

(/)

Un controlador, servicio y ejemplo DAO con Spring Boot y JSF

Última modificación: 30 de septiembre de 2018

por baeldung (<https://www.baeldung.com/author/baeldung/>)
(<https://www.baeldung.com/author/baeldung/>)



Java EE (<https://www.baeldung.com/category/java-ee/>)

Primavera (<https://www.baeldung.com/category/spring/>) +

JSF (<https://www.baeldung.com/tag/jsf/>)

Acabo de anunciar el nuevo curso *Learn Spring*, centrado en los fundamentos de Spring 5 y Spring Boot 2:

>> VER EL CURSO (/ls-course-start)

Si tiene algunos años de experiencia en **Linux** y está interesado en
comparar 
supuestamente 
(/contr



1. Introducción



JavaServer Faces es un marco de interfaz de usuario basado en componentes del lado del servidor. Originalmente se desarrolló como parte de Java EE. En este tutorial, **investigaremos cómo integrar JSF en una aplicación Spring Boot.**

Como ejemplo, implementaremos una aplicación simple para crear una lista de tareas pendientes.

2. Dependencias de Maven

Tenemos que extender nuestro *pom.xml* para usar tecnologías JSF:

```
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
<!--JSF-->
<dependency>
  <groupId>org.glassfish</groupId>
  <artifactId>javax.faces</artifactId>
  <version>2.3.7</version>
</dependency>
```

El artefacto *javax.faces* (<https://search.maven.org/search?q=a:javax.faces>) contiene las API JSF y las implementaciones también. La información detallada se puede encontrar aquí (<https://javaee.github.io/javaxserverfaces-spec/>).

3. Configuración del servlet JSF

El marco
de la inte
de las de



Comencemos creando una estructura estática en un archivo *index.xhtml* en el directorio *src / main / webapp*:

```
1 <f:view xmlns="http://www.w3c.org/1999/xhtml" (http://www.w3c.org/1999/x
2   xmlns:f="http://java.sun.com/jsf/core" (http://java.sun.com/jsf/core)"
3   xmlns:h="http://java.sun.com/jsf/html" (http://java.sun.com/jsf/html)"
4     <h:head>
5       <meta charset="utf-8"/>
6       <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1"/>
7       <title>T0-D0 application</title>
8     </h:head>
9     <h:body>
10      <div>
11        <p>Welcome in the T0-D0 application!</p>
12        <p style="height:50px">
13          This is a static message rendered from.xhtml.
14        </p>
15      </div>
16    </h:body>
17  </f:view>
```



El contenido estará disponible en `<your-url> /index.jsf`. Sin embargo, recibimos un mensaje de error en el lado del cliente si intentamos llegar al contenido en esta etapa:

```
1 There was an unexpected error (type=Not Found, status=404).
2 No message available
```

No habrá mensaje de error de backend. Aun así, podemos descubrir **que necesitamos un servlet JSF para manejar la solicitud** y la asignación de servlet para que coincida con el controlador.

Dado que estamos en Spring Boot, podemos ampliar fácilmente nuestra clase de aplicación para manejar la configuración requerida:





```

1  @SpringBootApplication
2  public class JsApplication extends SpringBootServletInitializer {
3
4      public static void main(String[] args) {
5          SpringApplication.run(JsApplication.class, args);
6      }
7
8      @Bean
9      public ServletRegistrationBean servletRegistrationBean() {
10         FacesServlet servlet = new FacesServlet();
11         ServletRegistrationBean servletRegistrationBean =
12             new ServletRegistrationBean(servlet, "*.jsf");
13         return servletRegistrationBean;
14     }
15 }

```

Esto se ve genial y bastante razonable, pero desafortunadamente aún no es lo suficientemente bueno. Cuando intentemos abrir `<your-url> /index.jsf` ahora obtendremos otro error:

```

1  java.lang.IllegalStateException: Could not find backup for factory javax

```

Desafortunadamente, necesitamos un *web.xml* junto a la configuración de Java. Vamos a crearlo en *src / webapp / WEB-INF*:

```

1  <servlet>
2      <servlet-name>Faces Servlet</servlet-name>
3      <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
4      <load-on-startup>1</load-on-startup>
5  </servlet>
6  <servlet-mapping>
7      <servlet-name>Faces Servlet</servlet-name>
8      <url-pattern>*.jsf</url-pattern>
9  </servlet-mapping>

```

Ahora, nuestra configuración está lista para funcionar. Abra `<your-url> /index.jsf`



Welcome f

This is a static message rendered from xhtml.

Antes de crear nuestra interfaz de usuario, creemos el backend de la aplicación.



4. Implementando el Patrón DAO

DAO significa objeto de acceso a datos. Por lo general, la clase DAO es responsable de dos conceptos. Encapsula los detalles de la capa de persistencia y proporciona una interfaz CRUD para una sola entidad. Puede encontrar una descripción detallada en este (<https://www.baeldung.com/java-dao-pattern>) tutorial.

Para implementar el patrón DAO, **primero definiremos una interfaz genérica** :

```
1 public interface Dao<T> {  
2  
3     Optional<T> get(int id);  
4     Collection<T> getAll();  
5     int save(T t);  
6     void update(T t);  
7     void delete(T t);  
8 }
```

Ahora creemos nuestra primera y única clase de dominio en esta aplicación de tareas:

```
1 public class Todo {  
2  
3     private int id;  
4     private String message;  
5     private int priority;  
6  
7     // standard getters and setters  
8  
9 }
```

La próxima clase será la implementación de *Dao <Todo>* . La belleza de este patrón es que podemos proporcionar una nueva implementación de esta interfaz en cualquier momento.

En consecuencia,
del código

Para nuestra

memoria :





```
1  @Component
2  public class TodoDao implements Dao<Todo> {
3
4      private List<Todo> todoList = new ArrayList<>();
5
6      @Override
7      public Optional<Todo> get(int id) {
8          return Optional.ofNullable(todoList.get(id));
9      }
10
11     @Override
12     public Collection<Todo> getAll() {
13         return todoList.stream()
14             .filter(Objects::nonNull)
15             .collect(Collectors.collectingAndThen(Collectors.toList(), Co
16     }
17
18     @Override
19     public int save(Todo todo) {
20         todoList.add(todo);
21         int index = todoList.size() - 1;
22         todo.setId(index);
23         return index;
24     }
25
26     @Override
27     public void update(Todo todo) {
28         todoList.set(todo.getId(), todo);
29     }
30
31     @Override
32     public void delete(Todo todo) {
33         todoList.set(todo.getId(), null);
34     }
35 }
```

5. La c



El objetivo

de persistencia. Mientras que la capa de servicio se coloca encima para manejar los requisitos comerciales.

Tenga en cuenta que la interfaz DAO será referenciada desde el servicio:



```
1  @Scope(value = "session")
2  @Component(value = "todoService")
3  public class TodoService {
4
5      @Autowired
6      private Dao<Todo> todoDao;
7      private Todo todo = new Todo();
8
9      public void save() {
10         todoDao.save(todo);
11         todo = new Todo();
12     }
13
14     public Collection<Todo> getAllTodo() {
15         return todoDao.getAll();
16     }
17
18     public int saveTodo(Todo todo) {
19         validate(todo);
20         return todoDao.save(todo);
21     }
22
23     private void validate(Todo todo) {
24         // Details omitted
25     }
26
27     public Todo getTodo() {
28         return todo;
29     }
30 }
```

Aquí, el servicio es un componente con nombre. Usaremos el nombre para hacer referencia al bean del contexto JSF.

Además, esta clase tiene un alcance de sesión que será satisfactorio para esta sencilla aplicación.

Para obtener más información sobre los ámbitos de Spring, eche un vistazo a este (<https://www.baeldung.com/spring-bean-scopes>) tutorial. **Dado que los**

ámbitos i
pena con

Más orien

(<https://www.baeldung.com/spring-custom-scope>) tutorial.





6. El controlador

Al igual que en una aplicación JSP, el controlador manejará la navegación entre las diferentes vistas.

A continuación, implementaremos un controlador minimalista. Navegará desde la página de apertura a la página de lista de tareas:

```
1  @Scope(value = "session")
2  @Component(value = "jsfController")
3  public class JsfController {
4
5      public String loadTodoPage() {
6          checkPermission();
7          return "/todo.xhtml";
8      }
9
10     private void checkPermission() {
11         // Details omitted
12     }
13 }
```

La navegación se basa en el nombre devuelto. Por lo tanto, *loadTodoPage* nos enviará a la página *todo.xhtml* que implementaremos a continuación.

7. Conectando JSF y Spring Beans

Veamos cómo podemos hacer referencia a nuestros componentes desde el contexto JSF. Primero, *ampliaremos* el *index.xhtml*:




```
1 <f:view
2   xmlns="http://www.w3c.org/1999/xhtml" (http://www.w3c.org/1999/xhtml)"
3   xmlns:f="http://java.sun.com/jsf/core" (http://java.sun.com/jsf/core)"
4   xmlns:h="http://java.sun.com/jsf/html" (http://java.sun.com/jsf/html)"
5   <h:head>
6     // same code as before
7   </h:head>
8   <h:body>
9     <div>
10      // same code as before
11      <h:form>
12        <h:commandButton value="Load To-do page!" action="#{jsfCon
13      </h:form>
14    </div>
15  </h:body>
16 </f:view>
```



Aquí presentamos un *commandButton* dentro de un elemento de formulario. **Esto es importante ya que cada elemento *UICommand* (por ejemplo, *commandButton*) debe colocarse dentro de un elemento *UIForm* (por ejemplo, formulario).**

En esta etapa, podemos iniciar nuestra aplicación y examinar `<your-url>/index.jsf`:

Welcome in the TO-DO application!

This is a static message rendered from xhtml.

(/wp-content/uploads/2018/09/2018-

Load To-do page!

09-21-13_55_06-TO-DO-application.png)

Desafortunadamente, obtendremos un error cuando hagamos clic en el botón:

```
1 The
2 java
3 /inc
4 Targ
```





El mensaje indica claramente el problema: el *jsfController* resolvió *nulo*. El componente correspondiente no se creó o al menos es invisible desde el contexto JSF.

En esta situación, esto último es cierto.

Necesitamos conectar el contexto Spring con el contexto *JSF* dentro de la aplicación *web* / *WEB-INF* / *faces-config.xml*:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <faces-config xmlns="http://xmlns.jcp.org/xml/ns/javaee (http://xmlns.jc
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance (http://www.w3.or
4   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig
5   version="2.2">
6     <application>
7       <el-resolver>org.springframework.web.jsf.el.SpringBeanFacesELRes
8     </application>
9 </faces-config>
```

¡Ahora que nuestro controlador está listo para funcionar, necesitaremos *todo.xhtml*!

8. Interactuando con un servicio de JSF

Nuestra página *todo.xhtml* tendrá dos propósitos. Primero, mostrará todos los elementos de tareas pendientes.

En segundo lugar, ofrezca la oportunidad de agregar nuevos elementos a la lista.

Para eso, el componente UI interactuará directamente con el servicio declarado anteriormente:



```
1 <f:view xmlns="http://www.w3c.org/1999/xhtml" (http://www.w3c.org/1999/x
2   xmlns:f="http://java.sun.com/jsf/core" (http://java.sun.com/jsf/core)"
3   xmlns:h="http://java.sun.com/jsf/html" (http://java.sun.com/jsf/html)"
4   <h:head>
5       <meta charset="utf-8"/>
6       <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1"/>
7       <title>T0-D0 application</title>
8   </h:head>
9   <h:body>
10      <div>
11          <div>
12              List of T0-D0 items
13          </div>
14          <h:dataTable value="#{todoService.allTodo}" var="item">
15              <h:column>
16                  <f:facet name="header"> Message</f:facet>
17                  #{item.message}
18              </h:column>
19              <h:column>
20                  <f:facet name="header"> Priority</f:facet>
21                  #{item.priority}
22              </h:column>
23          </h:dataTable>
24      </div>
25      <div>
26          <div>
27              Add new to-do item:
28          </div>
29          <h:form>
30              <h:outputLabel for="message" value="Message: "/>
31              <h:inputText id="message" value="#{todoService.todo.mes
32              <h:outputLabel for="priority" value="Priority: "/>
33              <h:inputText id="priority" value="#{todoService.todo.pr
34              <h:commandButton value="Save" action="#{todoService.sav
35          </h:form>
36      </div>
37  </h:body>
38 </f:view>
```



Los dos p
elemento

En el primero, usamos un elemento `dataTable` para representar todos los valores de `todoService.AllTodo`.

El segundo `div` contiene un formulario donde podemos modificar el estado del objeto `Todo` en el Servicio `Todo`.



Usamos el elemento *inputText* para aceptar la entrada del usuario, donde la segunda entrada se convierte automáticamente en un *int*. Con el *commandButton*, el usuario puede conservar (en la memoria ahora) el objeto *Todo* con *todoService.save*.

9. Conclusión

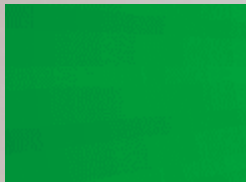
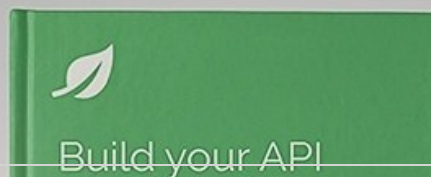
El marco JSF se puede integrar en el marco Spring. Debe elegir qué marco gestionará los beans. En este tutorial, utilizamos el marco Spring.

Sin embargo, el modelo de alcance es un poco diferente al marco JSF. Por lo tanto, podría considerar definir ámbitos personalizados en el contexto Spring.

Como siempre, el código está disponible en GitHub (<https://github.com/eugenp/tutorials/tree/master/spring-boot-mvc>).

Acabo de anunciar el nuevo curso *Learn Spring*, centrado en los fundamentos de Spring 5 y Spring Boot 2:

>> VER EL CURSO (/ls-course-end)





¿Estás aprendiendo a construir tu API con Spring ?

Enter your email address

>> Obtenga el libro electrónico

CATEGORÍAS

PRIMAVERA ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/](https://www.baeldung.com/category/spring/))

DESCANSO ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/](https://www.baeldung.com/category/rest/))

JAVA ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/](https://www.baeldung.com/category/java/))

SEGURIDAD ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/](https://www.baeldung.com/category/security-2/))

PERSISTENCIA ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/](https://www.baeldung.com/category/persistence/))

JACKSON ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/](https://www.baeldung.com/category/json/jackson/))

HTTP DE

KOTLIN (



SERIE

TUTORIAL DE JAVA "VOLVER A LO BÁSICO" (/JAVA-TUTORIAL)



[JACKSON JSON TUTORIAL \(/JACKSON\)](#)

[HTTPCLIENT 4 TUTORIAL \(/HTTPCLIENT-GUIDE\)](#)

[RESTO CON SPRING TUTORIAL \(/REST-WITH-SPRING-SERIES\)](#)

[TUTORIAL SPRING PERSISTENCE \(/PERSISTENCE-WITH-SPRING-SERIES\)](#)

[SEGURIDAD CON PRIMAVERA \(/SECURITY-SPRING\)](#)

ACERCA DE

[SOBRE BAELDUNG \(/ABOUT\)](#)

[LOS CURSOS \(HTTPS://COURSES.BAELDUNG.COM\)](https://courses.baeldung.com)

[TRABAJO DE CONSULTORÍA \(/CONSULTING\)](#)

[META BAELDUNG \(HTTP://META.BAELDUNG.COM/\)](http://meta.baeldung.com/)

[EL ARCHIVO COMPLETO \(/FULL_ARCHIVE\)](#)

[ESCRIBIR PARA BAELDUNG \(/CONTRIBUTION-GUIDELINES\)](#)

[EDITORES \(/EDITORS\)](#)

[NUESTROS COMPAÑEROS \(/PARTNERS\)](#)

[ANUNCIE EN BAELDUNG \(/ADVERTISE\)](#)

[TÉRMINOS DE SERVICIO \(/TERMS-OF-SERVICE\)](#)

[POLÍTICA DE PRIVACIDAD \(/PRIVACY-POLICY\)](#)

[INFORMACIÓN DE LA COMPAÑÍA \(/BAELDUNG-COMPANY-INFO\)](#)

[CONTACTO \(/CONTACT\)](#)

