МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«Новосибирский национальный исследовательский государственный университет»

(Новосибирский государственный университет)

Структурное подразделение Новосибирского государственного университета –

Высший колледж информатики Университета

КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

**Разработка модуля поддержки технологии SpringDataRest**

**для библиотеки Restler**

Руководитель Жидков А. П.

ООО «Эксельсиор» «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2016 г.

Студент V курса, Руденко О. В.

гр. 115а «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2016 г.

Новосибирск

2016

СОДЕРЖАНИЕ

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ 4](#_Toc450654004)

[ВВЕДЕНИЕ 6](#_Toc450654005)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 7](#_Toc450654006)

[2 ПОСТАНОВКА ЗАДАЧИ 9](#_Toc450654007)

[2.1 Описание поставленной задачи 9](#_Toc450654008)

[2.2 Описание используемых средств 9](#_Toc450654009)

[3 АНАЛИЗ ИМЕЮЩИХСЯ АНАЛОГОВ 11](#_Toc450654010)

[4 ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ 12](#_Toc450654011)

[5 НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ 14](#_Toc450654012)

[5.1 Основные нефункциональные требования 14](#_Toc450654013)

[5.2 Требования к параметрам оборудования 14](#_Toc450654014)

[6 ИСПОЛЬЗУЕМЫЕ ИНСТРУМЕНТЫ И ТЕХНОЛОГИИ 15](#_Toc450654015)

[7 РЕШЕНИЕ ПОСТАВЛЕННОЙ ЗАДАЧИ 16](#_Toc450654016)

[7.1 Описание архитектуры Restler 16](#_Toc450654017)

[7.2 Проектирование модуля 19](#_Toc450654018)

[*7.2.1* *Входные и выходные данные* 19](#_Toc450654019)

[*7.2.2* *Общая модель модуля* 19](#_Toc450654020)

[*7.2.3* *Описание классов и интерфейсов* 21](#_Toc450654021)

[*7.2.3.1* *Главный пакет модуля* 21](#_Toc450654022)

[*7.2.3.2* *Пакет methods* 23](#_Toc450654023)

[*7.2.3.3* *Пакет calls* 26](#_Toc450654024)

[*7.2.3.4* *Пакет proxy* 28](#_Toc450654025)

[7.3 Реализация модуля 29](#_Toc450654026)

[*7.3.1* *Общее описание реализации модуля* 29](#_Toc450654027)

[*7.3.2* *Создание и изменение ресурса со вложенными ресурсами* 32](#_Toc450654028)

[7.4 Анализ и оптимизация программного кода модуля 35](#_Toc450654029)

[*7.4.1* *Создание тестового сценария* 35](#_Toc450654030)

[*7.4.2* *Измерение времени выполнения и анализ результатов* 36](#_Toc450654031)

[7.5 Тестирование модуля 39](#_Toc450654032)

[*7.5.1* *Тестовый сервер* 39](#_Toc450654033)

[*7.5.2* *Тестовый клиент* 40](#_Toc450654034)

[*7.5.3* *Структура тестового проекта* 41](#_Toc450654035)

[*7.5.4* *Maven плагин для автоматизации запуска сервера* 42](#_Toc450654036)

[ЗАКЛЮЧЕНИЕ 43](#_Toc450654037)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 44](#_Toc450654038)

[ПРИЛОЖЕНИЕ А 46](#_Toc450654039)

[ПРИЛОЖЕНИЕ Б 48](#_Toc450654040)

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

* Сервер – программный компонент вычислительной системы, выполняющий сервисные (обслуживающие) функции по запросу клиента, предоставляя ему доступ к определённым ресурсам или услугам.
* Клиент – это аппаратный или программный компонент вычислительной системы, посылающий запросы серверу.
* Модуль библиотеки Restler – набор классов, реализующих определенные интерфейсы ядра Restler, целью которых является расширение функциональности библиотеки.
* SpringData-хранилище – интерфейс, предоставляющий необходимые методы для работы, связанной с хранением ресурса и его поиском.
* SpringData-ресурс – объект, представляющий информацию, хранящуюся в базе данных.
* Уникальный идентификатор ресурса – идентификатор, однозначно определяющий ресурс в хранилище. По уникальному идентификатору можно найти ресурс.
* Framework (фреймворк) – программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.
* HTTP (англ. HyperText Transfer Protocol – «протокол передачи гипертекста») – протокол прикладного уровня передачи данных (изначально — в виде гипертекстовых документов в формате HTML, в настоящий момент используется для передачи произвольных данных).
* Веб-служба (англ. web service) – идентифицируемая веб-адресом программная система со стандартизированными интерфейсами.
* URL (англ. Uniform Resource Locator, URL) – единый указатель ресурса, единообразный локатор (определитель местонахождения) ресурса.
* CRUD (англ. Create Read Update Delete) – аббревиатура обозначающая четыре основных операций над данными: создание, чтение, обновление, удаление.
* SPA (англ. Single Page Application) – одностроничное приложение; веб-приложение или веб-сайт, состоящий из одной веб-страницы и напоминающий настольное приложение.
* JVM – Java Virtual Machine (виртуальная машина Java), основная часть исполняющей системы Java, так называемой Java Runtime Environment.
* Прокси-объект – структурный шаблон проектирования, который предоставляет объект, который контролирует доступ к другому объекту, перехватывая все вызовы (выполняет функцию контейнера).
* Maven – это инструмент для сборки Java проекта.

ВВЕДЕНИЕ

В последнее время начали набирать популярность два подхода к разработке систем – микросервисы и одностраничные приложения (SPA).

Архитектурный стиль микросервисов [1] – это подход, при котором единое приложение строится как набор небольших сервисов, каждый из которых работает в собственном процессе и коммуницирует с остальными используя легковесные механизмы, как правило HTTP.

Одностраничные приложения (SPA) – веб-приложение или веб-сайт, состоящий из одной веб-страницы и напоминающий настольное приложение.

В обоих случаях сервер предоставляет HTTP интерфейс. Для разработки подобного сервера часто выбирают платформу Java с веб-фреймворком Spring MVC, который является частью более общего фреймворка Spring Framework.

Для использования интерфейса необходимо писать много шаблонного кода, что может привести к ошибкам и усложнить внесение модификаций в программный продукт. Для решения этой проблемы и был создан Restler – библиотека для автоматического формирования запросов по переданному представлению классов, отражающих интерфейс сервера.

В экосистеме Spring для работы с базами данных распространена технология Spring Data, которая позволяет получать информацию из базы данных, изменять, удалять и добавлять ее без программирования работы с базой данных, посредством описания структуры базы данных и операций над ней. Для этой технологии существует сопутствующая технология Spring Data Rest, которая транслирует HTTP-запросы в вызовы методов SpringData-хранилища на сервере.

Целью данной работы является разработка модуля для библиотеки Restler, который позволил бы создавать на стороне клиента объекты, представляющие удаленные SpringData-хранилища и работающие через HTTP интерфейс представленный Spring Data Rest.

1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Главная цель библиотеки Restler – избавить разработчика от написания однотипных HTTP-запросов. Restler осуществляет генерацию клиента по переданному представлению класса в JVM. Получив сгенерированный клиент, разработчик может легко вызывать необходимые ему методы и получить результат. На рисунке 1 схематично изображен пример того, где может использоваться Restler.

В приложении А приведено два примера: код клиента без Restler и код клиента с использованием Restler. Эти примеры показывают, как сильно сокращается объем кода с использованием Restler.

Restler в первую очередь нацелен на генерацию клиента для работы с Spring-управляемым сервером. Spring Framework [2] – это универсальный фреймворк с открытым исходным кодом для Java-платформы. Framework обеспечивает решения многих задач, с которыми сталкиваются Java-разработчики и организации, которые хотят создать информационную систему, основанную на платформе Java.

Для простой работы с различными базами данных Spring Framework содержит модуль Spring Data [3]. Для получения и управления объектами Spring Data, существует модуль Spring Data Rest [4], который обеспечивает работу с ресурсами Spring Data с помощью REST-запросов, что позволяет делать это по сети.

REST (Representational state transfer) [5] – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful - системами.

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

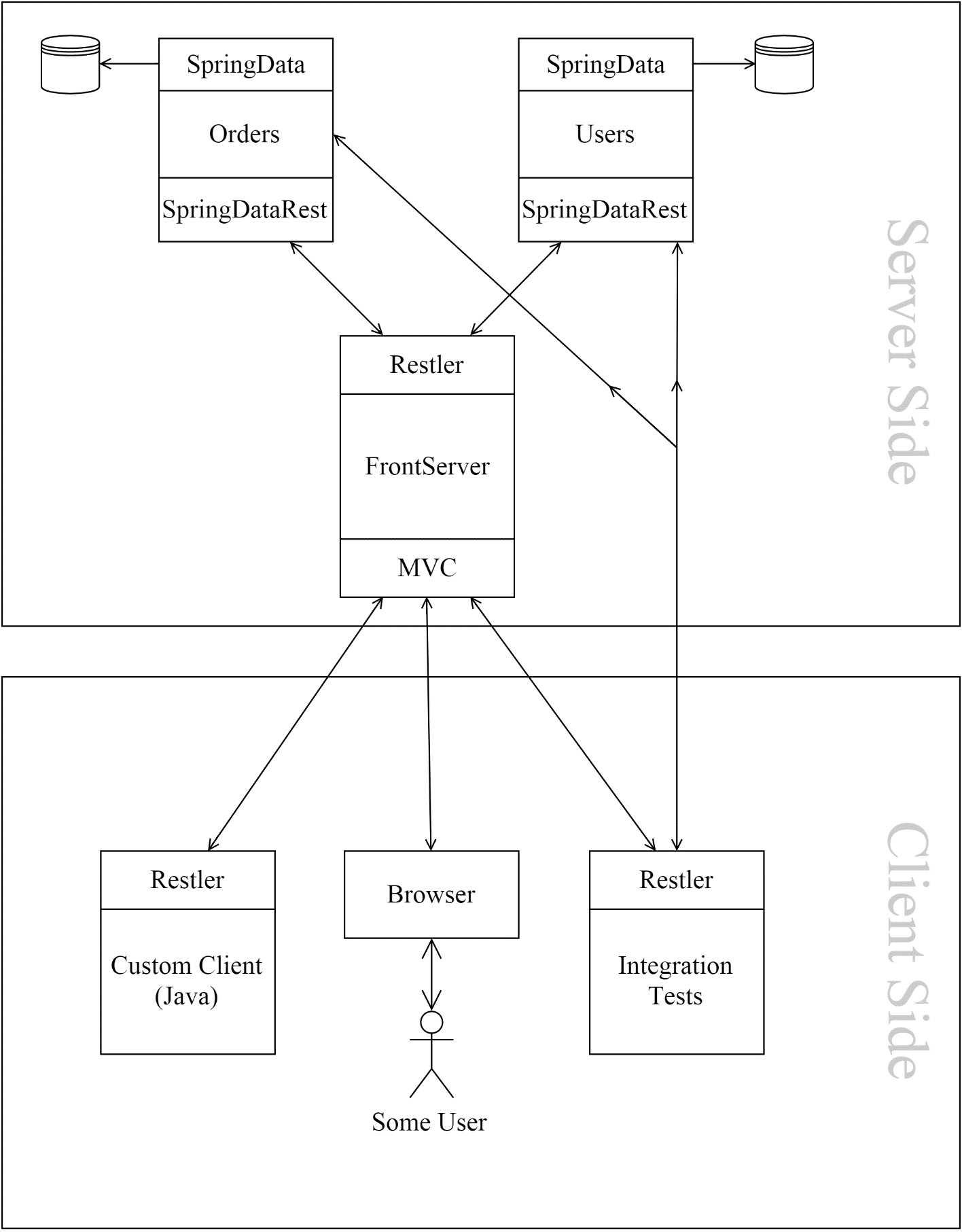


Рисунок 1 – Пример использования Restler

1. ПОСТАНОВКА ЗАДАЧИ
   1. Описание поставленной задачи

Основная задача данной работы – это добавление поддержки технологии Spring Data Rest в библиотеку Restler путем написания модуля расширения библиотеки. Модуль должен добавить поддержку Spring Data Rest в библиотеку Restler. Это позволит разработчику легко получить доступ к базе данных, находящейся на сервере, поддерживающим Spring Data Rest. Главная цель модуля – это дать разработчику возможность работать с расположенными на сервере объектами Spring Data без написания HTTP запросов к серверу. Для этого необходимо предоставить объекты с идентичным интерфейсом объектов Spring Data, находящимся на стороне сервера, но с замененной реализации на отправку запросов и обработку ответов.

В процессе выполнения задачи также необходимо собрать и проанализировать информацию по предметной области, провести поиск и анализ конкурентов, разработать функциональные и нефункциональные требования, спроектировать классы модуля, реализовать спроектированные классы, проанализировать время выполнения методов модуля и провести работы по оптимизации и провести тестирование.

* 1. Описание используемых средств

Для разработки модуля библиотеки Restler использован язык программирования Java и Spring Framework. По той причине, что библиотека Restler написана на языке Java, то для модуля подходит любой язык компилируемый в Java байт-код. Выбор языка Java был обусловлен тем, что альтернативные языки, в отличии от Java, влекут за собой дополнительные зависимости. Для минимизации зависимостей был выбран Java.

Spring Framework – популярный фреймворк для платформы Java. Благодаря разнообразным модулям из которых состоит Spring Framework, разработчики могут его использовать для различных задач. Модуль поддержки Spring Data Rest использует Spring для работы с сетью.

1. АНАЛИЗ ИМЕЮЩИХСЯ АНАЛОГОВ

В процессе поиска аналогов была найдена одна библиотека генерации клиента для SpringDataRest – Spring Data Web Exporter Client [6]. Данная библиотека не развивается с 2012 года, а также она не умеет корректно сохранять SpringData-ресурсы, содержащие другие ресурсы.

Также было найдено несколько аналогов для библиотеки Restler, но ни один из них не обладает прямой поддержкой Spring Data Rest. Ниже представлены найденные аналоги для библиотеки Restler.

Web Application Description Language (WADL) [7] – язык для описания клиента. Существуют утилиты для генерации клиента из файла содержащего описание на языке WADL. Restler в отличии от такого подхода, позволяет генерировать клиент во время исполнения. Нет способа работать с SpringDataRest.

Retrofit [8] библиотека, позволяющая генерировать HTTP клиент по конфигурации, создание которой ложится на плечи разработчика, использующего библиотеку. Так или иначе разработчику все равно приходится формировать запросы, но в этом случае не в виде строки, а в виде специальных объектов. Нет прямого способа работать с Spring Data Rest.

Alchemy Rest Client Generator [9] способен генерировать клиент по Java аннотациям. Нет поддержки Spring Data Rest.

Из анализа аналогов было выявлено, что существующие библиотеки для генерации клиента не обладают возможностью создать клиент, используя уже готовые классы SpringData, написанные при создании сервера, а также предоставляют пользователю классы, интерфейс которых не соответствует интерфейсу Spring Data классов.

1. ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

К данному программному продукту предъявляются следующие функциональные требования.

* 1. Возможность получить ресурс из SpringData-хранилища по уникальному идентификатору.
  2. Возможность добавить новый ресурс в SpringData-хранилище.
  3. Возможность изменить ресурс в SpringData-хранилище.
  4. Возможность удалить ресурс из SpringData-хранилища.
  5. Возможность получить все ресурсы из хранилища.
  6. Возможность получить два и более ресурсов из SpringData-хранилища по списку уникальных идентификаторов.
  7. Возможность добавить два и более ресурсов в SpringData-хранилище.
  8. Возможность изменить два и более ресурсов в SpringData-хранилище.
  9. Возможность удалить два и более ресурсов из SpringData-хранилища.
  10. Возможность получить ресурсы по номеру страницы.
  11. Возможность получить ресурсы, отсортированные по возрастанию по заданным полям ресурса.
  12. Возможность получить ресурсы, отсортированные по убыванию по заданным полям ресурса.
  13. Возможность очистить SpringData-хранилище от всех ресурсов.
  14. Модуль должен обрабатывать пользовательские методы SpringData-хранилища.
  15. Модуль должен представлять полученную информацию от сервера в виде Java объектов.
  16. Модуль должен работать с ресурсами состоящих из простых типов данных.
  17. Модуль должен работать с ресурсами, содержащими сложные типы данных.
  18. Модуль должен уметь обрабатывать ситуацию циклических ссылок между ресурсами.

1. НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ
   1. Основные нефункциональные требования

К данному программному продукту предъявляются следующие нефункциональные требования.

* + 1. Использование библиотеки не должно требовать от пользователя внесения дополнительных изменений на стороне сервера.
    2. Модуль должен правильно реагировать на возможные ошибки.
    3. Модуль должен быть разработан на языке Java.
  1. Требования к параметрам оборудования

Ниже представлены требования к программному обеспечению, необходимому для работы модуля.

* + 1. Установленная Java SE Runtime Environment 8 и выше.

Ниже представлены требования к параметрам оборудования необходимым для модуля:

* + 1. Частота процессора – 1 гигагерц и выше.
    2. Объем оперативной памяти – 512 мегабайта и выше.
    3. Свободное место – 1024 мегабайта и выше на жестком диске.
    4. Для работы с удаленным сервером требуется подключение к сети, в которой находится требуемый сервер.

1. ИСПОЛЬЗУЕМЫЕ ИНСТРУМЕНТЫ И ТЕХНОЛОГИИ

Для решения поставленной задачи было решено использовать среду разработки IntelliJ IDEA 16. Выбранная среда разработки поддерживает язык Java, который был использован для написания модуля, язык Kotlin, который использован для разработки тестового сервера, а также язык Groovy, который использован для написания тестов. В качестве системы контроля версиями был выбран git.

Для сборки проекта из исходного кода и управления зависимостями использовался Apache Maven [10].

Для генерации прокси-объектов использовалась библиотека cglib [11]. Byte Code Generation Library (cglib) высокоуровневое API для генерации и трансформации Java байт кода.

Для профилирования кода, написанного на Java использовался специальный инструмент VisualVM [12].

Для написания тестовых сценариев использовался SpockFramework [13].

1. РЕШЕНИЕ ПОСТАВЛЕННОЙ ЗАДАЧИ
   1. Описание архитектуры Restler

На рисунке 2 изображена схема архитектуры Restler. Компоненты на схеме условно разделены на четыре группы. Компоненты помеченные голубым участвуют в конфигурации библиотеки. Компоненты с зеленым фоном участвуют в создании клиента. Компоненты помеченные желтым обеспечивают работу с HTTP. Компоненты с фиолетовым фоном предоставляют интерфейс пользователю для использования библиотеки.

Restler – класс отвечающий за создание Service по заданным параметрам. CoreModule – интерфейс модуля, необходимого для расширения возможностей библиотеки Restler.

Основным механизмом обработки вызовов библиотеки Restler является цепочка Enhancer'ов. Enhancer – это интерфейс для классов, задача которых обработка вызова, а затем и обработка результата данного вызова. Enhancer был введен для возможности добавления особого поведения в обработку вызовов. Enhancer'ы в цепочке по очереди получает вызов и обрабатывают способом, реализуемым данным Enhancer'ом. Когда вызов доходит до конца цепи, он конвертируется в запрос и отправляется на сервер, затем результат передается в обратном порядке тем же Enhancer'ам. После того, как результат дошел обратно до начала цепи, он возвращается пользователю. Таким образом, цепь Enhancer'ов обрабатывает как вызов, так и результат, возвращаемый пользователю. Enhancer в коде представлен в виде интерфейса CallEnhancer.

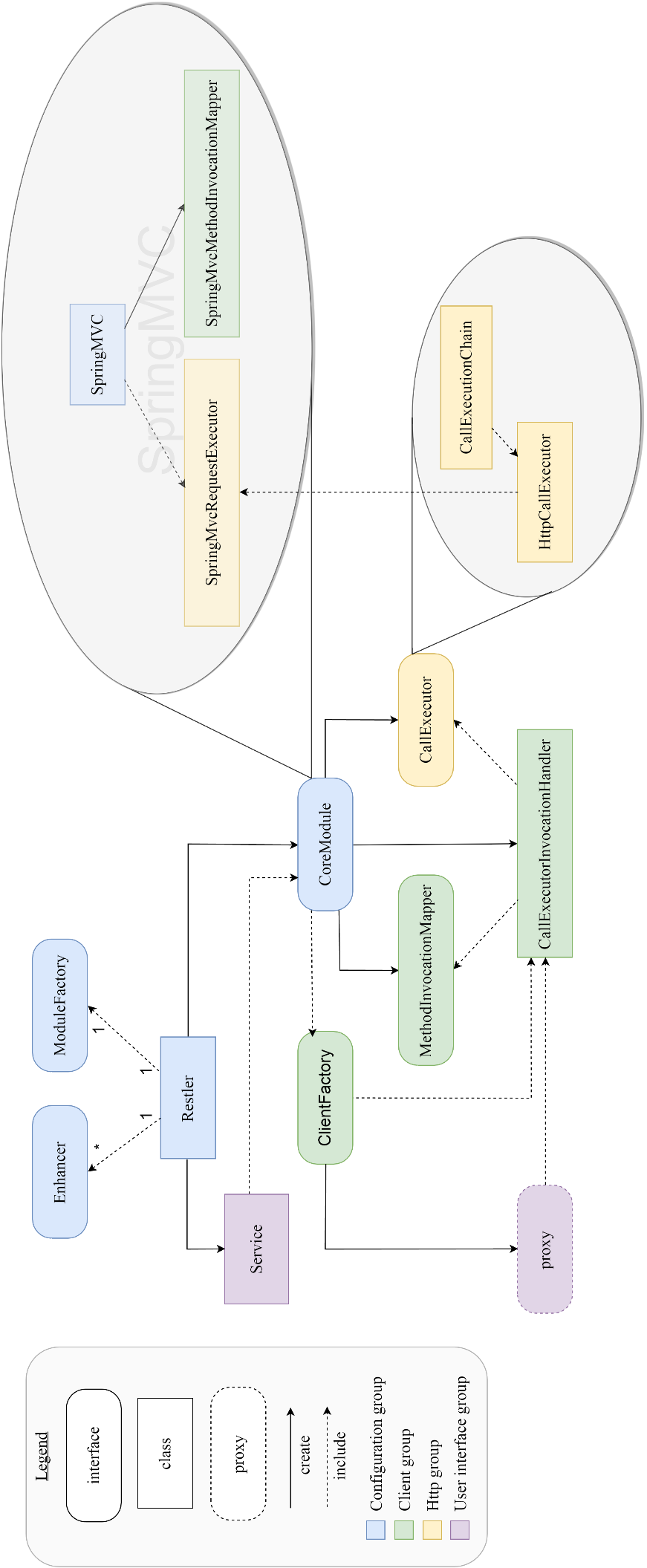


Рисунок 2 – Архитектура Restler

Restler на вход принимает набор пользовательских Enhancer'ов и фабрику (ModuleFactory), с помощью которой создается необходимый модуль с параметрами, переданными в Restler. Модуль создает MethodInvocationMapper и передает его в CallExecutionInvocationHandler, который затем используется для создания клиента. Сконфигурированный CoreModule передается в Service, который возвращается Restler'ом пользователю. С помощью Service пользователь может создать необходимый ему клиент по переданному интерфейсу.

CallExecutorInvocationHandler – класс, содержащий метод invoke, который будет вызываться в сгенерированном клиенте при каждом вызове метода клиента. В этом методе CallExecutorInvocationHandler использует MethodInvocationMapper для получения описания вызова по описанию java-метода клиента, а затем выполняет вызов с помощью объекта CallExecutor.

* 1. Проектирование модуля
     1. *Входные и выходные данные*

Для того чтобы иметь возможность получать SpringData-хранилище пользователю необходимо создать экземпляр Service. При создании пользователь передает параметры, которые задают необходимые свойства экземпляру Service. Подробное описание использования библиотеки описано в приложении Б.

Входные данные, получаемые от пользователя:

* начальный адрес сервера;
* список SpringData-хранилищ с которыми будет работать пользователь;
* максимальный размер кэша.

Дополнительные входные данные, получаемые от пользователя:

* логин и пароль, используемые для аутентификации.

На выходе библиотека отдает пользователю экземпляр класса Service, который затем используется пользователем для получения по заданным параметрам (входным данным) прокси-объектов SpringData-хранилищ.

Service содержит один метод, который позволяет получить нужное хранилище. Этот метод получает на вход представление класса SpringData-хранилища в JVM и возвращает прокси объект SpringData-хранилища.

* + 1. *Общая модель модуля*

Для добавления новых возможностей с помощью модулей в Restler существует ряд интерфейсов. Ниже перечислены интерфейсы, использующиеся в модуле добавляющим поддержку Spring Data Rest в Restler.

* CoreModule – главный интерфейс для модуля, позволяет управлять созданием клиента.
* MethodInvocationMapper – интерфейс классов, строящих вызов (Call) по описанию вызванного метода прокси-объекта. Разные модули библиотеки Restler имеют возможность реализовать данный интерфейс, что позволяет контролировать поведение прокси-объектов, создаваемых библиотекой.
* CallEnhancer – интерфейс, позволяющий встроиться в цепочку обработки вызовов.
* Call – интерфейс, позволяющий добавлять новые типы вызовов.

Модель взаимодействия модуля поддержки Spring Data Rest и ядра библиотеки Restler показана на рисунке 3.

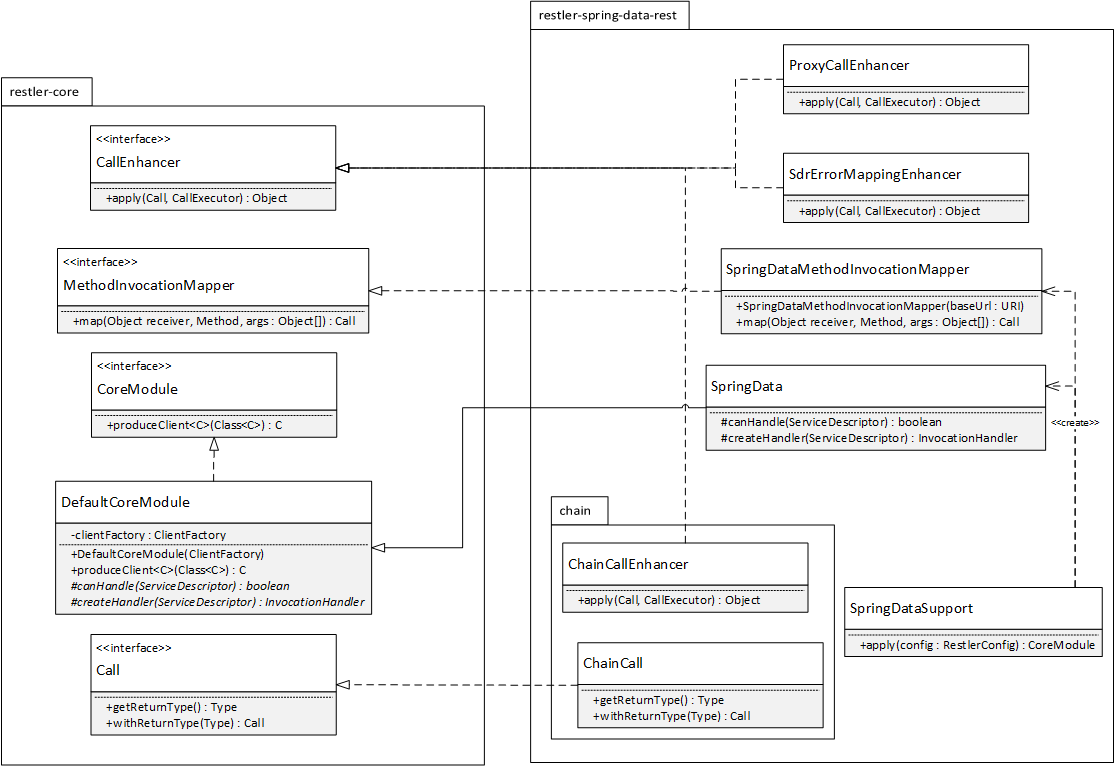


Рисунок 3 – Взаимодействие модуля и ядра библиотеки Restler

* + 1. *Описание классов и интерфейсов*

В пункте 7.2.2 описано то, как модуль взаимодействует с ядром библиотеки Restler. Далее изложено описание классов и интерфейсов по пакетам модуля, включая те классы, которые участвуют только во внутренней логике модуля.

* + - 1. *Главный пакет модуля*

В главный пакет модуля вынесены классы, предназначенные для корректной работы с ядром библиотеки Restler. На рисунке 4 представлена диаграмма классов пакета.

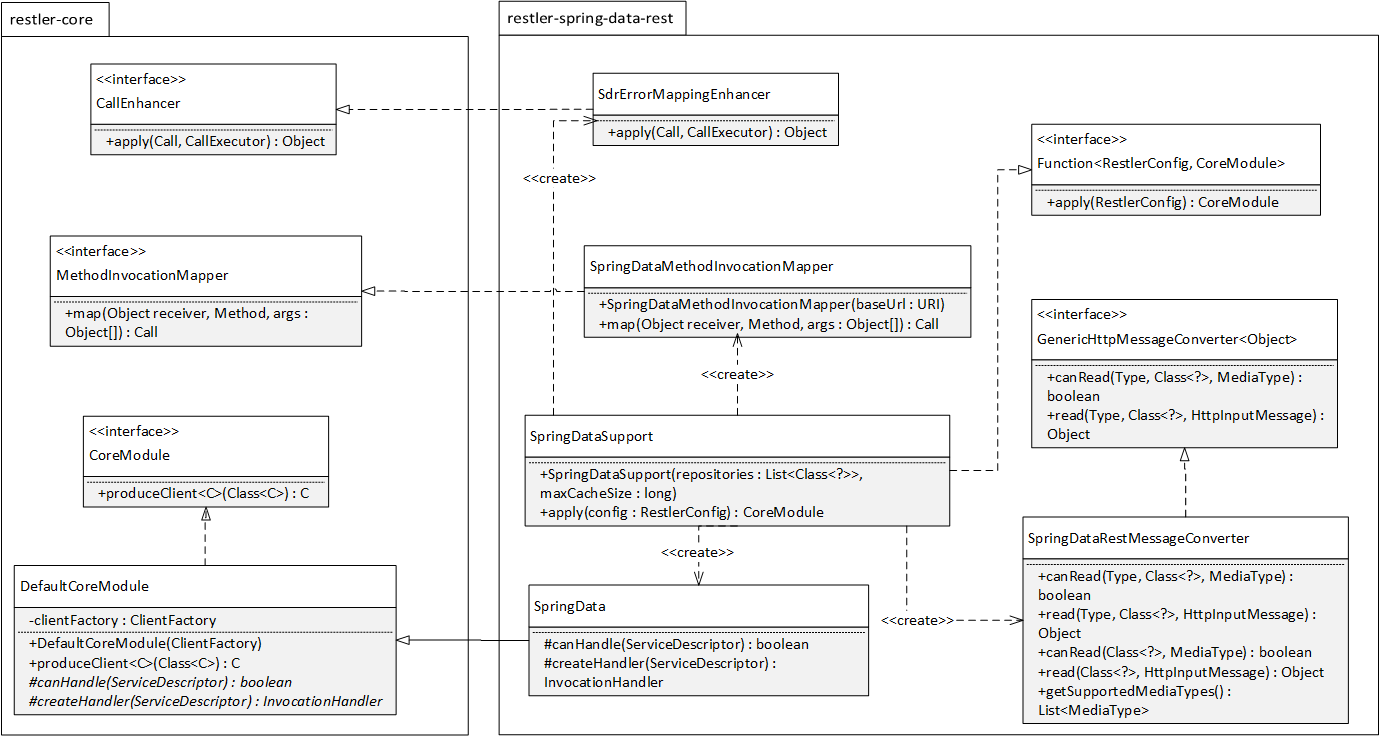


Рисунок 4 – Диаграмма классов главного пакета модуля

1. SpringDataSupport

Класс, реализующий интерфейс Function, получающий на вход объект RestlerConfig и возвращающий реализацию CoreModule. SpringDataSupport передается библиотеке Restler для корректного создания модуля и последующего его использования в ядре. Также данный класс создает компоненты, используемые модулем.

1. SpringDataMessageConverter

Класс, реализующий интерфейс GenericHttpMessageConverter. Производит десериализацию объектов из HTTP-ответов от сервера для возвращения результата пользователю. Был добавлен для обработки ответов от сервера в формате JSON и преобразования их в Java объекты.

1. SpringDataMethodInvocationMapper

Класс, реализующий интерфейс MethodInvocationMapper. Используется для создания вызова по описанию вызванного метода прокси-объекта.

1. SpringData

Главный класс модуля. Управляет созданием прокси-объектов SpringData-хранилища.

1. SdrErrorMappingEnhancer

Класс, реализующий интерфейс CallEnhancer. Позволяет перехватывать и обрабатывать исключение, возникающее при получении кода ошибки от сервера.

* + - 1. *Пакет methods*

Пакет methods состоит из классов, предназначенных для определения поведения прокси-объекта SpringData-хранилища при вызове одного из его методов. Диаграмма классов данного пакета представлена на рисунке 5.

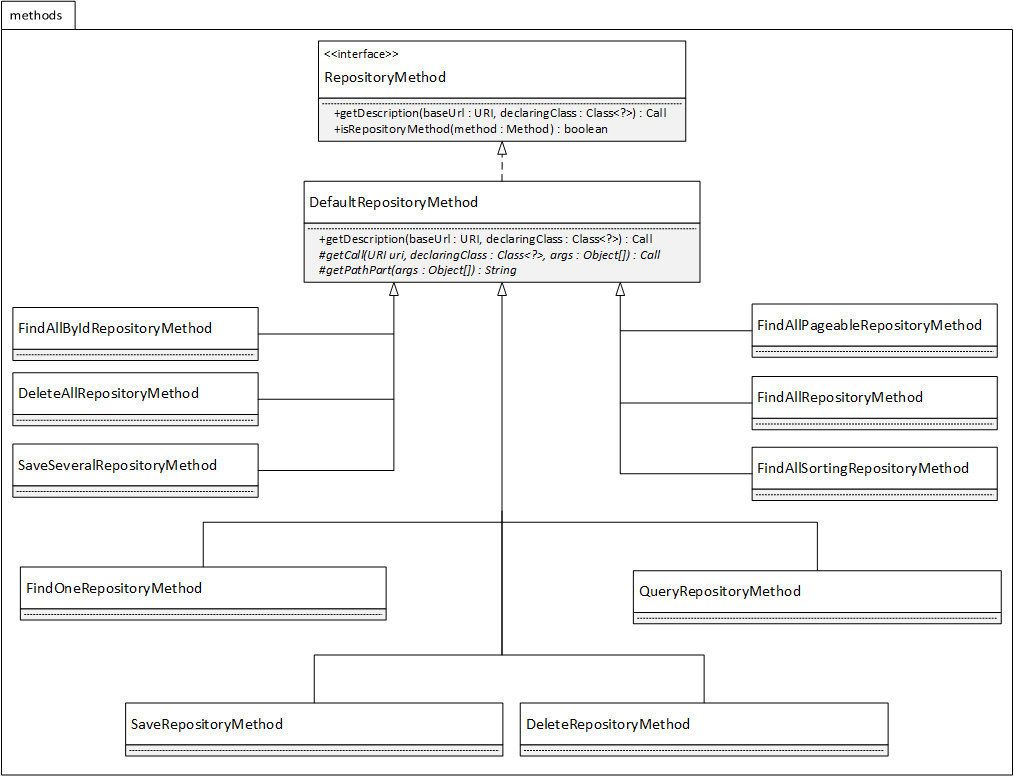


Рисунок 5 – Диаграмма классов пакета methods

1. RepositoryMethod

Интерфейс, представляющий какой-либо метод SpringData-хранилища. Используется в SpringDataMethodInvocationMapper для реализации поведения методов SpringData-хранилища.

Методы интерфейса:

* getDescription – возвращает описание вызова для исполнения метода;
* isRepositoryMethod – определяет является ли реализация RepositoryMethod подходящей для вызываемого метода прокси-объекта.

1. DefaultRepositoryMethod

Абстрактный класс, содержащий общий код для всех реализаций RepositoryMethod. Определяет два абстрактных метода, которые участвуют в реализации getDescription.

* getCall – возвращает описание вызова для конкретного метода.
* getPathPart – возвращает специфичную для метода часть пути.

1. FindOneRepositoryMethod

Класс, наследуемый от DefaultRepositoryMethod. Реализует поведение для поиска и получения ресурса из SpringData-хранилища на сервере. Возвращает HttpCall, представляющий описание запроса по HTTP протоколу на сервер.

1. SaveRepositoryMethod

Класс, наследуемый от DefaultRepositoryMethod. Реализует поведение для сохранения изменений ресурса в SpringData-хранилище на сервере. Возвращает ChainCall, представляющий описание последовательности вызовов.

1. QueryRepositoryMethod

Класс, наследуемый от DefaultRepositoryMethod. Реализует поведение при вызове пользовательских методов, возвращает HttpCall, представляющий описание запроса по HTTP протоколу на сервер.

1. DeleteAllRepositoryMethod

Класс, наследуемый от DefaultRepositoryMethod. Реаализует возможность удалить все ресурсы из хранилища.

1. DeleteRepositoryMethod

Класс, наследуемый от DefaultRepositoryMethod. Реализует возможность удалить ресурс из хранилища по уникальному идентификатору.

1. FindAllByIdRepositoryMethod

Класс, наследуемый от DefaultRepositoryMethod. Реализует возможность получить несколько ресурсов по их идентификаторам.

1. FindAllPageableRepositoryMethod

Класс, наследуемый от DefaultRepositoryMethod. Позволяет получить ресурсы из хранилища по номеру страниц, а также отсортированном в нужном порядке.

1. FindAllRepositoryMethod

Класс, наследуемый от DefaultRepositoryMethod. Реализует возможность получить все ресурсы из репозитория, а также несколько ресурсов по их идентификаторам.

1. FindAllSortingRepositoryMethod

Класс, наследуемый от DefaultRepositoryMethod. Реализует возможность получить все ресурсы отсортированными в некотором порядке.

1. SaveSeveralRepositoryMethod

Класс, наследуемый от DefaultRepositoryMethod. Реализует возможность сохранить несколько ресурсов.

* + - 1. *Пакет calls*

Пакет calls содержит классы для работы с вызовами порождаемыми данным модулем. Пакет содержит классы, реализующие интерфейс CallEnhancer и обрабатывающие нестандартные вызовы, создаваемые модулем. Экземпляры CallEnhancer классов создаются при инициализации модуля и встраиваются в цепочку Enhancer'ов библиотеки Restler. На рисунке 6 изображена диаграмма классов пакета. Ниже дано описание каждого класса пакета.

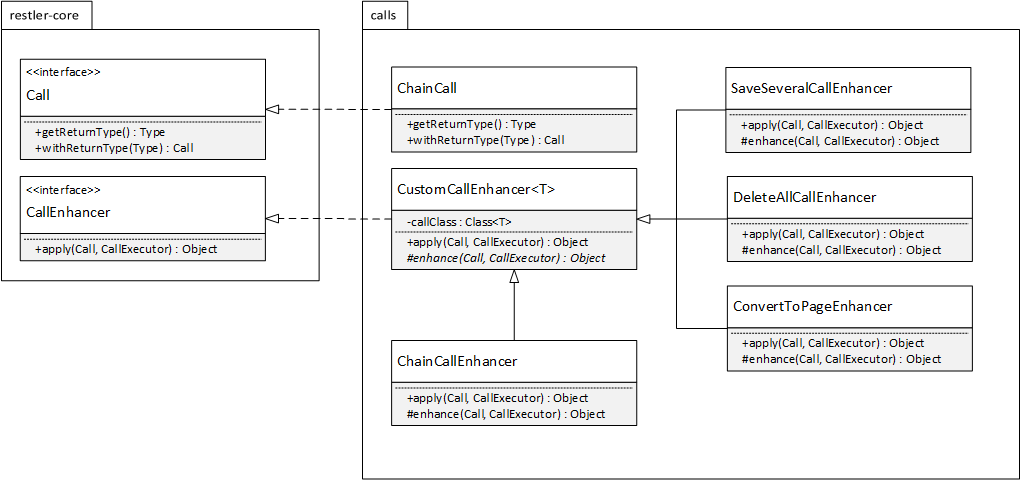


Рисунок 6 – Диаграмма классов пакета calls

1. ChainCall

Класс, реализующий интерфейс Call. Представляет собой описание последовательности запросов. Нужен для построения сложных запросов. Данный класс позволяет также управлять результатом каждого запроса путем передачи специальной функции, которая получает на вход предыдущее значение и текущее и возвращает объект.

1. CustomCallEnhancer

Абстрактный класс, реализующий интерфейс CallEnhancer. Описывает группу CallEnhancer'ов поведение которых заключается в перехвате определенного запроса. В конструктор передается представление перехватываемого класса, реализующего интерфейс Call. Если в CustomCallEnhancer приходит вызов переданного класса, то вызывается абстрактный метод enhance, иначе вызов передается дальше по цепочке.

1. ChainCallEnhancer

Класс, расширяющий класс CustomCallEnhancer. Необходим для перехвата и обработки ChainCall запросов. Приводит полученный объект типа Call к ChainCall, если это возможно, и вызывает необходимые методы экземпляра ChainCall.

1. ConvertToPageCallEnhancer

Класс, расширяющий класс CustomCallEnhancer. Обрабатывает вызовы, ожидающие получить в результате объект типа Page. Работа данного класса заключается в перехвате результата вызова и конвертации его в объект типа Page.

1. DeleteAllCallEnhancer

Класс, расширяющий класс CustomCallEnhancer. Перехватывает запрос на удаление всех ресурсов из хранилища и перенаправляется необходимому хранилищу.

1. SaveSeveralCallEnhancer

Класс, расширяющий класс CustomCallEnhancer. Перехватывает запрос на сохранение нескольких ресурсов в хранилище, разбирает на отдельные ресурсы и сохраняет по одному.

* + - 1. *Пакет proxy*

Пакет содержит классы, необходимые для работы прокси-объектов. Ниже дано описание каждого класса пакета. На рисунке 7 представлена диаграмма классов пакета proxy.

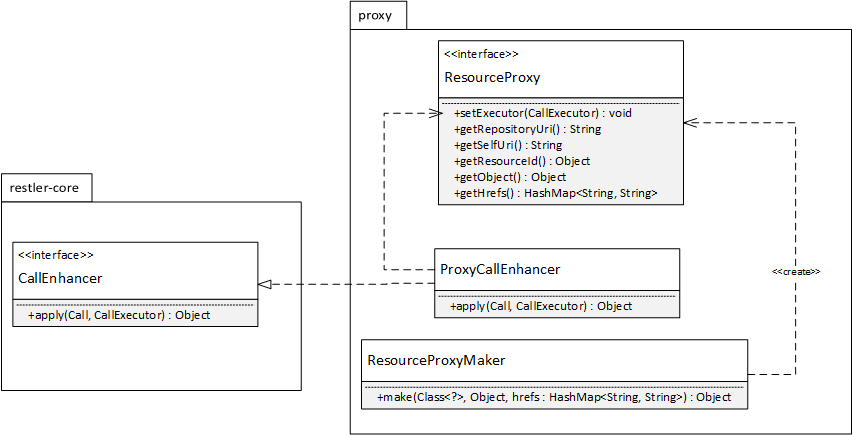


Рисунок 7 – Диаграмма классов пакета proxy

1. ResourceProxy

Интерфейс для прокси-объектов SpringData-ресурсов. Позволяет дополнить прокси-объект необходимыми методами для внутренней работы модуля над ресурсами.

1. ProxyCallEnhancer

Класс, реализующий интерфейс CallEnhancer. Необходим для обработки прокси-объектов ресурсов в цепочке вызовов.

1. ResourceProxyMaker

Класс, используемый для выноса логики создания прокси-объектов ресурса.

* 1. Реализация модуля
     1. *Общее описание реализации модуля*

Для реализации поддержки Spring Data Rest было выделено две основные сущности Spring Data – SpringData-хранилище (Repository) и SpringData-ресурс. SpringData- хранилище представляет интерфейс для работы с SpringData-ресурсами. В SpringData-хранилище содержатся как стандартные методы, так и пользовательские методы. Стандартные методы для CRUD хранилища:

* findOne(id) – находит один элемент по его уникальному идентификатору;
* findAll() – возвращает все ресурсы, находящиеся в хранилище;
* findAll(ids) – возвращает несколько ресурсов по их идентификаторам;
* save(resource) – обновляет ресурс, если ресурса нет, то добавляет его в хранилище;
* save(resources) – обновляет ресурсы в хранилище, отсутствующие ресурсы добавляются в хранилище;
* delete(id) – удаляет ресурс из хранилища по уникальному идентификатору;
* delete(resource) – удаляет ресурс из хранилища;
* delete(ids) – удаляет несколько ресурсов по их идентификаторам;
* delete(resources) – удаляет ресурсы из хранилища;
* deleteAll() - удаляет все ресусры из хранилища.

Стандартные методы для хранилища, работающего со страницами:

* findAll(sort) – возвращает ресурсы в отсортированном порядке и по полю указанных в аргументе sort;
* findAll(pageRequest) – возвращает ресурсы находящиейся на странице pageRequest.

Для удаленной работы с хранилищами используется Spring Data Rest.

Для построения запросов в модуле при вызове методов SpringData-хранилища используются специальные экземпляры классов, реализующих RepositoryMethod интерфейс. Каждый класс обрабатывает вызов определенного метода в SpringData-хранилище:

* FindOneRepositoryMethod – findOne();
* FindAllRepositoryMethod – findAll();
* FindAllByIdRepositoryMethod – findAll(ids);
* SaveRepositoryMethod – save(resource);
* SaveSeveralRepositoryMethod – save(resources);
* DeleteRepositoryMethod – delete(id), delete(ids), delete(resource) и delete(resources);
* DeleteAllRepositoryMethod – deleteAll();
* QueryRepositoryMethod – обрабатывает пользовательские методы.

Для хранилища, работающего со страницами предусмотрены следующие классы:

* FindAllSortingRepositoryMethod – findAll(sort);
* FindAllPageableRepositoryMethod – findAll(pageRequest).

Это позволяет вынести реализацию логики в отдельные классы.

Для того чтобы пользователь мог работать с удаленными ресурсами так же, как и с локальными, строится специальный объект ResourceProxy, заменяющий запросами к серверу часть методов настоящего ресурса, возвращающих другие ресурсы. ResourceProxy содержит дополнительную информацию, такую как:

* настоящий объект ресурса, который он заменяет;
* экземпляр класса, реализующий интерфейс CallExecutor, для выполнения запросов;
* ссылки расположения ассоциированных ресурсов на сервере;
* ссылку на местонахождение самого ресурса.

На рисунке 8 представлена цепочка выполнения вызова. Вызов по порядку проходит экземпляры классов, реализующих интерфейс CallEnhancer, до CallExecutor.

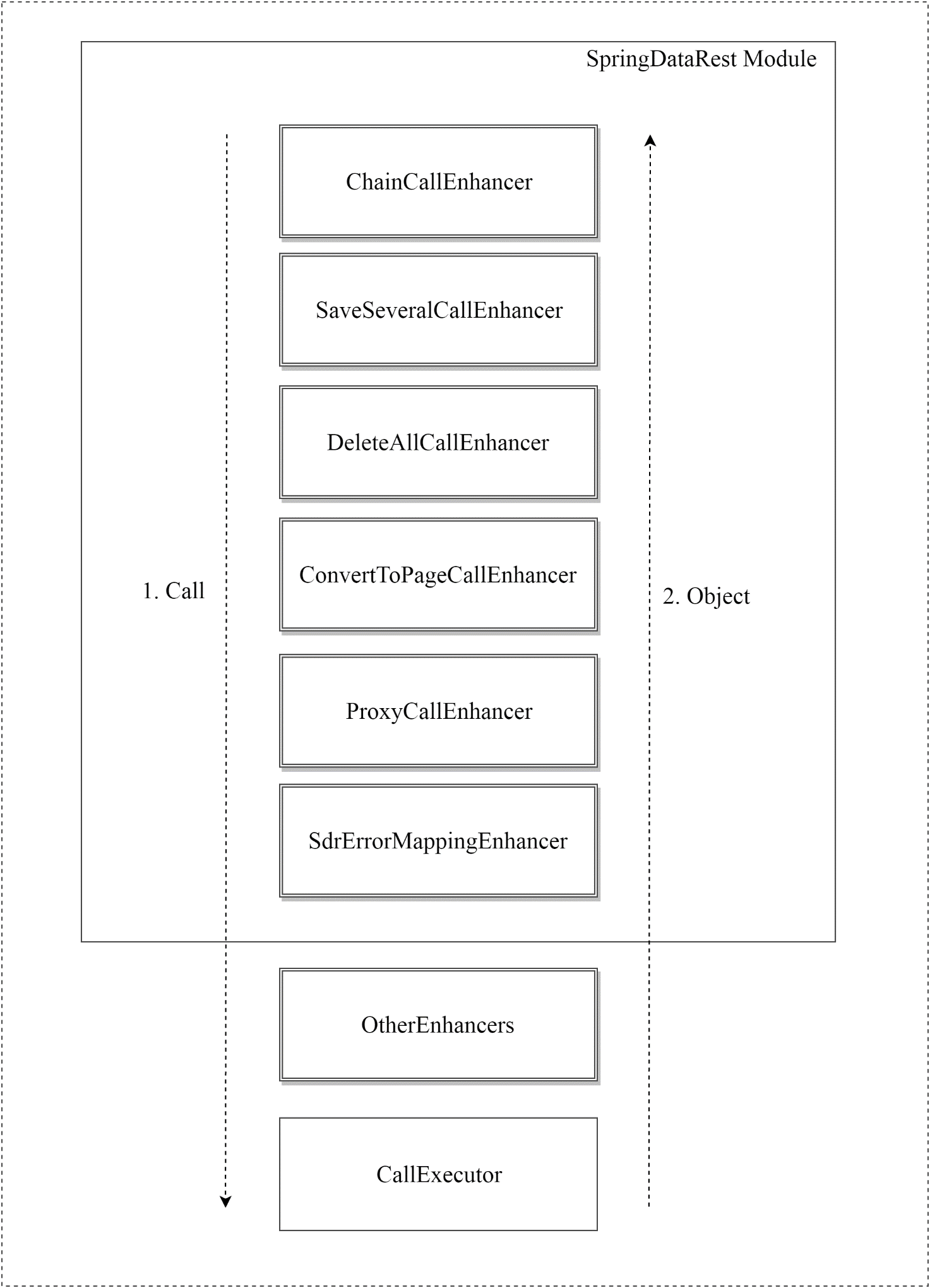


Рисунок 8 – Цепочка выполнения вызова

Когда CallExecutor выполнит вызов, он возвращает результат обратно. Таким образом, каждый CallEnhancer получает доступ к самому вызову, а затем и к результату. Этот механизм обеспечивает реализацию функционального требования 4.15. CallEnhancer позволяет внедриться в цепочку выполнения, тем самым давая возможность корректно расширять обработку вызовов в библиотеке.

Для модуля поддержки Spring Data Rest были добавлены следующие классы, реализующие интерфейс CallEnhancer.

* ChainCallEnhancer – обрабатывает специальный вызов ChainCall (цепочка вызовов), который содержит несколько вызовов со строгим порядком.
* ProxyCallEnhancer – перехватывает объекты типа ResourceProxy и инициализирует дополнительные данные. Также занимается кэшированием прокси объектов.
* SdrErrorMappingEnhancer – обрабатывает исключение HttpException в зависимости от кода ошибки из-за которого было вызвано исключение.
* ConvertToPageCallEnhancer – отслеживает ожидающий страницу вызов и конвертирует полученный результат в страницу.
* DeleteAllCallEnhancer – ожидает запрос на удаление всех ресурсов из хранилища и обрабатывает его.
* SaveSeveralCallEnhancer – ожидает запрос на сохранение нескольких ресурсов в хранилище и обрабатывает его.
  + 1. *Создание и изменение ресурса со вложенными ресурсами*

Ресурс со вложенными ресурсами можно представить в виде ориентированного графа. Для корректного сохранения изменений всего ресурса, необходимо определить порядок добавления новых ресурсов. По той причине, что нельзя добавлять связь на несуществующий ресурс на сервере, неправильный порядок добавления и изменения ресурсов может потерять связи между этими ресурсами. Также иногда один ресурс может ссылаться на второй, в то время как второй ресурс ссылается на первый, как на прямую, так и косвенно. В этом случае возникает цикл, который необходимо разорвать.

Решение вышеизложенных проблем заключается в отделении полезных данных ресурса от его связей с другими ресурсами. Таким образом, получается два списка: один с ресурсами без связей на другие ресурсы, другой с описанием связей между этими ресурсами.

В первую очередь обрабатываются ресурсы, помеченные на создание. Алгоритм добавляет к ресурсам связи, которые не могут потеряться и помечает их «решенными». Затем ищутся ресурсы у которых все связи «решены» и добавляется запрос на добавление данного ресурса в очередь. Если же таких ресурсов не было найдено, то выбирается связь, которая помечается на создание отдельным запросом, после того как все необходимые ресурсы будут созданы. Выбранная связь помечается «решенной». Этот процесс повторяется до тех пор, пока все ресурсы не будут созданы.

После того как все ресурсы, помеченные на создание были созданы алгоритм обрабатывает ресурсы, помеченные на изменение. Так как все необходимые ресурсы уже существуют, то алгоритму остается лишь добавить оставшиеся связи ресурсам и создать для них запросы на изменение.

Блок-схема описанного выше алгоритма представлена на рисунке 9.

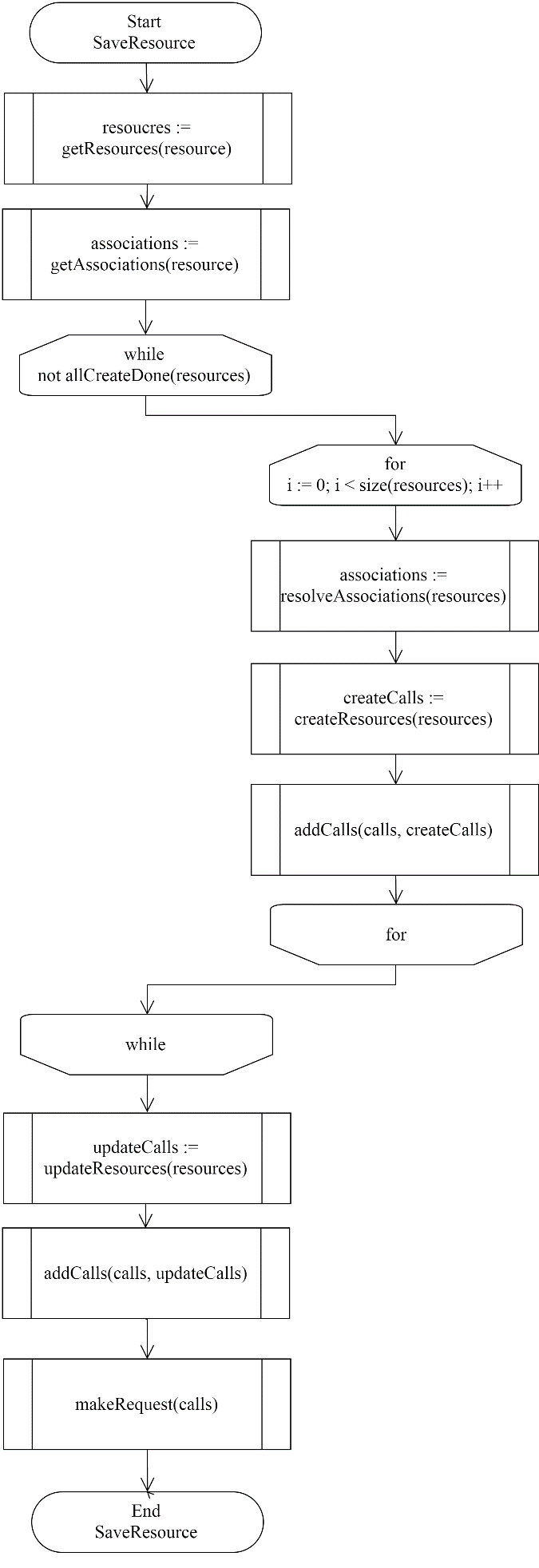


Рисунок 9 – Блок-схема алгоритма сохранения ресурса со вложенными ресурсами

* 1. Анализ и оптимизация программного кода модуля
     1. *Создание тестового сценария*

Для анализа программного кода модуля поддержки Spring Data Rest был написан тестовый сценарий. Тестовый сценарий состоит из тестового сервера и тестового клиента, для которого будут проводится измерения производительности. Тестовый сервер и тестовый клиент находятся на разных машинах, находящиеся в одной сети.

Цель тестового сценария создать достаточную нагрузку для измерения времени передачи данных по сети и времени работы методов самого модуля.

На сервере для работы с базой данных были созданы следующие SpringData-ресурсы:

* Person – представляет собой человека, у которого может быть несколько питомцев и несколько адресов;
* Pet – представляет собой питомца, у которого есть имя и владелец;
* Address – представляет собой адрес, который прикреплен к определенному человеку.

Для каждого ресурса было создано соответствующее SpringData-хранилище:

* PersonRepository;
* PetRepository;
* AddressRepository.

Тесты на клиенте разделены на восемь стадий:

1. создание тысячи персон и сохранение на сервере;
2. создание тысячи питомцев и сохранение на сервере;
3. создание по десять адресов для каждой персоны и сохранение на сервере;
4. получение всех персон с сервера;
5. получение всех питомцев с сервера;
6. получение всех адресов с сервера;
7. назначение персонам питомцев и адресов и сохранение изменений на сервере;
8. изменение адресов и сохранение изменений на сервере.
   * 1. *Измерение времени выполнения и анализ результатов*

Для измерения и анализа производительности использовался специальный инструмент VisualVM. Для профилирования производительности методов тестового сценария использовался инструментальный метод. На рисунке 10 изображена диаграмма, построенная по результатам выполнения тестового сценария. Диаграмма показывает, что метод RestTemplate.exchange занимает больше времени, чем остальные. Это метод принадлежит Spring фреймворку и используется для выполнения запросов.

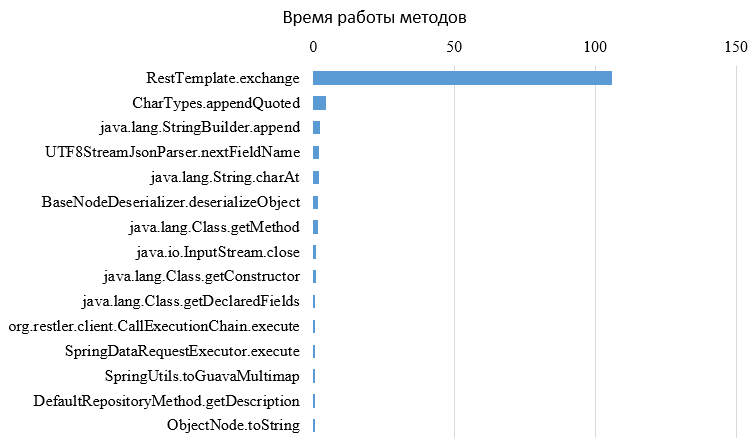


Рисунок 10 – Диаграмма времени выполнения методов

В таблице 1 приведено точное время выполнения методов, представленных на рисунке 10.

Таблица 1 – Время выполнения методов

|  |  |
| --- | --- |
| Метод | Время (в секундах) |
| RestTemplate.exchange | 106 |
| CharTypes.appendQuoted | 4,614 |
| java.lang.StringBuilder.append | 2,175 |
| UTF8StreamJsonParser.nextFieldName | 2,115 |
| java.lang.String.charAt | 2,037 |
| BaseNodeDeserializer.deserializeObject | 1,713 |
| java.lang.Class.getMethod | 1,482 |
| java.io.InputStream.close | 0,87 |
| java.lang.Class.getConstructor | 0,786 |
| java.lang.Class.getDeclaredFields | 0,703 |
| org.restler.client.CallExecutionChain.execute | 0,576 |
| SpringDataRequestExecutor.execute | 0,488 |
| SpringUtils.toGuavaMultimap | 0,482 |
| DefaultRepositoryMethod.getDescription | 0,469 |
| ObjectNode.toString | 0,449 |

После анализа результата профилирования, было вычислено что методы модуля org.restler.spring.data занимают около 6,6 процентов от всего времени выполнения тестового сценария. Больше всего времени уходит на работу метода RestTemplate.exchange (около пятидесяти семи процентов от всего времени выполнения тестового сценария). Этот метод посылает запрос по сети и ожидает ответ. Следовательно, RestTemplate.exchange осуществляет работу с сетью, что и влечет за собой большие расходы времени. RestTemplate.exchange работает с сетью синхронно, поэтому изменение работы с сетью на асинхронную модель может дать выигрыш по времени выполнения большого количества независимых запросов. Однако, это потребует значительного изменения архитектуры библиотеки Restler, что выходит за рамки данной работы.

В таблице 2 приведено общее время для каждой стадии тестового сценария.

Таблица 2 – Время выполнения стадий тестового сценария

|  |  |  |
| --- | --- | --- |
| № | Название стадии | Время (в секундах) |
| 1 | создание персон | 7,254 |
| 2 | создание питомцев | 4,396 |
| 3 | создание адресов | 40,537 |
| 4 | получение всех персон | 2,614 |
| 5 | получение всех питомцев | 2,470 |
| 6 | получение всех адресов | 22,718 |
| 7 | добавление питомцев и адресов персонам | 75,688 |
| 8 | изменение адресов | 45,775 |

Из результатов, приведенных в таблице 2, видно, что сохранение изменений композитных ресурсов (стадия семь), имеет самое большое время выполнения. Это происходит по той причине, что помимо самого ресурса, происходит сохранение изменений во всех ресурсах, на которые он ссылается.

Уменьшение времени работы сводится к уменьшению запросов, посылаемых серверу. По той причине, что при написании модуля изначально учитывался данный факт, то для выполнения задач уже используется минимально возможное количество запросов, следовательно, оптимизация не даст заметного прироста в скорости.

По причине того, что минимизация работы с сетью со стороны модуля поддержки Spring Data Rest была проведена во время проектирования и разработки, а остальные части модуля занимают несущественное время, выполнение работ по оптимизации модуля было сочтено нецелесообразным.

* 1. Тестирование модуля
     1. *Тестовый сервер*

Тестовый сервер представляет собой веб-сервер предоставляющий доступ к SpringData через REST. На рисунке 11 представлена модель данных для тестового сервера.

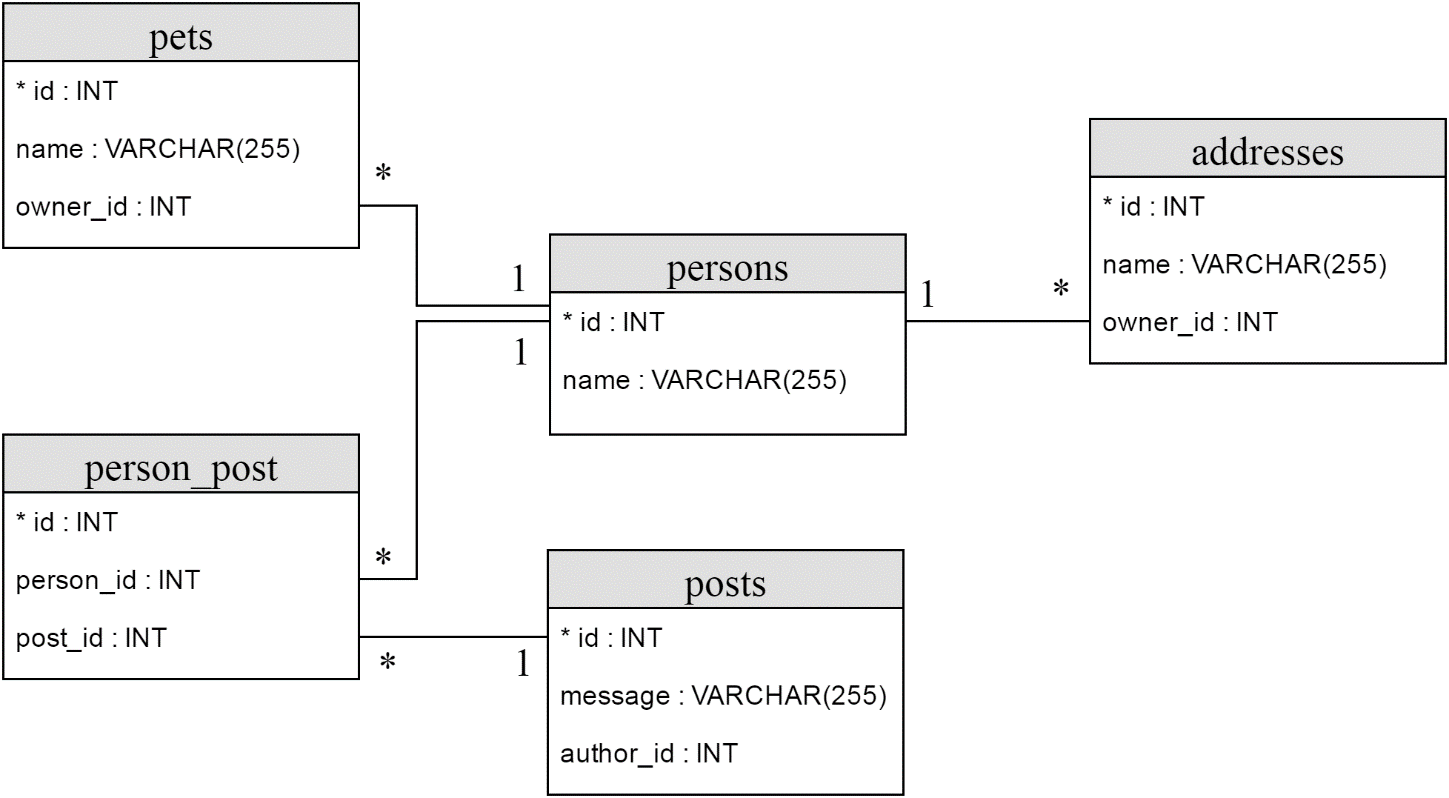


Рисунок 11 – Модель данных для тестового сервера

На сервере присутствуют два SpringData-хранилища:

* PersonsRepository – хранилище ресурсов типа Person;
* PetsRepository – хранилище ресурсов типа Pet;
* PostsRepository – хранилище ресурсов типа Post.

Сервер работает со следующими SpringData-ресурсами:

* Person представляют собой запись таблицы persons;
* Pet представляет собой запись таблицы pets;
* Address, представляющий запись таблицы addresses;
* Post представляет запись в таблице posts.
  + 1. *Тестовый клиент*

Тестовый клиент содержит в себе тесты, которые обращаются к тестовому серверу и проверяют полученный результат. Тестовые сценарии написаны на языке Groovy с использованием фреймворка SpockFramework.

Для модуля поддержки Spring Data Rest были написаны следующие тестовые сценарии:

* получение Person из PersonsRepository по идентификатору (функциональное требование 4.1);
* получение Person из PersonsRepository c помощью пользовательского метода findById (функциональное требование 4.14);
* получение Person из PersonsRepository с помощью пользовательского метода findByName (функциональное требование 4.14);
* получение всех SpringData-ресурсов из SpringData-хранилища (функциональное требование 4.5);
* получение нескольких SpringData-ресурсов из SpringData-хранилища по их уникальным идентификаторам (функциональное требование 4.6);
* получение вложенного ресурса из другого SpringData-ресурса (функциональное требование 4.17);
* получение у SprigData-ресурса данных простого типа (функциональное требование 4.16);
* получение нескольких ресурсов, расположенных на одной странице (функциональное требование 4.10);
* получение сортированного списка ресурсов в порядке возрастания по полю id (функциональное требование 4.11);
* получение сортированного списка ресурсов в порядке убывания по полю id (функциональное требование 4.12);
* корректная обработка циклической зависимости между SpringData-ресурсами (функциональное требование 4.18);
* сохранение изменения ресурса, включающего в себя другие ресурсы (функциональные требования 4.17 и 4.3);
* создание и добавление в SpringData-хранилище ресурса в SpringData-хранилище (функциональное требование 4.2);
* создание и добавление нескольких ресурсов в SpringData-хранилище (функциональное требование 4.7);
* удаление ресурса из SpringData-хранилища (функциональное требование 4.4);
* удаление всех SpringData-ресурсов из SpringData-хранилища (функциональное требование 4.13);
* сохранение изменений нескольких ресурсов в SpringData-хранилище (функциональное требование 4.8);
* удаление нескольких ресурсов из SpringData-хранилища (функциональное требование 4.9).

Данные тестовые сценарии полностью покрывают все функциональные требования.

* + 1. *Структура тестового проекта*

Для отделения серверной части от клиентской было создано три модуля:

* модуль для тестового сервера;
* модуль для тестового клиента;
* модуль для интерфейсов, используемых тестовым сервером и тестовым клиентом.

Такое разделение позволяет явно отделить клиент от сервера, что делает тестирование достоверным.

* + 1. *Maven плагин для автоматизации запуска сервера*

Для тестирования необходимо, чтобы тестовый сервер был запущен. Для того, чтобы сервер запускался автоматически до запуска тестов в maven и останавливался после окончания тестирования, был написан maven плагин.

На вход плагину подается путь к jar файлу сервера и, если необходимо, задержка до перехода к следующей стадии. Плагин имеет две команды start и stop. Команда start запускает jar файл в отдельном процессе, команда stop останавливает процесс.

Таким образом, до тестов вызывается команда start для собранного jar файла сервера, а по окончанию тестирования команда stop.

ЗАКЛЮЧЕНИЕ

Во время выполнения данной работы было проделано следующее:

* изучена предметная область;
* проведен анализ аналогов;
* собраны требования для модуля;
* спроектированы классы модуля;
* реализованы спроектированные классы;
* проведен анализ на необходимость оптимизации модуля;
* проведено тестирование модуля.

Анализ аналогов позволил определить ключевые функции модуля, на которых был сделан основной упор при разработке. Собранные требования были использованы в проектировании модуля и создании тестовых сценариев. При проектировании классов модуля была учтена архитектура библиотеки Restler для использования возможностей библиотеки.

В результате измерений и анализа времени выполнения методов модуля было выявлено, что большая часть времени уходит на передачу данных по сети, в то время как время выполнения методов модуля на порядок меньше. Было принято решение, не выполнять оптимизацию модуля по той причине, что выполнение оптимизации модуля не даст выигрыша по скорости.

В результате разработки модуля поддержки технологии Spring Data Rest для библиотеки Restler были получены следующие результаты.

* Для реализации модуля было спроектировано и написано 34 класса.
* Для обеспечения взаимодействия частей модуля между собой было создано 2 программных интерфейса.
* Для проверки отдельных частей модуля на соответствие требованиям было написано 23 тестовых сценариев.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. James Lewis, Martin Fowler. Microservices [Электронный ресурс] – URL: http://martinfowler.com/articles/microservices.html/. – Яз. англ. – (Дата обращ. 03.02.2016).
2. Spring Framework [Электронный ресурс] – URL: https://projects.spring.io/spring-framework/. – Яз. англ. – (Дата обращ. 26.01.2016).
3. Spring Data [Электронный ресурс] – URL: http://projects.spring.io/spring-data/. – Яз. англ. – (Дата обращ. 26.01.2016).
4. Spring Data Rest [Электронный ресурс] – URL: http://projects.spring.io/spring-data-rest/. – Яз. англ. – (Дата обращ. 26.01.2016).
5. REST [Электронный ресурс] – URL: https://www.w3.org/TR/2002/WD-webarch-20020830/#glossary-rest/. – Яз. англ. – (Дата обращ. 23.03.2016).
6. Spring Data Web Exporter Client [Электронный ресурс] – URL: https://github.com/onlyabout/spring-data-rest-client/. – Яз. англ. – (Дата обращ. 27.01.2016).
7. Спецификация WADL [Электронный ресурс] – URL: https://www.w3.org/Submission/wadl/. – Яз. англ. – (Дата обращ. 27.01.2016).
8. Retrofit [Электронный ресурс] – URL: http://square.github.io/retrofit/. – Яз. англ. – (Дата обращ. 27.01.2016).
9. Alchemy Rest Client Generator [Электронный ресурс] – URL: https://github.com/strandls/alchemy-rest-client-generator/. – Яз. англ. – (Дата обращ. 27.01.2016).
10. Apache Maven [Электронный ресурс] – https://maven.apache.org/what-is-maven.html/. – Яз. англ. – (Дата обращ. 03.02.2016).
11. CGLib [Электронный ресурс] – URL: https://github.com/cglib/. – Яз. англ. – (Дата обращ. 03.02.2016).
12. VisualVM [Электронный ресурс] – URL: http://visualvm.java.net/intro.html/. – Яз. англ. – (Дата обращ. 21.03.2016).
13. SpockFramework [Электронный ресурс] – URL: https://code.google.com/archive/p/spock/. – Яз. англ. – (Дата обращ. 11.04.2016).

ПРИЛОЖЕНИЕ А

Сравнение объема кода

Для сравнения объема кода ниже приведены два листинга кода – листинг 1 и листинг 2.

|  |
| --- |
| public class ManualTodosImpl implements Todos {  private static final String baseUrl = "http://localhost:8080/";  private RestTemplate restTemplate = new RestTemplate();  public ManualTodosImpl() {  ParanamerModule module = new ParanamerModule();  restTemplate.getMessageConverters().stream().  filter(c -> c instanceof MappingJackson2HttpMessageConverter).  forEach(c -> ((MappingJackson2HttpMessageConverter)c).getObjectMapper().registerModule(module));  }  @Override  public Todo create(Todo todo) {  return restTemplate.postForObject(baseUrl, todo, Todo.class);  }  @Override  public Todo update(String id, @RequestBody Todo todo) {  try {  return restTemplate.exchange(new RequestEntity<>(todo, null, HttpMethod.PUT, new URI(baseUrl + "/" + id)), Todo.class).getBody();  } catch (URISyntaxException e) {  throw new RuntimeException("Unexpected exception", e);  }  }  @Override  public Todo delete(@PathVariable String id) {  try {  return restTemplate.exchange(  new RequestEntity<>(null, null, HttpMethod.DELETE, new URI(baseUrl + "/" + id)), Todo.class).getBody();  } catch (URISyntaxException e) {  throw new RuntimeException("Unexpected exception", e);  }  }  @Override  public Todo get(@PathVariable String id) {  return restTemplate.getForObject(baseUrl + "/" + id, Todo.class);  }  @Override  public Todo[] list() {  return restTemplate.getForObject(baseUrl, Todo[].class);  }  } |

Листинг 1 – Создание клиентского кода без Restler

|  |
| --- |
| SpringMvcSupport springMvcSupport = new SpringMvcSupport();  springMvcSupport.addJacksonModule(new ParanamerModule());  Restler builder = new Restler("http://localhost:8080", springMvcSupport);  Service todosService = builder.build();  todos = todosService.produceClient(Todos.class); |

Листинг 2 – Создание клиентского кода при помощи Restler

В листинге 1 описана реализация класса для работы с сервером через контроллер Todos без использования библиотеки Restler. В листинге 2 представлено создание эквивалентного клиента с использованием библиотеки Restler.

ПРИЛОЖЕНИЕ Б

Руководство пользователя

1. Установка библиотеки и модуля поддержки Spring Data Rest

Для использования библиотеки Restler с модулем поддержки Spring Data Rest необходимо добавить следующую зависимость в Maven проект.

|  |
| --- |
| <dependency>  <groupId>org.restler</groupId>  <artifactId>restler-spring-data-rest</artifactId>  <version>0.5.0</version> </dependency> |

1. Инициализация библиотеки и модуля поддержки Spring Data Rest

Необходимо чтобы интерфейсы и классы с которыми будет работать модуль были доступны в проекте. А именно, интерфейсы SpringData-хранилищ и классы используемых в них SpringData-ресурсов, а также дополнительные классы данных, которые используются в самих ресурсах.

|  |
| --- |
| /\*Путь к серверу. \*/  String hostUri = "http://localhost:8080";  /\*Переменная, в которой будет сохранен список SpringData-хранилищ, используемых библиотекой. \*/  List<Class<?>> repositories = new ArrayList<>();  /\*Заполнение переменной repositories.\*/  Collections.addAll(repositories, Repository1.class, Repository2.class);  /\*Максимальный размер кэша прокси-объектов SpringData-ресурсов, используемого модулем. \*/  long maxCacheSize = 1000;  /\*Создание экземпляра класса, используемого Restler'ом для создания модуля поддержки Spring Data Rest. \*/  SpringDataSupport springDataSupport = new SpringDataSupport(repositories, maxCacheSize);  /\*Создание сервиса, используемого в дальнейшем для работы с библиотекой. \*/  Service service = new Restler(hostUri, springDataSupport).build(); |

Для инициализации библиотеки и модуля поддержки Spring Data Rest основными являются следующие строчки:

|  |
| --- |
| /\*Создание экземпляра класса, используемого Restler'ом для создания модуля поддержки Spring Data Rest. \*/  SpringDataSupport springDataSupport = new SpringDataSupport(repositories, maxCacheSize);  /\*Создание сервиса, используемого в дальнейшем для работы с библиотекой. \*/  Service service = new Restler(hostUri, springDataSupport).build(); |

Объект springDataSupport используется для инициализации самого модуля поддержки Spring Data Rest, он передается библиотеке Restler, которая в свою очередь строит объект типа Service. В данном случае объект service будет использоваться для генерации SpringData-хранилищ.

1. Использование библиотеки

После того как был получен объект service, можно начинать создавать SpringData-хранилища, с которыми мы будем работать. Это делается с помощью метода Service – produceClient(Class<?> repositoryClass). Метод возвращает прокси-объект класса repositoryClass. Прокси-объект имеет такой же интерфейс, что и repositoryClass, но при этом заменяет реализацию этого интерфейса на логику модуля. Таким образом, вызывая какой-либо метод прокси-объекта, этот вызов передается модулю поддержки Spring Data Rest, который в свою очередь обрабатывает его и делает запрос на сервер, получает ответ от сервера и конвертирует его в ожидаемый результат. Таким образом, работа с прокси-объектом класса repositoryClass не отличается от работы с объектом самого класса repositoryClass.

|  |
| --- |
| Repository1 repository1 = service.produceClient(Repository1.class);  Repository2 repository2 = service.produceClient(Repository2.class); |

* 1. *Работа с CrudRepository*

Например, следующий код запрашивает у SpringData-хранилища ресурс с уникальным идентификатором 10. Если ресурса с таким номером нет в хранилище, то будет возвращена нулевая ссылка (null), иначе будет возвращен прокси-объект класса Resource1.

|  |
| --- |
| Resource1 resource10 = Repository1.findOne(10); |

Прокси-объекты для ресурса заменяют логику геттеров, позволяя не загружать ресурс полностью с сервера, а получать необходимые данные по требованию. Например, если Resource1 ссылается на Resource2, то следующий код получит прокси-объект класса Resource2 с которым ассоциирован resource10.

|  |
| --- |
| Resource2 resource2 = resource10.getResource2(); |

Если необходимо изменить ресурс, то, например, если у Resource2 есть свойство Name, тогда следующий код изменит имя resource2 на сервере.

|  |
| --- |
| resource2.setName(“New Name”);  repository2.save(resource2); |

Если необходимо удалить resource2, то это можно сделать с помощью следующего кода.

|  |
| --- |
| repository2.delete(resource2); |

Для удаления ресурса по идентификатору можно воспользоваться методом delete(id).

|  |
| --- |
| repository2.delete(101); |

Если необходимо сохранить новый ресурс в хранилище, то можно воспользоваться следующим кодом.

|  |
| --- |
| /\*Создание нового ресурса с id: 100, name: “New Resource”. \*/  Resource2 newResource = new Resource2(100, “New Resource”);  /\*Добавление нового ресурса в хранилище. \*/  repository2.save(newResource); |

Если необходимо получить все ресурсы в хранилище.

|  |
| --- |
| List<Resource2> resources2 = (List<Resource2>)repository2.findAll(); |

Если необходимо получить несколько ресурсов по их идентификаторам.

|  |
| --- |
| List<long> identifications = new ArrayList<>();  Collections.addAll(identifications, 1, 3, 10, 23, 4);  /\*Вернет ресурсы под номерами 1, 3, 10, 23, 4. Ссылки тех ресурсов, что отсутствуют в хранилище, будут иметь значение null. \*/  List<Resource2> resources2 = (List<Resource2>)repository2.findAll(identifications); |

Если необходимо добавить несколько новых ресурсов.

|  |
| --- |
| List<Resource2> newResources2 = new ArrayList<>();  newResources2.add(new Resource2(50, “Resource2 50”));  newResources2.add(new Resource2(51, “Resource2 51”));  newResources2.add(new Resource2(42, “Resource2 42”));  repository2.save(newResources2); |

Если необходимо изменить несколько ресурсов.

|  |
| --- |
| List<Resource2> resources2 = repository2.findAll();  for(Long i = 0; i < resources2.size(); ++i) {  resources2.get(i).setName(“New Name ” + i.toString());  }  repository2.save(resources2); |

Также можно изменить и добавить новые ресурсы одновременно.

|  |
| --- |
| List<Resource2> resources2 = repository2.findAll();  for(Long i = 0; i < resources2.size(); ++i) {  resources2.get(i).setName(“New Name ” + i.toString());  }  resources2.add(new Resource2(50, “Resource2 50”));  resources2.add(new Resource2(51, “Resource2 51”));  resources2.add(new Resource2(42, “Resource2 42”));  repository2.save(resources2); |

Если необходимо удалить несколько ресурсов.

|  |
| --- |
| List<long> identifications = new ArrayList<>();  Collections.addAll(identifications, 1, 3, 10, 23, 4);  /\*Вернет ресурсы под номерами 1, 3, 10, 23, 4. Ссылки тех ресурсов, что отсутствуют в хранилище, будут иметь значение null. \*/  List<Resource2> resources2 = (List<Resource2>)repository2.findAll(identifications);  /\*Удаляет ресурсы под номером 1, 3, 10, 23, 4.\*/  repository2.delete(resources2); |

Или тоже самое, только по идентификаторам ресурсов.

|  |
| --- |
| List<long> identifications = new ArrayList<>();  Collections.addAll(identifications, 1, 3, 10, 23, 4);  /\*Удаляет ресурсы под номером 1, 3, 10, 23, 4.\*/  repository2.delete(identifications); |

Также можно удалить все ресурсы из хранилища.

|  |
| --- |
| repository2.deleteAll(); |

* 1. *Работа с PagingAndSortingRepository*

Если хранилище наследуется от интерфейса PagingAndSortingRepository, то ему доступны два новых метода. Первый метод findAll(Sort) позволяет получить ресурсы в отсортированном порядке.

|  |
| --- |
| /\*Описание правил сортировки. В данном случае ресурсы будут отсортированы в порядке убывания по полю id. \*/  Sort sort = new Sort(Sort.Direction.DESC, “id”);  /\*Получить ресурсы по заданным правилам сортировки. \*/  List<Resource1> resources1 = (List<Resource1>)repository1.findAll(sort); |

Второй метод findAll(Pageable) позволяет получить ресурсы, находящиеся на определенной странице.

|  |
| --- |
| /\*Запрос на получения ресурсов, находящихся на странице номер 3, размер страницы равен 10. \*/  Pageable pageRequest = new PageRequest(3, 10);  /\*Получить страницу номер 3 с размером 10. \*/  Page page = repository1.findAll(pageRequest);  /\*Получить ресурсы, находящиеся на странице. \*/  List<Resource1> resources1 = page.getContent(); |

С помощью метода findAll(Pageable) возможно также предварительно отсортировать ресурсы в нужном порядке.

|  |
| --- |
| /\*Запрос на получения отсортированных ресурсов в порядке убывания по полю id, находящихся на странице номер 3, размер страницы равен 10. \*/  Pageable pageRequest = new PageRequest(3, 10, **Sort.Direction.DESC, “id”**);  /\*Получить страницу номер 3 с размером 10. \*/  Page page = repository1.findAll(pageRequest);  /\*Получить отсортированные ресурсы, находящиеся на странице. \*/  List<Resource1> resources1 = page.getContent(); |

* 1. *Пользовательские методы в SpringData-хранилище*

Вызов пользовательских методов на клиенте ничем не отличается от вызовов этих же методов на сервере. Следующий код демонстрирует вызов двух пользовательских методов findByName и findById.

|  |
| --- |
| Resource1 resource101 = repository1.findById(101);  List<Resource1> resourcesByName = repository2.findByName(“Resource 101”); |

Метод findById(id) возвращает ресурс по уникальному идентификатору. Метод findByName(name) возвращает ресурсы с именем name.