Wyższa Szkoła Bankowa
we Wrocławiu

Programowanie aplikacji w Java

Maciej Gowin

Zjazd 8 - dzień 2

Linki

Opis

https://maciejgowin.github.io/wsb-java/

Kod źródłowy przykładów oraz zadań

https://github.com/MaciejGowin/wsb-programowanie-aplikacji-java

Komunikacja klient-serwer

Komunikacja klient-serwer opiera się na zapytaniach klienta wysyłanych do sewera, który na nie oczekuje. Po przetworzeniu odpowiedź na zapytanie jest wysyłana z powrotem do klienta.

Komunikacja klient-serwer

Przykład: przeglądarka (klient) i wysyła zapytanie o zawartość strony internetowej do serwera, oczekując odpowiedzi.

Protokół HTTP

Protokół HTTP (ang. HyperText Transfer Protocol) opisujący komunikację przy pomocy hipertekstowych dokumentów w sieci WWW (World Wide Web).
Jest podstawa dla opisu wymiany danych, gdzie dane reprezentowane są przy pomocy dokumentów hipertekstowych zawierających hiperlinki wskazujące na inne zasoby.

Protokół ten jest bezstanowy i działa w modelu klient serwer przy użyciu par zapytań i odpowiedzi.

Ujednolica format zapytań (ang. request) oraz odpowiedzi (ang. request).

> Typy danych nie ogranicza się do formatu HTML.

Protokół HTTP: komunikacja

Zapytanie

Klient wysyła zapytanie do serwera zawierające:

- linie zapytania składająca się z metody zapytania, żądany URL oraz wersję protokołu,

```
GET /players.html HTTP/1.1
```

- nagłówki zapytania (ang. request headers, opcjonalne) opisujące metadane w formacie klucz-wartość,

```
Content-Type: application/json
Authorization: abc-123-def-456
```

- ciało wiadomości (opcjonalne).

Protokół HTTP: komunikacja

Odpowiedź

Serwer wysyła odpowiedź do klienta zawierającą:

- linię statusy zawierającą wersję protokołu oraz kod statusu odpowiedzi,

```
HTTP/1.1 200 OK
```

- nagłówki zapytania (ang. request headers, opcjonalne) opisujące metadane w formacie klucz-wartość,

```
Content-Type: application/json
Authorization: abc-123-def-456
```

- ciało wiadomości (opcjonalne).

Protokół HTTP: metody

Główne metody zapytań HTTP:

- GET: żądanie pobrania zasobu,
- HEAD: żądanie pobrania informacji o zasobie,
- POST: żądanie przyjęcia danych, używane do tworzenia zasobów,
- PUT: żądanie przyjęcia danych dla konkretnego zasobu, używane do aktualizacji konkretnych zasobów,
- DELETE: żądanie usunięcia zasobu,
- OPTIONS: żądanie pobrania opcji,
- PATCH: żądanie przyjęcia danych dla aktualizacji części zasobu.

Protokół HTTP: kody odpowiedzi

Najczęściej spotykane kody odpowiedzi serwera HTTP:

Kody informacyjne: 1xx

- 100 Continue

Kody powodzenia: 2xx

- 200 OK
- 201 Created
- 202 Accepted
- 204 No Content

Protokół HTTP: kody odpowiedzi

Kody przekierowania: 3xx

- 301 Moved Permanently
- 302 Found (poprzednio Moved temporarily)

Protokół HTTP: kody odpowiedzi

Kody błędu klienta: 4xx

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 406 Not Acceptable
- 408 Request Timeout
- 409 Conflict
- 415 Unsupported Media Type
- 429 Too Many Requests

Protokół HTTP: kody odpowiedzi

Kody błędu serwera: 5xx

- 500 Internal Server Error
- 501 Not Implemented
- 502 Bad Gateway
- 503 Service Unavailable
- 504 Gateway Timeout

Aplikacje Web

Do tej pory tworzyliśmy aplikacje desktopowe. Java dostarcza biblioteki pozwalające na tworzenie aplikacji działających na zasadzie serwera oczekującego za zapytania oraz wysyłającego odpowiedzi do klienta inicjalizującego zapytania.

Programowanie: przykład 80

Przykładowa aplikacja Web przy użyciu natywnych rozwiązań.

```
package pl.wsb.programowaniejava.maciejgowin.przyklad69;

import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
import com.sun.net.httpserver.HttpServer;

import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.util.concurrent.Executors;

public class Main {

    public static void main(String[] args) throws Exception {
        HttpServer server = HttpServer.create(new InetSocketAddress(8080), 0);
        server.createContext("/", new BasicHttpHandler());
        server.setExecutor(Executors.newFixedThreadPool(10));
        server.start();
        System.out.println("Server started");
    }

    static class BasicHttpHandler implements HttpHandler {
        @Override
        public void handle(HttpExchange httpExchange) throws IOException {
            if ("GET".equals(httpExchange.getRequestMethod())) {
                response(httpExchange, 200, "Requested: " + httpExchange.getRequestURI());
            } else {
                response(httpExchange, 405, "Invalid method");
            }
        }

        public void response(HttpExchange httpExchange, int code, String content) throws IOException {
            httpExchange.sendResponseHeaders(code, content.length());
            OutputStream outputStream = httpExchange.getResponseBody();
            outputStream.write(content.getBytes());
            outputStream.flush();
            outputStream.close();
        }
    }
}
```

Jakarta Enterprise Edition (JEE)

Jakarta Enterprise Edition (kiedyś: Java Enterprise Edition) to rozszerzenie standardowej Java SE o zbiór specyfikacji dla aplikacji korporacyjnych opartych na sewerach aplikacji.

Architektura JEE opata jest na komponentach.

Standard ten wprowadza szereg interfejsów, które muszą byc zaimplementowane przez serwery aplikacji z nim zgodne. Implementacją referencyjna jest Glassfish.

Jakarta Enterprise Edition (JEE)

Z punktu widzenia programisty tworzenie aplikacji JEE polega na tworzeniu komponentów wykorzystujących interfejsy zdefiniowane przez standard JEE.

Komponenty te są następnie osadzane i uruchamiane na serwerze aplikacji.

JEE: zmiana nazwy

W 2017 roku Oracle postanowiło pozbyć się praw do Java EE na rzecz Eclipse Foundation, przy czym język pozostaje dalej własnością Oracle.

W związku z obostrzeniami prawnymi Eclipse Foundation była zobligowana do zmiany nazwy Java EE, jako że Oracle ciągle ma prawa do nazwy "Java".

Na podstawie głosowania przyjęto nazwę Jakarta EE.

JEE: implementacje

Nazwa	Dostawca
GlassFish	Eclipse
Open Liberty	IBM
IBM WebSphere Liberty	IBM
WildFly	Red Hat
Red Hat JBoss Enterprise Application Platform	Red Hat
TomEE	Apache

JEE: główne składowe

Standard JEE definiuje szereg specyfikacji. Do najważniejszych należą:

Specyfikacje Web

- Servlet: niskopoziomowe wsparcie oraz zarządzanie zapytaniami HTTP.
- WebSocket: wsparcie dla połączeń WebSocket.
- Server Pages (JSP): tworzenie dynamicznych stron internetowych opartych o HTML, XML oraz innego typu dokumenty.
- Server Faces (JSF): tworzenie interfejsów użytkownika przy użyciu komponentów.
- Expression Language (EL): język pozwalający na zagnieżdżanie oraz wykonywanie wyrażeń na poziomie stron.

JEE: główne składowe

Specyfikacje Web serwisów

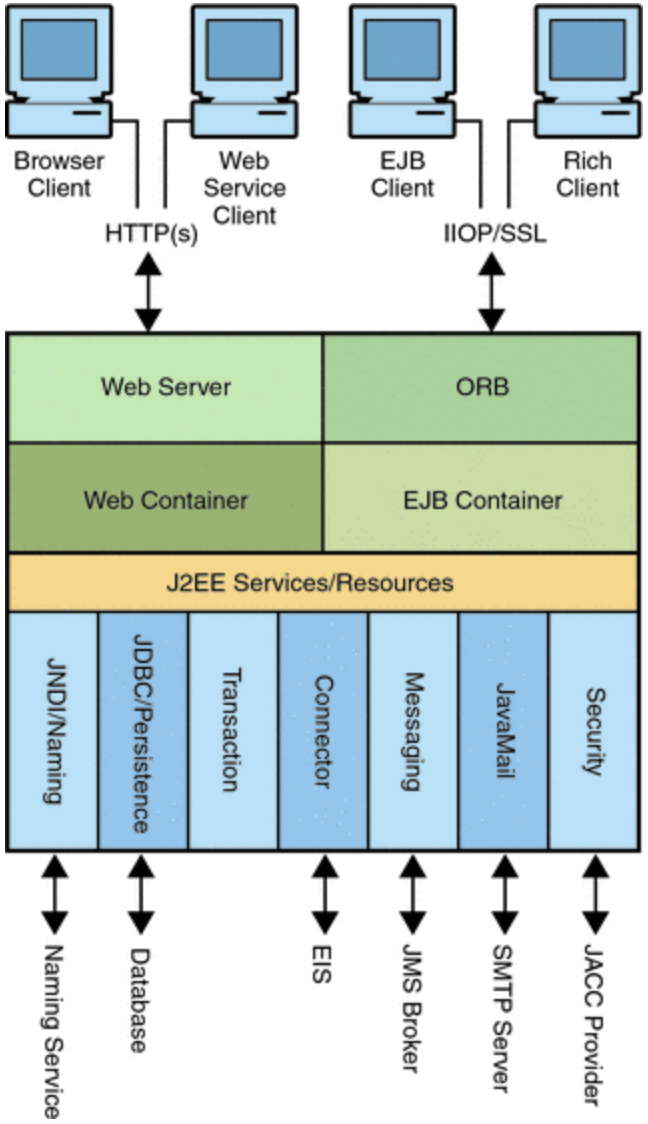
- RESTful Web Services (JAX-RS): wsparcie dla architektury serwisów REST.
- XML Web Services (JAX-WS): wsparcie dla architektury WebService.
- JSON Processing: zarządzanie informacjami w formacie JSON.
- JSON Binding (JAXB): mapowanie formatu JSON na obiekty (dwukierunkowe).
- XML Binding: mapowanie formatu XML na obiekty (dwukierunkowe).

JEE: główne składowe

Specyfikacje korporacyjne (enterprise)

- Contexts and Dependency Injection (CDI): wsparcie dla wstrzykiwanie zależności (ang. dependency injection).
- Enterprise Beans (EJB): wsparcie tworzenia komponentów działających w kontekście kontenera EJB, definiuje zależności pomiędzy ziarnami (ang. bean) a klientami oraz kontenerem.
- Persistence (JPA): mapowanie obiektowo relacyjne pomiędzy klasami Java oraz tabelami relacyjnych baz danych.
- Transactions (JTA): wsparcie dla transakcji, definiuje wysoko oraz nisko-poziomowe interfejsy.
- Messaging (JMS): komunikacja przy pomocy asynchronicznych wiadomości.
- Java Mail: obsługa maili.

JEE: główne składowe



Źródło: https://oracle.com/

JEE: problemy

JEE u podstawy miał wspierać tworzenie aplikacji poprzez odciążenie go od wykonywania powtarzalnych czynności oraz zapobiegać popełnianiu popularnych błędów.

Największą bolączką JEE są problemy z:

- utrzymywaniem ciężkich serwerów aplikacji,
- stabilność serwerów,
- konfiguracja spoczywająca na deweloperach,
- skomplikowana konfiguracja technologi EJB przy pomocy formatu XML.

JEE: Java Servlet

Częścią składową JEE jest Java Servlet. Serwlety są odpowiedzialne za przetwarzanie zapytań HTTP oraz generowanie odpowiedzi HTTP. Możemy o nich myśleć jak o specjalnego przeznaczenia klasach, obsługujących konkretny interfejs. Współpracują one z szeregiem technologi powiązanych takich jak JSP czy JSF.

Obsługę serwletów dostarcza kontener serwletów, który może działać niezależnie lub być częścią serwera aplikacyjnego JEE.

JEE: Apache Tomcat

Jednym z najpopularniejszych kontenerów serwletów zgodnym z Java Servlet jest Apache Tomcat.

Apache Tomcat jest dostępny pod adresem https://tomcat.apache.org/index.html.

Serwer aplikacyjny Apache TomEE jest rozszerzeniem Apache Tomcat implementującym Java EE (Tomcat + Java EE = TomEE).

JEE: Apache Tomcat

Po zainstalowaniu lub rozpakowaniu pliku dystrybucyjnego serwer Apache Tomcat. Może on zostać uruchomiony/zatrzymany przy pomocy poleceń:

Unix

```
$TOMCAT_HOME/bin/startup.sh
$TOMCAT_HOME/bin/shutdown.sh
```

Windows

```
$TOMCAT_HOME/bin/startup.bat
$TOMCAT_HOME/bin/shutdown.bat
```

Gdzie `$TOMCAT_HOME` to ścieżka instalacji aplikacji.

Po uruchomieniu serwera jest on domyślnie dostępny pod adresem:

```
http://localhost:8080/
```

JEE: Apache Tomcat

W celu uruchomienia nowej aplikacji na serwerze należy umieścić plik dystrybucyjny `war` w katalogu `$TOMCAT_HOME/webapps`.

Aplikacja zostanie udostępniona pod ścieżką zdefiniowaną poprzez nazwę pliku dystrybucyjnego aplikacji.

Dla przykładu: dla pliku `myapp.war` aplikacja zostanie udostępniona pod adresem `http://localhost:8080/myapp`.

Programowanie: przykład 81

Przykładowa implementacja serwletów ze standardu Java Servlets oraz uruchomienie w kontenerze Apache Tomcat.

```
package pl.wsb.programowaniejava.maciejgowin.przyklad81;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class RootServlet extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        resp.setStatus(200);
        resp.getWriter().print(HtmlGenerator.withBody("Root: " + req.getRequestURI() + " " + OffsetDateTime.now(), "yellow"));
        resp.getWriter().flush();
    }
}
```

Programowanie: przykład 81

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

    <servlet>
        <servlet-name>root</servlet-name>
        <servlet-class>
            pl.wsb.programowaniejava.maciejgowin.przyklad70.RootServlet
        </servlet-class>
    </servlet>

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

</web-app>
```

JEE: Java Servlet

Aplikacje webowe dystrybuowane są za pomocą plików `war`. Są one rozszerzonymi plikami `jar` o zmienionej strukturze wraz ze specyficznymi dla środowiska webowego ustawieniami.

Aby poinstruować aplikację budującą Apache Maven do wygenerowania plików `war` należy zmienić opcję pakowania:

```
<packaging>war</packaging>
```

Zmiana opcji wpływa też na cykl budowania i zmienia definicję kroków potrzebnych do zbudowania aplikacji.

JEE: Java Servlet

Zgodnie ze standardem Java Servlet definicja sposobu uruchomienia oraz konfiguracja serwletów odbywa się poprzez plik `web.xml`. Jest on umieszczany w katalogu `WEB-INF`.

Plik `web.xml` zawiera informacje o przyporządkowaniu (ang. mapping) ścieżek URL do serwletów, które je obsługują.

Serwer (kontener serwletów), taki jak Apache Tomcat, używa konfiguracji od odpowiedniej identyfikacji serwletów, które są w stanie obsłużyć dane żądanie oraz wywołania odpowiedniej metody z klasy obsługującej.

Spring Web MVC

Spring MVC (`spring-webmvc`) jest modułem Spring Framework służącym do budowy aplikacji webowych. Opiera się na wzorcu projektowym MVC (ang. Model-View-Controller).

Wspiera on wszystkie podstawowe części Spring Framework, takie jak Inversion of Control oraz Dependency Injection.

Moduł ten jest zależny od modułu `spring-context` .

Spring Web MVC

Spring MVC opiera się na idei Java Servlets. Aplikacje napisane przy użyciu Spring MVC są uruchamiane w kontenerach serwletów.

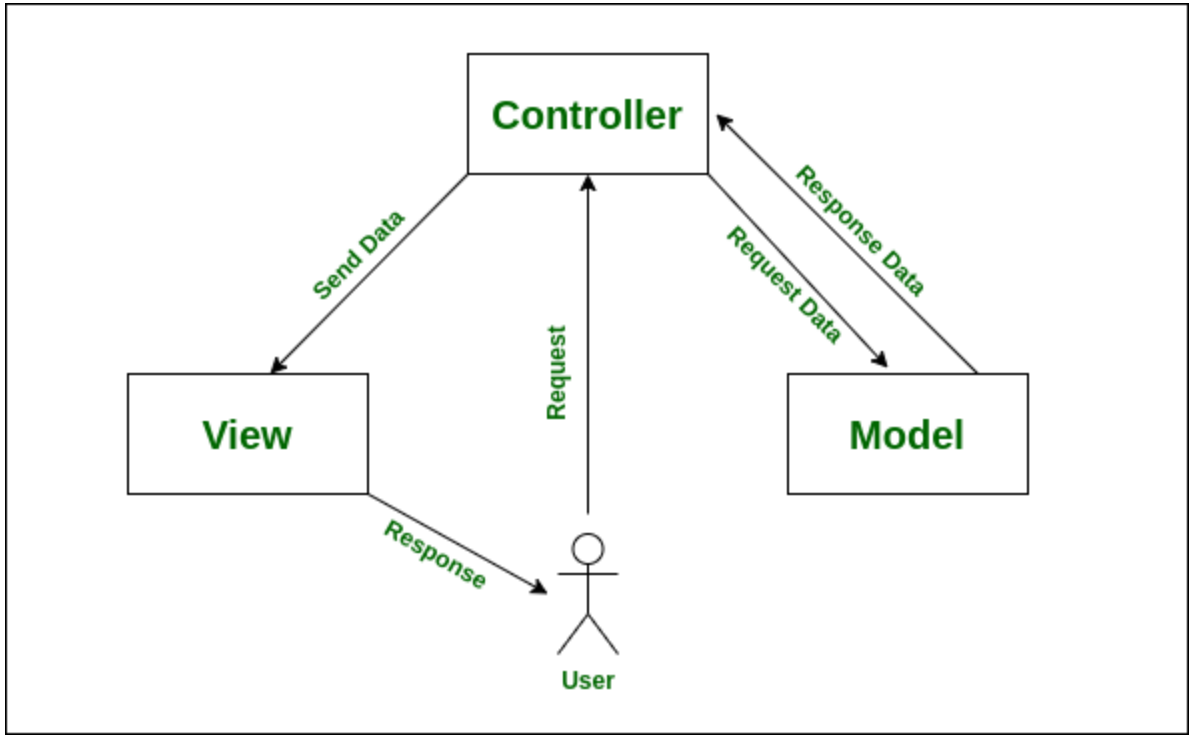
Do uruchomienia aplikacji Web opartych o Spring Framework wystarczy "lekki" kontener servletów taki jak: Apache Tomcat czy Eclipse Jetty.

Wzorce projektowe: MVC

MVC (ang. Model-View-Controller) jest wzorcem projektowym używanym przy projektowaniu interfejsów użytkownika, którego główną cechą jest podział programu na 3 podstawowe elementy. Choć elementy te współpracują ze sobą, rozdzielają warstwę reprezentacji danych od sposobu ich prezentacji.

Wzorzec ten jest szeroko używany podczas projektowania aplikacji webowych.

Wzorce projektowe: MVC



Źródło: https://www.geeksforgeeks.org/benefit-of-using-mvc/

Wzorce projektowe: MVC

Model

Model odpowiedzialny jest za definicje danych oraz zarządzanie danymi. Opisuje logikę programu, która jest niezależna od interfejsu użytkownika.

View

Widok reprezentuje dane modelu w określonym formacie. Możliwe jest definiowanie wielu widoków dla jednego modelu danych.

Controller

Kontroler przetwarza dane wejściowe oraz zamienia je na konkretne operacje na modelu, które następnie przesyłane są do widoku.

Spring Web MVC: DispatcherServlet

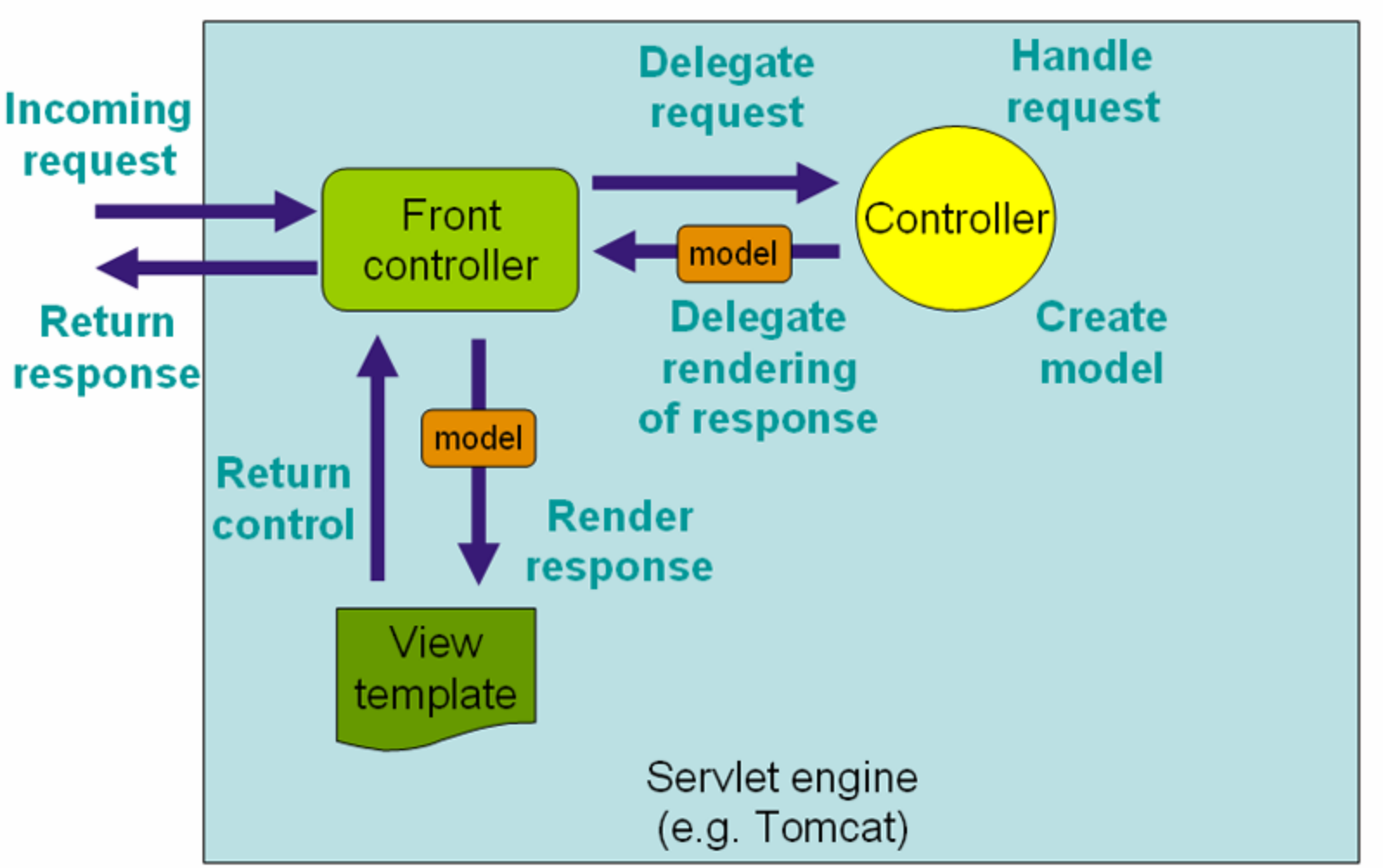
Spring Web MVC jest frameworkiem, który został zaprojektowany wokół centralnej klasy serwleta `DispatcherServlet` odpowiedzialnego za przekierowanie zapytań do mechanizmów obsługujących (ang. handler) wraz z pełną konfiguracją mapowań zapytań, rozpoznawania i wyszukiwania widoków (ang. views) oraz motywów (ang. themes), wsparcia dla tłumaczeń i lokalizacji (ang. locale).

Główny mechanizm obsługi zapytań jest oparty o adnotacje `@Controller` oraz `@RequestMapping` pozwalające na swobodną konfigurację obsługi oraz zachowań.

Należ podkreślić, że `DispatcherServlet` nie służy tylko do wsparcia przepływu danych w środowisku WEB. Jest on dodatkowo w pełni zintegrowany z kontenerem Spring IoC, dzięki czemu automatycznie wspiera wszystkie główne funkcjonalności znane z frameworka Spring.

Spring MVC: Front Controller

Rozwiązanie `DispatcherServlet` jest formą wzorca `Front Controller`



Źródło: https://docs.spring.io/

Programowanie: przykład 82

Aplikacje oparta na serwletach z kontekstem Spring Framework.

Spring Web MVC: konfiguracja web.xml

Podobnie jak w przypadku każdej aplikacji budowanej przy użyciu Java Servlet konfiguracja odbywa się poprzez plik `web.xml`.

W przypadku Spring Web MVC w podstawowej konfiguracji mamy do czynienia z jednym serwletem `DispatcherServlet`, który nie tylko definiuje mapowanie dla danego żądania, ale jest też odpowiedzialny za załadowanie pełnego kontekstu frameworku Spring.

Po dołączeniu `DispatcherServlet` automatycznie wyszukiwana jest konfiguracja kontekstu kontenera Spring w domyślnej lokalizacji pliku `<servlet-name>-context.xml`.

Spring Web MVC: konfiguracja kontekstu

Konfiguracja zawarta w pliku `<servlet-name>-context.xml` jest standardową konfiguracją kontenera IoC pozwalającą na znane już operacje takie jak: definiowanie ziaren, czy włączenie automatycznego wyszukiwania komponentów.

Konfiguracja ta posiada kilka kluczowych wskazówek mówiących o tym, gdzie zlokalizowane są komponenty frameworka Spring oraz jak powinny zostać załadowane.

Choć domyślnie kontekst Spring jest ładowany z pliku `<servlet-name>-context.xml` możemy też nadpisać domyślną konfiguracje oraz poinstruować `DispatcherServlet` do załadowania konfiguracji zdefiniowanej w kodzie Java (przy użyciu `AnnotationConfigWebApplicationContext`).

Spring Web MVC: konfiguracja z pominięciem web.xml

Rozpoczynając od standardu Java Servlet 3.0 konfiguracja serwletów może zostać przeniesiona do kodu Java co pozwala na pominięcie pliku `web.xml`. Kontener serwletów korzysta z mechanizmu SPI, który pozwala na dynamiczne ładowanie klas. Framework Spring w pełni wspiera ten mechanizm pozwalając na konfigurację aplikacji poprzez interfejs `WebApplicationInitializer`.

Podobnie w przypadku każdej innej konfiguracji kontekst IoC może ona zostać zdefiniowana w pliku XML lub z poziomu kodu Java.

Oba rozwiązania możemy używać zamiennie, ale będziemy koncentrować się na konfiguracji w pełni opartej na kodzie źródłowym Java.

Spring Web MVC: konfiguracja adnotacji MVC

`DispatcherServlet` automatycznie ładuje podstawowe komponenty, definiując podstawową konfigurację aplikacji. W przypadku nadpisania danego komponentu - domyślny zostanie ukryty.

Konfiguracja może zostań rozszerzona o najczęściej używane komponenty wspierające:

- konwersję typów,
- obsługę błędów,
- walidację,
- prezentację danych za pomocą formatów JSON, XML.

Spring Web MVC: konfiguracja adnotacji MVC

Włączenie funkcjonalności odbywa się za pomocą konfiguracji `web.xml`:

```
<mvc:annotation-driven />
```

Lub też z poziomu kodu Java.

```
@EnableWebMvc
```

Konfiguracja ta działą równolegle z `@ComponentScan` rozszerzając wsparcie aplikacji dla elementów specyficznych dla środowiska webowego.

Spring Web MVC: adnotacja @Controller

Adnotacja `@Controller` informuje, że dana klasa pełni rolę kontrolera. Definiuje ona ziarno o specjalnym przeznaczeniu.

Klasa kontrolera nie musi implementować żadnego interfejsu. Użycie adnotacji pozwoli na automatyczny i pełny dostęp do interfejsów specyficznych dla serwletów.

> Skanowanie komponentów (ang. component scan) automatycznie rozpoznaje adnotację i rejestruje klasę jako ziarno. Dzięki temu jest ona dostępna na poziomie kontenera Spring.

Spring Web MVC: @RequestMapping

Kontrolery są skanowane w poszukiwaniu adnotacji `@RequestMapping`, które definiują ścieżki zapytań oraz metody realizujące ich wywołanie.

```
@Controller
@RequestMapping("/articles")
public class AppointmentsController {

    @RequestMapping(method = RequestMethod.GET)
    @RequestMapping("/all")
    public String getAll() {
        return "page-articles-all";
    }

    @RequestMapping(method = RequestMethod.GET)
    @RequestMapping("/short")
    public String getShort() {
        return "page-articles-short";
    }
}
```

Spring Web MVC: @RequestMapping

Adnotacja `@RequestMapping` może zostać użyta na poziomie klasy lub też metody. Metody obsługują dane zapytania zdefiniowane poprzez ścieżkę.

W przypadku definicji ścieżki `@RequestMapping` na poziomie klasy oraz metody - ścieżka dostępu powstaje poprzez połączenie ścieżek dla klasy oraz metody.

Konfiguracja `@RequestMapping` wspiera definicję:

- ścieżki, np: "/articles/all",
- Http Verb, np: GET
- typów danych (ang. Media Types), np: "application/json".

Spring Web MVC: JSP

Użycie JSP pozwala na łatwiejszą definicję i generowanie dokumentów HTML oraz pełną integrację z danymi modelu.

Spring dostarcza bibliotekę znaczników pozwalającą na operacje na danych oraz definiującą podstawowe i najczęściej używane komponenty HTML.

Aby poprawnie skonfigurować obsługę JSP dla widoków Spring MVC, należy dostarczyć ziarno, definiujące sposób wyszukiwania widoków po nazwie opisujące ich nazwę oraz lokalizację. Dla przykładu używając konfiguracji opartej o plik konfiguracyjny XML:

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix">
        <value>/WEB-INF/jsp/</value>
    </property>
    <property name="suffix">
        <value>.jsp</value>
    </property>
</bean>
```

Spring Web MVC: @Controller - @RequestMapping - JSP

Podczas domyślnej definicji `@Controller` oraz `@RequestMapping` metody obsługujące zapytania mogą zwracać:

- `ModelAndView` : definiujący model oraz widok,
- `String` : definiujący logiczną nazwę widoku, w przypadku którego ustawienie modelu odbywa się poprzez parametr `ModelMap` .

Widoki są następnie wyszukiwane na podstawie definicji `ViewResolver` .

Spring Web MVC: adnotacje parametrów

Spring wspiera dodatkowe adnotacji pozwalające na pobranie dodatkowych informacji dotyczących zapytać. Adnotacje te są używane na poziomie argumentów metod obsługujących zapytania.

Do najczęściej używanych zaliczamy:

- `@PathVariable` : definicja elementu ścieżki zapytania,
- `@RequestParam` : definicja parametru łańcucha zapytań (ang. query string),
- `@RequestHeader` : definicja nagłówka zapytania,
- `@ModelAttribute` : definiuje model danych przesłanych podczas zapytania oraz pozwala na konwersję parametrów do modelu obiektu.

Programowanie: przykład 83

Konfiguracja projektu z poziomu kodu Java bez użycia konfiguracji `web.xml`, rozszerzenie o zmienne i parametry ścieżki oraz odczytanie modelu na poziomie JSP.

Programowanie: zadanie 31

Stwórz aplikację opartą na Spring Web MVC pozwalającą na konwersję walut.

Aplikacja powinna pozwalać na wpisanie wartość oraz waluty wejściowej i wyjściowej dla konwersji. Jako rezultat powinna zostać zwrócona wartość w walucie wyjściowej.

Konwersja walut oraz przeliczniki walut powinny być dostarczone przy użyciu ziaren zdefiniowanych na kontenerze.