Entity object template concept

BIV group

1.0, : Final

Table of Contents

Определение и назначение

Шаблон – набор заранее определённых значений полей для создания частично заполненного объекта.

Задача механизма шаблонов объектов – это заполнение полей для создаваемых объектов значениями по умолчанию. Данный механизм должен позволить вынести инициализацию полей для объектов определённых объектов из программного кода на уровень настройки (конфигурационный уровень), где им могут воспользоваться не только разработчики, а и аналитики, и внедренцы.

Данный механизм должен применяться точечно только для отдельных объектов в конкретных местах бизнес-процессов. Как правило это бизнес-объекты типа документов, бухгалтерских операций и т.п.

Механизм шаблонов располагается внутри микросервиса, в котором располагаются "шаблонизируемые" сущности.

Особенности и возможности

1. Шаблоны хранятся в базе данных микросервиса.
2. Шаблоны привязаны к сущности.
3. Шаблон имеет уникальное сокращение в рамках сущности.
4. Механизм шаблонов осуществляет подстановку значений-констант из шаблона в атрибуты создаваемого объекта.
    a. Для простых атрибутов типа поле со следующими типами данных: строка, целое число, дробное число, логическое.

Тип дата не поддерживается, т.к. при обсуждении не нашли реального примера заполнения атрибута константой с типом дата.

    b. Для атрибутов-ссылок на объект другой сущности. В данном случае в качестве константы указывается идентификатор объекта (записи).
5. Механизм шаблонов осуществляет подстановку динамически вычисляемых значений в атрибуты создаваемого объекта. Вычисление производится в специально зарегистрированных методах у applicatoin scoped bean.
    a. Для простых атрибутов специально зарегистрированный метод возвращает скалярное значение. Типы аналогичны предыдущему пункту.
    b. Для атрибутов-ссылок должен выбирается объект-ссылка.
6. Для использования механизма шаблонов необходимо вызвать метод newInstance у бина ObjectBuilder в коде.

    T.e. механизм шаблонов вызывается явно.

Классы Entity & DTO

Класс Entity - это класс, аннотированный @Entity и предназначенный для хранения объекта в базе данных.

Класс DTO - pojo-класс, предназначенный для передачи данных между бэкендом и фронтендом.

Механизм шаблонов может создавать экземпляры как Entity-классов, так и DTO-классов.

Предполагаемые сценарии работы:

1. Сценарий для записи в базу данных:
    a. Создаётся объект Entity-класса по шаблону.
    b. В коде заполняются другие атрибуты (поля) объекта.
    c. Объект сущности Entity-класса записывается в базу данных.
2. Сценарий для передачи объекта на фронтенд:
    a. Создаётся объект DTO-класса по шаблону.
    b. В коде заполняются другие поля объекта.
    c. Объект DTO-класса отправляется в фронтенд.
    d. Из фронтенда приходит уже до конца заполненный пользователем DTO.
    e. Из объекта DTO-класса создаётся новый объект Entity-класса без применения механизма шаблонов.
    f. Объект сущности Entity-класса записывается в базу данных.

Для успешного использования DTO-классов необходимо, чтобы имена полей DTO-класса совпадали с именами полей Entity-класса.

Пример entity-класса:

```
@Entity
@Table(name = "dst_inscontract")
@Getter
@Setter
@EntityInfo(name = "Договор страхования", brief = "ДогСтрах")
public class InsContract {

    @Id
    @Column(name = "id")
    @AttributeInfo(name = "Идентификатор", brief = "ИД")
    private Long id;

    @Column(name = "series")
    @AttributeInfo(name = "Серия", brief = "Серия")
    private String series;

    @Column(name = "num")
    @AttributeInfo(name = "Номер", brief = "Номер")
    private String num;

    @Column(name = "inssum")
    @AttributeInfo(name = "Страховая сумма", brief = "СтрахСумма")
    private BigDecimal insSum;

    @Column(name = "startdate")
    @AttributeInfo(name = "Дата начала", brief = "ДатаНачала")
    private Date startDate;

    @Column(name = "renew")
    @AttributeInfo(name = "Пролонгация", brief = "Пролонгация")
    private Boolean renew;
}
```

Пример dto-класса:

```
@Getter
@Setter
@NoArgsConstructor
public class InsContractDTO {
    private Long id;
    private String series;
    private String num;
    private BigDecimal insSum;
    private Date startDate;
    private Boolean renew;
}
```

Формат шаблона

Шаблон располагается в базе данных микросервиса в таблице cmn_object_template.

Поля таблицы cmn_object_template:

- id - идентификатор,
- entity_brief - сокращение сущности,
- brief - сокращение шаблона,
- structure - содержимое шаблона в формате json.

Формат поля structure:

```
{
    "СокращениеАтрибута1": "ЗначениеКонстантой",
    "СокращениеАтрибута2:calculate": "МетодДляРассчётаЗначенияАтрибута",
    "СокращениеАтрибута3": ...,
    ...
}
```

Где:

- СокращениеАтрибута - это brief из аннотации AttributeInfo.
- ЗначениеКонстантой - Строка, Целое, Дробное, Логическое значения.
- МетодДляРассчётаЗначенияАтрибута - это name из аннотации ObjectTemplateMehod для метода, реализующего вычисляемое значение (см. далее).

Пример:

```
{
    "Серия": "UL3",
    "ВалютаИД": 101,
    "Номер:calculate": "РассчитатьСледНомер",
    "СтрахСумма": 789456.12,
    "Пролонгация": true,
    "ДатаНачала:calculate": "РассчитатьДатуНачала",
    "СтрахПремия:calculate": "РассчитатьПремию",
    "ДатаОкончания:calculate": "РассчитатьДатуОкончания"
}
```

Методы для расчёта значений в шаблоне

Для методов, предназначенных для расчёта значений атрибутов, используется специальная аннотация

```
ObjectTemplateMethod(name = "Имя метода", description = "Описание метода")
```

Где:

- "Имя метода" используется в шаблоне.
- "Описание метода" в дальнейшем будет использоваться в пользовательском интерфейсе по настройке шаблонов.

Метод должен быть реализован в рамках ApplicationScoped бина.

Если для подобных методов создаётся отдельный бин, то он также должен быть аннотирован как @Unremovable, т.к. в коде не внедрения этого бина.

Формат метода:

```
@ObjectTemplateMethod(name = "Имя метода", description = "Описание метода")
public XXX getSomethig(Map<String, Object> contextParams) {
    return null;
}
```

Где:

- XXX - возвращаемый методом тип данных. Должен строго совпадать с типом поля атрибута.
- Map<String, Object> contextParams - обязательный параметр с контекстом исполнения метода (см. ниже).

Пример:

```
@ApplicationScoped
@Unremovable
public class TestBuildinMethods {

    @Inject
    AutoNumber autoNumber;

    @ObjectTemplateMethod(name = "РассчитатьСледНомер",
        description = "Рассчитывает следующий номер договора")
    public String getNextNumber(Map<String, Object> contextParams) {
        // Взять из контекста исполнения inProductVerId и
        // рассчитать номер очередного договора для данной версии продукта
        return (String) autoNumber.getNext(contextParams.get("inProductVerId"));
    }
}
```

Использование механизма шаблонов

Создание объекта по шаблону осуществляется при помощи вызова метода newInstance у бина ObjectBuilder.

```
public interface ObjectBuilder {
    <T> T newInstance(String entityBrief,
            Class<T> clazz,
            String templateBrief,
            Map<String, Object> contextParams) throws
        InstantiationException, IllegalAccessException, ObjectTemplateException;
}
```

Где:

- entityBrief - сокращение сущности, объект которой создаётся по шаблону.
- clazz - класс объекта (Entity-класс или DTO-класс).
- templateBrief - сокращение шаблона.
- contextParams - контекст (набор параметров), которые необходимо пробросить в методы, вычисляющие значения атрибутов.

Рассмотрим создание объекта на примере:

```
public InsContractDTO createContractByTemplate(
        String templateBrief, long insProductVerId) throws Exception {

    // Создание контекста исполнения методов. Контекст позволяет передать
    // в методы и между методами какие-либо значения
    Map<String, Object> params = new HashMap<>();

    // Пробросить в методы версию страхового продукта insProductVerId,
    // чтобы по ней рассчитать номер очередного договора
    // для данной версии страхового продукта
    params.put("insProductVerId", insProductVerId);

    // Вызов метода по созданию объекта. Создаём DTO-класс
    InsContractDTO result = builder.newInstance("ДогСтрах", InsContractDTO.class,
        templateBrief, params);

    // Присвоение значений остальным атрибутам, которые
    // не обрабатываются в шаблонизаторе
    result.setNote("Договорр создан по шаблону");

    // Вернуть DTO-объект
    return result;
}
```

Реализовать в следующих версиях

1. Для управления шаблонами в дальнейшем необходимо реализовать пользовательский интерфейс управления шаблонами микросервиса. Пока управление шаблонами осуществляется непосредственно через базу данных.
2. Продумать и при необходимости реализовать шаблонизацию сложного объекта, включающего в себя подчинённые объекты.

   Например, при создании экземпляра договора страхования создавать объекты страхования по договору и в свою очередь риски по объектам страхования.

   При более детальном рассмотрении конкретных примеров практическая ценность сложных шаблонов вызывает сомнения. Вернуться к этой теме после реализации структуры страхового продукта.