Team4
Spring-Webmvc

林柏劭 王廷峻 王郁婷 蔡昀達 陳熙 黃柏諭

Workload Allocation

	Source Code Tracing	Implementation	Presentation
林柏劭	DispatcherServlet, FrameworkServlet, HttpServletBean, HttpServlet, theme package, i18n package, support package	DispatcherServlet, MockHttpServletResponse test_DispatcherServlet	Spring-webmvc overview DispatcherServlet
黃柏諭	HandlerExecutionChain, HandlerMethod, HandlerInterceptor, MappedInterceptor, HandlerMapping, AbstractHandlerMapping, MatchableHandlerMapping, AbstractUrlHandlerMapping, SimpleUrlHandlerMapping, AbstractDetectingHandlerMapping, BeanNameUrlHandlerMapping, AbstractHandlerMethodMapping, RequestMappingInfoHandlerMapping, RequestMappingHandlerMapping, UrlPathHelper	AbstractUrlHandlerMapping(minor), UrlPathHelper, PatternParser, SimpleUrlHandlerMapping test, serveral code fix about whole HandlerMapping	HandlerMapping
王廷峻	HandlerAdapter, SimpleControllerHandlerAdapter, HttpRequestHandlerAdapter, RequestMappingHandlerAdapter, ServletRequestDataBinderFactory, ServletInvocableHandlerMethod, ExtendedServletRequestDataBinder, HandlerMethodArgumentResolver, HandlerMethodArgumentResolverComposite, HandlerMethodReturnValueHandler, HandlerMethodReturnValueHandlerComposite, AbstractMessageConverterArgmentResolver, RequetBdoyAdvice, AbstractMessageConverterMethodProcessor,RequestResponseBodyMethodProcessor, ResponseBodyAdvice, Controller PathVariableMethodArgumentResolver, RequestAttributeMethodArgumentResolver, RequestResponseBodyAdviceChain, ---------- WebDataBinderFactory, DefaultDataBinderFactory, InitBinderFactory, InvocableHandlerMethod, HttpMessageConverter, AbstractNamedValueArgumentResolver, ControllerAdviceBean	HandlerAdapter, SimpleControllerHandlerAdapter, Controller, AbstractUrlHandlerMapping	HandlerAdapter
王郁婷	ViewResolver, ViewResolverComposite, AbstractCachingViewResolver, BeanNameViewResolver, ContentNegotiatingViewResolver, UrlBasedViewResolver, ResourceBundleViewResolver, XmlViewResolver, InternalResourceViewResolver, AbstractTemplateViewResolver, GroovyMarkupViewResolver, InternalResourceView, RedirectView, SmartView View, AbstractView, AbstractUrlBasedView, AbstractTemplateView, GroovyMarkupView,	ViewResolver, ViewResolverComposite, ContentNegotiatingViewResolver, AbstractCachingViewResolver, UrlBasedViewResolver, InternalResourceViewResolver	ViewResolver
陳熙	XsltViewResolver, XmlViewResolver, ViewResolver, TilesViewResolver, ScriptTemplateViewResolver, InternalResourceViewResolver, FreeMarkerViewResolver, AbstractTemplateViewResolver, AbstractFeedView, ContentNegotiatingViewResolver	Aware, BeanNameAware, InitializingBean, Environment, EnvironmentCapable, PropertyResolver, ApplicationObjectSupport, MessageSourceAccessor, MessageSource, MessageSourceResolvable, WebUtils	View
蔡昀達	View, AbstractView, AbstractUrlBasedView, RedirectView, InternalResourceView, ScriptTemplateView, SmartView, TilesView, JstlView, XsltView, AbstractTemplateView, ModelAndView, BeanDefinitionParser, SimpleControllerHandlerAdapter, AbstractDetectingUrlHandlerMapping, BeanNameUrlHandlerMapping, HandlerMapping, StaticApplicationContext, StaticWebApplicationContext, MvcNamespaceHandler, MvcNamespaceUtils, Annotation,	HandlerExecutionChain, BeanDefinition, BeanDefinitionParser, AbstractBeanDefinitionParser, ParserContext, AbstractView, JstlView, AbstractUrlBasedView, RedirectView, InternalResourceView, MockRequestDispatcher, MockAsyncContext, MockServletContext, ModelAndView, SimpleControllerHandlerAdapter, HandlerMapping, AbstractDetectingUrlHandlerMapping, BeanNameUrlHandlerMapping, StaticApplicationContext, StaticWebApplicationContext, Test, pipeline	Python Implementation Demo

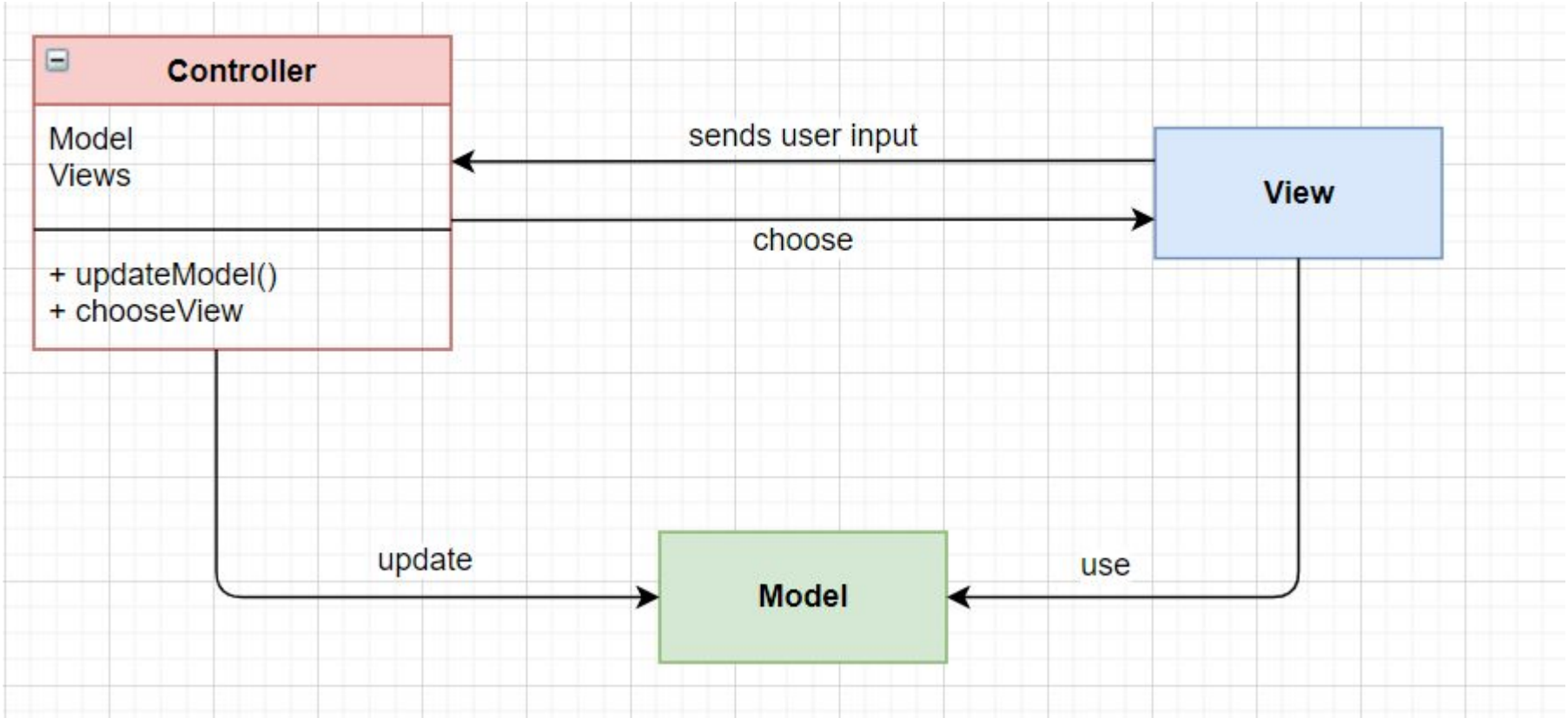
Outline

- Spring-webmvc Overview
- Trace Code Walkthrough
  - **DispatcherServlet (5min)**
  - **Handler Mapping (7min)**
  - **Handler Adapter (8min)**
  - **ViewResolver (7min)**
  - **View (7min)**
- Python Implementation (7min)
- Video Demo (4min)

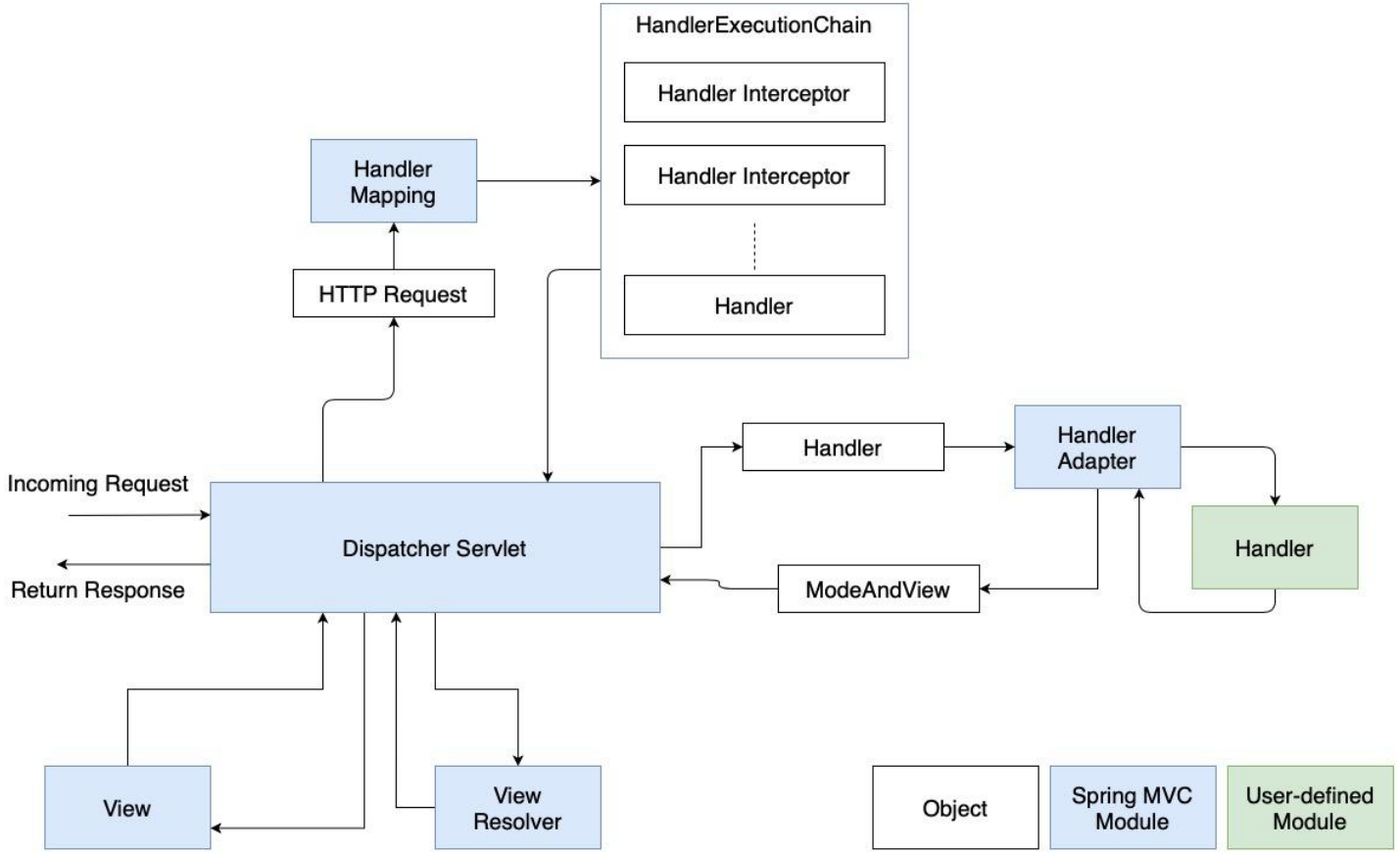
Spring-webmvc Overview

Presenter: **林柏劭**

MVC Pattern


Controller
Model
Views
+ updateModel()
+ chooseView
sends user input
choose
View
update
Model
use

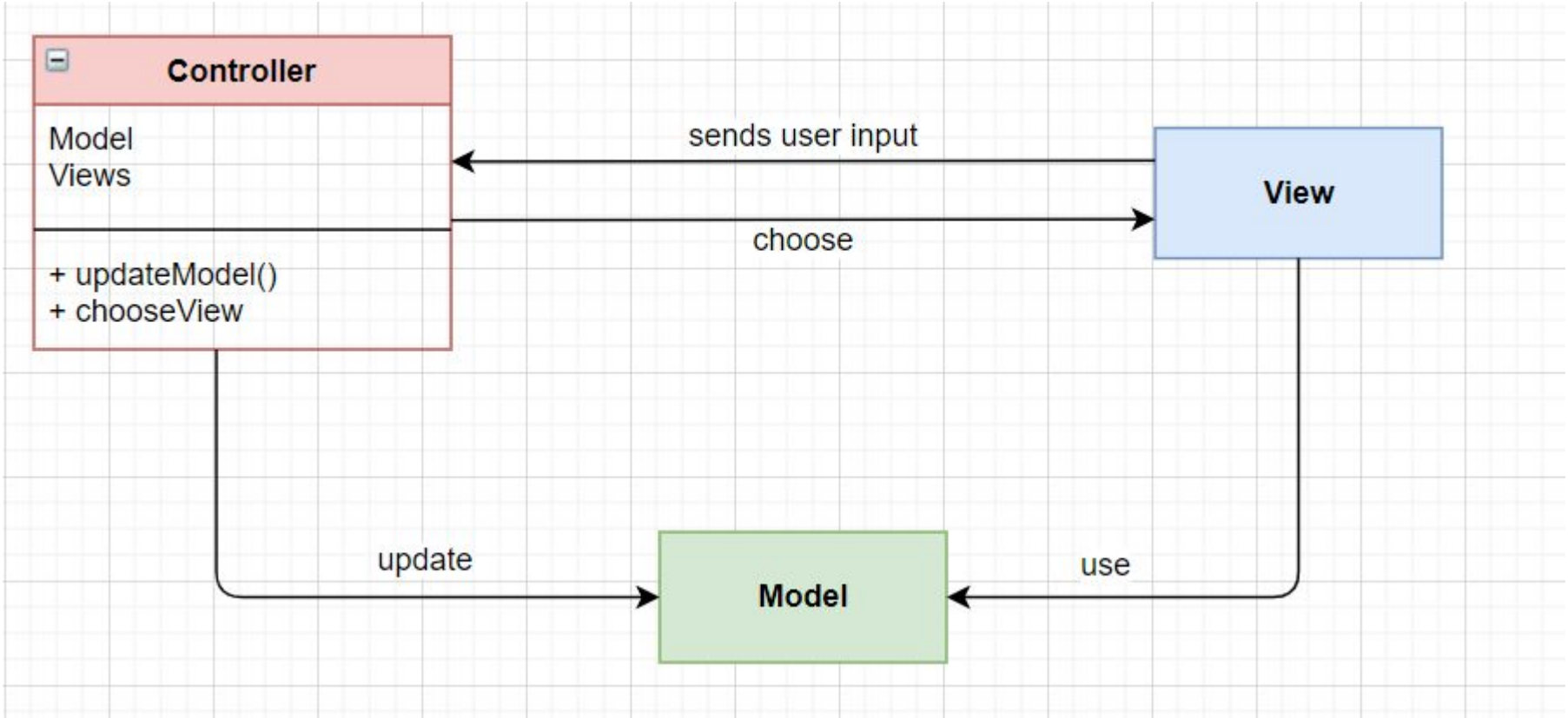
Process Flow



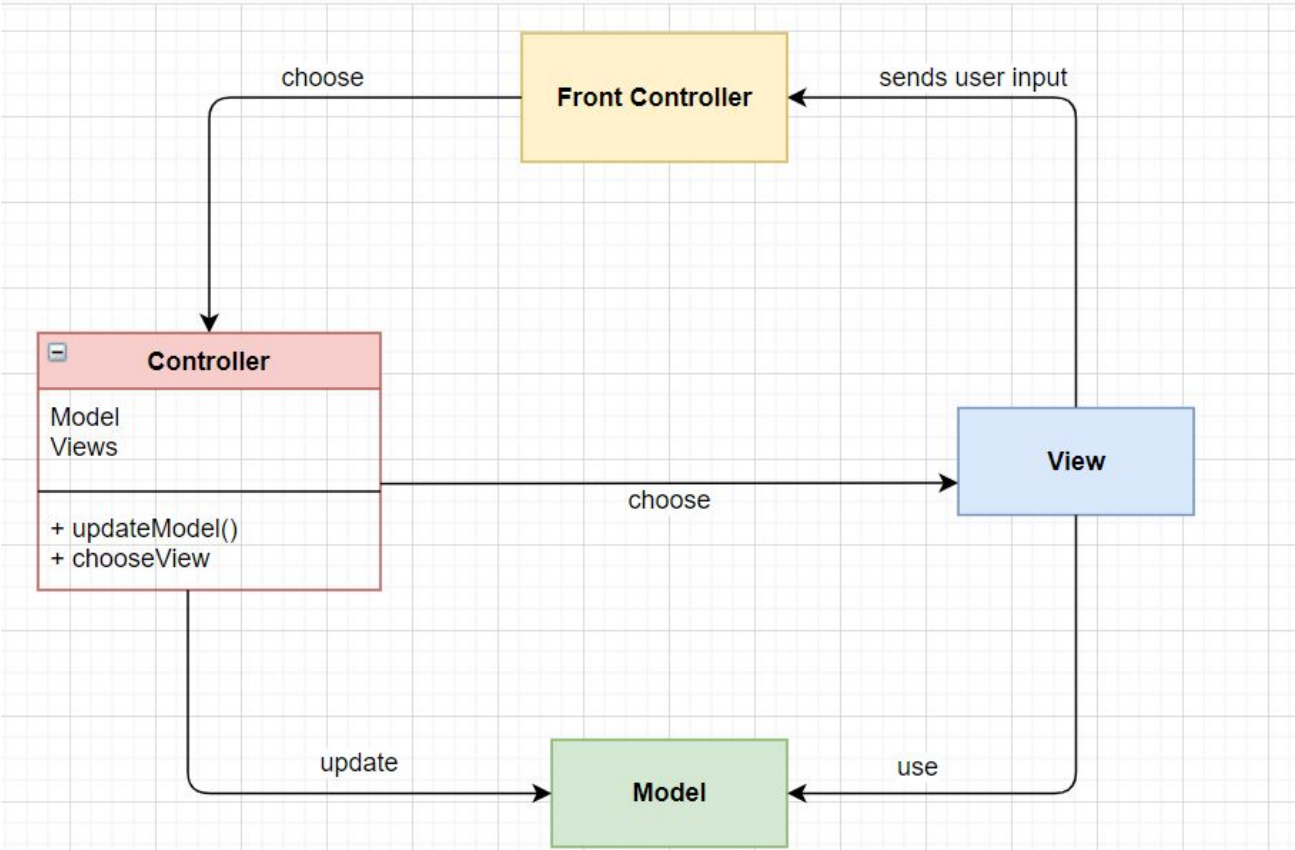
DispatcherServlet

Presenter: **林柏劭**

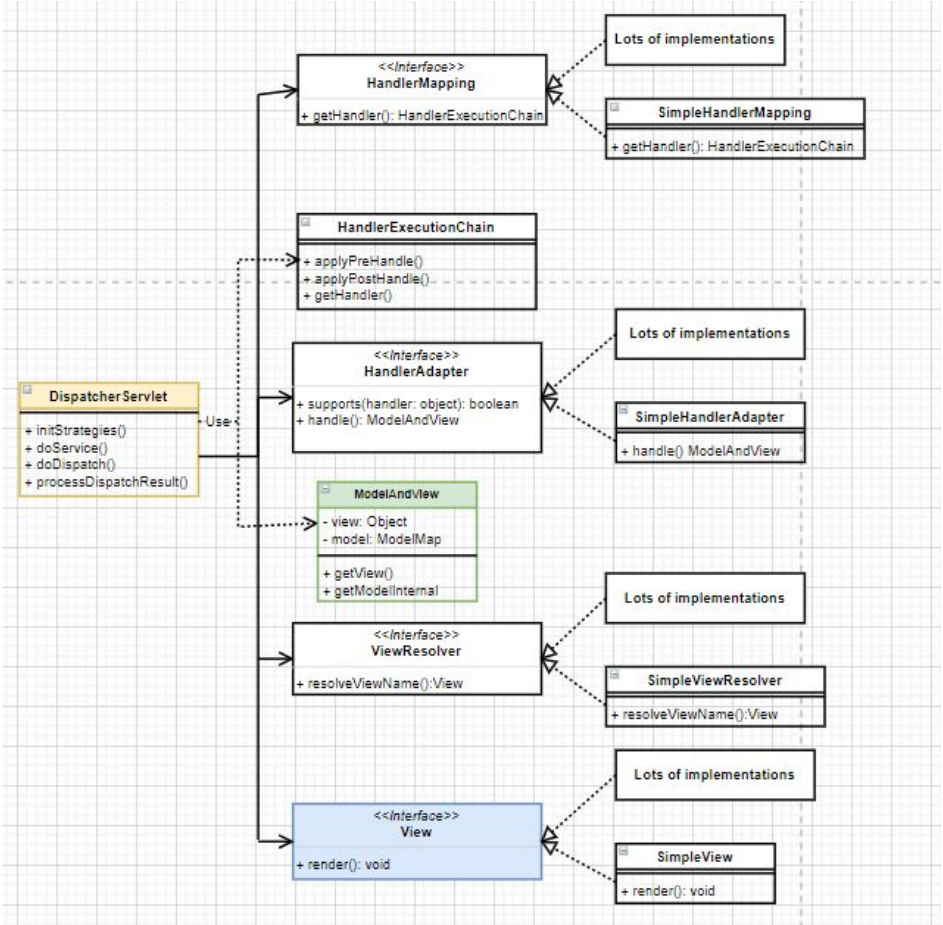
MVC Pattern


Controller
Model
Views
+ updateModel()
+ chooseView
sends user input
choose
View
update
Model
use

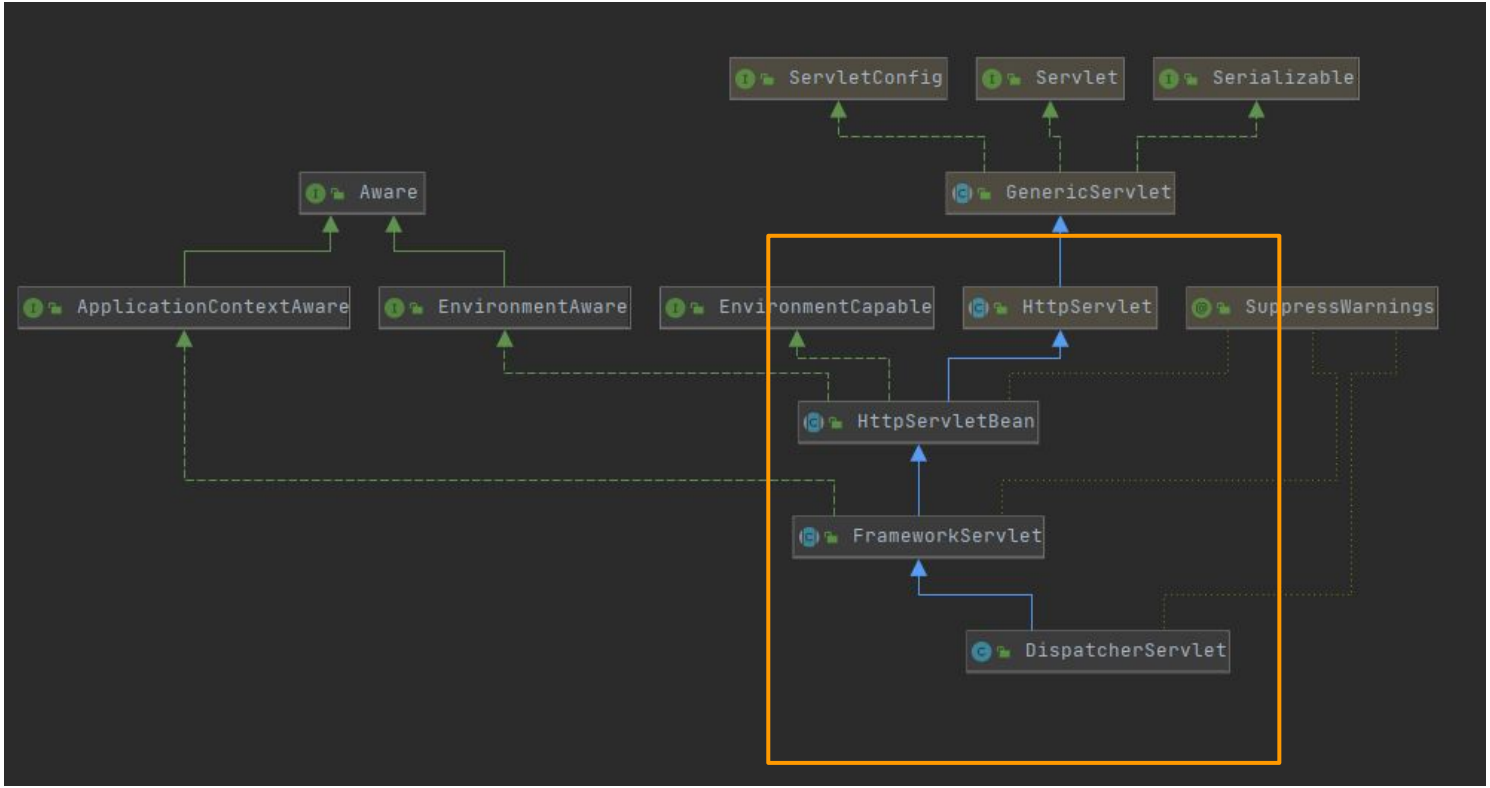
Front Controller Pattern



DispatcerServlet Class Diagram

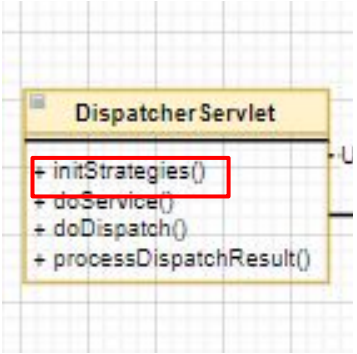


DispatcerServlet Class Diagram



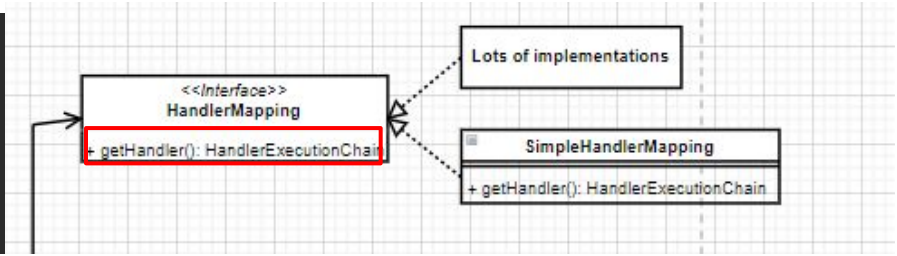
Source Code

```java
/**
 * This implementation calls {@link #initStrategies}.
 */
@Override
protected void onRefresh(ApplicationContext context) { initStrategies(context); }

/**
 * Initialize the strategy objects that this servlet uses.
 * <p>May be overridden in subclasses in order to initialize further strategy objects.
 */
protected void initStrategies(ApplicationContext context) {
    initMultipartResolver(context);
    initLocaleResolver(context);
    initThemeResolver(context);
    initHandlerMappings(context);
    initHandlerAdapters(context);
    initHandlerExceptionResolvers(context);
    initRequestToViewNameTranslator(context);
    initViewResolvers(context);
    initFlashMapManager(context);
}
```



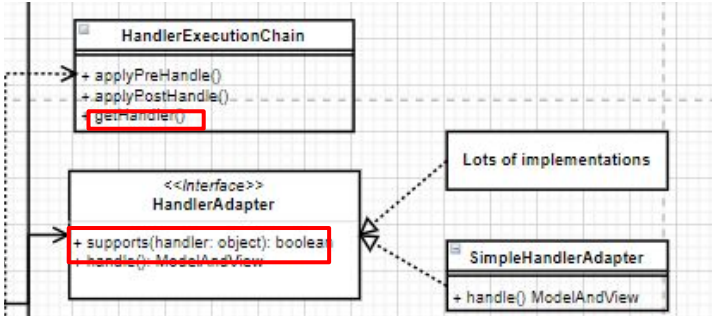
Source Code

```
mappedHandler = getHandler(processedRequest);
if (mappedHandler == null) {
    noHandlerFound(processedRequest, response);
    return;
}
```




```
protected HandlerExecutionChain getHandler(HttpServletRequest request) throws Exception {
    if (this.handlerMappings != null) {
        for (HandlerMapping mapping : this.handlerMappings) {
            HandlerExecutionChain handler = mapping.getHandler(request);
            if (handler != null) {
                return handler;
            }
        }
    }
    return null;
}
```

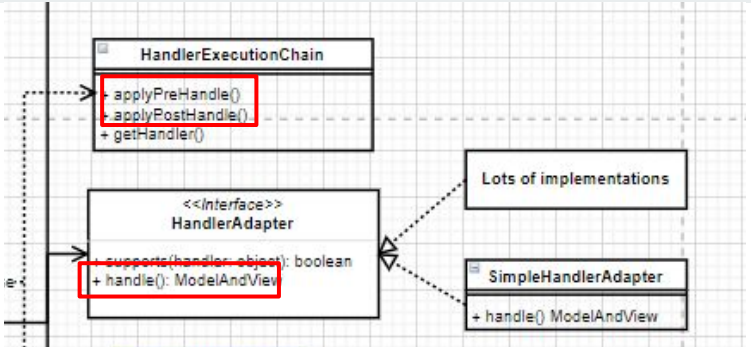
Source Code




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

```
protected HandlerAdapter getHandlerAdapter(Object handler) throws ServletException {
    if (this.handlerAdapters != null) {
        for (HandlerAdapter adapter : this.handlerAdapters) {
            if (adapter.supports(handler)) {
                return adapter;
            }
        }
    }
```

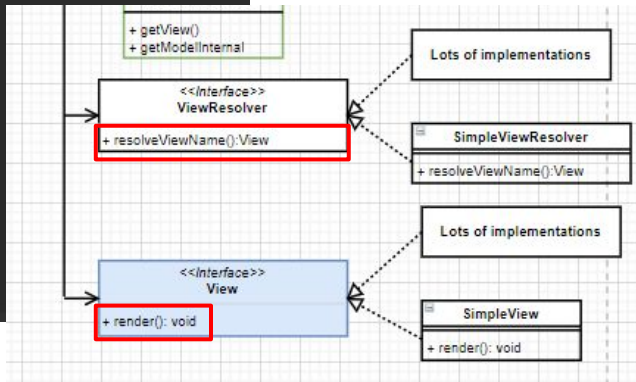
Source Code




```
if (!mappedHandler.applyPreHandle(processedRequest, response)) {
    return;
}

// Actually invoke the handler.
mv = ha.handle(processedRequest, response, mappedHandler.getHandler());

mappedHandler.applyPostHandle(processedRequest, response, mv);
```

```
view = resolveViewName(viewName, mv.getModelInternal(), locale, request);
```

```
protected View resolveViewName(String viewName, @Nullable Map<String, Object> model,
        Locale locale, HttpServletRequest request) throws Exception {

    if (this.viewResolvers != null) {
        for (ViewResolver viewResolver : this.viewResolvers) {
            View view = viewResolver.resolveViewName(viewName, locale);
            if (view != null) {
                return view;
            }
        }
    }
    return null;
}
```

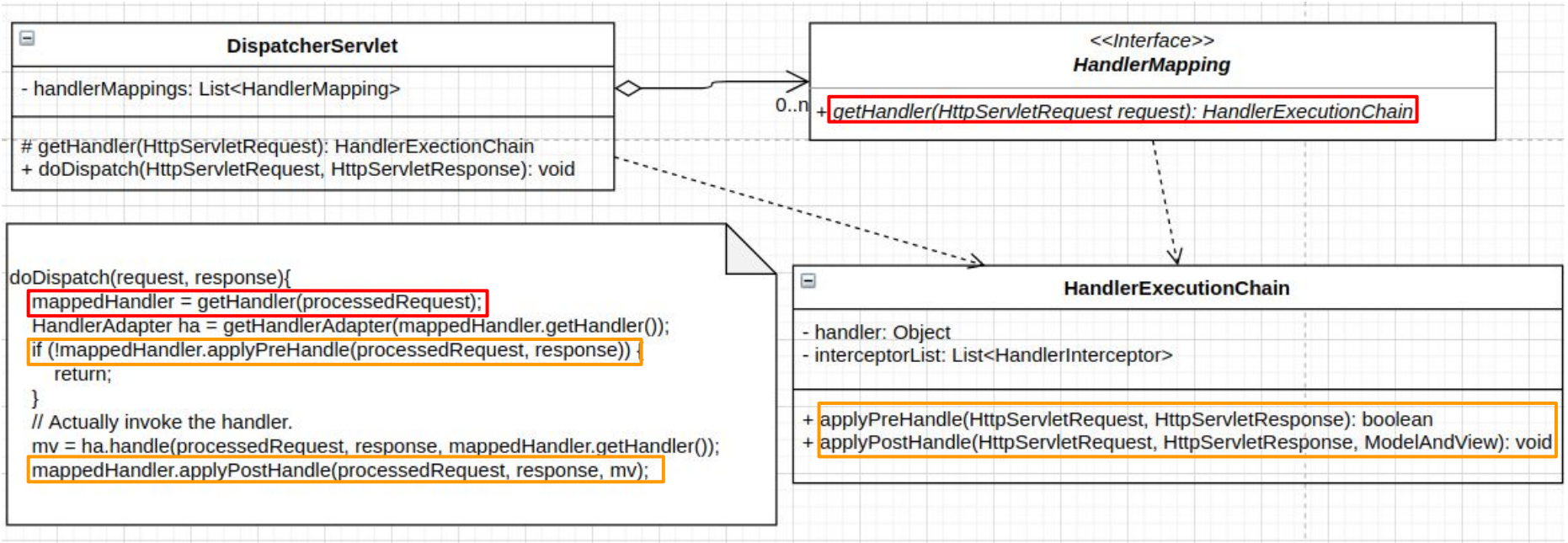



```
view.render(mv.getModelInternal(), request, response);
```

HandlerMapping

Presenter: **黃柏諭**

Mapping Pattern



Problem

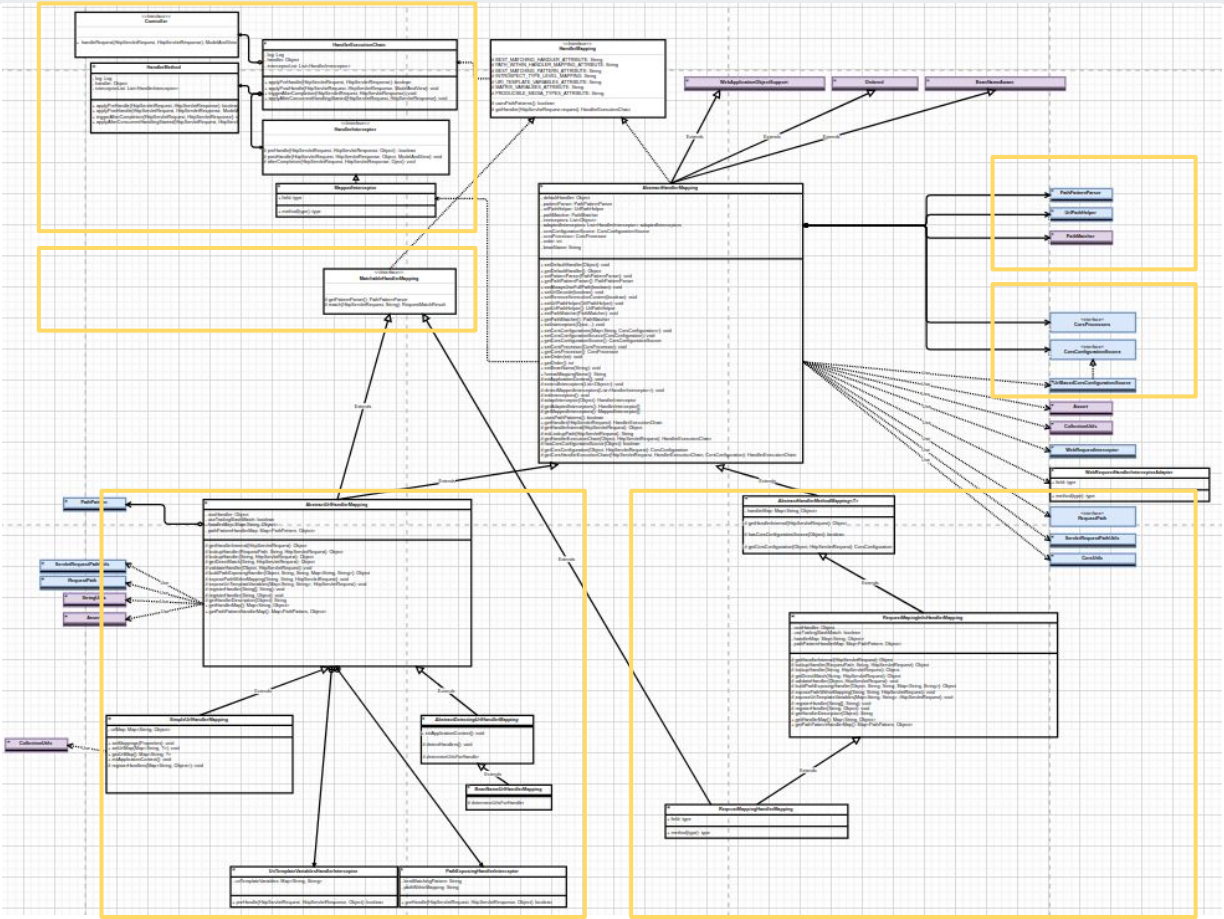
Code of DispatcherServlet should frequently be modified as different url mapping

Class Diagram

HandlerExecutionChain: One handler with some interceptors

Pattern Match: Parse urls pattern with different parser

Url Base Mapping: Map url to handler

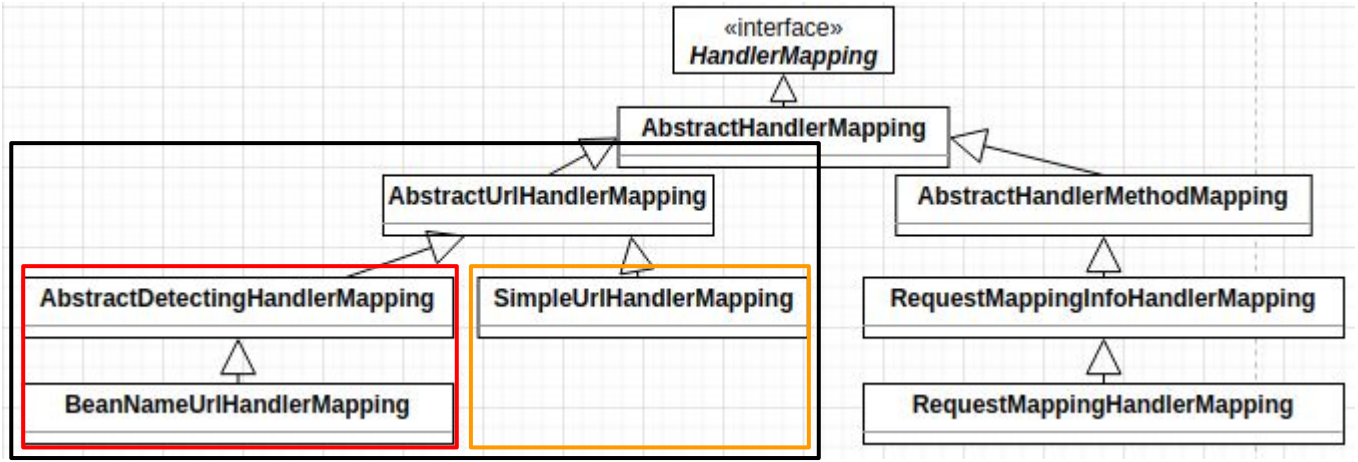


Pattern Parser: Multiple pattern parser such as AntPattern, Regex

CORS Request Process: Handle cross-origin resource sharing

Request Base Mapping: Map request to Handler method

Types of HandlerMapping

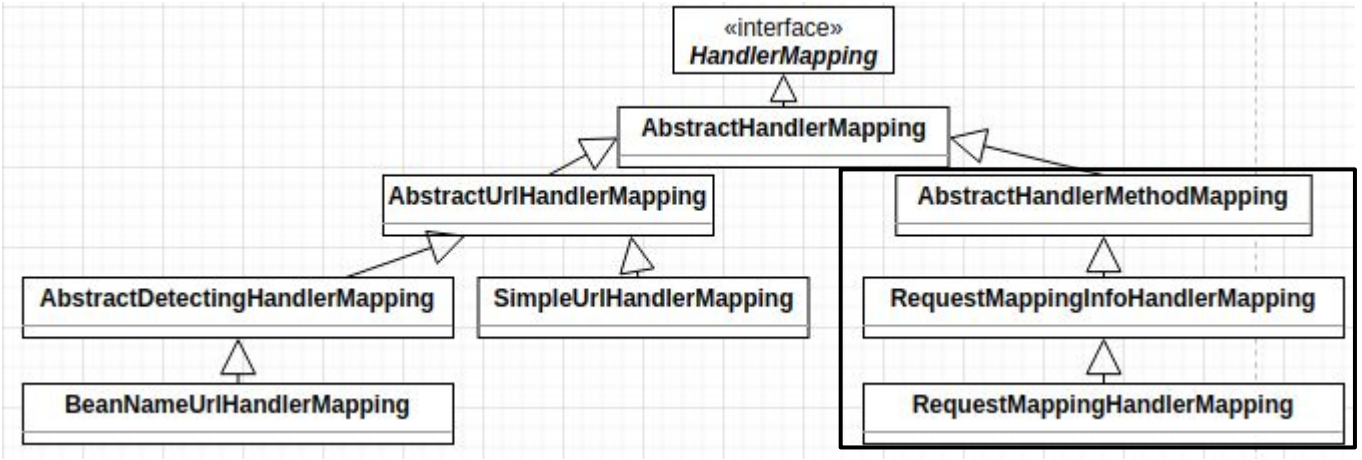



```java
@Configuration
public class BeanNameUrlHandlerMappingConfig {
    @Bean("/beanNameUrl")
    public WelcomeController welcome() {
        return new WelcomeController();
    }
}
```

```xml
<bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping" />
<bean name="/beanNameUrl" class="com.test.WelcomeController" />
```

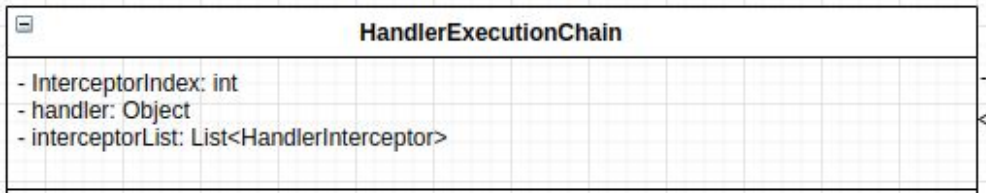
```xml
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <value>
            /simpleUrlWelcome=welcome
            /*/simpleUrlWelcome=welcome
        </value>
    </property>
</bean>
<bean id="welcome" class="com.test.WelcomeController" />
```

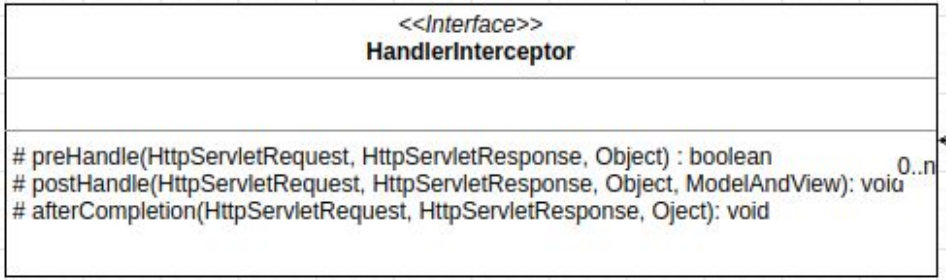
Types of HandlerMapping




```java
@Controller
public class ExampleController {

    @RequestMapping(value="/login", method=RequestMethod.GET)
    public ModelAndView login(HttpServletRequest req, HttpServletResponse resp) {
        // ...
    }
}
```

Intercepting Filter Pattern





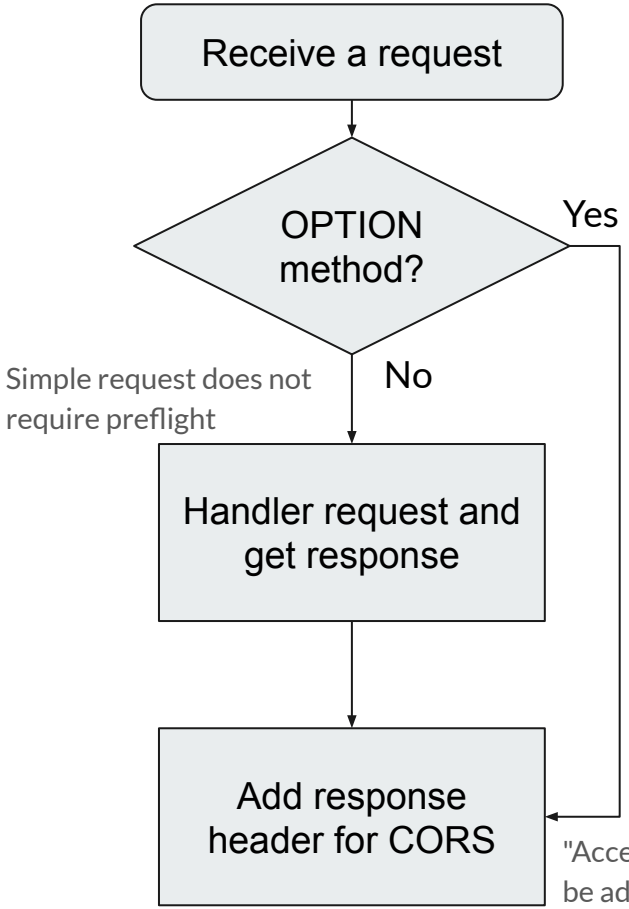

```
applyPreHandle(request, response){
    for (int i = 0; i < this.interceptorList.size(); i++) {
        HandlerInterceptor interceptor = this.interceptorList.get(i);
        if (!interceptor.preHandle(request, response, this.handler)) {
            triggerAfterCompletion(request, response, null);
            return false;
        }
        this.interceptorIndex = i;
    }
}

applyPostHandle(request, response) {
    for (int i = this.interceptorList.size() - 1; i >= 0; i--) {
        HandlerInterceptor interceptor = this.interceptorList.get(i);
        interceptor.postHandle(request, response, this.handler, mv);
    }
}
```

If an inceptor's prehandle method return false, stop the later inceptors.

If any prehandler return false, abort the request.

```
doDispatch(request, response){
    mappedHandler = getHandler(processedRequest);
    HandlerAdapter ha = getHandlerAdapter(mappedHandler.getHandler());
    if (!mappedHandler.applyPreHandle(processedRequest, response)) {
        return;
    }
    // Actually invoke the handler.
    mv = ha.handle(processedRequest, response, mappedHandler.getHandler());
    mappedHandler.applyPostHandle(processedRequest, response, mv);
```

Interceptor Example - CORS



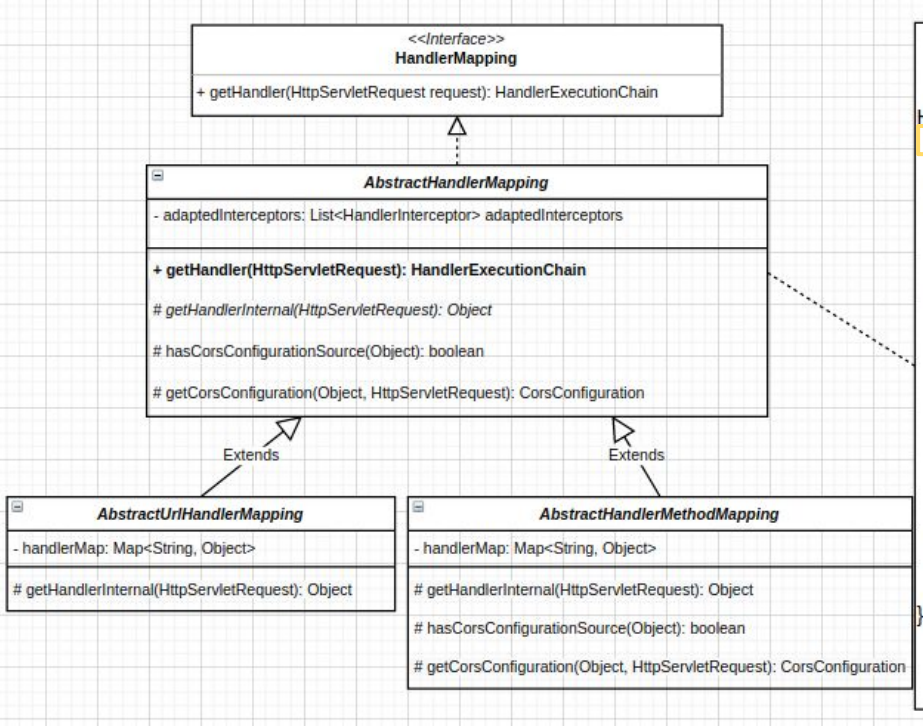
Problem

- Handler should be frequently modified to adapt new function

```
public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
        Object handler) throws Exception {
    this.corsData = new CorsData(request);

    if(this.corsData.isPreflighted()) {
        response.setHeader("Access-Control-Allow-Origin", origin);
        response.setHeader("Access-Control-Allow-Methods", allowMethods);

        return false;
    }

    return true;
}
```

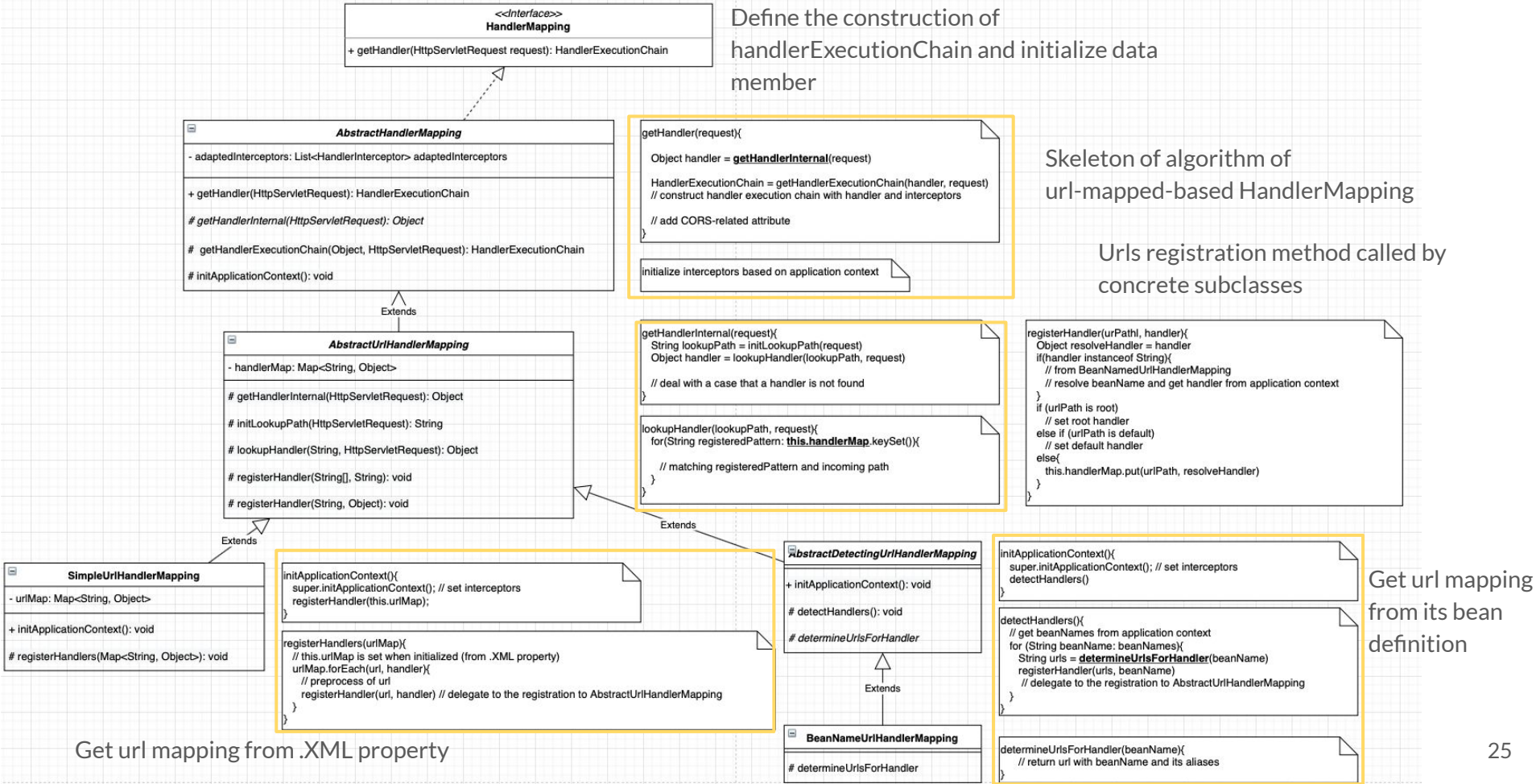
```
public void postHandle(HttpServletRequest request, HttpServletResponse response,
        Object handler, ModelAndView modelAndView) throws Exception {
    if(this.corsData.isSimple()) {
        response.setHeader("Access-Control-Allow-Origin", origin);
    }
}
```

Template Method Pattern




```
HandlerExecutionChain getHandler(HttpServletRequest request) {
    Object handler = getHandlerInternal(request);

    // ...

    HandlerExecutionChain executionChain = getHandlerExecutionChain(handler, request);

    if (hasCorsConfigurationSource(handler) || CorsUtils.isPreFlightRequest(request)) {
        CorsConfiguration config = getCorsConfiguration(handler, request);
        if (getCorsConfigurationSource() != null) {
            CorsConfiguration globalConfig = getCorsConfigurationSource().getCorsConfiguration(request);
            config = (globalConfig != null ? globalConfig.combine(config) : config);
        }
        if (config != null) {
            config.validateAllowCredentials();
        }
        executionChain = getCorsHandlerExecutionChain(request, executionChain, config);
    }

    return executionChain;
}
```

Template Method Pattern - 2



Source Code

AbstractHandlerMapping

```java
public final HandlerExecutionChain getHandler(HttpServletRequest request) throws Exception {
    Object handler = getHandlerInternal(request);
    if (handler == null) {
        handler = getDefaultHandler();
    }
    if (handler == null) {
        return null;
    }
    // Bean name or resolved handler?
    if (handler instanceof String) {
        String handlerName = (String) handler;
        handler = obtainApplicationContext().getBean(handlerName);
    }

    HandlerExecutionChain executionChain = getHandlerExecutionChain(handler, request);
```

```java
protected abstract Object getHandlerInternal(HttpServletRequest request) throws Exception;
```

AbstractUrlHandlerMapping

```java
protected Object getHandlerInternal(HttpServletRequest request) throws Exception {
    String lookupPath = initLookupPath(request);
    Object handler;
    if (usesPathPatterns()) {
        RequestPath path = ServletRequestPathUtils.getParsedRequestPath(request);
        handler = lookupHandler(path, lookupPath, request);
    }
    else {
        handler = lookupHandler(lookupPath, request);
    }
```

AbstractUrlHandlerMapping

```java
protected Object lookupHandler(String lookupPath, HttpServletRequest request)
    Object handler = getDirectMatch(lookupPath, request);
    if (handler != null) {
        return handler;
    }

    // Pattern match?
    List<String> matchingPatterns = new ArrayList<>();
    for (String registeredPattern : this.handlerMap.keySet()) {
        if (getPathMatcher().match(registeredPattern, lookupPath)) {
            matchingPatterns.add(registeredPattern);
        }
```

```java
protected void registerHandler(String urlPath, Object handler) throws BeansException, IllegalStateException {
    Assert.notNull(urlPath, message: "URL path must not be null");
    Assert.notNull(handler, message: "Handler object must not be null");
    Object resolvedHandler = handler;

    // Eagerly resolve handler if referencing singleton via name.
    if (!this.lazyInitHandlers && handler instanceof String) {
        String handlerName = (String) handler;
        ApplicationContext applicationContext = obtainApplicationContext();
        if (applicationContext.isSingleton(handlerName)) {
            resolvedHandler = applicationContext.getBean(handlerName);
        }
    }

    Object mappedHandler = this.handlerMap.get(urlPath);
    if (mappedHandler != null) {
        if (mappedHandler != resolvedHandler) {
            throw new IllegalStateException(
                    "Cannot map " + getHandlerDescription(handler) + " to URL path [" + urlPath +
                    "]: There is already " + getHandlerDescription(mappedHandler) + " mapped.");
        }
    }
    else {
        if (urlPath.equals("/")) {
            setRootHandler(resolvedHandler);
        }
        else if (urlPath.equals("/*")) {
            setDefaultHandler(resolvedHandler);
        }
        else {
            this.handlerMap.put(urlPath, resolvedHandler)
```

Source Code

SimpleUrlHandlerMapping

```java
@Override
public void initApplicationContext()
    super.initApplicationContext();
    registerHandlers(this.urlMap);
}
```

```java
protected void registerHandlers(Map<String, Object> urlMap) th
    if (urlMap.isEmpty()) {
        logger.trace("No patterns in " + formatMappingName());
    }
    else {
        urlMap.forEach((url, handler) -> {
            // Prepend with slash if not already present.
            if (!url.startsWith("/")) {
                url = "/" + url;
            }
            // Remove whitespace from handler bean name.
            if (handler instanceof String) {
                handler = ((String) handler).trim();
            }
            registerHandler(url, handler);
```

Delegate the url registration to AbstractUrlHandlerMapping#registerHandler

AbstractDetectingUrlHandlerMapping

```java
@Override
public void initApplicationContext()
    super.initApplicationContext();
    detectHandlers();
}
```

```java
protected void detectHandlers() throws BeansException {
    ApplicationContext applicationContext = obtainApplicationContext();
    String[] beanNames = (this.detectHandlersInAncestorContexts ?
            BeanFactoryUtils.beanNamesForTypeIncludingAncestors(applicationContext, Object.class) :
            applicationContext.getBeanNamesForType(Object.class));

    // Take any bean name that we can determine URLs for.
    for (String beanName : beanNames) {
        String[] urls = determineUrlsForHandler(beanName);
        if (!ObjectUtils.isEmpty(urls)) {
            // URL paths found: Let's consider it a handler.
            registerHandler(urls, beanName);
        }
    }
```

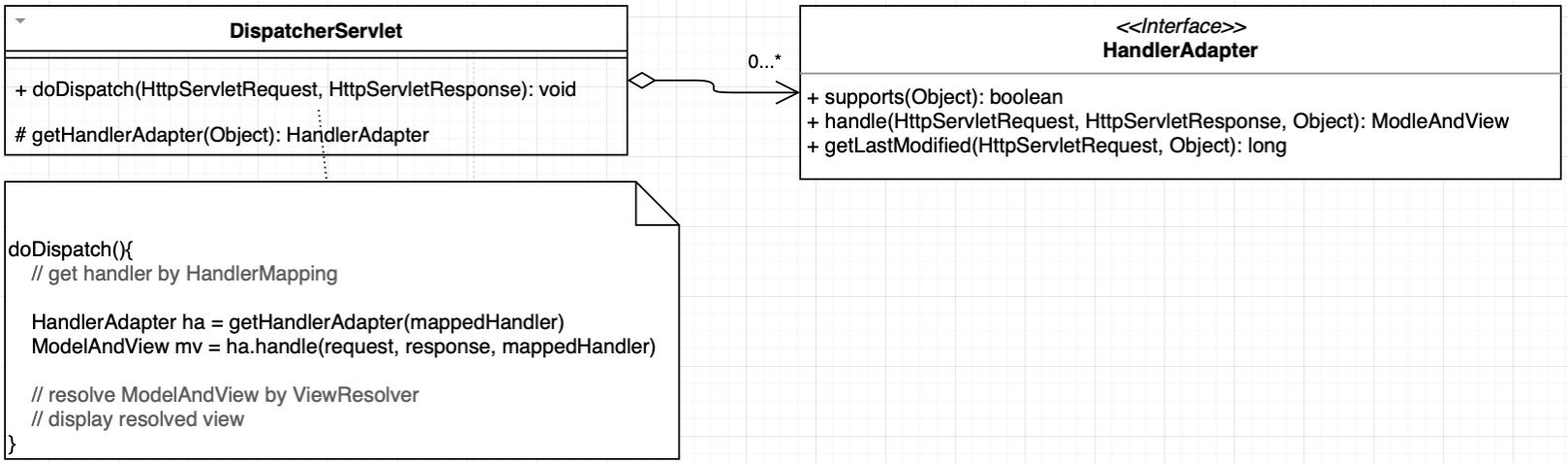
BeanNameUrlHandlerMapping

```java
@Override
protected String[] determineUrlsForHandler(String beanName) {
    List<String> urls = new ArrayList<>();
    if (beanName.startsWith("/")) {
        urls.add(beanName);
    }
    String[] aliases = obtainApplicationContext().getAliases(beanName);
    for (String alias : aliases) {
        if (alias.startsWith("/")) {
            urls.add(alias);
        }
    }
    return StringUtils.toStringArray(urls);
}
```

HandlerAdapter

Presenter: 王廷峻

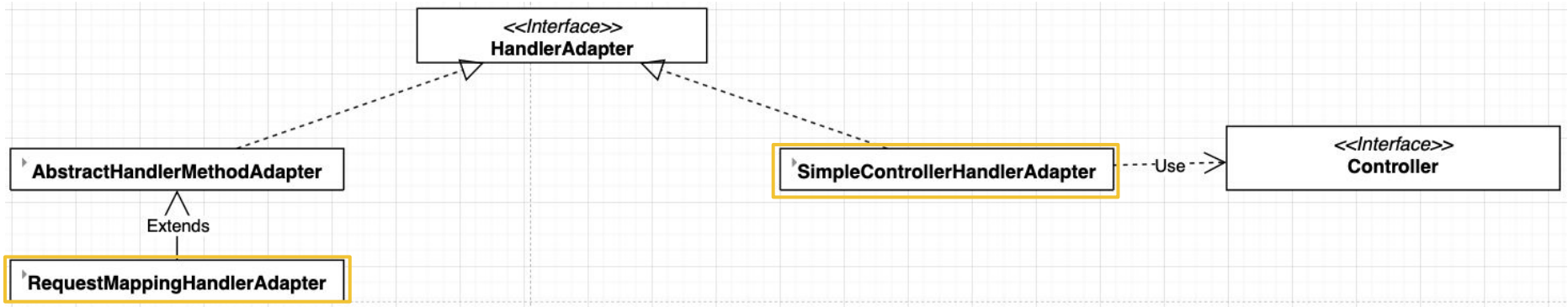
Adapter Pattern



Problem
Code of DispatcherServlet should frequently be modified as more handler types are defined

Intent
Abstract the common behavior of resolving a request to arguments and handling return value

Types of HandlerAdapter



Annotation-based controller

```
@RequestMapping(value = "/ex/foos/{id}", method = GET)
@ResponseBody
public String getFoosBySimplePathWithPathVariable(
    @PathVariable String id) {
        return "Get a specific Foo with id=" + id;
}
```

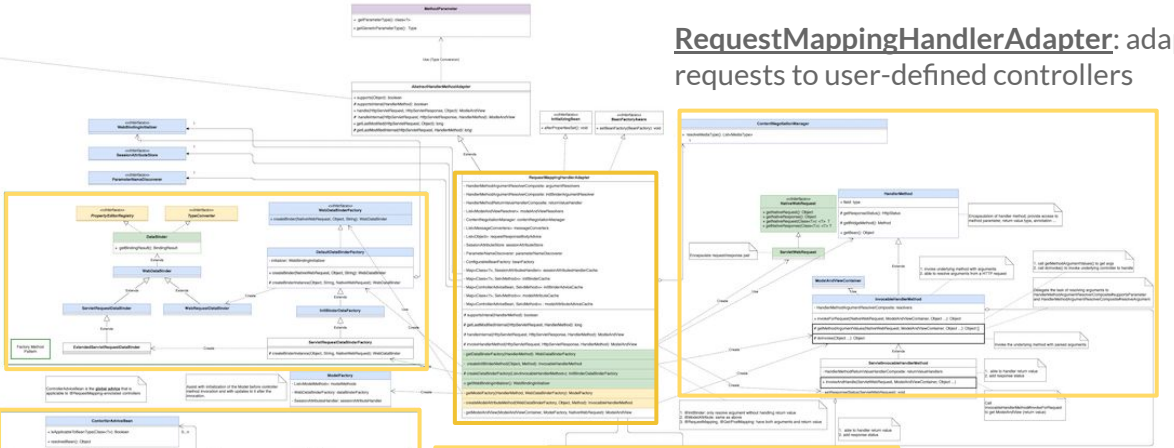
Custom controller extending Controller Interface

```
public class MyTestController extends Controller {

    @Override
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response){
        response.getWriter().write("my-test-controller");
        return null;
    }

}
```

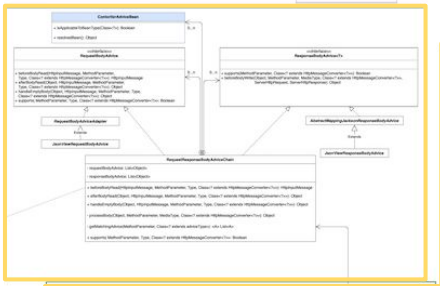
RequestMappingHandlerAdapter

RequestMappingHandlerAdapter: adapt HTTP requests to user-defined controllers

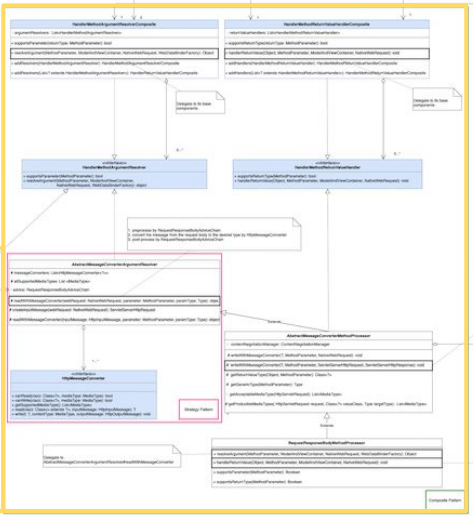
Data binding: tight the content of request with controller parameter



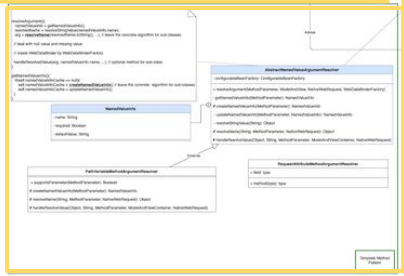
RequestResponseBodyAdviceChain: preprocess and post-process of reading request body and writing response body



Configure handler: add data binder factory, set argument resolver, and return value handler …



Implementation of argument resolver and return value handler



Data Binding Mechanism

Binding the content of a HTTP request (e.g., body, parameter, url, header ...) with parameters of a controller

@RequestBody: annotated parameter & request body

```
@PostMapping("/request")
public ResponseEntity postController(
  @RequestBody LoginForm loginForm) {

    exampleService.fakeAuthenticate(loginForm);
    return ResponseEntity.ok(HttpStatus.OK);
}
```

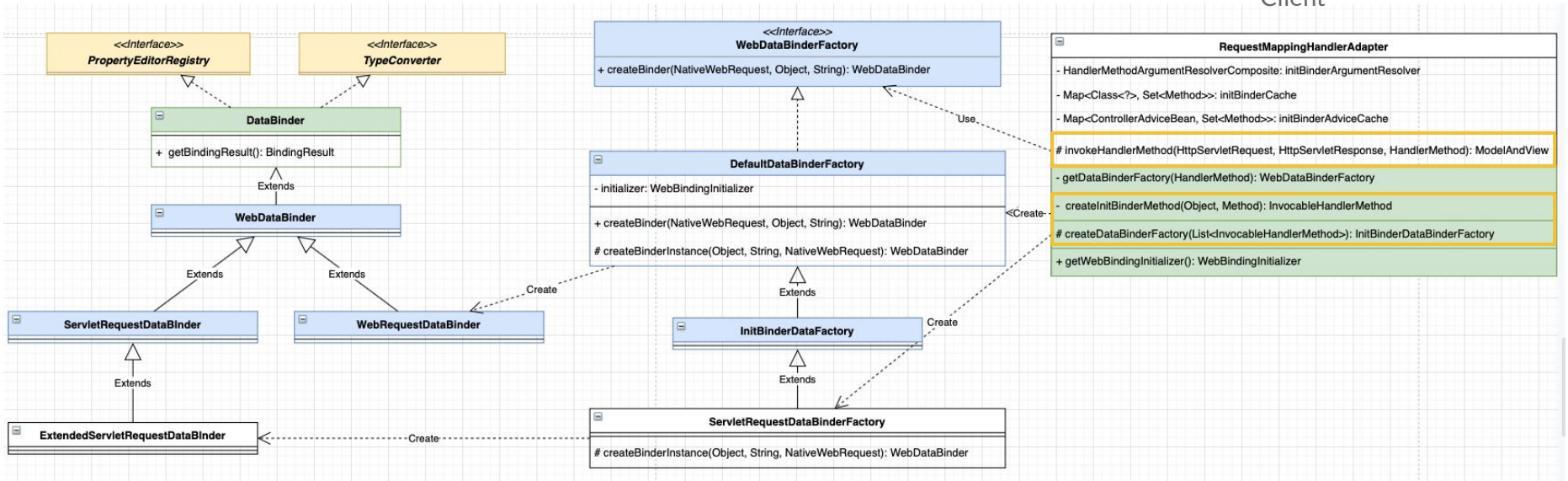
@PathVariable: annotated parameter & request URL

```
@GetMapping("/api/employees/{id}")
@ResponseBody
public String getEmployeesById(@PathVariable String id) {
    return "ID: " + id;
}
```

```
http://localhost:8080/api/employees/111
----
ID: 111
```

Factory Pattern

Instantiate an instance of the subclass of WebDataBinderFactory, then inject it into argumentResolver



Source Code

RequestMappingHandlerAdapter

```
protected ModelAndView invokeHandlerMethod(HttpServletRequest request,
        HttpServletResponse response, HandlerMethod handlerMethod) throws Exception {

    ServletWebRequest webRequest = new ServletWebRequest(request, response);

    WebDataBinderFactory binderFactory = getDataBinderFactory(handlerMethod);

    ServletInvocableHandlerMethod invocableMethod = createInvocableHandlerMethod(handlerMethod);
    if (this.argumentResolvers != null) {
        invocableMethod.setHandlerMethodArgumentResolvers(this.argumentResolvers);
    }
    if (this.returnValueHandlers != null) {
        invocableMethod.setHandlerMethodReturnValueHandlers(this.returnValueHandlers);
    }
    invocableMethod.setDataBinderFactory(binderFactory);

    // init modelAndView
}
```

```
private InvocableHandlerMethod createInitBinderMethod(Object bean, Method method) {
    InvocableHandlerMethod binderMethod = new InvocableHandlerMethod(bean, method);
    if (this.initBinderArgumentResolvers != null) {
        binderMethod.setHandlerMethodArgumentResolvers(this.initBinderArgumentResolvers);
    }
    binderMethod.setDataBinderFactory(new DefaultDataBinderFactory(this.webBindingInitializer));
    return binderMethod;
}

protected InitBinderDataBinderFactory createDataBinderFactory(List<InvocableHandlerMethod> binderMethods)
        throws Exception {

    return new ServletRequestDataBinderFactory(binderMethods, getWebBindingInitializer());
}
```

```
private WebDataBinderFactory getDataBinderFactory(HandlerMethod handlerMethod) throws Exception {
    Set<Method> methods = this.initBinderCache.get(handlerType);

    List<InvocableHandlerMethod> initBinderMethods = new ArrayList<>();
    // Global methods first
    this.initBinderAdviceCache.forEach((controllerAdviceBean, methodSet) -> {
        if (controllerAdviceBean.isApplicableToBeanType(handlerType)) {
            Object bean = controllerAdviceBean.resolveBean();
            for (Method method : methodSet) {
                initBinderMethods.add(createInitBinderMethod(bean, method));
            }
        }
    });
    for (Method method : methods) {
        Object bean = handlerMethod.getBean();
        initBinderMethods.add(createInitBinderMethod(bean, method));
    }
    return createDataBinderFactory(initBinderMethods);
}
```

RequestResponseBodyMethodProcessor#resolveArgument

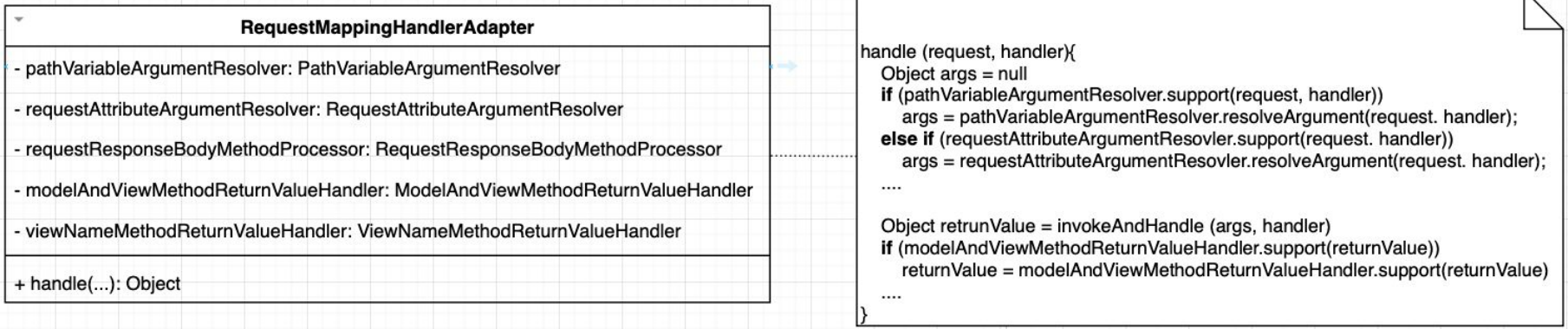
```
public Object resolveArgument(MethodParameter parameter, @Nullable ModelAndViewContainer mavContainer,
        NativeWebRequest webRequest, @Nullable WebDataBinderFactory binderFactory) throws Exception {

    // get arguments: args

    if (binderFactory != null) {
        WebDataBinder binder = binderFactory.createBinder(webRequest, arg, name);
        // ...
    }
    return adaptArgumentIfNecessary(arg, parameter);
}
```

ArgumentResolver/ReturnValueHandler

```java
// http://localhost:8080/api/employees/abc
@GetMapping("/api/employees/{id}")
@ResponseBody
public String getEmployeesById(@PathVariable String id) {
    return "ID: " + id;
}
```

```java
// http://localhost:8080/api/foos?id=abc
@GetMapping("/api/foos")
@ResponseBody
public String getFoos(@RequestParam String id) {
    return "ID: " + id;
}
```

```java
@ModelAttribute
public void addAttributes(Model model) {
    model.addAttribute("msg", "Welcome");
}
```



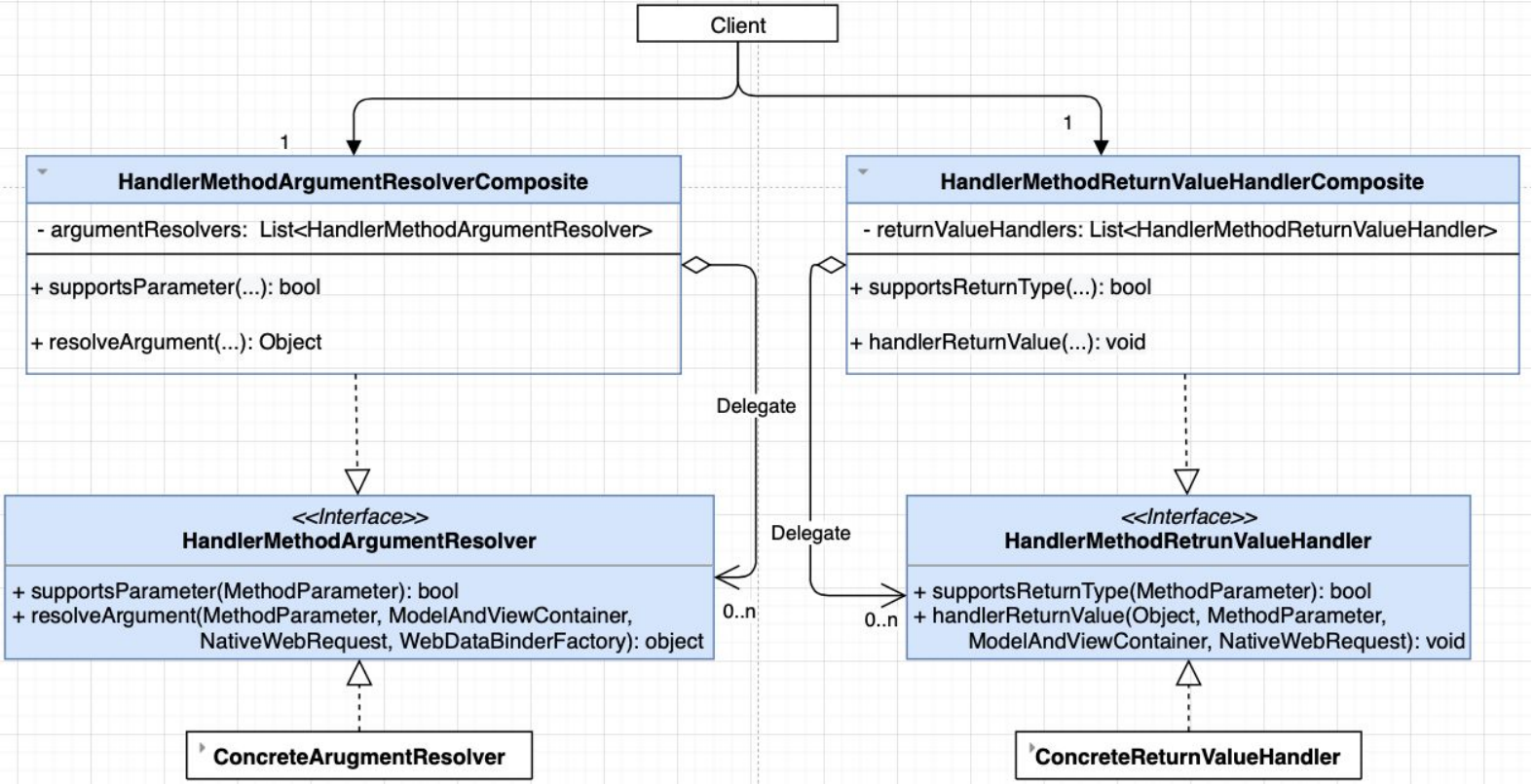
Problem

- RequestMappingHandlerAdapter has to be modified as more annotations are introduced
- Custom annotations and corresponding components aren't allowed

Intent

- Provide well-defined interfaces of resolving arguments and handling return values
- RequestMappingHandlerAdapter is programmed to those interfaces

Composite Pattern



Source Code

26 kinds of argument resolvers to support a variety of controller augments !!

Client - InvocableHandlerMethod#getMethodArgumentValues

```
for (int i = 0; i < parameters.length; i++) {
    MethodParameter parameter = parameters[i];

    if (!this.resolvers.supportsParameter(parameter)) {
        throw new IllegalStateException(formatArgumentError(parameter, "No suitable resolver"));
    }
    args[i] = this.resolvers.resolveArgument(parameter, mavContainer, request, this.dataBinderFactory);
}
return args;
```

HandlerMethodArgumentResolverComposite

```
public Object resolveArgument(MethodParameter parameter, @Nullable ModelAndViewContainer mavContainer,
        NativeWebRequest webRequest, @Nullable WebDataBinderFactory binderFactory) throws Exception {

    HandlerMethodArgumentResolver resolver = getArgumentResolver(parameter);
    return resolver.resolveArgument(parameter, mavContainer, webRequest, binderFactory);
}
private HandlerMethodArgumentResolver getArgumentResolver(MethodParameter parameter) {
    HandlerMethodArgumentResolver result = this.argumentResolverCache.get(parameter);
    if (result == null) {
        for (HandlerMethodArgumentResolver resolver : this.argumentResolvers) {
            if (resolver.supportsParameter(parameter)) {
                result = resolver;
                this.argumentResolverCache.put(parameter, result);
                break;
            }
        }
    }
    return result;
}
```

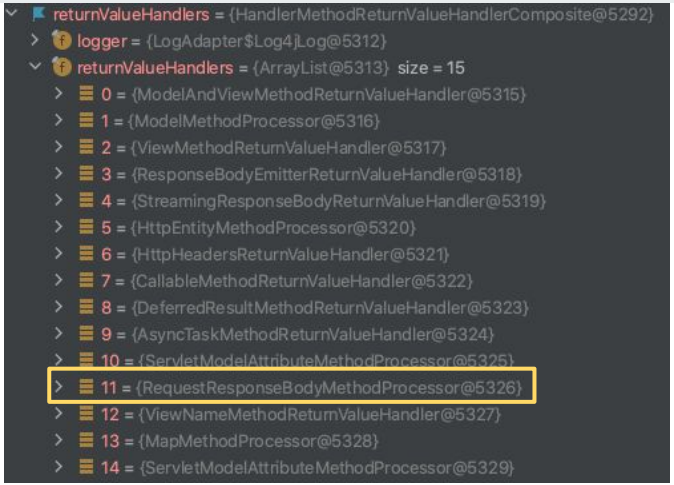
Put pairs of a parameter and an argument resolver the into cache to avoid repeated linear search

Source Code

15 kinds of return value handlers to support a variety of controller return value!!

ServletInvocableHandlerMethod#invokeAndHandle

```
public void invokeAndHandle(ServletWebRequest webRequest, ModelAndViewContainer mavContainer,
    Object... providedArgs) throws Exception {

    Object returnValue = invokeForRequest(webRequest, mavContainer, providedArgs);

    // if modelAndView isn't returned, then return

    this.returnValueHandlers.handleReturnValue(
        returnValue, getReturnValueType(returnValue), mavContainer, webRequest);
}
```



HandlerMethodReturnValueHandlerComposite

```
public void handleReturnValue(@Nullable Object returnValue, MethodParameter returnType,
        ModelAndViewContainer mavContainer, NativeWebRequest webRequest) throws Exception {

    HandlerMethodReturnValueHandler handler = selectHandler(returnValue, returnType);
    handler.handleReturnValue(returnValue, returnType, mavContainer, webRequest);
}

private HandlerMethodReturnValueHandler selectHandler(@Nullable Object value, MethodParameter returnType) {
    for (HandlerMethodReturnValueHandler handler : this.returnValueHandlers) {
        if (handler.supportsReturnType(returnType)) {
            return handler;
        }
    }
    return null;
}
```

RequestResponseBodyMethodProcessor

Controller

```
@PostMapping(value = "/content", produces = MediaType.APPLICATION_JSON_VALUE)
@ResponseBody
public ResponseTransfer postResponseJsonContent(
  @RequestBody LoginForm loginForm) {
    return new ResponseTransfer("JSON Content!");
}
```

```
@PostMapping(value = "/content", produces = MediaType.APPLICATION_XML_VALUE)
@ResponseBody
public ResponseTransfer postResponseXmlContent(
  @RequestBody LoginForm loginForm) {
    return new ResponseTransfer("XML Content!");
}
```

Request

```
curl -i \
-H "Accept: application/json" \
-H "Content-Type:application/json" \
-X POST --data
  '{"username": "johnny", "password": "password"}' "https://localhost:8080/.../content"
```

```
curl -i \
-H "Accept: application/xml" \
-H "Content-Type:application/json" \
-X POST --data
  '{"username": "johnny", "password": "password"}' "https://localhost:8080/.../content"
```

Response

```
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 20 Feb 2020 19:43:06 GMT

{"text":"JSON Content!"}
```

```
HTTP/1.1 200
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Thu, 20 Feb 2020 19:43:19 GMT

<ResponseTransfer><text>XML Content!</text></ResponseTransfer>
```

```
public class ResponseTransfer {
    private String text;

    // standard getters/setters
}
```

Problem

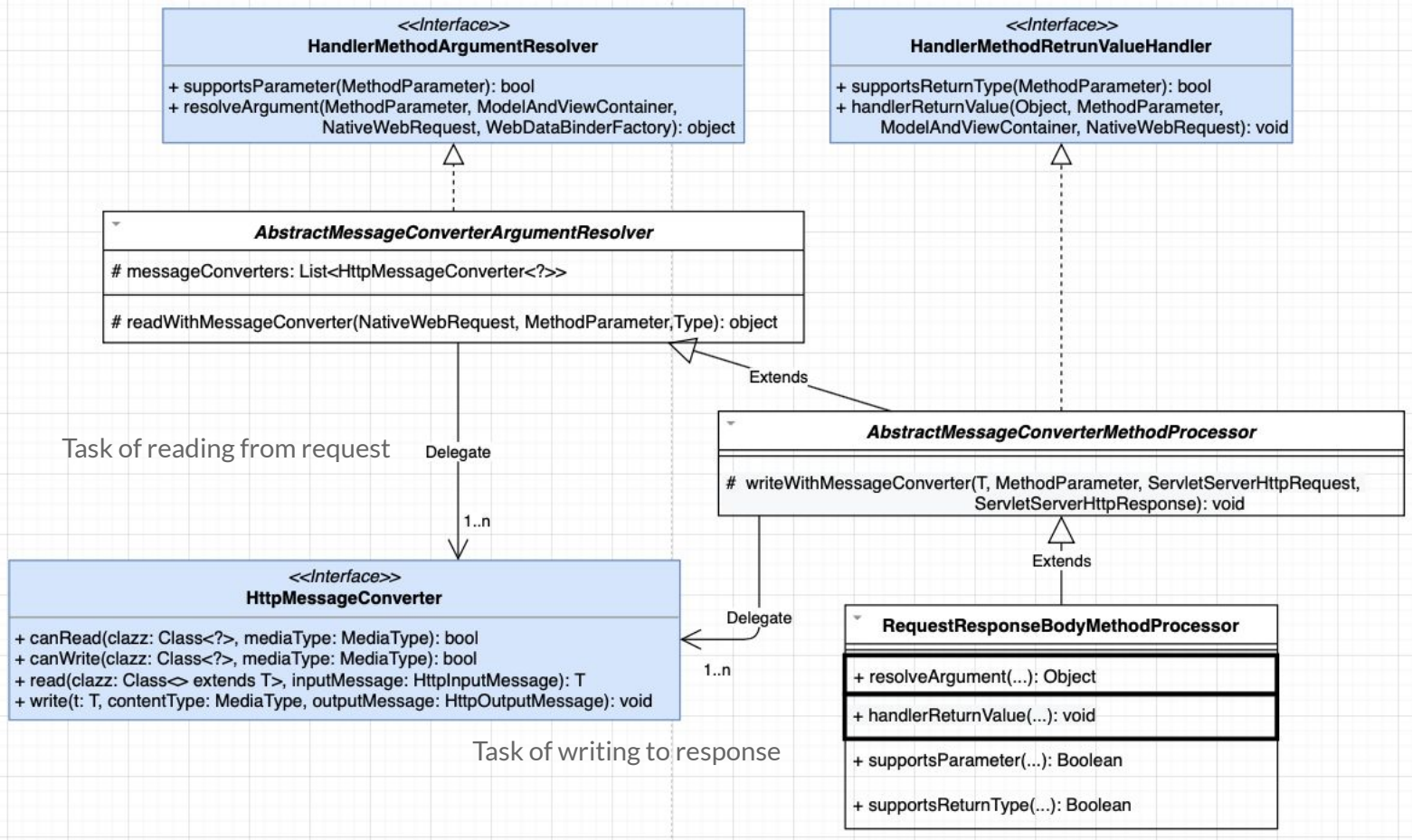
Body of requests and responses can be represented as various data types

Intent

- Delegate the task of data conversions to a family of algorithms (e.g., XML, JSON)
- Strategy lets the algorithms vary independently from clients that use it

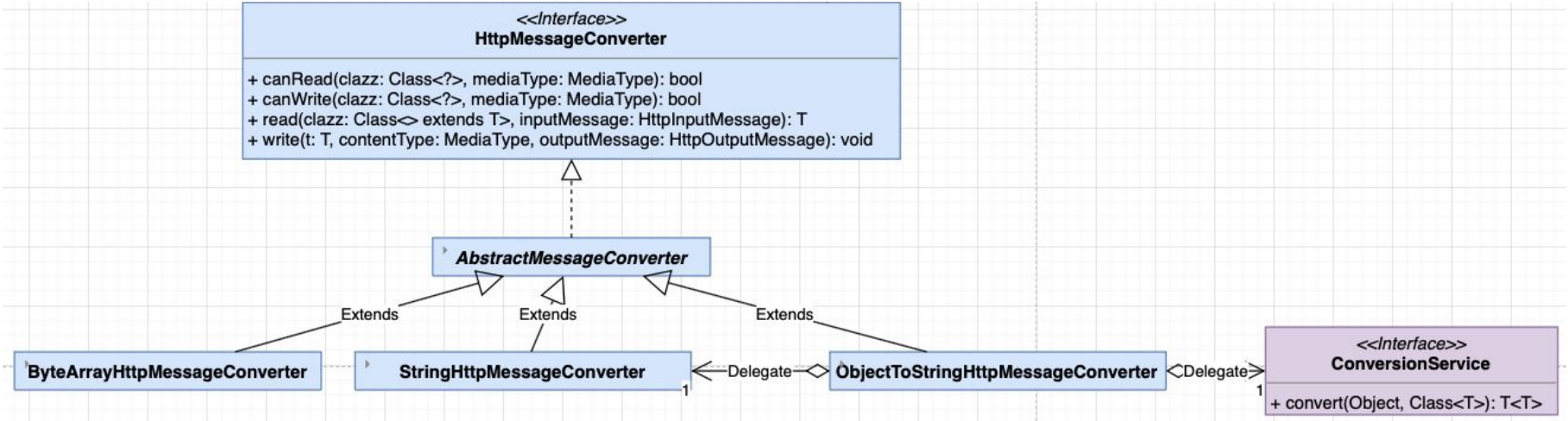
Ref: https://www.baeldung.com/spring-request-response-body

RequestResponseBodyMethodProcessor



Strategy Pattern

- Abstract common behaviors (e.g., read, write) of each encapsulated conversion algorithms
- Provide an interface for clients



input message -> byte array

byte array -> output message

Given Charset in HTTP request/response

input message -> string

string -> output message

With the help of StringHttpMessageConverter, ConversionService

input stream deserialization

target object serialization

Source code

AbstractMessageConverterMethodArgumentResolver

```java
public AbstractMessageConverterMethodArgumentResolver(List<HttpMessageConverter<?>> converters,
                                                      @Nullable List<Object> requestResponseBodyAdvice) {

    Assert.notEmpty(converters, "'messageConverters' must not be empty");
    this.messageConverters = converters;
    // .. other settings
}
```

Set HttpMessageConverter through a constructor (from RequestMappingHandlerAdapter)

```java
protected <T> Object readWithMessageConverters(HttpInputMessage inputMessage, MethodParameter parameter,
        Type targetType) throws IOException, HttpMediaTypeNotSupportedException, HttpMessageNotReadableException {

    // get contentType (type specified in HTTP header)
    // get targetClass
    // get contextClass

    for (HttpMessageConverter<?> converter : this.messageConverters) {
        Class<HttpMessageConverter<?>> converterType = (Class<HttpMessageConverter<?>>) converter.getClass();
        GenericHttpMessageConverter<?> genericConverter =
                (converter instanceof GenericHttpMessageConverter ? (GenericHttpMessageConverter<?>) converter : null);
        if (genericConverter != null ? genericConverter.canRead(targetType, contextClass, contentType) :
                (targetClass != null && converter.canRead(targetClass, contentType))) {
            if (message.hasBody()) {
                HttpInputMessage msgToUse =
                        getAdvice().beforeBodyRead(message, parameter, targetType, converterType);
                body = (genericConverter != null ? genericConverter.read(targetType, contextClass, msgToUse) :
                        ((HttpMessageConverter<T>) converter).read(targetClass, msgToUse));
                body = getAdvice().afterBodyRead(body, msgToUse, parameter, targetType, converterType);
            }
            else {
                body = getAdvice().handleEmptyBody(null, message, parameter, targetType, converterType);
            }
            break;
        }
    }
    return body;
}
```

GenericHttpMessageConverter: specialization of HttpMessageConverter, which can **convert a HTTP request into a target object with specified generic type**

- contentType: source type
- targetClass: target type

Delegate to a converter

Name-based Argument Resolver

```
@RequestMapping("/hello/{id}")
public String getDetails(@PathVariable(value="id") String id,
@RequestParam(value="param1", required=true) String param1,
@RequestParam(value="param2", required=false) String param2){
.......
}
```

Example

*other name-based argument resolvers: {ServletCookieValue, ExpressionValue, SessionAttribute, MatrixVariable, RequestHeader}MethodArgumentResolver

@PathVariable

```
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface PathVariable {

    @AliasFor("name")
    String value() default "";

    @AliasFor("value")
    String name() default "";

    boolean required() default true;
}
```

@RequestAttribute

```
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface RequestAttribute {

    @AliasFor("name")
    String value() default "";

    @AliasFor("value")
    String name() default "";

    boolean required() default true;
}
```

@RequestParam

```
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface RequestParam {

    @AliasFor("name")
    String value() default "";

    @AliasFor("value")
    String name() default "";

    boolean required() default true;

    String defaultValue() default ValueConstants.DEFAULT_NONE;
}
```

Problem

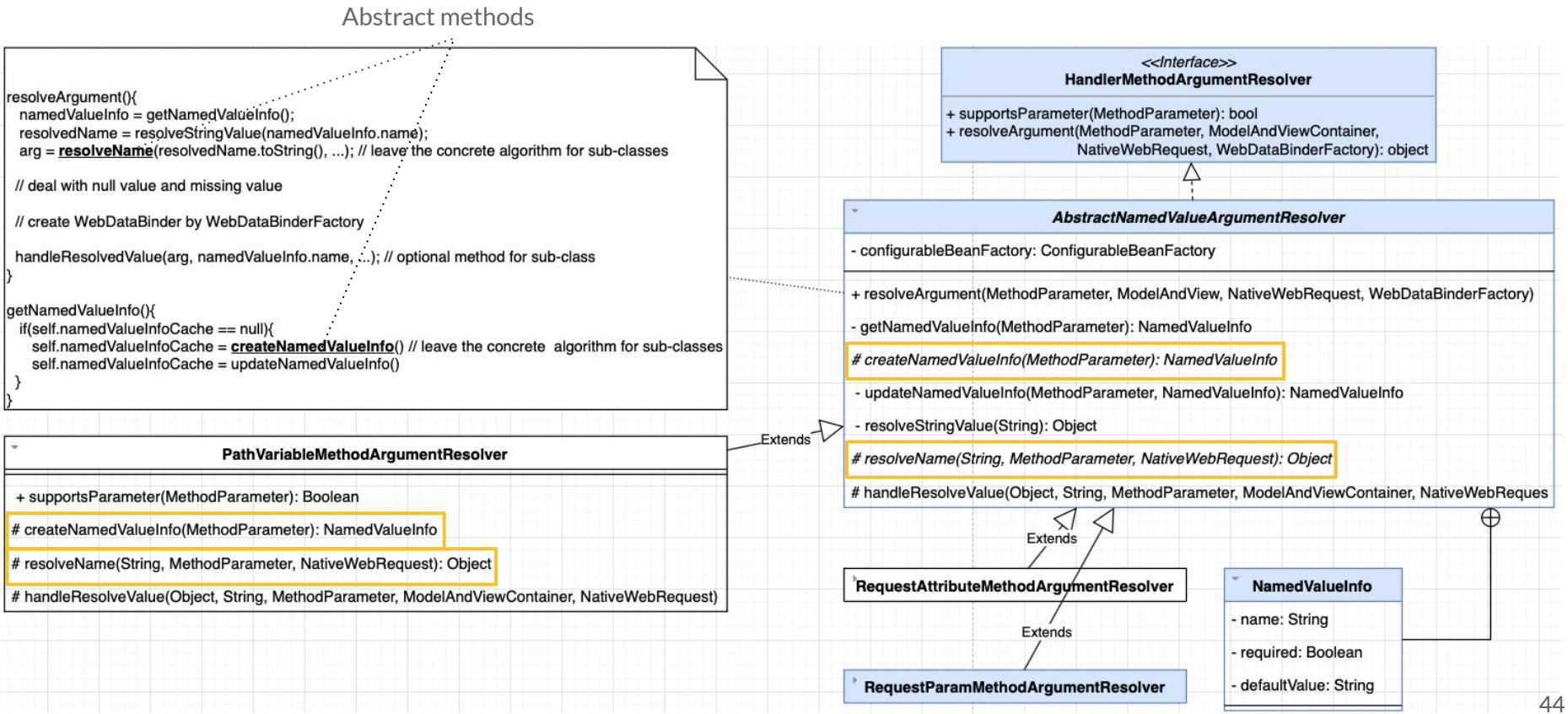
Part of the code of argument resolvers are duplicated, **they share the same skeleton of an algorithm, but vary in how to get name, value, and required from a request**

Intent

Abstract the skeleton of an algorithm in superclass, and leave the core logic in subclasses

Template Method Pattern

The skeleton of the algorithm is defined in AbstractNamedValueArgumentResolver



Source Code

AbstractNamedValueMethodArgumentResolver

```
public final Object resolveArgument(MethodParameter parameter, @Nullable ModelAndViewContainer mavContainer,
        NativeWebRequest webRequest, @Nullable WebDataBinderFactory binderFactory) throws Exception {

    NamedValueInfo namedValueInfo = getNamedValueInfo(parameter);
    MethodParameter nestedParameter = parameter.nestedIfOptional();

    Object resolvedName = resolveStringValue(namedValueInfo.name);

    Object arg = resolveName(resolvedName.toString(), nestedParameter, webRequest);
    if (arg == null) {
        if (namedValueInfo.defaultValue != null) {
            arg = resolveStringValue(namedValueInfo.defaultValue);
        }
        else if (namedValueInfo.required && !nestedParameter.isOptional()) {
            handleMissingValue(namedValueInfo.name, nestedParameter, webRequest);
        }
        arg = handleNullValue(namedValueInfo.name, arg, nestedParameter.getNestedParameterType());
    }
    else if ("".equals(arg) && namedValueInfo.defaultValue != null) {
        arg = resolveStringValue(namedValueInfo.defaultValue);
    }
    Loading...
    if (binderFactory != null) {
        WebDataBinder binder = binderFactory.createBinder(webRequest, null, namedValueInfo.name);
        arg = binder.convertIfNecessary(arg, parameter.getParameterType(), parameter);

    }

    handleResolvedValue(arg, namedValueInfo.name, parameter, mavContainer, webRequest);

    return arg;
}
private NamedValueInfo getNamedValueInfo(MethodParameter parameter) {
    NamedValueInfo namedValueInfo = this.namedValueInfoCache.get(parameter);
    if (namedValueInfo == null) {
        namedValueInfo = createNamedValueInfo(parameter);
        namedValueInfo = updateNamedValueInfo(parameter, namedValueInfo);
        this.namedValueInfoCache.put(parameter, namedValueInfo);
    }
    return namedValueInfo;
}
```

Extract namedValueInfo from a parameter

Resolve the name from namedValueInfo, which might contain placeholder, or regular expression

Resolve the processed name and get an input value from HTTP request (e.g., URL, attribute, parameter, …)

Handle missing value and null value

Bind the input value (i.e., argument) with the name from namedValueInfo

Empty method

Leave the namedValueInfo creation for a sub-class

```
protected abstract NamedValueInfo createNamedValueInfo(MethodParameter parameter);
protected abstract Object resolveName(String name, MethodParameter parameter, NativeWebRequest request);
```

Source Code

PathVariableMethodArgumentResolver

```java
@Override
protected NamedValueInfo createNamedValueInfo(MethodParameter parameter) {
    PathVariable ann = parameter.getParameterAnnotation(PathVariable.class);
    return new PathVariableNamedValueInfo(ann);
}

@Override
protected Object resolveName(String name, MethodParameter parameter, NativeWebRequest request) throws Exception {
    Map<String, String> uriTemplateVars = (Map<String, String>) request.getAttribute(
            HandlerMapping.URI_TEMPLATE_VARIABLES_ATTRIBUTE, RequestAttributes.SCOPE_REQUEST);
    return (uriTemplateVars != null ? uriTemplateVars.get(name) : null);
}
```

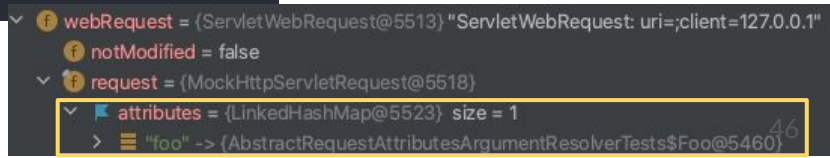
@RequestMapping(“/webApp/{name}”)

http://localhost:8080/webApp/**value**

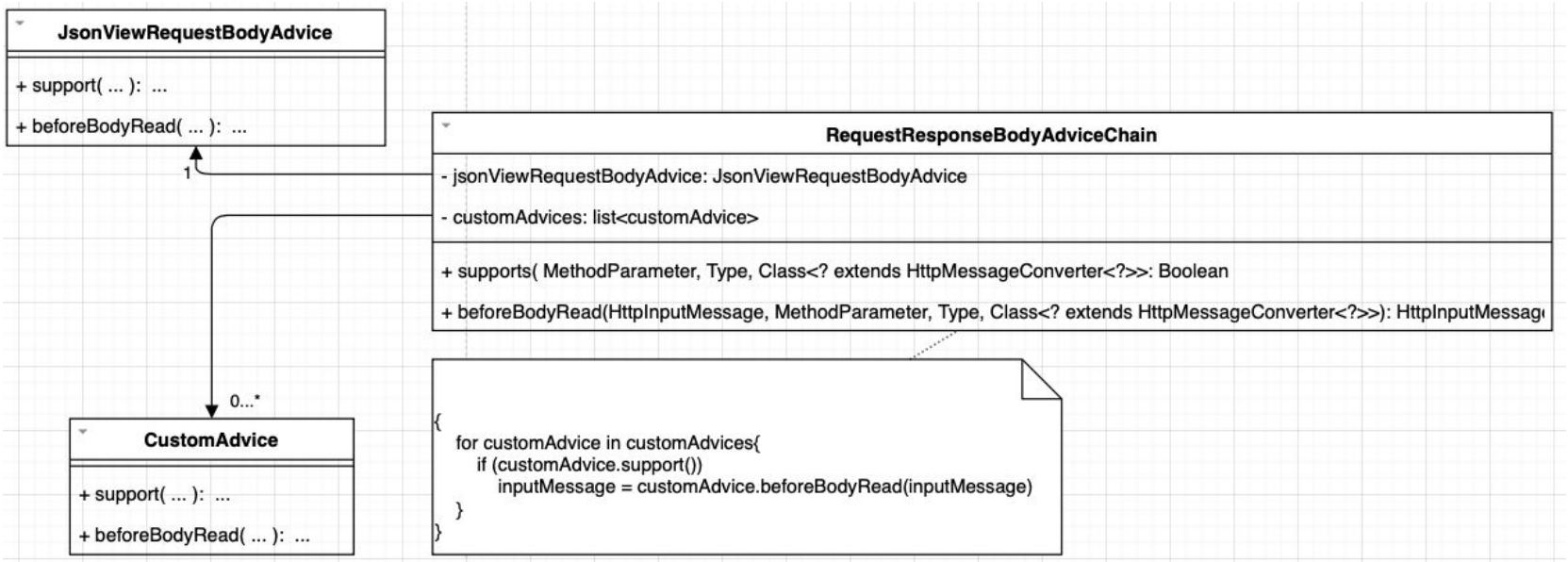
RequestAttributeMethodArgumentResolver

```java
@Override
protected NamedValueInfo createNamedValueInfo(MethodParameter parameter) {
    RequestAttribute ann = parameter.getParameterAnnotation(RequestAttribute.class);
    return new NamedValueInfo(ann.name(), ann.required(), ValueConstants.DEFAULT_NONE);
}

@Override
protected Object resolveName(String name, MethodParameter parameter, NativeWebRequest request){
    return request.getAttribute(name, RequestAttributes.SCOPE_REQUEST);
```

request.setAttribute(“foo”, foo, ...)



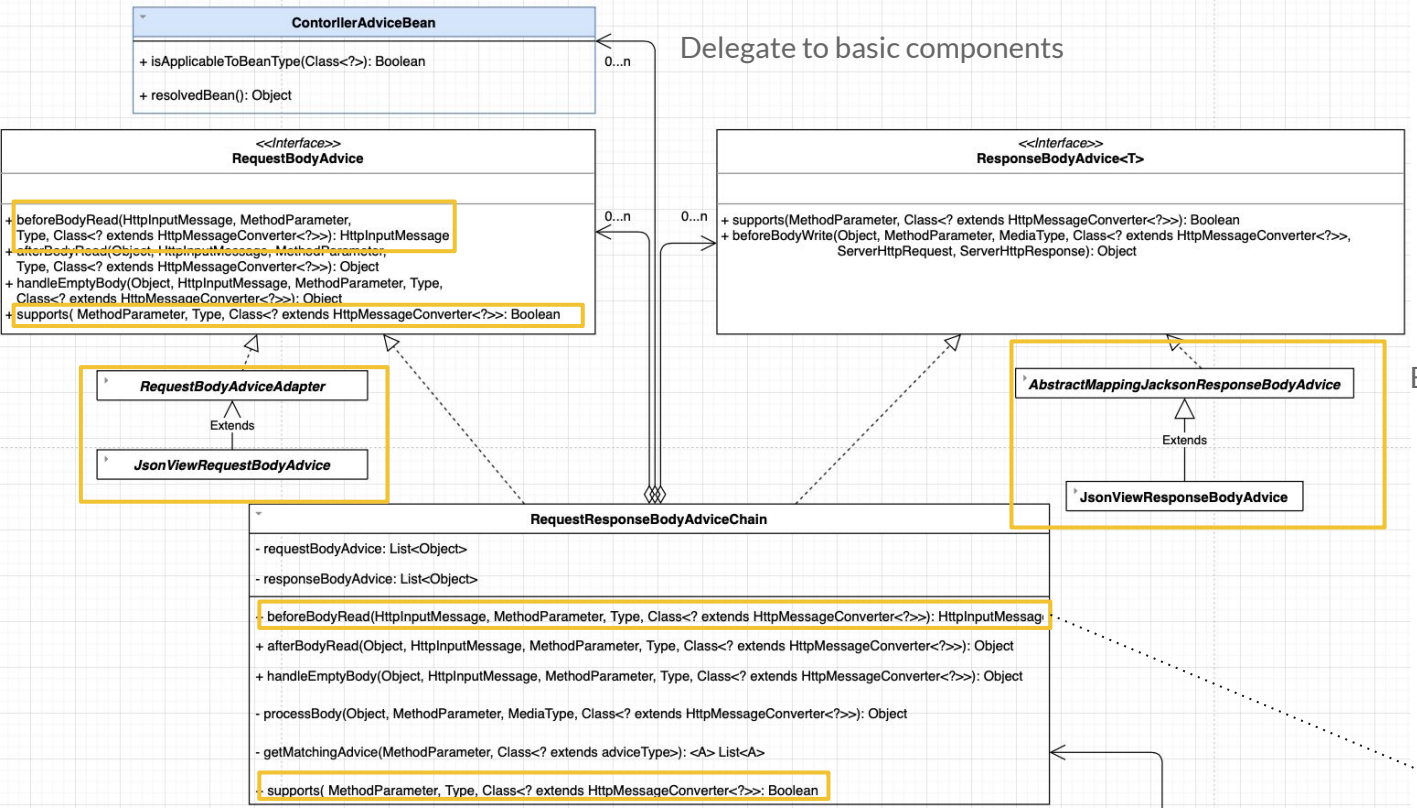
RequestResponseBodyAdvice



Problem

Intent

Composite / Intercepting Filter Pattern



- Pre/post processing for requests and responses
- Go through each filter (i.e., advice)

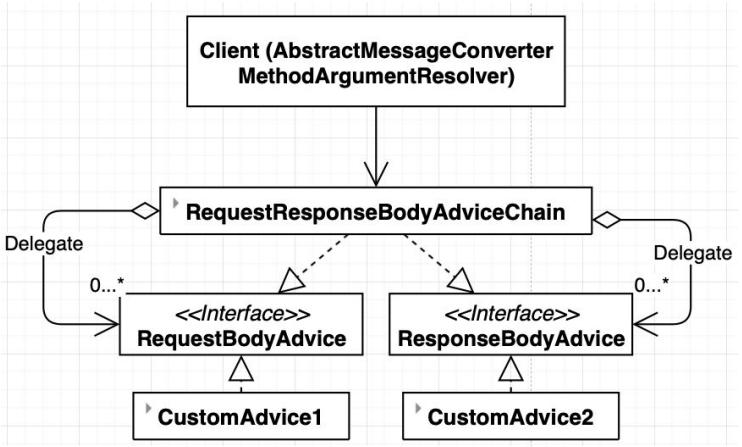
```
beforeBodyRead(){
   for(RequestBodyAdvice advice: requestBodyAdvice){
      if(advice.supports()){
         request = advice.beforeBodyRead(request, ...)
      }
   }
   return request
}
```

Source code

AbstractMessageConverterMethodArgumentResolver#readWithMessageConverters

```
try {
    // message: input message
    for (HttpMessageConverter<?> converter : this.messageConverters) {
        // get converterType
        // get available genericConverter
        if (message.hasBody()) {
            HttpInputMessage msgToUse =
                    getAdvice().beforeBodyRead(message, parameter, targetType, converterType);
            body = genericConverter.read(targetType, contextClass, msgToUse) ;
            body = getAdvice().afterBodyRead(body, msgToUse, parameter, targetType, converterType);
        }
        else {
            body = getAdvice().handleEmptyBody(null, message, parameter, targetType, converterType);
        }
        break;
    }
}
```

RequestResponseBodyAdviceChain




```
private final List<Object> requestBodyAdvice = new ArrayList<>(4);
private final List<Object> responseBodyAdvice = new ArrayList<>(4);
```

```
for (RequestBodyAdvice advice : getMatchingAdvice(parameter, RequestBodyAdvice.class)) {
    if (advice.supports(parameter, targetType, converterType)) {
        request = advice.beforeBodyRead(request, parameter, targetType, converterType);
    }
}
return request;
```

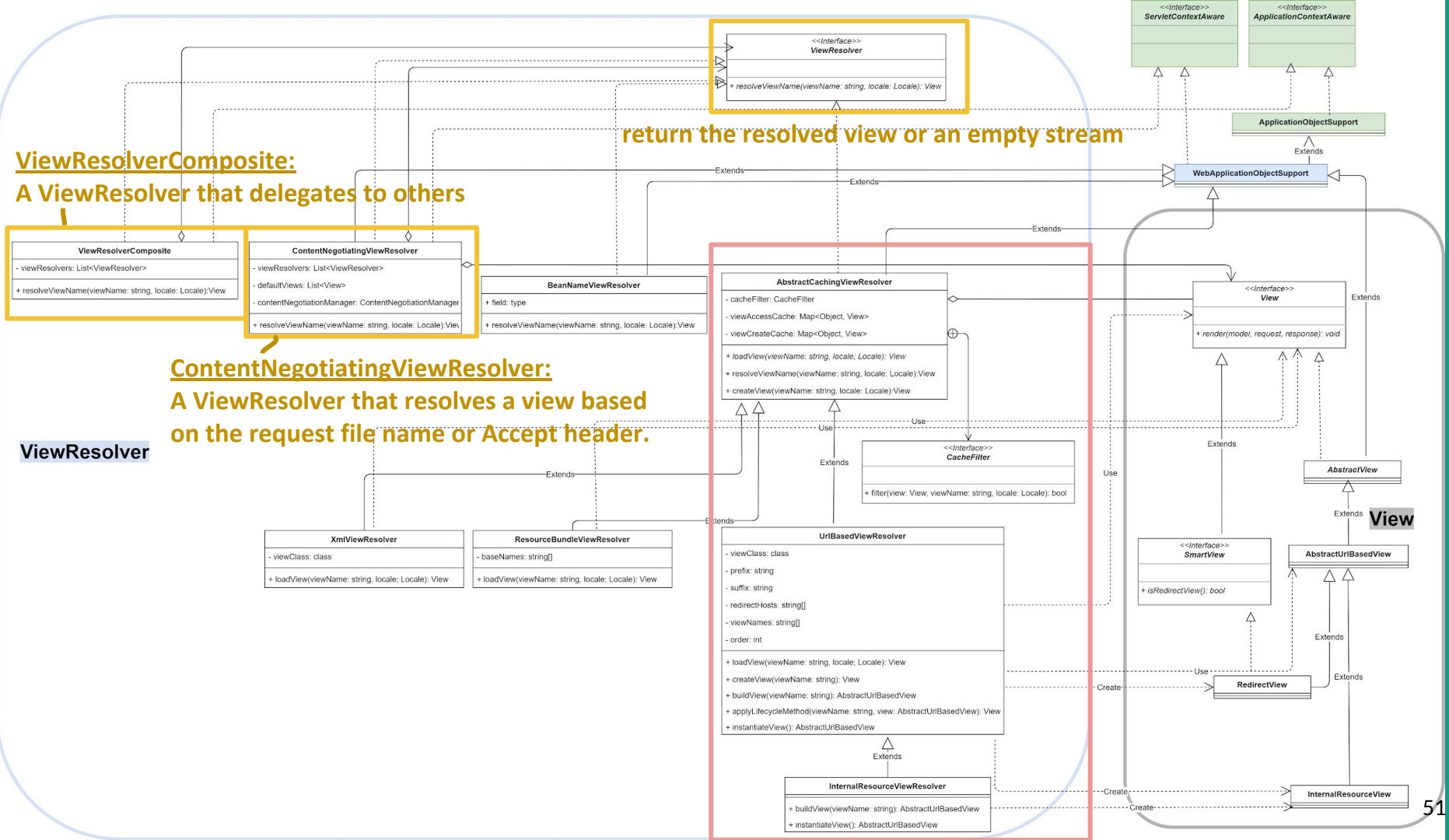
```
for (RequestBodyAdvice advice : getMatchingAdvice(parameter, RequestBodyAdvice.class)) {
    if (advice.supports(parameter, targetType, converterType)) {
        body = advice.afterBodyRead(body, inputMessage, parameter, targetType, converterType);
    }
}
return body;
```

RequestResponseBodyAdviceChain

Each basic advice is applied to a request/body

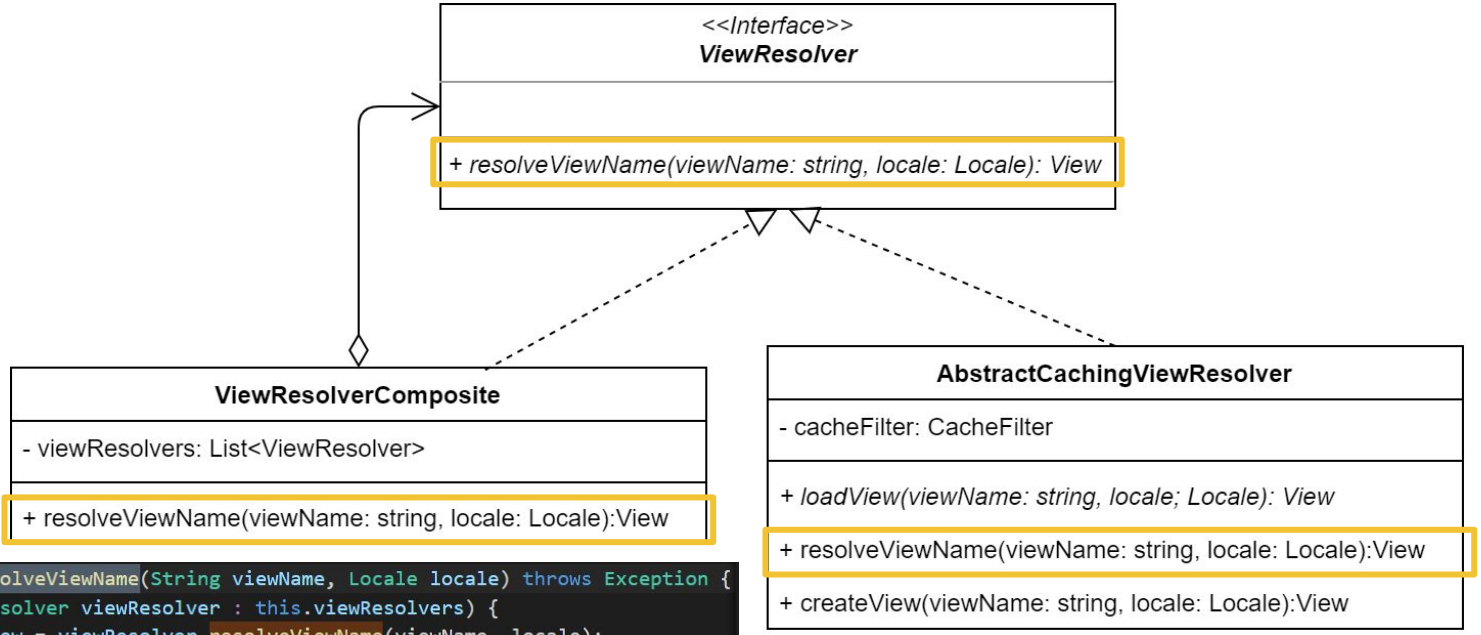
ViewResolver

Presenter: 王郁婷


ViewResolver
+ resolveViewName(viewName: string, locale: Locale): View
return the resolved view or an empty stream
ViewResolverComposite:
A ViewResolver that delegates to others
ViewResolverComposite
- viewResolvers: List<ViewResolver>
+ resolveViewName(viewName: string, locale: Locale):View
ContentNegotiatingViewResolver
- viewResolvers: List<ViewResolver>
- defaultViews: List<View>
- contentNegotiationManager: ContentNegotiationManager
+ resolveViewName(viewName: string, locale: Locale):View
ContentNegotiatingViewResolver:
A ViewResolver that resolves a view based on the request file name or Accept header.
ViewResolver
BeanNameViewResolver
+ field: type
+ resolveViewName(viewName: string, locale: Locale):View
AbstractCachingViewResolver
- cacheFilter: CacheFilter
- viewAccessCache: Map<Object, View>
- viewCreateCache: Map<Object, View>
+ loadView(viewName: string, locale; Locale): View
+ resolveViewName(viewName: string, locale; Locale):View
+ createView(viewName: string, locale: Locale):View
CacheFilter
+ filter(view: View, viewName: string, locale: Locale): bool
XmlViewResolver
- viewClass: class
+ loadView(viewName: string, locale; Locale): View
ResourceBundleViewResolver
- baseNames: string[]
+ loadView(viewName: string, locale; Locale): View
UrlBasedViewResolver
- viewClass: class
- prefix: string
- suffix: string
- redirectHosts: string[]
- viewNames: string[]
- order: int
+ loadView(viewName: string, locale; Locale): View
+ createView(viewName: string): View
+ buildView(viewName: string): AbstractUrlBasedView
+ applyLifecycleMethod(viewName: string, view: AbstractUrlBasedView): View
+ instantiateView(): AbstractUrlBasedView
InternalResourceViewResolver
+ buildView(viewName: string): AbstractUrlBasedView
+ instantiateView(): AbstractUrlBasedView
View
View
+ render(model, request, response): void
SmartView
+ isRedirectView(): bool
AbstractView
AbstractUrlBasedView
RedirectView
InternalResourceView
ServletContextAware
ApplicationContextAware
ApplicationObjectSupport
WebApplicationObjectSupport
Extends
Use
Create

ViewResolverComposite - Composite Pattern

Abstract ***resolveViewName in ViewResolver interface*** and implement the delegation process in **resolveViewName** of ViewResolverComposite so dispatcherServlet could treat specific viewResolvers and composite uniformly




```java
public View resolveViewName(String viewName, Locale locale) throws Exception {
    for (ViewResolver viewResolver : this.viewResolvers) {
        View view = viewResolver.resolveViewName(viewName, locale);
        if (view != null) {
            return view;
        }
    }
    return null;
}
```

ContentNegotiatingViewResolver

getCandidateView:

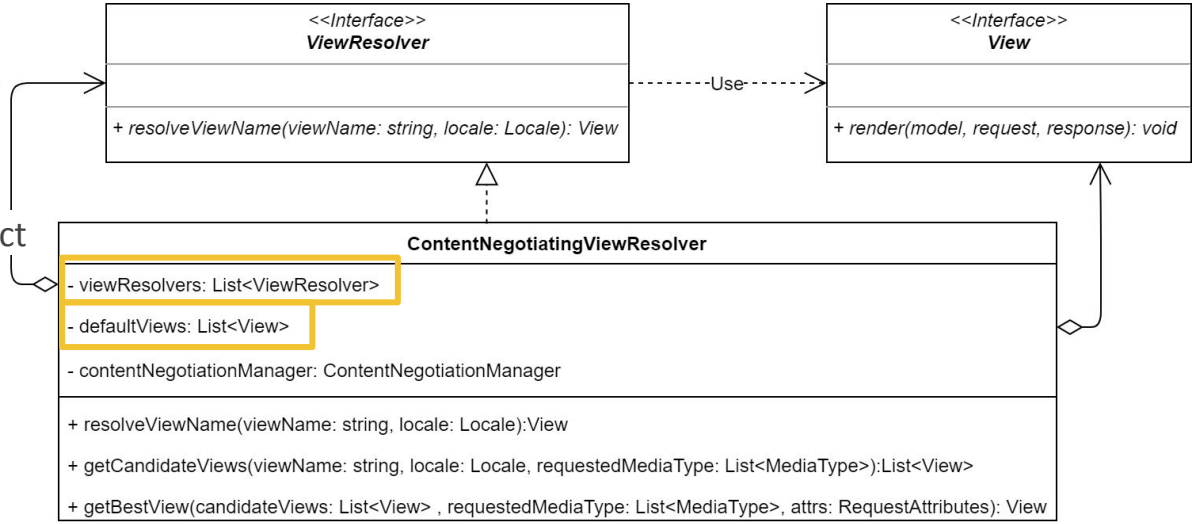
- **delegates to registered viewResolvers and get all candidateViews**

getBestView:

- uses the **requested media type** to select a suitable View for a request.

resolveViewName:

- **getBestView from all candidateViews** that resembles the representation requested by the client.

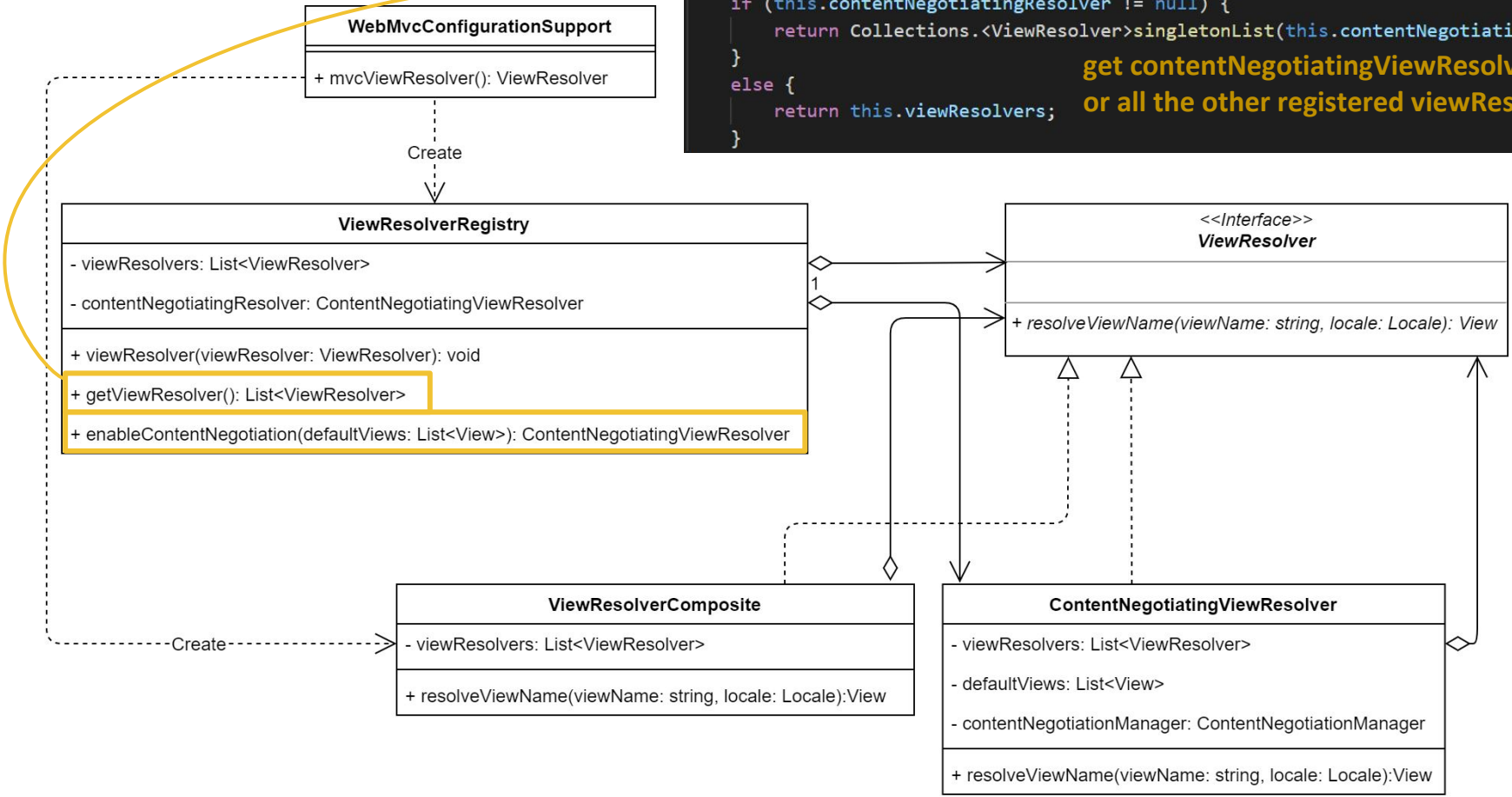



```java
public View resolveViewName(String viewName, Locale locale) throws Exception {
    RequestAttributes attrs = RequestContextHolder.getRequestAttributes();
    Assert.state(attrs instanceof ServletRequestAttributes, "No current ServletRequestAttributes");
    List<MediaType> requestedMediaTypes = getMediaTypes(((ServletRequestAttributes) attrs).getRequest());
    if (requestedMediaTypes != null) {
        List<View> candidateViews = getCandidateViews(viewName, locale, requestedMediaTypes);
        View bestView = getBestView(candidateViews, requestedMediaTypes, attrs);
        if (bestView != null) {
            return bestView;
```

ViewResolverRegistry

```
protected List<ViewResolver> getViewResolvers() {
    if (this.contentNegotiatingResolver != null) {
        return Collections.<ViewResolver>singletonList(this.contentNegotiatingResolver);
    }
    else {
        return this.viewResolvers;
    }
```

get contentNegotiatingViewResolver
or all the other registered viewResolvers



WebMvcConfigurationSupport

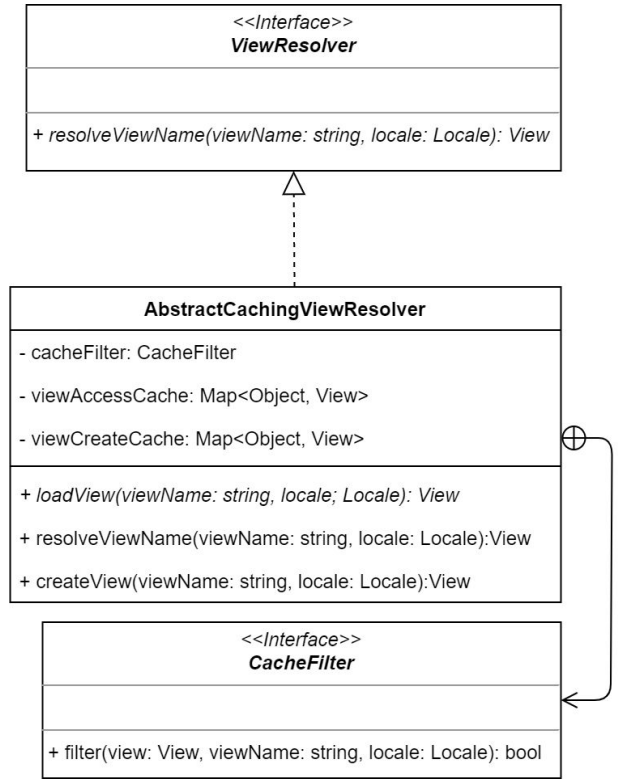
```
public ViewResolver mvcViewResolver(
        @Qualifier("mvcContentNegotiationManager") ContentNegotiationManager contentNegotiationManager) {
    ViewResolverRegistry registry =
            new ViewResolverRegistry(contentNegotiationManager, this.applicationContext);
    configureViewResolvers(registry);

    if (registry.getViewResolvers().isEmpty() && this.applicationContext != null) {
        String[] names = BeanFactoryUtils.beanNamesForTypeIncludingAncestors(
                this.applicationContext, ViewResolver.class, true, false);
        if (names.length == 1) {
            registry.getViewResolvers().add(new InternalResourceViewResolver());
        }
    }

    ViewResolverComposite composite = new ViewResolverComposite();
    composite.setOrder(registry.getOrder());
    composite.setViewResolvers(registry.getViewResolvers());
    if (this.applicationContext != null) {
        composite.setApplicationContext(this.applicationContext);
    }
    if (this.servletContext != null) {
        composite.setServletContext(this.servletContext);
    }
    return composite;
}
```

AbstractCachingViewResolver

Caches View objects once resolved ⟶ Solving view resolution performance problem

- cacheFilter: CacheFilter
- viewAccessCache: Map<Object, View>
- viewCreationCache: Map<Object, View>

```java
public View resolveViewName(String viewName, Locale locale) throws Exception {
    if (!isCache()) {
        return createView(viewName, locale);
    }
    else {
        Object cacheKey = getCacheKey(viewName, locale);
        View view = this.viewAccessCache.get(cacheKey);
        if (view == null) {
            synchronized (this.viewCreationCache) {
                view = this.viewCreationCache.get(cacheKey);
                if (view == null) {
                    // Ask the subclass to create the View object.
                    view = createView(viewName, locale);
                    if (view == null && this.cacheUnresolved) {
                        view = UNRESOLVED_VIEW;
                    }
                    if (view != null && this.cacheFilter.filter(view, viewName, locale)) {
                        this.viewAccessCache.put(cacheKey, view);
                        this.viewCreationCache.put(cacheKey, view);
                    }
```



Template Method Pattern

Abstract ***loadView*** method and call in **createView**

Subclasses should implement **loadView** to build the View object for a specific view name and locale

UrlBasedViewResolver.loadView

```
protected View loadView(String viewName, Locale locale) throws Exception {
    AbstractUrlBasedView view = buildView(viewName);
    View result = applyLifecycleMethods(viewName, view);
    return (view.checkResource(locale) ? result : null);
}
```

UrlBasedViewResolver

View names can either be resource URLs themselves, or get augmented by specified **prefix** and/or **suffix**.

createView:

- Check for **“redirect:”** / **“forward:”** prefix, if so, create RedirectView / InternalResourceView
- Else **fall back to superclass implementation: calling loadView()**

```
protected View createView(String viewName, Locale locale) throws Exception {
    if (!canHandle(viewName, locale)) {
        return null;
    }
```

If this resolver is not supposed to handle the given view, return null to pass on to the next resolver in the chain.

```
    if (viewName.startsWith(REDIRECT_URL_PREFIX)) {
        String redirectUrl = viewName.substring(REDIRECT_URL_PREFIX.length());
        RedirectView view = new RedirectView(redirectUrl,
                isRedirectContextRelative(), isRedirectHttp10Compatible());
        String[] hosts = getRedirectHosts();
        if (hosts != null) {
            view.setHosts(hosts);
        }
        return applyLifecycleMethods(REDIRECT_URL_PREFIX, view);
    }
```

Check for “redirect:” prefix

```
    if (viewName.startsWith(FORWARD_URL_PREFIX)) {
        String forwardUrl = viewName.substring(FORWARD_URL_PREFIX.length());
        InternalResourceView view = new InternalResourceView(forwardUrl);
        return applyLifecycleMethods(FORWARD_URL_PREFIX, view);
    }
```

Check for “forward:” prefix

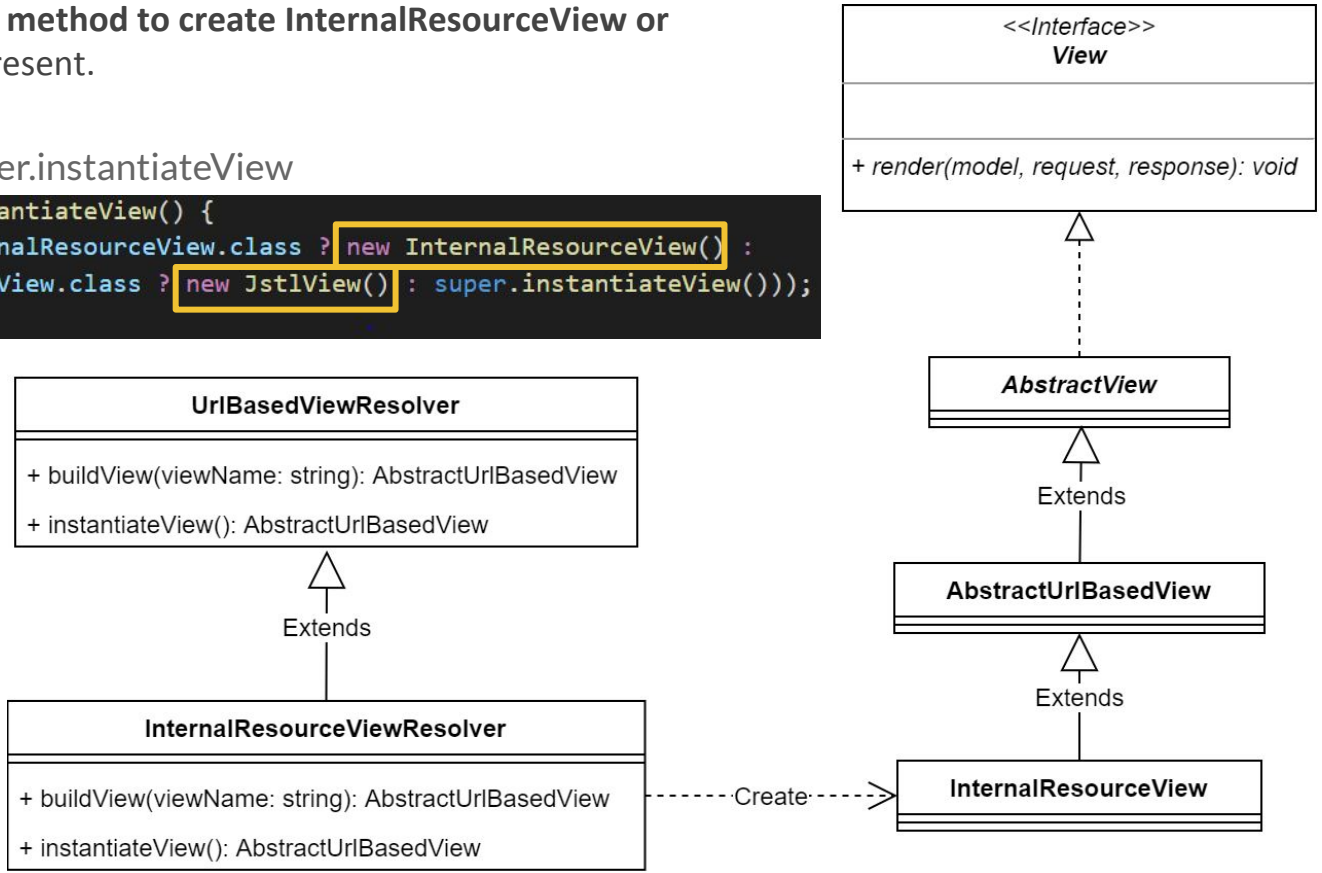
```
    return super.createView(viewName, locale);
}
```

InternalResourceViewResolver

Override instantiateView() method to create InternalResourceView or JstlView if the JSTL API is present.

InternalResourceViewResolver.instantiateView

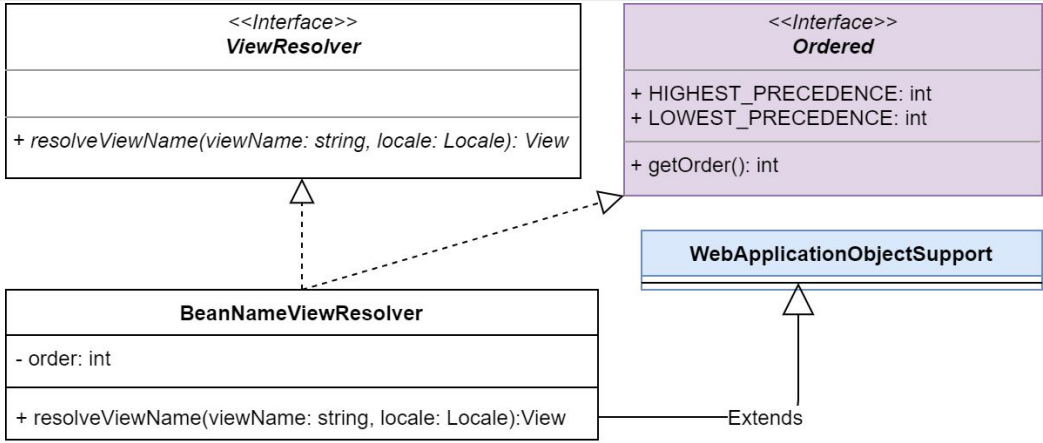
```
protected AbstractUrlBasedView instantiateView() {
    return (getViewClass() == InternalResourceView.class ? new InternalResourceView() :
            (getViewClass() == JstlView.class ? new JstlView() : super.instantiateView()));
}
```



BeanNameViewResolver

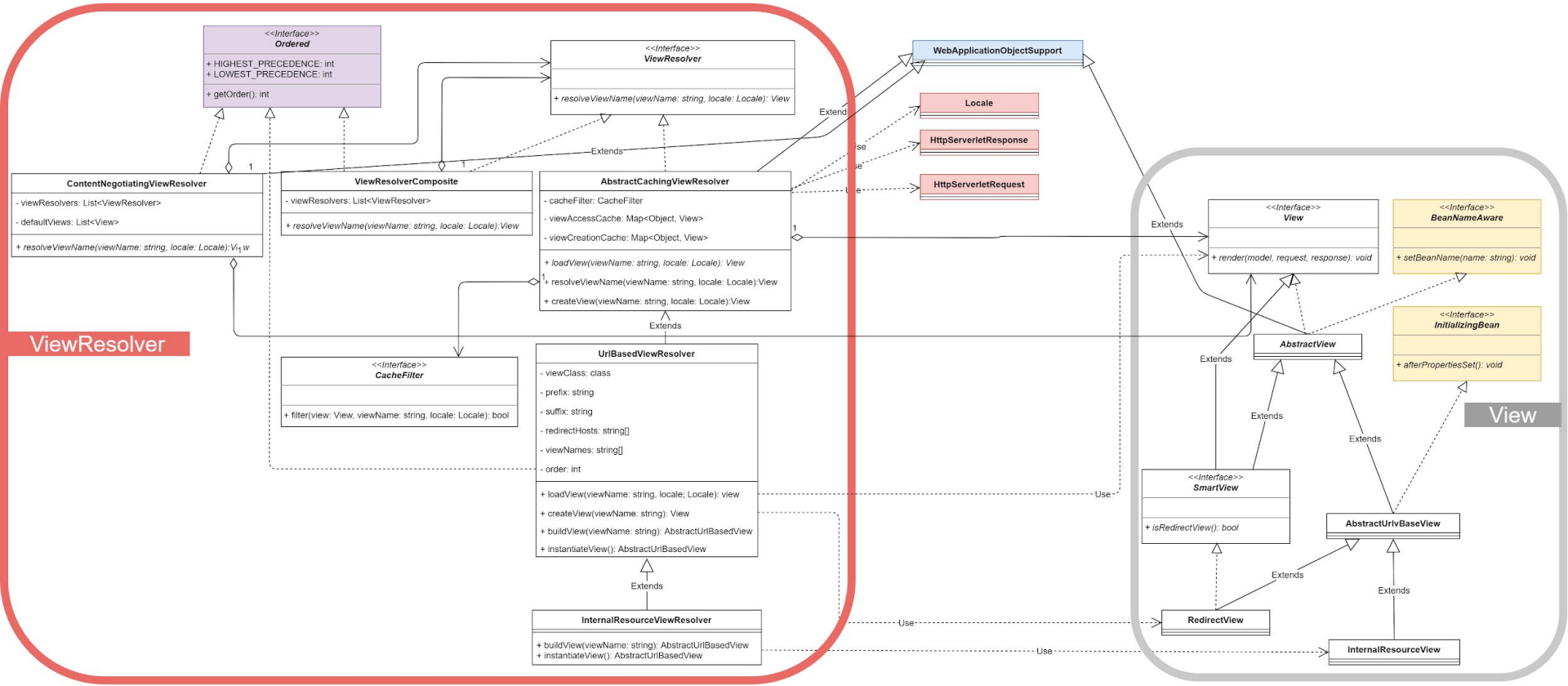
Interprets a view name as a bean name in the current application context.(extend from **WebApplicationObjectSupport**)

Implements the **Ordered** interface to allow for flexible participation in ViewResolver chaining.



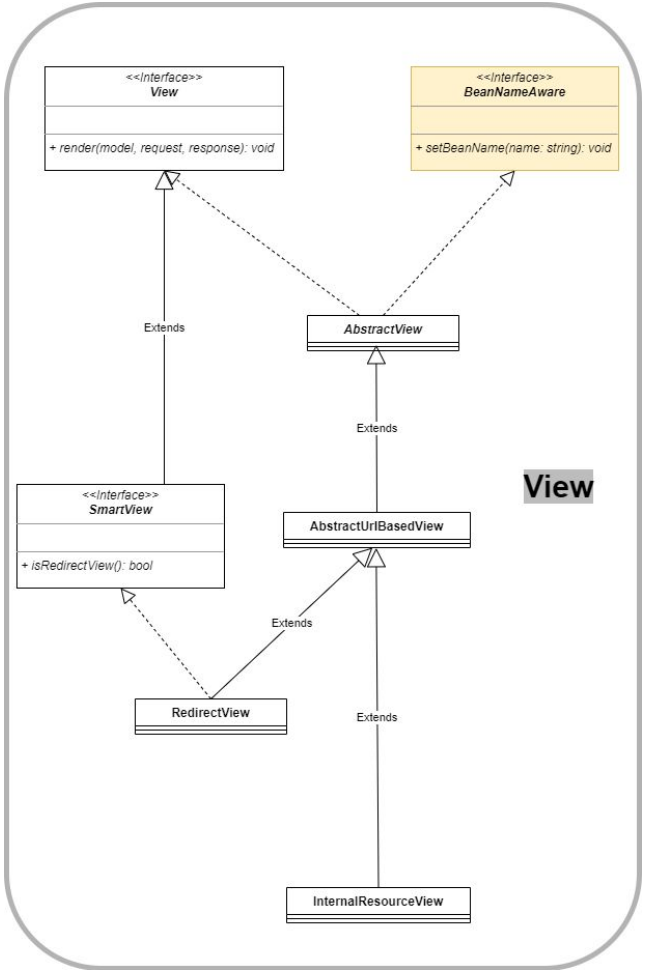

```java
public View resolveViewName(String viewName, Locale locale) throws BeansException {
    ApplicationContext context = obtainApplicationContext();
    if (!context.containsBean(viewName)) {
        return null;     → Allow for ViewResolver chaining
    }
    if (!context.isTypeMatch(viewName, View.class)) {
        if (logger.isDebugEnabled()) {
            logger.debug("Found bean named '" + viewName + "' but it does not implement View");
        }
        return null;     → Allow for ViewResolver chaining
    }
    return context.getBean(viewName, View.class);
}
```

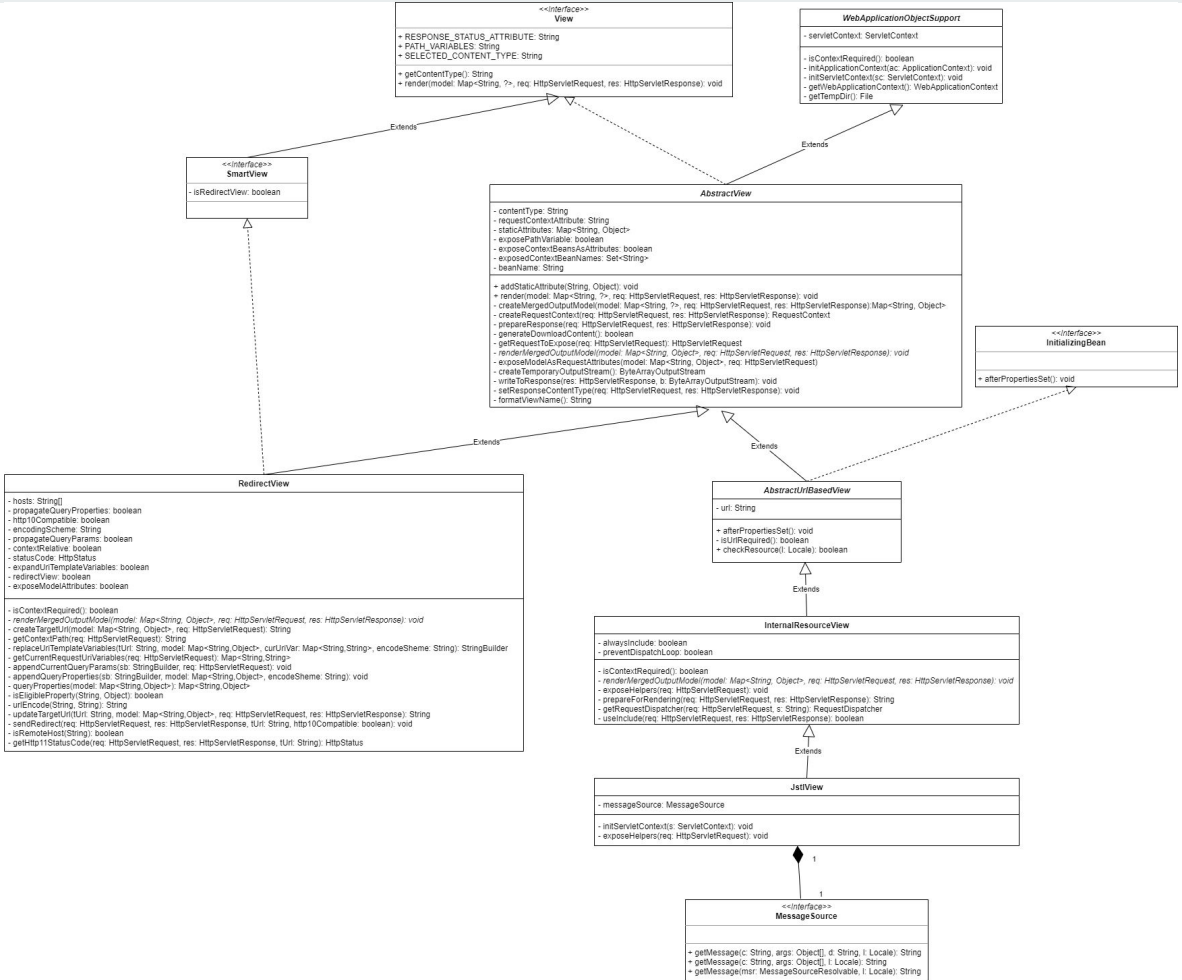
Implementation



View

Presenter: **陳熙**

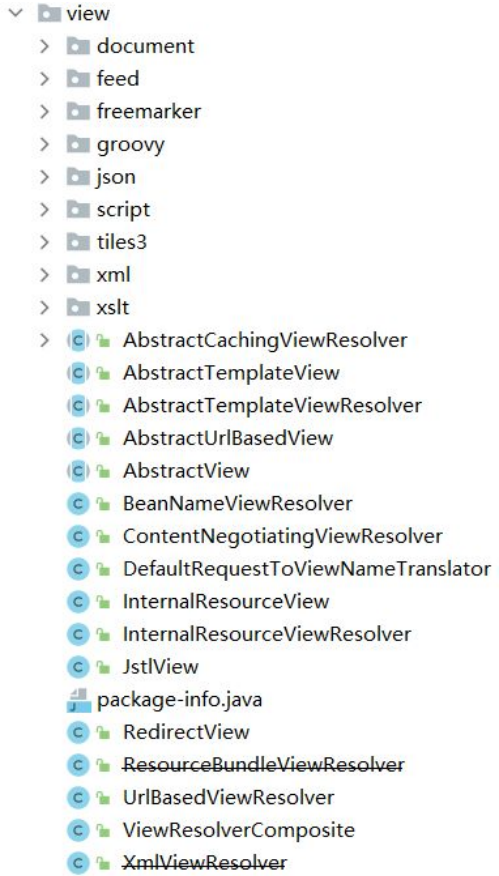

<<Interface>>
View
+ render(model, request, response): void
<<Interface>>
BeanNameAware
+ setBeanName(name: string): void
Extends
AbstractView
Extends
<<Interface>>
SmartView
+ isRedirectView(): bool
AbstractUrlBasedView
Extends
RedirectView
Extends
InternalResourceView
View


<<Interface>>
View
+ RESPONSE_STATUS_ATTRIBUTE: String
+ PATH_VARIABLES: String
+ SELECTED_CONTENT_TYPE: String
+ getContentType(): String
+ render(model: Map<String, ?>, req: HttpServletRequest, res: HttpServletResponse): void
WebApplicationObjectSupport
- servletContext: ServletContext
- isContextRequired(): boolean
- initApplicationContext(ac: ApplicationContext): void
- initServletContext(sc: ServletContext): void
- getWebApplicationContext(): WebApplicationContext
- getTempDir(): File
Extends
Extends
<<Interface>>
SmartView
- isRedirectView: boolean
AbstractView
- contentType: String
- requestContextAttribute: String
- staticAttributes: Map<String, Object>
- exposePathVariable: boolean
- exposeContextBeansAsAttributes: boolean
- exposedContextBeanNames: Set<String>
- beanName: String
+ addStaticAttribute(String, Object): void
+ render(model: Map<String, ?>, req: HttpServletRequest, res: HttpServletResponse): void
- createMergedOutputModel(model: Map<String, ?>, req: HttpServletRequest, res: HttpServletResponse):Map<String, Object>
- createRequestContext(req: HttpServletRequest, res: HttpServletResponse): RequestContext
- prepareResponse(req: HttpServletRequest, res: HttpServletResponse): void
- generateDownloadContent(): boolean
- getRequestToExpose(req: HttpServletRequest): HttpServletRequest
- renderMergedOutputModel(model: Map<String, Object>, req: HttpServletRequest, res: HttpServletResponse): void
- exposeModelAsRequestAttributes(model: Map<String, Object>, req: HttpServletRequest)
- createTemporaryOutputStream(): ByteArrayOutputStream
- writeToResponse(res: HttpServletResponse, b: ByteArrayOutputStream): void
- setResponseContentType(req: HttpServletRequest, res: HttpServletResponse): void
- formatViewName(): String
<<Interface>>
InitializingBean
+ afterPropertiesSet(): void
Extends
Extends
RedirectView
- hosts: String[]
- propagateQueryProperties: boolean
- http10Compatible: boolean
- encodingScheme: String
- propagateQueryParams: boolean
- contextRelative: boolean
- statusCode: HttpStatus
- expandUriTemplateVariables: boolean
- redirectView: boolean
- exposeModelAttributes: boolean
- isContextRequired(): boolean
- renderMergedOutputModel(model: Map<String, Object>, req: HttpServletRequest, res: HttpServletResponse): void
- createTargetUri(model: Map<String, Object>, req: HttpServletRequest): String
- getContextPath(req: HttpServletRequest): String
- replaceUriTemplateVariables(tUri: String, model: Map<String,Object>, curUriVar: Map<String,String>, encodeSheme: String): StringBuilder
- getCurrentRequestUriVariables(req: HttpServletRequest): Map<String,String>
- appendCurrentQueryParams(sb: StringBuilder, req: HttpServletRequest): void
- appendQueryProperties(sb: StringBuilder, model: Map<String,Object>, encodeSheme: String): void
- queryProperties(model: Map<String,Object>): Map<String,Object>
- isEligibleProperty(String, Object): boolean
- urlEncode(String, String): String
- updateTargetUri(tUri: String, model: Map<String,Object>, req: HttpServletRequest, res: HttpServletResponse): String
- sendRedirect(req: HttpServletRequest, res: HttpServletResponse, tUri: String, http10Compatible: boolean): void
- isRemoteHost(String): boolean
- getHttp11StatusCode(req: HttpServletRequest, res: HttpServletResponse, tUri: String): HttpStatus
AbstractUrlBasedView
- url: String
+ afterPropertiesSet(): void
- isUriRequired(): boolean
+ checkResource(l: Locale): boolean
Extends
InternalResourceView
- alwaysInclude: boolean
- preventDispatchLoop: boolean
- isContextRequired(): boolean
- renderMergedOutputModel(model: Map<String, Object>, req: HttpServletRequest, res: HttpServletResponse): void
- exposeHelpers(req: HttpServletRequest): void
- prepareForRendering(req: HttpServletRequest, res: HttpServletResponse): String
- getRequestDispatcher(req: HttpServletRequest, s: String): RequestDispatcher
- useInclude(req: HttpServletRequest, res: HttpServletResponse): boolean
Extends
JstlView
- messageSource: MessageSource
- initServletContext(s: ServletContext): void
- exposeHelpers(req: HttpServletRequest): void
1
1
<<Interface>>
MessageSource
+ getMessage(c: String, args: Object[], d: String, l: Locale): String
+ getMessage(c: String, args: Object[], l: Locale): String
+ getMessage(msr: MessageSourceResolvable, l: Locale): String

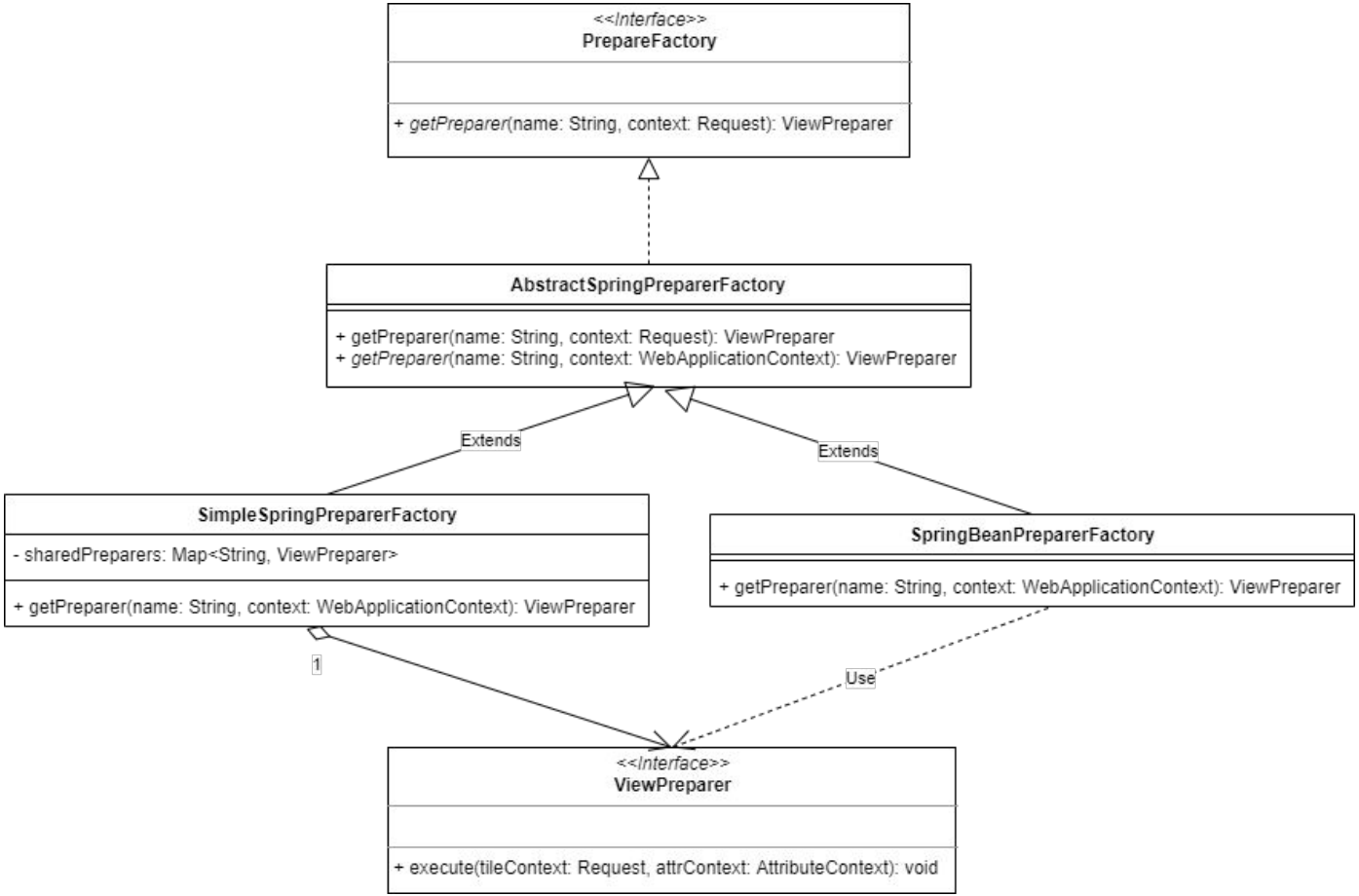
View Technologies

Spring MVC supports many kinds of view technology.

- Thymeleaf
- FreeMarker
- Groovy Markup
- Script Views
- JSP and JSTL
- Tiles
- RSS and Atom
- PDF and Excel
- Jackson
- XML Marshalling
- XSLT



IoC


<<Interface>>
PrepareFactory
+ getPreparer(name: String, context: Request): ViewPreparer
AbstractSpringPreparerFactory
+ getPreparer(name: String, context: Request): ViewPreparer
+ getPreparer(name: String, context: WebApplicationContext): ViewPreparer
Extends
Extends
SimpleSpringPreparerFactory
- sharedPreparers: Map<String, ViewPreparer>
+ getPreparer(name: String, context: WebApplicationContext): ViewPreparer
SpringBeanPreparerFactory
+ getPreparer(name: String, context: WebApplicationContext): ViewPreparer
1
Use
<<Interface>>
ViewPreparer
+ execute(tileContext: Request, attrContext: AttributeContext): void

Singleton

```
public class SimpleSpringPreparerFactory extends AbstractSpringPreparerFactory {

    /** Cache of shared ViewPreparer instances: bean name -> bean instance. */
    private final Map<String, ViewPreparer> sharedPreparers = new ConcurrentHashMap<>( initialCapacity: 16);

    @Override
    protected ViewPreparer getPreparer(String name, WebApplicationContext context) throws TilesException {
        // Quick check on the concurrent map first, with minimal locking.
        ViewPreparer preparer = this.sharedPreparers.get(name);
        if (preparer == null) {
            synchronized (this.sharedPreparers) {
                preparer = this.sharedPreparers.get(name);
                if (preparer == null) {
                    try {
                        Class<?> beanClass = ClassUtils.forName(name, context.getClassLoader());
                        if (!ViewPreparer.class.isAssignableFrom(beanClass)) {
                            throw new PreparerException(
                                    "Invalid preparer class [" + name + "]: does not implement ViewPreparer interface");
                        }
                        preparer = (ViewPreparer) context.getAutowireCapableBeanFactory().createBean(beanClass);
                        this.sharedPreparers.put(name, preparer);
                    }
                    catch (ClassNotFoundException ex) {
                        throw new NoSuchPreparerException("Preparer class [" + name + "] not found", ex);
                    }
                }
            }
        }
        return preparer;
    }
}
```

double-checked locking

From DispatcherServlet to View

DispatcherServlet > doDispatch()

```
        // Actually invoke the handler.
        mv = ha.handle(processedRequest, response, mappedHandler.getHandler());

        if (asyncManager.isConcurrentHandlingStarted()) {   asyncManager: WebAsyncManager@5894
            return;
        }

        applyDefaultViewName(processedRequest, mv);
        mappedHandler.applyPostHandle(processedRequest, response, mv);
    }
    catch (Exception ex) {
        dispatchException = ex;
    }
    catch (Throwable err) {
        // As of 4.3, we're processing Errors thrown from handler methods as well,
        // making them available for @ExceptionHandler methods and other scenarios.
        dispatchException = new NestedServletException("Handler dispatch failed", err);
    }
    processDispatchResult(processedRequest, response, mappedHandler, mv, dispatchException);
```

From DispatcherServlet to View

DispatcherServlet > processDispatchResult()

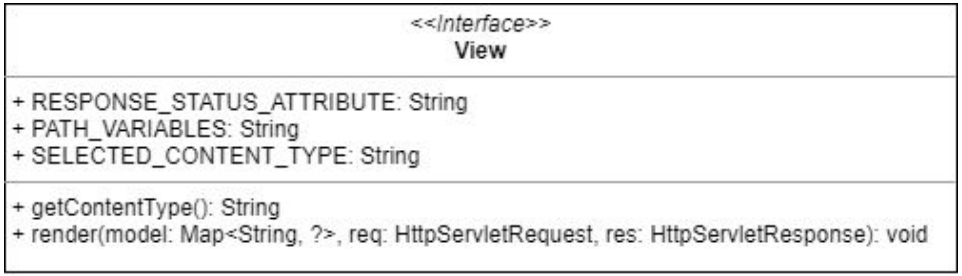
```
// Did the handler return a view to render?
if (mv ≠ null && !mv.wasCleared()) {
    render(mv, request, response);   response: ResponseFacade@5653    mv
    if (errorView) {
        WebUtils.clearErrorRequestAttributes(request);
    }
}
else {
    if (logger.isTraceEnabled()) {
        logger.trace("No view rendering, null ModelAndView returned.");
    }
}
```

DispatcherServlet > render()

```
View view;   view: "org.springframework.web.servlet.view.InternalResourceView: name 'success'; URL [/WEB-INF/pages/
String viewName = mv.getViewName();   viewName: "success"
if (viewName ≠ null) {
    // We need to resolve the view name.
    view = resolveViewName(viewName, mv.getModelInternal(), locale, request);   locale: "en"    viewName: "success"
    if (view == null) {
        throw new ServletException("Could not resolve view with name '" + mv.getViewName() +
                "' in servlet with name '" + getServletName() + "'");
    }
}
else {
    // No need to lookup: the ModelAndView object contains the actual View object.
    view = mv.getView();
    if (view == null) {
        throw new ServletException("ModelAndView [" + mv + "] neither contains a view name nor a " +
                "View object in servlet with name '" + getServletName() + "'");
    }
}
```

```
if (mv.getStatus() ≠ null) {
    response.setStatus(mv.getStatus().value());
}
view.render(mv.getModelInternal(), request, response);
```

View Interface



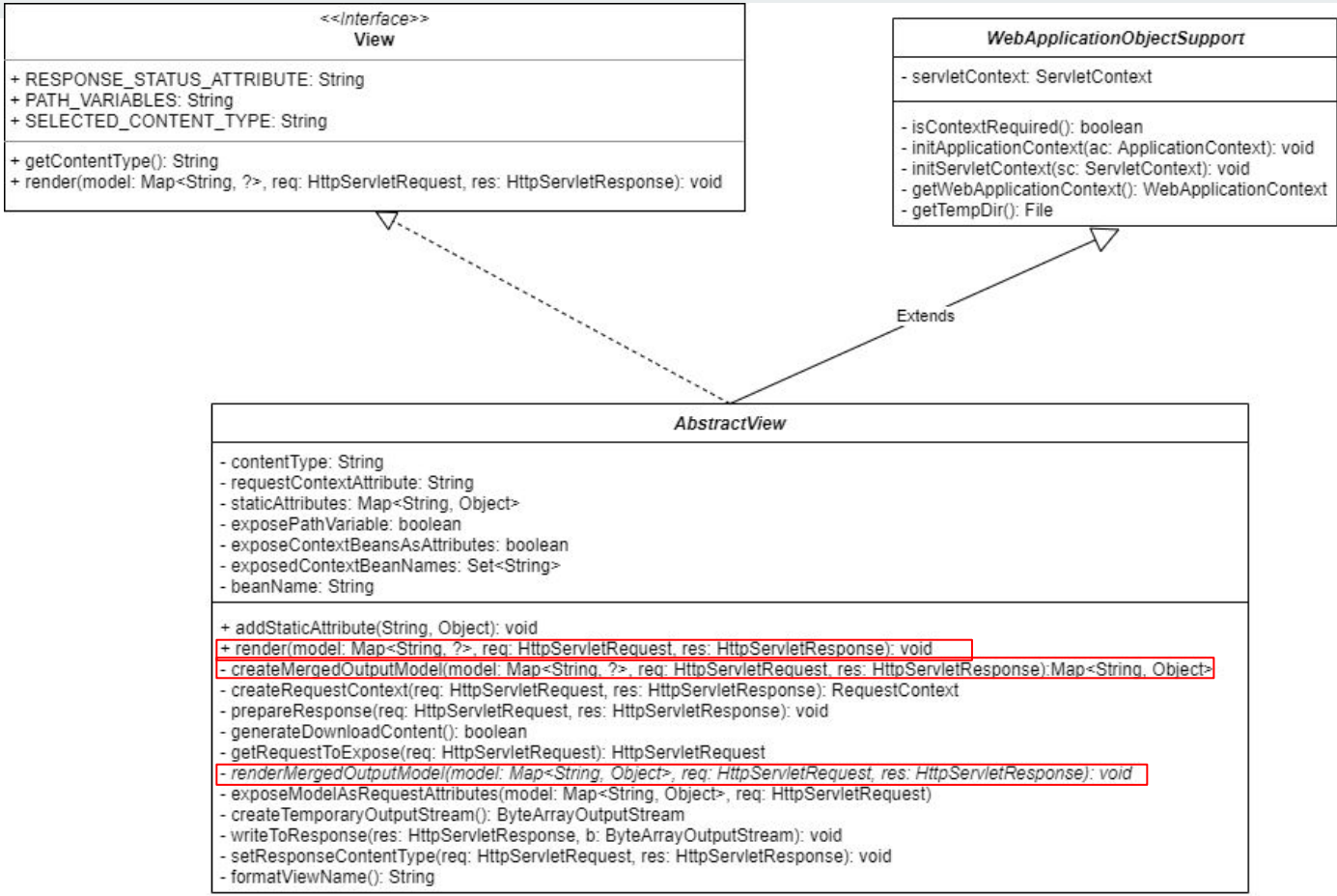
getContentType(): return the content type of the view

render(): Render the view with given model.

Step1: Prepare the request(Set the model as request object)

Step2: Render (e.g. including the JSP)

Abstract View



Support for static attributes.

Static attributed(specified in the View instance configuration) will be merged with the given dynamic attributes(e.g. path variable) for render operation.

Abstract View

AbstractView > render()

```
@Override
public void render(@Nullable Map<String, ?> model, HttpServletRequest request,   model:  size = 2    request:
        HttpServletResponse response) throws Exception {   response: ResponseFacade@5653

    if (logger.isDebugEnabled()) {
        logger.debug("View " + formatViewName() +
                ", model " + (model ≠ null ? model : Collections.emptyMap()) +
                (this.staticAttributes.isEmpty() ? "" : ", static attributes " + this.staticAttributes));
    }

    Map<String, Object> mergedModel = createMergedOutputModel(model, request, response);   model:  size = 2
    prepareResponse(request, response);
    renderMergedOutputModel(mergedModel, getRequestToExpose(request), response);   response: ResponseFacade@
}
```

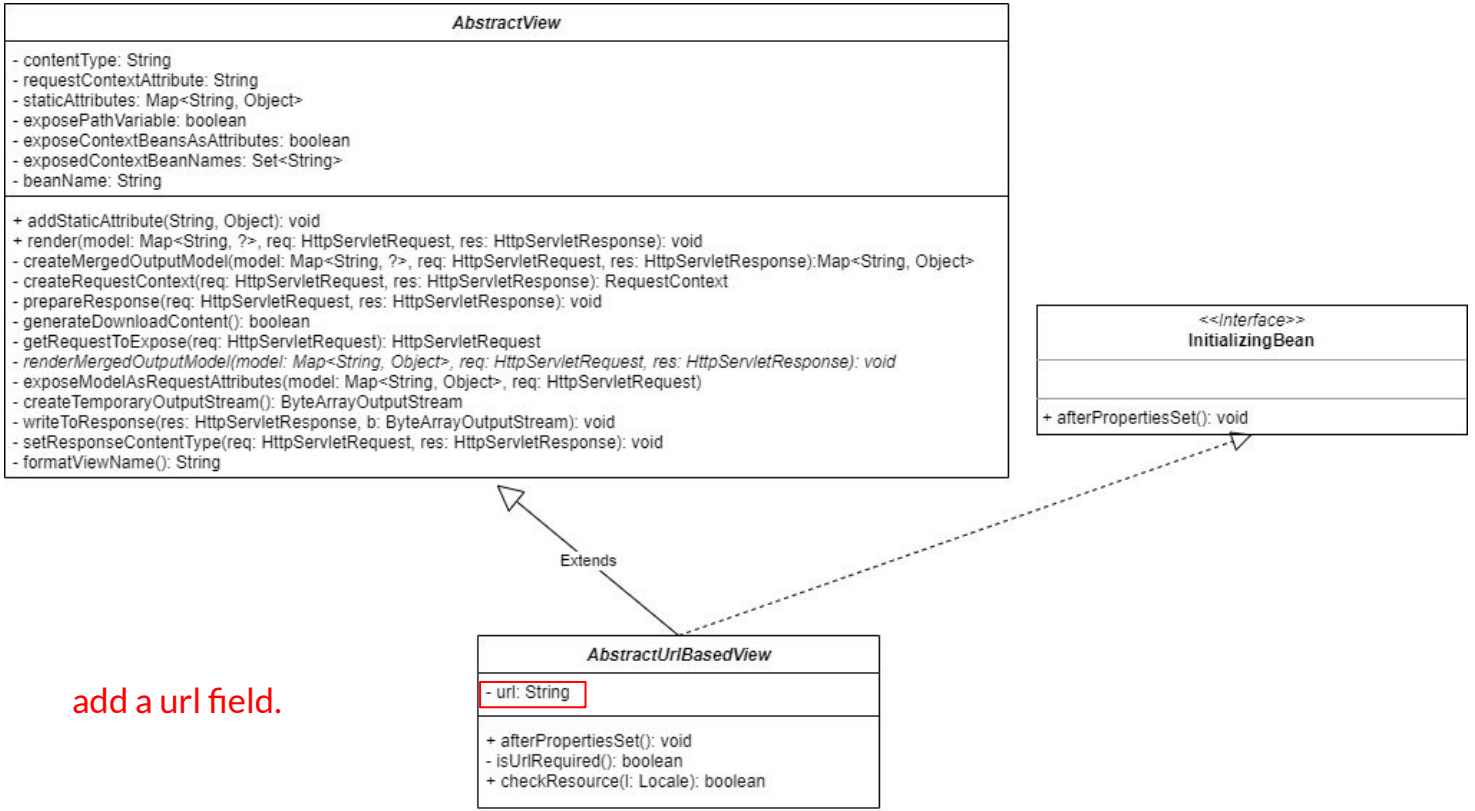
AbstractView > createMergedOutputModel()

```
protected Map<String, Object> createMergedOutputModel(@Nullable Map<String, ?> model,
        HttpServletRequest request, HttpServletResponse response) {

    /unchecked/
    Map<String, Object> pathVars = (this.exposePathVariables ?
            (Map<String, Object>) request.getAttribute(View.PATH_VARIABLES) : null);

    // Consolidate static and dynamic model attributes.
    int size = this.staticAttributes.size();
    size += (model ≠ null ? model.size() : 0);
    size += (pathVars ≠ null ? pathVars.size() : 0);

    Map<String, Object> mergedModel = new LinkedHashMap<>(size);
    mergedModel.putAll(this.staticAttributes);
    if (pathVars ≠ null) {
        mergedModel.putAll(pathVars);
    }
    if (model ≠ null) {
        mergedModel.putAll(model);
    }

    // Expose RequestContext?
    if (this.requestContextAttribute ≠ null) {
        mergedModel.put(this.requestContextAttribute, createRequestContext(request, response, mergedModel));
    }

    return mergedModel;
}
```

Dynamic attributes

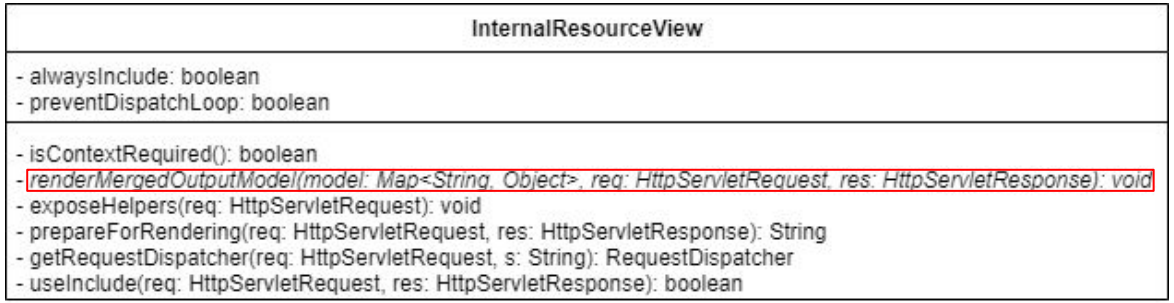
Static attributes

AbstractUrlBasedView



add a url field.

InternalResourceView



Wrapper for a resource such as a JSP.

Expose model objects as request attributes and forward the request to specified URL.

```
<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">

    <property name="prefix" value="/WEB-INF/jsp/"/>

    <property name="suffix" value=".jsp"/>

  </bean>
```

Every view name returned from a handler will be translated to a JSP resource (for example: "myView" -> "/WEB-INF/jsp/myView.jsp"), using this view class by default.

InternalResourceView

Expose model objects as request attributes

Forward the request to specified URL

ApplicationDispatcher

```
protected void renderMergedOutputModel(
        Map<String, Object> model, HttpServletRequest request, HttpServletResponse response) throws Exception {

    // Expose the model object as request attributes.
    exposeModelAsRequestAttributes(model, request);
    // Expose helpers as request attributes, if any.
    exposeHelpers(request);
    // Determine the path for the request dispatcher.
    String dispatcherPath = prepareForRendering(request, response);
    // Obtain a RequestDispatcher for the target resource (typically a JSP).
    RequestDispatcher rd = getRequestDispatcher(request, dispatcherPath);
    if (rd == null) {
        throw new ServletException("Could not get RequestDispatcher for [" + getUrl() +
                "]: Check that the corresponding file exists within your web application archive!");
    }
    // If already included or response already committed, perform include, else forward.
    if (useInclude(request, response)) {
        response.setContentType(getContentType());
        if (logger.isDebugEnabled()) {
            logger.debug("Including [" + getUrl() + "]");
        }
        rd.include(request, response);
    }
    else {
        // Note: The forwarded resource is supposed to determine the content type itself.
        if (logger.isDebugEnabled()) {
            logger.debug("Forwarding to [" + getUrl() + "]");
        }
        rd.forward(request, response);
    }
}
```

Python Implementation

Presenter: **蔡昀達**

Coding styles

	Java	Python
Coding style	spring-javaformat	PEP 8
Document	Javadoc	Pydoc
Transform rules	constant, final, protected, interface, functional interface, function overloading	hackmd

Deal with java objects/dependencies 1

- Use java ⇔ python corresponding modules
  - same functional
  - few method name different
- For example
  - re
  - urllib
  - datetime
  - pytz
  - logging

- Use python Mock Objects
  - implement necessary methods only
- For example
  - ServletOutputStream
  - ResponseServletOutputStream
  - ByteArrayInputStream
  - SessionCookieConfig
  - AsyncEvent
  - UriUtils
  - BeanUtils
  - BeanFactoryUtils
  - ….. many others

Deal with java objects/dependencies 2

- Use python build-in types replace java objects
  - python build-in types are also objects with sufficient methods
  - original implementation mix usage of basic types and objects
- For example
  - StringTokenizer
  - StringBuilder

Java

```
protected void appendQueryProperties(StringBuilder targetUrl,
protected String updateTargetUrl(String targetUrl,
protected void appendCurrentQueryParams(StringBuilder targetUrl,
protected StringBuilder replaceUriTemplateVariables(
        String targetUrl, Map<String, Object> model, M
```

Python

```
def update_target_url(self, targetUrl: str, model:
def append_current_query_params(self, targetUrl: str, requ
def append_query_properties(self, targetUrl: str, mod
def replace_uri_template_variables(
    self,
    targetUrl: str,
    model: dict,
```

Deal with java objects/dependencies 3

- Complete reimplementation
  - Many dependencies from spring-web / spring-core / spring-bean / spring-context
  - Use as test fixtures instead of a part of the spring-mvc
- For example
  - Cookie
  - HttpSession
  - HttpServletRequest
  - HttpServletResponse
  - RequestDispatcher
  - ServletConfig
  - ServletContext
  - ….. many others

Anonymous Classes

- Dynamic overrides
- Accessing Local Variables of the Enclosing Scope
- Declaring and Accessing Members of the Anonymous Class

```
public abstract class AbstractCachingViewResolver extends WebApplicationObjectSupport {

    /** Map from view key to View instance, synchronized for View creation. */
    @SuppressWarnings("serial")
    private final Map<Object, View> viewCreationCache =
            new LinkedHashMap<Object, View>(DEFAULT_CACHE_LIMIT, 0.75f, true) {
                @Override
                protected boolean removeEldestEntry(Map.Entry<Object, View> eldest) {
                    if (size() > getCacheLimit()) {
                        viewAccessCache.remove(eldest.getKey());
                        return true;
                    }
                    else {
                        return false;
                    }
                }
            };
```

Tricky cases

```
class AbstractCachingViewResolver():

    def set_cache_limit(self, s: int):
        self.s = s

    def get_cache_limit(self):
        return self.s

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    class my_order_dict(OrderedDict):
        def __setitem__(self, key, value):
            if self.getcache_limit() <= self.__len__():
                self.popitem(last=False)
            return super().__setitem__(key, value)

    viewCreationCache = my_order_dict()
    viewCreationCache.getcache_limit = get_cache_limit


tmp = AbstractCachingViewResolver()
tmp.set_cache_limit(3)
for i in range(5):
    tmp.viewCreationCache[i] = i
    print(tmp.viewCreationCache.keys())
```

```
TypeError                                 Traceback (mos
```

```
class AbstractCachingViewResolver():

    def set_cache_limit(self, s: int):
        self.s = s

    def get_cache_limit(self):
        return self.s

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        class my_order_dict(OrderedDict):
            def __setitem__(self, key, value):
                if self.getcache_limit() <= self.__len__():
                    self.popitem(last=False)
                return super().__setitem__(key, value)
        getcache_limit = self.get_cache_limit

        self.viewCreationCache = my_order_dict()

tmp = AbstractCachingViewResolver()
tmp.set_cache_limit(3)
for i in range(5):
    tmp.viewCreationCache[i] = i
    print(tmp.viewCreationCache.keys())
```

```
odict_keys([0])
odict_keys([0, 1])
odict_keys([0, 1, 2])
odict_keys([1, 2, 3])
odict_keys([2, 3, 4])
```

Function overloading

- naive solution is not be enough
  - default value for parameter

```
public ModelAndView()
public ModelAndView(String viewName)
public ModelAndView(View view)
public ModelAndView(String viewName, @Nullable Map<String, ?> model)
public ModelAndView(View view, @Nullable Map<String, ?> model)
public ModelAndView(String viewName, HttpStatus status)
public ModelAndView(@Nullable String viewName, @Nullable Map<String, ?> model, @Nullable HttpStatus status)
public ModelAndView(String viewName, String modelName, Object modelObject)
public ModelAndView(View view, String modelName, Object modelObject)
```

```
protected void initApplicationContext(ApplicationContext context)
protected void initApplicationContext()
```

```
def init_application_context(self, context=None)
def init_application_context(self)
```

Function overloading

- Implement metaclass to support overloading
- Typing as signature
- Support default values of NoneType (Nullable)
- Support subclass signature (prefix tree search)
- Disadvantage
  - metaclass conflicts with abstract class modules
  - must write typing for every function

```
def _register(self, types, meth):
    if types in self._methods:
        raise Exception(f"Duplicate signatures. {types}")
    self._methods[types] = meth

def register(self, meth):
    '''
    Register a new method as a multimethod
    '''
    sig = inspect.signature(meth)

    # Build a type signature from the method's annotations
    types = [[]]
    for name, parm in sig.parameters.items():
        if name == 'self':
            continue
        if parm.annotation is inspect.Parameter.empty:
            raise TypeError(
                'Argument {} must be annotated with a type'.format(na
            )
        if not isinstance(parm.annotation, type):
            raise TypeError(
                'Argument {} annotation must be a type'.format(name)
            )
        if parm.default is not inspect.Parameter.empty:
            for i in types:
                self._register(tuple(i), meth)

        new = deepcopy(types)

        # append annotation
        for i in types:
            i.append(parm.annotation)

        # append None type
```

Function overloading

```
register ->  (<class 'int'>, <class 'int'>)
================================================
register ->  (<class 'int'>, <class 'object'>)
================================================
register ->  (<class 'int'>, <class 'str'>)
register ->  (<class 'int'>, <class 'str'>, <class 'str'>)
register ->  (<class 'int'>, <class 'str'>, <class 'NoneType'>)
================================================
register ->  (<class 'int'>, <class 'str'>, <class '__main__.a'>)
================================================
=== start search ===
```

	signature	target
0	(<class 'int'>, <class 'int'>)	(<class 'int'>, <class 'str'>, <class '__main__.b'>)
1	(<class 'int'>, <class 'object'>)	(<class 'int'>, <class 'str'>, <class '__main__.b'>)
2	(<class 'int'>, <class 'str'>)	(<class 'int'>, <class 'str'>, <class '__main__.b'>)
3	(<class 'int'>, <class 'str'>, <class 'str'>)	(<class 'int'>, <class 'str'>, <class '__main__.b'>)
4	(<class 'int'>, <class 'str'>, <class 'NoneType'>)	(<class 'int'>, <class 'str'>, <class '__main__.b'>)
5	(<class 'int'>, <class 'str'>, <class '__main__.a'>)	(<class 'int'>, <class 'str'>, <class '__main__.b'>)

```
Found ->  (<class 'int'>, <class 'str'>, <class '__main__.a'>)
```

```python
class a():
    pass

class b(a):
    pass

class Demo(metaclass=MultipleMeta):
    def p(self, x:int, y:int):
        pass

    def p(self, x:int, y: object):
        pass

    def p(self, x:int, y: str, z: str = None):
        pass

    def p(self, x:int, y:str, z:a):
        pass

demo = Demo()
demo.p(123, "123", b())
```

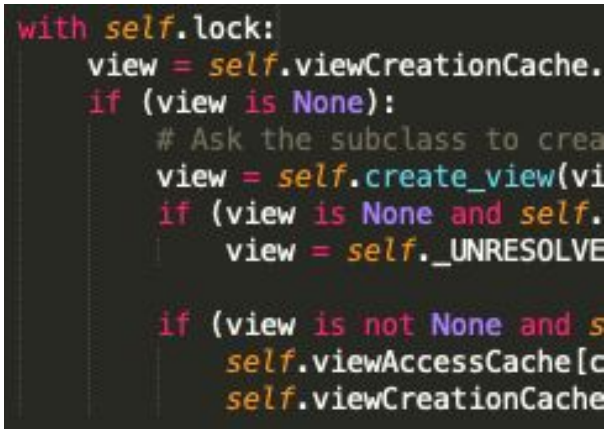
Class methods before instantiate

- Metaclass can access cls
- Metaclass is reusable

```
class HandlerMappingInterfaceMeta(type):

    def __init__(cls, *args, **kwargs):
        cls.BEST_MATCHING_HANDLER_ATTRIBUTE: str = cls.__name__ + ".bestMatchingHa
        cls.LOOKUP_PATH: str = cls.__name__ + ".lookupPath"
        cls.PATH_WITHIN_HANDLER_MAPPING_ATTRIBUTE: str = cls.__name__ + ".pathWith
        cls.BEST_MATCHING_PATTERN_ATTRIBUTE: str = cls.__name__ + ".pathWithinHand
        cls.INTROSPECT_TYPE_LEVEL_MAPPING: str = cls.__name__ + ".bestMatchingPatt
        cls.URI_TEMPLATE_VARIABLES_ATTRIBUTE: str = cls.__name__ + ".introspectTyp
        cls.MATRIX_VARIABLES_ATTRIBUTE: str = cls.__name__ + ".uriTemplateVariable
        cls.PRODUCIBLE_MEDIA_TYPES_ATTRIBUTE: str = cls.__name__ + ".matrixVariabl


# https://stackoverflow.com/questions/57349105/python-abc-inheritance-with-specifi
class CombinedMeta(HandlerMappingInterfaceMeta, ABCMeta):
    pass


class HandlerMappingInterface(ABC, metaclass=CombinedMeta):

    def uses_path_patterns(self) -> bool:
        return False

    @abstractmethod
    def get_handler(self, request: HttpServletRequest):
        raise NotImplementedError
```

Sync vs Async

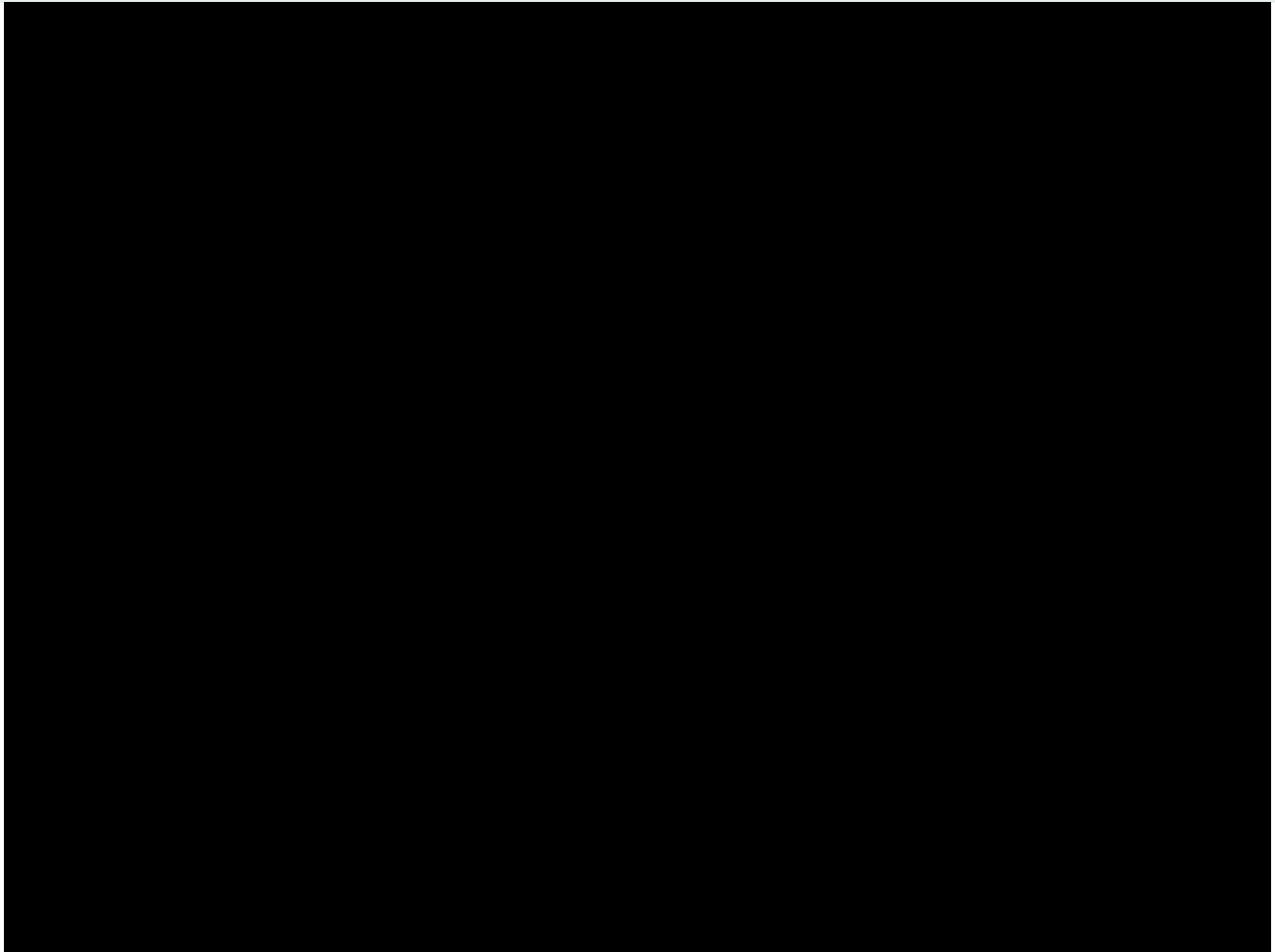
- spring async support
  - concurrent.ThreadPoolTaskExecutor
- python
  - asyncio
  - threading



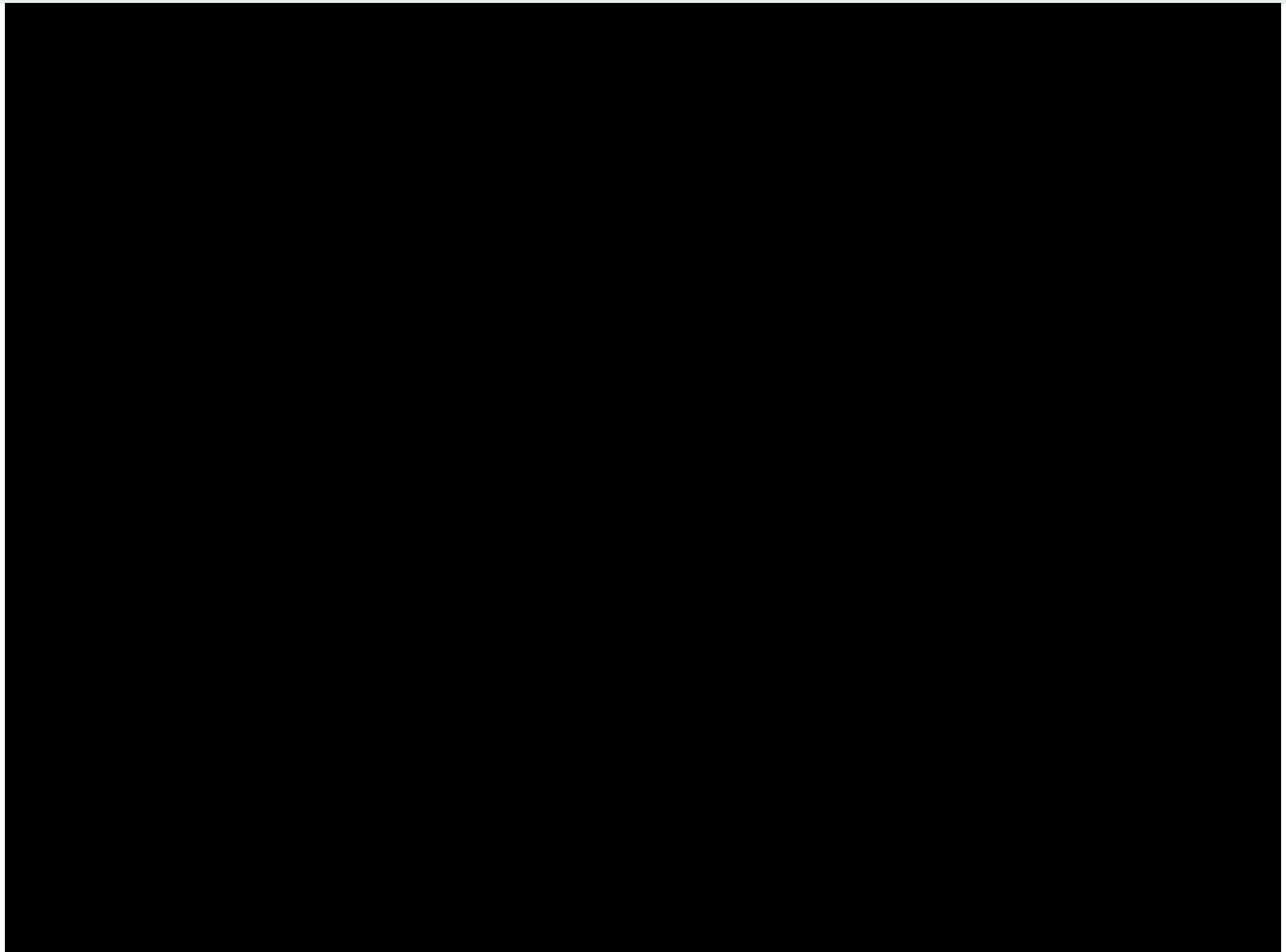
Demo

Presenter: **蔡昀達**

Tests



Demo



Project Folder

Gitlab

- Source Code
- Demo Video
- Slide