**Жизненный цикл**

При поступлении от клиента (браузера) запроса необходимо выполнить определенную последовательность действий, связанную с анализом параметров запроса и, при необходимости, подготовить ответ. При использовании в WEB приложении фреймворков типа Struts, JSF **жизненный цикл** фреймворка выполняет все необходимые действия, которые пришлось бы делать программисту.

**Жизненный цикл JSF** управляет входными параметрами запроса от клиента, UI-компонентами и синхронизирует их с отображаемой на клиентской стороне странице, т.е. в окне браузера. Также он обрабатывает запросы со стороны клиента на предоставлении изображений, стилей CSS, скриптов и т.п., что необходимо браузеру для правильного представления страницы.

При запуске приложения **JSF 2** контейнер приложения загружает классы директории WEB-INF/classes, указанные в конфигурационном файле faces-config.xml. Для каждого класса устанавливается время жизни. Варианты времени жизни managed bean компонентов определяются либо в файле конфигурации **faces-config.xml**, либо при помощи [аннотаций](http://java-online.ru/jsf.xhtml) согласно принятому соглашению:

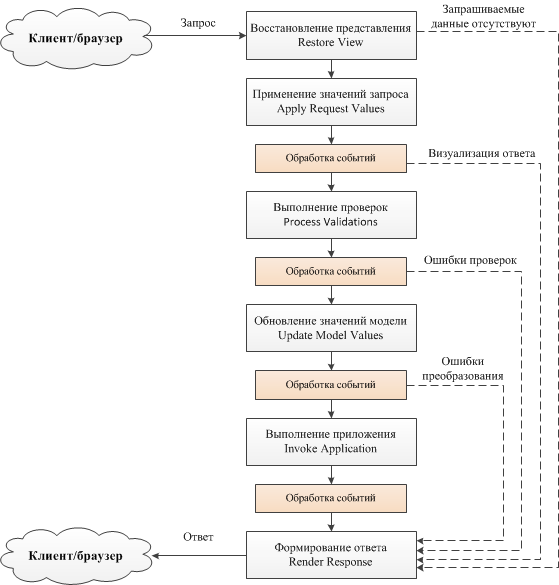
|  |  |
| --- | --- |
| **Время жизни** | **Описание** |
| @RequestScoped | Используется по умолчанию. При каждом HTTP запросе создается новый компонент. Если форма содержит данные, которые отправляются на сервер для обработки, то компонент будет создаваться 2 раза. Первоначально компонент создаётся при открытии страницы, с которой компонент связан. Повторно компонент создаётся по отправке формы. |
| @SessionScoped | Компонент создаётся один раз при создании сессии и используется на протяжении жизни сессии. Managed bean обязательно должен быть Serializable |
| @ApplicationScoped | Компонент создаётся один раз при обращении и используется на протяжении жизни приложения. |
| @ViewScoped | Компонент создаётся один раз при обращении к странице, и используется ровно столько, сколько пользователь находится на странице (включая ajax запросы). |
| @CustomScoped(value="#{myMap}") | Компонент создаётся и сохраняется в коллекции типа Map. Областью жизни управляет программист. |
| @NoneScoped | Компонент 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. |
| [flash scope](http://java-online.ru/jsf-lifecycle.xhtml#flashscope) | Объекты внутри этой области жизни будут доступны для последующего запроса, после чего удаляются. Другими словами объект во flash scope переживёт redirect, после чего будет удален. |

**Этапы жизненного цикла**

**Жизненный цикл** запроса-ответа проходит несколько **этапов** :

* восстановление представления;
* получение параметров запроса;
* проверка значений, конвертирование;
* обновление свойств managed bean;
* обработка приложения;
* формирование ответа.

Схематично **этапы жизненного цикла** обработки запроса в JSF показаны на следующем рисунке. Сплошной линией отображен основной поток выполнения. Пунктирной линией - альтернативный поток, который зависит от атрибутов компонентов, ошибок конвертации или проверок.



**Фазы жизненного цикла**

* Восстановление представления (Restore View) : для запрошенной страницы либо извлекается дерево компонентов, если страница уже открывалась, либо создается новое дерево компонентов, если страница запрашивается впервые. Для компонентов запрашиваемой страницы восстанавливаются их прошлые состояния (форма заполняет вводимыми значениями).
* Применение значений запроса (Apply Request Values) : анализ HTTP запроса, объектам дерева компонентов присваиваются соответствующие им значения из запроса. Если к компоненту подключен конвертер, то значение обрабатывается/конвертируется. При возникновении ошибки, все последующие шаги пропускаются. Если компонент ввода (UIInput) содержит валидатор и имеет свойство immediate="true", то этот компонент будет валидироваться в этой фазе. Также, при необходимости, события (нажатие кнопки или ссылки) добавляются в очередь событий.
* Выполнение проверок (Process Validations) : преобразование строковых значений в "локальные значения" и применение валидации дерева компонентов. Если в результате валидации компонента возникает ошибка, то она сохраняется и JSF пропускает все последующие шаги обработки запроса до фазы "Формирования ответа" для предупреждения пользователя об ошибке.
* Обновление значений модели (Update Model Values) : обновление свойства managed bean информацией из дерева компонентов.
* Выполнение приложения (Invoke Application) : JSF обрабатывает события, которые были сгенерированы нажатием на кнопки и ссылки. На данном этапе также решаются вопросы, связанные с навигацией приложения, если это необходимо. Если один из компонентов формы имеет свойство **immediate="true"**, то он должен был быть обработан в фазе "Применение значений запроса".
* Формирование ответа (Render Response) : JSF создает ответ, основываясь на данных, полученных на предыдущих шагах. Информация страницы обновляется данными из managed bean и генерируется html страница с помощью Renderers. Если на предыдущих шагах происходили какие-либо ошибки, то они инкапсулируются в тег <messages>.

*Страница Java Server Faces представлена в виде дерева компонентов. Корневым компонентом дерева является UIViewRoot.*

**Фаза жизненного цикла Restore View**

В фазе **Restore View** фреймворк выполняет следующие действия :

* Проверка FacesContext текущего запроса. Если он содержит UIViewRoot, то
  + Установить значение locale для UIViewRoot, извлекаемую из метода ExternalContext.html#getRequestLocale() текущего запроса.
  + Для каждого компонента дерева проверить значение ValueBinding связи "binding". Если связь определена, то у ValueBinding вызвать метод setValue().
  + Не предпринимать дальнейших действий на этой стадии.
* Определение идентификатора текущего запроса вида (view identifier) :
  + если используется маппинг с префиксом (например, "/faces/\*"), то в качестве viewId использовать значение, определенное на месте символа '\*';
  + если используется маппинг с суффиксом (например, "\*.faces"), то в качестве viewId назначается путь, по которому был сделан запрос;
  + Если идентификатор определить не удаётся, то вызвать исключение.
* Вызов метода ViewHandler#restoreView() с передачей в качестве параметров FacesContext и идентификатора вида. Результат выполнения (возможно) UIViewRoot.
  + если restoreView() возвращает null, то последовательно вызываются ViewHandler#createView() и FacesContext#renderResponse();
  + если запрос не содержит POST-данных или параметров (query parameters), то вызывается FacesContext#renderResponse().
* Сохранение созданного или восстановленного UIViewRoot в FacesContext'е.

В конце данной фазы свойство viewRoot у FacesContext'а текущего запроса будет содержать сохранённое состояние вида, которое было в предыдущем запросе или новый вид, созданный с помощью ViewHandler#createView() для указанного идентификатора вида.

**Фаза жизненного цикла Apply Request Values**

На данном **этапе жизненного цикла** каждый компонент обновляет своё состояние на основании информации текущего запроса (параметры, заголовки, cookies и т.д.). Фреймворк должен вызвать метод UIViewRoot#processDecodes, который вызовет методы UIComponent#processDecodes для всех компонентов дерева. Для UIInput-компонентов будет выполнено преобразование данных.

В процессе декодирования данных запроса компоненты могут вызвать определенные действия :

* компоненты, реализующие интерфейс ActionSource (например, UICommand), выполняют проверку своей активности и, при необходимости, добавят ActionEvent в очередь событий. Данное событие будет вызвано в конце фазы Apply Request Values или в конце фазы Invoke Application, в зависимости от состояния свойства immediate у активированного компонента.
* компоненты, реализующие интерфейс EditableValueHolder (например, UIInput) с установленным свойством immediate в true, вызовут преобразование и валидацию (включая потенциально запуск событий ValueChangeEvent, которые по-нормальному происходят в фазе Process Validations).

В конце этой фазы все компоненты, которые реализуют интерфейс EditableValueHolder, будут обновлены в соответствии с данными, переданными в запросе. Если при преобразовании и валидации возникнут ошибки, то с помощью метода FacesContext#addMessage для соответствующих компонентов будут добавлены сообщения, и их свойство valid будет выставлено в false.

Если в данной **фазе жизненного цикла** какой-либо из слушателей событий (event listener) или методов decode() вызвал один из методов

* FacesContext#responseComplete(), то обработка текущего запроса должна быть немедленно прекращена
* FacesContext#renderResponse(), то управление должно немедленно перейти к фазе Render Response.

Иначе управление должно перейти к фазе Process Validations.

**Фаза жизненного цикла Process Validations**

В фазе Process Validations вызывается метод UIViewRoot#processValidators(), которые для каждого компонента дерева вызывает метод UIComponent#processValidators(). Компоненты с установленным в true свойством immediate уже выполнили свою валидацию на предыдущем шаге.

Если на данном **этапе жизненного цикла** возникнут ошибки преобразования или валидации, то будет вызван метод FacesContext#addMessage для добавления сообщения об ошибке соответствующих компонентов, и их свойство valid будет выставлено в false.

Если в данной фазе какой-либо из вызванных методов validate() или слушатель событий (event listener) вызвал один из методов

* FacesContext#responseComplete(), то обработка текущего запроса должна быть немедленно прекращена
* FacesContext#renderResponse(), то управление должно немедленно перейти к фазе Render Response.

Иначе управление должно перейти к фазе Update Model Values.

**Фаза жизненного цикла Update Model Values**

На данном этапе можно константировать, что параметры запроса корректны, и что локальное значение каждого компонента дерева с корневым элементом UIViewRoot было обновлено. Наступило время обновления данных в модели приложения для подготовки к выполнению событий приложения, которые находятся в очереди.

В данной фазе вызывается метод UIViewRoot#processUpdates(), после чего для каждого компонента дерева будут вызваны методы UIComponent#processUpdates(). Обновление модели выполняется в методе updateModel() компонента. В процессе обновления модели в очередь могут быть добавлены события, которые будут будут распространены заинтересованным слушателям событий.

На завершающей стадии данного этапа все объекты модели будут обновлены значениями, содержащимися в соответствующих компонентах.

Если какой-либо из вызванных методов validate() или слушатель событий (event listener) на данном этапе вызвал один из методов

* FacesContext#responseComplete(), то обработка текущего запроса должна быть немедленно прекращена
* FacesContext#renderResponse(), то управление должно немедленно перейти к фазе Render Response.

Иначе управление должно перейти к фазе **Invoke Application**.

**Фаза жизненного цикла Invoke Application**

Все обновления модели выполнены и оставшиеся события передаются для обработки приложению. Вызывается метод UIViewRoot#processApplication() для обработки событий, у которых определен идентификатор фазы PhaseId.INVOKE\_APPLICATION.

Перехватить обработку событий можно переопределением **ActionListener**.

**Фаза жизненного цикла Render Response**

Render Response - это завершающая фаза жизненного цикла JSF, в которой формируется ответ. Ответ формируется каскадно, как и во всех предыдущих фазах : вызываются методы encodeXX() каждого компонента. Эти методы определяют, как компоненты (точнее, их визуализаторы — renderers) будут отображаться клиенту. Язык разметки ответа может быть любым : HTML, XHTML, XML и т.п. Кроме генерации ответа на данном этапе происходит сохранение текущего состояния вида, обеспечивая возможность его восстановления при последующих обращениях к странице.

Таким образом, в фазе **Render Response** фреймворк выполняет два действия:

* формирует ответ для клиента;
* сохраняет состояние результата для последующий запросов.

По завершении рендеринга конечное состояние вида должно быть сохранено посредством методов класса [StateManager](http://docs.oracle.com/cd/E17802_01/j2ee/j2ee/javaserverfaces/1.1_01/docs/api/javax/faces/application/StateManager.html). Эта информация о состоянии должна быть доступна при последующем запросе, чтобы ей можно было воспользоваться в фазе Restore View.

*К фазе****Render Response****также относится и объединение статического описания страницы («шаблон»), с ее динамическим наполнением. Но это уже связано и с используемыми фреймворками типа Facelets, Tiles.*