# JSF

Echemos un vistazo al próximo capítulo, que es Desarrollo de aplicaciones web usando JavaServer Faces. En esta sesión, debemos describir el ciclo de vida y la arquitectura de JSF, comprender la sintaxis de JSF, observar las bibliotecas de componentes de JSF y las formas en que podemos aplicar la validación y los convertidores a los componentes de la interfaz de usuario, utilizando las plantillas de la interfaz de usuario, describiendo la navegación de JSF y manejando la localización. y producir mensajes, usar lenguaje de expresión en páginas JavaServer Faces, usar CDI Beans y usar algunas características nuevas, como soporte para componentes AJAX.

Están sucediendo muchas cosas en esta sesión. Hay una especie de gran entorno de desarrollo. Bueno, vayamos paso a paso.

Primero, analicemos los conceptos de JSF. JavaServer Faces es un marco de componentes del lado del servidor diseñado para desarrollar aplicaciones web. Pero es muy diferente, aunque pueda sonar similar. Pero, de hecho, es muy diferente de JavaServer Pages.

Recuerde, JSP, JavaServer Pages, se convirtieron en servlets. Bueno, las páginas JSF no lo son. Son interpretados por un servlet preescrito, que se llama FacesServlet. Asignas ese servlet a la URL. Bueno, eso es usar un archivo web.xml, así que asigna el servlet a la URL, caras/\* por ejemplo. La idea es que esa estrella sirva como comodín, para cualquier página que desee que FacesServlet maneje, interprete.

Entonces, cuando invoca una página JSF real, en realidad está invocando FacesServlet y hace que interprete una página en particular. Y un FacesServlet es un controlador preescrito que maneja el ciclo de vida de un documento JSF. Así que está codificado en ese control exactamente lo que va a hacer. Y necesitamos aprender sobre ese ciclo de vida y cómo escribimos nuestro código para conectarlo a ese ciclo para que funcione correctamente allí. Además, FacesServlet controlará las navegaciones entre estas páginas. Entonces, las navegaciones del ciclo de vida, bueno, esencialmente actuarán como un controlador.

La página en sí, la página JSF, también conocida como Facelet, parece ser solo un documento XHTML. Por lo tanto, contiene marcas HTML directas si lo necesita. Pero también contiene JSF UIComponents. Estos JSF UIComponents son marcas dentro de una página que tienen un código Java correspondiente que generará algún tipo de salida para el cliente, producirá el diseño, producirá el contenido visual que vería un cliente.

De hecho, los controladores de UIComponent lo hacen con la ayuda de un código adicional, que se conoce como RenderKit, cuyo trabajo es tener los componentes del controlador para producir y usar nuestros diseños visibles. Por lo tanto, podemos concluir que aparentemente la página JSF en realidad no contiene directamente ningún código Java. Es vista pura, en un sentido de controlador de vista de modelo. Así que básicamente se trata de crear el diseño. Su trabajo es crear una presentación visual de un contenido para el cliente.

FacesServlet que actuará como controlador para él. Por lo tanto, no necesita escribir ese controlador, como lo hizo cuando trabajábamos con otros marcos de desarrollo web, como JSP. Estábamos escribiendo controladores usando servlets. Estamos escribiendo controladores usando filtros web. Esta vez el controlador ya está disponible para usted de la caja. Por lo tanto, puede concentrarse en la producción del diseño dentro de una página, en lugar de solo codificar en un controlador en sí.

Además, si desea llamar a su código Java personalizado, bueno, invoque su lógica comercial, no podemos hacerlo colocando dicho código en la página JSF. No está permitido poner código Java directo en él. Pero puede usar el lenguaje de expresión y hacer referencia al código necesario en su modelo, en CDI Beans, a través del lenguaje de expresión, que coloca estas expresiones en la página. Entonces, la lógica comercial estará en sus CDI Beans detrás de una página.

Entonces, de hecho, como puede ver, el marco está diseñado para representar la segregación del controlador de vista de modelo puro dentro del entorno Java EE. Primero, echemos un vistazo a cómo se registra FacesServlet. Es bastante sencillo. Hay un fragmento de un archivo web.xml aquí. Y están estos. Ese es su FacesServlet, asignado a alguna URL, caras/\* por ejemplo.

Y luego, por supuesto, si desea llamar a una página en particular, como la página XHTML de ese producto, simplemente pasa eso como ese argumento, si lo desea, a FacesServlet. Así que tenga en cuenta que no está llamando a demostraciones/productos. Estás llamando demostraciones/caras/producto. Así que quieres que FacesServlet maneje esa página en particular.

¿Por qué estamos usando web.xml? Porque no estamos escribiendo FacesServlet. Ya está escrito. Así que no tenemos forma de anotarlo. Bueno, yo diría que no puedes ponerle la notación, porque ya está ahí. Y la única forma en que puede cambiar la forma en que se asigna a la URL es simplemente escribiéndolo en un archivo web.xml.

Pero, por supuesto, web.xml tiene prioridad sobre cualquier anotación que pudiera haber en ese servlet en primer lugar. Y no importa Podríamos mapearlo a cualquier URL que nos guste. Podrían ser caras/\*. Pero, francamente, no tiene por qué ser así. Supongo que es una URL tan buena como cualquier otra.

Próximo. Ahora, la configuración de Faces se puede almacenar en un archivo faces-config.xml. Este es un archivo de configuración utilizado por el propio FacesServlet. Técnicamente hablando, es opcional. Realmente no tienes que crearlo. Pero hay ciertas cosas que no podrá lograr sin ese archivo. Por ejemplo, para registrar paquetes de recursos y locales para la localización de la aplicación, necesitaría faces-cofig para eso.

En cuanto a otras características, puede lograrlas a través de otros medios, no necesariamente mediante el uso de caras-config, tal vez solo anotaciones. Así que no es obligatorio, pero probablemente quieras usarlo. Verá en los primeros días, cuando usábamos Managed Beans, que es, recuerde, un marco predecesor, antes de que tuviéramos los CDI Beans, Managed Beans era algo que usaba exclusivamente el tiempo de ejecución de JSF. Y se registraron en faces-config.xml.

Bueno, en estos días, eso ya no es importante. Quiero decir, todavía podrías hacerlo. Pero puede simplemente anotar CDI Beans y usarlos en su lugar. Entonces, por esa razón en particular, se podría emitir faces-config.xml. Ya no tienes que hacer esa configuración. Pero puede haber otras razones por las que aún desee usar face-config. Bueno, en realidad nos encontraremos con algunos de ellos más adelante en este capítulo.

Ahora, ¿qué hay en esa página de Facelet? Bueno, como dije, es un documento XHTML. Por lo tanto, podría contener marcas HTML directas. Eso sí, las marcas XHTML, por lo que debe hacer el cierre adecuado de todas las etiquetas. No puede dejar la etiqueta sin estas barras inclinadas. Así que siga las reglas XML, básicamente. Así que no es exactamente un documento HTML. Tiene que ajustarse a las reglas XML. Pero, en términos generales, puede colocar algunas marcas HTML allí, pero no solo marcas HTML.

También puede poner allí los UIComponents que son los JSF UIComponents, provenientes de JSF Component Libraries. Y en este caso particular, estos son los ejemplos de dichos componentes. De hecho, provienen de la biblioteca JSF. No son marcas HTML.

Los controladores de UIComponent correspondientes, junto con el kit de representación, finalmente producen algunas marcas HTML que entregamos al cliente. Pero la página en sí aquí no contiene HTML directo. Contiene estos elementos UIComponent. De hecho, están renderizando algo de HTML.

Pero también tienen funcionalidades de manejo del lado del servidor. Y aquí hay algunos ejemplos de eso. Como puede ver en esta esquina de la pantalla, tenemos un CDI Bean. Bueno, por razones de conveniencia, le damos un nombre: pm, como en Product Manager. Solicitaremos código, por ejemplo.

Y tenemos algunas operaciones en ese Bean, como la operación "buscar", y luego quizás algunas propiedades, métodos getter/setter. Si desea acceder a propiedades y objetos, puede usar estas expresiones con notaciones de puntos. Aquí pm.product-- presumiblemente el producto tiene id, dot id. Entonces, la notación punto, punto, punto que le permite acceder a las propiedades de sus objetos y llamar a los métodos, así como encontrar la invocación del método, contra el oyente de acción.

Así que esta es la idea. Los componentes en la página no solo representan la salida, entregan la salida al cliente, sino que también están asociados con expresiones que le permiten relacionar el componente y sus funcionalidades con el código de soporte en los CDI Beans. Así es como desencadena una lógica de negocios en la aplicación, la ejecución de esa lógica y obtiene y establece valores de la aplicación en la página y fuera de la página con expresiones allí.

Tenga en cuenta que las expresiones no usan dólares. Usan ese signo hash, un prefijo ligeramente diferente aquí. Y eso es porque llamamos a estas expresiones diferidas. No se ejecutan dentro de un ciclo de vida de servlet, sino dentro de un ciclo de vida JSF. Estas expresiones se interpretan esencialmente por etapas de un ciclo de vida de FacesServlet.

Entonces por eso es diferido. El ciclo de vida del servlet es un ciclo de vida del propio FacesServlet. Pero luego un FacesServlet crea un ciclo de vida de manejo de una página. Y en un contexto de ese ciclo de vida, estas expresiones funcionarían.

Ahora, hablando del ciclo de vida, ahí está. Así que sigamos este ciclo y expliquemos lo que está pasando. Primero, el cliente llama a la página. Hermoso. FacesServlet recibe esa llamada, averigua a qué página quiere llamar el cliente. Y FacesServlet verifica si se accede a la página por primera vez, o si el cliente ya accedió a esta página. Por lo tanto, podría ser como una solicitud inicial, o puede ser una devolución de llamada posterior.

Supongamos, por un momento, que esta es una solicitud inicial. Hablaremos de la devolución de datos en un momento. Pero empecemos con la solicitud inicial. Entonces, el cliente inicialmente llama a la página. Así que FacesServlet necesita crear eso, lo llamaremos fase de creación de vista.

Si se trata de una devolución de datos, esa misma fase se conocerá como vista de restauración. Porque la página ya estará allí, ya será accedida por el cliente, como una llamada posterior. Pero en una primera llamada, FacesServlet analiza la página JSF. Y de alguna manera lo prepara para ser manejado por esa persona que llama. Así que la página debe estar lista.

Si se trata de una llamada inicial, es probable que el cliente no envíe ningún valor. Es solo llamar a una página. No hay parámetros especiales. No hay necesidad de aplicarlos. No necesitan validar nada, solo invocar la página tal como está sin ningún tipo de parametrización adicional. En cuyo caso, todas las fases, de la dos a la cinco, se saltan esencialmente. Ellos no están haciendo eso. En la convocatoria inicial, después de la primera fase, una solicitud pasa directamente a la fase seis.

Así que FacesServlet básicamente dice, esa es la primera llamada a la página, ponla en la memoria. Ahora, obtenga valores de CDI Bean, recupere valores de CDI Beans, cualquiera que sea el valor al que apuntan las expresiones en una página. Entonces obtenemos estos valores. Aplicamos estos valores a los elementos de la interfaz de usuario que están en la página y representamos la salida. Deje que los UIComponents produzcan cualquier salida que produzcan.

Así que esa es la llamada inicial. El usuario obtiene la página. Si los elementos de una página a la que nos referimos están modelados en CDI Beans a través de expresiones, el usuario verá los valores actuales de los elementos del modelo que se muestran en esa página. Sin embargo, lo que sucede a continuación es que el usuario puede desear cambiar estos valores. Así que está recibiendo del servidor, en la llamada inicial, una página que muestra algunos campos y una estrella. Y puede haber algunos valores en estos campos. Y su usuario comienza a escribir en estos campos, modificando datos.

Entonces, el usuario modifica los datos, envía ese formulario al servidor nuevamente. Eso es muy importante. JavaServer Faces procesa los formularios enviados de forma recursiva. Entonces, si tiene una página A que generó algún resultado, algún formulario que está viendo, sus valores escritos, lo envía, lo envía a la página A, siempre.

Entonces FacesServlet obtendrá esta solicitud de devolución de datos. Será una llamada a la misma página, todo bien. Pero no será una llamada inicial. Será eso más adelante, posterior convocatoria. Y esta vez la llamada contendrá nuevos valores que un usuario ha ingresado en el formulario.

Así que ahora tienen que ser procesados. Cómo se procesan: fase dos del ciclo de vida. Tome los valores que envía el usuario a través de la solicitud y aplíquelos a los UIComponents que componen esa página. Entonces, por ejemplo, si hubo un UIComponent que generó una salida, que era un campo de entrada, y el usuario escribió algo en ese campo, ese valor que el usuario escribió, ese parámetro de protocolo HTTP que está obteniendo, FacesServlet va a extráigalo y aplíquelo inmediatamente al UIComponent correspondiente que ha generado ese campo en particular en primer lugar.

Entonces, existe una especie de relación entre lo que está viendo en HTML y lo que sucede con el UIComponent en un nivel de servidor. Estos elementos se relacionan directamente. Si es necesario, en esa etapa, se podrían aplicar conversiones de tipos de datos. Bueno, supongo que si es algo más complejo que un simple texto o número, es posible que tengas que convertir valores.

Y luego comienzas a procesar validaciones. Aparentemente, JavaServer Faces conoce el marco conocido como Bean Validation y ciertamente puede procesar las validaciones en esa etapa. Entonces, los valores se aplican a la página y se validan. Bien. Estamos en la fase tres: la validación tiene éxito. Hablaremos de lo que sucede cuando falla en un momento.

Ahora vamos a la fase cuatro. En la fase cuatro, toma valores que no enfrenta un servidor, aquí toma valores de UIComponents y luego los aplica a CDI Beans. Así que ahora estamos insertando los valores de la página en el modelo. Así es como los elementos en una página, las expresiones que usamos para referirnos a los CDI Beans, los valores viajarán hacia los CDI Beans.

Luego ejecutaremos los métodos de lógica empresarial que se ajusten a estos Beans, ejecutaremos cualquier método de acción lógica. Así que hemos aplicado valores que un usuario ha transmitido. Y le pedimos al modelo que realice acciones.

Una vez que finaliza la ejecución de los métodos de acción, finaliza esta fase cinco, luego avanzamos a la fase seis, en la cual, nuevamente, tomamos valores de CDI Beans y los aplicamos nuevamente a nuestros UIComponents en la página. Porque pueden haber cambiado. El modelo puede haber modificado cosas. Por lo tanto, debemos volver a aplicar a la página estos valores de modelo recién adquiridos y luego dejar que los UIComponents produzcan cualquier salida que necesiten producir.

Una cosa más. En caso de cualquier error durante cualquier fase, digamos, por ejemplo, falló en las validaciones o conversiones de tipos de datos, en su modelo lanzó la excepción, o lo que sea que suceda, tiene un problema, este ciclo de vida se interrumpirá. E independientemente de en qué fase interrumpa, salta directamente a la fase seis, Respuesta de procesamiento. Así que no importa lo que salió mal, el ciclo se interrumpe y usted avanza a Componentes de la interfaz de usuario, Respuesta de representación.

Un par de cosas que quiero que recuerdes de toda esta descripción que podrían ser útiles. En primer lugar, no estás escribiendo ese código. Ya está escrito. Es lo que FacesServlet hace por ti. Todo lo que hace, escribe expresiones en las páginas para crear un vínculo entre el elemento de la interfaz de usuario en una página y un CDI Bean. Y es el trabajo de FacesServlet determinar cuándo tomar el valor de una página y aplicarlo al Bean, y cuándo tomar el valor del Bean y aplicarlo a la página, precisamente en las fases cuatro y seis. Así es como están sincronizados.

Echemos un vistazo a la siguiente parte de esta presentación. ¿Qué son estos UIComponents con los que puedes construir páginas? Seguramente, puedes simplemente poner HTML directo. Sí. Pero entonces, ¿qué son estas bibliotecas dinámicas de UIComponent? Hay bastantes de ellos, de verdad.

Entonces puede ver algunas bibliotecas de caras que producen código HTML que procesa alguna lógica en una página. En realidad, podría usar las mismas bibliotecas que ha usado para JavaServer Pages, JSTL Core Library o JSTL Function Library. Todavía funcionarán para los JSF al igual que para los JSP. Entonces eso es lo mismo.

Esta es una lista de bibliotecas que viene de fábrica con el servidor Java EE 7. Así que este es un conjunto estándar de bibliotecas. Además de eso, hay muchas bibliotecas de terceros disponibles que pueden producir capacidades de diseño o manejo mucho más sofisticadas además de estas. Y seguramente, incluso puede escribir sus propias bibliotecas si lo desea, con sus propios componentes y elementos de interfaz de usuario, a partir de los cuales puede componer páginas. Por lo tanto, es una selección bastante amplia de estos elementos de la interfaz de usuario de los que podría beneficiarse.

Durante este capítulo, vamos a echar un vistazo a algunos de estos elementos. Y habrá una demostración un poco más tarde que cubrirá el uso de bibliotecas de terceros. Echemos un vistazo a un ejemplo de la Biblioteca UIComponent en la Biblioteca HTML JSF.

En esta página en particular, estamos viendo la forma en que Facelet, con sus componentes UI que provienen de esa biblioteca de componentes UI, bueno, estos otros componentes que son parte de la página, producen HTML. Entonces, en este caso, estamos viendo un formulario con algunos campos de entrada, con algunos botones. Y los UIComponents que observas crean elementos HTML reales. Echemos un vistazo más de cerca a cómo están creando exactamente estos elementos.

Por ejemplo, tenga en cuenta este atributo de acción dentro de un formulario. No se encuentra en ninguna parte en una página JSF, ¿verdad? Este marcado en una página JSF no lo contiene. ¿Por qué sería ese el caso?

Bueno, verá, cuando el elemento de la interfaz de usuario en una página JSF produce un HTML, que muestra algún campo de texto de entrada en una pantalla, el valor que el usuario escribirá en ese campo deberá enviarse de nuevo a ese mismo componente de la interfaz de usuario. Debe volver a aplicarse al mismo UIComponent para que se procese correctamente. Y eso significa que el formulario no tiene otro lugar al que enviarse sino recursivamente a la misma página. Y es por eso que el atributo de acción siempre apunta a la misma página que produjo el formulario.

El atributo de acción HTML siempre apuntará allí. Es por eso que no está presente en el marcado del formulario H dentro de la propia página JSF. Porque es un poco excesivo. No hay ningún otro lugar al que pueda conducir.

La otra cosa interesante que me gustaría señalar es que la página aún podría contener elementos HTML simples aquí, es decir, simplemente se transmiten desde la página de Facelet al HTML generado tal cual. Y ahora hay otra cosa interesante. Y esa es la forma en que manejamos la identidad de los campos, formularios y elementos dentro de una página. Así que echemos un vistazo más de cerca a estas identidades.

A un formulario se le asigna la identidad "editar". A un campo se le da la identidad de "id" oa otro campo "find". Mira el HTML generado. Es una forma de identificar el campo como perteneciente a ese formulario. Es parte de esa forma. El atributo de identidad es opcional. En realidad, no tienes que usarlo en absoluto. Pero si desea, más tarde, poner algún código JavaScript en la página que acceda dinámicamente a los elementos de la página HTML, ¿cómo sabría qué elemento es qué?

La identificación se generará automáticamente para usted si no la ingresa. Pero puede que no sea particularmente conveniente. Entonces, si desea confiar en sus identidades particulares de campos para obtener los beneficios del procesamiento dinámico de componentes dentro de una página, en el nivel del cliente, tal vez con algo de JavaScript, eso es precisamente lo que debe hacer; Ingrese estas identificaciones solo para su propia conveniencia.

Y lo siguiente interesante que veremos es la biblioteca HTML Passthrough. El concepto HTML Passthrough nos permite combinar marcas HTML con capacidades JSF UIComponent. Y hay varias formas en que se puede hacer.

Primero, existe esta biblioteca llamada Passthrough. Bueno, aquí le asignamos el prefijo p. Y mira cómo usamos eso. Tenemos un elemento de interfaz de usuario llamado inputText. No es un marcado HTML. Es el JSF UIComponent.

Lo que queremos es que el elemento HTML que se creará en función de ese componente de la interfaz de usuario contenga un atributo adicional llamado "tipo". Desafortunadamente, el componente de interfaz de usuario JSF llamado inputText realmente no tiene el atributo de tipo. Un elemento HTML que produce puede tenerlo. Pero en sí mismo no lo tiene. Entonces agregará ese atributo de transferencia. Y aparecerá en la salida generada de la página allí para usted.

Un enfoque ligeramente diferente para lograr lo mismo con una biblioteca central en la que agrega un atributo de transferencia no como un atributo, sino como un elemento del componente de la interfaz de usuario. Pero el resultado es el mismo. Entonces, en este caso, se agregó un atributo adicional. Quiero decir, no tiene que ser el tipo de atributo. Podría ser cualquier atributo HTML, lo que sea. No importa. Pero así es como se agrega un poco de HTML a la salida producida por el componente de interfaz de usuario JSF.

Y hay un ejemplo opuesto. Con una biblioteca JSF, puede construir lo que claramente es solo un elemento HTML simple, pero agregar a ese elemento HTML bits de las capacidades de manejo de componentes de la interfaz de usuario de JSF. Eso significa que involucraría ese HTML con el procesamiento dinámico de eventos y expresiones del ciclo de vida JSF en un nivel de servidor.

Y como resultado, usted produce el HTML combinado, que es parcialmente HTML sin formato, parcialmente la salida producida por la interfaz de usuario, los componentes de la interfaz de usuario JSF. Por cierto, esta es una característica nueva en JSF, en la última versión del tiempo de ejecución de JSF, que agrega esa flexibilidad a la página. En última instancia, eso significa que puede hacer que una página sea más liviana. Porque no tiene que procesar todo, desde los componentes de la interfaz de usuario, o no tiene que construir HTML voluminoso. Puede simplemente recortar fragmentos de HTML donde lo necesite, o recortar fragmentos de las funcionalidades del componente JSF donde lo necesite. Es muy flexible. Ese es otro ejemplo de las funcionalidades de la biblioteca de componentes de la interfaz de usuario.

Ahora echemos un vistazo a los validadores y convertidores. Los convertidores se aplican a los valores que envía desde el cliente al servidor. Y luego se puede aplicar la validación. Entonces, son las fases dos y tres del ciclo de vida de JSF cuando se realiza el proceso de aplicar valores de solicitud y luego los validamos.

Por qué los convertidores son importantes: independientemente del tipo de campo que cree en formato HTML, desafortunadamente, el protocolo HTTP transmite todo como texto. Puede decir que un campo en particular y un cliente HTML es un número. Puedes decir que es una cita. ¿A quien le importa? El protocolo HTTP, que es ese medio entre el cliente y un servidor, de todos modos, transmitirá todo, es solo una cadena. Porque no hay otros tipos de datos disponibles allí. Es solo una limitación.

Por lo tanto, cuando los valores llegan al servidor, solo llegan al servidor como texto. Y si desea que los valores se presenten como números, fechas o cualquier otra cosa, cualquier otro tipo, es posible que deba proporcionar una conversión. La conversión se puede escribir como una conversión personalizada. Para que pueda escribir su propio convertidor personalizado. Pero luego, algunos convertidores están disponibles listos para usar, como el convertidor de fecha, hora, convertidor de número. Simplemente están ahí fuera de la caja.

Así que es bastante sencillo. Haces referencia a un convertidor. Si es uno de los convertidores precodificados, como estos convertidores de fecha, hora o número, tiene su propio elemento para atenderlo. Los convertidores pueden estar asociados con parametrización adicional, como, por ejemplo, fecha, patrón o, no sé, código de moneda, o el hecho de que no es un número cualquiera, sino un número de moneda.

Además, si desea utilizar conversiones personalizadas (por ejemplo, escribe su propio convertidor), puede hacer referencia a su propio convertidor utilizando un convertidorId. Bueno, aquí estoy usando un convertidor precodificado, que ya existe en el paquete javax.faces. ¿Pero adivina que? Podrías haber escrito uno personalizado.

Como parte de su ejercicio, escribirá un convertidor personalizado. Así que eso aún está por venir. Lo harás en una práctica. Y así es básicamente como puede hacer referencia al convertidor desde la página.

Ahora, en cuanto a los validadores, probablemente ya hayas adivinado su naturaleza. Y te recuerdan algo: API de validación de Bean. Y eso es lo que básicamente representan. Así que tienes estos elementos que puedes poner en una página para asociar campos de entrada con validaciones: expresiones regulares, longitudes, rangos de valores. Y también hay una forma de registrar un validador personalizado. Entonces, al igual que con el convertidor, puede hacer convertidores personalizados y puede hacer validadores personalizados si es necesario.

Para mejorar la reutilización del código dentro de su aplicación, puede usar plantillas de JavaServer Faces. Ahora, la plantilla JSF le permite describir una estructura de su página con marcadores de posición en los que desea insertar contenido dinámico futuro en esa página. Cada marcador de posición tendrá un nombre.

Y alrededor de estos marcadores de posición, puedes hacer lo que quieras. La plantilla puede contener cualquier código normal, solo texto HTML repetitivo, lo que quieras. Y luego, si inserta algo en esa parte de una plantilla, define una página JavaServer Faces como una composición. Y para cada región nombrada, define qué es lo que desea colocar en esa región. Podría ser solo texto. Podría ser la inclusión dinámica de otra página. ¿Qué hay sobre eso?

Ahí vas. Entonces es una composición de diferentes componentes, elementos, que reciclan la misma estructura de plantilla. Si tiene un diseño estándar que varias páginas tendrán que seguir, use una plantilla. En realidad, reciclará su código de una manera bastante eficiente.

NAVIGATION

Describamos ahora la navegación de JavaServer Faces. Está controlado por el servlet Faces. Ahora recuerda el truco que estábamos haciendo antes. Sabemos que cuando enviamos algún tipo de formulario en JSF, regresa a la misma página para ser manejado por la misma página.

Entonces, ¿cómo se pasa de una página a otra? Bueno, puedes controlarlo con una propiedad llamada acción, que está asociada con algún tipo de componente de comando. Podría ser enlace de comando. Podría ser un botón de comando, cualquiera de estos. En realidad, un ejemplo aquí muestra un enlace. Otro ejemplo muestra el botón. La única diferencia entre un enlace y un botón es cómo se presentan visualmente al usuario como una URL, una URL en la que se puede hacer clic o un botón en la página. Aparte de eso, tienen las mismas funcionalidades. Así que espere que funcionen de la misma manera.

Digamos, por ejemplo, que tiene algún tipo de página JavaServer Faces llamada lista. Echemos un vistazo a este primero. Y esta última página contiene un botón llamado buscar. Y cuando haces clic en ese botón, quieres ver una respuesta del servidor, quieres ir, si quieres, una página de edición. Así que hay otra página llamada editar. Cuando haga clic en ese botón, la página de lista enviará el formulario recursivamente a la misma página. Entonces la solicitud volverá a la página de lista.

Irás por el ciclo de vida de seis fases. Pero en la fase seis, justo después de que termine de invocar el modelo, aplicó valores, invocó el modelo, todo está bien, en la fase seis, en lugar de representar el resultado de la página de lista, generará el resultado de cualquier página en la que se realice la acción. El atributo dice que debe representar la salida. Entonces, esencialmente, es solo un nombre de página.

Usted [INAUDIBLE] dice, maneje la entrada producida por el usuario dentro del ciclo de vida de una página de lista. Pero luego renderice la salida desde la página de edición. Entonces, el usuario verá la salida producida por la página de edición como resultado. Para el usuario, los campos son como si el usuario simplemente navegara a otra página porque lo que ellos... eso es lo que un usuario ve en la pantalla.

Bueno, en realidad, la URL en un navegador todavía dirá la palabra "lista" porque ahí es donde se envió la solicitud. Cuando está realizando acciones de reenvío o inclusión del lado del servidor, el navegador no sabe que otra página está mostrando el resultado porque el navegador no está realizando la solicitud explícita hacia la página de edición. El navegador envió una solicitud a la página de lista. Es solo que un servidor decidió producir la salida desde otra página. Entonces, aquí hay una falta de coincidencia entre lo que ve en una URL en un navegador y el resultado que observa. Esto se conoce como navegación implícita JSF. Así que simplemente coloque el nombre de la página en la propiedad de acción y obtendrá la representación de la salida de esa página.

Puede controlar a qué página va mediante programación. Y ese es un segundo ejemplo. La propiedad de acción aquí en la página de edición apunta a algún tipo de bean. Y ese bean contiene un método. Un nombre de método es irrelevante. Se llama navegar. Pero puede que lo hayas llamado como quieras. Lo importante es que ese método debe devolver una cadena. Y esa cadena podría ser el nombre de otra página o ahora, si devuelve un valor nulo, entonces representa la salida desde la misma página.

Entonces, la página de edición se envía a sí misma, maneja las primeras cinco fases de un ciclo de vida. Se aplica toda la entrada del usuario. Se invoca el modelo. Entonces está llamando a ese método de navegación. El método de navegación es decidir lo que sea: permanecer en la misma página, representar el resultado de la página de edición, en cuyo caso devuelve un valor nulo, o volver a la página de la lista. Entonces navega a la lista simplemente devolviendo el nombre de la página allí. Eso es todo. Es bastante sencillo. Eso le permite determinar dinámicamente el paradero de la página, bueno, navega; es mejor decir, por supuesto, qué página produce la salida para el usuario. Así es como funciona la navegación JSF.

Derecha. De forma predeterminada, puede utilizar la navegación implícita en JavaServer Faces simplemente enviando el nombre de la página como propiedad de la acción. Sin embargo, si decidió que no quiere usar el nombre de la página, puede crear un alias para ella en el archivo faces-config.xml. Básicamente, lo que dice el ejemplo aquí es esto.

Digamos que estamos viendo la página agregada. Si la página agregada contiene un componente de comando (botón de comando, enlace de comando) que tiene una propiedad de acción con una lista de valores, en lugar de ir a una página de lista, irá a la página llamada Lista de productos.

Probablemente porque no quería escribir Lista de productos, pensó que era demasiado larga y quería un alias más corto. Entonces puede configurar un alias más corto dentro de faces-config.xml. Si la página agregada envía la vista como el valor de la propiedad de acción, va a Vista del producto. Sin embargo, también puedes hacer esta cosa interesante. Comodín de SQL. Caso de navegación.

Entonces, tiene cualquier página en la aplicación, cualquiera, que envíe la propiedad de acción de un componente de comando llamado inicio. Y eso irá a Home.xhtml. Bueno, en realidad, lo que en este caso en particular es completamente excesivo, porque Home.xhtml ya se llama Home.

Entonces el alias y la página coinciden. Pero supongo que el objetivo de este ejemplo es mostrarte realmente el uso del comodín, que puedes definir la regla de navegación para todas las páginas o para una página específica como en el primer ejemplo. ¿OK?

Así que ahora otra característica nueva e interesante del tiempo de ejecución de Java ServerFaces, la capacidad de crear Faces Flows. Un flujo de caras describe una progresión de varias páginas. Podría estar asociado con su propio archivo de configuración. Puede describir Faces Flow en Faces config.xml. Esa es una opción. Puede describir Faces Flow usando una anotación llamada definición de flujo. O puede describir Faces Flow en un archivo de configuración separado, que es más o menos como una réplica de la configuración de Faces, pero solo para ese flujo.

Ahora el flujo le permite describir un grupo de páginas con un punto de inicio específico designado. Debe iniciar el Flujo donde el Flujo dice que debe iniciarse. Así que digamos que es la página de pago, ese es su punto de entrada. Tienes que empezar por ahí.

Y luego avanza a través de la navegación a través de ese Flujo durante varias páginas hasta llegar a algún tipo de conclusión de Flujo. El final de ese flujo. Así que puedes describir diferentes resultados, cómo podría terminar el Flujo. El flujo tiene su propio ámbito de memoria, que es muy similar a la anotación FlowScoped de la sesión que podría usarse para describir beans CDI que se colocan en el flujo.

Similar a la Sesión, pero es más corta que una Sesión. Una sesión [INAUDIBLE] es esencialmente mientras el usuario permanece activo. Para todas las páginas, para todas las URL a las que accede ese usuario, mientras ese navegador esté activo, seguirá siendo la misma sesión.

Pero un flujo solo dura mientras el usuario avanza a través de una colección particular de páginas. Una vez que el usuario abandona esa colección de páginas, eso es todo. Ese es el final del ámbito de flujo. El flujo se puede tratar lógicamente un poco como una función. Puede pasar parámetros de entrada al flujo y los resultados del flujo pueden devolver valores. No tienes que hacerlo, pero ciertamente puedes.

Entonces, en este caso particular, imaginemos que me gustaría revisar mi pedido. Así que estoy haciendo esta finalización de compra aquí. He pedido algunos productos y quiero completar la compra. Necesito ingresar información sobre los detalles de mi carrito de compras y continuar con el pago.

Y podría tener éxito o fracasar en ese proceso. Así que este podría ser el posible resultado de ese Flujo. Probablemente tendría algún tipo de parámetro de entrada que identifique qué orden estoy revisando. Supongo que si estoy produciendo algún pago, necesito una referencia de pago para confirmar que todo está bien.

Ahí vas. Le permite crear, en cierto modo, una especie de transacción comercial de una cantidad de páginas donde ese Flujo, donde está limitando el alcance de los beans CDI solo a ese Flujo. Así que no los está arrastrando a lo largo de la memoria durante una sesión completa. Simplemente lo ayuda a organizar mejor su aplicación para encapsular fragmentos de funcionalidades en una especie de áreas de la aplicación que están delimitadas por los límites de flujo.

Ahora hay una serie de páginas aquí, a partir de esta página y habrá un par de páginas más, discutir los atributos específicos de los componentes de la interfaz de usuario. Entonces, cuando está construyendo un componente de interfaz de usuario en una página JSF, debe comprender algunas diferencias sutiles entre usar este o aquel atributo del componente de interfaz de usuario.

Y comenzaremos aquí con dos atributos. Se llaman Action y ActionListener. Bueno, es algo obvio que es fácil confundirlos debido a la similitud del nombre. Dichos atributos están presentes en cualquier componente de comando. Botón de comando, enlace de comando. Entonces tienen atributos ActionListener y Action.

ActionListener se puede usar para vincular una expresión para que apunte a un método en un bean CDI que tiene una firma muy particular. Es un método vacío y acepta el evento Acción como argumento. Y está ahí para procesar la lógica empresarial o la aplicación para reaccionar a ese clic en el botón.

El otro método, que es el método de acción, apunta a un bean CDI una operación a través de la expresión. No acepta el evento Acción como argumento. No no. Pero devuelve una cadena. Y verá que la cadena es un nombre de caso de navegación. Es donde quieres ir. A qué página quieres ir. Desde qué página desea renderizar la salida.

Así que es claramente una naturaleza diferente del método. Supongo que puedes usar ambos, ¿verdad? Ah, y por supuesto, debes recordar el orden en que se invocaron. Se activa el primer método ActionListener y luego se invoca el método Action.

Y la razón por la cual es bastante obvia. El método de acción activará potencialmente la representación de una salida de otra página. Entonces, eso sería lo último que hará esta página antes de pasar el control a alguna otra página JSF que potencialmente pueda generar la salida. Muy bien, esa es la diferencia entre Action y ActionListener.

A continuación, la diferencia entre los atributos Value y Binding. Una vez más, hay una gran similitud entre lo que hacen el valor y los atributos vinculantes. Y, sin embargo, hay una distinción importante. Por lo tanto, los atributos de valor y enlace podrían asociarse con varios componentes de la interfaz de usuario.

Y un atributo de valor se usa para hacer referencia a algún tipo de valor que debería tener ese componente de la interfaz de usuario, como esa ID de producto. Bien, presumiblemente hay algún tipo de frijol CDI. Ese bean CDI tiene un componente de producto. Me gustaría obtener ese producto y luego acceder a su ID. Ahí tienes Así que es solo una especie de, no sé, valor entero, tal vez.

¿Qué pasa con la unión? El enlace le permite acceder no al valor de un elemento, sino al componente de la interfaz de usuario en sí. Eso se asigna al componente de la interfaz de usuario, llamado campo PID en este caso. Representa ese marcado. Se llama texto de entrada. ¿Derecha? Y hay un controlador para ese marcado de texto de entrada, que es una clase de Java llamada Texto de entrada HTML. Esta es una implementación física detrás de esto.

Entonces, si desea alterar dinámicamente las cosas en el nivel del elemento de la interfaz de usuario en sí, no solo manipular con su valor, sino manipular con el elemento de la interfaz de usuario en sí, bueno, puede hacerlo. Puedes hacerlo en este tipo de bean. ¿Derecha? Sin embargo, hay algo de lo que necesito hablar con respecto a ese bean en particular

Verá, incluso cambié el nombre aquí a BackingBean. ¿Qué es el frijol de respaldo? Cuando crea tales beans que hacen referencia al enlace del componente, que representa un elemento de interfaz de usuario específico en una página, ¿se puede usar este bean en otra página? En realidad no, porque la otra página no tiene ese elemento de interfaz de usuario, ¿verdad?

Entonces, normalmente, el bean CDI se puede inyectar en cualquier lugar que desee. No tiene afiliación a una página específica. Pero si comienza a hacer referencia a los enlaces, esencialmente está creando un enlace codificado entre ese elemento de la interfaz de usuario en una página específica y la presentación de ese elemento en Java dentro del bean que lo admite, lo que hace que el ser no sea realmente utilizable para otras páginas.

Eso puede llevarnos a usar un ámbito diferente para ese bean en lugar del ámbito de solicitud, lo que potencialmente significa que ese bean podría ser utilizado por otra página. Tal vez le gustaría usar un alcance diferente llamado View Scoped. Porque el alcance de Vista vincula el bean a una página específica. Por lo tanto, no estaría en una situación en la que accidentalmente está en otra página que también intenta hacer referencia a ese bean y descubre que no tiene el elemento necesario dentro de la página a la que se puede vincular. Entonces, tal vez en esa circunstancia particular, le gustaría usar un bean de alcance de vista en lugar de alcance de solicitud, por ejemplo. Solo porque probablemente no reutilices ese componente.

Personalmente, creo que preferiría no usar el enfoque de la propiedad Binding la mayor parte del tiempo, porque si quiero modificar dinámicamente lo que sucede en ese componente, ¿qué me impide modificarlo a través de expresiones asociadas con las propiedades del componente?

Entonces, ¿por qué codificaría programáticamente en un bean una lógica que altera el componente de la interfaz de usuario si, en una página, solo puedo poner una propiedad que apunte a la expresión y cualquier bean puede establecer ese valor de expresión? Así que es bastante sencillo, ¿no es así, sí? Probablemente será más fácil que codificar el bean que es específico para una página determinada. Pero tienes ambos enfoques, así que elijas lo que elijas.

Otro atributo interesante que me gustaría comentar es el atributo llamado inmediato. Es un nombre muy confuso. Derecha. Lo que realmente significa. Depende de dónde se aplique el atributo inmediato. Se puede aplicar a los componentes de entrada. Se puede aplicar a componentes de mando. O a ambos.

Echemos un vistazo al atributo inmediato que se aplica a algún componente de entrada. Así que imagina que hay una forma. Hay un campo y tiene atributos inmediatos establecidos en verdadero. Significa que dicho campo va a ser validado antes que otros. Entonces, en lugar de posponer la validación del campo a la fase tres, la validación ocurrirá de inmediato (eso probablemente explicaría el nombre del campo) en la fase dos, justo cuando se aplican los valores solicitados.

Entonces, en otras palabras, si tiene un formulario y varios campos en un formulario, y un campo es inmediato, ese campo se aplicará primero a su contraparte del componente de la interfaz de usuario antes que a todos los demás, luego se validará y solo si es válido. ¿Se aplicarían otros campos en ese formulario a otros componentes de la interfaz de usuario? De lo contrario, el ciclo de vida de la solicitud se interrumpirá y se le bajará a la respuesta de representación de la fase seis. Ese es el atributo inmediato asociado con un componente de entrada.

Ahora, ¿qué pasa si el atributo inmediato está asociado solo con algún componente de comando? Así que no está en ningún campo. No, solo en un botón, ¿sí? ESTÁ BIEN. Entonces, en este caso, qué sucederá, hace clic en ese botón y su formulario, sus otros campos en ese formulario, no se enviarán en absoluto. Básicamente, simplemente envía el comando al servidor diciendo en qué botón hizo clic, en lugar de procesar todo el formulario con todos los campos.

Imagine que está viendo un registro en particular en su tabla, en su formulario JSF, y hay un botón Eliminar allí. Así que haces clic en ese botón Eliminar. ¿Quieres procesar todos los campos y aplicar sus valores? ¿No porque? Porque estás a punto de borrarlo de todos modos. ¿A quien le importa? Todo lo que necesita hacer es decir eliminar ese registro. Por lo tanto, no necesita aplicar todos los campos. Basta con que marques ese botón con la propiedad Inmediato.

O tal vez, hay otro ejemplo, desea navegar desde esta página a otra página sin enviar el formulario. Me gusta el botón Cancelar. No quiero enviar el formulario, solo quiero ir a otra página. Entonces, nuevamente, puede marcar ese botón como Inmediato y evitará que se procesen los campos del formulario.

Finalmente, puede usar una combinación del atributo inmediato en un campo (campo de entrada) y en un botón. Si usa ambos, esencialmente envía parcialmente un formulario. Estás diciendo que no envíes el formulario completo. Cuando haga clic en ese botón, simplemente envíe ese campo. Entonces, en este caso particular, imagine que tiene algún tipo de campo llamado Descuento. Y tienen el botón Aplicar descuento. Y que presentará el valor del descuento, pero no presentará ID Nombre y Precio.

Tal vez tenga más botones en ese formulario que envían otros valores. Ahora, por qué no, es posible que no todos tengan necesariamente una propiedad inmediata. Entonces, ese botón en particular lo hace, por lo que es para el envío parcial del formulario. Como puede ver, toda la API está diseñada para establecer este vínculo entre el elemento de la interfaz de usuario que ve en la página HTML y un componente de la interfaz de usuario con el que lo maneja en un servidor.

Es por eso que llamamos a este enfoque basado en componentes, porque establecemos esa relación entre cosas específicas dentro del nivel del cliente y los componentes de manejo correspondientes dentro del nivel del servidor. Y podemos controlar cuáles se aplican y cómo se procesan con atributos como Inmediato.

Si desea manejar mediante programación las fases de un ciclo de vida de un servlet JSF o, no sé, ejecutar expresiones mediante programación como en este ejemplo, puede usar para eso el objeto llamado FacesContext. Entonces, el ejemplo aquí obtiene el intérprete del lenguaje de expresiones, la fábrica de expresiones, construye la expresión.

Básicamente, este es el tipo de expresión que podría haber colocado en una página en primer lugar. Pero desea ejecutarlo desde algún bean CDI. Ahí vas. Simplemente puede ejecutarlo desde el bean CDI con ese objeto de contexto.

O podría haber otras razones por las que usa FacesContext. Por ejemplo, si desea saltar la fase del ciclo de vida mediante programación. Algún tipo de condición sobre la cual desea mantener ciertas fases o, no sé, progresar hasta la representación de la respuesta. O producir algunos mensajes, que en realidad vamos a ver. Habrá más ejemplos de uso de FacesContext porque también estamos a punto de producir mensajes.

Pero antes de hablar sobre cómo producimos mensajes en JSF, mencionemos el hecho de que es posible que desee realizar la localización de su aplicación. Ahora bien, hay una parte de este tema que en realidad no es específica de JSF. Si desea localizar cualquier aplicación Java, debe crear versiones de los paquetes de recursos que tenga.

Entonces crea sus archivos de propiedades, recuerde que tenía un archivo de propiedades de validación. Bueno, independientemente de los archivos de propiedades que tenga, puede crear versiones de estos archivos para diferentes países e idiomas. El idioma puede ser solo una abreviatura de dos letras como esa. Pero también puede haber variaciones entre países. Por ejemplo, en\_gb versus en\_us. ¿Sí? Así que eso es inglés, de acuerdo, pero con un sabor ligeramente diferente.

Entonces país, variaciones de idioma. Y traduces los paquetes. Básicamente, proporciona traducciones para todas las propiedades del paquete a cualquier otro idioma que desee. Derecha. Así que esa es una especie de enfoque general para la localización. Específicamente en JSF, además de hacer referencia a paquetes y traducir paquetes, también debe registrar todas las configuraciones regionales admitidas en el archivo Faces.config. Tienes que registrarlos.

Si creó una traducción de su paquete de recursos a algún idioma que no se menciona en Faces.config, no se utilizará. Debe asegurarse de que el idioma del paquete y las extensiones de país estén enumerados explícitamente en Faces.config.

Luego le das a tu paquete un buen alias, mensaje, lo que sea. Y ahí tienes Usa eso en una página en tus expresiones. ¿Derecha? Así que eso se menciona en este mensaje de precio del artículo. Luego, si el usuario elige usar un idioma, eso mostrará ese aviso en un idioma. Elige usar otro, bueno, ese será un idioma diferente entonces.

La configuración regional, que es la forma de elegir qué idioma le gustaría usar al usuario, también se puede establecer como una propiedad programática. Por lo tanto, la configuración regional de la propiedad en un elemento de vista determina el idioma en el que se mostrará la página. Así que puedes cambiar de idioma dinámicamente si lo deseas.

ESTÁ BIEN. Ahora, mostrando cualquier otro mensaje. ¿Cómo haces eso? Pueden ser mensajes de validación. Puede ser, bueno, cualquier otra información o mensaje de error, lo que necesite mostrar. El ejemplo en esta página habla de validación, pero como digo, también podría ser cualquier otro mensaje.

Entonces, digamos que tiene algunas propiedades de validación del paquete de recursos. Tienes tu nombre valor pares de mensajes. Y luego, si desea mostrar mensajes asociados con la página, use el elemento Mensajes. Entonces, por ejemplo, si algún artículo en esa página falla en la validación, como el precio del artículo, por ejemplo, mostrará el mensaje de error con las propiedades del mensaje de validación donde colocó el marcado de Mensajes en una página.

Alternativamente, puede crear un marcado de mensaje que muestre errores, no para todos los elementos en un formulario, sino solo para un elemento específico. Por lo tanto, existe este campo Para que relaciona el marcado del mensaje con un ID de elemento en particular. Es como un mensaje de error solo para ese campo. Además, los mensajes se pueden asociar con CSS, hoja de estilo en cascada, cláusulas visuales.

Así que esto podría ser mensajes de error, mensajes fatales, mensajes de advertencia, mensajes de información. Tu elección. Y puede crear diferentes estilos visuales con los que se debe mostrar cada uno de estos diferentes tipos de mensajes. Por supuesto, en este caso particular, estamos viendo el ejemplo de los mensajes de validación, por lo que probablemente serán solo mensajes de error. Pero, ¿qué pasa con cualquier otro mensaje arbitrario que le gustaría mostrar?

Todavía usa los mensajes, o marcas de mensajes, en un formulario. Los elementos de la interfaz de usuario siguen siendo los mismos. Pero, ¿cómo se genera realmente ese mensaje arbitrario desde el bean CDI? Bueno, esa es la respuesta. Objeto FacesContext, ¿verdad? Recordaremos este. Consígueme la instancia actual de FacesContext. Cree un nuevo mensaje de Caras. Nivel de gravedad: información, advertencia, error, lo que necesite. Describir mensaje corto y largo. Mensaje corto, mensaje largo, mensaje detallado. El detalle podría ser nulo o podría haber algún texto allí. Lo que quieras, de verdad. Y luego agregue el mensaje al FacesContext. Ahí tienes Eso es todo.

Así que eso es lo que hace un bean CDI. En la página en la que tiene sus mensajes o marcas de mensajes, dónde los coloca en una página, se mostrará el mensaje que acaba de crear, con cualquier estilo con el que esté asociado el componente de la interfaz de usuario, según el tipo de mensaje. que has producido aquí. Info, aviso, error, lo que necesites.

ESTÁ BIEN. Estamos casi al final de esta presentación, pero hay otra cosa que me gustaría mencionar. Este es un ejemplo de cómo registra un bean administrado. El predecesor antes de los beans CDI. Antes de los beans CDI, teníamos que registrar los beans administrados en un archivo faces-config.xml.

Todavía puedes hacerlo. Quiero decir, incluso con frijoles CDI todavía puedes hacerlo. Probablemente sea un poco excesivo. Verá, si quiere usar beans CDI, todo lo que necesita hacer es marcar qué alcance tienen y eso es todo. ¿Sí? Es solo una anotación simple, como el alcance de la solicitud, por ejemplo. Mientras que en faces-config tienes que deletrearlo como el registro. Eventualmente es el mismo alcance y todo, seguro. Pero es solo más trabajo escribir todo ese xml, supongo.

Si desea que un bean haga referencia a otro, puede hacerlo con la propiedad administrada. Eso le permite crear una inyección. Pero nuevamente, en un mundo de beans CDI, ¿por qué no usar la anotación @inject? Eso simplemente inyectará otro bean y eso es todo. No necesitarás un faces-config.xml para ello.

Pero por si acaso, si encuentra una aplicación de beans administrados, tal vez una versión anterior del tiempo de ejecución de JSF antes de que se introdujeran los beans CDI, entonces puede encontrar este estilo de codificación. Además, mire esta nota al pie. Debe tener en cuenta el hecho de que, según el ámbito en el que se encuentre un bean administrado, un bean CDI, puede o no hacer referencia al bean que está en otro ámbito.

Por ejemplo, imagine que está en el bean que tiene RequestScoped. ¿Cuántas sesiones podría haber para una solicitud? Es solo una sesión. Es una solicitud de un usuario en particular. Entonces, desde la perspectiva de la solicitud, el alcance de la sesión no es ambiguo. Es obvio. Es esa sesión. Es posible que no haya otra sesión para una solicitud determinada. Entonces la inyección de sesión funcionará. Podría inyectar un bean de sesión en un bean de solicitud. No hay problema.

¿Qué hay del revés? ah Bueno, tenemos un problema. Porque para un bean con ámbito de sesión, el ámbito de la solicitud definitivamente no es uno, sino muchos. Durante esa sesión hay tantas solicitudes diferentes, por lo que nadie sabe cuál de ellas está hablando. Entonces eso definitivamente crearía una confusión.

Así que tenga cuidado con lo que inyecta dónde. El alcance tiene que ser inequívocamente resoluble. Entonces, desde la perspectiva, por ejemplo, del alcance de una solicitud, puede inyectar cualquier otra cosa que desee. Pero al revés, probablemente no.

Por último, otro punto interesante sobre el tiempo de ejecución de JSF es una novedad. Cuenta con la capacidad de agregar AJAX (código API XML y JavaScript asíncrono) y procesar dinámicamente los componentes de la interfaz de usuario dentro de un nivel de cliente a través de eventos de JavaScript. En este caso particular, echemos un vistazo. Tenemos una inclusión de un componente de JavaScript en la página. Esta es una biblioteca JavaScript JSF estándar, pero también puede incluir bibliotecas personalizadas.

Así que es su archivo JavaScript. Lo incluyes en la página. Y luego, mira lo que está pasando aquí. Es realmente emocionante. Tiene este evento AJAX, que asocia con un campo de entrada. El evento en este caso particular como un evento de cambio. Así que puede haber todo tipo de eventos AJAX diferentes. El evento de cambio es solo un ejemplo.

Y eso significa que este evento se activará cuando un usuario escriba en el campo. En el momento en que el usuario escribe. Y luego dices, ejecuta el artículo de precio. El artículo de precio es ese elemento, ¿no es así? ¿Qué es una ejecución? Envialo. En realidad, solo está creando algo que no es diferente de la funcionalidad del atributo Inmediato. Está tomando un elemento particular en una página y lo está enviando al servidor.

Ahora, ¿qué está pasando? No estás enviando toda la página. Solo estás enviando el único elemento fuera de él. Por lo tanto, la página no se actualizará porque no la está procesando desde cero. Ah, pero luego dices, ¿sabes qué? Renderice uno de los elementos. Entonces, cuando envía el nuevo valor del precio, aunque no está recuperando toda la página, porque no lo solicitó, está obteniendo un nuevo valor para el artículo con descuento.

Supongamos que el servidor lo vuelve a calcular. Hay algo de lógica en cómo debería funcionar el descuento. Entonces, el valor del descuento cambiará cuando cambie el precio. Pero el resto de la página no se recargará. Esa es la idea. Simplemente procese parcialmente los componentes de la interfaz de usuario en particular en una página en lugar de toda la página.

Ojo, esto no siempre funciona como crees. Porque, por ejemplo, si tiene la intención de realizar una navegación, entonces esta actualización parcial de la página está completamente fuera de la ventana. No tiene sentido actualizar parcialmente un elemento en la página si está a punto de renderizar toda la página desde cero, ¿verdad?

Así también, si necesita representar muchos elementos, por lo que no es solo un elemento que necesita actualizar, sino muchos elementos que necesita actualizar, o enviar más de un elemento, enviar varias cosas, tal vez valga la pena simplemente enviar un formulario completo en lugar de procesarlo campo por campo. Depende de lo que será más rápido, no estoy muy seguro. En una proporción de contenido que es el mismo en la página frente al contenido que en realidad cambia dinámicamente.

Así que ten cuidado con eso. Esa proporción tiene que ser significativa. Realmente debe asegurarse de que la página sea lo suficientemente grande y que el contenido que desea procesar dinámicamente sea lo suficientemente pequeño para que este algoritmo en particular sea eficiente. Pero, en términos generales, se pueden procesar diferentes tipos de eventos AJAX. Puede procesar tipos de eventos precodificados con JSF JSLibrary.

Puede crear sus propios controladores de JavaScript, invocar servicios [INAUDIBLES], invocar sockets, cualquier cosa que haya estado haciendo hasta ahora en JavaScript en este curso, sería perfectamente aplicable al tiempo de ejecución de JSF.

Para obtener más información sobre AJAX y sus funcionalidades, consulte el curso de capacitación Desarrollo de aplicaciones web JavaScript y HTML5, que de hecho he mencionado en bastantes ocasiones.

Anteriormente en este capítulo, mencioné que además de las bibliotecas de componentes estándar, JavaServer Faces viene con muchos marcos extendidos de terceros y bibliotecas adicionales que quizás desee usar. Por nombrar solo algunos, PrimeFaces, Oracle ADF Faces, Apache MyFaces, IceFaces, OmniFaces. Hay muchos de ellos. Puede decidir que desea utilizar estas bibliotecas adicionales junto con, o en lugar del conjunto de bibliotecas JSF principal.

Y en realidad hay una breve demostración que me gustaría mostrarles para demostrar el uso de estas bibliotecas alternativas. Esta demostración se ve mejor después de terminar el ejercicio de este capítulo. Porque lo que sucederá es que en este ejercicio para este capítulo en particular, se le indicará que cree una aplicación JSF utilizando bibliotecas JSF estándar. Y luego, esa demostración adicional mía muestra la misma aplicación pero compilada con PrimeFaces en su lugar. Es una elección aleatoria. Podría haber elegido una biblioteca diferente, pero bueno, ¿por qué no PrimeFaces? Es solo un ejemplo de lo que las bibliotecas alternativas podrían hacer por usted.

Bueno, supongo que eso es todo por esta sesión en particular. Así que hemos aprendido bastante sobre las formas en que funcionan las aplicaciones JSF. Lo más importante es que debe recordar el ciclo de vida JSF de seis fases, porque comprenderlo es absolutamente crucial. Todo lo demás que está sucediendo en una aplicación JSF podría repararse a través de su código, pero ese ciclo de vida está codificado en el servlet de Faces. Usted encaja en su código alrededor de las seis fases.

Así que entender claramente lo que están haciendo es importante. Ahora su código se ocupa de una página con solo producción de diseño, con componentes de interfaz de usuario HTML y JSF. En beans CDI con cualquier otro evento de lógica de negocios que le gustaría manejar, cualquier otra lógica de negocios. Puede controlar la navegación por las páginas, las plantillas, las validaciones, las conversiones, las localizaciones, producir mensajes, usar expresiones para hacer referencia a su código y, posiblemente, también agregar procesamiento dinámico de JavaScript a través de AJAX al nivel de cliente de la aplicación.

Con respecto al ejercicio, lo que está a punto de hacer en la práctica de este capítulo es crear una versión JSF de la aplicación que busque productos, muestre una lista de productos y maneje la actualización del producto. Básicamente es exactamente lo mismo que ha estado haciendo con servlets y JSP, pero esta vez lo haremos con JSF. Bueno, démosle una oportunidad.