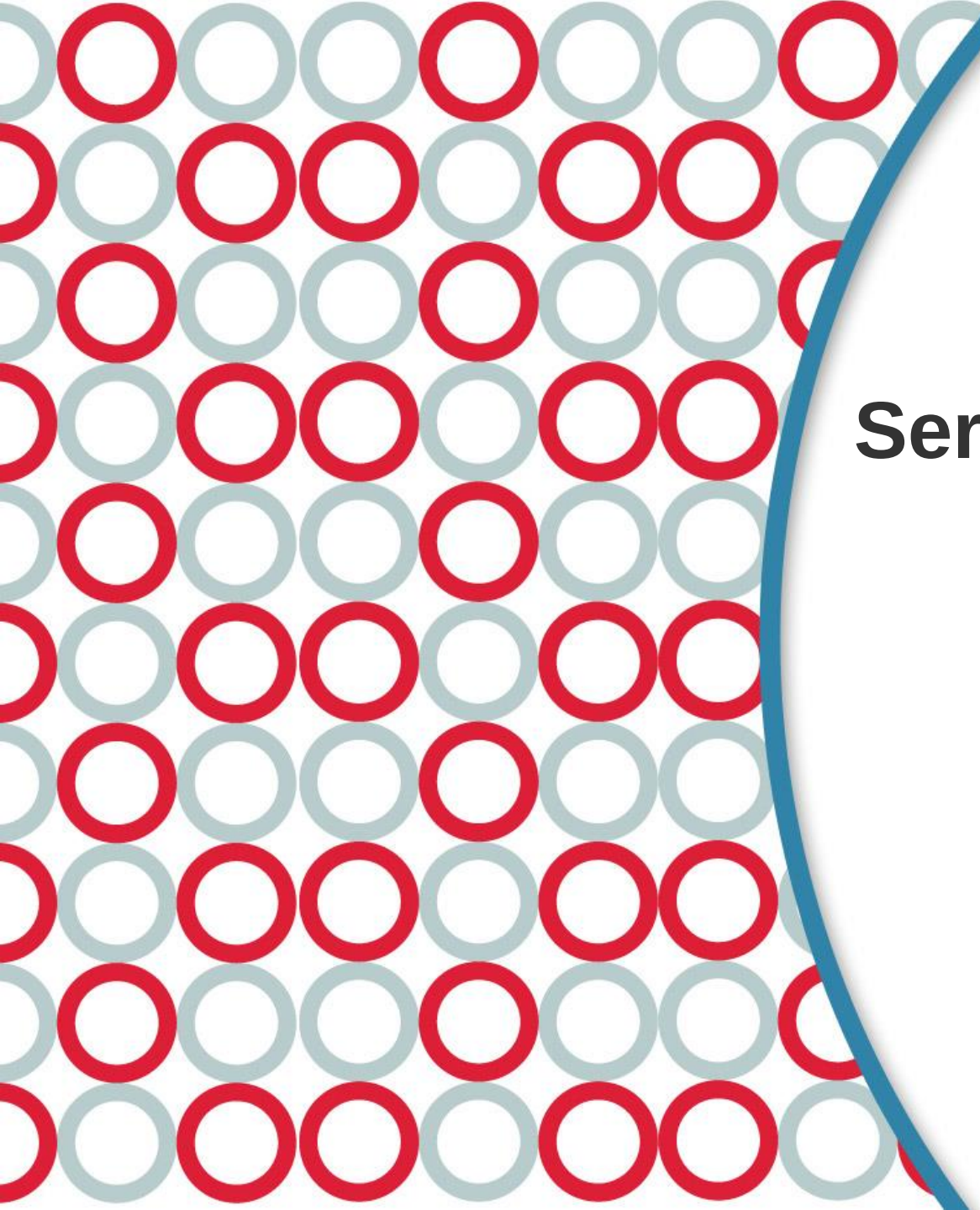


May 2022,
Course Java

Servlet Container, Servlets, JSPs

Trayan Iliev

tiliev@iproduct.org
http://iproduct.org

Where to Find the Code?

Java Academy projects and examples are available @GitHub:
https://github.com/iproduct/java-fundamentals-2022.git

Agenda for This Session

- Introduction
- HTTP Clients
- web.xml
- Servlets
- JSPs & Expression Language (EL)
- Tags (JSTL)

World Wide Web (WWW) – Main Concepts

- The idea for World Wide Web is suggested by Tim Berners-Lee in 1989 in CERN.
- Documents in World Wide Web, called ***web pages***, могат can contain *text, images, video, and other multimedia components*, and the connections between them are specified using *hyperlinks*.
- ***Web Sites*** include multiple connected *web pages* for a specific purpose
- They are **deployed** on a ***Web Server*** and are accessed using ***Web Client*** (*web browser – IE, Mozilla, Chrome*), using a protocol called: ***Hypertext Transfer Protocol (HTTP)***

HTTP Clients

- HTTP Clients allow to make HHTP requests to a Web Server:
- *Web browsers – Mozilla, Chrome, Edge etc.*
- *Generic HTTP clients – Postman, Insomnia, etc.*
- *Java HTTP Clients – Apache HttpClient, new Java 17 standard java.net.http.HttpClient*
- *Spring reactive HTTP client – Web Client*
- *Many more …*

Example: New Java 17 HttpClient

```
HttpClient client = HttpClient.newBuilder()
        .version(Version.HTTP_1_1)
        .followRedirects(Redirect.NORMAL)
        .connectTimeout(Duration.ofSeconds(20))
        .proxy(ProxySelector.of(new
InetSocketAddress("proxy.example.com", 80)))
        .authenticator(Authenticator.getDefault())
        .build();
HttpResponse<String> response = client.send(request,
BodyHandlers.ofString());
System.out.println(response.statusCode());
System.out.println(response.body());
```

Example: New Java 17 HttpClient - Sync

```
HttpClient client = HttpClient.newBuilder()
        .version(Version.HTTP_1_1)
        .followRedirects(Redirect.NORMAL)
        .connectTimeout(Duration.ofSeconds(20))
        .proxy(ProxySelector.of(new
InetSocketAddress("proxy.example.com", 80)))
        .authenticator(Authenticator.getDefault())
        .build();
HttpResponse<String> response = client.send(request,
BodyHandlers.ofString());
System.out.println(response.statusCode());
System.out.println(response.body());
```

Example: New Java 17 HttpClient - Async

```
HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create("https://foo.com/"))
        .timeout(Duration.ofMinutes(2))
        .header("Content-Type", "application/json")
        .POST(BodyPublishers.ofFile(Paths.get("file.json")))
        .build();
client.sendAsync(request, BodyHandlers.ofString())
        .thenApply(HttpResponse::body)
        .thenAccept(System.out::println);
```

Futures in Java 8 - I

- Future (implemented by FutureTask) – represents the result of an cancelable asynchronous computation. Methods are provided to check if the computation is complete, to wait for its completion, and to retrieve the result of the computation (blocking till its ready).
- RunnableFuture – a Future that is Runnable. Successful execution of the run method causes Future completion, and allows access to its results.
- ScheduledFuture – delayed cancelable action that returns result. Usually a scheduled future is the result of scheduling a task with a ScheduledExecutorService

Future Use Example

```
Future<String> future = executor.submit(
    new Callable<String>() {
        public String call() {
            return searchService.findByTags(tags);
        }
    }
);

DoSomethingOther();

try {
    showResult(future.get()); // use future result
} catch (ExecutionException ex) { cleanup(); }
```

Futures in Java 8 - II

- CompletableFuture – a Future that may be explicitly completed (by setting its value and status), and may be used as a CompletionStage, supporting dependent functions and actions that trigger upon its completion.
- CompletionStage – a stage of possibly asynchronous computation, that is triggered by completion of previous stage or stages (CompletionStages form Direct Acyclic Graph – DAG). A stage performs an action or computes value and completes upon termination of its computation, which in turn triggers next dependent stages. Computation may be Function (apply), Consumer (accept), or Runnable (run).

CompletableFuture Example - I

```
private CompletableFuture<String>
  longCompletableFutureTask(int i, Executor executor) {
    return CompletableFuture.supplyAsync(() -> {
      try {
         Thread.sleep(1000); // long computation :)
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
      return i + "-" + "test";
  }, executor);
}
```

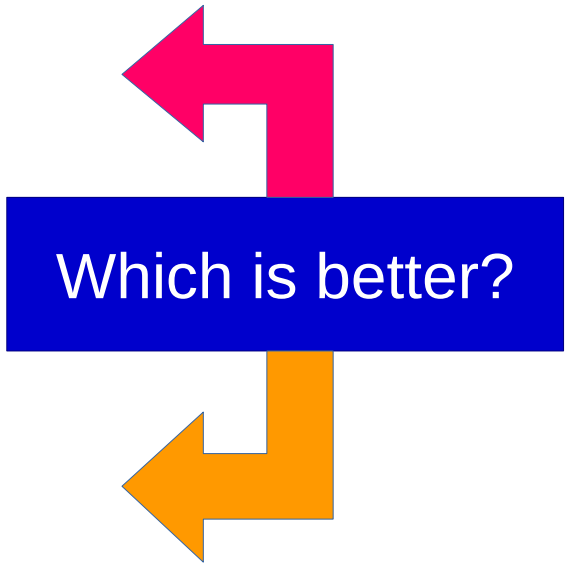
CompletableFuture Example - II

```
ExecutorService executor =  ForkJoinPool.commonPool();
//ExecutorService executor = Executors.newCachedThreadPool();
public void testlCompletableFutureSequence() {
List<CompletableFuture<String>> futuresList =
  IntStream.range(0, 20).boxed()
    .map(i -> longCompletableFutureTask(i, executor)
    .exceptionally(t -> t.getMessage()))
    .collect(Collectors.toList());
CompletableFuture<List<String>> results =
  CompletableFuture.allOf(
    futuresList.toArray(new CompletableFuture[0]))
    .thenApply(v -> futuresList.stream()
      .map(CompletableFuture::join)
      .collect(Collectors.toList())
    );
```

CompletableFuture Example - III

```
 try {
   System.out.println(results.get(10, TimeUnit.SECONDS));
 } catch (ExecutionException | TimeoutException
          | InterruptedException e) {
   e.printStackTrace();
 }
 executor.shutdown();
}


// OR just:
System.out.println(results.join());
executor.shutdown();
```



CompletionStage

- Computation may be Function (apply), Consumer (accept), or Runnable (run) – e.g.:

```
completionStage.thenApply( x -> x * x )
        .thenAccept(System.out::print )
        .thenRun( System.out::println )
```

- Stage computation can be triggered by completion of 1 (then), 2 (combine), or either 1 of 2 (either)
- Functional composition can be applied to stages themselves instead to their results using compose
- handle & whenComplete – support unconditional computation – both normal or exceptional triggering

CompletionStages Composition

```
public void testlCompletableFutureComposition() throws
InterruptedException, ExecutionException {
  Double priceInEuro = CompletableFuture.supplyAsync(()
     -> getStockPrice("GOOGL"))
    .thenCombine(CompletableFuture.supplyAsync(() ->
       getExchangeRate(USD, EUR)), this::convertPrice)
    .exceptionally(throwable -> {
       System.out.println("Error: " +
         throwable.getMessage());
            return -1d;
       }).get();

  System.out.println("GOOGL stock price in Euro: " +
     priceInEuro );
}
```

New in Java 9: CompletableFuture

- Executor **defaultExecutor()**
- CompletableFuture<U> **newIncompleteFuture()**
- CompletableFuture<T> **copy()**
- CompletionStage<T> **minimalCompletionStage()**
- CompletableFuture<T> **completeAsync(**
- Supplier<? extends T> supplier[, Executor executor]**)**
- CompletableFuture<T> **orTimeout(**
- long timeout, TimeUnit unit**)**
- CompletableFuture<T> **completeOnTimeout(**
- T value, long timeout, TimeUnit unit**)**

Ex.1: Async CDI Events with CF

```
@Inject @CpuProfiling private Event<CpuLoad> event; ...
IntervalPublisher.getDefaultIntervalPublisher(
  500, TimeUnit.MILLISECONDS) // Custom CF Flow Publisher
  .subscribe(new Subscriber<Integer>() {
    @Override public void onComplete() {}
    @Override public void onError(Throwable t) {}
    @Override public void onNext(Integer i) {
      event.fireAsync(new CpuLoad(
        System.currentTimeMillis(), getJavaCPULoad(),
areProcessesChanged()))
      .thenAccept(event -> {
        logger.info("CPU load event fired: " + event);
      }); } //firing CDI async event returns CF
    @Override public void onSubscribe(Subscription
subscription) {subscription.request(Long.MAX_VALUE);} });
```

Ex.2: Reactive JAX-RS Client - CF

```
CompletionStage<List<ProcessInfo>> processesStage =
  processes.request().rx()
    .get(new GenericType<List<ProcessInfo>>() {})
    .exceptionally(throwable -> {
      logger.error("Error: " + throwable.getMessage());
      return Collections.emptyList();
    });

CompletionStage<Void> printProcessesStage =
  processesStage.thenApply(proc -> {
    System.out.println("Active JAVA Processes: " + proc);
    return null;
});
```

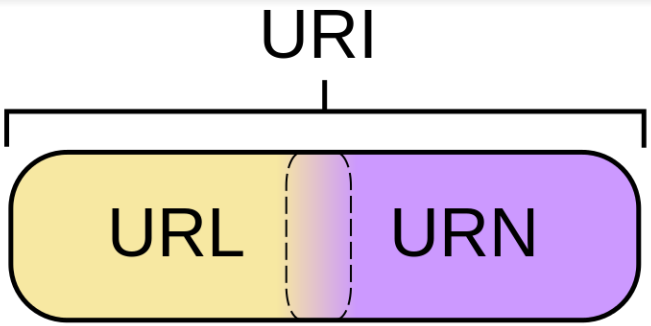
Ex.2: Reactive JAX-RS Client - CF

```
(- continues -)

printProcessesStage.thenRun( () -> {
  try (SseEventSource source =
    SseEventSource.target(stats).build()) {
      source.register(System.out::println);
      source.open();
      Thread.sleep(20000); // Consume events for 20 sec
  } catch (InterruptedException e) {
    logger.info("SSE consumer interrupted: " + e);
  }
})
.thenRun(() -> {System.exit(0);});
```

URLs, IP Addresses, and Ports

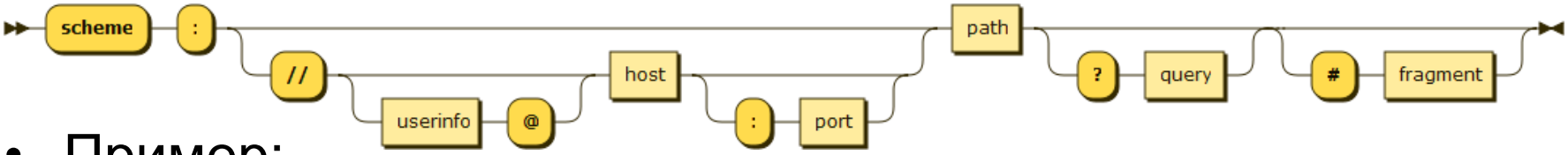
- Uniform Resource Identifier - URI:
  - Uniform Resource Locator – URL
  - Uniform Resource Name – URN



- Формат:

  scheme://domain:port/path?query_string#fragment_id

Схеми: http://, https://, file://, ftp://. news://, mailto:, telnet://

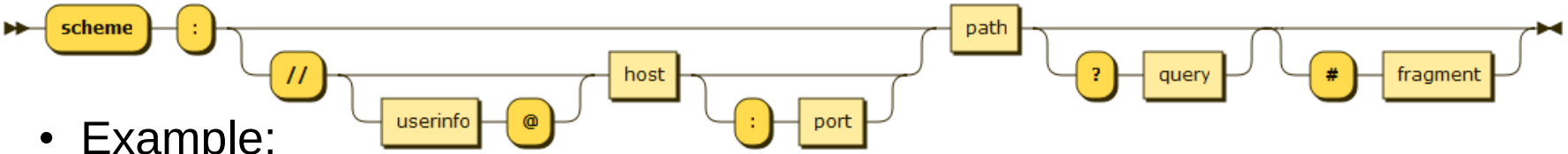


- Пример:
  http://en.wikipedia.org/wiki/Uniform_resource_locator#History

URL Structure (2)

- URLs pointing to **dynamic resources** (CGI scripts, Java Servlet / JSP, ASP, PHP, etc.) often include:
  - **?** list of request query parameters (query string) – e.g.: ?courseId=1211&role=student
  - **#** fragment identifier – defines the fragment (part) of the resource we want to access, used by asynchronous javascript page loading (AJAX) applications to encode the local page state – e.g. #view=fitb&nameddest=Chapter3
- The wole syntax is:

scheme://domain:port/path?query_string#fragment_id



- Example:
- http://en.wikipedia.org/wiki/Uniform_resource_locator#History

Types of Web Sites

- Static Web sites – show the same information for all visitors – can include hypertext, images, videos, navigation menus, etc.
- Dynamic Web sites – change and tune the content according to the specific visitor
  - server-site – use server technologies for dynamic web content (page) generation (data comes drom DB)
  - client-side – use JavaScript and asynchronous data actualization

Media Types. Multimedia

- Text – a linear sequence of character data
- Graphics – raster and vector images
- Animation – sequence of changing images (frames) with different frame-rates
- Audio – can be discretized analog signal (e.g. MP3) or just musical notes (MIDI)
- Video – different file formats, can be linear or interactive
- 3D Graphics / Animation – using 3D modelling and rendering techniques to achieve realistic, real-world like visualization
- And more – haptic feedback, smells, ...
- Multimedia – combining different media types in a coherent and consitent way to achieve better/more realistic user experience

Comparison between HTML and XML

- **HTML:**

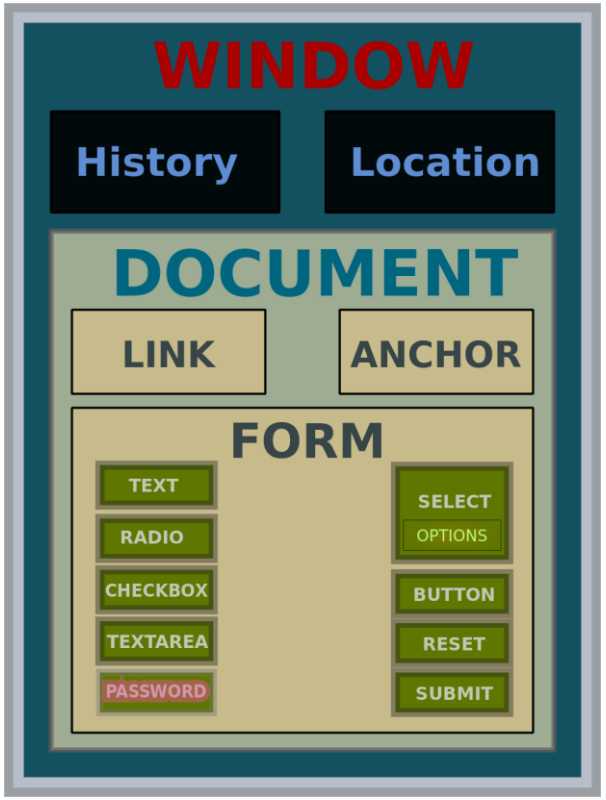
```
<html>
<head>
<title>Page of John Smith</title>
</head>
<body>
<p>John Smith</p>
</body>
</html>
```

- **XML:**

```
<?xml version="1.0"
      encoding="UTF-8"?>
<name>
<first>John</first>
<last>Smith</last>
</name>
```

Infomation hierachies – Document Object Models (DOM)

- Comaprison between XML and HTML – different purpose:
  - HTML – specific purpose (formatting and visualization)
  - XML – no specific purpose – different information structures
- HTML Document Object Model - DOM
- XML Document Object Model – DOM



Источник: Wikipedia, Автор: John Manuel – JMK, Лиценз: GNU Free Documentation License, Version 1.2,адрес: http://en.wikipedia.org/wiki/File:JKDOM.SVG

HTML Elements

- Tags, elements, attributes
- Document tree – types of nodes
- Element content – simple, mixed
- Document type – dictionary. HTML/XHTML/XML validation:
  - ***Document Type Definition (DTD), XML Schema***
- XML visualization in a web browser:
  - ***Cascading Style Sheets – CSS***
  - Interactivity – programming language ***JavaScript (EcmaScript)*** for excution of client side code in the browser

Basics of (X)HTML (1)

- XML markup and content – markup is enclosed between **<** and **>** (tags) or between **&** and **;** (entities), everyting other thsan that is **content**
- XML parser and user application – processes and analysis the markup and sends structured information to to user application
- Tags: ** , , **
- XML element – logical element in the XML document tree – simple or mixed content model:

** I amGeorge. **

Basics of (X)HTML (2)

- Attribute – key-value pair, included inside the tag:

```
<div id='15' style="color:red">
    Learn HTML for a day
</div>
```

- **HTML / XHTML декларации:**
  - **HTML5:** <!DOCTYPE html>
  - **HTML 4.01:** <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
  - **XHTML 1.0:** <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

Well Formatted HTML

- Root element of HTML Document should be **html.**
- HTML start with openeing and end with a closing tag.
- Tags should be properly nested (nested) – e.g.: ...Content...
- Attributes are inside the tag and always enclose their values in parenthesis.
- Tags and attributes are recommended to be in lowercase.
- Symbols **<**, **>**, **&**, **'**, **“** are changed to entities entities: **< > & ' "**

HTML 5 Base Template

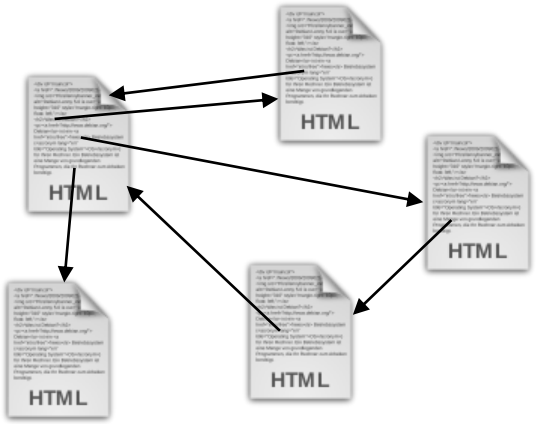
```
<!DOCTYPE html>
<html>
   <head>
      <title>Insert title here</title>
      <meta charset="UTF-8">
      <meta name="viewport"
          content="width=device-width, initial-scale=1.0">
      <style type="text/css">
         ... Example CSS ...
      </style>
   </head>
   <body>
       ... Example HTML ...
   </body>
</html>
```

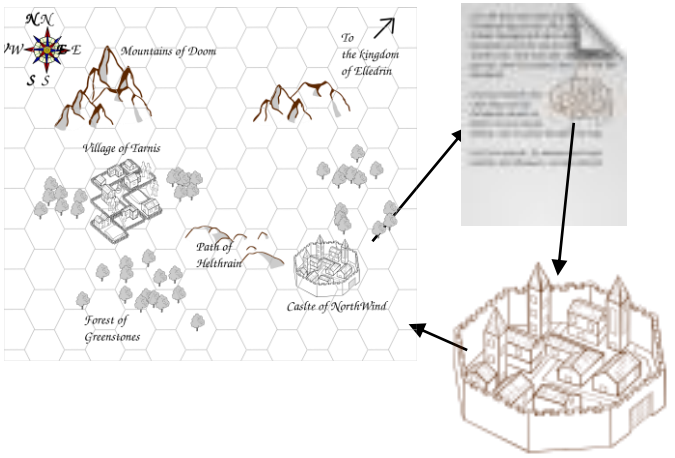
HTML 5 Extended Template

```
<!DOCTYPE html>
<html>
   <head>
      <title>Insert title here</title>
      <meta charset="UTF-8">
      ...
   </head>
   <body>
      <!-- Enable HTML 5 elements in IE 7+8 -->
      <!--[if lt IE 9]>
         <script src="dist/html5shiv.js"></script>
      <![endif]-->
      ... Example HTML ...
   </body>
</html>
```

Hypertext & Hypermedia

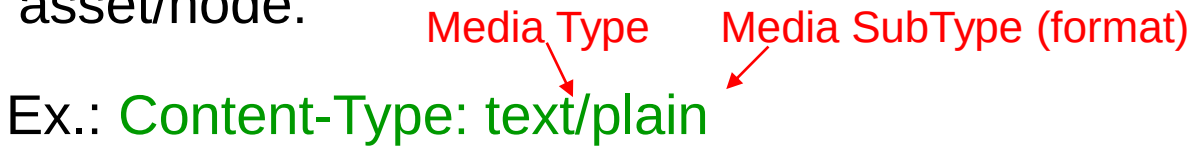
- Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text
- HTTP is the protocol to exchange or transfer hypertext
- Hypermedia - extension of the term hypertext, is a nonlinear medium of information which includes multimedia (text, graphics, audio, video, etc.) and hyperlinks of different media types (e.g. image or animation/video fragment can be linked to a detailed description.





Multipurpose Internet Mail Extensions (MIME)

- Different types of media are represented using different text/binary encoding formats – for example:
  - Text -> plain, html, xml ...
  - Image (Graphics) -> gif, png, jpeg, svg ...
  - Audio & Video -> mp3, ogg, webm ...
- Multipurpose Internet Mail Extensions (MIME) allows the client to recognize how to handle/present the particular multimedia asset/node:

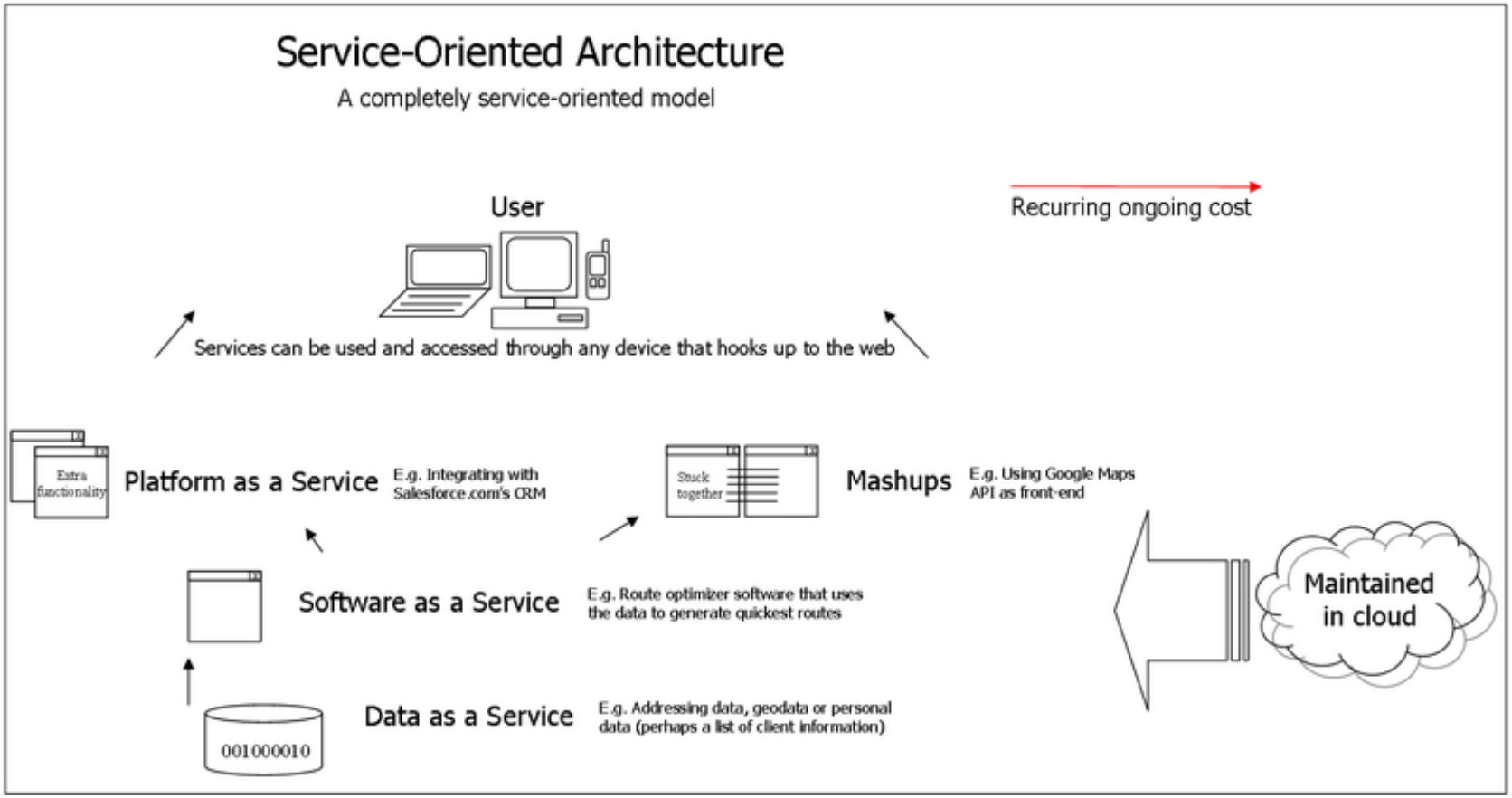


- More examples for standard MIME types: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types
- Vendor specific media (MIME) types: application/vnd.*+json/xml

REST Architectural Properties

- According to **Dr. Roy Fielding** [Architectural Styles and the Design of Network-based Software Architectures, 2000]:
  - Performance
  - Scalability
  - Reliability
  - Simplicity
  - Extensibility
  - Dynamic evolvability
  - Cusomizability
  - Configurability
  - Visibility
- All of them should be present in a desired Web Architecture and REST architectural style tries to preserve them by consistently applying several **architectural constraints**

Service Oriented Architecture (SOA). Mashups



Representational State Transfer (REST) [1]

- REpresentational State Transfer (REST) is an architecture for accessing distributed hypermedia web-services
- The resources are identified by URIs and are accessed and manipulated using an HHTP interface base methods (GET, POST, PUT, DELETE, OPTIONS, HEAD, PATCH)
- Information is exchanged using representations of these resources
- Lightweight alternative to SOAP+WSDL -> HTTP + Any representation format (e.g. JavaScript™ Object Notation – JSON)

Representational State Transfer (REST) [2]

- Identification of resources – URIs
- Representation of resources – e.g. HTML, XML, JSON, etc.
- Manipulation of resources through these representations
- Self-descriptive messages - Internet media type (MIME type) provides enough information to describe how to process the message. Responses also explicitly indicate their cacheability.
- Hypermedia as the engine of application state (aka **HATEOAS**)
- Application contracts are expressed as **media types** and [semantic] link realtions (**rel** attribute - RFC5988, "Web Linking")

[Source: http://en.wikipedia.org/wiki/Representational_state_transfer]

Hypermedia As The Engine Of Application State (HATEOAS) – New Link Header (RFC 5988) Example

```
Content-Length →1656

Content-Type →application/json

Link →<http://localhost:8080/polling/resources/polls/629>;
rel="prev"; type="application/json"; title="Previous poll",
<http://localhost:8080/polling/resources/polls/632>;
rel="next"; type="application/json"; title="Next poll",
<http://localhost:8080/polling/resources/polls>;
rel="collection"; type="application/json"; title="Polls
collection", <http://localhost:8080/polling/resources/polls>;
rel="collection up";  type="application/json"; title="Self
link", <http://localhost:8080/polling/resources/polls/630>;
rel="self"
```

Simple Example: URLs + HTTP Methods

Uniform Resource Locator (URL)	GET	PUT	POST	DELETE
Collection, such ashttp://api.example.com/comments/	**List the URIs and perhaps other details of the collection's members.**	**Replace the entire collection with another collection.**	**Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.**	**Delete the entire collection.**
Element, such ashttp://api.example.com/comments/11	**Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.**	**Replace the addressed member of the collection, or if it does not exist, create it.**	Not generally used. Treat the addressed member as a collection in its own right and create a new entry in it.	**Delete the addressed member of the collection.**

Source: https://en.wikipedia.org/wiki/Representational_state_transfer

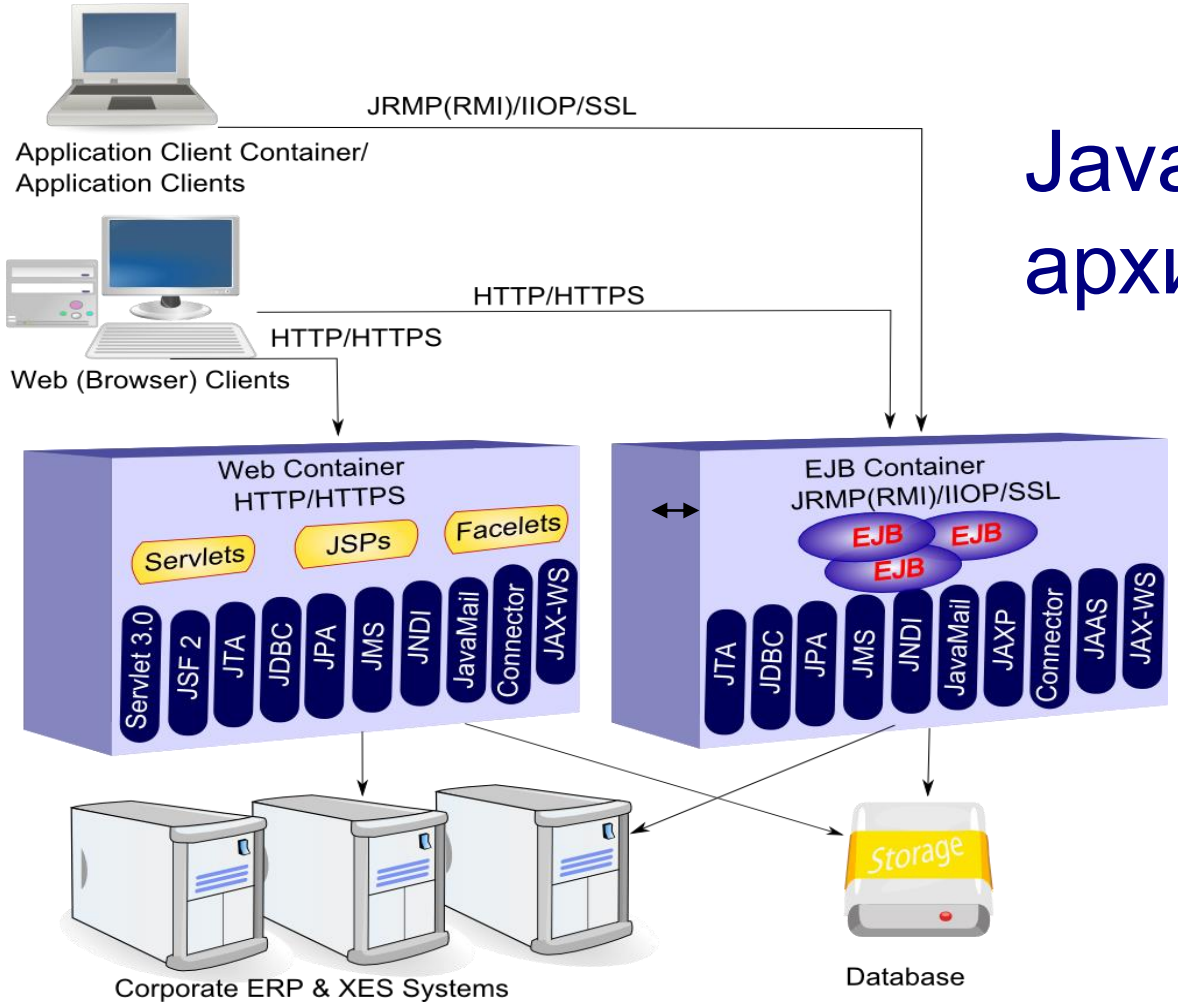
Java™ Platform, Enterprise Edition 8

- Java™ Platform Enterprise Edition е спецификация разработена от Oracle® (Sun) заедно с партньори като BEA Systems, Borland, E.piphany, Hewlett-Packard, IBM, Inria, Novell, Red Hat, SAP, Sybase, Apache и др. за да улесни създаването на надеждни, конфигурируеми, мащабируеми, лесно оперирaщи помежду си и платформено-независими сървърни приложения и компоненти на езика Java™.
- Базирана е върху Java™ SE
- Вече част от Jakarta (Ecpipse Foundation) -> Jakarta EE 9

Java™ Platform, Enterprise Edition

- Java™ EE дефинира:
  - Програмни модели за разработка на приложения и програмни интерфейси (API) за създаване на разпределени, многокомпонентни приложения, тяхното пакетиране и инсталиране
  - Множество от готови интегрирани услуги и APIs намаляващи времето за разработка, сложността на приложенията и подобряващи тяхната производителност
  - Обща логическа архитектура интегрираща различни компоненти и контейнери за компоненти

Java™ EE архитектура



Java™ EE 6/7Архитектура (1)

- Основни компоненти в Java™ EE архитектурата:
  - Базирана върху Java™ SE
  - Java™ EE компоненти
    - Web Components – Servlets, JSPs, Facelets, Web Services (SOAP, REST)
    - EJB™ Componenets – Session EJBs, Persistence Entities, Message Driven EJBs
  - Java™ EE среда за изпълнение
    - Сървъри
    - Контейнери – Web и EJB контейнери
    - Application Client контейнери

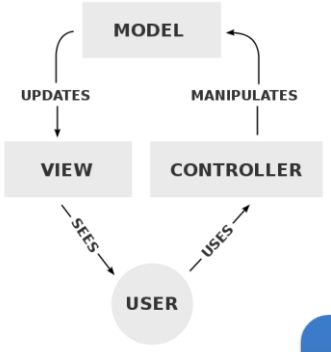
Основни API в Java™ EE

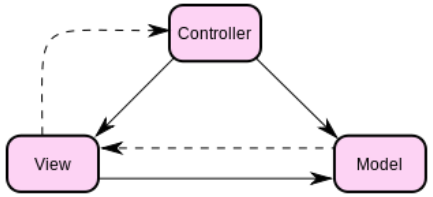
[https://en.wikipedia.org/wiki/Java_EE_version_history#Java_EE_7_.28June_12.2C_2013.29]

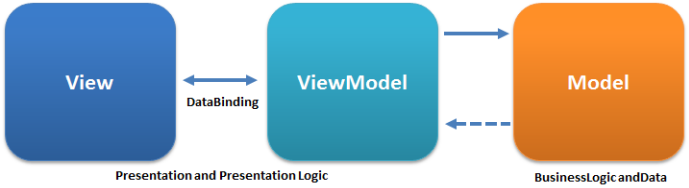
- javax.ejb.* - EJB
- javax.enterprise.inject.* - CDI
- javax.enterprise.context.* - CDI
- javax.jms.* - JMS
- **javax.servlet.* - Servlet API, JSP, JSTL, Expression Language (EL)**
- javax.faces.* - JSF, Facelets, Components
- javax.mail – Java Mail
- javax.persistence – JPA
- javax.transaction – JTA
- javax.validation – Validation API
- javax.xml.stream - StAX
- javax.resource.* - Java EE Connector Architecture
- javax.jws - JAX-WS
- javax.ws.rs - JAX-RS (RESTful Services)

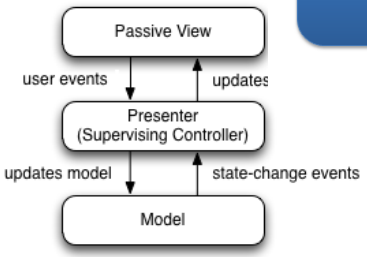
MVC Comes in Different Flavors - II

- MVC
- MVVM
- MVP



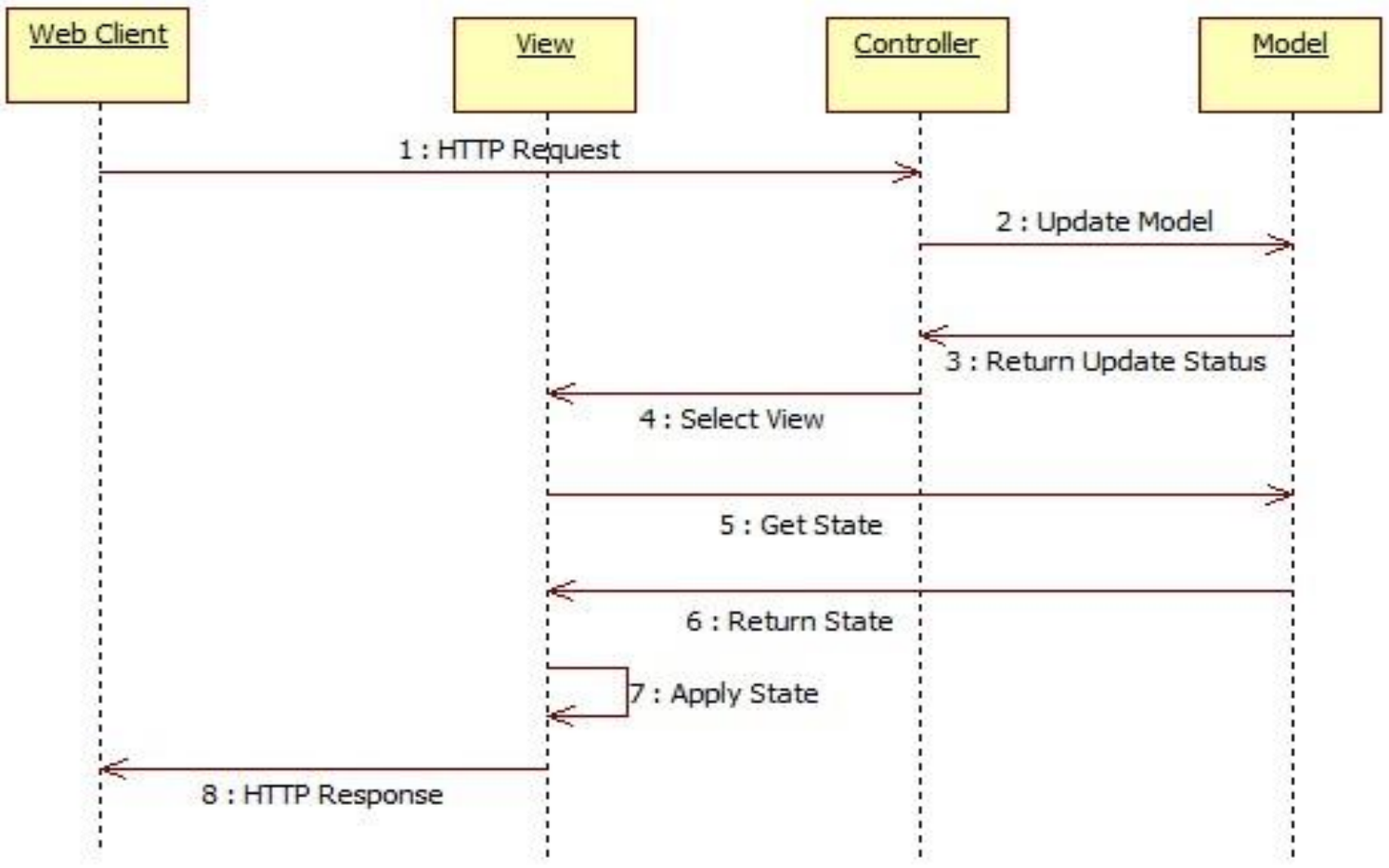




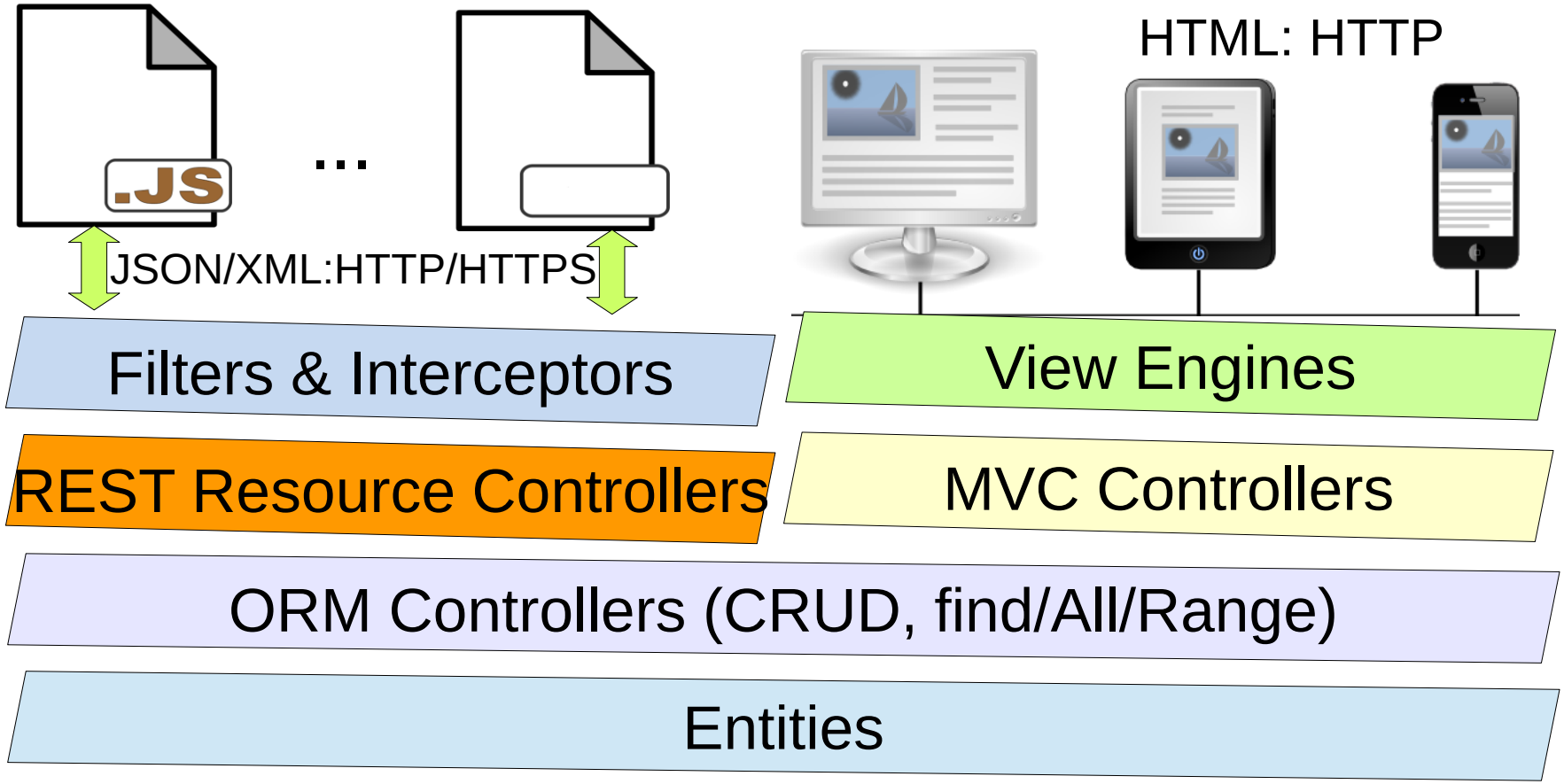


Sources:https://en.wikipedia.org/wiki/Model_View_ViewModel#/media/File:MVVMPattern.png, https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter#/media/File:Model_View_Presenter_GUI_Design_Pattern.png
License: CC BY-SA 3.0, Authors:Ugaya40, Daniel.Cardenas

Web MVC Interactions Sequence Diagram



N-Tier Architectures



Apache Tomacat v9 Web Server

- Поддържа Servlet 4.0 API
- Стартиране/ спиране: **bin\startup.bat** или bin\catalina.bat start (Windows) (*.sh – Unix, Linux)
- Директорийна структура:
  - bin - изпълними файлове и стартиращи/стоп скриптове
  - conf - конфигурационни файлове
  - lib - библиотеки и класове
  - logs - Log и Output файлове
  - webapps – уеб приложения зареждани автоматично
  - work – временни работни директории за уеб прилож.
  - temp -използва се от JVM съхранение на temp файлове

Структура на Java™ уеб приложение (WAR)

- **Web Archive (WAR) root – /**
  - index.html, other.htm, ... – стандартни HTML страници
  - index.jsp, other.jsp, ... – JavaServer™ Pages (JSP) страници
  - js / myscript.js, ... – директория съдържаща JavaScript ресурси
  - css / main.css, ... – директория съдържаща CSS ресурси
  - images / logo.png, ... – директория с граф. изображения, снимки
  - **WEB-INF**
    - web.xml – конфигурационен файл за конкретната инсталация на уеб приложението (deployment descriptor)
    - glassfish-web.xml – опционален конфигурационен файл с настройки специфични за конкретния уеб сървър
    - **lib** – директория, включва .jar библиотеки с java класове
    - **classes** – директория, включва компилираните java класове

Дескриптор на разпространение web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
    id="WebApp_ID" version="4.0">
    <servlet>
        <servlet-name>DispatcherServlet</servlet-name>
        <servlet-class>invoicing.DispatcherServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>DispatcherServlet</servlet-name>
        <url-pattern>/DispatcherServlet</url-pattern>
    </servlet-mapping>
```

Дескриптор на разпространение web.xml (2)

```
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Структура на дескриптора на разпространение web.xml

- icon
- display-name
- description
- distributable
- context-param
- filter
- filter-mapping
- listener
- servlet
- servlet-mapping
- session-config
- mime-mapping
- welcome-file-list
- error-page
- taglib
- resource-env-ref
- resource-ref
- security-constraint
- login-config
- security-role
- env-entry
- ejb-ref
- ejb-local-ref

Сървърна поддръжка

- Java™ EE 6 сървъри:
  - GlassFish/Payara (https://payara.gitbooks.io/payara-server/content/)
  - RedHat's JBoss/WildFly (https://wildfly.org/)
  - Apache TomEE (https://tomee.apache.org/)
  - Oracle's WebLogic
  - IBM's WebSphere
  - SAP Netweaver
  - Resin, JOnAS, JEUS, . . .
- Олекотени уеб сървъри:
  - Apache Tomcat, Jetty, Undertow и много други

Жизнен цикъл на сървлета

- Зареждане на класа на сървлета от уеб контейнера
- Уеб контейнерът създава инстанция на класа
- Инициализира инстанцията на сървлета като извиква метода **init()**
- Извиква **service()** метода за всяка заявка подавайки **request** и **response** обекти като аргументи
- Преди да деактивира (премахне) сървлета контейнера извиква неговия метод **destroy()**

Основна структура на сървлети (1)

- Основни методи на класа HttpServlet:
  - doGet – за HTTP GET заявки
  - doPost – за HTTP POST заявки
  - doPut – за HTTP PUT заявки
  - doDelete – за HTTP DELETE заявки
  - init and destroy – за управление на ресурсите
  - getServletInfo – дава информация за сървлета
  - service – получава всички HTTP заявки и играе ролята на диспечер – не трябва да се предефинира директно

Основна структура на сървлети (2)

- Основни методи на класа GenericServlet:
  - getInitParameter, getInitParameterNames – дават възможност за декларативно конфигуриране
  - getServletConfig – връща обект от тип ServletConfig, който съдържа инфромация предавана от уеб контейнера на сървлета
  - getServletContext – връща обект от тип ServletContext дефиниращ множество методи, които сървлетът използва за да комуникира с контейнера – например да получи MIME типа на файл, да диспечеризира заявки или да пише в log файл

Клас HttpServlet – методи: doGet, doPost

```
response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println(docType +"<HTML>\n" +
    "<HEAD><TITLE>Hello</TITLE></HEAD>\n" +

     "<BODY BGCOLOR=\"#FDF5E6\">\n" +

    "<H1>Hello</H1>\n" +       "</BODY></HTML>");
```

Инсталиране на сървлети

- Инсталиране и конфигуриране на сървлет и JSP™ софтуер
- Динамична регистрация на сървлети – анотации @WebServlet и @WebInitParam.
- Уеб компоненти и WAR архиви
- Структура на дескриптора на разпространението (deployment descriptor) – web.xml

Пример за сървлет с използване на @WebServlet и @WebInitParam анотации (1)

```
@WebServlet(urlPatterns = "/HelloWorld",
    initParams = {
    @WebInitParam(name="bgcolor", value="yellow"),
    @WebInitParam(name="message", value="Hello from Servlet 3.0")
    })
public class HelloWorld extends HttpServlet {
    private String bgcolor;
    private String  message;

public void init() throws ServletException {
    bgcolor = getInitParameter("bgcolor");
    bgcolor = (bgcolor != null) ? bgcolor: "white";
    message = getInitParameter("message");
    message = (message != null) ? message: "Hello";
}
```

Пример за сървлет с използване на @WebServlet и @WebInitParam анотации (2)

```
   protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<html>");
      out.println("<head>");
      out.println("<title>Hello World!</title>");
      out.println("</head>");
      out.println("<body bgcolor='" + bgcolor + "'>");
      out.println("<h1>" + message + "</h1>");
      out.println("</body>");
      out.println("</html>");
   }
}
```

Дескриптор на разпространение web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.5" 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">
    <servlet>
        <servlet-name>DispatcherServlet</servlet-name>
        <servlet-class>invoicing.DispatcherServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>DispatcherServlet</servlet-name>
        <url-pattern>/DispatcherServlet</url-pattern>
    </servlet-mapping>
```

Дескриптор на разпространение web.xml (2)

```
<session-config>
    <session-timeout>
        30
    </session-timeout>
</session-config>
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

Структура на дескриптора на разпространение web.xml

- icon
- display-name
- description
- distributable
- context-param
- filter
- filter-mapping
- listener
- servlet
- servlet-mapping
- session-config
- mime-mapping
- welcome-file-list
- error-page
- taglib
- resource-env-ref
- resource-ref
- security-constraint
- login-config
- security-role
- env-entry
- ejb-ref
- ejb-local-ref

Обработка на параметри на форми с помощта на сървлети.

- Методи на **javax.servlet.ServletRequest** за обработка на параметри на форми:
  - **getParameter(String name)** – връща стойността на параметър с даденото име във формата като низ (String)
  - **getParameterValues(java.lang.String name)** – връща всички стойности на параметъра като масив от низове
  - **getParameterNames()** - връща java.util.Enumeration<String> с имената на всички параметри във формата
  - **getParameterMap()** - връща асоциативен списък (java.util.Map<String, String[]>) с всички имена и стойности на параметри във формата

Отстраняване на грешки (debugging) на сървлети

Методи за откриване и отстраняване на грешки в сървлети:

- **System.out.println()** и **javax.servlet.GenericServlet.log()** – отпечатване на междинни резултати за диагностика на работата на сървлета в **log** файла на сървъра
- Използване на инструментите за отстраняване на грешки (**Debugger tools**) на интегрираната среда за разработка (IDE) – например Eclipse и NetBeans предлагат такива
- Използване на **Firebug** и други подобни инструменти за инспекция на HTML и JavaScript кода, който се визуализира и изпълнява вътре в уеб браузъра, както и за преглед на съдържанието на HTTP заявките и отговорите от сървъра
- Използване на **отделни класове** за отделните задачи

Java™: Заглавни части на HTTP заявки

- Методи на класа HttpServletRequest за достъп до заглавните части:
  - **getCookies()**
  - **getAuthType()**
  - **getRemoteUser()**
  - **getContentLength()**
  - **getContentType()**
  - **getDateHeader()**
  - **getIntHeader()**
  - **getHeaderNames()**
  - **getHeader()**
  - **getHeaders()**

Java™: Заглавни части на HTTP заявки

- Основни параметри на HTTP заявката:
  - **getMethod()**
  - **getRequestURI()**
  - **getQueryString()**
  - **getProtocol()**

Заглавни части на HTTP заявки

- В **HTTP 1.0** всички заглавни части са опционални
- В **HTTP 1.1** са опционални всички заглавни части без **Host**
- Необходимо е винаги да се проверява дали съответната заглавна част е различна от **null**

Request Header Example

host	localhost:8080
user-agent	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:21.0) Gecko/20100101 Firefox/21.0
accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
accept-language	en,bg;q=0.7,en-us;q=0.3
accept-encoding	gzip, deflate
dnt	1
referer	http://localhost:8080/examples/servlets/index.html
connection	keep-alive

Структура на заявка

GET */context/Servlet* **HTTP/1.1**

Host: *Client_Host_Name*

Header2: *Header2_Data*

...

HeaderN: *HeaderN_Data*

<Празен ред>

POST */context/Servlet* **HTTP/1.1**

Host: *Client_Host_Name*

Header2: *Header2_Data*

...

HeaderN: *HeaderN_Data*

<Празен ред>

POST_Data

Структура на отговор на заявка

HTTP/1.1 200 OK

Content-Type: text/html

Header2: *Header2_Data*

...

HeaderN: *HeaderN_Data*

<Празен ред>

<!DOCTYPE *Document_Type_Definition***>**

```
<html>
    <head>
        <title>...</title>
    </head>
    <body>
        ...
    </body>
</html>
```

Структура на отговор на заявка

HTTP/1.1 200 OK

Content-Type: application/json

Header2: *Header2_Data*

...

HeaderN: *HeaderN_Data*

<Празен ред>

```
[{  "id":1,
    "name":"Novelties in Java EE 7 ...",
    "description":"The presentation is ...",
    "created":"2014-05-10T12:37:59",
    "modified":"2014-05-10T13:50:02",
 },
 {  "id":2,
    "name":"Mobile Apps with HTML5 ...",
    "description":"Building Mobile ...",
    "created":"2014-05-10T12:40:01",
    "modified":"2014-05-10T12:40:01",
 }]
```

Заглавни части на HTTP заявка - RFC2616

- Accept
- Accept-Charset
- Accept-Encoding
- Accept-Language
- Authorization
- Connection
- Content-Length
- Cookie
- Host
- If-Modified-Since
- If-Unmodified-Since
- Referer
- User-Agent

Статус кодове на отговор на заявка

- 100 Continue
- 101 Switching Protocols
- 200 OK
- 201 Created
- 202 Accepted
- 203 Non-Authoritative Information
- 204 No Content
- 205 Reset Content
- 301 Moved Permanently
- 302 Found
- 303 See Other
- 304 Not Modified
- 307 Temporary Redirect
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found

Статус кодове на отговор на заявка

- **405 Method Not Allowed**
- **415 Unsupported Media Type**
- **417 Expectation Failed**
- **500 Internal Server Error**
- **501 Not Implemented**
- **503 Service Unavailable**
- **505 HTTP Version Not Supported**

Java™: Методи за установяване на заглавни части на HTTP отговори

- **setHeader(String headerName, String headerValue)**
- **setDateHeader(String header, long milliseconds)**
- **setIntHeader(String header, int headerValue)**
- **setContentType(String mimeType)**
- **setContentLength(int length)**
- **addCookie(Cookie c)**
- **sendRedirect(String address)**

Заглавни части на HTTP отговори

- **Allow**
- **Cache-Control**
- **Pragma**
- **Connection**
- **Content-Disposition**
- **Content-Encoding**
- **Content-Language**
- **Content-Length**
- **Content-Type**
- **Expires**
- **Last-Modified**
- **Location**
- **Refresh**
- **Retry-After**
- **Set-Cookie**
- **WWW-Authenticate**

Обектни обхвати (Scopes)

- **Web context** – клас: **javax.servlet.ServletContext**, съдържа уеб компоненти достъпни за цялото приложение
- **Session** – клас: **javax.servlet.http.HttpSession**, съдържа уеб компоненти достъпни в рамките на потребителската сесия
- **Request** – клас: **javax.servlet.ServletRequest**, съдържа уеб компоненти достъпни в рамките на HTTP заявката
- **Page** – клас: **javax.servlet.jsp.JspContext**, съдържа обекти достъпни в рамките на JSP страницата

Конструиране на HttpServletResponse чрез вграждане на ресурси

- Създаване на обект от тип **RequestDispatcher**:

• RequestDispatcher dispatcher = getServletContext().
• getRequestDispatcher("/datatable");

- Включване на маркъп генериран от друг уеб ресурс в отговора (HttpServletResponse) – include:

• if (dispatcher != null) dispatcher.**include**(request, response);

- Цялостно делегиране генерирането на отговор на друг уеб ресурс – forward:

• if (dispatcher != null) dispatcher.**forward**(request, response);

Новости в JDBC™ 4.1 (Java 7): try-with-resources

java.sql.Connection, java.sql.Statement и java.sql.ResultSet имплементират интерфейса **AutoCloseable**:

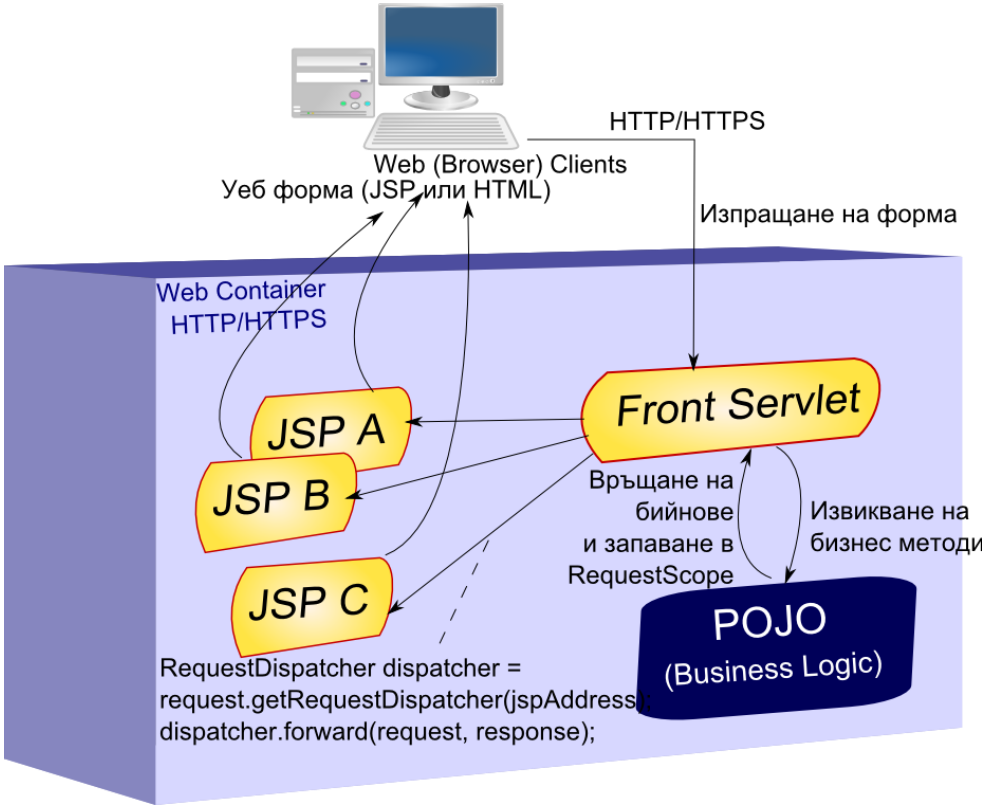
```
Class.forName("com.mysql.jdbc.Driver");                    //Load MySQL DB driver
try (Connection c = DriverManager.getConnection(dbUrl, user, password);
   Statement s = c.createStatement() ) {
   c.setAutoCommit(false);
   int records = s.executeUpdate("INSERT INTO product "     //Insert new product
       + "VALUES ('CP-00002', 'Lenovo', " + "790.0, 'br', 'Latop')");
   System.out.println("Successfully inserted "+ records + " records.");
   records = s.executeUpdate("UPDATE product "               //Update product price
                  + "SET price=470, description='Classic laptop' "
                  + "WHERE code='CP-00001'");
   System.out.println("Successfully updated "+ records + " records.");
   c.commit();                                               //Finish transaction
}
```

Трислойна архитектура: презентация, бизнес логика и данни: Model -View-Controller-MVC design pattern, Model 2

Servlet (Controller) + JSPs (Views) + POJOs (Model)

Предимства на MVC:

- Разделяне на труда между уеб дизайнери и програмисти на Java™
- Възможност за независима промяна на презентационната логика и визуалното представяне на данните
- По-лесна поддръжка, модификация и разширяване
- Улеснена навигация



Литература и интернет ресурси

- JSR 342: JavaTM Platform, Enterprise Edition 7 (Java EE 7) Specification – https://www.jcp.org/en/jsr/detail?id=342
- Java EE 7 Tutorial – https://docs.oracle.com/javaee/7/tutorial/
- GlassFish Application Server – http://glassfish.java.net/
- Introducing the Java EE 6 Platform – http://java.sun.com/developer/technicalArticles/JavaEE/JavaEE6Overview.html
- Java Platform, Enterprise Edition във Wikipedia – http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition
- Java EE 5 Tutorial – http://java.sun.com/javaee/5/docs/tutorial/doc/

Литература и интернет ресурси

- JSR 244: JavaTM Platform, Enterprise Edition 5 (Java EE 5) Specification – http://jcp.org/en/jsr/detail?id=244
- Java EE 5 Tutorial – http://java.sun.com/javaee/5/docs/tutorial/doc/
- Hall, M., Brown, L., Core Servlets and JavaServer Pages – http://pdf.coreservlets.com/
- Страница за JavaServer Pages на уеб сайта на Oracle® – http://www.oracle.com/technetwork/java/javaee/jsp/index.html
- JavaServer Pages в Wikipedia – http://en.wikipedia.org/wiki/JavaServer_Pages
- Ръководство за JavaServer Pages– http://www.jsptut.com/
- JavaWorld: Understanding JavaServer Pages Model 2 architecture – http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

Литература и интернет ресурси

- JavaServer Pages (JSP) Syntax Reference – http://java.sun.com/products/jsp/syntax/2.0/syntaxref20.html
- JSP Simple Tags Explained by Andy Grant http://articles.sitepoint.com/article/jsp-2-simple-tags
- JavaServer Pages Standard Tag Library 1.1 Tag Reference – http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html
- Java EE 5 Tutorial – http://java.sun.com/javaee/5/docs/tutorial/doc/

Thank's for Your Attention!

Trayan Iliev

CEO of IPT – Intellectual Products & Technologies

http://iproduct.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD

https://plus.google.com/+IproductOrg