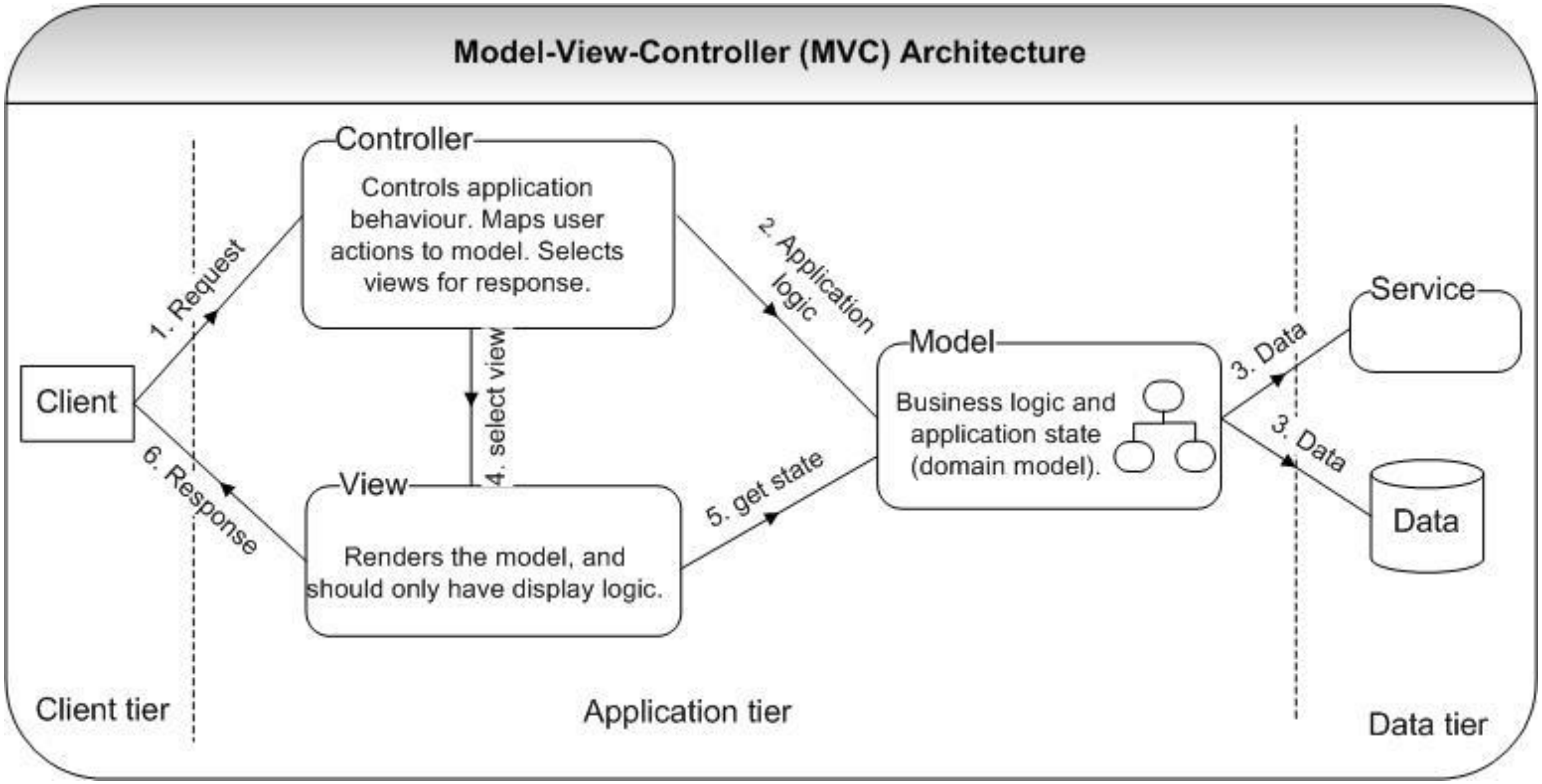
Spring MVC

Spring:: MVC :: Introduction

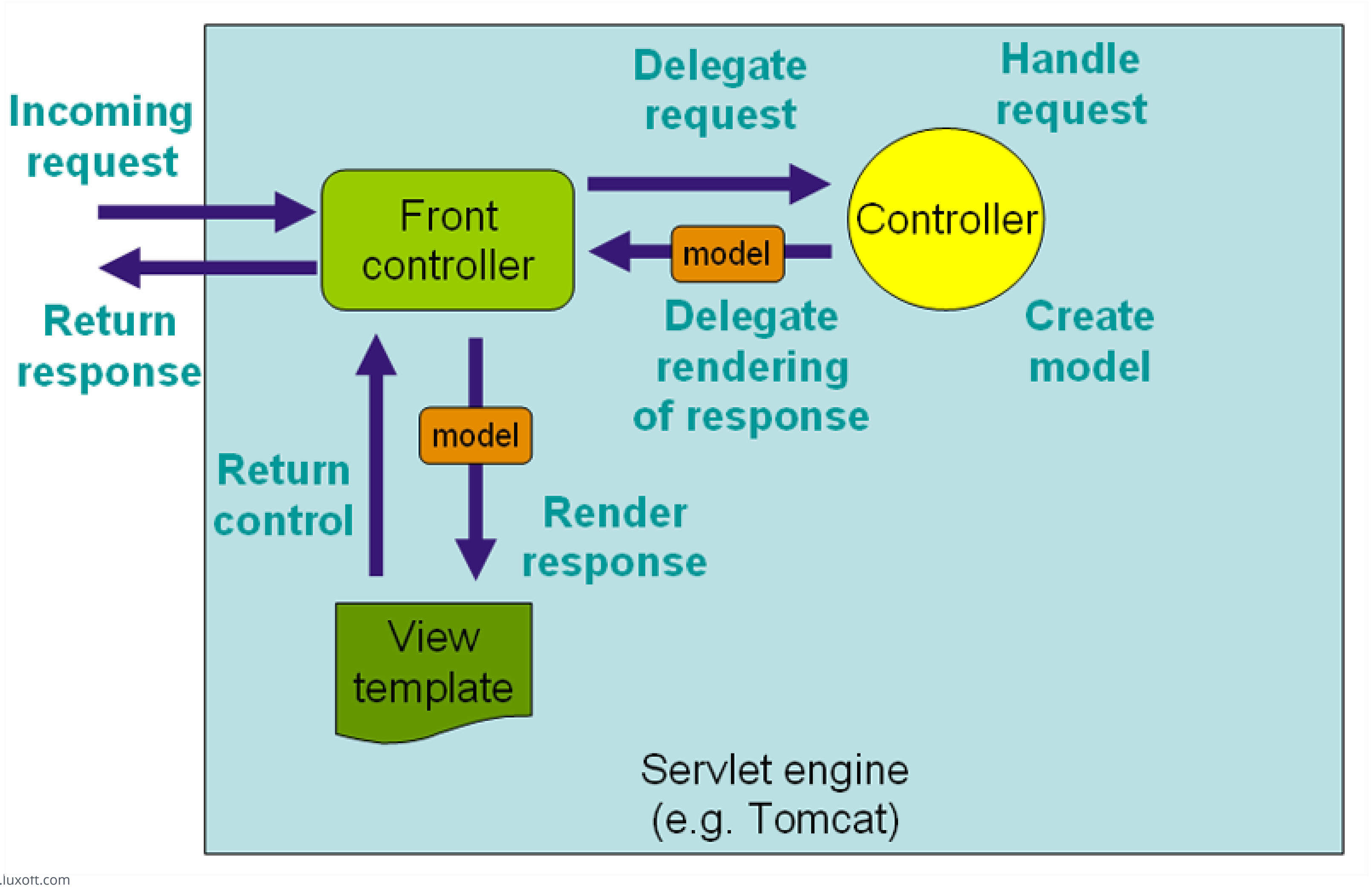
§ **MVC model:**



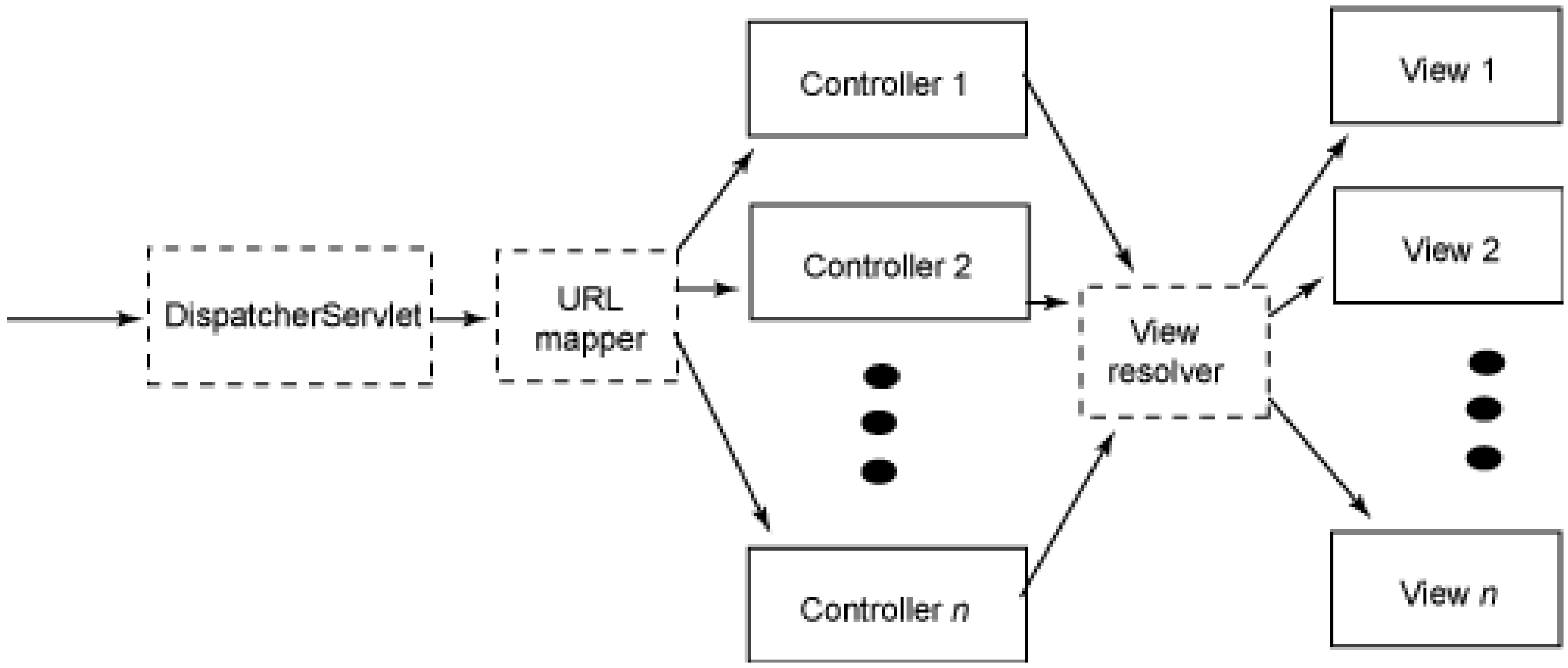
Model-View-Controller (MVC) Architecture
Controller
Controls application behaviour. Maps user actions to model. Selects views for response.
1. Request
2. Application logic
Client
Model
Business logic and application state (domain model).
4. select view
3. Data
Service
3. Data
Data
6. Response
View
Renders the model, and should only have display logic.
5. get state
Client tier
Application tier
Data tier

Spring:: MVC :: Introduction

- Spring model is more flexible:



Spring :: MVC :: Introduction



WebApplicationContext

- This is an extension of **ApplicationContext** that has some extra features necessary for web applications (for example, association with **ServletContext**).
- Adds three scopes of bean lifecycle that are only available in web context (request, scope, global).
- Special bean types can only exist in **WebApplicationContext**.

Spring MVC

Configuration

- To initialize a context, add **ContextLoaderListener** to web.xml
- During context initialization, beans defined in files **applicationContext.xml** and **[servlet-name]-servlet.xml** (except for lazy-init beans) are instantiated (for each **DispatcherServlet**)
- File set used in beans instantiation can be changed by specifying **contextConfigLocation** parameter in application descriptor.

Spring:: MVC :: WebApplicationContext (web.xml file)

```
<servlet>
          <servlet-name>dispatcher</servlet-name>
     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
     <init-param>
               <param-name>contextConfigLocation</param-name>
          <param-value>/WEB-INF/config/servlet-config.xml</param-value>
     </init-param>
</servlet>

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

Spring:: MVC :: @Controller

Ex1, Ex2

```
@Controller
public class HelloController {

    @RequestMapping(value ="/greeting")
    public String sayHello (Model model)
    {
        model.addAttribute("greeting", "Hello my friend!");

        return "hello-page";
    }
}
```

§ And add component-scan to application-context.xml:

```
<mvc:annotation-driven/>

<context:component-scan base-package="com.luxoft.jva010.mvc.controller"/>


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

@RequestMapping

Spring:: MVC :: @RequestMapping

Configuring mapping with **@RequestMapping** annotations:

Specified for:

- Class with **@Controller** annotation: path to all controller methods;
- Controller’s methods:
  - If class path is specified then the path is relative;
  - Absolute path if not specified for class;

Spring:: MVC :: @RequestMapping

```
@Controller
@RequestMapping("/owners/{ownerId}")
public class RelativePathUriTemplateController
{
  @RequestMapping("/pets/{petId}")
  public void findPet(@PathVariable String ownerId,
                             @PathVariable String petId, Model model)
  {
        // implementation
  }
}


@RequestMapping(value="/owners/{ownerId}/pets/{petId}”)
public String findPet(@PathVariable String ownerId,
                             @PathVariable String petId, Model model)
{
        // implementation
}
```

- URL patterns are supported;
- @PathVariable is used to bind variable (argument) to the value of URL template;
- Used in implementing RESTful services;

Spring:: MVC :: @RequestMapping, @RequestParam

- Binding to request parameters is made through **@RequestParam** annotation

```
@RequestMapping(method = RequestMethod.POST)
public String setupForm(
    @RequestParam("petId") int petId)
```

Spring:: MVC :: @RequestMapping, @RequestParam

- By default, such parameters are mandatory, but could be turned to optional with **required=false** command.

```
@RequestMapping(method = RequestMethod.GET)
public String showItem(
    @RequestParam(value="id", required=false) int id)
```

Request headers

Method catches only requests which has **Content-Type** header, which value starts with **text/ :**

```
@RequestMapping(value = "/headers", headers="content-type=text/*")
public String headersText(Model model)
{
        model.addAttribute("contentType", "text/*");

        return HEADERS_VIEW_NAME;
}
```

Request headers

Method catches only requests which has **Content-Type** header, which value equals to **application/json**

```
@RequestMapping(value = "/headers",
                                        headers="content-type=application/json")
public ResponseEntity<String> headersJson()
{
          String json = "{\"contentType\":\"application/json\"}";

          HttpHeaders responseHeaders = new HttpHeaders();
          responseHeaders.setContentType(MediaType.APPLICATION_JSON);

          return new ResponseEntity<String>(json, responseHeaders,
                                                  HttpStatus.CREATED);
}
```

Use of standard HttpServletRequest and HttpSession in controller

To provide easier work on legacy code we can use **javax.servlet.http.HttpServletRequest** and **javax.servlet.http.HttpSession**

```
@RequestMapping(value = "/some-url")
public String httpSession(HttpServletRequest request,
         HttpSession session)
{
        session.getId();

        return OTHER_VIEW_NAME;
}
```

Locale in Spring MVC

```
@RequestMapping(value = "/locale")
public String locale(Locale locale, Model model)
{
        model.addAttribute("content", "Locale language: " +
                                                            locale.getLanguage() );
        return OTHER_VIEW_NAME;
}
```

Principal contains information about autorized user.

```
@RequestMapping(value = "/principal")
public String principal(Principal principal, Model model)
{

	model.addAttribute("user", "Principal: " + principal.getName());

	return OTHER_VIEW_NAME;
}
```

Processing of additional requests in @RequestParam

As with **@PathVariable**, annotation **@RequestParam** translates request parameter to variable. Also you shouldn’t worry about type: Spring will cast it itself.

```
@RequestMapping(value = "/requestparam")
public String requestParam(@RequestParam("foo") int foo, Model model)
{
        model.addAttribute("content", "foo=" + foo );

        return OTHER_VIEW_NAME;
}
```

If the param is not required,
just set required property of **@RequestParam**:

@RequestParam(value="foo", **required=false**) int foo

Processing headers using @RequestHeader:

```
@RequestMapping(value = "/requestheader")
public String requestHeader(
                    @RequestHeader("User-Agent") String userAgent,
                    Model model)
{
          model.addAttribute("content", "User-Agent: " + userAgent );

          return OTHER_VIEW_NAME;
}
```

Supported arguments of handler methods:

§ **Request / Response objects** Subtypes can be used (for example, ServletRequest or HttpServletRequest);

§ **Servlet API Session object** (HttpSession). An argument of this type enforces the presence of a corresponding session when it is absent;

§ **WebRequest** and NativeWebRequest (org.springframework.web.context.request package) allows for request parameters access without ties to Servlet/Portlet API ;

§ **java.util.Locale** for the current request locale;

§ **InputStream / Reader** for access to the request's content;

§ **OutputStream / Writer** for generating the response's content;

§ **java.security.Principal** containing the currently authenticated user;

§ **@PathVariable** annotated parameters for access to URI template variables ;

§ **@RequestParam** annotated parameters for access to request parameters ;

Supported arguments of handler methods :

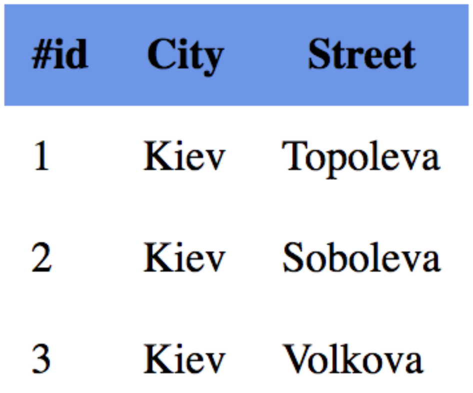
- **@RequestHeader** annotated parameters for access to specific request headers;
- **@RequestBody** for access to the request body;
- **java.util.Map** / org.springframework.ui.**Model** / org.springframework.ui.**ModelMap** to access to the model that is exposed to the web view;
- org.springframework.validation.**Errors** / org.springframework.validation.**BindingResult** : for access/validation results for an object. This argument shall follow the validated object **immediately**, as each object is validated and tied to validation result separately;
- org.springframework.web.bind.support.**SessionStatus** for controlling session state and session closing (it triggers cleanup of session attributes);

Supported method return types:

- **ModelAndView**: for returning model and View name;
- **Model**: for returning model. View name will be determined through RequestToViewNameTranslator ;
- **Map** is analogous to Model ;
- **View** is a view object. Model will be automatically inserted (changes introduced to model with handler argument will be inserted);
- **String** is a logical view name that is handled by ViewResolver;
- **void**: if method handles a response itself or if view name is determined through **RequestToViewNameTranslator**;
- If the method is annotated through **@ResponseBody**, the return type is written to the response HTTP body;
- Any other return type is considered to be a single model attribute using name specified through **@ModelAttribute** at the method level or automatically generated based on the return type;

Forms

```
<c:forEach items="${addresses}" var="address">
    <tr>
        <td><c:out value="${address.id}"></c:out></td>
        <td><c:out value="${address.city}"></c:out></td>
        <td><c:out value="${address.street}"></c:out></td>
    </tr>
</c:forEach>
```



#id	City	Street
1	Kiev	Topoleva
2	Kiev	Soboleva
3	Kiev	Volkova

We can use this code:

```
@RequestMapping("/addresses")
public String getAddresses(Model model)
{

    model.addAttribute("addresses",
                fillWithGenerated(new ArrayList<>()));

    return "addresses";
}
```

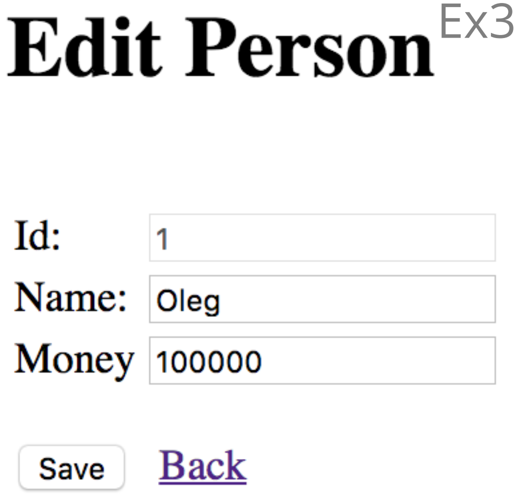
Or we can use @ModelAttribute

```
@ModelAttribute("addresses")
public List<Address> getData()
{
   return fillWithGenerated(new ArrayList<>());
}

@RequestMapping("/addresses")
public String getAddresses(Model model)
{
   return "addresses";
}
```

Spring :: MVC :: @ModelAttribute

Ex3

```
<form:form modelAttribute="personBean" method="POST"
                    action="/persons/edit/${personBean.id}">
<table>
    <tr>
      <td><form:label path="id">Id:</form:label></td>
      <td><form:input path="id" disabled="true"/></td>
    </tr>
  <tr>
        <td><form:label path="name">Name:</form:label></td>
        <td><form:input path="name"/></td>
    </tr>
  <tr>
    <td><form:label path="money">Money</form:label></td>
        <td><form:input path="money"/></td>
    </tr>
</table>
<br>
    <input type="submit" value="Save" />&nbsp;&nbsp;<a href="/persons/all">Back</a>
</form:form>
```



Edit Person

Id: 1
Name: Oleg
Money 100000

Spring :: MVC :: @ModelAttribute

```
@Controller
public class PersonController
{
    @Autowired
    private PersonService personService;

    @RequestMapping(value = "/persons/edit/{id}", method = RequestMethod.GET)
    public String getEdit(@PathVariable Long id, Model model)
    {
        System.out.println("#getEdit");

        model.addAttribute("personBean", personService.getById(id));
        return "person";
    }

    @RequestMapping(value = "/persons/edit/{id}", method = RequestMethod.POST)
    public String saveEdit(@ModelAttribute("personBean") Person person,
                            @PathVariable Integer id, Model model)
    {
        System.out.println("#saveEdit");

        personService.save(person);

        return "redirect:/persons/all";
    }
}
```

So we have 2 scenarios of using **@ModelAttribute**:

- Method annotation – to get the model attribute:

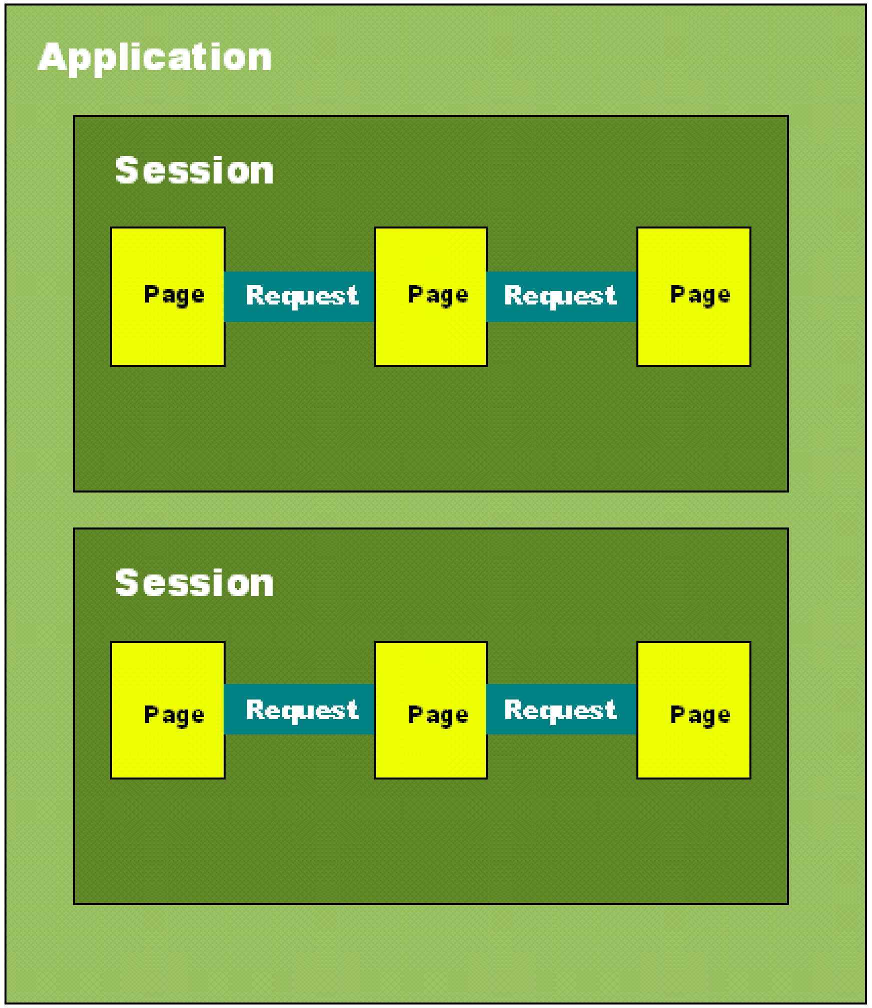
```
@ModelAttribute("addresses")
public List<Address> getData()
{
   return fillWithGenerated(new ArrayList<>());
}
```

- Annotation to the method parameter – links module attribute to the parameter. It can be used to get access to data entered by the user in the form:

```
public String saveEdit(@ModelAttribute("personBean") Person
person,…
```

Call to **@ModelAttribute** happens **before** calling method annotated by **@RequestMapping**.

Scopes in web-application: how long the bean does exist



Scope	Description
singleton	The single instance in IoC container
prototype	One definition, but multiple instances
request	One definition in single HTTP request Example: `<bean id="loginAction" class="com.foo.LoginAction" scope="request"/>`
session	Obe definition is HTTP session Example: `<bean id="userPreferences" class="com.foo.UserPreferences" scope="session"/>`
global session	One definition in a global HTTP-session (only for portlets).

Spring:: MVC :: @SessionAttributes

@SessionAttributes is specified at the level of controller class;

- Declares which model attributes should be stored in session for transferring between requests;
- Initialized when you put object into model;
- Refreshed from HTTP parameters, when controller method is being executed

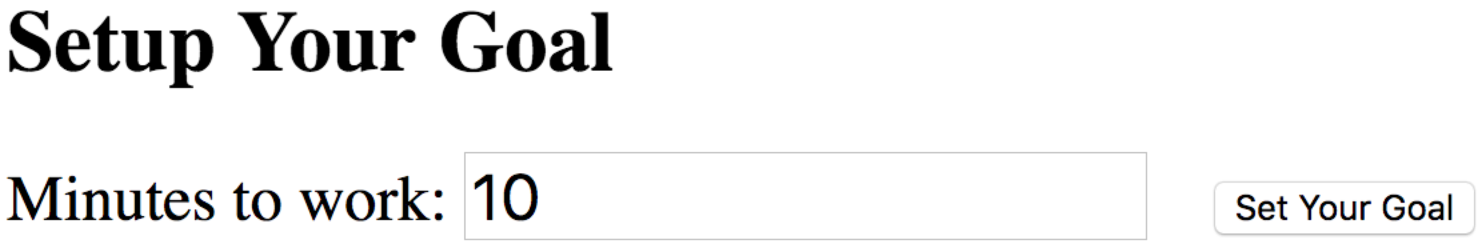
Processing POST request, Spring is doing the following:

Without @SessionAttributes: Spring creates new instance of User and fill it with form data

With @SessionAttributes: Spring gets User from session (it was placed to session when executing GET-request, i.e. @SessionAttributes was presented), refresh fields by form values and send to saveForm().

Spring :: MVC :: @SessionAttributes

```
@Controller
@SessionAttributes("goal")
public class GoalController
{

    @RequestMapping(value = "goal", method = RequestMethod.GET)
    public String addGoal(Model model, HttpSession session)
    {
        Goal goal = new Goal();
        goal.setMinutes(10);
        model.addAttribute("goal", goal);

        session.setAttribute("goal", goal);

        return "goal";
    }

    @RequestMapping(value = "goal", method = RequestMethod.POST)
    public String updateGoal(@ModelAttribute("goal") Goal goal)
    {
        System.out.println("Minutes updated: " + goal.getMinutes());

        return "redirect:minutes";
    }

}
```



Setup Your Goal

Minutes to work: 10

Exception handling

Spring:: MVC :: Exceptions

- Handling exceptions:
  - **`HandlerExceptionResolver`** interface and one ready **`SimpleMappingExceptionResolver`** implementation.
  - Mapping between exception classes and view names based on the **`Properties`**.
  - Catches exceptions thrown by handlers and passes them via model.
  - More flexible mechanism as compared to **web.xml**, as far as its own implementation can be done.

```
        <bean id="exceptionResolver"
class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
                <property name="exceptionMappings">
                        <value>
                                com.luxoft.jva010.mvc.CustomException=customException
                                java.lang.SecurityException=error403
                                java.lang.Throwable=error
                        </value>
                </property>
        </bean>
```

Spring :: MVC :: @ExceptionHandler

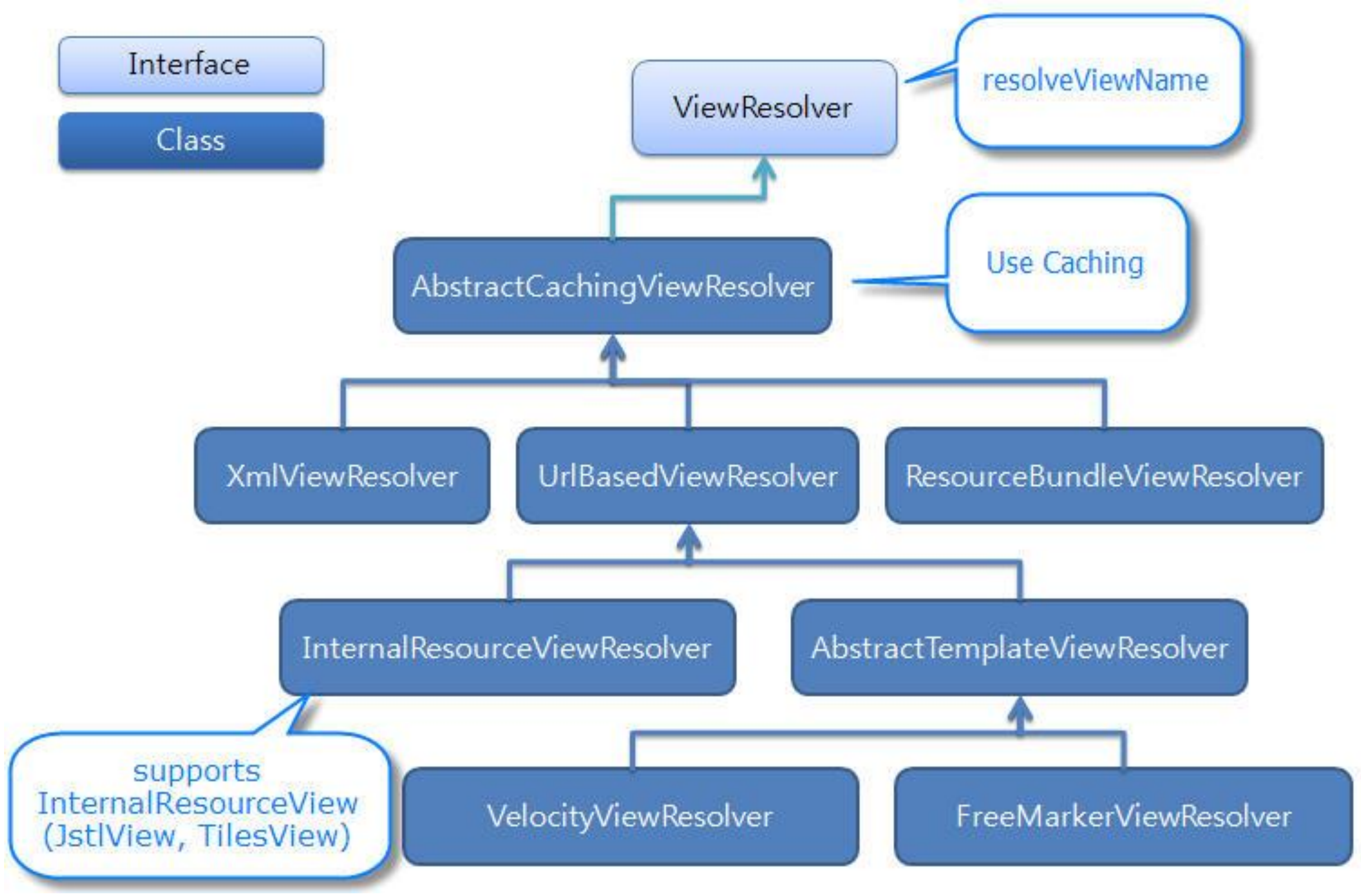
```
@Controller
public class ExceptionController {

    @RequestMapping("/exception")
    public ModelAndView ise() {
        throw new IllegalStateException
                            ("--> This is your exception message");
    }

    @ExceptionHandler(IllegalStateException.class)
    public ModelAndView handle(IllegalStateException ise) {

        ModelAndView mav = new ModelAndView("exception");
        mav.addObject("msg", ise.getMessage());

        return mav;
    }

}
```

View resolvers

Configuration example for **JstlView**:

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

To resolve View using the result of controller method **ViewResolver** is used:


Interface
Class
ViewResolver
resolveViewName
AbstractCachingViewResolver
Use Caching
XmlViewResolver
UrlBasedViewResolver
ResourceBundleViewResolver
InternalResourceViewResolver
AbstractTemplateViewResolver
supports InternalResourceView (JstlView, TilesView)
VelocityViewResolver
FreeMarkerViewResolver

Spring :: MVC :: ViewResolver

- All `ViewResolver` implementations are caching data processed by `view`
- It's possible to change it, by setting `cache` to `false`.
- Also it's possible to remove view from cache programmatically:

```
removeFromCache(String viewName, Locale loc)
```

Use of **`ResourceBundleViewResolver`**

```xml
<bean class="org.springframework.web.servlet.view.
                                ResourceBundleViewResolver">
       <property name="basename" value="spring-views" />
</bean>
```

spring-views.properties (file should be in classpath):

```
WelcomePage.(class)=org.springframework.web.servlet.view.JstlView
WelcomePage.url=/WEB-INF/pages/WelcomePage.jsp
```

Controller:

```java
public class WelcomeController extends AbstractController {

    protected ModelAndView handleRequestInternal(HttpServletRequest req,
                HttpServletResponse resp) throws Exception {
        return new ModelAndView("WelcomePage");
    }
}
```

Use of **`XmlViewResolver`**:

```
<bean class="org.springframework.web.servlet.view.
                                  XmlViewResolver">
  <property name= "location" value="/WEB-INF/spring-views.xml"/>
</bean>
```

/WEB-INF/spring-views.xml:

```
<bean id="WelcomePage"
        class="org.springframework.web.servlet.view.JstlView">
    <property name="url" value="/WEB-INF/pages/WelcomePage.jsp" />
</bean>
```

Controller:

```
public class WelcomeController extends AbstractController {
    protected ModelAndView handleRequestInternal(HttpServletRequest req,
                HttpServletResponse resp) throws Exception {
        return new ModelAndView("WelcomePage");
    }
}
```

Spring:: MVC :: ViewResolver

- All view resolvers implement **`Ordered`** interface. This allows to have several resolvers that work in a certain sequence.
- For this reason **`order`** property should be set. If not specified explicitly, such a resolver will work the last.
- This may be necessary in some cases to redefine certain views if a single resolver doesn't support all **`View`** implementations used.

Localization

Spring :: MVC :: Localization

hello.jsp:

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>

<h1><spring:message code="greeting"/></h1>
```

servlet-config.xml:

```
<bean id="messageSource"
        class="org.springframework.context.support.ResourceBundleMessageSource"
            p:basename="messages"/>
```

messages.properties:

```
greeting=Hello
```

Spring :: MVC :: Localization

servlet-config.xml:

```
<mvc:interceptors>
     <bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"
                               p:paramName="language"/>
</mvc:interceptors>

<bean id="localeResolver"
                      class="org.springframework.web.servlet.i18n.SessionLocaleResolver"
                      p:defaultLocale="en"/>
```

Spring:: MVC :: Localization:: LocaleResolver

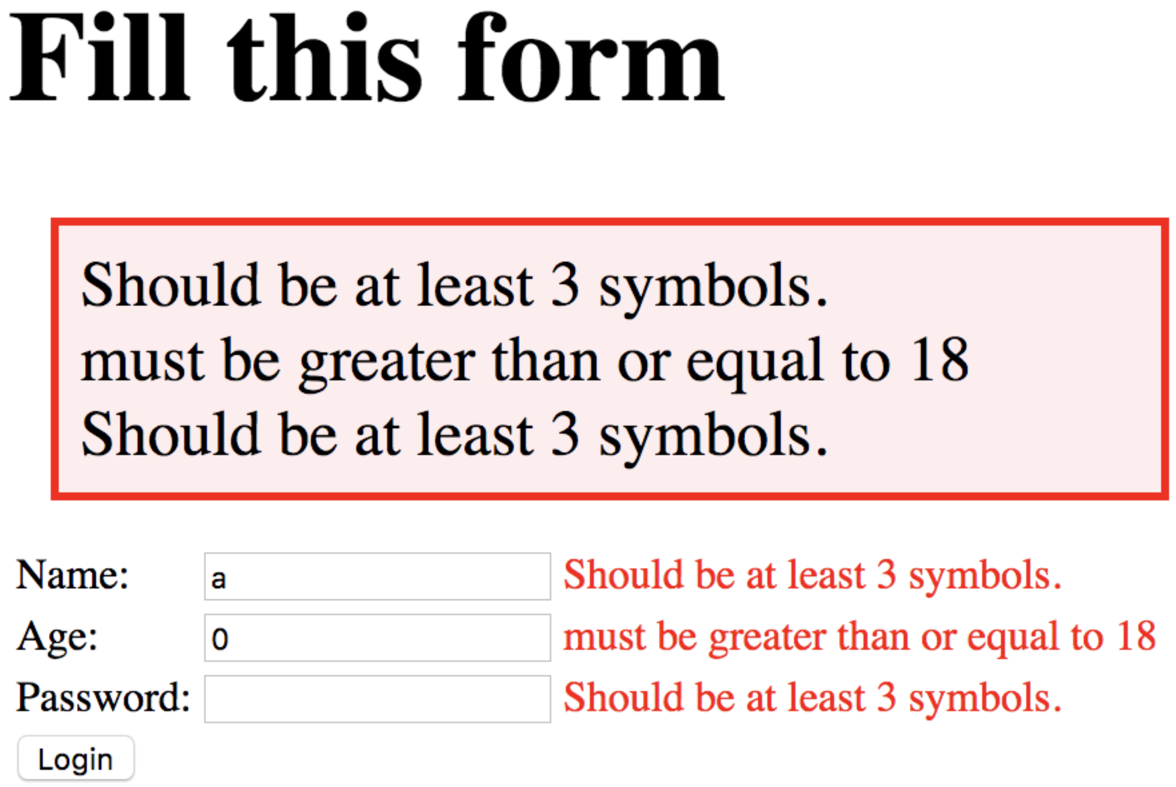
- **AcceptHeaderLocaleResolver** (default resolver): inspects “accept-language” header in the HTTP request;
- **CookieLocaleResolver**: inspect a cookie to retrieve locale;
- **SessionLocaleResolver**: retrieves locales from the sessions;
- **FixedLocaleResolver**: retrieves locale specified when bean is configured;

Ex. 3

Validation

Spring :: MVC :: Validation

Ex. 5



Fill this form

Should be at least 3 symbols.
must be greater than or equal to 18
Should be at least 3 symbols.

Name:	a	Should be at least 3 symbols.
Age:	0	must be greater than or equal to 18
Password:		Should be at least 3 symbols.

Spring :: MVC :: Validation

pom.xml, for Tomcat 8

```
<dependency>
   <groupId>javax.validation</groupId>
   <artifactId>validation-api</artifactId>
   <version>2.0.1.Final</version>
</dependency>

<dependency>
   <groupId>org.hibernate.validator</groupId>
   <artifactId>hibernate-validator</artifactId>
   <version>6.0.14.Final</version>
</dependency>

<dependency>
   <groupId>javax.el</groupId>
   <artifactId>javax.el-api</artifactId>
   <version>3.0.0</version>
   <scope>provided</scope>
</dependency>

<dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>slf4j-api</artifactId>
   <version>1.7.25</version>
</dependency>
```

Spring :: MVC :: Validation

Tag **`<form:errors>`**:

Has attributes:

- **`modelAttribute/commandName`**: property for which binding errors are rendered (* = all errors) ;
- **`cssClass`**: css class bound to block;

- Generates html element **`<span>`** that renders binding errors through <br/> ;
- You can create several of them in different page areas for rendering binding errors of specific properties;

```
<form:form modelAttribute="user">
   <form:errors path="*" cssClass="errorblock" element="div"/>
   …
</form:form>
```

Spring :: MVC :: Validation

```
public class UserForm
{
    @Size(min = 3, max = 20, message = "Should be at least 3 symbols.")
    private String name;

    @NumberFormat(style = NumberFormat.Style.NUMBER)
    @Min(18) @Max(100)
    private int age;

    @Size(min = 3, max = 20, message = "Should be at least 3 symbols.")
    private String password;

}
```

Spring :: MVC :: Validation

```
<form:form modelAttribute="user">
    <form:errors path="*" cssClass="errorblock" element="div"/>
    <table>
        <tr>
            <td>Name:</td>
            <td><form:input path="name"/></td>
            <td><form:errors path="name" cssClass="error"/></td>
        </tr>
        <tr>
            <td>Age:</td>
            <td><form:input path="age"/></td>
            <td><form:errors path="age" cssClass="error"/></td>
        </tr>
        <tr>
            <td>Password:</td>
            <td><form:input path="password"/></td>
            <td><form:errors path="password" cssClass="error"/></td>
        </tr>
        <tr>
            <td colspan="3">
                <input type="submit" value="Login"/>
            </td>
        </tr>
    </table>
</form:form>
```

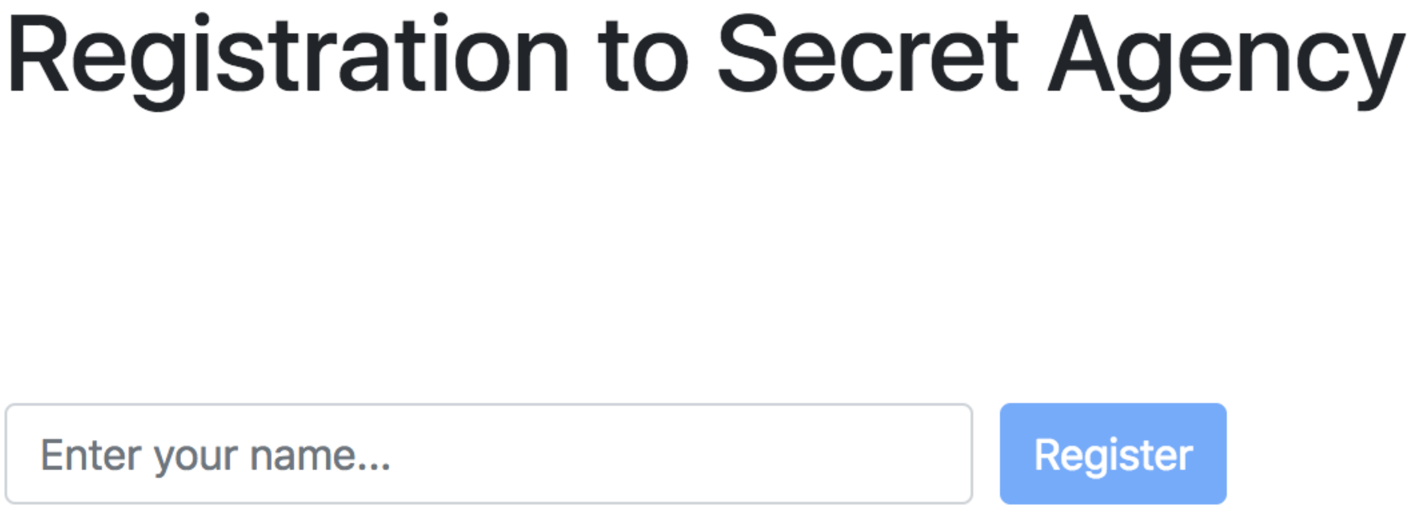
Spring :: MVC :: Validation

```
@Controller
@RequestMapping("login")
public class ValidationController
{
    @RequestMapping(method = RequestMethod.GET)
    public String showForm(Model model)
    {
        UserForm userForm = new UserForm();
        model.addAttribute("user", userForm);

        return "user-form";
    }

    @RequestMapping(method = RequestMethod.POST)
    public String processForm(@Valid @ModelAttribute("user") UserForm userForm,
                              BindingResult result, Model model)
    {
        if (result.hasErrors())
        {
            return "user-form";
        }

        model.addAttribute("user", userForm);

        return "success";
    }
}
```

AJAX

Spring :: MVC :: AJAX

Check name on the server when user typing…


Registration to Secret Agency

Spring :: MVC :: AJAX

pom.xml, for Tomcat 8

```xml
<dependency>
   <groupId>com.fasterxml.jackson.core</groupId>
   <artifactId>jackson-core</artifactId>
   <version>2.9.8</version>
</dependency>

<dependency>
   <groupId>com.fasterxml.jackson.core</groupId>
   <artifactId>jackson-databind</artifactId>
   <version>2.9.8</version>
</dependency>
```

Spring :: MVC :: AJAX

```
@Controller
public class NamesController
{
    @Autowired
    private NamesService namesService;

    @RequestMapping(value = "/names", method = RequestMethod.POST)
    public @ResponseBody void addName(@RequestParam String name)
    {
        namesService.add(name);
    }

    @RequestMapping(value ="/names", method = RequestMethod.GET)
        public @ResponseBody List<String> sayHello ()
        {
                return namesService.getAll();
        }

        @RequestMapping(value = "/names/check", method = RequestMethod.GET)
        public @ResponseBody Boolean checkName(@RequestParam String name)
        {
                return namesService.contains(name);
        }
}
```

Spring :: MVC :: AJAX

```
@RestController
@RequestMapping("/names")
public class NamesRestController
{
    @Autowired
    private NamesService namesService;

        @PostMapping
    public void addName(@RequestParam String name)
    {
        namesService.add(name);
    }

     @GetMapping
        public List<String> sayHello ()
        {
                return namesService.getAll();
        }

        @GetMapping(value = "/check")
        public Boolean checkName(@RequestParam String name)
        {
                return namesService.contains(name);
        }
}
```

Spring :: MVC :: AJAX

Using jQuery to process the request

```
function check()
{
    var name = $('input#name')[0].value;
    if (name.length > 2)
    {
        $.ajax({
            method:"GET",
            url:"/names/check?name=" + name
        })
        .done(function (resp)
        {
                    // what to do with the response
        })
    }
    else
    {
        $('button#register')[0].disabled = true;
    }
}
```