DispatcherServlet

Spring Web Mvc

Index

Servlet

자바를 사용해 웹 어플리케이션을 동적으로 생성하는 서버측 프로그램을 말한다

자바 서블릿은 웹 서버의 성능을 향상 시키기 위해 사용되는 자바 클래스의 일종이다

Request 당 Thread를 만들어서 또는 Thread pool에서 Thread를 가져오는 식으로 처리한다

(Process당 자원을 공유하는 Thread)

기존에는 요청당 Process를 만들었다.

Servlet Interface를 구현한게 HttpServlet 그래서 중요하다.

자바 기반이라 OS Platform의 독립적이라는 특징이 있다.

DispatcherServlet

Spring Mvc의 핵심 요소 HttpServlet을 extends한 것

Primary purpose is to handle incoming web requests matching configured URL pattern.

FrontController 역할을 한다.

(중앙집중형 컨트롤러를 제일 앞에둬서 서버로 들어오는 모든 요청을 먼저 받아서 처리하게 만든다.)

(클라이언트가 보낸 요청을 받아서 공통적인 작업을 먼저 수행후에 적절한 세부 컨트롤러로 작업을 위임한다.)

(클라이언트에게 보낼 뷰를 선택해서 최종 결과를 생성하는 작업을 수행한다.)

```
public class MyWebApplicationInitializer implements WebApplicationInitializer {

    @Override
    public void onStartup(ServletContext servletCxt) {

        // Load Spring web application configuration
        AnnotationConfigWebApplicationContext ac = new AnnotationConfigWebApplicationContext();
        ac.register(AppConfig.class);
        ac.refresh();

        // Create and register the DispatcherServlet
        DispatcherServlet servlet = new DispatcherServlet(ac);
        ServletRegistration.Dynamic registration = servletCxt.addServlet("app", servlet);
        registration.setLoadOnStartup(1);
        registration.addMapping("/app/*");
    }
}
```

DispatcherServlet 작동원리

특정 URL로 요청 했을때 무슨 일이 일어나는지

Request -> ServletContainer DispatcherClass doService() -> doDispatch()
-> processDispatchResult -> render()

doService(): 요청을 처리하는데 필요한 Bean들을 Request에 주입한다.

doDispatch: URL과 요청 정보를 기준으로 어떤 컨트롤러가 사용할것인지 결정하고
이 선택한 컨트롤러를 DispatchServlet이 호출한다.

processDispatchResult에서 타고 들어가면 render()가 나오는데 Model And View를 만드는 역할을 한다.

DispatcherServlet 작동원리

doService()

```
protected void doService(HttpServletRequest request, HttpServletResponse response) throws Exception {
    …
    request.setAttribute(WEB_APPLICATION_CONTEXT_ATTRIBUTE, this.getWebApplicationContext());
    request.setAttribute(LOCALE_RESOLVER_ATTRIBUTE, this.localeResolver);
    request.setAttribute(THEME_RESOLVER_ATTRIBUTE, this.themeResolver);
    request.setAttribute(THEME_SOURCE_ATTRIBUTE, this.getThemeSource());
    if (this.flashMapManager != null) {
        FlashMap inputFlashMap = this.flashMapManager.retrieveAndUpdate(request, response);
        if (inputFlashMap != null) {
            request.setAttribute(INPUT_FLASH_MAP_ATTRIBUTE, Collections.unmodifiableMap(inputFlashMap));
        }

        request.setAttribute(OUTPUT_FLASH_MAP_ATTRIBUTE, new FlashMap());
        request.setAttribute(FLASH_MAP_MANAGER_ATTRIBUTE, this.flashMapManager);
    }

    try {
        this.doDispatch(request, response);
    } finally {
        if (!WebAsyncUtils.getAsyncManager(request).isConcurrentHandlingStarted() && attributesSnapshot != null) {
            this.restoreAttributesAfterInclude(request, attributesSnapshot);
        }

    }

}
```

DispatcherServlet 작동원리

doDispatch()

```
protected void doDispatch(HttpServletRequest request, HttpServletResponse response) throws Exception {
    …
    try {
        try {
            ModelAndView mv = null;
            Object dispatchException = null;

            try {
                processedRequest = this.checkMultipart(request);
                multipartRequestParsed = processedRequest != request;
                mappedHandler = this.getHandler(processedRequest);
                if (mappedHandler == null) {
                    this.noHandlerFound(processedRequest, response);
                    return;
                }

                HandlerAdapter ha = this.getHandlerAdapter(mappedHandler.getHandler());

                …

                this.processDispatchResult(processedRequest, response, mappedHandler, mv, (Exception)dispatchException);
            } catch (Exception var22) {
                this.triggerAfterCompletion(processedRequest, response, mappedHandler, var22);
            } catch (Throwable var23) {
                this.triggerAfterCompletion(processedRequest, response, mappedHandler, new NestedServletException("Handler processing failed", var23));
            }
        } f
        }
}
```

DispatcherServlet 작동원리

doDispatch()

```
protected void doDispatch(HttpServletRequest request, HttpServletResponse response) throws Exception {
    …
    try {
        try {
            ModelAndView mv = null;
            Object dispatchException = null;

            try {
                processedRequest = this.checkMultipart(request);
                multipartRequestParsed = processedRequest != request;
                mappedHandler = this.getHandler(processedRequest);
                if (mappedHandler == null) {
                    this.noHandlerFound(processedRequest, response);
                    return;
                }

                HandlerAdapter ha = this.getHandlerAdapter(mappedHandler.getHandler());

                …

                mv = ha.handle(processedRequest, response, mappedHandler.getHandler());

                this.processDispatchResult(processedRequest, response, mappedHandler, mv, (Exception)dispatchException);
            } catch (Exception var22) {
                this.triggerAfterCompletion(processedRequest, response, mappedHandler, var22);
            } catch (Throwable var23) {
                this.triggerAfterCompletion(processedRequest, response, mappedHandler, new NestedServletException("Handler processing failed", var23));
            }
        } f
        }
}
```

DispatcherServlet 작동원리

doDispatch() - this.getHandler

URL과 요청 정보를 기준으로 어떤 핸들러 오브젝트
즉 컨트롤러를 사용할 것인지 결정하는 로직을 담당한다.

DispatcherServlet은 하나 이상의 핸들러 매핑을 가질 수 있고

기본적으로 두개의 핸들러 매핑을 가진다.

BeanNameUrlHandlerMapping,
DefaultAnnotaionHandlerMapping (Deprecated)
RequestMappingHandlerMapping
(@RequestMapping, @Controller)

Spring Mvc는 자주 사용되는 전략을 Default로 설정해두고 있다.

필요한 전략은 확장해서 사용하면 된다.

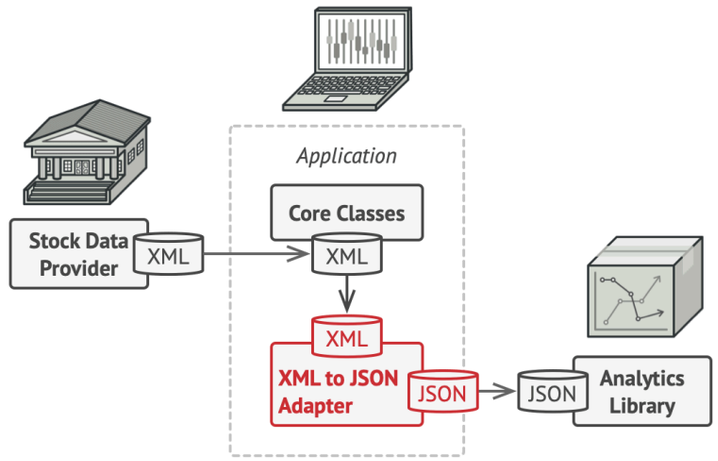
DispatcherServlet 작동원리

doDispatch() - this.getHandlerAdapter

HandlerAdapter는 HandlerMapping으로 선택한 컨트롤러를 DispatcherServlet이 호출할 때 사용하는 Adapter이다.

Adapter종류
HttpRequestHandlerAdapter
SimpleControllerHandlerAdapter
RequestMappingHandlerAdapter - (RequestMappingHandlerMapping)

Why Adapter?
- **Controller Type에는 제한이 없고 Controller 호출 방법은 Type마다 다르다**
- **그래서 Controller를 결정했다고 해도 호출 방법을 DispatcherServlet이 알 수 있는 방법이 없다.**



Adapter Pattern 그림

DispatcherServlet 작동원리

doDispatch() - ha.handle()

handle() -> handleInternal() -> invokeHandlerMethod() 이 과정을 통해서 Handler Method를 호출한다.

이미 handlerMethod안에서는 어떤 메소드를 호출해야하는지 다 결정되어 있다.

메소드 호출은 Java reflection을 통해서 호출가능하다.

```
protected ModelAndView handleInternal
(HttpServletRequest request, HttpServletResponse response, HandlerMethod handlerMethod) throws Exception {
    …
       mav = this.invokeHandlerMethod(request, response, handlerMethod);

    if (!response.containsHeader("Cache-Control")) {
        if (this.getSessionAttributesHandler(handlerMethod).hasSessionAttributes()) {
            this.applyCacheSeconds(response, this.cacheSecondsForSessionAttributeHandlers);
        } else {
            this.prepareResponse(response);
        }
    }

    return mav;
}
```

DispatcherServlet 작동원리

HandlerMethodReturnValueHandler - returnValueHandler

Handler에서 처리할 Return 값을 어떻게 처리할 것인가?

다양한 Return을 제공할 수 있다. (15가지)

HttpEntityMethodProcessor,
HttpHeadersReturnValueHandler,
MapMethodProcessor,
ModelAndViewMethodReturnValueHandler,
ModelAndViewResolverMethodReturnValueHandler,
ModelAttributeMethodProcessor,
ModelMethodProcessor,
RequestResponseBodyMethodProcessor,
ResponseBodyEmitterReturnValueHandler,
ServletModelAttributeMethodProcessor,
StreamingResponseBodyReturnValueHandler,
ViewMethodReturnValueHandler,
ViewNameMethodReturnValueHandler
DeferredResultMethodReturnValueHandler,
AsyncTaskMethodReturnValueHandler

References

https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html

https://medium.com/@sarovanakumar/java-servlet-lifecycle-example-648b4ddcabca

토비의 스프링 3.1 Vol2 스프링의 기술과 선택

http://iloveulhj.github.io/posts/spring/spring-dispatchsersvlet.html

https://refactoring.guru/design-patterns/adapter