Система состояний

BIV Group

1.0.1, : Final

Table of Contents

Введение . . . 1
Абстрактная машина состояний. . . . 2
- Схема состояний . . . 2
- Машина состояний. . . . 2
- Порядок работы с абстрактной машиной состояний. . . . 3

Интеграция с JPA . . . 5
- Подготовка БД к использованию системы состояний. . . . 5
- Особенности описания Entity документа . . . 6
- Конфигурация схемы состояний. . . . 7
- Использование системы состояний. . . . 8

Работа с событиями . . . 11
- Подсистема событий . . . 11
- Исполнители подсистемы событий . . . 12
  - Событие инициализации . . . 12
  - Событие перед переходом . . . 12
  - Событие после перехода . . . 14
  - Обработчики событий . . . 14

Введение

Система состояний представляет собой библиотеку, предназначенную для использовании в системах автоматизации документооборота. Система состояний позволяет:

1. Задать схему состояний, включающую:
    a. Состояния
    b. Переходы между состояниями.
    c. Права на выполнение переходов.
    d. Метаданные событий, возникающих при выполнении переходов.
2. В соответствии со схемой выполнять переходы между состояниями.
3. Задавать схему состояний в виде json или в реляционной БД.
4. Реализовывать обработчики событий через CDI
5. Использовать систему состояний для JPA Entity через интеграцию с JPA.
6. Задавать информацию о схеме состояний через конфигурацию Quarkus.
7. Расширять систему состояний, путем реализации CDI Beans.

Данная реализация системы состояний состоит из двух частей:

1. Абстрактная машина состояний, которая по метаданным (схеме состояний) может выполнять переходы в соответствии с правами.
2. Интеграция с JPA, при которой уже идет оперирование сущностями.

Абстрактная машина состояний.

Схема состояний

Схема состояний идентифицируется именем и версией схемы.

Схема состояний состоит из:

1. Привязки к документу, для которого описана схема.
2. Состояний.
3. Переходов между состояними.
4. Прав на переходы по ролям.
5. Метаданных событий, возникающих при переходах.

Схема состояний описана в виде набора интерфейсов в модуле state-api. В нем же определена generic имплементация модели. Модель описана максимально обобщенно, без привязки к методам хранения, методом описания сущностей и тому подобное. Начальное состояние схемы определяется как состояние, в которое нет переходов. Конечное состояние схемы определяется как состояние, из которого нет переходов.

Машина состояний.

Машина состояний определяется интерфейсом StateMachine. Эталонная реализация находится в модуле state-core.

```
/**
 * Интерфейс системы состояний
 * @author mvolkov
 */
public interface StateMachine {
    /**
     * Установить текущую схему системы состояний
     * @param shemaName
     * @param schemaVersion
     * @return
     */
    boolean setMachineSchema(String shemaName, String schemaVersion);
     /**
     * Сделать переход из состояния. При этом будут:
     * 1. Проверяться права текущего пользователя на возможность перехода
     * 2. Вызываться события до перехода.
     * 3. Если события не отменили переход, то выполняется переход.
     * 4. Вызываются асинихронно события после перехода.
     * @param fromState состояние, из которого делается переход.
     * @param transition переход.
     * @param context - произвольный объект, который передается в обработчики событий.
```

```
     * @return результирующее состояние или null, если переход не был выполнен.
     */
    State makeTransition(State fromState,Transition transition, Object context);

    /**
     * Получить схему состояний
     * @return
     */
    StateMachineSchema getMachineSchema();

    /**
     * Получить начальные состояния.
     * Начальное состояние - состояние в которое нет переходов.
     * @return
     */
    Set<State> getInitialStates();
    /**
     * Получить конечные состояния.
     * Конечное состояние - состояние из которого нет переходов.
     * @return
     */
    Set<State> getEndState();
    /**
     * Получить переход по имени.
     * @param transitionName
     * @return Переход или null, если не найден
     */
    Transition getTransitinByName(String transitionName);
    /**
     * Получить состояние по имени.
     * @param stateName
     * @return Состояние или null, если не найден
     */
    State getStateByName(String stateName);
    /**
     * Получить доступные переходы для данного пользователя с учетом прав.
     * Права реализованы через quarkus-security. Будет работать с любой реализацией на
базе quarkus-security (oidc,jwt oauth2 и подобные)
     * @param state
     * @return
     */
    Set<Transition> getAvaibleTransitions(State state);
}
```

Порядок работы с абстрактной машиной состояний.

Для работы с абстрактной сиземой состояний необходимо подключить следующие модули:

1. state-api - интерфейсы системы состояний и простая (generic) имплементация схемы состояний.
2. state-core - реализация машины состояний.
3. Один из модулей state-loader-json или state-locader-hb или другой подобный модуль, которые содержать загрузчик схемы состояний. Загрузчики схемы состояний подключаются автоматически через CDI. Если подключены оба загрузчика, то использован будет тот, с которого первого успешно загрузилась схема с заданным именем и версией.

Для использования интерфейс машины состояний должен через CDI быть заинжекчен в сервис, где предполагается его применять.

```
@Path("/state")
public class StateMachineExample {
    @Inject
    StateMachine sm;


    // использование объекта.
}
```

В каждом методе при каждом использовании объекта сначала должна быть установлена схема состояний методом setMachineSchema. Так как схема кэшируется в ОЗУ, то метод работает быстро.

Предполагаемый алгоритм работы с объектом:

1. При создании нового объекта необходимо получить список начальных состояний и установить одно из них у документа.
2. При открытии на редактирование страницы с документом должны быть вычитан список доступных переходов методом getAvaibleTransitions. Предполагается, что для перехода на интерфейсе будут созданы элементы управления (кнопки, значения в выпадающем списке и так далее)
3. При воздействии на элемент управления, связанный с переходом, в сервисе необходимо получить переход по имени (getTransitionByName) и выполнить переход методом makeTransition.

Интеграция с JPA

Многие сервисы работают с реляционными базами данных через стандарт JPA. Поэтому очень часто система состояний используется с JPA. Для более удобного использования системы состояний внутри quarkus с JPA была реализована интеграция.

Подготовка БД к использованию системы состояний.

Перед использованием системы состояний небоходимо:

1) Создать таблицу, предназначенную для хранения состояний. Структура таблицы должна обеспечивать чтение ее через следующую JPA Entity:

```java
/**
 * Entity для хранения состояний
 * @author mvolkov
 */
@Entity
@Table(name = "SM_STATE")
public class StateEntity {

    @Id
    private Long id;

    /**
     * Имя состояния
     */
    @Column(name = "name")
    private String name;
    /**
     * Текстовое имя состояния.
     * Может быть использовано для отображения в интерфейсе.
     */
    @Column(name = "fullName")
    private String fullName;
    /**
     * Имя схемы состояний
     */
    @Column(name = "schemaName")
    private String schemaName;
    /**
     * Версия схемы состояний
     */
    @Column(name = "schemaVersion")
    private String schemaVersion;

}
```

2) Заполнить ее состояниями, которые есть в схемах состояний.

Особенности описания Entity документа

Пример описания entity документа:

```
@Entity
@Table(name = "EXAMPLE_CONTRACT")
@EntityListeners(value = {StateMachineJPAEventListener.class})
public class Contract {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;
    @Column(name = "number")
    private String number;
    @Column(name = "premium")
    private Double premium;

    @StateField(configKey = "firstConfig")
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "state")
    private StateEntity state;
}
```

При описании Entity документа необходимо выполнить следующие требования:

1. С помошью аннотации @EntityListeners(value = {StateMachineJPAEventListener.class}) подключить обработчик событий для Entity системы состояний.
2. Описать поле (или поля) состояния.
    a. Поле должно быть типа StateEntity
    b. Должна быть описана связь ManyToOne
    c. Над полем должна быть задана аннотация @StateField(configKey = <Имя конфигурации схемы состояний>)

Конфигурация схемы состояний.

Для инициализации полей с состояниями внутри документа необходимо знать:

1. Схему состояний (имя и версия).
2. Начальное состояние в схеме.

Эти параметры задаются в конфигурации, например, в файле application.properties:

```
quarkus.statemachine.firstConfig.schemaname = test
quarkus.statemachine.firstConfig.schemaversion = 1
quarkus.statemachine.firstConfig.initstate = FirstState
```

где firstConfig - имя конфигурации схемы состояний, которая описывается в аннотации StateField

Использование системы состояний.

Для использования системы состояний с JPA необходимо дополнительно к модулям системы состояний подключить модули:

1. jpa-entity-api - интерфейс, аннотация и entity StateEntity
2. jpa-entity - имплементация
3. jpa-state-integration - расширение quarkus для обработки конфигурации и кэширования entity состояний для схем.

Работа системы состояний с JPA выполняется через интерфейс EntityStateMachine:

```
/**
 * Интеграция машины состояний с JPA
 * @author mvolkov
 */
public interface EntityStateMachine {


    /**
     * Получить переход по имени перехода
     * @param name
     * @return
     */
    Transition getTransitionByName(String name);
    /**
     * Получение списка доступных переходов по состоянию
     * @param state
     * @return
     */
    Set<Transition> getAvaibleTransitions(StateEntity state);
    /**
     * Получение списка доступных переходов из состояния для данного пользователя
     * при условии, что в Entity присутствует одно поле с состоянием
     * Иначе возвращает null
     * @param entity
     * @return
     */
    Set<Transition> getAvaibleTransitions(Object entity);
    /**
     * Получение списка доступных переходов из состояния для данного пользователя
     * по имени поля, в котором храниться состояние объекта
     * @param entity
     * @param stateFieldName
     * @return
     */
    Set<Transition> getAvaibleTransitions(Object entity, String stateFieldName);

    /**
```

```
     * Получение начального состояния схемы по имени и версии схемы
     * @param schemaName
     * @param schemaVersion
     * @return
     */
    StateEntity getInitialState(String schemaName, String schemaVersion);
    /**
     * Получение начального состояния схемы по имени конфига для схемы
     * @param configKey
     * @return
     */
    StateEntity getInitialState(String configKey);
    /**
     * Получение начального состояния схемы по Entity классу,
     * при условии, что в классе одно поле с состоянием
     * @param clazz
     * @return
     */
    StateEntity getInitialState(Class clazz);


    /**
     * Выполнить переход в системе состояний.
     * В entity меняется значение поля состояние.
     * Метод работает, если в entity одно поле с состоянием
     * @param entity
     * @param transition
     * @return - null, если ошибка, иначе состояние в результате перехода
     */
    StateEntity makeTransition(Object entity, Transition transition);
    /**
     * Выполнить переход в системе состояний.
     * В entity меняется значение поля состояние.
     * @param entity
     * @param transition
     * @param stateFieldName - имя свойства, в котором храниться состояние
     * @return
     */
    StateEntity makeTransition(Object entity, Transition transition, String
stateFieldName);

    /**
     * Выполнить переход в системе состояний.
     * @param stateEntity
     * @param transition
     * @return
     */
    StateEntity makeTransition(StateEntity stateEntity, Transition transition);
}
```

Следует учесть, что в entity документа может быть несколько полей с состояниями. Однако, в большинстве случаев это поле одно. Поэтому в интерфейсе предусмотрены методы для такого случая для упрощения работы с системой состояний. Предполагаемый порядок работы следующий:

1. При создании нового entity документа автоматически инициализируется начальное состояние.
2. Для отображения доступных переходов у уже созданного entitiy документа на интерфейсе их можно получить с помощью метода getAvaibleTransition(Object entity). Автоматически будет найдена и использована нужная схема состояний и учтены права на переходы для данного пользователя.
3. При выполнении перехода на интерфейсе сервис на сервере должен:
    a. Получить переход по имени getTransitionByName
    b. Выполнить переход одним из методов makeTransition(..).

Работа с событиями

Предусмотрено три уровня работы с событиями:

1. Подсистема событий (CDI, vertx, MQ и подобные)
2. Исполнители событий (java, js, Groovy и др.)
3. Собственно сами обработчики событий.

Подсистема событий

Интерфейс подсистемы событий:

```
/**
 * Интерфейс подсистемы событий
 * @author mvolkov
 */
public interface EventCaller {
    /**
     * Вызвать инициализирующее события,
     * которое позволяет получить исполнителям событий метаданные событий.
     * @param events
     */
    void callInitEvents(Set<Event> events);

    /**
     * Вызвать событие до перехода
     * @param transition
     * @param userName
     * @param userRoles
     * @param context
     * @return
     */
    boolean callBeforeEvents(Transition transition, String userName, Set<String>
userRoles, Object context);

    /**
     * Вызывать событие после перехода
     * @param transition
     * @param userName
     * @param userRoles
     * @param context
     */
    void callAfterEvents(Transition transition, String userName, Set<String>
userRoles, Object context);

}
```

На текущий момент существует имплементация событий через CDI. Однако, при необходимости может быть реализована и другая подсистема.

Исполнители подсистемы событий

Реализация исполнителей подсистемы событий зависит от самой подсистемы событий. Рассмотрим исполнители для CDI. Исполнитель событий - CDI объект, который получает события системы состояний и обеспечивает выполнение предметного кода обработчиков событий Исполнитель событий получает все события, которые возникают в процессе работы системы состояний. Задача исполнителя событий заключается в том:

1. Понять, может ли он обработать данное событие.
2. Если может обрабатывать, то выбрать предметные обработчики события, выполнить их.

Для CDI объекта исполнителя событий должен использоваться скоп ApplicationScope или Singleton, так как иначе в экземпляре может не сохраниться информации по инициализации события.

Согласно спецификации CDI (https://docs.jboss.org/cdi/spec/2.0/cdi-spec.html#events) события бывают синхронные и асинхронные.

Событие инициализации

Событие инициализации исполнителя событий вызывается тогда, когда первый раз загружается схема в кэш. Событие позволяет получить метаданные событий схемы исполнителю событий.

```
/**
 * Интерфейс инициализации исполнителя событий
 * @author mvolkov
 */
public interface StateInitEvent {
    /**
     * Получить метаданные событий
     * @return
     */
    Set<Event> getEventsMetadata();
}
```

Событие перед переходом

Событие перед переходом выполняется в синхронном режиме, так как оно может остановить переход. Событие не должно долго выполняться, так как это может затормозить выполнение бизнес-метода, где выполняется переход. Событие описывается интерфейсом:

```
/**
 * Интерфейс события
 * @author mvolkov
 */
public interface StateEvent {
    /**
     * Выполняемый переход
     * @return
     */
    public Transition getTransition();

    /**
     * Имя пользователя
     * @return
     */
    public String getUserName();

    /**
     * Список ролей
     * @return
     */
    public Set<String> getRoles();

    /**
     * Объект с контекста
     * @return
     */
    public Object getContext();

    /**
     * Признак того, что выполнение перехода остановлено.
     * @return
     */
    public boolean isAbortTransition();

    /**
     * Остановить выполнение перехода
     */
    void abortTransition();
}
```

Исполнителю события совместно с событием инициализации передается информация достаточная, чтобы понять какой реальный обработчик выполнять. Исполнитель может, вызвав метод abortTransition() остановить выполнение перехода. Исполнение события происходит в том же контексте и в том же потоке, в котором выполняется основная программа. Поэтому метод исполнителя должен не начинать транзакцию, а продолжать существующую. Аннотация над методом должна быть (@Transactional(Transactional.TxType.MANDATORY))

Если обработчик события не выполняет операций, которые могут потребовать транзакцию, то аннотацию не надо ставить. Использование иных типов транзакций может повлечь ошибку в основном методе сервиса, где используется транзакция.

Событие после перехода

Событие после перехода вызывается в асинхронном режиме после перехода в отдельном потоке. В поток передается посредством ManagedExecutor текущий контекст выполнения. Однако, есть некоторые правила, которым должен подчиняться обработчик:

1. Нельзя менять поля передаваемой Entity документа, так как скорей всего выполнение метода сервиса уже закончилось и некому завершить транзакцию и сохранить данный entity. Если нужно поменять объект документа, то получите по ид его снова и меняйте.
2. Выполнение обработчика должно быть в отдельной новой транзакции.
3. Данные security передаются вместе с событием. Их и надо использовать в событии.

Обработчики событий

На текущий момент обработчики событий не реализованы.