# —— Defining Bean in XML ——

<bean id="..." class="...">

<!-- configuration for this bean here -->

</bean>

——————— Dependency Injection Methods ——————

Constructor:

<bean id="employeeService"

class="org.packt.Spring.chapter2.dependencyinjection.EmployeeServiceImpl">

<constructor-arg ref="employeeDao" />

</bean>

<bean id="employeeDao"

class="org.packt.Spring.chapter2.dependencyinjection.EmployeeDaoImpl">

</bean>

Setter:

<bean id="employeeService"

class="org.packt.Spring.chapter2.dependencyinjection.EmployeeServiceImpl">

<property name="employeeDao" ref="employeeDao" />

</bean>

<bean id="employeeDao"

class="org.packt.Spring.chapter2.dependencyinjection.EmployeeDaoImpl">

</bean>

factory method:

<bean id="theStage"

class="com.springinaction.springidol.Stage"

factory-method="getInstance" />

Using p: to wire beans:

<bean id="kenny" class="com.springinaction.springidol.Instrumentalist"

p:song = "Jingle Bells"

p:instrument-ref = "saxophone" />

setSong and setInstrument are on the bean Instrumentalist

wiring collections:

<bean id="hank"

class="com.springinaction.springidol.OneManBand">

<property name="instruments">

<list>

<ref bean="guitar" />

<ref bean="cymbal" />

<ref bean="harmonica" />

</list>

</property>

</bean>

Inner Bean:

<bean id="atmBean" class="org.packt.Spring.chapter2.setterinjection.ATM">

<property name="printer">

<bean class="org.packt.Spring.chapter2.setterinjection.Printer">

<property name="message"

value="The balance information is printed by Printer for the account number"></property>

</bean>

</property>

</bean>

public class ATM {

private Printer printer;

public Printer getPrinter() {

return printer;

}

public void setPrinter(Printer printer) {

this.printer = printer;

}

public void printBalance(String accountNumber) {

getPrinter().printBalance(accountNumber);

}

}

Abstract Bean Inheritance:

<bean id="indianEmployee" class="org.packt.Spring.chapter2.beaninheritance.Employee"

abstract="true">

<property name="country" value="India"></property>

</bean>

<bean id="employeeBean" parent="indianEmployee">

<property name="employeeId" value="1065"></property>

<property name="employeeName" value="Ravi Kant Soni"></property>

</bean>

———————— Using Properties ———————

<util:properties id="settings"

location="classpath:settings.properties"/>

<property name="accessToken" value="#{settings['twitter.accessToken']}"/>

<property name="homePath" value="#{systemEnvironment['HOME']}"/>

<property name="homePath" value="#{systemProperties['application.home']}"/>

# —— Autowring And Autodiscovery ———

<bean id="kenny"

class="com.springinaction.springidol.Instrumentalist"

autowire="byName">

<property name="song" value="Jingle Bells" />

</bean>

—— Autowiring using Annotation ————

<context:annotation-config />in spring context file

@Autowired:

on private members:

@Autowired

private Instrument instrument;

on constructors:

@Autowired

public Instrumentalist(Instrument instrument) {

this.instrument = instrument;

}

on methods:

@Autowired

public void setInstrument(Instrument instrument) {

this.instrument = instrument;

}

@Autowired(required=false)

private Instrument instrument;

———— Using @value and @inject ——————

# ————Auto Discovery of Bean without declaring Beans in the XML ———

<context:component-scan>

@Component—A general-purpose stereotype annotation indicating that the class is a Spring component

@Controller—Indicates that the class defines a Spring MVC controller

@Repository—Indicates that the class defines a data repository

@Service—Indicates that the class defines a service

————— Java based Configruation————

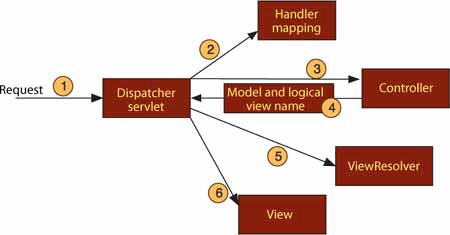
Needs:

<context:component-scan>

and:

@Configuration on the Java class

# Spring MVC



Setting up MVC:

<servlet>

<servlet-name>spitter</servlet-name>

<servlet-class> org.springframework.web.servlet.DispatcherServlet </servlet-class>

<load-on-startup>1</load-on-startup> </servlet>

it would look for spitter-servlet.xml in web-inf to local the application context file

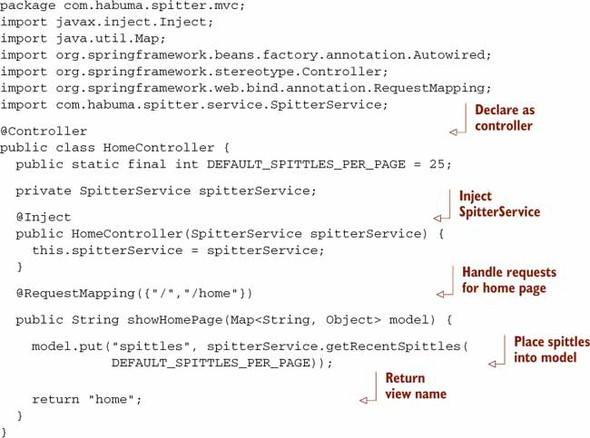
for handling static resources:



add this to your spring context .xml file to enable annotations like @controller:

<mvc:annotation-driven/>

Example of Controller



Resolving View:

<bean class= "org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/views/"/> <property name="suffix" value=".jsp"/>

</bean>

Add Listener to load other context file. It is added in web.xml:

<listener>

<listener-class>com.zappos.zommons.spring.web.servlet.EncryptedContextLoaderListener</listener-class>

</listener>

--- Dispatcher Servlet ------

<servlet>

<servlet-name>mvc</servlet-name>

<servlet-class>com.zappos.zommons.spring.web.servlet.EncryptedDispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>

classpath:/mvc-context.xml

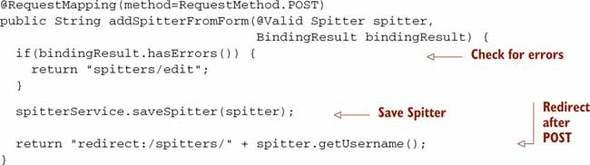
</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

Handling Form:



Handling Request with path variables:

@RequestMapping(value="/{username}", method=RequestMethod.GET)

public String showSpitterProfile(@PathVariable String username, Model model) {

model.addAttribute(spitterService.getSpitter(username)); return "spitters/view"; }

# Spring REST

GET Request:



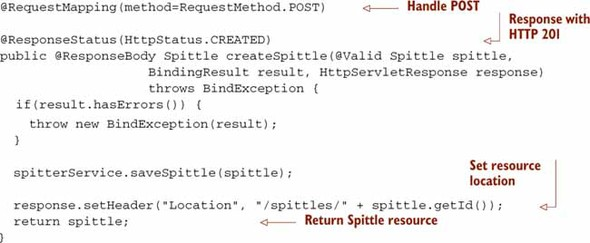
Put Request:

@RequestMapping(value="/{id}", method=RequestMethod.PUT) @ResponseStatus(HttpStatus.NO\_CONTENT)

public void putSpittle(@PathVariable("id") long id,@Valid Spittle spittle)

{ spitterService.saveSpittle(spittle); }

POST Request:



##### Returning Resource State in the Response Body:

@RequestMapping(value = "/{username}", method = RequestMethod.GET,

headers = {"Accept=text/xml, application/json"})

public @ResponseBody

Spitter getSpitter(@PathVariable String username)

{ return spitterService.getSpitter(username); }

HttpMessageConverters are automatically registered by Spring Context; only libraries such as for JSON need to be in path;

##### Receiving Resource State in the Request Body:

@RequestMapping(value = "/{username}", method = RequestMethod.PUT,headers = "Content-Type=application/json")

@ResponseStatus(HttpStatus.NO\_CONTENT)

public void updateSpitter(@PathVariable String username,

@RequestBody Spitter spitter)

{ spitterService.saveSpitter(spitter); }

Content Negotiation View Resolver:

<bean class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">

<property name="mediaTypes">

<map>

<entry key="json" value="application/json" />

<entry key="xml" value="text/xml" />

<entry key="htm" value="text/html" />

</map> </property>

<property name="defaultContentType" value="text/html" />

</bean>

### WRITING REST CLIENTS

public Spittle[] retrieveSpittlesForSpitter(String username)

{

return new RestTemplate().getForObject(

"http://localhost:8080/Spitter/spitters/{spitter}/spittles", Spittle[].class, username);

}

public Spittle[] retrieveSpittlesForSpitter(String username)

{

ResponseEntity<Spittle[]> response = new RestTemplate().getForEntity( "http://localhost:8080/Spitter/spitters/{spitter}/spittles", Spittle[].class, username); if(response.getStatusCode() == HttpStatus.NOT\_MODIFIED)

{ throw new NotModifiedException(); }

return response.getBody();

}

----------- Posting form data -----------

public Spitter postSpitterForObject(Spitter spitter) {

RestTemplate rest = new RestTemplate();

return rest.postForObject("http://localhost:8080/Spitter/spitters",spitter, Spitter.class);

}

RestTemplate rest = new RestTemplate();

ResponseEntity<Spitter> response = rest.postForEntity( "http://localhost:8080/Spitter/spitters", spitter, Spitter.class);

Spitter spitter = response.getBody();

URI url = response.getHeaders().getLocation();