The ContentNegotiatingViewResolver does not resolve views itself, but delegates to other [ViewResolver](http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/ViewResolver.html" \o "interface in org.springframework.web.servlet)s. By default, these other view resolvers are picked up automatically from the application context, though they can also be set explicitly by using the [viewResolvers](http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/view/ContentNegotiatingViewResolver.html" \l "setViewResolvers-java.util.List-) property. **Note** that in order for this view resolver to work properly, the [order](http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/view/ContentNegotiatingViewResolver.html#setOrder-int-) property needs to be set to a higher precedence than the others (the default is [Ordered.HIGHEST\_PRECEDENCE](http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/core/Ordered.html" \l "HIGHEST_PRECEDENCE)).

This view resolver uses the requested [media type](http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/MediaType.html) to select a suitable [View](http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/View.html) for a request. The requested media type is determined through the configured [ContentNegotiationManager](http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/accept/ContentNegotiationManager.html" \o "class in org.springframework.web.accept). Once the requested media type has been determined, this resolver queries each delegate view resolver for a [View](http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/View.html) and determines if the requested media type is [compatible](http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/MediaType.html#includes-org.springframework.http.MediaType-) with the view's [content type](http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/View.html#getContentType--)). The most compatible view is returned.

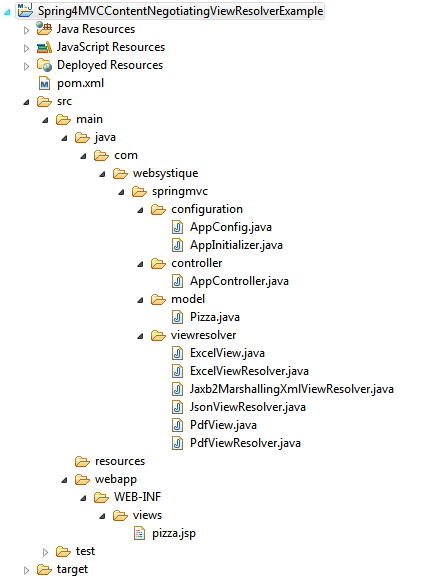
Additionally, this view resolver exposes the [defaultViews](http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/view/ContentNegotiatingViewResolver.html" \l "setDefaultViews-java.util.List-) property, allowing you to override the views provided by the view resolvers. Note that these default views are offered as candidates, and still need have the content type requested (via file extension, parameter, or Accept header, described above).

For example, if the request path is /view.html, this view resolver will look for a view that has the text/html content type (based on the html file extension). A request for /view with a text/html request Accept header has the same result.

**Since:** 3.0

**Spring 4 MVC ContentNegotiatingViewResolver example**

ContentNegotiatingViewResolver is an implementation of ViewResolver, which uses the requested media type (based on filetype extension, URL parameter specifying type of output format or accept header) to select a suitable View for a request. ContentNegotiatingViewResolver does not resolve view by itself but delegates to other ViewResolver you can configure to handle specific views(XML,JSON,PDF,XLS,HTML,..).



**Step 2: Update pom.xml with required dependencies**

|  |
| --- |
| <?xml version="1.0"?>  <project      xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> <http://maven.apache.org/xsd/maven-4.0.0.xsd>"      xmlns="<http://maven.apache.org/POM/4.0.0>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>">        <modelVersion>4.0.0</modelVersion>      <groupId>com.websystique.springmvc</groupId>      <artifactId>Spring4MVCContentNegotiatingViewResolverExample</artifactId>      <packaging>war</packaging>      <version>1.0.0</version>      <name>Spring4MVCContentNegotiatingViewResolverExample</name>        <properties>          <springframework.version>4.0.6.RELEASE</springframework.version>      </properties>        <dependencies>          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-core</artifactId>              <version>${springframework.version}</version>          </dependency>          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-web</artifactId>              <version>${springframework.version}</version>          </dependency>          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-webmvc</artifactId>              <version>${springframework.version}</version>          </dependency>            <!-- Needed for XML View (with JAXB2) -->          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-oxm</artifactId>              <version>${springframework.version}</version>          </dependency>            <!-- Needed for JSON View -->          <dependency>              <groupId>com.fasterxml.jackson.core</groupId>              <artifactId>jackson-databind</artifactId>              <version>2.4.1.3</version>          </dependency>          <dependency>              <groupId>com.fasterxml.jackson.core</groupId>              <artifactId>jackson-annotations</artifactId>              <version>2.4.1</version>          </dependency>            <!-- Needed for PDF View -->          <dependency>              <groupId>com.lowagie</groupId>              <artifactId>itext</artifactId>              <version>4.2.1</version>          </dependency>            <!-- Needed for XLS View -->          <dependency>              <groupId>org.apache.poi</groupId>              <artifactId>poi</artifactId>              <version>3.10-beta2</version>          </dependency>            <!-- Servlet dependencies -->          <dependency>              <groupId>javax.servlet</groupId>              <artifactId>javax.servlet-api</artifactId>              <version>3.1.0</version>          </dependency>          <dependency>              <groupId>javax.servlet</groupId>              <artifactId>jstl</artifactId>              <version>1.2</version>          </dependency>          <dependency>              <groupId>javax.servlet.jsp</groupId>              <artifactId>javax.servlet.jsp-api</artifactId>              <version>2.3.1</version>          </dependency>        </dependencies>          <build>          <pluginManagement>              <plugins>                  <plugin>                      <groupId>org.apache.maven.plugins</groupId>                      <artifactId>maven-compiler-plugin</artifactId>                      <version>3.2</version>                      <configuration>                          <source>1.6</source>                          <target>1.6</target>                      </configuration>                  </plugin>                  <plugin>                      <groupId>org.apache.maven.plugins</groupId>                      <artifactId>maven-war-plugin</artifactId>                      <version>2.4</version>                      <configuration>                          <warSourceDirectory>src/main/webapp</warSourceDirectory>                          <warName>Spring4MVCContentNegotiatingViewResolverExample</warName>                          <failOnMissingWebXml>false</failOnMissingWebXml>                      </configuration>                  </plugin>              </plugins>          </pluginManagement>            <finalName>Spring4MVCContentNegotiatingViewResolverExample</finalName>      </build>  </project> |

spring-oxm is included to support XML output generation (using JAXB2). jackson-databind & jackson-annotationsprovide JSON output support. itext provide PDF generation library to support PDF output. Apache POI will help producing XLS output format.

**Step 3: Create Spring Configuration Class**

com.websystique.springmvc.configuration.AppConfig

|  |
| --- |
| package com.websystique.springmvc.configuration;    import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.ComponentScan;  import org.springframework.context.annotation.Configuration;  import org.springframework.http.MediaType;  import org.springframework.oxm.jaxb.Jaxb2Marshaller;  import org.springframework.web.accept.ContentNegotiationManager;  import org.springframework.web.servlet.ViewResolver;  import org.springframework.web.servlet.config.annotation.ContentNegotiationConfigurer;  import org.springframework.web.servlet.config.annotation.EnableWebMvc;  import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;  import org.springframework.web.servlet.view.ContentNegotiatingViewResolver;  import org.springframework.web.servlet.view.InternalResourceViewResolver;  import org.springframework.web.servlet.view.JstlView;    import com.websystique.springmvc.model.Pizza;  import com.websystique.springmvc.viewresolver.ExcelViewResolver;  import com.websystique.springmvc.viewresolver.JsonViewResolver;  import com.websystique.springmvc.viewresolver.Jaxb2MarshallingXmlViewResolver;  import com.websystique.springmvc.viewresolver.PdfViewResolver;    @Configuration  @EnableWebMvc  @ComponentScan(basePackages = "com.websystique.springmvc")  public class AppConfig extends WebMvcConfigurerAdapter {        /\*       \* Configure ContentNegotiationManager       \*/      @Override      public void configureContentNegotiation(ContentNegotiationConfigurer configurer) {          configurer.ignoreAcceptHeader(true).defaultContentType(                  MediaType.TEXT\_HTML);      }        /\*       \* Configure ContentNegotiatingViewResolver       \*/      @Bean      public ViewResolver contentNegotiatingViewResolver(ContentNegotiationManager manager) {          ContentNegotiatingViewResolver resolver = new ContentNegotiatingViewResolver();          resolver.setContentNegotiationManager(manager);            // Define all possible view resolvers          List<ViewResolver> resolvers = new ArrayList<ViewResolver>();            resolvers.add(jaxb2MarshallingXmlViewResolver());          resolvers.add(jsonViewResolver());          resolvers.add(jspViewResolver());          resolvers.add(pdfViewResolver());          resolvers.add(excelViewResolver());            resolver.setViewResolvers(resolvers);          return resolver;      }        /\*       \* Configure View resolver to provide XML output Uses JAXB2 marshaller to       \* marshall/unmarshall POJO's (with JAXB annotations) to XML       \*/      @Bean      public ViewResolver jaxb2MarshallingXmlViewResolver() {          Jaxb2Marshaller marshaller = new Jaxb2Marshaller();          marshaller.setClassesToBeBound(Pizza.class);          return new Jaxb2MarshallingXmlViewResolver(marshaller);      }        /\*       \* Configure View resolver to provide JSON output using JACKSON library to       \* convert object in JSON format.       \*/      @Bean      public ViewResolver jsonViewResolver() {          return new JsonViewResolver();      }        /\*       \* Configure View resolver to provide PDF output using lowagie pdf library to       \* generate PDF output for an object content       \*/      @Bean      public ViewResolver pdfViewResolver() {          return new PdfViewResolver();      }        /\*       \* Configure View resolver to provide XLS output using Apache POI library to       \* generate XLS output for an object content       \*/      @Bean      public ViewResolver excelViewResolver() {          return new ExcelViewResolver();      }        /\* Configure View resolver to provide HTML output This is the       \* default format       \* in absence of any type suffix.       \*/      @Bean      public ViewResolver jspViewResolver() {          InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();          viewResolver.setViewClass(JstlView.class);          viewResolver.setPrefix("/WEB-INF/views/");          viewResolver.setSuffix(".jsp");          return viewResolver;      }    } |

First step is to create the ContentNegotiationManager which is used to determine the requested media types of a request by delegating to a list of ContentNegotiationStrategy instances. By default PathExtensionContentNegotiationStrategy is consulted (which uses the URL extension e.g. .xls, .pdf,.json..) , followed by ParameterContentNegotiationStrategy(which uses the request parameter ‘format=xls’ e.g.), followed by HeaderContentNegotiationStrategy (which uses HTTP Accept Headers).

|  |
| --- |
| public void configureContentNegotiation(ContentNegotiationConfigurer configurer) {      configurer.ignoreAcceptHeader(true).defaultContentType(              MediaType.TEXT\_HTML);  } |

In our example, we will be using the URL extension to help determine the media types. Also, we have set the default media type to TEXT\_HTML in absence of file extension or when the filetype is unknown, that means JSP view resolver will be used when no [known] URL extension found.

Below is the content of **pizza.jsp** used by default JSP view resolver

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=ISO-8859-1"  pageEncoding="ISO-8859-1"%>  <%@ taglib prefix="c" uri="<http://java.sun.com/jsp/jstl/core>" %>    <html>  <head>      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">      <title>Pizza JSP View</title>  </head>  <body>      <table border="1">          <tr>          <td>NAME</td>          <td>Flavor</td>          <td>Toppings</td>          </tr>          <tr>              <td>${pizza.name}</td>              <td>${pizza.flavor}</td>              <td>                  <c:forEach var="item" items="${pizza.toppings}">                      <c:out value="${item}"/>&nbsp;                  </c:forEach>              </td>          </tr>      </table>  </body>  </html> |

Next step is to configure ContentNegotaionViewResolver itself,

|  |
| --- |
| public ViewResolver contentNegotiatingViewResolver(ContentNegotiationManager manager) {      ContentNegotiatingViewResolver resolver = new ContentNegotiatingViewResolver();      resolver.setContentNegotiationManager(manager);        // Define all possible view resolvers      List<ViewResolver> resolvers = new ArrayList<ViewResolver>();        resolvers.add(jaxb2MarshallingXmlViewResolver());      resolvers.add(jsonViewResolver());      resolvers.add(jspViewResolver());      resolvers.add(pdfViewResolver());      resolvers.add(excelViewResolver());        resolver.setViewResolvers(resolvers);      return resolver;  } |

We need to set the ContentNegotiationManager which will be injected by Spring, and different resolvers for each possible output format our application might produce.

Finally, we have created different view resolvers for XML, JSON, PDF, XLS and HTML output which we will discuss next.

**Step 4: Create Different View Resolvers**

Let’s now create tha actual view resolvers itself.

**XML View Resolver:**

This view resolver relies on JAXB2 Marshalling/unmarshalling to produce XML output. The domain class needs to be annotated with JAXB2 annotations.

com.websystique.springmvc.viewresolver.Jaxb2MarshallingXmlViewResolver

|  |
| --- |
| package com.websystique.springmvc.viewresolver;    import java.util.Locale;    import org.springframework.oxm.Marshaller;  import org.springframework.web.servlet.View;  import org.springframework.web.servlet.ViewResolver;  import org.springframework.web.servlet.view.xml.MarshallingView;    public class Jaxb2MarshallingXmlViewResolver implements ViewResolver {        private Marshaller marshaller;          public Jaxb2MarshallingXmlViewResolver(Marshaller marshaller) {          this.marshaller = marshaller;      }          @Override      public View resolveViewName(String viewName, Locale locale) throws Exception {          MarshallingView view = new MarshallingView();          view.setMarshaller(marshaller);          return view;      }    } |

Below is the domain object (annotated with standard XML annotations) for our example:

com.websystique.springmvc.model.Pizza

|  |
| --- |
| package com.websystique.springmvc.model;    import java.util.ArrayList;  import java.util.List;    import javax.xml.bind.annotation.XmlElement;  import javax.xml.bind.annotation.XmlRootElement;    @XmlRootElement(name = "pizza")  public class Pizza {        private String name;        private String flavor;        private List<String> toppings = new ArrayList<String>();        public Pizza(){        }        public Pizza(String name){          this.name = name;          this.flavor = "spicy";          this.toppings.add("Cheese");          this.toppings.add("bakon");      }        @XmlElement      public void setName(String name) {          this.name = name;      }        public String getName() {          return name;      }        @XmlElement      public void setFlavor(String flavor) {          this.flavor = flavor;      }        public String getFlavor() {          return flavor;      }        public List<String> getToppings() {          return toppings;      }        @XmlElement      public void setToppings(List<String> toppings) {          this.toppings = toppings;      }    } |

**JSON View Resolver:**

This view resolver is using Spring MappingJackson2JsonView to get the view used to convert POJO to JSON.

com.websystique.springmvc.viewresolver.JsonViewResolver

|  |
| --- |
| package com.websystique.springmvc.viewresolver;    import java.util.Locale;    import org.springframework.web.servlet.View;  import org.springframework.web.servlet.ViewResolver;  import org.springframework.web.servlet.view.json.MappingJackson2JsonView;    public class JsonViewResolver implements ViewResolver{        @Override      public View resolveViewName(String viewName, Locale locale) throws Exception {          MappingJackson2JsonView view = new MappingJackson2JsonView();          view.setPrettyPrint(true);          return view;        }    } |

**PDF View Resolver:**

This view resolver is using lowagie itext library to actually generate PDF output.Also note that actual view extends from Spring AbstractPdfView which itself internally uses lowagie itext library.

com.websystique.springmvc.viewresolver.PdfView

|  |
| --- |
| package com.websystique.springmvc.viewresolver;    import java.awt.Color;  import java.util.Map;    import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;    import org.springframework.web.servlet.view.document.AbstractPdfView;    import com.lowagie.text.Document;  import com.lowagie.text.Element;  import com.lowagie.text.pdf.PdfPTable;  import com.lowagie.text.pdf.PdfWriter;  import com.websystique.springmvc.model.Pizza;    public class PdfView extends AbstractPdfView {        @Override      protected void buildPdfDocument(Map<String, Object> model,              Document document, PdfWriter writer, HttpServletRequest request,              HttpServletResponse response) throws Exception {            Pizza pizza = (Pizza) model.get("pizza");            PdfPTable table = new PdfPTable(3);          table.getDefaultCell().setHorizontalAlignment(Element.ALIGN\_CENTER);          table.getDefaultCell().setVerticalAlignment(Element.ALIGN\_MIDDLE);          table.getDefaultCell().setBackgroundColor(Color.lightGray);            table.addCell("Name");          table.addCell("Flavor");          table.addCell("Toppings");            table.addCell(pizza.getName());          table.addCell(pizza.getFlavor());            StringBuffer toppings = new StringBuffer("");          for (String topping : pizza.getToppings()) {              toppings.append(topping);              toppings.append(" ");          }          table.addCell(toppings.toString());          document.add(table);        }    } |

com.websystique.springmvc.viewresolver.PdfViewResolver

|  |
| --- |
| package com.websystique.springmvc.viewresolver;    import java.util.Locale;    import org.springframework.web.servlet.View;  import org.springframework.web.servlet.ViewResolver;    public class PdfViewResolver implements ViewResolver{        @Override      public View resolveViewName(String viewName, Locale locale) throws Exception {          PdfView view = new PdfView();          return view;        }    } |

**XLS View Resolver:**

This view resolver is using Apache POI library to actually generate Microsoft XLS output.Also note that actual view extends from Spring AbstractExcelView which itself internally uses Apache POI library.

com.websystique.springmvc.viewresolver.ExcelView

|  |
| --- |
| package com.websystique.springmvc.viewresolver;    import java.util.Map;    import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;    import org.apache.poi.hssf.usermodel.HSSFWorkbook;  import org.apache.poi.ss.usermodel.Cell;  import org.apache.poi.ss.usermodel.CellStyle;  import org.apache.poi.ss.usermodel.IndexedColors;  import org.apache.poi.ss.usermodel.Row;  import org.apache.poi.ss.usermodel.Sheet;  import org.springframework.web.servlet.view.document.AbstractExcelView;    import com.websystique.springmvc.model.Pizza;    public class ExcelView extends AbstractExcelView {        @Override      protected void buildExcelDocument(Map<String, Object> model,              HSSFWorkbook workbook, HttpServletRequest request,              HttpServletResponse response) throws Exception {            Pizza pizza = (Pizza) model.get("pizza");            Sheet sheet = workbook.createSheet("sheet 1");          CellStyle style = workbook.createCellStyle();          style.setFillForegroundColor(IndexedColors.GREY\_40\_PERCENT.index);          style.setFillPattern(CellStyle.SOLID\_FOREGROUND);          style.setAlignment(CellStyle.ALIGN\_CENTER);          Row row = null;          Cell cell = null;          int rowCount = 0;          int colCount = 0;            // Create header cells          row = sheet.createRow(rowCount++);            cell = row.createCell(colCount++);          cell.setCellStyle(style);          cell.setCellValue("Name");            cell = row.createCell(colCount++);          cell.setCellStyle(style);          cell.setCellValue("Flavor");            cell = row.createCell(colCount++);          cell.setCellStyle(style);          cell.setCellValue("Toppings");            // Create data cells          row = sheet.createRow(rowCount++);          colCount = 0;          row.createCell(colCount++).setCellValue(pizza.getName());          row.createCell(colCount++).setCellValue(pizza.getFlavor());            StringBuffer toppings = new StringBuffer("");          for (String topping : pizza.getToppings()) {              toppings.append(topping);              toppings.append(" ");          }          row.createCell(colCount++).setCellValue(toppings.toString());            for (int i = 0; i < 3; i++)              sheet.autoSizeColumn(i, true);        }    } |

com.websystique.springmvc.viewresolver.ExcelViewResolver

|  |
| --- |
| package com.websystique.springmvc.viewresolver;    import java.util.Locale;    import org.springframework.web.servlet.View;  import org.springframework.web.servlet.ViewResolver;    public class ExcelViewResolver implements ViewResolver{        @Override      public View resolveViewName(String viewName, Locale locale) throws Exception {          ExcelView view = new ExcelView();          return view;        }    } |

That is all needed for ContentNegotaingViewResolver configuration.

To complete the example and make it runnable, let’s add the missing Spring MVC configuration peaces.

**Step 5: Create Controller class**

Below is a trivial REST based controller for our example.

com.websystique.springmvc.controller.AppController

|  |
| --- |
| package com.websystique.springmvc.controller;    import org.springframework.stereotype.Controller;  import org.springframework.ui.ModelMap;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;    import com.websystique.springmvc.model.Pizza;    @Controller  public class AppController {        @RequestMapping(value="/pizzavalley/{pizzaName}", method = RequestMethod.GET)      public String getPizza(@PathVariable String pizzaName, ModelMap model) {            Pizza pizza = new Pizza(pizzaName);          model.addAttribute("pizza", pizza);            return "pizza";        }    } |

**Step 6: Create Initialization Class**

Add an initializer class implementing WebApplicationInitializer as shown below(which in this case acts as replacement of any spring configuration defined in web.xml). During Servlet 3.0 Container startup, this class will be loaded and instantiated and its onStartup method will be called by servlet container.

com.websystique.springmvc.configuration.AppInitializer

|  |
| --- |
| package com.websystique.springmvc.configuration;    import javax.servlet.ServletContext;  import javax.servlet.ServletException;  import javax.servlet.ServletRegistration;    import org.springframework.web.WebApplicationInitializer;  import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;  import org.springframework.web.servlet.DispatcherServlet;    public class AppInitializer implements WebApplicationInitializer {        public void onStartup(ServletContext container) throws ServletException {            AnnotationConfigWebApplicationContext ctx = new AnnotationConfigWebApplicationContext();          ctx.register(AppConfig.class);          ctx.setServletContext(container);            ServletRegistration.Dynamic servlet = container.addServlet(                  "dispatcher", new DispatcherServlet(ctx));            servlet.setLoadOnStartup(1);          servlet.addMapping("/");      }    } |

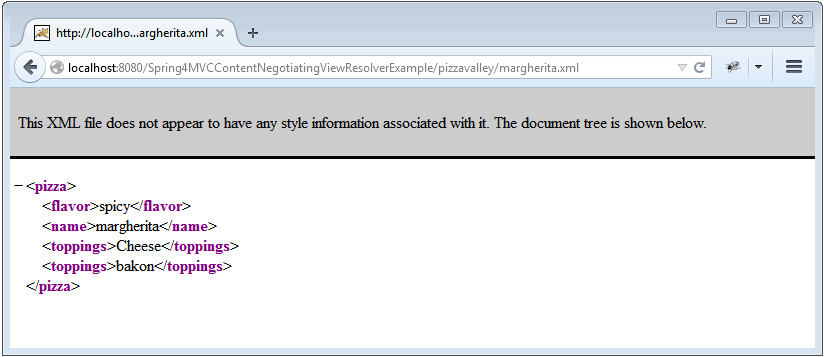
**UPDATE:** Note that above class can be written even more concisely [**and it’s the preferred way**], by extendingAbstractAnnotationConfigDispatcherServletInitializer base class, as shown below:

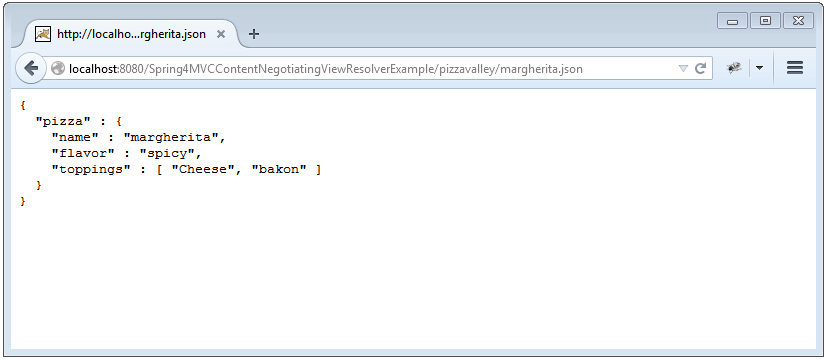
|  |
| --- |
| package com.websystique.springmvc.configuration;    import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;    public class AppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {        @Override      protected Class<?>[] getRootConfigClasses() {          return new Class[] { AppConfig.class };      }        @Override      protected Class<?>[] getServletConfigClasses() {          return null;      }        @Override      protected String[] getServletMappings() {          return new String[] { "/" };      }    } |

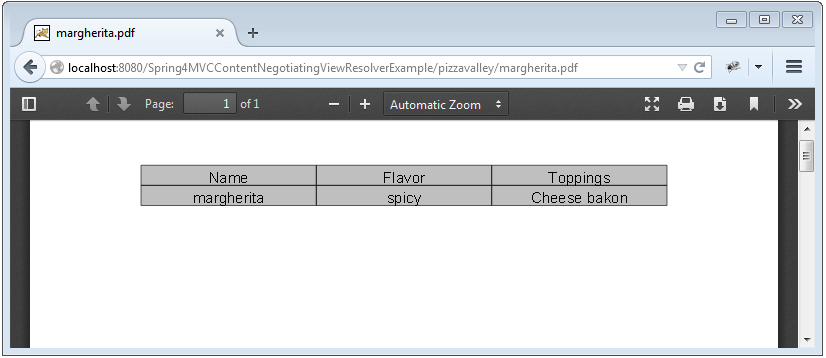
**Step 7: Build and Deploy the application**

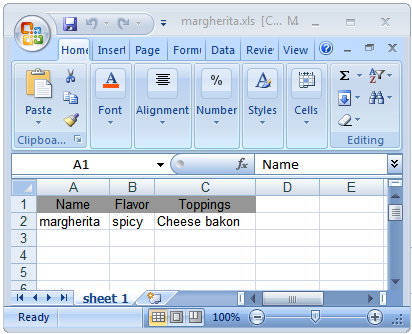
Now build the war (via eclipse or maven [ mvn clean install]). Deploy the war to a Servlet 3.0 container. Since here i am using Tomcat, i will simply put this war file into tomcat webapps folder and click on start.bat inside tomcat bin directory.

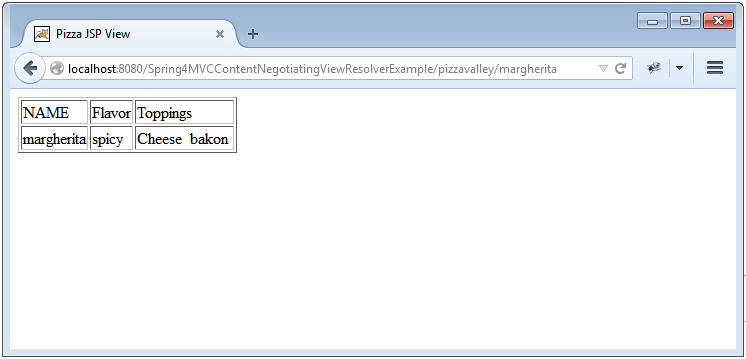
Run it.Below are the snapshot of sample run triggering deffernt outputs (notice URL extensions)











***Download Source Code***

[**Download Now!**](http://websystique.com/?smd_process_download=1&download_id=772)