception {
        System.out.println(”Inside Pre Handle Method”);
        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,
ModelAndView modelAndView) throws Exception {
        System.out.println(”Inside postHandle  Method”);

    }

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler,
Exception ex) throws Exception {
        System.out.println(”Inside afterCompletion  Method”);

    }



}
```

- Once PostHandle Method get's called after execution of PostHandle Method afterCompletion method is being called
- Both PostHandle and afterCompletion method are executing after controller
- When any exception occur post will not get executed but afterCompletion get ecectued same like as finally.

Now We have to register our custom interceptors, so we can decide which controller are gonna used it.

```
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import javax.sql.DataSource;
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicInteger;

@Configuration
public class AppConfig implements  WebMvcConfigurer {


    @Autowired
    MyCustomInterceptor myCustomInterceptor;



    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(myCustomInterceptor)
                .addPathPatterns("/api/*")// Apply to these URL Patterns
                .excludePathPatterns("/api/updateUser","/api/deleteUser"); //Exclude for these URL
pattwerns
    }
}
```

After hitting this API: http://localhost:8080/api/getUser

Output

Inside Pre Handle Method

Hitting GetUser from DB.

Inside postHandle Method

Inside afterCompletion Method

Custom Interceptors For Requests after reaching to specific Controller class

Step1: - Creation of Custom annotations

We can create Custom Annotation using Keyword **@interface** java annotation

```
public @interface MyCustomAnnotations{

}
```

```
public class User{
  @MyCustomAnnotation
  public void updateUser(){
    //some business logic
  }
}
```

2 Important Meta Annotations [has to be clear] Properties are:

@Target:

- This meta annotation, tels where we can apply the particular annotation on method or class or constructor etc.

```
@Target(ElementType.METHOD)
public @interface MyCustomAnnotations{

}
```

```
@Target(ElementType.CONSTRUCTOR,ElementType.METHOD,ElementType.PARAMETER,ElementType.FIELD)
public @interface MyCustomAnnotations{

}
```

@Retention:

- this meta annotation tell, how the particular annotation will be stored in java.

RetentionPolicy.SOURCE

- Annotation will be discarded by compiler itself and its not even recorded in .class file.

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.SOURCE)
public @interface MyCustomAnnotations{

}
```

```
public class User{
  @MyCustomAnnotation
  public void updateUser(){
    //some business logic
  }
}
```

```
//User.class file after compilation
public class User{

  public class User(){}

  public void updateUser(){
    //some business logic
  }
}
```

RetentionPolicy.CLASS

- Annotation will be recorded in .class fle but ignored by JVM during runtime

```
//User.class file after compilation
public class User{

  public class User(){}

  //this will ignore by JVM
  @MyCustomAnnotation
  public void updateUser(){
    //some business logic
  }
}
```

RetentionPolicy.RUNTIME

- Annotation will be recorded in .class and also available during run time

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface MyCustomAnnotations{

}
```

```
public class User{
  @MyCustomAnnotation
  public void updateUser(){
    //some business logic
  }
}
```

```
//User.class file after compilation
public class User{

  public class User(){}

  //During Reflection this will not ignored by JVM
  @MyCustomAnnotation
  public void updateUser(){
    //some business logic
  }
}
```

How to create Custom Annotation with methods (more like a fields):

- No parameter, no body
- Return type is restricted to

  > Primitive type (int, boolean, double etc.)

  > String

  > Enum

  > Class<?>

  > Annotations

  > Array of above types

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface MyCustomAnnotations{
  String key() default ”defaultKeyName”;
}
```

```
public class User{
  @MyCustomAnnotation(key=”userKey”)
  public void updateUser(){
    //some business logic
  }
}
```

Step1: Completed Creating MyCustomAnnotation

```
package com.springProject.SpringCourseProject.customAnnotation;

import com.springProject.SpringCourseProject.Enum.MyCustomEnum;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface MyCustomAnnotation {
    int intKey() default 0;

    String stringKey() default ”defaultStringKey”;

    Class<?> classTypeKey() default String.class;

    MyCustomEnum enumKey() default MyCustomEnum.ENUM_VAL1;

    String[] stringArrayKey() default {”default”,”default2”};

    int[] intArrayKey() default {1,2};
}
```

```
package com.springProject.SpringCourseProject.component;

import com.springProject.SpringCourseProject.Enum.MyCustomEnum;
import com.springProject.SpringCourseProject.customAnnotation.MyCustomAnnotation;
import jakarta.annotation.PostConstruct;
import org.springframework.context.annotation.Scope;
import org.springframework.context.annotation.ScopedProxyMode;
import org.springframework.stereotype.Component;

@Component
public class User{
    public User(){
        System.out.println("Initialization User");
    }



    @MyCustomAnnotation(intKey = 10,stringKey = "user",classTypeKey = User.class,enumKey =
MyCustomEnum.ENUM_VAL1)
    public void updateUser(){
        //business logic
    }
}
```

Step 2: Creation Of Custom Interceptor

1. Creating My Custom Annotation

```
@Component

public class User{
    public User(){

    @MyCustomAnnotation(name = "user")
    public String getUserDetails(){
        return "Hey! This Is Saurav";
    }



}
```

```
package com.springProject.SpringCourseProject.customAnnotation;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface MyCustomAnnotation {
    String name() default "";
}
```

```
@RestController
@RequestMapping(value = "/api/")
public class UserController {

    @Autowired
    User user;

    @GetMapping(path = "/getUser")
    public String getUserDetails(){
        System.out.println("Hitting GetUser from DB.");
        asyncMethodTest();
        return user.getUserDetails();
    }
}
```

Now We are creating Our own CustomInterceptoe

```
package com.springProject.SpringCourseProject.Interceptor;



import com.springProject.SpringCourseProject.customAnnotation.MyCustomAnnotation;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.reflect.MethodSignature;
import org.springframework.stereotype.Component;

import java.lang.reflect.Method;

@Component
@Aspect
public class MyCustomInterceptorUsesCustomAnnotation {

    //combination @Before and @After
    @Around("@annotation(com.springProject.SpringCourseProject.customAnnotation.MyCustomAnnotation)")
    public void invoke(ProceedingJoinPoint joinPoint) throws Throwable{

        System.out.println("do something before actual method");

        Method method = ((MethodSignature)joinPoint.getSignature()).getMethod();
        if (method.isAnnotationPresent(MyCustomAnnotation.class)){
            MyCustomAnnotation annotation = method.getAnnotation(MyCustomAnnotation.class);
```

```
                System.out.println("name from Annotation: "+annotation.name());
            }

            joinPoint.proceed();
            System.out.println("do something after actual method");

        }
    }
```

After hitting this API: http://localhost:8080/api/getUser

Output

Inside asyncMethodTest() of usercontroller Current thread name: http-nio-8080-exec-2

do something before actual method

name from Annotation: user

do something after actual method

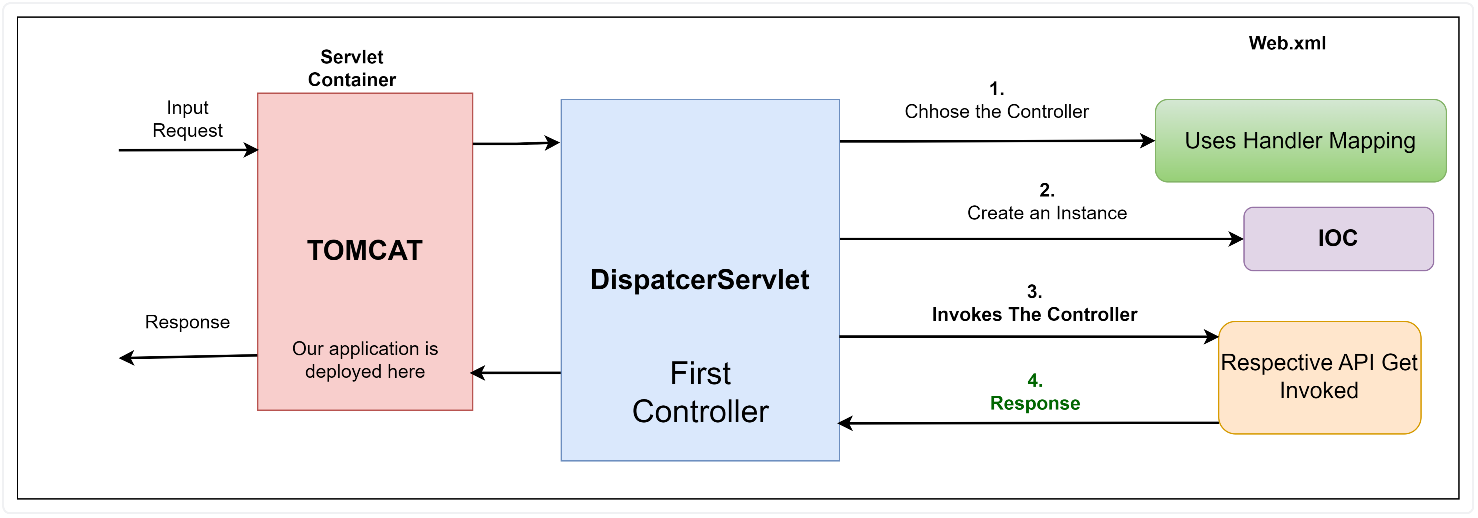
Spring boot: Filters Vs Interceptors

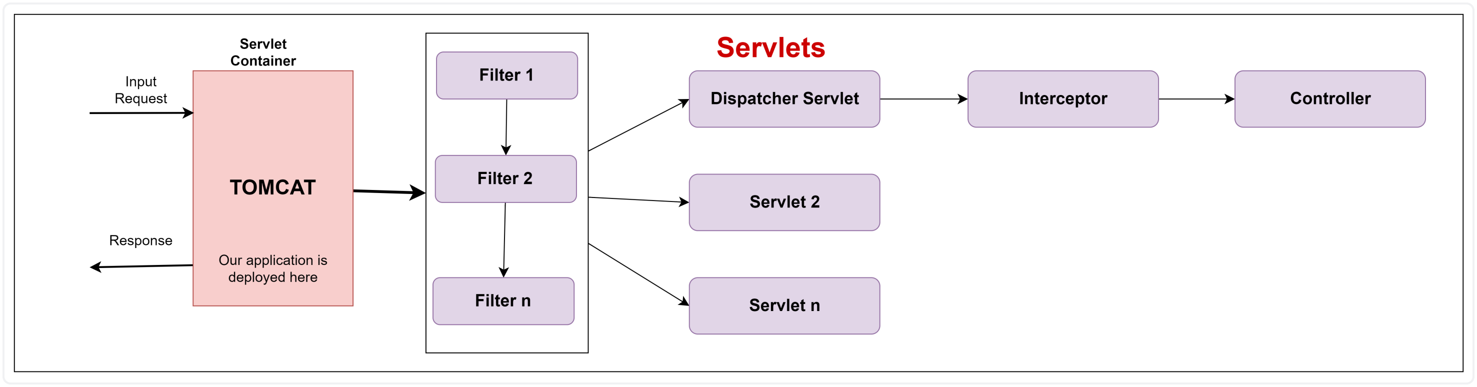
Filter:

- It intercepts the HTTP Request and Response, before they reach to ther servlet.

Interceptor:

- It's specific to spring framework, and intercept HTTP Request and response, before they reach to the controller.





What is servlet:

- Servlet is nothing but a java class, which accepts the incomming request, process it and return the response.

- In traditional Monolithic architucture we have different servlet for different thing.
- Like one servlet for REST API'S , ONE for SOAP API, and another one for File upload and retrive.

By the use of Contoller we reduce te use of MULTIPLE SERVLETS

We can create multiple servlet like:

- Servlet 1: can be configured to handle REST APIs.
- Servlet: can be configuired to handle SOAP APIS etc....

Similarly like this, "DispatcherServlet" is kind of servlet provided by spring, and by default it's configured to handle all API "/*".

Filter:

- It is used when we want to intercept HTTP Request and Response and add logic agnostic of the underlying servlets.
- We can have many filters and have ordering between them too.

Interceptors:

- It is used when we want to intercept HTTP Request and response and add logic specific to a particular servlet

- We can have many Interceptor and have ordering between them too.

Multiple Interceptors and its ordering:

- In previous video, we already saw, how to add 1 interceptor.
- How **preHandle**, **postHandle** and **afterCompletion**
- Now here, will show how to add more than 1.
- Aso if **preHandle** returns false, next interceptor and controller will not get invoked itself.

```
package com.springProject.SpringCourseProject.config;

@Configuration
public class AppConfig implements AsyncConfigurer, WebMvcConfigurer {



    @Autowired
    MyCustomInterceptor myCustomInterceptor;

    @Autowired
    MyCustomInterceptor2 myCustomInterceptor2;


    @Override
    public void addInterceptors(InterceptorRegistry registry) {

        //ordering
        //1->first executed this
        registry.addInterceptor(myCustomInterceptor)
                .addPathPatterns("/api/*")// Apply to these URL Patterns
                .excludePathPatterns("/api/updateUser","/api/deleteUser","/api/getUser"); //Exclude for
these URL pattwerns



        //ordering
        //2->and then executed this
        registry.addInterceptor(myCustomInterceptor2)
                .addPathPatterns("/api/*")
                .excludePathPatterns("/api/updateUser");
    }


}
```

```
@GetMapping(path = "/checkOrderingOfInterceptors")
public String checkOrderingOfInterceptors(){

    System.out.println("checkOrderingOfInterceptors controller hitted.");

    return "checkOrderingOfInterceptors controller hitted.";
}
```

hitting API : > http://localhost:8080/api/checkOrderingOfInterceptors

OUTPUT:

```
Inside Pre Handle Method of MyCustomInterceptor
Inside Pre Handle Method of MyCustomInterceptor2
checkOrderingOfInterceptors controller hitted.
Inside postHandle  Method of MyCustomInterceptor2
Inside postHandle  Method MyCustomInterceptor
Inside afterCompletion  Method of MyCustomInterceptor2
Inside afterCompletion  Method MyCustomInterceptor
```

Filter Example:

```
package com.springProject.SpringCourseProject.filter;

import jakarta.servlet.*;

import java.io.IOException;

public class MyFilter1 implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        Filter.super.init(filterConfig);
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain
filterChain) throws IOException, ServletException {
        System.out.println("MyFilter1 inside");
        filterChain.doFilter(servletRequest,servletResponse);
        System.out.println("MyFilter1 completed");
    }

    @Override
    public void destroy() {
        Filter.super.destroy();
    }
}
```

```
package com.springProject.SpringCourseProject.filter;

import jakarta.servlet.*;

import java.io.IOException;

public class MyFilter2 implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        Filter.super.init(filterConfig);
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain
filterChain) throws IOException, ServletException {
        System.out.println("MyFilter2 inside");
        filterChain.doFilter(servletRequest,servletResponse);
        System.out.println("MyFilter2 completed");
    }

    @Override
    public void destroy() {
        Filter.super.destroy();
    }
}
```

```
package com.springProject.SpringCourseProject.config;

import com.springProject.SpringCourseProject.filter.MyFilter1;
import com.springProject.SpringCourseProject.filter.MyFilter2;
import org.springframework.boot.web.servlet.FilterRegistrationBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class FilterConfiguration {

    @Bean
    public FilterRegistrationBean<MyFilter1> myFirstFilter(){
        FilterRegistrationBean<MyFilter1> filter1FilterRegistrationBean = new FilterRegistrationBean<>();
        filter1FilterRegistrationBean.setFilter(new MyFilter1());
        filter1FilterRegistrationBean.addUrlPatterns("/*");
        filter1FilterRegistrationBean.setOrder(2);
        return filter1FilterRegistrationBean;
    }

    @Bean
    public FilterRegistrationBean<MyFilter2> mySecondFilter(){
        FilterRegistrationBean<MyFilter2> filterFilterRegistrationBean = new FilterRegistrationBean<>();
        filterFilterRegistrationBean.setFilter(new MyFilter2());
        filterFilterRegistrationBean.addUrlPatterns("/*");
        filterFilterRegistrationBean.setOrder(1);
        return filterFilterRegistrationBean;
    }
}
```

```
@GetMapping(path = "/checkOrderingOfFilters")
public String checkOrderingOfFilters(){

    System.out.println("checkOrderingOfFilters controller hitted.");

    return "checkOrderingOfFilters controller hitted.";
}
```

hitting API: > http://localhost:8080/api/checkOrderingOfFilters

I am just excluding this path from interceptor checkOrderingOfFilters

OUTPUT:

```
MyFilter2 inside
MyFilter1 inside
checkOrderingOfFilters controller hitted.
MyFilter1 completed
MyFilter2 completed
```

Now I am including this path **checkOrderingOfFilters** to inceptor as well

Again hitting this API: > http://localhost:8080/api/checkOrderingOfFilters

```
MyFilter2 inside
MyFilter1 inside
Inside Pre Handle Method of MyCustomInterceptor
Inside Pre Handle Method of MyCustomInterceptor2
checkOrderingOfFilters controller hitted.
Inside postHandle  Method of MyCustomInterceptor2
Inside postHandle  Method MyCustomInterceptor
Inside afterCompletion  Method of MyCustomInterceptor2
Inside afterCompletion  Method MyCustomInterceptor
MyFilter1 completed
MyFilter2 completed
```