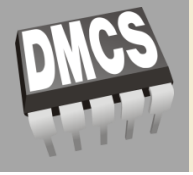


SPRING FRAMEWORK

Katedra Mikroelektroniki i Technik Informatycznych
Politechniki Łódzkiej
ul. Wólczanska 221/223 budynek B18,
90-924 Łódź

dr inż. Jakub Chłapiński

3. Spring Web MVC

DispatcherServlet

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Centralnym elementem architektury Spring Web MVC jest klasa DispatcherServlet, implementująca mechanizm przekierowania obsługi żądania HTTP do odpowiedniego kontrolera
- Ponadto klasa DispatcherServlet uruchamia ApplicationContext dla aplikacji internetowej oraz wstrzykuje go do ServletContext, pozwalając na łatwe odwołanie się do ApplicationContext z poziomu całej aplikacji opartej o servlety

DispatcherServlet (2)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Konfiguracja DispatcherServlet

/WEB-INF/web.xml

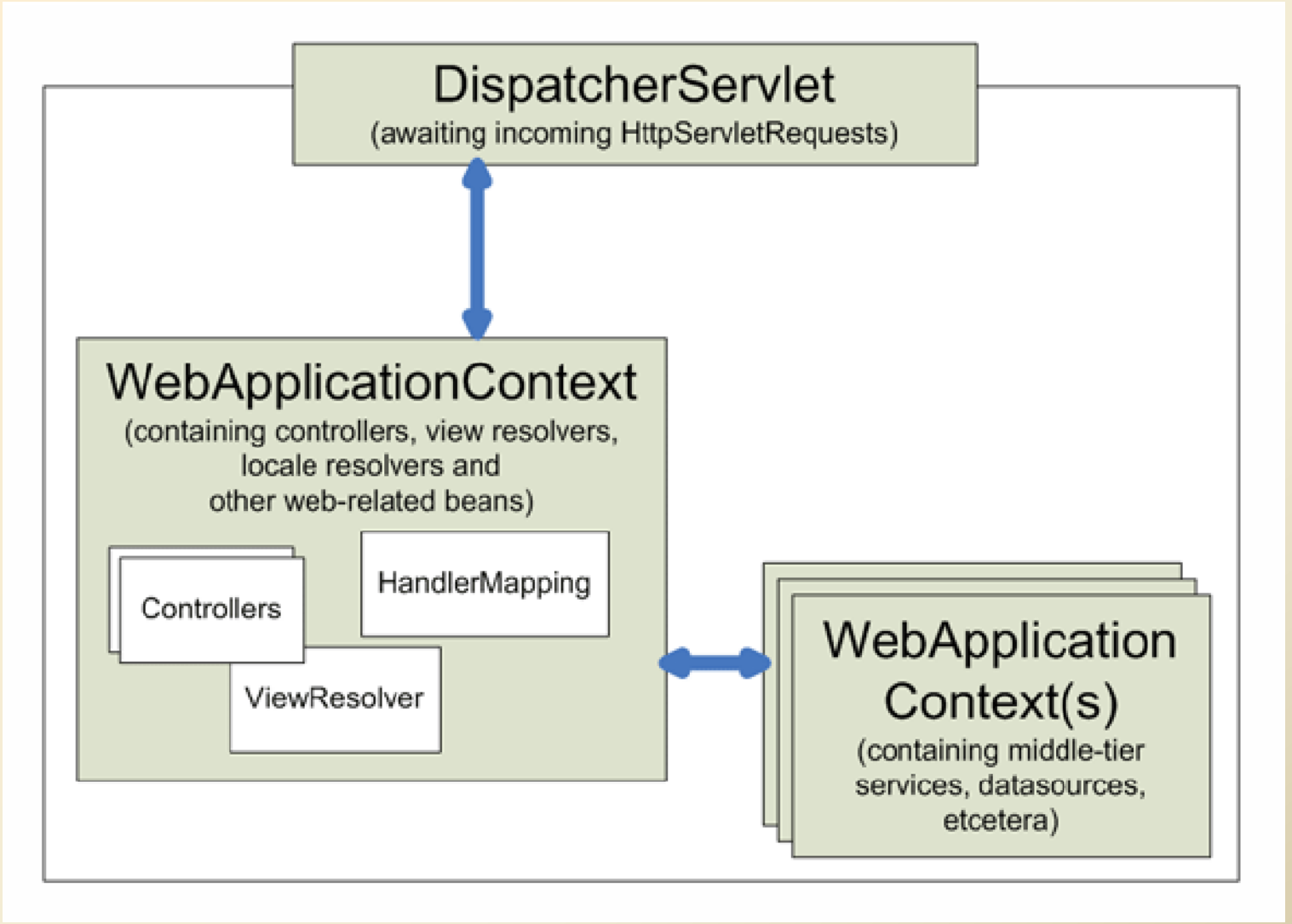
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
  ...
  <servlet>
    <servlet-name>sklep</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>
        /WEB-INF/sklep-dao.xml
        /WEB-INF/sklep-service.xml
        /WEB-INF/sklep-mvc.xml
      </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
     <servlet-name>sklep</servlet-name>
     <url-pattern>*.form</url-pattern>
  </servlet-mapping>

  ...
</web-app>
```

DispatcherServlet (3)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl



DispatcherServlet (4)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- W ogólności można w obrębie jednej aplikacji zdefiniować wiele obiektów DispatcherServlet, z których każdy może posiadać osobną konfigurację dla ApplicationContext, jest to jednak rzadko spotykane rozwiązanie

Rodzaje komponentów

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Spring Web MVC posiada kilka rodzajów beanów, służących do obsługi poszczególnych mechanizmów obsługi żadań HTTP
  - Controllers - implementacja w aplikacji przepływu sterowania w aplikacji
  - Handler mappings - nadzór i wykonanie pre- i post-procesingu oraz wywoływanie kontrolerów w obsłudze żadań w oparciu o zdefiniowane kryteria
  - View resolvers - dostarczenie konkretnej implementacji widoku na podstawie jego logicznej nazwy
  - Locale resolvers - dostarczenie danych o lokalizacji klienta
  - Theme resolvers - dostarczenie informacji o stylu w jakim należy prezentować klientowi widoki
  - Multipart file resolver - wsparcie dla uploadu plików binarnych na serwer
  - Handler exception resolvers - wsparcie dla mapowania widoków dla błędów (wyjątków) lub inne mechanizmy obsługi i raportowania błędów

Obsługa żądania HTTP

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

1. Do obiektu HttpServletRequest dołączany jest ApplicationContext w atrybucie o kluczu zdefinowanym w DispatcherServlet.WEB_APPLICATION_CONTEXT_ATTRIBUTE
2. Do żądania przypisywany jest obiekt locale resolver jeśli jest używany
3. Do żadania przypisywany jest obiekt theme resolver (jeśli jest używany)
4. Jeśli jest obecny, żądanie jest analizowane przez multipart file resolver, jeżeli żądanie zawiera fragmenty pliku, obiekt żądania opakowywany jest w MultipartHttpServletRequest i przekazywany do dalszego przetwarzania
5. Poszukiwany jest odpowiedni handler dla żądania, jeżeli znaleziony to uruchamia on łańcuch przetwarzania w nim zdefiniowany (preprocesory, postprocesory, kontrolery)
6. Jeśli łańcuch przetwarzania zwróci model renderowany jest widok, w innym wypadku przetwarzanie jest zakończone

Kontrolery

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Spring w wersji od 2.5 pozwala na wygodne implementowanie kontrolerów jedynie przy pomocy adnotacji dla klas POJO
- Aby kontroler został dołączony do konfiguracji wystarczy zdefiniować go jako zwykły bean

```
@Controller
public class WelcomeController {

  @RequestMapping("/welcome1")
  public ModelAndView welcome1 () {
    ModelAndView mav = new ModelAndView();
    mav.setViewName("welcome"); //nazwa logiczna widoku
    mav.addObject("message", "Welcome!");
    return mav;
  }

  @RequestMapping("/welcome2")
  public String welcome2(ModelMap model) {
    model.addAttribute ("message", "Welcome!");
    return "welcome"; //nazwa logiczna widoku
  }
}
```

@RequestMapping

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Adnotacja @RequestMapping umożliwia automatyczne zarejestrowanie kontrolera w mechanizmie mapowania ścieżki żądania do łańcucha przetwarzania
- @RequestMapping nad klasą kontrolera określa ścieżkę w URI dla wszystkich metod kontrolera

```
@Controller
@RequestMapping("\powitanie")
public class WelcomeController {
  @RequestMapping("pierwsze")
  public ModelAndView welcome1 () {
    ModelAndView mav = new ModelAndView();
    mav.setViewName("welcome"); //nazwa logiczna widoku
    mav.addObject("message", "Welcome!");
    return mav;
  }
  @RequestMapping("drugie")
  public String welcome2(ModelMap model) {
    model.addAttribute ("message", "Welcome!");
    return "welcome"; //nazwa logiczna widoku
  }
}
```

@RequestMapping (2)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Można również zawężać wykonanie danej metody dla typu żądania (GET lub POST) oraz przypadku gdy w URL znajdują się określone parametry

```
// metoda wykona się tylko dla metody GET i jeżeli w URL jest parametr akcja=znajdz
@RequestMapping(value="\do", method=RequestMethod.GET, params="akcja=znajdz")
public String znajdz() { ... }

// wykona się jeżeli w URL jest parametr akcja o dowolnej wartości
@RequestMapping(value="\do", params="akcja")
public String akcja() { ... }

// wykona się dla metody POST i jeżeli w URL nie ma parametru akcja
@RequestMapping(value="\do", method=RequestMethod.POST, params="!akcja")
public String reakcja() { ... }
```

@PathVariable

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Adnotacja @PathVariable pozwala na korzystanie z możliwości definiowania ścieżek URI w oparciu o wzory

```
@Controller
public class UserController {

  @RequestMapping(value="/users/{userId}", method=RequestMethod.GET)
  public String showUser(@PathVariable String userId, Model model) {
    User user = userService.find(userId);
    model.addAttribute("user", user);
    return "showUser";
  }

}
```

@ModelAttribute

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Adnotacja @ModelAttribute umożliwia automatyczne bindowanie danych z formularzy do obiektów domenowych
- Przy stosowaniu tej adnotacji trzeba zwrócić uwagę na kolejność argumentów w adnotowanych metodach jeśli potrzebny jest obiekt BindingResult lub Errors (musi być bezpośrednio po obiekcie bindowanym)

```
@RequestMapping(method = RequestMethod.POST)
public String process(@ModelAttribute("osoba") Osoba osoba,
  BindingResult result, Model model) { ... } // OK.

@RequestMapping(method = RequestMethod.POST)
public String process(@ModelAttribute("osoba") Osoba osoba,
  Errors errors, Model model) { ... } // OK.

@RequestMapping(method = RequestMethod.POST)
public String process(@ModelAttribute("osoba") Osoba osoba,
  Model model, BindingResult result) { ... } // ZLE!!!
```

@ModelAttribute (2)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Adnotacja @ModelAttribute zastosowana nad metodą umożliwia też automatyczne umieszczenie danych zwracanych przez metodę w modelu przed wywolaniem metod z @RequestMapping

```
@Controller
@RequestMapping("/osoba")
public class OsobaController {
  // ...

  @ModelAttribute("tytuly")
  public Collection<Tytul> wstawTytuly () {
    return this.tytulService.getAll();
  }

  @RequestMapping(method = RequestMethod.POST)
  public String submit(@ModelAttribute("osoba") Osoba osoba, Model model) {
    // ...
  }
}
```

@RequestParam

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Adnotacja @RequestParam umożliwia automatyczne mapowanie parametrów z żądania do zmiennych w kodzie Java

```
@RequestMapping(method = RequestMethod.GET)
public String show(@RequestParam("userId") int userId, ModelMap model) {
  User user = userService.get(userId);
  model.addAttribute("user", user);
  return "userForm";
}
```

@RequestBody

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Adnotacja @RequestBody umożliwia dostęp do zawartości (body) żądania HTTP

```
@RequestMapping(value = "/upload", method = RequestMethod.PUT)
public void zapisz(@RequestBody String body, Writer writer) throws IOException {
  writer.write(body);
}
```

@RequestHeader

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Zastosowanie adnotacji @RequestHeader umożliwia dostęp do wartości określonych w nagłówku żądania HTTP

```
@RequestMapping("/sciezka")
public void analizujNaglowek(@RequestHeader("Accept-Encoding") String encoding,
  @RequestHeader("Keep-Alive") long keepAlive) {
  // ...
}
```

@ResponseBody

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Adnotacja @ResponseBody umożliwia umieszczenie danych bezpośrednio w odpowiedzi na żądanie HTTP

```
@RequestMapping(value = "/raport", method = RequestMethod.PUT)
@ResponseBody
public String raport() {
  return "Raport miesieczny";
}
```

HttpEntity<?>

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Zastosowanie HttpEntity<?> umożliwia łatwy dostęp do poszczególnych elementów składowych żądań i dopowiedzi HTTP (nagłówki, parametry, zawartości itd.)

```
@RequestMapping("/raport")
public ResponseEntity<String> handle(HttpEntity<byte[]> requestEntity) {
  String requestHeader = requestEntity.getHeaders().getFirst("ETag"));
  byte[] requestBody = requestEntity.getBody();
  // ...

  HttpHeaders responseHeaders = new HttpHeaders();
  responseHeaders.set("ETag", requestHeader);
  return new ResponseEntity<String>("Tekst odpowiedzi", responseHeaders,
    HttpStatus.CREATED);
}
```

@SessionAttribute

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Do przechowywania zmiennych w sesji można używać adnotacji @SessionAttribute

```
@Controller
@RequestMapping("/osoba")
@SessionAttribute("osoba")
public class OsobaController {
  // ...

  @RequestMapping(method = RequestMethod.POST)
  public String submit(@ModelAttribute("osoba") Osoba osoba, SessionStatus status) {
    // ...
    status.setComplete(); // usuniecie zmiennych sesyjnych kontrolera
  }
}
```

@CookieValue

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Adnotacja @CookieValue jest stosowana do pobrania wartości parametru cookie z żądania HTTP

```
@RequestMapping("/sciezka")
public void processSessionInfo(@CookieValue("JSESSIONID") String cookie) {
  //...
}
```

@InitBinder

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Dzięki adnotacji @InitBinder można dokonać dodatkowej inicjalizacji obiektu WebDataBinder zanim uruchomiony zostanie proces bindowania danych z żądania do modelu
- Możliwe jest również zaimplementowanie w tym celu beanu WebBindingInitializer dla wszystkich kontrolerów
- Prawidłowa inicjalizacja bindera jest bardzo ważna ze względu na bezpieczeństwo danych

```
@Controller
public class OsobaController {

  @InitBinder
  public void initBinder(WebDataBinder binder) {
    binder.setAllowedFields({"imie", "nazwisko"}); // bindownie tylko wybranych pol
    binder.setRequiredFields({"imie"}); // blad w przypadku braku okreslonych pol
  }

  // ...
}
```

Mapowanie żądań

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Przed wersją 2.5 w Spring MVC konieczne było jawne zdefiniowanie beanu(ów) implementujących interfejs HandlerMapping dokonujących mapowania żądań do kontrolerów
- Od wersji 2.5 w DispatcherServlet domyślnie używany jest bean klasy DefaultAnnotationHandlerMapping, który wyszukuje w kontenerze Spring beany o stereotypie @Controller z adnotacją @RequestMapping i na tej podstawie dokonuje mapowania żądań
- W większości przypadków nie jest konieczna ingerencja w domyślne ustawienia tego beanu, jest to jednak możliwe, np.:

```
<bean id="handlerMapping"
  class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
  <property name="interceptors">
    <bean class="example.MyInterceptor"/>
  </property>
</bean>
```

Interceptory

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- W proces mapowania żądań do kontrolerów można również dołączać tzw. interceptory dokonując pre- lub postprocesingu żądań
- Interceptory musza implementować interfejs HandlerInterceptor

```
public interface HandlerInterceptor {

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

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

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

}
```

Dopasowywanie widoków

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Nazwa widoku zwracana w ModelAndView przez kontroler obsługujący dane żądanie jest w istocie nazwą logiczną
- Aby dopasować konkretną implementację widoku do jego nazwy logicznej w Spring MVC stosowane są beany implementujące interfejs ViewResolver

```
public interface ViewResolver {

  View resolveViewName(String viewName, Locale locale) throws Exception;

}
```

Dopasowywanie widoków (2)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Spring MVC zawiera cały szereg implementacji interfejsu ViewResolver, pozwalających na integrację z widokami JSP, Velocity, XSLT, itd.
- Beany ViewResolver można również łączyć w łańcuchy, gdzie w przypadku gdy dany ViewResolver nie dopasuje widoku do nazwy, wywołany zostanie kolejny, itd.
- Większość implementacji ViewResolver implementuje Ordered

```
<!-- InternalResourceViewResolver jest automatycznie dodawany na koncu lancucha -->
<bean id="jspViewResolver"
  class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
  <property name="prefix" value="/WEB-INF/jsp/"/>
  <property name="suffix" value=".jsp"/>
</bean>

<bean id="excelViewResolver"
  class="org.springframework.web.servlet.view.XmlViewResolver">
  <property name="order" value="1"/> <!-- z interfejsu Ordered -->
</bean>
```

Dopasowywanie widoków (3)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Spring MVC zawiera cały szereg implementacji interfejsu ViewResolver, pozwalających na integrację z widokami JSP, Velocity, XSLT, itd.
  - XmlViewResolver - implementacja pozwalająca na konfigurację w oddzielnym pliku XML (domyślnie /WEB-INF/views.xml)
  - ResourceBundleViewResolver - implementacja z konfiguracją w postaci plików .properties
  - UrlBasedViewResolver - implementacja, w której dopasowanie widoku odbywa się na podstawie URL żądania
    - InternalResourceViewResolver
    - VelocityViewResolver
    - FreeMarkerViewResolver
  - ContentNegotiatingViewResolver - dopasowanie widoku na podstawie nazwy pliku lub nagłówków w żądaniu

Generowanie widoków

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- W zależności od typu widoku przekierowanie sterowania do widoku zachodzi przy wykorzystaniu RequestDispatcher.forward() lub RequestDispatcher.include() z Servlet API (np. widoki JSP) lub poprzez bezpośrednie zapisanie widoku w HttpServletResponse (np. Velocity)

Przekierowanie HTTP

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- W razie potrzeby wykonania przekierowania HTTP można w kontrolerze utworzyć gotową implementację widoku z wykorzystaniem klasy RedirectView, co pozwoli na pominięcie standardowego mechanizmu i wykonanie przekierowania przez HttpServletResponse.sendRedirect()
- Podobny efekt można uzystać poprzez użycia prefiksu "redirect:" w nazwie logicznej widoku

```
@Controller
public class OsobaController {

  @RequestMapping("/osoba")
  public ModelAndView pokaz(@RequestParam("id") Long id) {
    Osoba osoba = osobaService.get(id);
    if (osoba != null) {
      return new ModelAndView("osobaForm");
    } else {
      return new ModelAndView(new RedirectView("/blad.html"));
    }
  }
}
```

Przekierowanie HTTP (2)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Przekierowanie można również uzyskać poprzez użycie prefiksu "redirect:" w nazwie logicznej widoku

```
// przekierowanie względem kontekstu servletu
return new ModelAndView("redirect:/blad.html");

// przekierowanie absolutne
return new ModelAndView("redirect:http://www.domena.pl/blad.html");
```

Internacjonalizacja

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- W trakcie obsługi żądania DispatcherServlet wykorzystuje LocaleResolver (jeżeli zdefiniowano) do ustalenia wersji językowej aplikacji
  - AcceptHeaderLocaleResolver - wersja językowa ustalana na podstawie nagłówka "accept-language" w żądaniu
  - CookieLocaleResolver - wersja językowa ustalana na podstawie wartości cookie
  - SessionLocaleResolver - wersja językowa ustalana na podstawie zmiennej sesyjnej
- Bieżące ustawienie wersji językowej można odczytać poprzez RequestContext.getLocale()

Internacjonalizacja (2)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Aby zaimplementować funkcjonalność wybierania wersji językowej w aplikacji przez użytkownika można wykorzystać LocaleChangeInterceptor
- Przykładowy URL: www.domena.pl/osoba?id=1&lang=en_US

```
<bean id="localeChangeInterceptor"
  class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
  <property name="paramName" value="lang"/>
</bean>

<bean id="localeResolver"
  class="org.springframework.web.servlet.i18n.CookieLocaleResolver"/>

<bean id="urlMapping"
  class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
  <property name="interceptors">
    <list>
      <ref bean="localeChangeInterceptor"/>
    </list>
  </property>
</bean>
```

Motywy

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Aby dodać w aplikacji wsparcie dla motywów należy dodać do kontenera bean implementujący interfejs org.springframework.ui.context.ThemeSource
- Domyślnie wykorzystywana klasa WebApplicationContext implementuje również ThemeSource, jednak deleguje tą funkcjonalność do konkretnej implementacji, domyślnie ResourceBundleThemeSource
- Aby wykorzystać inną implementację ThemeSource wystarczy dodać ją do kontenera, przy uruchomieniu WebApplicationContext zostanie ona wykorzystana zamiast domyślnej

Motywy (2)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Przykład użycia motywów zdefiniowanych w plikach .properties z wykorzystaniem ResourceBundleThemeSource

/WEB-INF/classes/czarny.properties

```
tlo=/resources/czarny/tlo.jpg
glownyStyl=/resources/czarny/glowny.css
```

/WEB-INF/classes/ bialy.properties

```
tlo=/resources/bialy/tlo.jpg
glownyStyl=/resources/bialy/glowny.css
```

/WEB-INF/jsp/index.jsp

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<html>
  <head>
    <link rel="stylesheet" href="<spring:theme code='glownyStyl'/>" type="text/css"/>
  </head>

  <body style="background=<spring:theme code='tlo'/>">
    <center>Witaj</center>
  </body>
</html>
```

Motywy (3)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Implementacja ResourceBundleThemeSource umożliwia definiowanie osobnych motywów dla różnych wersji językowych analogicznie jak w przypadku MessageSource, poprzez dodanie nazwy wersji językowej do nazwy pliku, np.:
  - /WEB-INF/classes/bialy_pl_PL.properties
  - /WEB-INF/classes/bialy_en_US.properties

Motywy (4)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Do ustalenia bieżącego motywu wykorzystywane są implementacje interfejsu ThemeResolver (analogicznie jak w przypadku LocaleResolver)
  - FixedThemeResolver - implementacja w której bieżący motyw zdefiniowany jest statycznie (przydatne do testowania)
  - SessionThemeResolver - bieżący motyw zdefiniowany w zmiennej sesyjnej
  - CookieThemeResolver - bieżący motyw zapisany w pliku cookie w przeglądarce klienta
- Ponadto można też wykorzystać ThemeChangeInterceptor do zmiany motywu przy pomocy parametru URL (analogicznie jak w przypadku LocaleChangeInterceptor)

Upload plików

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Domyślnie DispatcherServlet nie dokonuje automatycznego łączenia wielu żądań zawierających fragmenty przesyłanego pliku, ponieważ może zachodzić konieczność zaimplementowania nietypowego przetwarzania w aplikacji użytkownika
- DispatcherServlet wykrywa żądania typu multipart i kieruje ich przetwarzanie do implementacji interfejsu MultipartResolver jeżeli taka została dodana do kontenera
- MultipartResolver opakowuje HttpServletRequest w obiekt MultipartHttpServletRequest, który umożliwia odczytanie przesyłanego pliku

Upload plików (2)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- W Spring MVC zaimplementowano MultipartResolver w klasie CommonsMultipartResolver, wykorzystującej bibliotekę Apache Commons FileUpload (commons-fileupload.jar) która musi zostać dołączona do aplikacji

```
<bean id="multipartResolver"
  class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
  <property name="maxUploadSize" value="100000"/>
</bean>
```

Upload plików (3)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Przykład strony formularza do uploadu plików

```
<html>
<head>
  <title>Wysyłanie pliku</title>
</head>
<body>
<form method="post" action="/plik" enctype="multipart/form-data">
  <input type="text" name="name"/>
  <input type="file" name="file"/>
  <input type="submit"/>
</form>
</body>
</html>
```

Upload plików (4)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Przykładowy kod kontrolera do obsługi wysyłania pliku na serwer

```
@Controller
public class PlikController {

  @RequestMapping(value = "/plik", method = RequestMethod.POST)
  public String plik(@RequestParam("name") String name,
         @RequestParam("file") MultipartFile file) {
    if (!file.isEmpty()) {
      byte[] bytes = file.getBytes();
      // zapisanie pliku
      return "redirect:/plikWyslany.html";
    } else {
      return "redirect:/bladWysylania.html";
    }
  }

}
```

Obsługa wyjątków

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Spring MVC posiada mechanizm przekierowywania sterowania w sytuacjach wystąpienia określonych wyjątków. Mechanizm ten jest dużo bardziej elastyczny niż ten dostępny w Servlet API
- Do przekierowania sterowania dla wyjątków służą implementacje interfejsu HandleExceptionResolver

```
public interface HandlerExceptionResolver {

  ModelAndView resolveException(HttpServletRequest request,
    HttpServletResponse response, Object handler, Exception ex);

}
```

Obsługa wyjątków (2)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Dostępna jest klasa SimpleMappingExceptionResolver pozwalająca na zdefinowanie prostego mapowania nazw widoków do klasy wyjątku

```
<bean id="exceptionMapping"
  class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
  <property name="exceptionMappings">
    <props>
      <prop key="example.ExampleException">exampleError</prop>
      <prop key="java.lang.Exception">error</prop>
    </props>
  </property>
</bean>
```

Obsługa wyjątków (3)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Domyślnie w DispatcherServlet zdefiniowany jest DefaultHandlerExceptionResolver, w którym w przypadku określonych wyjątków Spring MVC ustawiany jest określony kod statusu HTTP w HttpServletResponse:
  - ConversionNotSupportedException - 500 (Internal Server Error)
  - HttpMediaTypeNotAcceptableException - 406 (Not Acceptable)
  - HttpMediaTypeNotSupportedException - 415 (Not Supported)
  - itd.

Obsługa wyjątków (4)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Aby zdefiniować obsługę wyjątku w kodzie kontrolera można posłużyć się adnotacją @ExceptionMapping

```
@Controller
public class PlikController {

  // ...

  @ExceptionHandler(IOException.class)
  public String wyjatek(IOException ex, HttpServletRequest request) {
    return "blad";
  }

}
```

Konfiguracja

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Aby ułatwić konfigurację Spring MVC w Spring w wersji 3 wprowadzono przestrzeń nazw XML o nazwie "mvc", w której zdefiniowano tagi ułatwiające proces konfiguracji:
  - **mvc:annotation-driven** - do kontenera dodany zostanie DefaultAnnotationHandlerMapping i AnnotationMethodHandlerAdapter

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:mvc="http://www.springframework.org/schema/mvc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
  http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
  http://www.springframework.org/schema/mvc
  http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

  <mvc:annotation-driven conversionService="..." />

</beans>
```

Konfiguracja (2)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- **mvc:interceptors** - umożliwia zdefiniowanie listy interceptorów dodawanych do wszystkich beanów implementujących HandlerMapping

```
<mvc:interceptors>

  <bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor" />

  <mvc:interceptor>
    <mapping path="/secure/*"/>
    <bean class="org.example.SecurityInterceptor" />
  </mvc:interceptor>

</mvc:interceptors>
```

Konfiguracja (3)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- **mvc:view-controller** - dodaje do kontenera ParameterizableViewController wykonujący bezpośrednie przekierowanie do widoku.

```
<mvc:view-controller path="/index" view-name="index"/>
```

Konfiguracja (4)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- **mvc:resources** - umożliwia wykorzystanie klasy ResourceHttpRequestHandler do zoptymalizowanej obsługi statycznych zasobów aplikacji

```
<mvc:resources mapping="/resources/**" location="/public-resources/"/>
```

Biblioteka tagów "form"

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Aby ułatwić definiowanie formularzy na stronach JSP odwzorowujących obiekty domenowe, przygotowano szereg użytecznych tagów zebranych w bibliotekę o nazwie "form"

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
```

Biblioteka tagów "form" (2)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Tag <form:form> pozwala zdefiniować formularz HTML reprezentujący wskazany obiekt w modelu
- Poszczególne rodzaje pól formularza HTML uzyskuje się poprzez użycie tagów: input, checkbox, radiobutton, hidden, textarea
- Odpowiadające pole w obiekcie domenowym określa się poprzez atrybut "path"

```
<form:form commandName="osoba" method="POST" action="/osoba">
  <form:label path="imie">Imię:</form:label>
  <form:input path="imie" /><br/>
  <form:label path="nazwisko">Nazwisko:</form:label>
  <form:input path="nazwisko" /></br>
  <input type="submit" value="Wyślij" />
</form:form>
```

Biblioteka tagów "form" (3)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Lista wyboru definiowana jest przez tag <form:select>, natomiast poszczególne wartości do wyboru można określić poprzez <form:option> lub <form:options>

```
<form:form commandName="osoba" method="POST" action="/osoba">
  <form:select path="tytul" items="${tytuly}"/>
</form:form>
```

```
<form:form commandName="osoba" method="POST" action="/osoba">
  <form:select path="tytul">
    <form:options items="${tytuly}">
    <form:option value="Inny">
  </form:select>
</form:form>
```

Biblioteka tagów "form" (4)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Błędy bindowania i walidacji można zamieścić na formularzu przy pomocy tagu <form:errors>

```
<form:form commandName="osoba" method="POST" action="/osoba">
  <form:label path="imie">Imię:</form:label>
  <form:input path="imie" />
  <form:errors path="imie" /><br/>
  <form:label path="nazwisko">Nazwisko:</form:label>
  <form:input path="nazwisko" />
  <form:errors path="nazwisko" /></br>
  <input type="submit" value="Wyślij" />
</form:form>
```