Уведомления – реализация.

Оглавление

[Общие положения. 1](#_Toc383590789)

[Сервис уведомлений. 2](#_Toc383590790)

[Профили системы и пользователей. 6](#_Toc383590791)

[Сервис получения каналов доставки использующий подсистему профилей 10](#_Toc383590792)

[Набора каналов доставки 11](#_Toc383590793)

[Канал отправки уведомления по электронной почте 13](#_Toc383590794)

[Канал отправки уведомления в папку «Входящие уведомления» 13](#_Toc383590795)

[Сервис формирования текста сообщения 14](#_Toc383590796)

[Сервис формирования ссылки 16](#_Toc383590797)

[Подсистема формирования уведомлений по событиям 16](#_Toc383590798)

[Периодическое задание формирующее уведомления по расписанию 21](#_Toc383590799)

[Задача подсистемы workflow для формирования уведомлений 23](#_Toc383590800)

[Доработка сервисов динамических групп и контекстных ролей 24](#_Toc383590801)

[Сервис поиска доменных объектов с помощью DOEL, запроса или класса 24](#_Toc383590802)

[Приложение 1. Состав работ и трудоемкость разработки. 27](#_Toc383590803)

# Общие положения.

Подсистема уведомлений состоит из следующих компонент:

1. Сервис уведомлений.
2. Набора каналов доставки.
3. Сервис формирования текста сообщения.
4. Сервис формирования ссылки.
5. Подсистема формирования уведомлений по событиям
6. Периодическое задание, формирующее уведомления по расписанию.
7. Задача подсистемы workflow для формирования уведомлений.

Схема подсистемы уведомлений предоставлена на рисунке 1.



Рис. 1. Схема подсистемы отправки уведомлений. Красные стрелки обозначают использование подсистемы, Синие внутренние обращения, зеленые обращения системы к внешним сервисам.

# Сервис уведомлений.

Сервис уведомлений представляет собой EJB со следующим интерфейсом:

**public** **interface** NotificationService {

/\*\*

\* Отправка уведомления после успешного завершения транзакции. Метод вызывается бизнес методами во время открытой транзакции. Реальная отправка происходит после удачной завершенной транзакции

\* **@param** notificationType

\* Тип уведомления

\* **@param** sender

\* Идентификатор персоны отправителя, может быть nullБ в этом случае подставится текущая персона

\* **@param** addresseeList

\* список адресатов. Может быть персона, группа, динамическая группа или контекстная роль

\* **@param** priority

\* приоритет сообщения. Влияет на отображение данного сообщения, но не влияет на очередность отправки

\* **@param** context

\* Контекст сообщения. Содержит информацию о объекте системы, относительно которой производится отправка

\* уведомления

\*/

**void** sendOnTransactionSuccess(String notificationType, Id sender, List<NotificationAddressee> addresseeList, NotificationPriority priority,

NotificationContext context);

/\*\*

\* Метод асинхронной отправки уведомления. Используется из метода sendOnTransactionSuccess.

\* Отправка производится независимо от результата транзакции.

\* **@param** notificationType

\* тип сообщения

\* **@param** sender

\* идентификатор персоны отправителя

\* **@param** addresseeList

\* список адресатов

\* **@param** priority

\* приоритет

\* **@param** context

\* контекст сообщения

\* **@return**

\*/

Future<Boolean> sendNow(String notificationType, Id sender, List<NotificationAddressee> addresseeList,

NotificationPriority priority,

NotificationContext context);

}

Особенность работы метода send такова, что он при выполнение формирует список сообщений в атрибутах транзакции, а реальное отправление осуществляется при успешном окончание транзакции при получение события beforeCompletion (Выбран метод beforeCompletion потому что есть подозрение что в методе afterCompletion ничего нельзя менять в базе, так как транзакция уже завершена, если это не подтвердится то необходимо использовать метод afterCompletion). Для этого необходимо применить сервис UserTransactionServiceImpl. Отправка в конце транзакции осуществляется в асинхронном режиме под правами системы путем вызова метода sendNow этого же EJB. Метод необходимо вызывать не напрямую из класса, а получив предварительно экземпляр EJB путем вызова метода SessionContext. getBusinessObject. Если при вызове метода send параметр sender равен null, то при вызове sendAsync передается параметр sender – идентификатор пользователя вызвавшего метод send. Данный идентификатор получается с помощью Сервиса CurrentUserAccessor и будет использоваться каналами для определения отправителя.

Адресатом сообщения могут быть следующие классы имплементирующие интерфейс NotificationAddressee: NotificationAddresseePerson, NotificationAddresseeGroup, NotificationAddresseeDynamicGroup, NotificationAddresseeContextRole. Метод sendNow раскрывает список адресатов в список персон и для каждой персоны получает список каналов, которые вычисляются исходя настройки каналов для соответствующей персоны и общих настроек системы. Список каналов для системы в целом и для отдельной персоны в частности хранятся в соответствующих профилях системы, которые будут описаны ниже. Далее сервис уведомлений вызывает метод send каждого полученного выше канала и передает в него параметры notificationType, personId, priority, context. Далее канал выполняет отправку сообщения исходя из собственной реализации.

Еще одной задачей сервиса уведомлений является сбор информации о всех каналах отправки уведомлений путем получения всех спринг бинов имплементирующих интерфейс NotificationChannel и создание реестра данных каналов по имени. Имя канала получается путем вызова метода NotificationChannel.getName.

При работе сервис определяет список каналов доставки конкретного сообщения с помощью спринг сервиса получения списка каналов для сообщения NotificationChannelSelector со следующим интерфейсом:

**public** **interface** NotificationChannelSelector {

/\*\*

\* Получение списка каналов для сообщения исходя из типа сообщения, адресата и приоритета

\* **@param** notificationType

\* тип сообщения

\* **@param** addressee

\* идентификатор персоны адресата

\* **@param** priority

\* приоритет сообщения

\* **@return**

\*/

List<String> getNotificationChannels(

String notificationType,

Id addressee,

NotificationPriority priority);

}

Параметр notificationType – это строковой идентификатор сообщения. По этому идентификатору каналы получают у сервиса формирования текста сообщения непосредственно текст сообщения.

Параметр priority – приоритет, является экземпляром перечисления NotificationPriority и может принимать три значения HIGH, NORMAL, LOW. Код перечисления:

**public** **enum** NotificationPriority {

*HIGH*,

*NORMAL*,

*LOW*

}

Приоритет используется каналами для установки визуального флага сообщения, а так же сервисом отправки уведомлений при получение списка каналов из профилей пользователя и системы.

Параметр context сообщения является экземпляром класса NotificationContext и содержит информацию о контекстных объектах для сообщения. Например, содержит информацию о доменном объекте документа, ссылку на который необходимо будет прикрепить к почтовому сообщению, или атрибуты которого необходимо будет отобразить в SMS сообщение. Контекст может содержать неограниченное количество идентификаторов доменных объектов и иных объектов, информация из которых может быть использована сервисом формирования текста сообщения. За содержание объектов в контексте и их имена отвечает код отправляющий сообщение. Уведомления одного типа всегда должны иметь одинаковое количество и одинаково названные объекты контекста. Код класса NotificationContext:

**public** **class** NotificationContext **implements** Dto{

**private** Map<String, Dto> contextObjects = **new** Hashtable<String, Dto>();

/\*\*

\* Добавление контекстного объекта

\* **@param** name

\* **@param** object

\*/

**public** **void** addContextObject(String name, Dto object) {

contextObjects.put(name, object);

}

/\*\*

\* Получение контекстного объекта

\* **@param** name

\* **@return**

\*/

**public** Dto getContextObject(String name) {

**return** contextObjects.get(name);

}

/\*\*

\* Получение имен всех контекстных объектов

\* **@return**

\*/

**public** Set<String> getContextNames() {

**return** contextObjects.keySet();

}

}

## Профили системы и пользователей.

Подсистема профилей системы и пользователей не входит в состав подсистемы отправки уведомлений, но используется для получения списка каналов. Интерфейс подсистемы профиля:

**public** **interface** ProfileService {

/\*\*

\* Получение профиля системы. Профиль содержит данные профиля без учета иерархии профилей. Предназначен для

\* редактирования системных профилей администраторами при его вызове должен создаваться AdminAccessToken

\* **@param** name имя профиля

\* **@return**

\*/

Profile getProfile(String name);

/\*\*

\* Получения профиля персоны. Профиль содержит данные профиля без учета иерархии профилей. Предназначен для

\* редактирования пользовательских профилей администраторами. При его вызове должен создаваться AdminAccessToken

\* **@param** personId

\* **@return**

\*/

Profile getPersonProfile(Id personId);

/\*\*

\* Сохранения профиля системы. Профиль содержит данные профиля без учета иерархии профилей. Предназначен для

\* редактирования системных профилей администраторами при его вызове должен создаваться AdminAccessToken

\*/

**void** setProfile(Profile profile);

/\*\*

\* Получение пользовательского профиля. Профиль содержит данные профиля пользователя с учетом иерархии профилей.

\* Предназначен для работы под провами простого пользователя

\* **@return**

\*/

PersonProfile getPersonProfile();

/\*\*

\* Сохранение пользовательского профиля. Профиль содержит данные профиля пользователя с учетом иерархии профилей.

\* Предназначен для работы под провами простого пользователя.

\* При сохранения профиля меняются данные только профиля пользователя. Данные системных профилей остаются не изменными.

\* **@param** profile

\*/

**void** setPersonProfile(PersonProfile profile);

}

Подсистема профилей предназначена для получения значения профиля по имени профиля для конкретного пользователя. Каждый пользователь имеет свой профиль со своими значениями. Например пользователь может выбрать язык, в котором ему необходимо присылать уведомления. Отличительной особенностью профиля в отличие от обычного доменного объекта являются:

* Неизвестное количество хранимых ключей. То есть каждая подсистема может хранить в профиле неограниченное значение профильных значений. Например подсистема GUI может хранить высоту и ширину окна и положения его на экране. Отсюда следует что в хранилище профилей не должно быть жестко зашито названия ключей профиля.
* Профили являются расширяемыми. То есть существует иерархия профилей. На самом нижнем уровне существует профиль пользователя. Профиль пользователя имеет родительский системный профиль, который в свою очередь так же имеет родительский. При получение пользовательского профиля сначала получается цепочка от самого нижнего до самого вышестоящего у которого атрибут родительский профиль не заполнен. Зачитываются атрибуты из профиля самого высокого уровня (корневого профиля) и полученные данные сохраняются в объекте PersonProfile. Далее тоже самое происходит по всей цепочки профилей с верху в низ, и на каждом этапе данные в PersonProfile дополняются данными из зачитываемого профиля, причем если атрибут профиля уже существует то более низкий профиль его перезаписывает. Так происходит до самого нижнего уровня, который является профилем пользователя.

Для хранения данных профиля следующие типы доменных объектов:

**profile** - базовый тип для профилей.

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| parent | Reference на тип profile | Идентификатор родительского профиля. В случае если атрибут равен null то запись является профилем родительского уровня. |

**system\_profile** – системный профиль. Является наследником типа PROFILE

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| Name | String | Имя профиля |

**person\_profile** –профиль пользователя. Является наследником типа PROFILE. Не имеет атрибутов. Выделен в отдельный тип для обеспечения ссылочной целостности и не возможности сослатся на тип system\_profile или profile из типа person. На данный тип создается ссылка в типе person

Модификация типа **person**

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| profile | Reference на тип person\_profile | Ссылка на профиль пользователя |

**profile\_value** - базовый тип для хранения одного значения профиля

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| profile | Reference на тип profile | Ссылка на профиль пользователя |
| key | String | Имя ключа профиля |
| readonly | Boolean | Флаг невозможности переопределить значение профиля конечным пользователем. Поля профиля помеченные данным флагом могу правится только администраторами. Пользователь не может переопределить в своем пользовательском профиле данный атрибут профиля, если в системном профиле атрибут помечен как readonly |

**profile\_value\_long –** тип для хранения численных значений

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| value | Long | Значение профиля |

**profile\_value\_string –** тип для хранения строковых значений

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| value | String | Значение профиля |

**profile\_value\_boolean –** тип для хранения булевых значений

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| value | Boolean | Значение профиля |

**profile\_value\_date –** тип для хранения значений типа дата

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| value | Date | Значение профиля |

**profile\_value\_locale –** тип для хранения значений типа ссылка на справочник локалей. Для хранения в профиле ссылочных полей, по мере необходимости, надо будет создавать подобные типы с разными типами на которые ссылаются поле value. Подсистема профилей должна создаватся с таким расчетом, что таких типов будет много и они должны будут зачитываться из конфигурации при старте системы. При сохранение в профиль значения типа ReferenceValue необходимо будет из конфигурации найти соответствующий сохраняемому типу доменный объект из конфигурации и создать его запись.

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| value | Reference на тип locale | Значение профиля |

Работа с профилем осуществляется в двух режимах.

1. Режим администратора. В этом режиме используются методы getProfile, getPersonProfile(Id personId) и setProfile. При этом получаются и редактируются записи непосредственно полученного профиля без учета иерархии профилей. Пользователь не входящий в группу администраторов при вызове данных методов должен получать ошибку.
2. Режим пользователя. Для этого предназначены методы getPersonProfile() и setPersonProfile(PersonProfile profile). При получение профиля методом getPersonProfile() зачитываются данные всех профилей в иерархии профилей находящихся выше по уровню относительно текущего пользовательского профиля. Данные всех профилей собираются в объект PersonProfile причем профили более низкого уровня перезатирают значения атрибутов профилей более высокого уровня, этим реализовывается возможность переопределения значений системных профилей. Если в профиле более высокого уровня установлен атрибут readonly то данное значение не может быть переопределено профилем пользователя. При сохранение пользовательского профиля сохраняются данные только в профиль пользователя. В случае если сохраняемое значение профиля пользователя отличается от значения в родительском профиле с учетом иерархии профилей то атрибут сохраняется в пользовательском профиле. Если атрибут не отличается от значения в родительском профиле то доменный объект хранящий данный атрибут удаляется из пользовательского профиля.

Для передачи данных о флаге readonly используются наследники класов StringValue, LongValue, DateValue, BooleanValue и ReferenceValue соответсвенно ProfileStringValue, ProfileLongValue, ProfileDateValue, ProfileBooleanValue и ProfileReferenceValue. Каждый из этих наследников должен имплементить интерфейс ProfileValue:

**public** **interface** ProfileValue {

/\*\*

\* Получение флага только для чтения

\* **@return**

\*/

**boolean** isReadOnly();

/\*\*

\* Установка флага только для чтения

\* **@param** readOnly

\*/

**void** setReadOnly(**boolean** readOnly);

}

# Сервис получения каналов доставки использующий подсистему профилей

Задача сервиса следующая: необходимо позволять включать и выключать уведомления в зависимости от канала доставки, типа сообщения и приоритета.

Сценарий работы будет следующий – пользователь для всех типов уведомлений для канала устанавливает нужный приоритет сообщений, для определенных типов сообщения того же канала уточняет приоритет, который он получал бы через канал. Например. Пользователь по каналу SMS получает сообщения только с наивысшим приоритетом. Исключение сообщение о регистрации документа. Такие сообщения пользователь хочет получать с приоритетом NORMAL.

Исходя из такого сценария необходимо хранить в профиле ключи следующего вида:

NOTIFICATION.CHANNEL.NOTIFICATION\_TYPE=PRIORITY

Где:

NOTIFICATION – константа, добавляется всегда

CHANNEL - имя канала доставки

NOTIFICATION\_TYPE – тип сообщения

PRIORITY – приоритет сообщения

В качестве NOTIFICATION\_TYPE допускается использовать символ \*, что означает все типы сообщений.

В качестве PRIORITY могут быть следующие строковые константы:

DISABLE – сообщения отключены;

HIGH – доставляются сообщения только с приоритетом HIGH;

NORMAL – доставляются сообщения только с приоритетом HIGH и NORMAL;

LOW – доставляются все сообщения.

Алгоритм работы следующий. Подсистема отправляет сообщение. Сначала сервис получения каналов в профиле ищет ключ равный NOTIFICATION.CHANNEL.NOTIFICATION\_TYPE, где вместо CHANNEL и NOTIFICATION\_TYPE данные и сообщения. Если такой находится то принимается решение исходя из приоритета сообщения и приоритета из профиля по алгоритму описанному выше. Если ключ не найден, то ищется ключ NOTIFICATION.CHANNEL.\*. Если ключ найден то принимается решение исходя из приоритета сообщения и приоритета из профиля. Если не найден то ищется ключ NOTIFICATION.\* и так же принимается решение как и в первых двух случаях. Если не найден не один из вариантов, то сообщение не отправляется.

Внимание, при данном подходе надо иметь ввиду, что в случае, если указать в системном профиле более детализированное включение сообщения, то есть указать и канал и способ доставки, то более общая настройка в профиле пользователя (содержащая в качестве типа сообщения звездочку) не перекроет системную настройку, и надо на уровне пользовательского профиля перекрыть именно детализированную настройку с определенным типом сообщения и определенным каналом. Надо иметь это ввиду при проектирование пользовательского интерфейса.

Профиль так же используется сервисом формирования текста сообщения для получения языка текста сообщения для пользователя.

# Набора каналов доставки

Канал доставки представляет из себя класс имплементирующий интерфейс NotificationChannelHandle и аннотированный с помощью анотации NotificationChannel.

**public** **interface** NotificationChannelHandle {

/\*\*

\* Отправка сообщения с помощью канала

\* **@param** notificationType

\* тип сообщения

\* **@param** senderId

\* идентификатор персоны отправителя. Может быть null в случае если отправитель система

\* **@param** addresseeId

\* идентификатор персоны адресата

\* **@param** priority

\* приоритет

\* **@param** context

\* контекст сообщения

\*/

**void** send(String notificationType, Id senderId, Id addresseeId, NotificationPriority priority, NotificationContext context);

}

**public** **@interface** NotificationChannel {

/\*\*

\* имя канала

\* **@return**

\*/

String name();

/\*\*

\* Описание канала

\* **@return**

\*/

String description();

}

При старте сервера специальный сервис NotificationChannelLoader просматривает все классы аннотированные данной аннотацией и создает реестр каналов, ключом элемента реестра является строка возвращаемая методом name () аннотации. Интерфейс сервиса NotificationChannelLoader:

**public** **interface** NotificationChannelLoader {

/\*\*

\* Получение списка имен всех каналов отправки уведомлений

\* **@return**

\*/

List<String> getNotificationChannelNames();

/\*\*

\* Получение информации о канале доставки

\* **@param** channelName

\* **@return**

\*/

NotificationChannelInfo getNotificationChannelInfo(String channelName);

/\*\*

\* Получение обработчика канала отправки уведомления по его имени

\* **@param** channelName

\* **@return**

\*/

NotificationChannelHandle getNotificationChannel(String channelName);

}

Сервис отправки уведомлений вызывает метод send канала и передает информацию об уведомление каналу. Далее канал выполняет отправку уведомления по своему персональному алгоритму. Платформа в базовой поставке предоставляет два канала отправки уведомлений: канал папки «Входящие уведомления» и канал отправки по электронной почте. Все каналы могут пользоваться публичными сервисами платформы (EJB), внутренними сервисами платформы (DAO) в том числе сервисами формирования текста уведомления и сервисом формирования ссылок.

## Канал отправки уведомления по электронной почте

Данный канал использует Spring Mail для отправки сообщений по электронной почте. Канал использует сервис формирования текста уведомлений для получения текста и заголовка письма. Все настройки требуемые для обеспечения работы канала (smtp сервер, логин, пароль пользователя SMTP сервера, адрес отправителя по умолчанию) должны в server.properties. Канал разрабатывается таким образом, чтобы можно было его переиспользовать на другом сервере, передав задание на отправку уведомления с помощью сервиса распределенных задач, основанном на JMS.

## Канал отправки уведомления в папку «Входящие уведомления»

Данный канал создает запись доменного объекта с типом notifications. GUI периодически опрашивает записи в таблице notifications и отображает их в интерфейсе. Тип содержит следующие атрибуты:

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| from | Reference | Идентификатор персоны отправителя сообщения |
| to | Reference | Идентификатор персоны адресата сообщения |
| subject | String | Заголовок сообщения |
| body | String | Текст сообщения |
| priority | String | Приоритет сообщения |
| new | Boolean | Флаг нового сообщения, после открытия данного сообщения GUI сбрасывает этот флаг в 0. Используется для выделения новых сообщений на интерфейсе и оптимизации запросов новых сообщений |

Для формирования текста сообщения канал использует сервис генерации текста сообщения.

# Сервис формирования текста сообщения

Сервис формирования текста сообщения представляет собой spring бин и предназначен для формирования текста сообщения по типу сообщения, идентификатору адресата и контексту. Шаблон сообщения хранится в специальном доменном объекте notification\_text в виде скрипта на языке шаблонов Apache Velocity (язык шаблонов может быть изменен разработчиком подсистемы, при выборе необходимо опираться на максимальную простоту и понятность использования языка шаблонов). Интерфейс сервиса:

**public** **interface** NotificationTextFormer {

/\*\*

\* Метод формирует текст сообщения по типу сообщения и каналу

\* **@param** notificationType

\* тип сообщения

\* **@param** notificationPart

\* имя фрагмента сообщения. Используется если сообщение состоит из нескольких частей, например почтовое

\* сообщение состоит из заголовка и тела сообщения

\* **@param** addressee

\* адресат сообщения

\* **@param** locale

\* Идентификатор локали

\* **@param** channel

\* имя канала

\* **@param** context

\* контекст сообщения

\* **@return**

\*/

String format(String notificationType, String notificationPart, Id addressee, Id locale, String channel,

NotificationContext context);

/\*\*

\* Формирование всех частей текста уведомления по его типу

\* **@param** notificationType

\* тип сообщения

\* **@param** addressee

\* идентификатор персоны адресата

\* **@param** locale

\* Идентификатор локали

\* **@param** channel

\* имя канала

\* **@param** context

\* контекст сообщения

\* **@return**

\*/

List<NotificationText> format(String notificationType, Id addressee, Id locale, String channel,

NotificationContext context);

}

Для хранения шаблонов сообщений сервис использует доменные объекты типа notification\_text со следующими полями:

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| notification\_type | String | Тип сообщения |
| notification\_part | String | Имя фрагмента сообщения |
| locale | Reference | Идентификатор системного справочника языков |
| channel | String | Имя канала |
| notification\_text | String | Текст сообщения на языке шаблонов сообщения |

При формирование сообщения сервис получает запись notification\_text по переданным параметрам. Далее создает контекст движка языка шаблонов и внедряет туда следующие объекты:

* Объект сессии ru.intertrust.cm.core.tools.Session под именем session.
* Объекты из параметра context, каждый объект под именем с которым этот объект добавляли в context. При этом все объекты типа DomainObject оборачиваются классом DomainObjectAccessor.
* Все спринговые бины платформы под именами с какими они зарегистрированы в Spring контексте платформы.

Далее выполняется скрипт на языке шаблонов и полученный результат возвращается. При формирование сообщения необходимо учитывать часовой пояс пользователя, которому отправляются сообщения при формирование дат со временем в тексте сообщения. Часовой пояс пользователя хранится в профиле пользователя.

Для работы сервиса используется системный справочник языков locale, который содержит следующие атрибуты:

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| name | String | Наименование языка |

# Сервис формирования ссылки

Сервис формирования ссылок представляет из себя spring бин со следующим интерфейсом:

public interface UrlFormer {

/\*\*

\* Формирует url к объекту системы. Формирование url производится в контексте определенного клиента платформы.

\* @param clientName

\* имя клиента

\* @param addressee

\* идентификатор персоны адресата

\* @param objectId

\* идентификатор доменного объекта

\* @return

\*/

URL getUrl(String clientName, Id addressee, Id objectId);

Сервис конфигурируется с помощью XML конфигурации следующего вида:

<url-config name=”GWT-CLIENT”>

<![CDATA[Текст скрипта на языке разметки шаблонов]]>

</url-config>

При вызове метода getUrl ищется конфигурация name равным переданному параметру. У полученного доменного объект берется текст внутри тэга, которое передается движку языка шаблонов Apache Velocity (язык шаблонов может быть изменен разработчиком подсистемы, в первую очередь необходимо учитывать удобство написания конечных шаблонов). Так же в движок внедряются все спринговые бины системы под теми же именами как они зарегистрированы в spring контексте, и также, передается обертки DomainObjectAccessor над переданным параметром objectId и addressee под именами domainObject и addressee. Сформированная таким образом строка возвращается в качестве результата работы метода.

# Подсистема формирования уведомлений по событиям

Платформа предоставляет три точку расширения для формирования уведомлений. Точка расширения реагирует на создание, смену статуса, изменение и удаление доменного объекта. Точка расширения имеет настройки, которые выносятся в конфигурацию системы на уровне ядра. Конфигурация точки расширения:

|  |  |  |
| --- | --- | --- |
| **Тэг/атрибут** | **Тип** | **Описание** |
| notification | Тэг | Тэг конфигурации верхнего уровня |
| name | Атрибут, String | Имя конфигурации |
| notification-type | Тэг, String | Тип уведомления |
| addressee | Тэг, FindObjectsConfig | Тег описывающий адресатов. Может содержать внутри теги получения адресатов с помощью запроса, с помощью контектной роли или контекстной группы, а также может быть указан класс, с помощью которого получается список адресатов. |
| find-person | Тэг, FindObjectsConfig | Тэг описывающий поиск персон с помощью запроса, вщуд или класса см. «Сервис поиска доменных объектов с помощью класса, запроса или DOEL выражения» |
| query | Тэг, String | Запрос получения адресатов. Принимает параметр текущего доменного объекта в параметре {0} Запрос возвращает список идентификаторов персон |
| class-name | Тэг, String | Имя класса определяющего список адресатов. Класс должен имплементировать интерфейс NotificationAddresseeReceiver |
| doel | Тэг, String | DOEL выражение определяющеee список адресатов. Класс должен имплементировать интерфейс NotificationAddresseeReceiver |
| context-role | Тэг, string | Имя контекстной роли относительно измененного доменного объекта |
| dynamic-group | Тэг, string | Имя динамической группы относительно измененного доменного объекта |
| trigger | Тэг | Описание события, по возникновению которого отправляются уведомления. Событие сожжет быть описано непосредственно здесь, или ссылается на именованное событие уровня конфигурации |
| name | Атрибут, String | Имя именованного события. Если атрибут заполнен то остальное содержание тэга игнорируется. |
| domain-object-type | Тэг, String | Тег содержит имя типа доменного объекта, при изменение которого возникает событие. |
| event | Тэг, String | Тег содержит тип события. Может быть CREATE, CHANGE, CHANGE\_STATUS, DELETE |
| fields | Тэг | Тег описывающий имена полей, изменение которых приводит к возникновению события (проверяется для типа события CHANGE) |
| field | Тэг, String | Имя поля в списке полей, может быть множество полей. Если нет ни одного поля то при любом изменение доменного объекта возникает событие |
| statuses | Тэг | Тег описывающий имена статусов, при установки которых возникает событие (проверяется для типа события CHANGE-STATUS) |
| status | Тэг, String | Имя статуса в списке статусов |
| class-name | Тэг, String | Имя класса, отвечающего за проверку возникновения события. Класс должен имплементировать интерфейс Trigger |
| conditions-script | Тэг, String | JavaScript результат выполнения которого определяет возникновение события. В скрипт внедряется информация о доменном объекте и измененных полях |

Для описания независимого события и возможности сослаться на него в конфигурации добавляется описание именованного события. Конфигурация именованного события:

|  |  |  |
| --- | --- | --- |
| **Тэг/атрибут** | **Тип** | **Описание** |
| named-trigger | Тэг | Тэг конфигурации верхнего уровня |
| name | Атрибут, String | Имя конфигурации |
| trigger | Тэг | Описание события, по возникновению которого отправляются уведомления. Событие сожжет быть описано непосредственно здесь, или ссылается на именованное событие уровня конфигурации |
| name | Атрибут, String | Имя именованного события. Если атрибут заполнен то остальное содержание тэга игнорируется. |
| type-name | Тэг, String | Тег содержит имя типа доменного объекта, при изменение которого возникает событие. |
| event | Тэг, String | Тег содержит тип события. Может быть CREATE, CHANGE, CHANGE\_STATUS, DELETE |
| fields | Тэг | Тег описывающий имена полей, изменение которых приводит к возникновению события (проверяется для типа события CHANGE) |
| field | Тэг, String | Имя поля в списке полей, может быть множество полей. Если нет ни одного поля то при любом изменение доменного объекта возникает событие |
| statuses | Тэг | Тег описывающий имена статусов, при установки которых возникает событие (проверяется для типа события CHANGE-STATUS) |
| status | Тэг, String | Имя статуса в списке статусов |
| class-name | Тэг, String | Имя класса, отвечающего за проверку возникновения события. Класс должен имплементировать интерфейс TriggerService |
| script | Тэг, String | JavaScript результат выполнения которого определяет возникновение события. В скрипт внедряется информация о доменном объекте и измененных полях |

Интерфейс EventTrigger

**public** **interface** EventTrigger {

/\*\*

\* Метод определяющий факт возникновения события

\* **@param** event

\* тип события CREATE, CHANGE, CHANGE\_STATUS, DELETE

\* **@param** domainObject

\* длменный объект по которому произошло событие

\* **@param** changedFields

\* измененные поля

\* **@return** Возвращается флаг сработал триггер или нет

\*/

**boolean** isTriggered(String eventType, DomainObject domainObject, List<FieldModification> changedFields);

/\*\*

\* Получение списка имен триггеров сработавших на изменение доменного объекта.

\* **@param** domainObject

\* длменный объект по которому произошло событие

\* **@param** changedFields

\* измененные поля

\* **@return** Возвращается список сработавших триггеров, если не сработал не один триггер возвращается пустой список

\*/

List<String> getTriggeredEvents(DomainObject domainObject, List<FieldModification> changedFields);

}

Интерфейс NotificationAddresseeReceiver

**public** **interface** NotificationAddresseeReceiver {

/\*\*

\* Метод возвращает список адресатов

\* **@param** domainObject доменный объект по которому произошло событие

\* **@return**

\*/

List<NotificationAddressee> getNotificationAddressee(DomainObject domainObject);

}

XSD тип тэга trigger в теге named-trigger и notification является одним и тем же XSD типом. Для определения условия возникновения события необходимо создать спринг сервис (Сервис событий), принимающий на вход конфигурацию триггера и измененный доменный объект. Данный сервис будет использоваться как сервисом формирования уведомлений, так и иными сервисами, реагирующие на события. Интерфейс сервиса:

**public** **interface** TriggerService {

/\*\*

\* Метод проверки возникновения события

\* **@param** event

\* тип события

\* **@param** domainObject

\* доменный объект

\* **@param** changedFields

\* измененные поля

\* **@return**

\*/

**boolean** isTriggered (String eventType, DomainObject domainObject, List<FieldModification> changedFields);

}

Точки расширения при возникновение события проверяет конфигурацию уведомлений и при совпадение всех условий описанных в тэге trigger отправляет уведомление с типом notification-type пользователям описанным в тэге addressee. В контекст сообщения добавляется текущий измененный доменный объект. Отправляют уведомления полученным персонам с помощью сервиса NotificationService.

# Периодическое задание формирующее уведомления по расписанию

Для отправки уведомлений по расписанию в платформе реализовано периодическое задание для отправки уведомлений согласно настройке этого периодического задания. Данное задание является периодическим заданием платформы то есть имплементирует интерфейс ScheduleTaskHandle и аннотирован с помощью @ScheduleTask. Данное задание имеет тип SheduleType.Multipliable то есть можно создать множество заданий этого типа с разными параметрами. Параметр задания это класс имплементирующий интерфес ScheduleTaskParameters следующего вида:

**public** **class** NotificationTaskConfig **implements** ScheduleTaskParameters {

**private** **static** **final** **long** *serialVersionUID* = 2618754657538579112L;

/\*\*

\* Описание способа получения доменных объектов

\*/

@Attribute

**private** FindObjectsConfig findDomainObjects;

/\*\*

\* Описание способа получения персон

\*/

@Attribute

**private** FindObjectsConfig findPersons;

/\*\*

\* Тип сообщения

\*/

@Attribute

**private** String notificationType;

/\*\*

\* Приоритет сообщения

\*/

@Attribute

**private** NotificationPriority notificationPriority;

/\*\*

\* Флаг типа формирования сообщений, относительно доменного объекта или отностительно персоны

\*/

@Attribute

**private** NotificationTaskMode taskMode;

**public** String getNotificationType() {

**return** notificationType;

}

**public** **void** setNotificationType(String notificationType) {

**this**.notificationType = notificationType;

}

**public** NotificationPriority getNotificationPriority() {

**return** notificationPriority;

}

**public** **void** setNotificationPriority(NotificationPriority notificationPriority) {

**this**.notificationPriority = notificationPriority;

}

**public** FindObjectsConfig getFindDomainObjects() {

**return** findDomainObjects;

}

**public** **void** setFindDomainObjects(FindObjectsConfig findDomainObjects) {

**this**.findDomainObjects = findDomainObjects;

}

**public** FindObjectsConfig getFindPersons() {

**return** findPersons;

}

**public** **void** setFindPersons(FindObjectsConfig findPersons) {

**this**.findPersons = findPersons;

}

**public** NotificationTaskMode getTaskMode() {

**return** taskMode;

}

**public** **void** setTaskMode(NotificationTaskMode taskMode) {

**this**.taskMode = taskMode;

}

}

Периодическое задание работает следующим образом. В случае если атрибут taskMode равен BY\_DOMAIN\_OBJECT - при запуске вызывается метод getFindDomainObjects и с помощью сервиса FindObjectsService получается список доменных объектов. Далее вызывается метод getFindPersons и с помощью сервиса FindObjectsService получаются персоны для каждого полученного на первом шаге доменного объекта. Выполняется формрование уведомлений каждому пользователю с помощью метода NotificationService.sendOnTransactionSuccess. В метод передается тип сообщения полученное методом getNotificationType и приоритет сообщения, полученное методом getNotificationPriority. В контекст добавляется доменный объект под именем document. В случае если атрибут taskMode равен BY\_PERSON то происходит сначала поиск пользователей с помощью конфигурации поиска описанного в атрибуте getFindPersons, а затем получается список доменных объектов для каждого пользователя с помощью метода getFindDomainObjects. Далее отправка происходит как и в первом варианте. В контекст добавляется доменный объект под именем document. Для облегчения создания подобных задач можно воспользоваться импортом из CSV файлов, параметры в этом случае передаются в виде XML.

# Задача подсистемы workflow для формирования уведомлений

Для отправки уведомлений из процессов необходимо создать класс сервиса workflow, который выполняет отправку сообщений с помощью метода NotificationService.transactionSend. Класс должен настраиваться следующими полями:

* addressee – строка в виде xml, по которой формируется список адресатов. Для облегчения формирования данной строки необходимо создать метод session.getNotificationAddressee(…)
* context – строковой идентификатор основного доменного объекта процесса. Данные доменный объект будет передан в качестве контекста сообщения под именем document
* notificationType – строка, тип сообщения
* notificationPriority – строка, одно из значений HIGH, NORMAL, LOW.

При получение управления сервис берет из полей задачи необходимые для отправки данные, и формирует уведомления используя метод NotificationService.transactionSend.

Для облегчения формирования списка адресатов в классе сессии необходимо создать пять методов: четыре метода по созданию каждого типа адресата (createPersonAddressee, createGroupAddreseee, createDynamicGroupAddressee, createContextRoleAddressee), и один метод принимающий на вход массив адресатов, а на выходе xml строка, которая далее передается в поле сервиса. Сервис выполняет обратную задачу десериализации XML в массив адресатов, и далее отправка этим адресатам сообщений.

# Доработка сервисов динамических групп и контекстных ролей

В рамках задач рассылки уведомлений требуется расширить функциональность сервиса формирования динамических групп DynamicGroupService и сервиса формирования контекстных ролей PermissionService. В данных сервисах требуется добавить метод получения состава персон, исходя из имени группы или роли и идентификатора контекстного доменного объекта. Сигнатура методов данных сервисов:

DynamicGroupService{

…

List<Id> getPersons(Id contextId, groupName);

}

PermissionService {

…

List<Id> getPersons(Id contextId, roleName);

}

В обоих методах возвращается список идентификаторов персон, соответствующих динамической группе или контекстной роли.

# Сервис поиска доменных объектов с помощью DOEL, запроса или класса

При разработки подсистемы отправки уведомлений периодически возникает потребность получить список идентификаторов доменных объектов с помощью одного из способов: с помощью DOEL выражения, с помощью запроса, с помощью класса. Данная потребность возникает достаточно часто, и по этому необходимо создать сервис, выполняющий эти действия. Сервис представляет собой spring сервис со следующим интерфейсом:

**public** **interface** DomainObjectFinderService {

/\*\*

\* Метод получения списка идентификаторов доменных объектов по одному из параметров: запроса, класса или DOEL

\* выражения

\* **@param** getObjectsConfig

\* Параметр в котором содержится или имя класса или запрос или doel выражение

\* **@param** contextDomainObjectId

\* идентификатор доменного объекта, относительно которого производятся вычисления. Идентификатори

\* передается в запрос в параметре номер 1 ({1}), а в класс как параметр метода получения списка

\* объектов.

\* **@return**

\*/

List<Id> findObjects(FindObjectsConfig getObjectsConfig, Id contextDomainObjectId);

}

В качестве параметра методу findObjects передается экземпляр класса FindObjectsConfig, а в нем есть поле типа интерфейс FindObjectsType, которое может быть FindObjectsDoelConfig, FindObjectsQueryConfig, FindObjectsClassConfig. Данные классы могут мапится на XML благодаря классу FindObjectsConfig, в котором с помощью аннатаций описаны теги XML на которые производится маппинг XML файла.

**public** **class** FindObjectsConfig **implements** Dto{

@ElementUnion({

@Element(name="class-name", type=FindObjectsClassConfig.**class**),

@Element(name="doel", type=FindObjectsDoelConfig.**class**),

@Element(name="query", type=FindObjectsQueryConfig.**class**)

})

**private** FindObjectsType findObjectType;

**public** FindObjectsType getFindObjectType() {

**return** findObjectType;

}

**public** **void** setFindObjectType(FindObjectsType findObjectType) {

**this**.findObjectType = findObjectType;

}

}

При работе сервиса, если на вход подается FindObjectsConfig содержащий FindObjectsClassConfig то создается экземпляр класса, имя которого указано внутри FindObjectsClassConfig. Этот класс жолжен имплементить интерфейс DomainObjectFinder:

/\*\*

\* Интерфейс получения списка идентификаторов доменных объектов относительно контекстного доменного объекта

\* **@author** larin

\*

\*/

**public** **interface** DomainObjectFinder {

/\*\*

\* Метод получения списка идентификатолв доменных объектов относительно контекстного доменного объекта

\* **@param** contextDomainObjectId

\* **@return**

\*/

List<Id> findObjects(Id contextDomainObjectId);

}

В случае если передается FindObjectsQueryConfig то выполняется содержащийся в нем запрос а идентификатор доменного объекта передается в качестве параметра с номером 1.

В случае если передается FindObjectsDoelConfig то выполняется DOEL выражение, содержащееся внутри класса, относительно переданного контекстного доменного объекта.

# Приложение 1. Состав работ и трудоемкость разработки.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task Name | Duration | Start | Finish | Predecessors | Resource Names |
| **Подсистема отпраки уведомлений** | **20 days** | **Tue 25.02.14** | **Mon 24.03.14** |  |  |
| Написание всех интерфейсов | 1 day | Tue 25.02.14 | Tue 25.02.14 |  | Разработчик 1 |
| Доработка сервиса динамических групп и контекстных ролей | 2 days | Wed 26.02.14 | Thu 27.02.14 | 2 | Разработчик 1 |
| Сервис поиска доменных объектов с помощью класса, запроса или DOEL выражения | 2 days | Fri 28.02.14 | Mon 03.03.14 | 3 | Разработчик 1 |
| Сервис получения списка каналов для сообщения (упрощенный вариант без использования профиля) | 1 day | Tue 04.03.14 | Tue 04.03.14 | 4 | Разработчик 1 |
| Сервис отправки уведомлений | 2 days | Wed 05.03.14 | Thu 06.03.14 | 5 | Разработчик 1 |
| Доработка классов интеграции с движком процессов Activiti | 2 days | Fri 07.03.14 | Mon 10.03.14 | 6 | Разработчик 1 |
| Сервис формирования событий | 3 days | Wed 26.02.14 | Fri 28.02.14 | 2 | Разработчик 2 |
| Подсистема формирования уведомлений по событиям | 3 days | Mon 03.03.14 | Wed 05.03.14 | 8 | Разработчик 2 |
| Канал доставки EMAIL | 2 days | Thu 06.03.14 | Fri 07.03.14 | 9 | Разработчик 2 |
| Канал доставки «Папка входящие уведомления» Включая разработку интерфейса отображения уведломления и окна сообщения о пришедшем уведомлении | 5 days | Mon 10.03.14 | Fri 14.03.14 | 10 | Разработчик 2 |
| Сервис формирования текста сообщения | 3 days | Wed 26.02.14 | Fri 28.02.14 | 2 | Разработчик 3 |
| Сервис формирования ссылки | 2 days | Mon 03.03.14 | Tue 04.03.14 | 12 | Разработчик 3 |
| Периодическое задание формирующее уведомление по рассписанию | 5 days | Wed 05.03.14 | Tue 11.03.14 | 13 | Разработчик 3 |
| Подсистема профилей | 3 days | Wed 12.03.14 | Fri 14.03.14 | 14 | Разработчик 3 |
| Сервис получения списка каналов для сообщения (полный вариант с использованием профиля) | 3 days | Mon 17.03.14 | Wed 19.03.14 | 15 | Разработчик 3 |
| Тестовые конфигурации и интеграционные тесты | 5 days | Tue 11.03.14 | Mon 17.03.14 | 7 | Разработчик 1 |
| Отладка и исправление ошибок | 3 days | Thu 20.03.14 | Mon 24.03.14 | 16 | Разработчик 1;Разработчик 2;Разработчик 3 |

