**Знакомство с JavaServer Faces (JSF) 2 (с)**

**Вступление**

JSF 2 является Java стандартом ([JSR 314](http://www.jcp.org/en/jsr/detail?id=314)), реализацией которого занимаются [Oracle Mojarra](http://javaserverfaces.java.net/) и [Apache MyFaces](http://myfaces.apache.org/index.html). JSF 2 предоставляет разработчику большое количество новых возможностей, таких как стандартный механизм поддержки AJAX, проверку входных данных с помощью аннотаций, библиотеку facelets, как стандартный движок для шаблонизации страниц, множество готовых компонентов. Все это можно свободно использовать при построении **web** приложений любой сложности.

JSF входит в поставку любого JEE-сервера приложений, но не в ходит в Tomcat, так как *Tomcat* только лишь *контейнер Web-приложений*, но никак не JEE-сервер.

Из наиболее известных third-party JSF компонентов, хотелось бы отметить [PrimeFaces](http://www.primefaces.org/showcase-labs/ui/home.jsf), [ICEfaces](http://component-showcase.icefaces.org/component-showcase/showcase.iface), [RichFaces](http://livedemo.exadel.com/richfaces-demo/index.jsp), [PrettyFaces](http://ocpsoft.com/prettyfaces/), [OpenFaces](http://openfaces.org/demo/overview/homepage.jsf). Хотя JSF 2 по своей сути является компонентно-ориентированной платформой, однако с её помощью можно писать и MVC ориентированные приложения.

При этом модель (Model) – это коллекция классов, которые служат для сохранения, изменения и получения данных; представление (View) – это jsf страница в xhtml формате; поведение (Controller) – это managed bean, который интерпретирует данные, введённые пользователем, и информирует модель о необходимости соответствующей реакции. JSF также хорошо интегрируется со [Spring](http://www.springsource.com/developer/spring).

Последняя версия JSF 2.2.6 ([4 марта](http://ru.wikipedia.org/wiki/4_%D0%BC%D0%B0%D1%80%D1%82%D0%B0) [2014 года](http://ru.wikipedia.org/wiki/2014_%D0%B3%D0%BE%D0%B4))

**Сравнение JSF 2 с MVC (servlet + jsp)**

Преимущества JSF 2 по сравнению с MVC (servlet + jsp):

1. JSF обеспечивает программиста хорошим набором API, что даёт возможность создавать свои компоненты. Также можно воспользоваться уже готовыми компонентами, такими как: PrimeFaces, ICEfaces, RichFaces, PrettyFaces, OpenFaces и др.
2. С помощью JSF легко указать, с помощью какого Java кода будет обрабатываться форма.
3. JSF расширяет/добавляет возможность валидировать/конвертировать значения.
4. JSF дает вам возможность без использования JavaScript, а также AJAX ориентированных фреймворков, таких, как Jquery, EXT-JS, Prototype и др. писать AJAX ориентированное приложение, используя простые в использовании компоненты JSF.
5. JSF имеет встроенную возможность валидировать поля форм, конвертировать string поля ко многим другим типам данных. Если валидация/конвертация прошла с ошибкой, форма может быть автоматически показана с ошибкой вместе с предыдущими установленными значениями. Валидация/конвертация проходит на серверной стороне.
6. Хотя у JSP есть jsp:include тег для вставки контента, однако JSF имеет полноценную поддержку системы шаблонизации страниц, которая позволяет создавать макет страницы, либо ее часть для дальнейшего использования.
7. Централизованная файловая конфигурации JSF, что позволяет вместо жесткого кодирования данных в программе Java выносить ее в конфигурационный файл. Что позволяет без редактирования Java кода и перекомпиляции изменить эти данные.

Недостатки JSF 2 по сравнению с MVC (servlet + jsp):

1. Если у вас уже есть приложение, которое написано на servlet + jsp, и у вас есть необходимость сделать в нем некоторый AJAX функционал, то гораздо проще использовать, например, Jquery (если вы знаете JavaScript), чем перевести это приложение на JSF.
2. По сравнению с JSP и servlet на JSF написано заметно меньше документации.
3. У JSF приложения есть много моментов, которые тяжело отследить, понять и оптимизировать, например генерация html/css/javascript кода выбранным компонентом, и др.
4. Существует большое количество IDE, которые поддерживают стандартный JSP и servlet, однако JSF 2 начали поддерживать недавно, так как это достаточно новая технология. Последний на данный момент релиз JSF 2 был выпущен [2009-06-28](http://en.wikipedia.org/wiki/JavaServer_Faces#JSF_versions).

**Сравнение JSF 2 с JSF 1.2**

Новые возможности JSF 2 по сравнению с JSF 1.2:

* Библиотека facelets и View Declaration Language.
* Composite Components.
* Ajax.
* Частичное сохранение состояния (State Saving).
* System Events.
* Улучшенная навигация.
* Поддержка GET.
* View Scope.
* Flash Scope.
* Custom Scopes.
* Аннотации над Managed Bean.
* Аннотации над компонентами.

В целом же, забегая вперед, библиотеки jsf-api.jar и jsf-impl.jar нужны, если вы используйте контейнер Tomcat 6/7. Если же вы используйте сервер Java EE 6(например Glassfish 3, JBoss 6, WebLogic 11g ...), то он уже имеет встроенную поддержку JSF 2.0. 

**В JSF встречаются два вида запросов от клиента:**

* **Первый запрос (*initial request*)** - пользователь запрашивает страницу в первый раз
* **Postback** - отправка формы со страницы, которая была показана при первом запросе

*Существует 4 варианта запросов / ответов, которые можно встретить, работая с JSF платформой:*

1. Не JSF запрос - не JSF ответ (например: servlet -> servlet/jsp...)
2. Не JSF запрос - JSF ответ (например: servlet -> jsf страница)
3. JSF запрос - не JSF ответ (например: jsf страница -> отдать zip файл)
4. JSF запрос - JSF ответ (например: jsf страница -> jsf страница)

JSF представляет собой набор пользовательских компонентов в виде дерева, которые формируют представление. Представление представлено объектом UIViewRoot, который связан с активным [FacesContext](http://docs.oracle.com/javaee/6/api/javax/faces/context/FacesContext.html). Состояние представления может сохраняться как на клиентской стороне (в hidden полях), так и на серверной (по умолчанию). Во время выполнения, JSF реализация создаёт представление при первом обращении (запросе), либо восстанавливает уже созданное.

**Когда клиент отправляет форму** (*postback*),

JSF конвертирует отправленные данные, проверяет их,

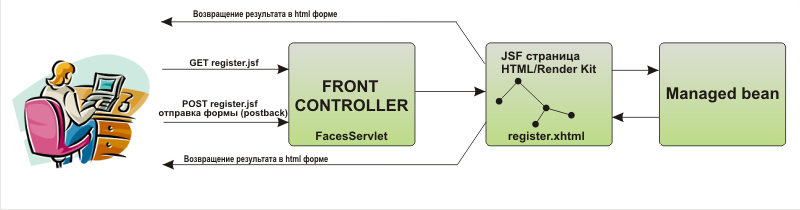
сохраняет в *managed bean*,

находит представление для навигации,

восстанавливает значения компонента из *managed bean*,

генерирует ответ по представлению.

Все эти действия JSF описывает с помощью 6 упорядоченных процессов, которыми можно управлять (вызывая например метод *renderResponse()* у активного *FacesContext*, либо используя свойство компонентов *immediate="true"*). Каждый раз, как пользователь отправляет запрос на сервер, один либо более процессов принимают участие в его обработке, после чего формируется и отправляется ответ. Графически это можно показать следующим образом:



**Managed bean** - это обычный Java бин, который зарегистрирован в JSF и управляется JSF платформой. В JSF *managed bean* используются в качестве модели для компонентов и имеют свою область жизни (*scope*), которую можно задать, как при помощи аннотации, так и в конфигурационном файле *faces-config.xml*.

Обработка аннотаций в JSF 2 происходит при запуске приложения. Сканируются классы, которые находятся в директории *WEB-INF/classes* и в *WEB-INF/lib/\*.jar*, причём jar'ы сканируется в случае, если в них присутствует файл */META-INF/faces-config.xml*.

Параметры, указанные в конфигурационном файле *faces-config.xml* затирают аннотации.

Аннотации для managed beans JSF находятся в пакете javax.faces.bean

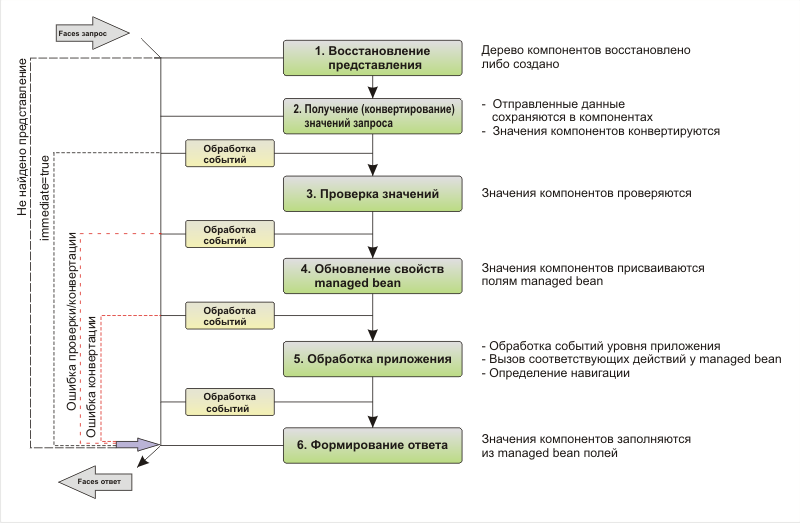
Рассмотрим варианты областей жизни, которые задаются в *managed bean* при помощи аннотаций:

1. **@RequestScoped** - используется по умолчанию. Создаётся новый инстанс *managed bean* на каждый HTTP запрос. Если, например форма будет содержать данные, которые необходимо будет отправить на сервер для обработки, то инстанс данного бина будет создаваться 2 раза: 1 - создаётся по первому запросу (*initial request*), 2 - создаётся по отправке формы (*postback*)
2. **@SessionScoped** - инстанс создаётся один раз при обращении пользователя к приложению, и используется на протяжении жизни сессии. *Managed bean* обязательно должен быть Serializable
3. **@ApplicationScoped** - инстанс создаётся один раз при обращении, и используется на протяжении жизни всего приложения. Не должен иметь состояния, а если имеет, то должен синхронизировать доступ, так как доступен для всех пользователей
4. **@ViewScoped** - инстанс создаётся один раз при обращении к странице, и используется ровно столько, сколько пользователь находится на странице (включая ajax запросы)
5. **@CustomScoped(value="#{someMap}")** - инстанс создаётся и сохраняется в Map. Программист сам управляет областью жизни
6. **@NoneScoped** - инстанс создаётся, но не привязывается ни к одной области жизни. Полезно применять в *managed bean*'е, на который ссылаются другие *managed bean*'ы, имеющие область жизни

**Фазы жизненного цикла JSF запроса** **- ответа** можно представить следующим образом:

1. Восстановление представления
2. Получение (конвертирование) значений запроса и обработка событий
3. Проверка значений и обработка событий
4. Обновление свойств *managed bean* и обработка событий
5. Обработка приложения и событий
6. Формирование ответа

Полный жизненный цикл запроса в JSF показан ниже в виде схемы, где с помощью толстой линии показан основной поток выполнения, тогда как альтернативный поток - пунктирной линией. Альтернативный поток зависит от атрибутов компонента, ошибок конвертации или проверок.



Рассмотрим более подробно фазы жизненного цикла приложения.

1. **Восстановление представления** - в этой фазе JSF пытается найти представление, которое соответствует запрошенной странице. Если представление не существует, то будет создано новое представление. Значения, обработчики событий, валидаторы и конверторы загружаются с компонентами, которым они принадлежат. Инициализируются языковые настройки. Если в этой фазе выполняется первый запрос (*initial request*), то представление не имеет состояния, и JSF пропускает все остальные шаги обработки этого запроса до фазы *Формирование ответа*. Если же в этой фазе выполняется *postback*, тогда JSF изменяет созданное состояние представления, и отправляет запрос на фазу *Получение (конвертирование) значений запроса и обработка событий*
2. **Получение (конвертирование) значений запроса и обработка событий** - в этой фазе JSF расшифровывает HTTP запрос, получает HTTP параметры и передаёт их компонентам в дереве компонентов. Представление рекурсивно проходится по дереву компонентов, и вызывает у каждого компонента метод *decode()*. Каждый компонент определяет свой параметр в HTTP запросе (по id), и заменяет существующее значение на новое (*setSubmittedValue()*). При этом, если к компоненту привязан конвертер, то значение конвертируется. Если в процессе конвертации произошли ошибки, то ошибка сохраняется и JSF пропускает все остальные шаги обработки этого запроса до фазы *Формирование ответа* для предупреждения пользователя об ошибке. Если компонент ввода (UIInput) содержит валидатор и имеет свойство *immediate="true"*, то этот компонент будет валидироваться в этой фазе.
3. **Проверка значений и обработка событий** - в этой фазе JSF валидирует дерево компонентов, вызывая у каждого компонента метод *validate()*. Если валидация компонента прошла с ошибкой, то ошибка сохраняется и JSF пропускает все остальные шаги обработки этого запроса до фазы *Формирование ответа* для предупреждения пользователя об ошибке.
4. **Обновление свойств *managed bean* и обработка событий** - в этой фазе JSF обновляет *managed bean* информацией из дерева компонентов.
5. **Обработка приложения и событий** - в этой фазе JSF обрабатывает события, которые пришли от кнопок и ссылок. Если один из этих компонентов имеет свойство *immediate="true"*, то он должен был быть уже обработан в фазе *Получение (конвертирование) значений запроса и обработка событий*. На данном этапе также происходит вычисление навигации для приложения, если навигация необходима.
6. **Формирование ответа** - в этой фазе JSF просит web-контейнер создать ответ, основываясь на данных, полученных из предыдущих шагов. Компоненты на странице обновляются данными из *managed bean* и генерируют html страницу с помощью Renderers. Если на предыдущих шагах происходили какие-либо ошибки, то они инкапсулируются в тег <messages>. Для выполнения этих действий JSF спецификация требует наличия обработчика представления (*view handler*).

Во 2-ой, 3-ей, 4-ой, 5-ой фазах компонент может инициировать-создать событие. Все события складываются в очередь событий компонента, и после выполнения фазы, JSF для каждого события в очереди вызывает оповещение по всем слушателям данного компонента о событии. Слушатели оповещаются в порядке, в котором они были зарегистрированы в компоненте.

Любой компонент в JSF в любой из 5-и первых фаз может вмешаться в процесс обработки запроса. Например, при вызове метода *renderResponse()* у активного *FacesContext* приведёт к тому, что JSF пропустит все остальные шаги обработки этого запроса до фазы *Формирование ответа*. Также *FacesContext* имеет метод *responseComplete()*, который обрывает выполнение JSF запроса, подразумевая, что пользователь уже получил ответ от кого-то другого.

**JSF 2 работает минимум с двумя конфигурационными файлами:**

1. **faces-config.xml** — конфигурационный файл JavaServer Faces, который должен находиться в директории *WEB-INF* проекта. В этом файле могут находиться настройки *managed bean*, конвертеры, валидаторы, локализация, навигации и другие настройки, связанные с JSF
2. **web.xml** - стандартный конфигурационный файл, который представляет собой ядро Java web приложения

**Несколько графических примеров работы JSF 2.**

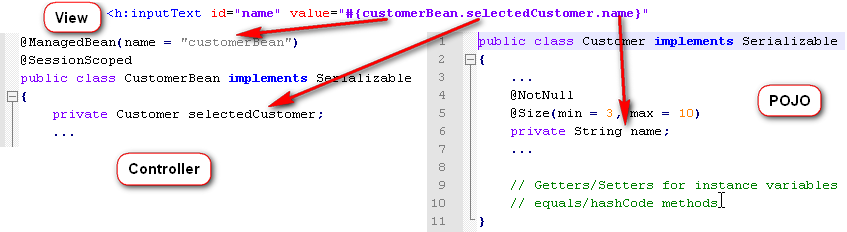
Пример отправки и обработки формы с данными на сервере:



На данном рисунке изображено две кнопки. При нажатии на кнопку с действием *action="#{customerBean.cancel}"*, срабатывает метод *cancel* в *managed bean*, который возвращает в качестве результата строку, которая указывает на элемент навигации. Данный элемент навигации описан в файле *faces-config.xml* и говорит о том, что необходимо выполнить *forward* на представление */pages/register.xhtml*, по которому будет построен html. При нажатии на кнопку *actionListener="#{customerBean.save}"* выполняется действие *save(ActionEvent event)* в *managed bean* при этом пользователь остаётся на том же представлении, что и был раньше (AJAX запрос). Хотелось бы выделить основные события, которые могут зажигаться из пользовательского интерфейса:

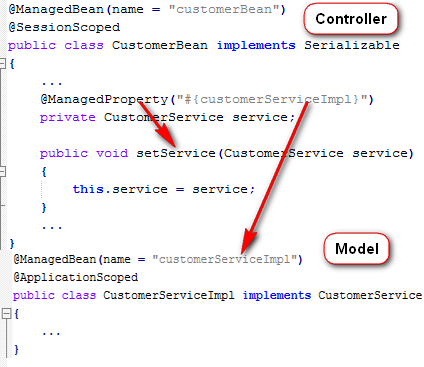
1. **ActionEvent** - вызывает компонент (например кнопка или ссылка), активированный пользователем (пример: *<h:commandLink actionListener="#{someBean.someAction}"/>*)
2. **ValueChangeEvent** - вызывает компонент (например check box/list box/combo box/radio buttons), чьё состояние было изменено (пример:*<h:selectOneMenu value="#{someBean.someProperty}" valueChangeListener="#{someBean.listener}"/>*)
3. **AjaxBehaviorEvent** - вызывает ajax компонент (пример: *<f:ajax event="change" listener="#{someBean.listener}"/>*)

Пример заполнения свойств *managed bean* данными из формы:

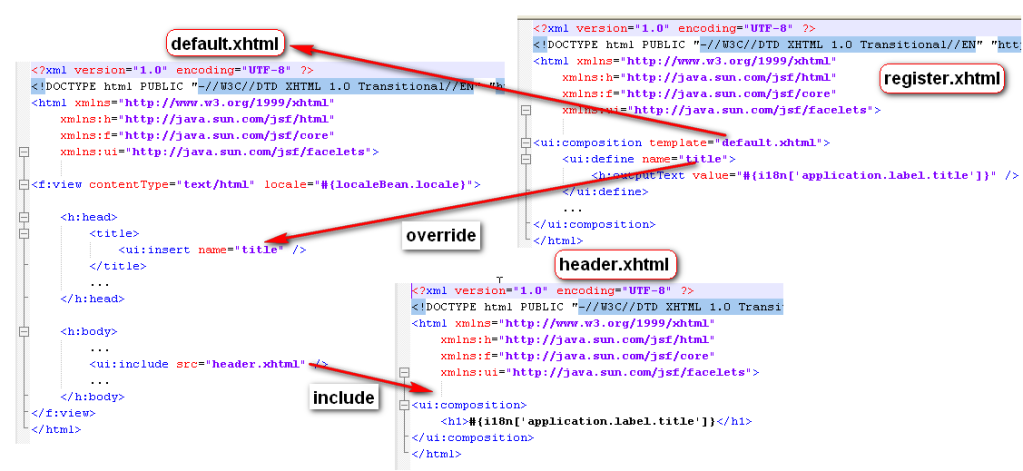


В JSF 2 с помощью **dependency injection** мы можем иньектить другие *managed bean*, которые существуют в приложении. JSF выполняет иньекцию объекта с помощью [EL](http://en.wikipedia.org/wiki/Expression_Language) через вызов соответсвующего setter'а и только во время создания инстанса.

Пример *managed bean* dependency injection:



Пример шаблонизации страниц:



На данном рисунке изображено два действия, которые отмечены, как *override* и *include*.

*include* - контент из файла *header.xhtml* вставляется в шаблон *default.xhtml*.

*override* - данное обозначение означает, что будет перезаписан весь контент, который находится внутри *<ui:insert name="title" />* тега (шаблон *default.xhtml*) контентом, который находится в *<ui:define name="title">* теге (*register.xhtml*).

**Задание:**

Архитектура:

Тип проекта – web [проект.](ПроектJSF.docx)

Разделение классов на слои(model view controller logic и тп).

Использование БД с предыдущей лаб.раб.

Функционал:

1) Функции: CRUD 2ух таблиц + поиск

2) При добавлении записи использовать выпадающий список

3) Валидация форм

4) Применение шаблона [Facelets](#_Теги_Facelets)(по желанию)

# Теги Facelets

Технология Facelets был первоначально разработана как альтернатива обработчику представлений на основе JSP, применявшемуся в ранних версиях JSF. В версии JSF 2.0 Технология Facelets заменила JSP в качестве применявшейся по умолчанию в JSF технологии представления. Платформа Facelets не только является лучшим обработчиком представлений, но и поддерживает целый ряд тегов, предназначенных для реализации шаблонов и других целей.

Теги Facelets могут быть сгруппированы по нескольким категориям:

1. Включение содержимого из других страниц XHTML — ui:include
2. Формирование страниц из шаблонов — ui:composition, ui:decorate, ui:insert, ui:define, ui:param
3. Создание пользовательских компонентов без написания кода Java — ui:component, ui:fragment
4. Различные утилиты — ui:debug, ui:remove, ui:repeat

Чтобы иметь возможность использовать теги Facelets, необходимо добавить следующее объявление пространства имен к конкретным страницам JSF

|  |  |
| --- | --- |
| [?](http://javasource.ru:5050/articles.xhtml?artlink=facelets-tags) | xmlns:ui="<http://java.sun.com/jsf/facelets>" |

В таблице ниже приведены краткие сведения о тегах Facelets.

|  |  |
| --- | --- |
| **Тег** | **Описание** |
| ui:include | Включает содержимое из другого файла XML |
| ui:composition | Будучи использованным без атрибута template, этот тег определяет последовательность элементов, которая может быть вставлена в другом месте. Композиция может иметь переменные части указанные с помощью дочерних тегов ui:insert.  При использовании с атрибутом template, загружается шаблон. Дочерние теги этого тега определяют переменные части шаблона. Содержимое шаблона заменяет этот тег. |
| ui:decorate | Будучи использованным без атрибута template этот тег определяет страницу, в которую могут быть вставлены части. Переменные части задаются с помощью дочерних тегов ui:insert.  При использовании с атрибутом template, загружается шаблон. Дочерние теги этого тега определяют переменные части шаблона. |
| ui:define | Определяет содержимое, которое вставляется в шаблон с помощью соответствующих тегов ui:insert. |
| ui:insert | Вставляет содержимое в шаблон. Это содержимое определяется в теге, который загружает шаблон. |
| ui:param | Задает параметр, передаваемый во включенный файл или шаблон. |
| ui:component | Этот тег идентичен ui:composition, за исключением того, что создает компонент добавляемый к дереву компонентов. |
| ui:fragment | Этот тег идентичен ui:decorate, за исключением того, что создает компонент добавляемый к дереву компонентов. |
| ui:debug | Этот тег позволяет пользователю с помощью определенной комбинации клавиш вывести на экран окно отладки, в котором показаны иерархия компонентов для текущей страницы и переменные с областью действия приложения. |
| ui:remove | Реализация JSF удаляет все, что находится в этом теге |
| ui:repeat | Выполняет итерации по списку, массиву, результирующему набору или отдельному объекту. |