# 2.Spring MVC 基礎

## ＭＶＣ與三層式架構

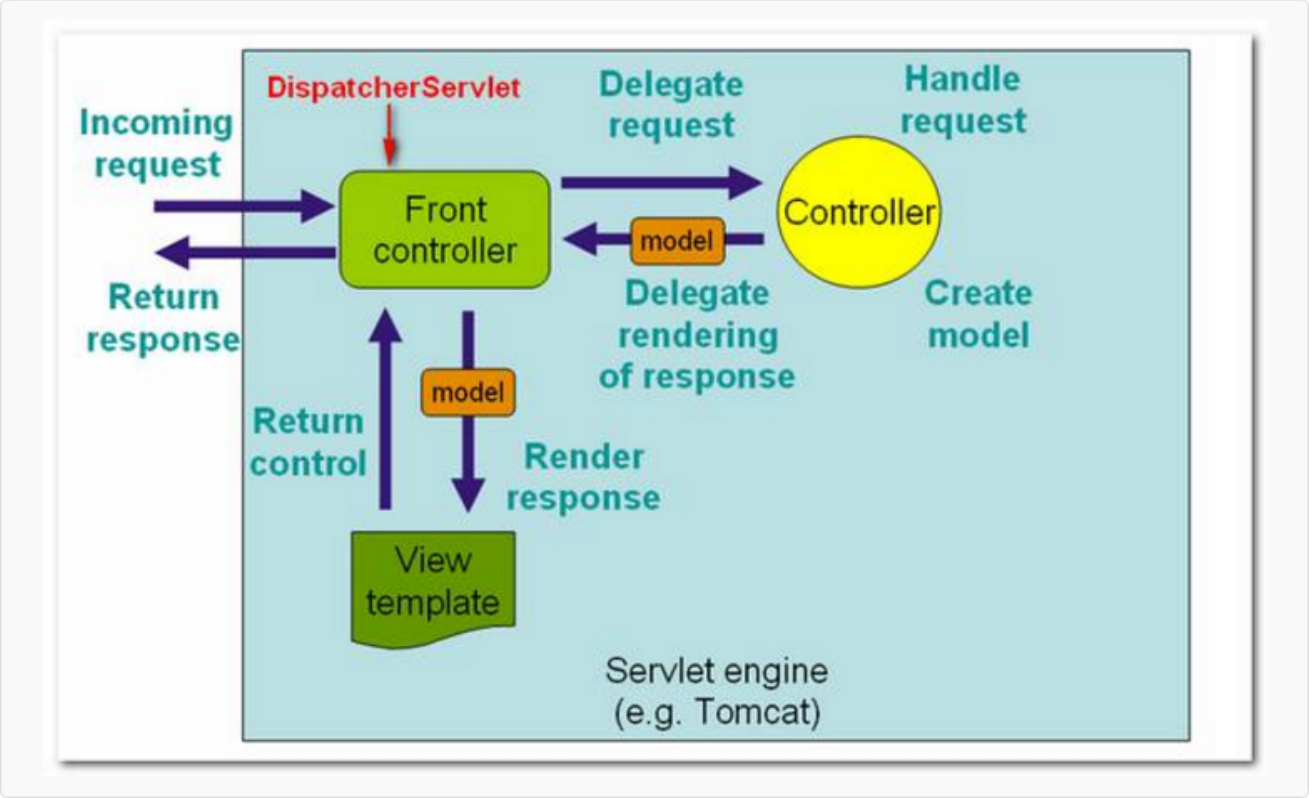
MVC：Model + View + Controller

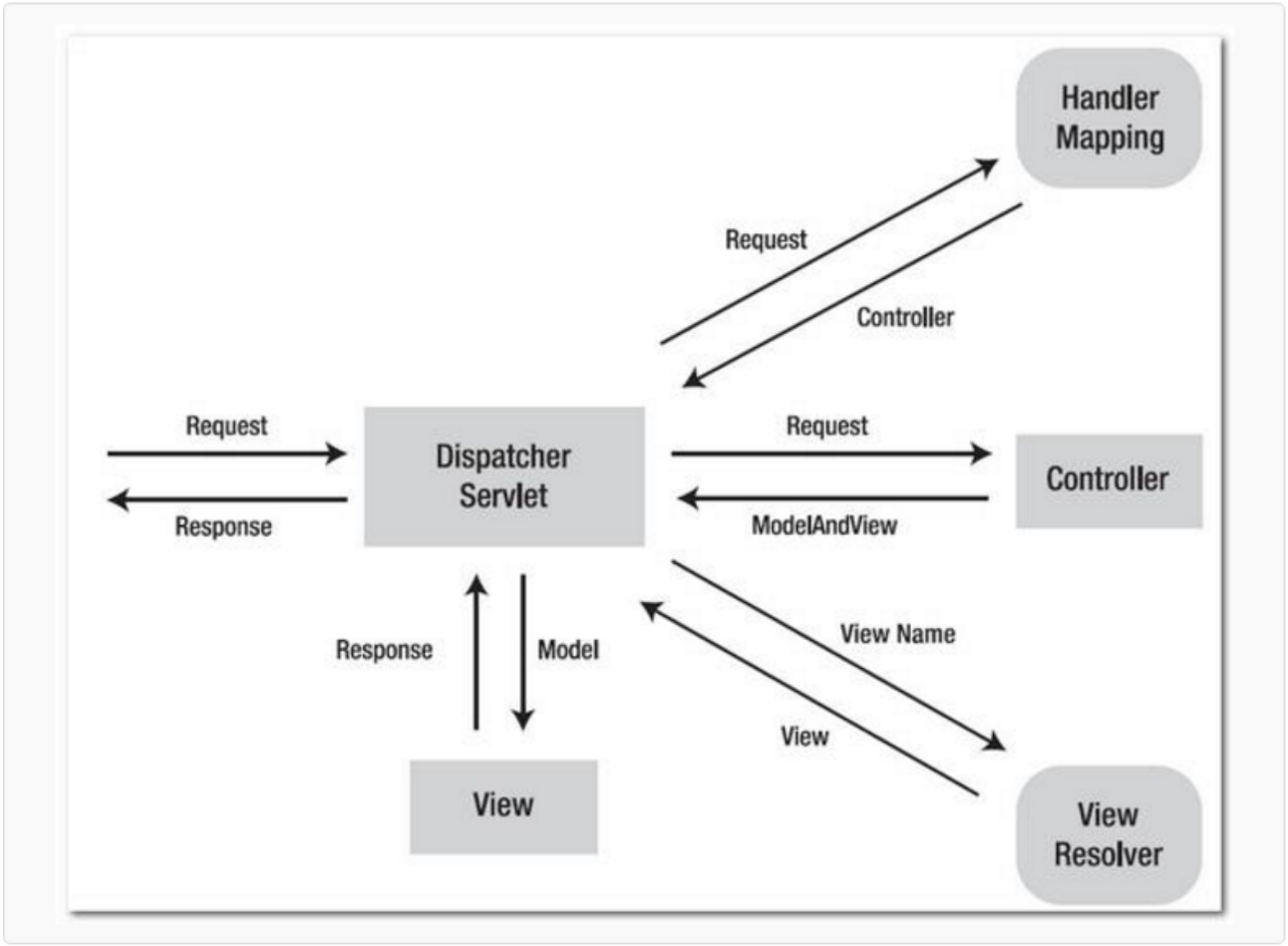
三層式架構：Presentation tier + Application tier + Data tier （展現層+應用層+資料存取層）

兩者有何關係? 很多人以為M是對應資料存取層，V是層現層，C是應用層；然而實際上MVC框架只是對應到三層式架構的展現層，M是數據模型，是Server 端在處理完請求後，將要返回給Client端的資料保存在此模型物件內，用來和View 之間做資料交換傳值用，在Spring MVC內有一個專門的class 叫Model ，實際上就是 Map 類型；V指的是Client 端要展現的視圖頁面，包含JSP、freeMarker、Velocity、Thymeleaf、JSON、PDF...等；C是控制器，負責回應處理請求的Handler，亦即Spring MVC中注解@Controller 的物件。

而三層式架構是整個Web應用系統的架構，一般專案中會建立Service類別做為應用層即業務邏輯處理，另外再建立DAO 類別負責資料存取層。

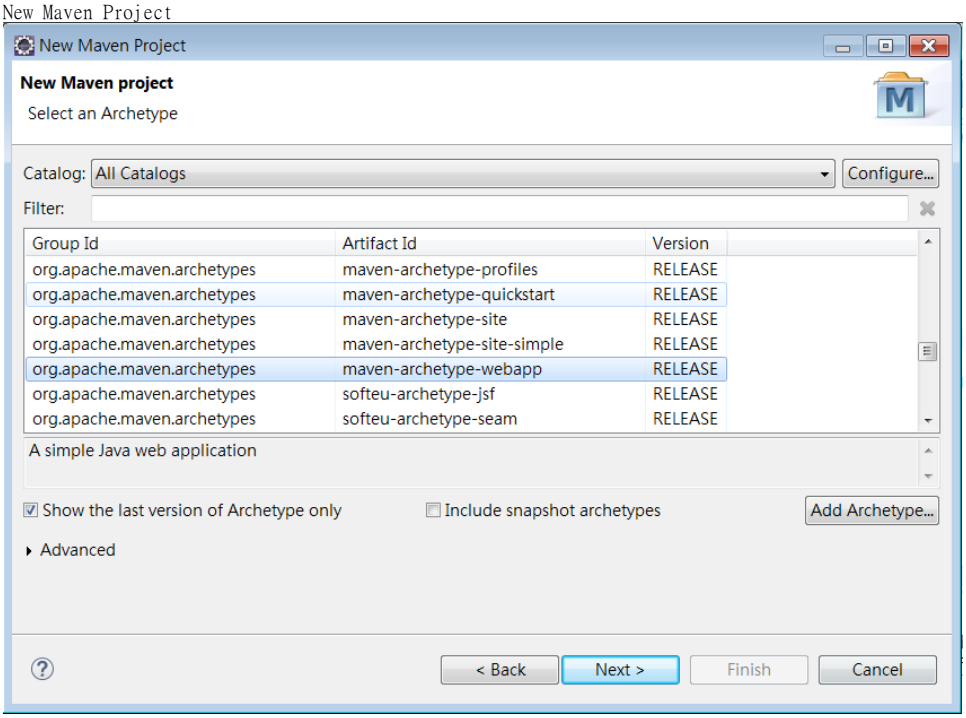
Spring MVC includes most of the same basic concepts as other so-called web MVC frameworks. Incoming requests enter the framework via a *Front Controller*. In the case of Spring MVC, this is an actual Java Servlet called `DispatcherServlet`. Think of `DispatcherServlet` as the gatekeeper. It doesn't perform any real web or business logic, but rather delegates to POJOs called *Controllers* where the real work is done (either in whole or via the back-end). When the work has been done, it's the responsibility of *Views* to produce the output in the proper format (whether that's a JSP page, Velocity template, or JSON response). *Strategies* are used to decide which Controller (and which method(s) inside that Controller) handles the request, and which View renders the response. The Spring container is used to wire together all these pieces. It all looks something like this:

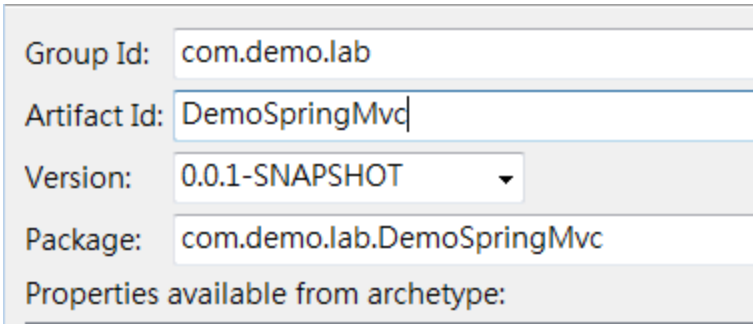





示例1

- New Maven Project






- 修改 pom.xml

**pom.xml**

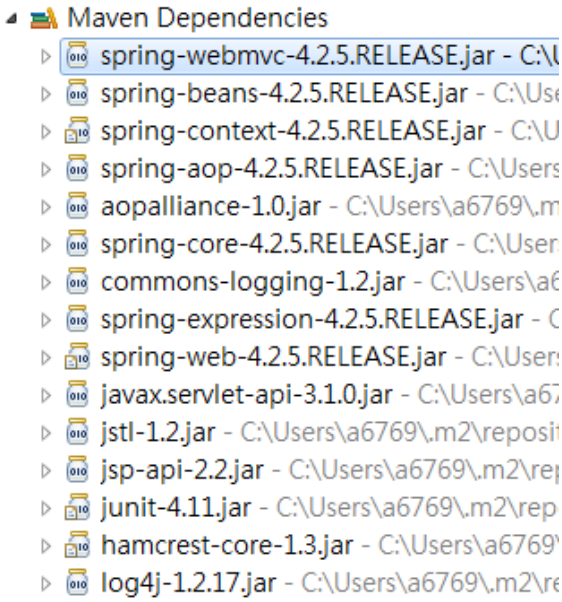
```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.demo.lab</groupId>
  <artifactId>DemoSpringMvc</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>DemoSpringMvc Maven Webapp</name>
  <url>http://maven.apache.org</url>

	<properties>
		<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
		<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
		<spring.version>4.3.5.RELEASE</spring.version>
		<java.version>1.8</java.version>
		<jsp.version>2.2</jsp.version>
		<jstl.version>1.2</jstl.version>
		<servlet.version>3.1.0</servlet.version>
	</properties>

	<dependencies>
		<!-- Spring framework -->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-webmvc</artifactId>
			<version>${spring.version}</version>
		</dependency>

		<!--  web app  api -->
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>javax.servlet-api</artifactId>
			<version>3.1.0</version>
			<scope>provided</scope>
		</dependency>
		<dependency>
			<groupId>jstlt</groupId>
			<artifactId>jstl</artifactId>
			<version>${jstl.version}</version>
		</dependency>
		<dependency>
			<groupId>javax.servlet.jsp</groupId>
			<artifactId>jsp-api</artifactId>
			<version>${jsp.version}</version>
			<scope>provided</scope>
		</dependency>

		<!-- junit & log4j -->
		<dependency>
			<groupId>junit</groupId>
			<artifactId>junit</artifactId>
			<version>4.11</version>
			<scope>test</scope>
```

```
                </dependency>
                <dependency>
                        <groupId>log4j</groupId>
                        <artifactId>log4j</artifactId>
                        <version>1.2.17</version>
                </dependency>
        </dependencies>

        <build>
                <finalName>DemoSpringMvc</finalName>
                <plugins>
                        <plugin>
                                <artifactId>maven-compiler-plugin</artifactId>
                                <version>3.1</version>
                                <configuration>
                                        <source>${java.version}</source>
                                        <target>${java.version}</target>
                                </configuration>
                        </plugin>

                        <!-- :  Jetty  -->
                        <plugin>
                                <groupId>org.eclipse.jetty</groupId>
                                <artifactId>jetty-maven-plugin</artifactId>
                                <version>9.2.11.v20150529</version>
                                <configuration>
                                        <webApp>
                                                <contextPath>/</contextPath>
                                        </webApp>
                                        <httpConnector>
                                                <port>18080</port>
                                        </httpConnector>
                                        <stopKey>STOP</stopKey>
                                        <stopPort>9999</stopPort>
                                        <scanIntervalSeconds>5</scanIntervalSeconds>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
</project>
```

說明：

- 載入spring-webmvc 、spring-tx framework
- 我們View的部分會用到JSP以及JSTL Tag，故也需要加入對應的Servlet Dependency，加入Servlet 3.1.0是因為之後會以Java Config的方式初始化Java Web程式，先前的版本不支援。
- 目前Maven專案預設JDK的Complier Version是1.5，改用JDK1.8
- 加載Jetty 可直接測試時的Web 容器

- 專案上按右鍵->Maven->Update Project->OK




- 修改 src\main\webapp\WEB-INF\web.xml
  - Spring MVC 是透過名為 DispatcherServlet 的程式來將使用者的 Request 導向我們的 Controller (處理請求的控制程式 )。而為了讓 JavaEE Container 啟動時知道我們將會使用這個 DispatcherServlet 所以我們必須先在設定檔宣告，該設定檔即為 WEB-INF\web.xml，設定如下：

**web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                                 http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
             version="3.1">

        <display-name>Archetype Created Web Application</display-name>

        <servlet>
        <servlet-name>dispatcherServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>
                /WEB-INF/dispatcher-servlet.xml
            </param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

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

</web-app>
```

  - Servlet ：可以看到上述 web.xml 中定義了 Servlet 名稱 (別名) 為dispatcherServlet，實作 Class 為剛剛提到的 DispatcherServlet 。而 load-on-startup 指定為 1，可以解讀為當你在 web.xml 裡面定義很多個 Servlet 的情況下， Jave EE Container 啓動時載入這些 Servlet 的順序。若不指定的話，它就會隨機選擇載入時間點。
  - Servlet-mapping：指明了所有符合 url-pattern '/' 的請求，都會交由dispatcherServlet 來處理。例如：當使用者發出 /query 的 Request ，因符合定義 "/" 路徑底下，此時 Web 容器會將Request 交由 dispatcherServlet 負責處理。
  - 設定 web.xml 只是讓 JavaEE Container 認識 Servlet，針對 Spring 功能還有許多參數需要去宣告，因此我們還需要一個 Spring 核心功能設定檔，這個檔案預設讀取位置是在 /WEB-INF 底下，設定檔預設名稱為 <servletname>-context.xml ，官方教學雖然是這樣寫但實際測試名稱應該是 <servletname>-servlet.xml。若你沒新增設定檔或設定檔名稱錯誤，於啟動專案時會出現找不到該設定檔的錯誤訊息，因此到時後可以檢查錯誤訊息裡面顯示的設定檔名稱。本例中是直接指明設定檔的路徑。
- 在 src\main\webapp\WEB-INF\ 下新增 dispatcher-servlet.xml

dispatcher-servlet.xml

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

        <context:component-scan base-package="com.demo.lab.controller" />
        <context:annotation-config />

        <mvc:annotation-driven />

        <mvc:resources location="/resources/" mapping="/resources/**" />

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

- <context：component-scan base-package="com.demo.lab.controller" />：
  聲明 Spring 掃描特定 package 底下含有@Component, @Controller, @Repository, @Service等注解的Class為 Bean。
- <mvc：annotation-driven /> ：
  因為我們將使用Annotation來作URL Mapping，故必須宣告<mvc: annotation-driven>
  此參數為將 Request 自動導向對應 Controller 的功能開啓。
- <mvc:resources>：
  resources通常也會作mapping，以免網頁存取不到相對路徑的靜態檔案如css檔以及圖檔等。
- ViewResolver bean：
  定義一個render jsp網頁的bean，其類別為org.springframework.web.servlet.view.InternalResourceViewResolver，在jsp網頁中常用的JSTL標籤故一併宣告，另外通常Controller傳回的是邏輯名稱，故在Resolver這邊需要宣告prefix告訴Spring MVC 網頁的資料夾在哪裏，宣告suffix告訴Spring MVC網頁副檔名為.jsp。

- 在 src\main\java 下新增 package: com.demo.lab.controller，並新增 HelloController.java

HelloController.java

```
package com.demo.lab.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class HelloController {

        @RequestMapping(value="/", method=RequestMethod.GET)
        public String hello(Model model){
                model.addAttribute("hello", "Hello World!"); //hellojspHello World!
                return "index";
        }
}
```

- @Controller 將此程式標註為處理 Request 的 Controller，裡面有一個名為 hello 的 Method，而這個 Method 帶有 @RequestMapping ( value="/" )，其意思為當使用者 Request 為 /的HTTP GET 請求時，Request 會對應到 HelloController 裡的 hello Method。此方法帶有Model參數，以利傳遞變數到jsp網頁，此時該 Method 執行完就會交由對應的 View 來顯示，意即 /WEB-INF/views/index.jsp。

- Controller 處理完後會傳給View層中的ViewResolver產生回應Response的頁面或是其他格式如JSON或是XML，Controller、Model、View層間通常以org.springframework.ui.Model物件來傳遞data，該物件本質上是個Map。

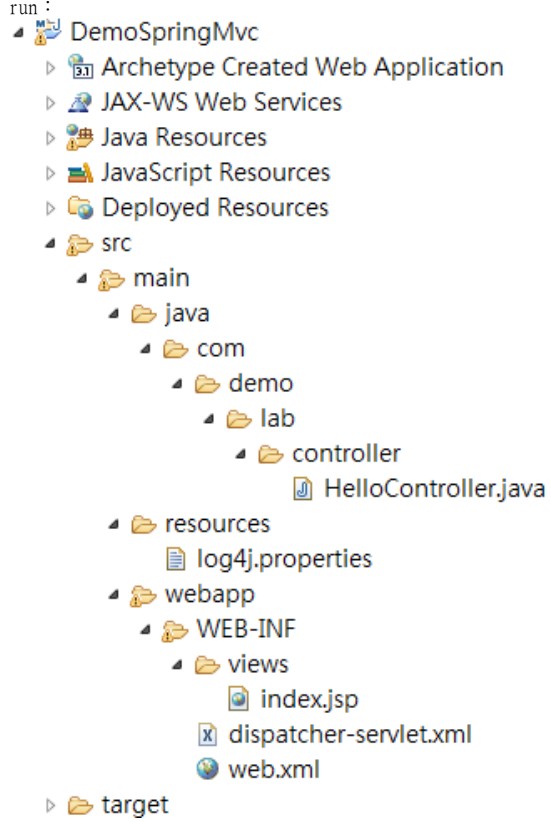
- 在 src\main\webapp\WEB-INF\ 下新增 views 目錄，並新增 index.jsp

**index.jsp**

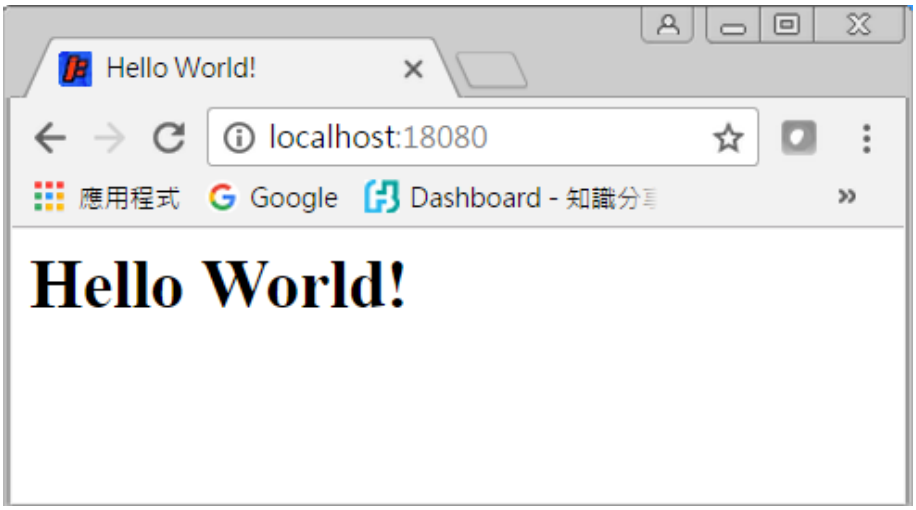
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<html>
        <head>
                <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
                <title>${hello}</title>
        </head>
        <body>
                <h1>${hello}</h1>
        </body>
</html>
```

  - 存取model變數用${}，此處指明取得 model 裏的hello attribute。

- run：



```
[INFO] Started o.e.j.m.p.JettyWebAppContext@31fc71ab{/,file:/D:/Workspace/DemoSpringMvc/src/main/webapp/,
[WARNING] !RequestLog
[INFO] Started ServerConnector@7b66322e{HTTP/1.1}{0.0.0.0:18080}
[INFO] Started @7308ms
[INFO] Started Jetty Server
[INFO] Starting scanner at interval of 5 seconds.
```




## 示例2 - 加入 DAO

- 範例取自 http://www.codejava.net/frameworks/spring/spring-mvc-with-jdbctemplate-example
- 修改 pom.xml

**pom.xml**

```
        <!--  2 properties -->
        <properties>
        ...
                <jt400.version>7.1.0</jt400.version>
                <dbcp.version>1.4</dbcp.version>
        </properties>

        <!--  3  dependencies -->
        <dependencies>
                ...
                <!-- spring jdbc -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-jdbc</artifactId>
                        <version>${spring.version}</version>
                </dependency>

                <!-- jt400 -->
                <dependency>
                        <groupId>com.fubonlife</groupId>
                        <artifactId>JT400</artifactId>
                        <version>${jt400.version}</version>
                </dependency>

                <!-- apache common datasource -->
                 <dependency>
                        <groupId>commons-dbcp</groupId>
                        <artifactId>commons-dbcp</artifactId>
                        <version>${dbcp.version}</version>
                </dependency>
        </dependencies>
```

- 在 src\main\java 下新增 package: com.demo.lab.model，並新增 Contact.java

Contact.java

```
package com.demo.lab.model;

public class Contact {
    private int id;
    private String name;
    private String email;
    private String address;
    private String telephone;

    public Contact() {
            super();
    }

    public Contact(String name, String email, String address, String telephone) {
        this.name = name;
        this.email = email;
        this.address = address;
        this.telephone = telephone;
    }

        public int getId() {
                return id;
        }

        public void setId(int id) {
                this.id = id;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

        public String getEmail() {
                return email;
        }

        public void setEmail(String email) {
                this.email = email;
        }

        public String getAddress() {
                return address;
        }

        public void setAddress(String address) {
                this.address = address;
        }

        public String getTelephone() {
                return telephone;
        }

        public void setTelephone(String telephone) {
                this.telephone = telephone;
        }
}
```

- 在 src\main\java 下新增 package: com.demo.lab.dao，並新增 ContactDAO.java

ContactDAO.java

```
package com.demo.lab.dao;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

import javax.sql.DataSource;

import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.ResultSetExtractor;
import org.springframework.jdbc.core.RowMapper;

import com.demo.lab.model.Contact;

/**
 * An implementation of the ContactDAO interface.
 *
 */
public class ContactDAO {

    private JdbcTemplate jdbcTemplate;

    public ContactDAO(DataSource dataSource) {
        jdbcTemplate = new JdbcTemplate(dataSource);
    }

    public void saveOrUpdate(Contact contact) {
        if (contact.getId() > 0) {
            // update
            String sql = "UPDATE contact SET name=?, email=?, address=?, "
                        + "telephone=? WHERE contact_id=?";
            jdbcTemplate.update(sql, contact.getName(), contact.getEmail(),
                    contact.getAddress(), contact.getTelephone(), contact.getId());
        } else {
            // insert
            String sql = "INSERT INTO contact (name, email, address, telephone)"
                        + " VALUES (?, ?, ?, ?)";
            jdbcTemplate.update(sql, contact.getName(), contact.getEmail(),
                    contact.getAddress(), contact.getTelephone());
        }

    }

    public void delete(int contactId) {
        String sql = "DELETE FROM contact WHERE contact_id=?";
        jdbcTemplate.update(sql, contactId);
    }

    public List<Contact> list() {
        String sql = "SELECT * FROM contact";
        List<Contact> listContact = jdbcTemplate.query(sql, new RowMapper<Contact>() {

            @Override
            public Contact mapRow(ResultSet rs, int rowNum) throws SQLException {
                Contact aContact = new Contact();

                aContact.setId(rs.getInt("contact_id"));
                aContact.setName(rs.getString("name"));
                aContact.setEmail(rs.getString("email"));
                aContact.setAddress(rs.getString("address"));
                aContact.setTelephone(rs.getString("telephone"));

                return aContact;
            }

        });

        return listContact;
    }

    public Contact get(int contactId) {
```

```
        String sql = "SELECT * FROM contact WHERE contact_id=" + contactId;
        return jdbcTemplate.query(sql, new ResultSetExtractor<Contact>() {

            @Override
            public Contact extractData(ResultSet rs) throws SQLException,
                    DataAccessException {
                if (rs.next()) {
                    Contact contact = new Contact();
                    contact.setId(rs.getInt("contact_id"));
                    contact.setName(rs.getString("name"));
                    contact.setEmail(rs.getString("email"));
                    contact.setAddress(rs.getString("address"));
                    contact.setTelephone(rs.getString("telephone"));
                    return contact;
                }

                return null;
            }

        });
    }
}
```

說明：

- 使用 Spring-jdbc 框架 存取data
- constructor 中的 dataSource 是由 bean 注入

- 修改 src\main\webapp\WEB-INF\dispatcher-servlet.xml

dispatch-servlet.xml

```
<beans>
        ...
        <!-- <context:property-placeholder location="classpath:my.properties"/>  -->
        <!--2 bean -->
        <bean id="f06DataSource" class="org.apache.commons.dbcp.BasicDataSource" init-method="
getLogWriter">
                <property name="driverClassName" value="com.ibm.as400.access.AS400JDBCDriver"></property>
                <property name="url" value="jdbc:as400://10.42.16.36/LIBIT050"></property>
                <property name="username" value="axxxx"></property>
                <property name="password" value="xxxxxxx"></property>  <!--400 -->
                <property name="initialSize" value="20"></property>
                <property name="maxActive" value="50"></property>
                <property name="maxIdle" value="20"></property>
                <property name="minIdle" value="10"></property>
        </bean>

        <bean id="contactDAO" class="com.demo.lab.dao.ContactDAO" >
                <constructor-arg ref="f06DataSource" />
        </bean>

        <!-- You can split your XML configuration file into multiple files to make it more modular.
        Here is an example of importing three other XML configuration files from a main configuration
file.
        <import resource="module2/config2.xml"/>
    <import resource="/resources/config3.xml"/>
        -->
</beans>
```

說明：

- ApplicationContext configuration file 是可以切分成多個，由一個主要的import 進來，做到模組化
- 建立 DataSource bean ，呼叫init-method 做初始化
- 可以使用 <context:property-placeholder .../> 標籤載入properties file，建Bean 的 property 可用參數
- 使用constructor based 方式建立DAO bean, 並將DataSource bean注入到 DAO的建構子參數
- 其它常見建Bean 方式
    - Creating A Bean Instance with A Factory Method /

```
<bean id="localDate" class="java.time.LocalDate"
    factory-method="now"/>
```

- Using A Destroy Method

```
<bean id="executorService" class="java.util.concurrent.Executors"
    factory-method="newCachedThreadPool"
    destroy-method="shutdown"/>
```

- Passing Mutiple Arguments to a Constructor

```
<bean name="featuredProduct" class="springintro.bean.Product">
    <constructor-arg name="name" value="Ultimate Olive Oil"/>
    <constructor-arg name="description"
        value="The purest olive oil on the market"/>
    <constructor-arg name="price" value="9.95"/>
</bean>
```

- Setter-based Dependency Injection

```
public class Employee {
    private String firstName;
    private String lastName;
    private Address homeAddress;

    public Employee() {
    }

    //getter & setter

}
```

```
public class Address {
  private String line1;
    private String line2;
    private String city;
    private String state;
    private String zipCode;
    private String country;

    public Address(String line1, String line2, String city,
            String state, String zipCode, String country) {
        this.line1 = line1;
        this.line2 = line2;
        this.city = city;
        this.state = state;
        this.zipCode = zipCode;
        this.country = country;
    }

    //getter & setter

}
```

```
<bean name="simpleAddress" class="springintro.bean.Address">
    <constructor-arg name="line1" value="151 Corner Street"/>
    <constructor-arg name="line2" value=""/>
    <constructor-arg name="city" value="Albany"/>
    <constructor-arg name="state" value="NY"/>
    <constructor-arg name="zipCode" value="99999"/>
    <constructor-arg name="country" value="US"/>
</bean>

<bean name="employee1" class="springintro.bean.Employee">
    <property name="homeAddress" ref="simpleAddress"/>
    <property name="firstName" value="Junior"/>
    <property name="lastName" value="Moore"/>
</bean>
```

- XX

- package: com.demo.lab.controller 下新增 ContactController.java

ContactController.java

```
package com.demo.lab.controller;

import java.io.IOException;
import java.util.List;

import javax.servlet.http.HttpServletRequest;

import org.apache.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import com.demo.lab.dao.ContactDAO;
import com.demo.lab.model.Contact;

@Controller
public class ContactController {
        final static Logger log = Logger.getLogger(ContactController.class);

    @Autowired
    private ContactDAO contactDAO;


    @RequestMapping(value="/lab2")
    public ModelAndView listContact(ModelAndView model, HttpServletRequest request) throws IOException{
            log.info("ContactController, reuest url:" + request.getRequestURL());

        List<Contact> listContact = contactDAO.list();
        model.addObject("listContact", listContact);
        model.setViewName("lab2");

        return model;
    }

    @RequestMapping(value = "/newContact", method = RequestMethod.GET)
    public ModelAndView newContact(ModelAndView model) {
        Contact newContact = new Contact();
        model.addObject("contact", newContact);
        model.setViewName("ContactForm");
        return model;
    }

    @RequestMapping(value = "/saveContact", method = RequestMethod.POST)
    public ModelAndView saveContact(@ModelAttribute Contact contact) {
        contactDAO.saveOrUpdate(contact);
        return new ModelAndView("redirect:/lab2");
    }

    @RequestMapping(value = "/deleteContact", method = RequestMethod.GET)
    public ModelAndView deleteContact(HttpServletRequest request) {
        int contactId = Integer.parseInt(request.getParameter("id"));
        contactDAO.delete(contactId);
        return new ModelAndView("redirect:/lab2");
    }

    @RequestMapping(value = "/editContact", method = RequestMethod.GET)
    public ModelAndView editContact(HttpServletRequest request) {
        int contactId = Integer.parseInt(request.getParameter("id"));
        Contact contact = contactDAO.get(contactId);
        ModelAndView model = new ModelAndView("ContactForm");
        model.addObject("contact", contact);

        return model;
    }
}
```

說明：

- saveContact(@ModelAttribute Contact contact) post request ContactForm.jsp

  <form:form action="saveContact" method="post" modelAttribute="contact">

- **Model**: is an interface it contains four addAttribute and one merAttribute method.

  **ModelMap**: implements Map interface. It also contains Map method.

  **ModelAndView**: is just a container for both a `ModelMap` and a view object. It allows a controller to return both as a single value.

- 在 src\main\webapp\WEB-INF\views\下, 新增 lab2.jsp

lab2.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Contact Manager Home</title>
    </head>
    <body>
        <div align="center">
            <h1>Contact List</h1>
            <h3><a href="/newContact">New Contact</a></h3>
            <table border="1">
                <th>No</th>
                <th>Name</th>
                <th>Email</th>
                <th>Address</th>
                <th>Telephone</th>
                <th>Action</th>

                <c:forEach var="contact" items="${listContact}" varStatus="status">
                <tr>
                    <td>${status.index + 1}</td>
                    <td>${contact.name}</td>
                    <td>${contact.email}</td>
                    <td>${contact.address}</td>
                    <td>${contact.telephone}</td>
                    <td>
                        <a href="/editContact?id=${contact.id}">Edit</a>
                            
                        <a href="/deleteContact?id=${contact.id}">Delete</a>
                    </td>

                </tr>
                </c:forEach>
            </table>
        </div>
    </body>
</html>
```

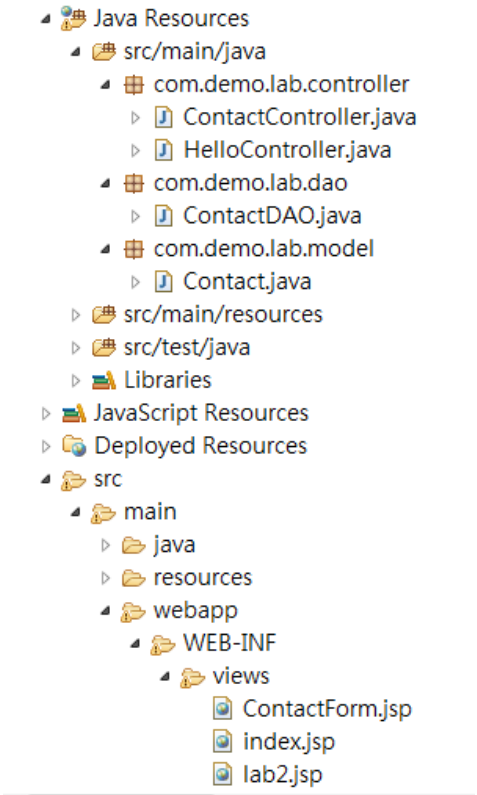
再新增 ContactForm.jsp

ContactForm.jsp

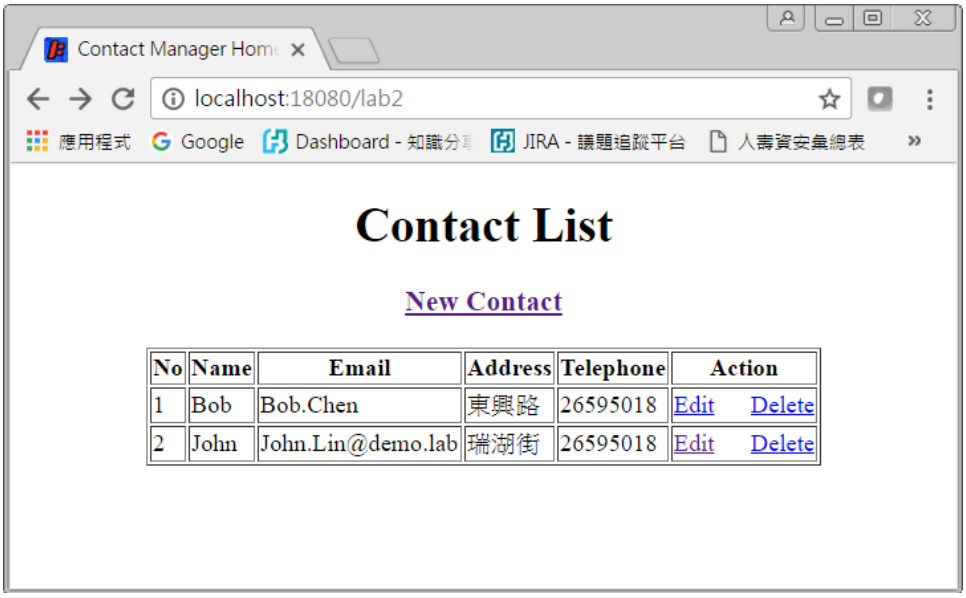
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>New/Edit Contact</title>
</head>
<body>
    <div align="center">
        <h1>New/Edit Contact</h1>
        <form:form action="saveContact" method="post" modelAttribute="contact">
        <table>
            <form:hidden path="id"/>
            <tr>
```

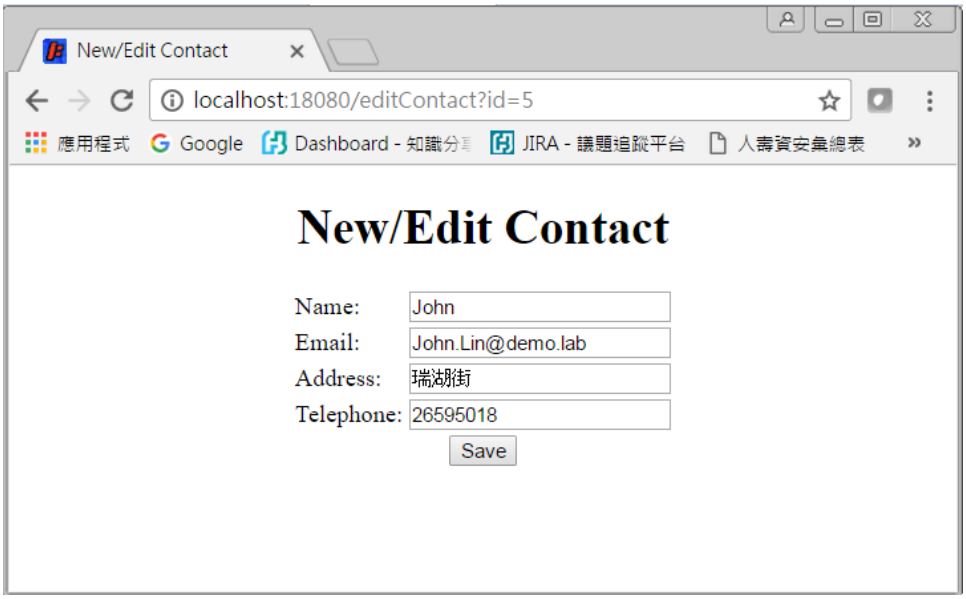
```
                <td>Name:</td>
                <td><form:input path="name" /></td>
            </tr>
            <tr>
                <td>Email:</td>
                <td><form:input path="email" /></td>
            </tr>
            <tr>
                <td>Address:</td>
                <td><form:input path="address" /></td>
            </tr>
            <tr>
                <td>Telephone:</td>
                <td><form:input path="telephone" /></td>
            </tr>
            <tr>
                <td colspan="2" align="center"><input type="submit" value="Save"></td>
            </tr>
        </table>
        </form:form>
    </div>
</body>
</html>
```

- run：




```
[INFO] Initializing Spring FrameworkServlet 'dispatcherServlet'
2017-02-28 15:09:08,652 INFO  [main] [DispatcherServlet] FrameworkServlet 'dispatcherServlet': initialization started
2017-02-28 15:09:08,671 INFO  [main] [XmlWebApplicationContext] Refreshing WebApplicationContext for namespace 'dispatcherServlet-servlet': startup date [Tue
2017-02-28 15:09:08,697 INFO  [main] [XmlBeanDefinitionReader] Loading XML bean definitions from ServletContext resource [/WEB-INF/dispatcher-servlet.xml]
2017-02-28 15:09:09,146 INFO  [main] [AutowiredAnnotationBeanPostProcessor] JSR-330 'javax.inject.Inject' annotation found and supported for autowiring
2017-02-28 15:09:15,803 INFO  [main] [RequestMappingHandlerMapping] Mapped "{[/lab2]}" onto public org.springframework.web.servlet.ModelAndView com.demo.lab.c
2017-02-28 15:09:15,804 INFO  [main] [RequestMappingHandlerMapping] Mapped "{[/newContact],methods=[GET]}" onto public org.springframework.web.servlet.ModelAr
2017-02-28 15:09:15,805 INFO  [main] [RequestMappingHandlerMapping] Mapped "{[/saveContact],methods=[POST]}" onto public org.springframework.web.servlet.Model
2017-02-28 15:09:15,805 INFO  [main] [RequestMappingHandlerMapping] Mapped "{[/deleteContact],methods=[GET]}" onto public org.springframework.web.servlet.Mode
2017-02-28 15:09:15,805 INFO  [main] [RequestMappingHandlerMapping] Mapped "{[/editContact],methods=[GET]}" onto public org.springframework.web.servlet.ModelA
2017-02-28 15:09:15,806 INFO  [main] [RequestMappingHandlerMapping] Mapped "{[/],methods=[GET]}" onto public java.lang.String com.demo.lab.controller.HelloCon
2017-02-28 15:09:15,970 INFO  [main] [RequestMappingHandlerAdapter] Looking for @ControllerAdvice: WebApplicationContext for namespace 'dispatcherServlet-serv
2017-02-28 15:09:16,017 INFO  [main] [RequestMappingHandlerAdapter] Looking for @ControllerAdvice: WebApplicationContext for namespace 'dispatcherServlet-serv
2017-02-28 15:09:16,105 INFO  [main] [SimpleUrlHandlerMapping] Mapped URL path [/resources/**] onto handler 'org.springframework.web.servlet.resource.Resource
2017-02-28 15:09:16,206 INFO  [main] [DispatcherServlet] FrameworkServlet 'dispatcherServlet': initialization completed in 7554 ms
[INFO] Started o.e.j.m.p.JettyWebAppContext@75361cf6{/,file:/D:/Workspace/DemoSpringMvc/src/main/webapp/,AVAILABLE}{file:/D:/Workspace/DemoSpringMvc/src/main/
[WARNING] !RequestLog
[INFO] Started ServerConnector@50110971{HTTP/1.1}{0.0.0.0:18080}
[INFO] Started @16334ms
[INFO] Started Jetty Server
[INFO] Starting scanner at interval of 5 seconds.
```




## Writing Request-Handling Methods

A request-handling method can have a mix of argument types as well as one of a variety of return types. For example, if you need access to the HttpSession object in your method, you can add HttpSession as an argument and Spring will pass the correct object for you:

```
@RequestMapping("/uri")
public String myMethod(HttpSession session) {
    ...
    session.addAttribute(key, value);
    ...
}
```

Or, if you need the client locale and the HttpServletRequest, you can include both as method arguments like this.

```
@RequestMapping("/uri")
public String myOtherMethod(HttpServletRequest request,
        Locale locale) {
    ...
    // access Locale and HttpServletRequest here
    ...
}
```

Here is the list of argument types that can appear as arguments in a request-handling method.

- javax.servlet.ServletRequest or javax.servlet.http.HttpServletRequest
- javax.servlet.ServletResponse or javax.servlet.http.HttpServletResponse
- javax.servlet.http.HttpSession
- org.springframework.web.context.request.WebRequest or org.springframework.web.context.request.NativeWebRequest
- java.util.Locale
- java.io.InputStream or java.io.Reader
- java.io.OutputStream or java.io.Writer
- java.security.Principal
- HttpEntity<?> parameters
- java.util.Map / org.springframework.ui.Model / org.springframework.ui.ModelMap
- org.springframework.web.servlet.mvc.support.RedirectAttributes
- org.springframework.validation.Errors / org.springframework.validation.BindingResult
- Command or form objects
- org.springframework.web.bind.support.SessionStatus
- org.springframework.web.util.UriComponentsBuilder
- Types annotated with @PathVariable, @MatrixVariable, @RequestParam, @RequestHeader, @RequestBody, or @RequestPart.

Of special importance is the org.springframework.ui.Model type. This is not a Servlet API type, but rather a Spring MVC type that contains a Map. Every time a request-handling method is invoked, Spring MVC creates a Model object and populates its Map with potentially various objects.

A request-handling method can return one of these objects.

- A ModelAndView object
- A Model object
- A Map containing the attributes of the model
- A View object
- A String representing the logical view name
- void
- An HttpEntity or ResponseEntity object to provide access to the Servlet response HTTP headers and contents
- A Callable
- A DeferredResult
- Any other return type. In this case, the return value will be considered a model attribute to be exposed to the view

# 加入 Interceptor

Spring MVC 中的Interceptor 攔截請求類似於Servlet 的Filter，可對每一個請求處理前後做相關處理。

主要有兩種實現方式，第一種方式是直接繼承Spring 已經提供的實現了HandlerInterceptor  interface的抽象類別HandlerInterceptorAdapter (或者自己實作HandlerInterceptor  interface的類別)；第二種方式是實現Spring的WebRequestInterceptor interface，或者是繼承實現了WebRequestInterceptor 的類別。

HandlerInterceptor interface 中定義了三個方法，我們就是通過這三個方法來對請求進行攔截處理的。

1. preHandle (HttpServletRequest request, HttpServletResponse response, Object handle) 方法
    - 顧名思義，該方法會在請求處理之前進行調用。SpringMVC 中的Interceptor 是鏈式的調用的，在一個請求中可以同時存在多個Interceptor，各個Interceptor 的調用會依據它的定義順序依次執行。
    - 每個Interceptor 最先執行的都是preHandle 方法，所以可以在這個方法中進行一些前置初始化操作或者是對當前請求的一個前置處理，也可以在這個方法中進行一些判斷來決定請求是否要繼續進行下去。
    - 該方法的返回值是Boolean 類型的，當它返回false時，表示請求結束，後續的Interceptor 和Controller 都不會再執行；當返回值為true 時就會繼續調用下一個Interceptor 的preHandle 方法，如果已經是最後一個Interceptor 的時候就會是調用當前請求的Controller 方法。
2. postHandle (HttpServletRequest request, HttpServletResponse response, Object handle, ModelAndView modelAndView) 方法
    - postHandle 方法和afterCompletion 方法都只能是在當前所屬的Interceptor 的preHandle 方法的返回值為true 時才能被調用。
    - postHandle 方法，是在當前請求進行處理之後，也就是Controller 方法調用之後執行，但是它會在DispatcherServlet 進行viewResolve 之前被調用，所以我們可以在這個方法中對Controller 處理之後的ModelAndView 對象進行操作。
    - postHandle 方法被調用的方向跟preHandle 是相反的，也就是說先聲明的Interceptor 的postHandle 方法反而會後執行。
3. afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handle, Exception ex) 方法
    - 該方法也是需要當前對應的Interceptor 的preHandle 方法的返回值為true 時才會執行。顧名思義，該方法將在整個請求結束之後，也就是在DispatcherServlet 進行viewResolve之後執行。
    - 這個方法的主要作用通常是用於進行資源清理工作的。

- 在 src\main\java 下新增 package: com.demo.lab.interceptor ，並新增 DemoLvlInterceptor.java

**DemoLvlInterceptor.java**

```
package com.demo.lab.interceptor;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import org.apache.log4j.Logger;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

public class DemoLv1Interceptor extends HandlerInterceptorAdapter {

        final static Logger log = Logger.getLogger(DemoLv1Interceptor.class);

    @Override
    public boolean preHandle(HttpServletRequest request,
            HttpServletResponse response, Object handler) throws Exception {
        log.info("DemoLv1Interceptor-preHandle, reuest url:" + request.getRequestURL());

            long startTime = System.currentTimeMillis();
        request.setAttribute("startTime", startTime);
        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request,
            HttpServletResponse response, Object handler,
            ModelAndView modelAndView) throws Exception {
            log.info("DemoLv1Interceptor-postHandle, reuest url:" + request.getRequestURL());

        long startTime = (Long) request.getAttribute("startTime");
        request.removeAttribute("startTime");
        long endTime = System.currentTimeMillis();
        log.info("request process time: " + (endTime - startTime) + "ms");
    }

    @Override
    public void afterCompletion(HttpServletRequest request,
            HttpServletResponse response, Object handler, Exception ex)
    throws Exception {
            log.info("DemoLv1Interceptor-afterCompletion, reuest url:" + request.getRequestURL());
    }
}
```

- 在新增 DemoLv2Interceptor.java

DemoLv2Interceptor.java

```
package com.demo.lab.interceptor;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.log4j.Logger;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

public class DemoLv2Interceptor extends HandlerInterceptorAdapter {

        final static Logger log = Logger.getLogger(DemoLv2Interceptor.class);

    @Override
    public boolean preHandle(HttpServletRequest request,
            HttpServletResponse response, Object handler) throws Exception {
        log.info("DemoLv2Interceptor-preHandle, reuest url:" + request.getRequestURL());
        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request,
            HttpServletResponse response, Object handler,
            ModelAndView modelAndView) throws Exception {
         log.info("DemoLv2Interceptor-postHandle, reuest url:" + request.getRequestURL());
    }
```

```
    @Override
    public void afterCompletion(HttpServletRequest request,
            HttpServletResponse response, Object handler, Exception ex)
    throws Exception {
            log.info("DemoLv2Interceptor-afterCompletion, reuest url:" + request.getRequestURL());
    }
}
```

- 修改 src\main\webapp\WEB-INF\dispatcher-servlet.xml

  Error rendering macro 'code': Invalid value specified for parameter 'com.atlassian.confluence.ext.code.render.InvalidValueException'

```
<!--  interceptor -->
<mvc:interceptors>
        <!-- beanInterceptormvc:interceptorsInterceptor -->
        <bean class="com.demo.lab.interceptor.DemoLv1Interceptor" />
        <mvc:interceptor>
                <mvc:mapping path="/lab2" />
                <!-- mvc:interceptor -->
                <bean class="com.demo.lab.interceptor.DemoLv2Interceptor" />
        </mvc:interceptor>
</mvc:interceptors>
```

- 說明：
  - 由上面的示例可以看出可以利用mvc:interceptors聲明一系列的攔截器，然後它們就可以形成一個攔截器鏈，攔截器的執行順序是按聲明的先後順序執行的，先聲明的攔截器的方法會先行。
  - 在mvc:interceptors標籤下聲明interceptor主要有兩種方式：
    1. 直接定義一個Interceptor實現類別的bean。使用這種方式聲明的Interceptor攔截器將會對所有的請求進行攔截。
    2. 使用mvc:interceptor標籤進行聲明。使用這種方式進行聲明的Interceptor可以通過mvc:mapping子標籤來定義需要進行攔截的請求路徑。
- run：
  - 分別訪問 http://localhost:18080/ 和 http://localhost:18080/lab2
  -

```
[INFO] Started ServerConnector@4337afd{HTTP/1.1}{0.0.0.0:18080}
[INFO] Started @14560ms
[INFO] Started Jetty Server
[INFO] Starting scanner at interval of 5 seconds.
2017-03-02 12:01:42,086 INFO  [qtp385784873-18] [DemoLv1Interceptor] DemoLv1Interceptor-preHandle, reuest url:http://localhost:18080/
2017-03-02 12:01:42,104 INFO  [qtp385784873-18] [DemoLv1Interceptor] DemoLv1Interceptor-postHandle, reuest url:http://localhost:18080/
2017-03-02 12:01:42,104 INFO  [qtp385784873-18] [DemoLv1Interceptor] request process time: 17ms
2017-03-02 12:01:43,072 INFO  [qtp385784873-18] [DemoLv1Interceptor] DemoLv1Interceptor-afterCompletion, reuest url:http://localhost:18080/
2017-03-02 12:01:46,714 INFO  [qtp385784873-19] [DemoLv1Interceptor] DemoLv1Interceptor-preHandle, reuest url:http://localhost:18080/lab2
2017-03-02 12:01:46,715 INFO  [qtp385784873-19] [DemoLv2Interceptor] DemoLv2Interceptor-preHandle, reuest url:http://localhost:18080/lab2
2017-03-02 12:01:46,725 INFO  [qtp385784873-19] [ContactController] ContactController, reuest url:http://localhost:18080/lab2
2017-03-02 12:01:47,134 INFO  [qtp385784873-19] [DemoLv2Interceptor] DemoLv2Interceptor-postHandle, reuest url:http://localhost:18080/lab2
2017-03-02 12:01:47,134 INFO  [qtp385784873-19] [DemoLv1Interceptor] DemoLv1Interceptor-postHandle, reuest url:http://localhost:18080/lab2
2017-03-02 12:01:47,135 INFO  [qtp385784873-19] [DemoLv1Interceptor] request process time: 419ms
2017-03-02 12:01:47,335 INFO  [qtp385784873-19] [DemoLv2Interceptor] DemoLv2Interceptor-afterCompletion, reuest url:http://localhost:18080/lab2
2017-03-02 12:01:47,336 INFO  [qtp385784873-19] [DemoLv1Interceptor] DemoLv1Interceptor-afterCompletion, reuest url:http://localhost:18080/lab2
```

# 加入全域 Exception Handler

Spring MVC 提供@ControllerAdvice 可以達到全域性的Exception Handler的需求。@ControllerAdvice 簡單講是增強型的@Controller（所以使用<context:component-scan>掃描時也能掃描到），可對類別內有注解@ExceptionHandler、@InitBinder、@ModelAttribute的方法套用到所有 @RequestMapping注解的方法。

It is typically used to define {@link ExceptionHandler @ExceptionHandler},
* {@link InitBinder @InitBinder}, and {@link ModelAttribute @ModelAttribute}
* methods that apply to all {@link RequestMapping @RequestMapping} methods.

使用該注解非常簡單，大多數時候其實只@ExceptionHandler比較常用，其他兩個用到的情境非常少，主要是用來把所有@Controller 發生的異常導到此控制器，而不是在各個@Controller注解的控制器內各自處理。

1. @ModelAttribute當需要設置全域Model 設定共同的attribute時比較有用。
2. @InitBinder當需要全域註冊時比較有用。
3. @ExceptionHandler，異常處理器，此注解的作用是當出現其定義的異常時進行處理的方法。

示例

- 修改 com.demo.lab.controller.ContactController.java ，新增以下2個 method

**ContactController.java**

```
    //2 method
@RequestMapping(value = "/copyContact", method = RequestMethod.GET)
public ModelAndView copyContact(HttpServletRequest request) {
```

```
        int contactId = Integer.parseInt(request.getParameter("name"));  //
        Contact contact = contactDAO.get(contactId);
        ModelAndView model = new ModelAndView("ContactForm");
        model.addObject("contact", contact);

        return model;
    }


    @RequestMapping(value = "/throwException", method = RequestMethod.GET)
    public ModelAndView raiseException(HttpServletRequest request) {
            throw new IllegalArgumentException("");
    }
```

- 在 package: com.demo.lab.controller 下新增ExceptionController.java

ExceptionController.java

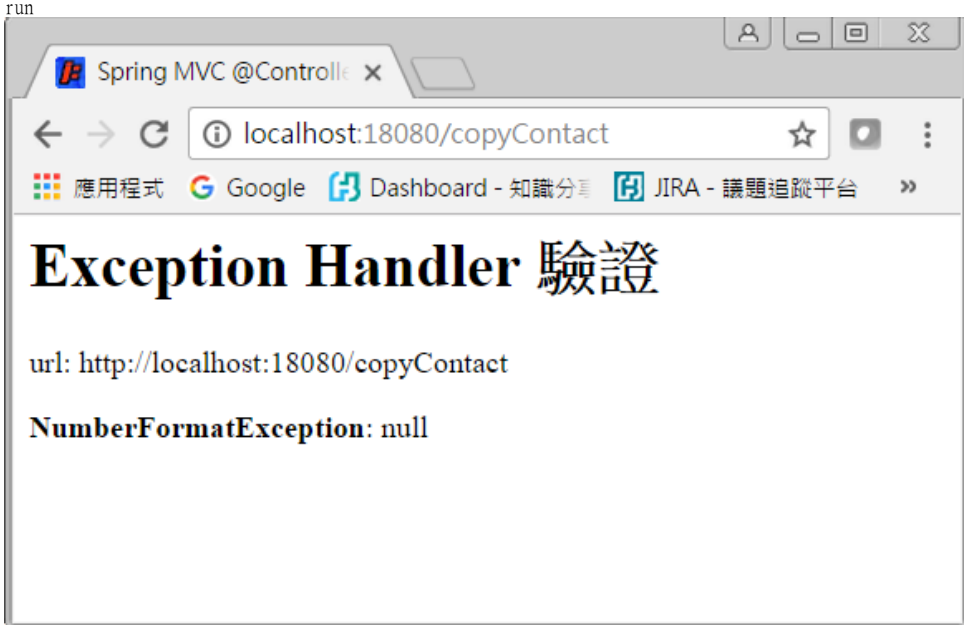
```
package com.demo.lab.controller;

import java.sql.SQLException;

import javax.servlet.http.HttpServletRequest;

import org.apache.log4j.Logger;
import org.springframework.dao.DataAccessException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.servlet.ModelAndView;

@ControllerAdvice
public class ExceptionController {

        final static Logger log = Logger.getLogger(ExceptionController.class);

        @ExceptionHandler(Exception.class)
        public ModelAndView exceptionHandler(HttpServletRequest request, Exception ex) {
                log.error(ex.getMessage(), ex);

                ModelAndView mav = new ModelAndView("exception");
                mav.addObject("url", request.getRequestURL());
                mav.addObject("name", ex.getClass().getSimpleName());
                mav.addObject("message", ex.getMessage());

                return mav;
        }

        @ExceptionHandler(IllegalArgumentException.class)
        public ModelAndView illegalArgumentExceptionHandler(HttpServletRequest request, Exception ex) {
                log.error(ex.getMessage(), ex);

                ModelAndView mav = new ModelAndView("exception");
                mav.addObject("url", request.getRequestURL());
                mav.addObject("name", ex.getClass().getSimpleName());
                mav.addObject("message", ex.getMessage());

                return mav;
        }

         @ExceptionHandler({SQLException.class, DataAccessException.class})
          public String databaseExceptionHandler(HttpServletRequest request, Exception ex) {
                 log.error(ex.getMessage(), ex);
            return "databaseError";
          }
}
```

- 在  src\main\webapp\WEB-INF\views\下, 新增 exception.jsp

exception.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Spring MVC @ControllerAdvice example</title>

</head>
<body>
        <h1>Exception Handler </h1>

        <p>url: ${url}</p>
        <b>${name}</b>:  ${message}
</body>
</html>
```

- run




## 加入Converters and Formatters

## 其它 Spring MVC 常見注解

### @Controller

標注在class上，表明這個class是Spring MVC裡的Controller，將其聲明為Spring 的一個Bean。 Dispatcher Servlet 會自動掃描標注了此注解的class，並將Web 請求 mapping 到標注了 @RequestMapping 的方法上。要特別指出的是，在聲明普通Bean的時候，使用 @Component、@Service、@Repository和@Controller 都是等同的，因為@Service、@Repository和@Controller都組合了@Component注解；但在 Spring MVC裡Dispatcher Servlet只會針對@Controller做掃描。

### @RequestMapping

用來mapping Web請求(含訪問路徑和參數)與負責處理該請求的class 和method。 @RequestMapping可標注在class和method上，標注在method上的路徑會繼承標注在class上的路徑。@RequestMapping支援Servlet的request 和response作為參數，也支援對request和response的媒體類型進行配置。

### @RequestBody

表明request的參數是在request body中，而不是直接在URL的? 後面。此注解標注在method的接收參數前。

### @ResponseBody

表明將method的返回值放在 response body中，而不是返回一個頁面。在很多基於Ajax的請求，可以以此注解返回資料而不是整個頁面；此注解可標注在返回值前或方法上。

### @PathVariable

用來接收路徑參數， 此注解標注在參數前。

### @RestController

是一個組合注解，組合了@Controller和@ResponseBody，這意味著若要設計一個返回值是放在response body的 controller時，使用此注解是較直覺的。若沒有用此注解，要實現上述功能，則需自己在程式碼中加@Controller 和 @ResonseBody兩個注解。

## 示例3 - @RestController & Json/XML convert

- 修改 pom.xml，添加 jackson 相關依賴，獲得物件與 json及xml 之間的轉換功能

**pom.xml**

```
        <!--  1  dependencies -->
        <dependencies>
                ...
                <!-- jackson dataformat -->
                 <dependency>
                        <groupId>com.fasterxml.jackson.dataformat</groupId>
                        <artifactId>jackson-dataformat-xml</artifactId>
                        <version>2.5.3</version>
                </dependency>
        </dependencies>
```

- package: com.demo.lab.controller 下新增 DemoLab3Controller.java

**Contact.java**

```
package com.demo.lab.controller;

import javax.servlet.http.HttpServletRequest;

import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.demo.lab.model.Contact;
```

```
@RestController  //1
@RequestMapping("/lab3")  //2
public class DemoLab3Controller {

	@RequestMapping(produces = "text/plain;charset=UTF-8")  //3
	public String index(HttpServletRequest request){  //4
		return "url:" + request.getRequestURL() + " can access";
	}

	@RequestMapping(value = "/pathvar/{str}", produces = "text/plain;charset=UTF-8")  //5
	public String demoPathVar(@PathVariable String str, HttpServletRequest request){
		return "url:" + request.getRequestURL() + " can access, str: " + str;
	}

	@RequestMapping(value = "/requestParam", produces = "text/plain;charset=UTF-8")  //6
	public String passRequestParam(Long id, HttpServletRequest request){
		return "url:" + request.getRequestURL() + " can access, id: " + id;
	}

	@RequestMapping(value = "/obj", produces = "application/json;charset=UTF-8")  //7
	public String passObj(Contact contact, HttpServletRequest request){
		return "url:" + request.getRequestURL() + " can access, contact: " + contact;
	}

	@RequestMapping(value = {"/name1", "/name2"}, produces = "text/plain;charset=UTF-8")  //8
	public String remove(HttpServletRequest request){
		return "url:" + request.getRequestURL() + " can access";
	}

	@RequestMapping(value = "/getjson", produces = "application/json;charset=UTF-8")  //9
	public Contact getJson(Contact contact){
		contact.setId(contact.getId() + 1);
		contact.setName(contact.getName() + "xx");
		return contact;  //10
	}

	@RequestMapping(value = "/getxml", produces = "application/xml;charset=UTF-8")  //11
	public Contact getXml(Contact contact){
		contact.setId(contact.getId() + 1);
		contact.setName(contact.getName() + "xx");
		return contact;  //12
	}
}
```

- 說明：
  1. 使用@RestController 聲明是控制器，並且將method的返回值放在 response body中，而不是返回一個頁面。
  2. 指示Dispatcher Servlet 將訪問路徑為 /lab3 開頭的，交由此控制器處理。裏面各method 的 @RequestMapping路徑都是 base在此路徑下。
  3. 此方法未標識路徑，因此使用class 層級定義的路徑/lab3；produces可定制返回的 response的 MIME type和 charset。
  4. 展示可接受 HttpServletRequest作為參數，當然也可以接受 HttpServletResponse作為參數。
  5. 展示接受路徑參數，並在method 參數前結合 @PathVairable 使用，例如訪問路徑為 /lab3/pathvar/abc。
  6. 展示一般 request參數的取得方式，例如訪問路徑為 /lab3/requestParam?id=1。
  7. 展示參數取得轉換為物件，例如訪問路徑為 /lab3/obj?id=5&name=xyz。
  8. 展示mapping不同的路徑到相同的method上的方法，例如訪問路徑為 /lab3/name1 或 /lab3/name2。
  9. 定制返回資料的MIME type為 json。
  10. 直接返回物件，物件會自動轉換為 json。
  11. 定制返回資料的MIME type為 xml。
  12. 直接返回物件，物件會自動轉換為 xml。


localhost:18080/lab3/g
localhost:18080/lab3/getjson?id=1&name=abc&email=abc.lin
應用程式
Google
Dashboard - 知識分
JIRA - 議題追蹤平台
人壽資安彙總表
{"id":2,"name":"abcxx","email":"abc.lin","address":null,"telephone":null}


localhost:18080/lab3/g
localhost:18080/lab3/getxml?id=1&name=abc&email=abc.lin
應用程式
Google
Dashboard - 知識分
JIRA - 議題追蹤平台
人壽資安彙總表
This XML file does not appear to have any style information associated with it. The document tree is shown below.
<Contact xmlns="">
<id>2</id>
<name>abcxx</name>
<email>abc.lin</email>
<address/>
<telephone/>
</Contact>