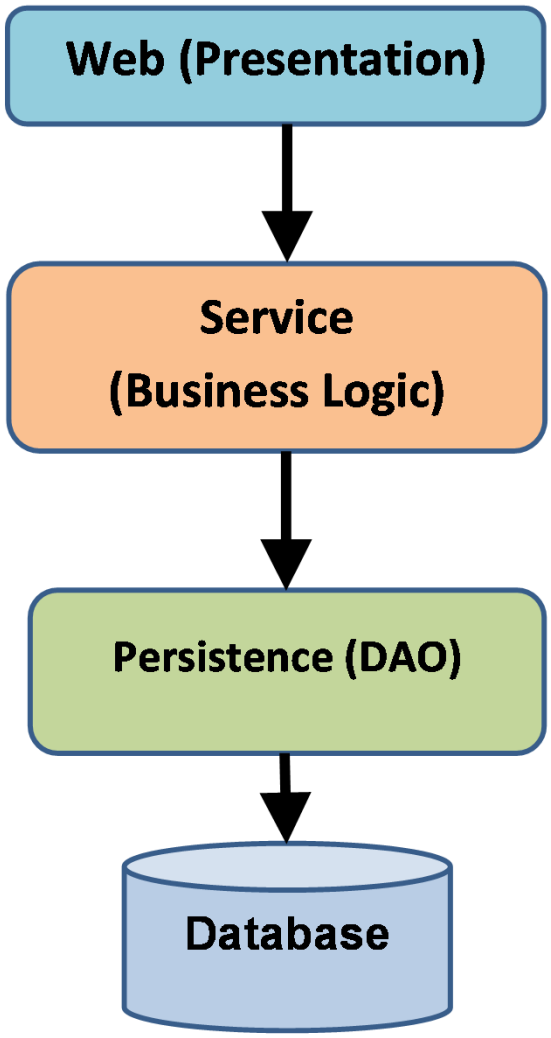
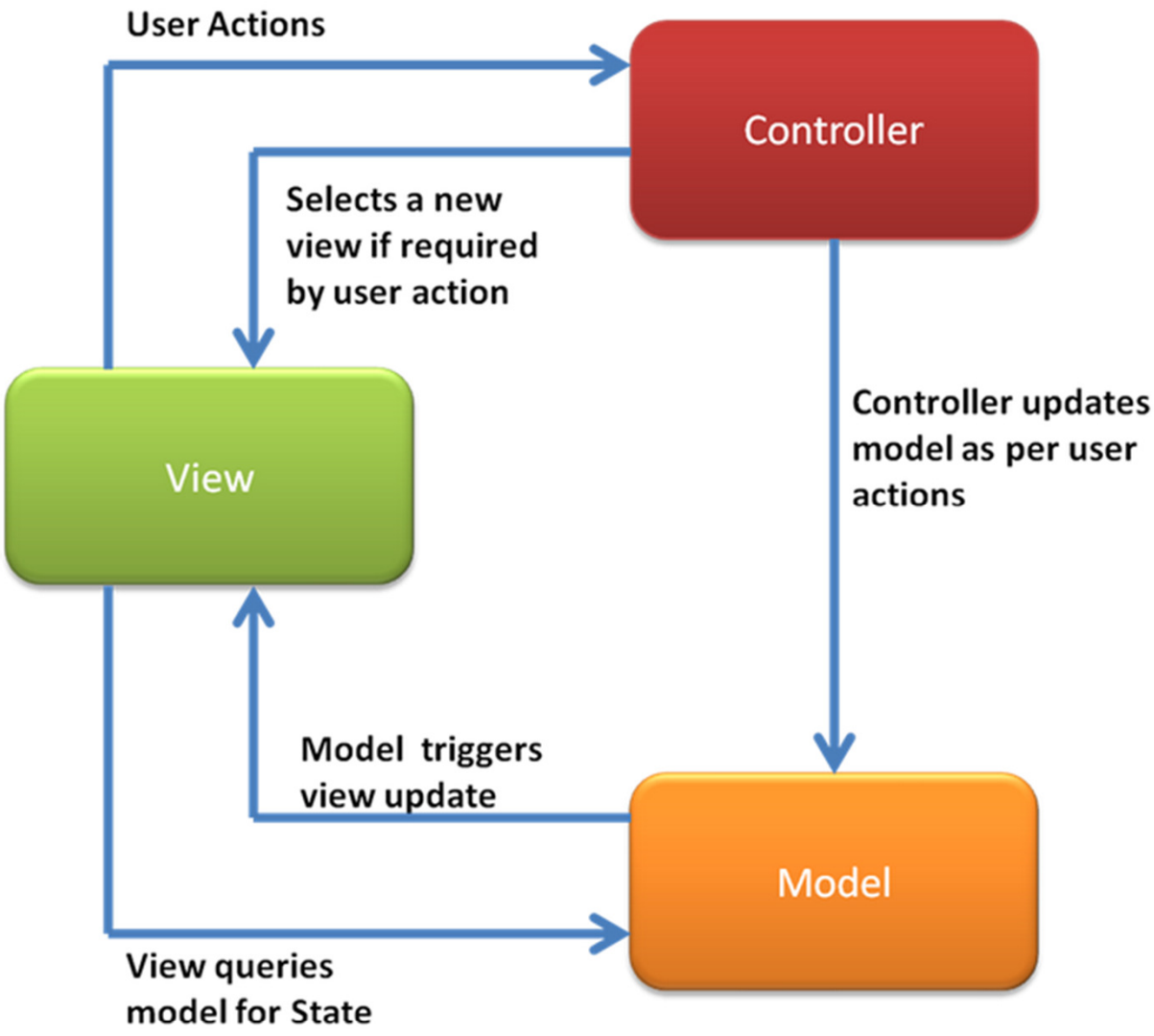
The Web Tier

- **Handle interactions with the client browser**
- **Responsible for presenting results of business logic operations**

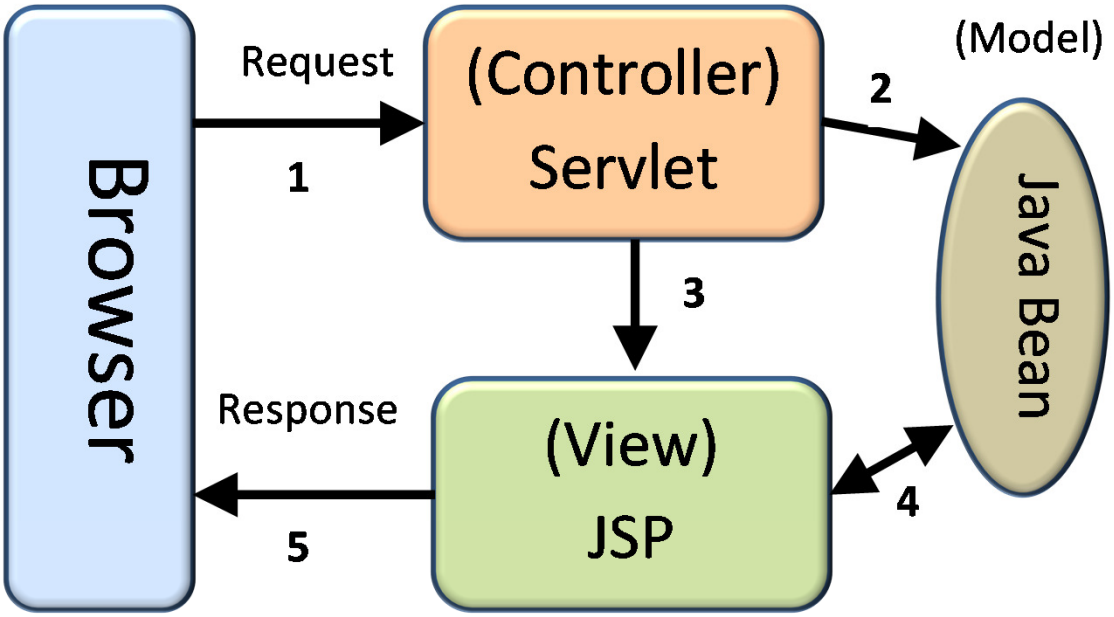


Traditional MVC Design Pattern



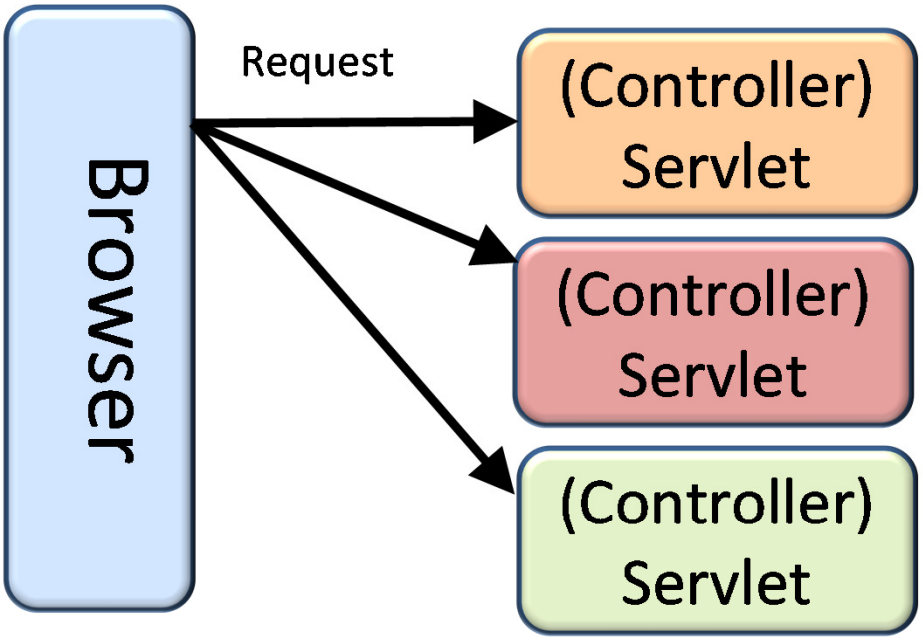
1. The **Controller** handles user events and calls on business logic to perform the user's request
2. The **Model** contains the data and operations necessary to perform the business logic
3. The **View** presents data from the Model

Traditional Web MVC



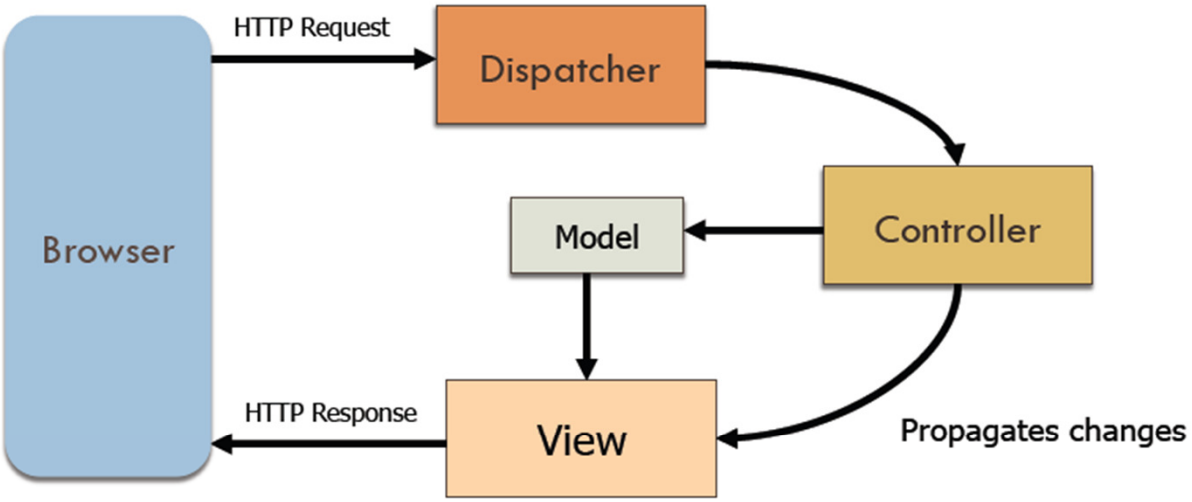
1. Browser request goes directly to the appropriate Controller (Servlet)
2. Controller calls on Java Beans to perform business logic
3. Controller passes result to the View (JSP)
4. View presents the result
5. Browser displays updated view

Traditional Web MVC



1. Browser directly contacts each controller
2. Each controller sends back a result directly to the browser

J2EE Front Controller Design Pattern

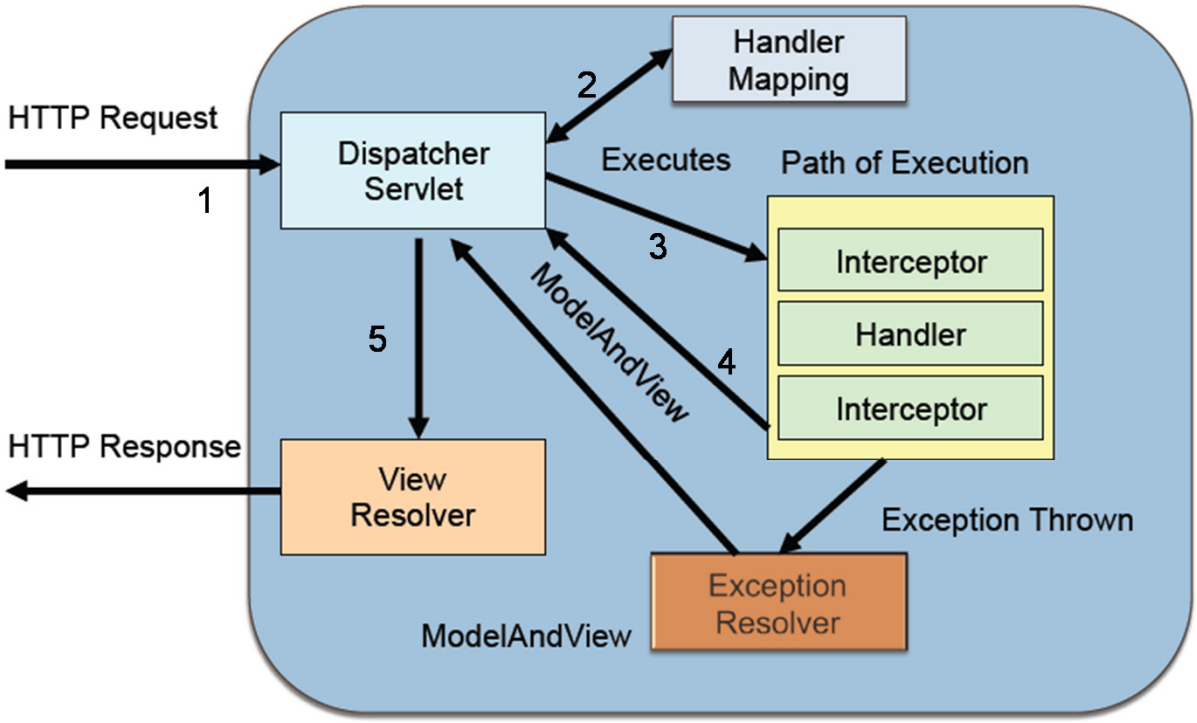


- All requests from the user are sent through one **Dispatcher** (a servlet) which will select the appropriate **Controller** (also a servlet)
- There may be many Controllers but there will only be one Dispatcher
- The Controller will return its View to the Dispatcher which will return it to the browser

Benefits of a Front Controller

- Centralizes request processing and view selection
- Single point of access into your system
- Allows us to apply common logic to multiple requests (such as security checks)
- Promotes code reuse across requests
- Easier to test

Spring MVC LifeCycle



1. Request is received by the **DispatcherServlet**
2. DispatcherServlet looks up the Controller that is mapped to the requested URL (performed by the **HandlerMapping**)
3. DispatcherServlet forwards request to the **Controller**
4. Controller manages business logic execution, then creates a ModelAndView object that provides the result (Model) and the name of the View that will present the result
5. DispatcherServlet forwards the Model object to the ViewResolver which selects and calls the correct View

Configuring Spring MVC

- First configure the Web application to direct all requests to the DispatcherServlet
  - This is done in the web.xml file

web.xml

```
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

- The <servlet> tag specifies that the name of the DispatcherServlet is appServlet and provides the configuration file for the Spring container
- The <servlet-mapping> tag maps URL patterns to the DispatcherServlet
  - In our case, all URLs will be mapped to the DispatcherServlet

servlet-context.xml

- The name of this file is defined in the web.xml file (but is commonly called servlet-context.xml)
  - Is loaded on startup of the web server
  - Spring configuration file that defines the web context

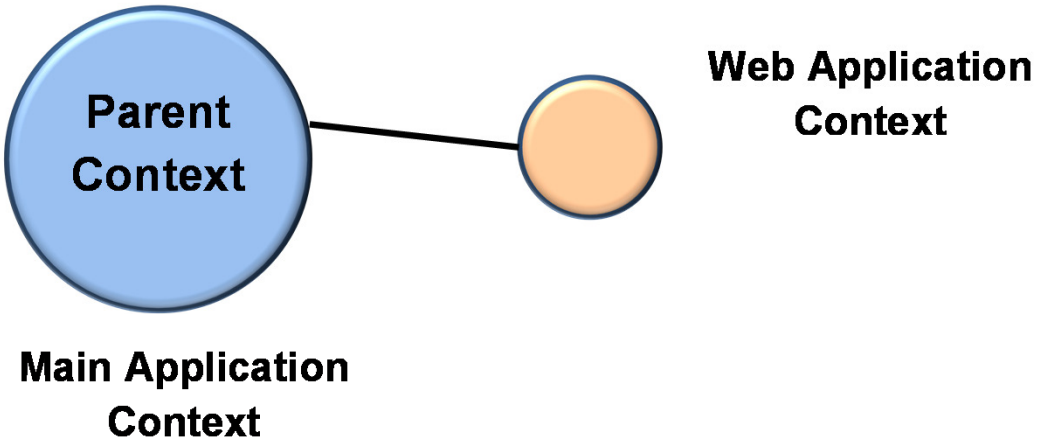
servlet-context.xml

```
<!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->

<!-- Enables the Spring MVC @Controller programming model -->
<annotation-driven />

<!-- Handles HTTP GET requests for /resources/** by efficiently serving up static
     resources in the ${webappRoot}/resources directory -->
<resources mapping="/resources/**" location="/resources/" />

<!-- Resolves views selected for rendering by @Controllers to .jsp resources in
    the /WEB-INF/views directory -->
<beans:bean
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <beans:property name="prefix" value="/WEB-INF/views/" />
  <beans:property name="suffix" value=".jsp" />
</beans:bean>

<context:component-scan base-package="org.npu.mvc" />
```

- Note the <resources> tag which specifies that static pages in the resources folder will not be mapped to DispatcherServlet
  - Put images, .css files, etc. in this folder
- Note the declaration of a ViewResolver bean that tells the DispatcherServlet where to find the views (.jsp in this case)

servlet-context.xml

- It is common to have servlet-context.xml serve only as the web context
- The Middle Tier (Service, DAO) can have its own context to get a clean separation
  - servlet-context.xml becomes a child of your main application context
  - It can resolve beans from the parent context, but other contexts can't resolve beans from it (if a bean is not found in the child context, the parent context is searched)



Separate Middle Tier Context

- To separate the web context from other contexts
  - Add the following to the web.xml file

web.xml

```
<!-- The definition of the Root Spring Container shared by all Servlets and Filters -->
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/spring/root-context.xml</param-value>
</context-param>

<!-- Creates the Spring Container shared by all Servlets and Filters -->
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

- The parent context will be found in root-context.xml
- The <listener> tag is required to bootstrap the Spring container application context and place beans in the root application context

A Simple Servlet (Controller)

- You can now write your Servlet as a POJO
  - Add the @Controller annotation
  - Add a @RequestMapping annotation
    - Tells the DispatcherServlet to map a URL to a specific method – the Handler

The following Controller is provided in the Spring MVC Template project from STS:

```
@Controller
public class HomeController {

  private static final Logger logger =
            LoggerFactory.getLogger(HomeController.class);

  // Simply selects the home view to render by
  // returning its name.
  @RequestMapping(value = "/home", method = RequestMethod.GET)
  public String home(Locale locale, Model model) {
    /* do something */

    return "home";
  }

}
```

A Simple Servlet (Controller)

The following Controller is provided in the Spring MVC Template project from STS

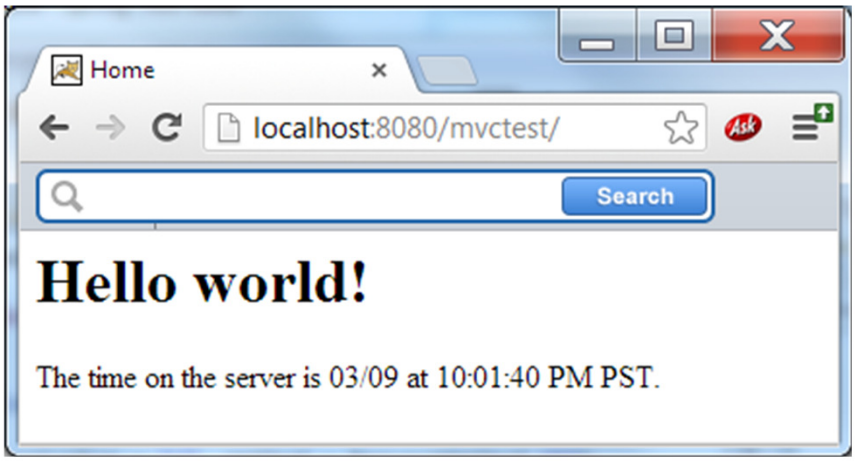
- The data to be displayed by the View is put in the **Model** object
- The name of the View is returned

```
@Controller
public class HomeController {

  private static final Logger logger =
            LoggerFactory.getLogger(HomeController.class);

  // Simply selects the home view to render by
  // returning its name.
  @RequestMapping(value = "/", method = RequestMethod.GET)
  public String home(Locale locale, Model model) {
    logger.info("Welcome home! The client locale is {}.",
          locale);

    Date date = new Date();
    DateFormat dateFormat =
          DateFormat.getDateTimeInstance(DateFormat.LONG,
          DateFormat.LONG, locale);

    String formattedDate = dateFormat.format(date);
    model.addAttribute("serverTime", formattedDate );

    return "home";
}
```

Calling a Servlet

- The View is rendered by home.jsp

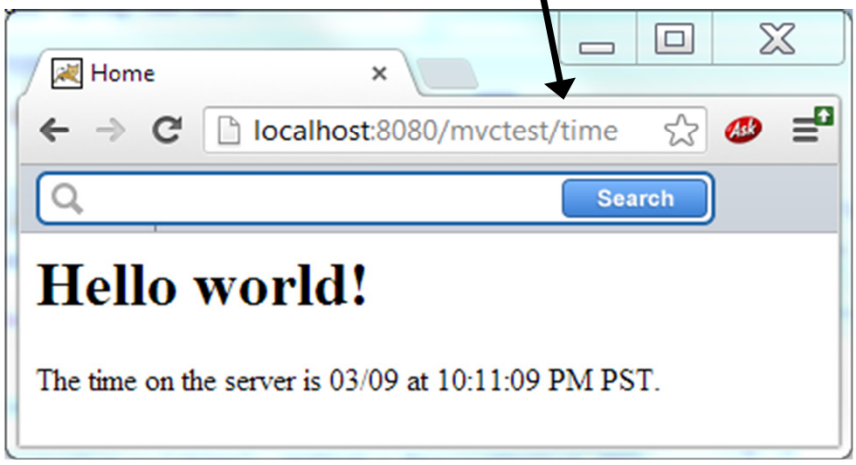
```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<html>
<head>
  <title>Home</title>
</head>
<body>
<h1>
  Hello world!
</h1>

<P>  The time on the server is ${serverTime}. </P>
</body>
</html>
```



A Simple Servlet (Controller)

Changing the path specified in @RequestMapping changes the URL of the servlet

```
@Controller
public class HomeController {
  private static final Logger logger =
            LoggerFactory.getLogger(HomeController.class);

  // Simply selects the home view to render by
  // returning its name.
  @RequestMapping(value = "/time", method = RequestMethod.GET)
  public String home(Locale locale, Model model) {

    ...
    return "home";
  }
}
```



@RequestMapping

- It's not necessary to specify the HTTP Command (GET or POST), but it is better to do so

```
@RequestMapping(value = "/time")
public String home(Locale locale, Model model) {
```

- The mapping path can be extended, as desired

```
@RequestMapping(value = "/firstex/time", method = RequestMethod.GET)
public String home(Locale locale, Model model) {
```

- Adding .* allows any suffix to be mapped to the handler method
  - Can be used to hide the fact that the page is rendered by a servlet (e.g., time.html)

```
@RequestMapping(value = "/time.*", method = RequestMethod.GET)
public String home(Locale locale, Model model) {
```

@RequestMapping on a Class

- May be placed on a class
  - Allows mapping of all requests within a path to a Controller

```
@Controller
@RequestMapping("/time/*")
public class HomeController {

  @RequestMapping(value="/home", method = RequestMethod.GET)
  public String home(Locale locale, Model model) {
  }

  @RequestMapping(value="/info", method = RequestMethod.GET)
  public String dateInfo(Locale locale, Model model) {
  }
```

Would map to /time/home and /time/info

@RequestMapping on a Class

- May be placed on a class
  - Use RequestMethod type to map to a method

```
@Controller
@RequestMapping("/time")
public class HomeController {

  @RequestMapping(method = RequestMethod.GET)
  public String home(Locale locale, Model model) {
  }

  @RequestMapping(method = RequestMethod.POST)
  public String dateInfo(Locale locale, Model model) {
  }
```

- home() would handle a GET command with URL /time
- dateInfo() would handle a POST command with URL /time