

МИНОБРНАУКИ РОССИИ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет информатики
Кафедра программной инженерии

УДК 004.457

ДОПУСТИТЬ К ЗАЩИТЕ В ГАК
Зав.кафедрой, профессор, д.ф-м.н.
_____ О.А.Змеев
«_____» _____ 2014 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
РАЗРАБОТКА ПРОТОТИПА СИСТЕМЫ АНАЛИЗА И МОНИТОРИНГА ТРАФИКА В
ВЫСОКОНАГРУЖЕННЫХ СЕТЯХ НА ОСНОВЕ ДАННЫХ, ПОЛУЧЕННЫХ ПО
ПРОТОКОЛУ SFLOW

по основной образовательной программе подготовки бакалавров
010400 – Информационные технологии

Периков Игорь Олегович

Руководитель ВКР
директор ООО «Нетпоинт»
_____ И.А.Кудрявцев
«_____» _____ 2014 г.

Автор работы
студент группы № 1401
_____ И.О.Периков

Электронная версия работы
помещена в электронную библиотеку

Администратор электронной
библиотеки факультета
_____ Е.Н.Якунина

Реферат

Выпускная квалификационная работа 40 с., 16 рис., 15 источников, 1 приложение.

SFLOW, МОНИТОРИНГ СЕТЕЙ, ПЛАТФОРМА JAVA, MODEL-VIEW-PRESENTER, VAADIN

Объект исследования – технология sFlow.

Цель работы – проектирование и реализация прототипа системы анализа и мониторинга трафика в высоконагруженных сетях на основе данных, полученных по протоколу sFlow.

Методы исследования – объектно-ориентированный анализ и проектирование, экспериментальный на базе ЭВМ.

Результаты – разработан прототип приложения, имеющий основной функционал и готовый к его наращиванию.

Оглавление

Введение.....	4
1. sFlow	5
2. Используемые технологии.....	8
2.1. Vaadin	9
3. Разработка.....	11
3.1. Проектирование.....	11
3.1.1. MVP	11
3.1.2. Схема хранения данных.....	12
3.2. Реализация	13
3.2.1. Создание и сборка проекта.....	13
3.2.2. Пакетная обработка.....	16
3.2.3. MVP	17
3.2.4. Конфигурирование и выполнение запроса	20
3.2.5. Обновление пользовательского интерфейса	24
3.2.7. Определение приложения по номеру порта и всех адресов подсети	26
4. Руководство пользователя.....	28
Заключение	36
Список использованных источников	37
Приложение А. Руководство по установке.....	39

Введение

В больших сетях часто возникают проблемы, связанные с ее производительностью, для устранения проблемы ее необходимо локализовать, что затруднительно ввиду размера сети. Так, например, недобросовестный сотрудник может пользоваться торрент-клиентами на рабочем месте. Для таких случаев было бы очень удобно иметь инструмент, позволяющий администратору или другому ответственному лицу определять использование работниками ненадлежащих программ, а также обнаруживать узкие места в работе сети.

Другой проблемой является организация и подсчет провайдером исходящего и входящего трафика клиента, а также определение объема использования пользователем дополнительных услуг, например IP-телефонией.

В условиях работы в высоконагруженных сетях необходимо, чтобы это решение было легкомасштабируемым, позволяло эффективно работать с сетевыми устройствами, обладающими скоростями от гигабита в секунду и выше.

Для решения данной проблемы существует несколько технологий, позволяющих получать информацию о передаваемых по каналам связи пакетах без сколько-нибудь значимого влияния на производительность сети. Данная работа посвящена разработке инструмента мониторинга и анализа, основанного на использовании технологии sFlow.

1. sFlow

sFlow – это технология, предназначенная для мониторинга высоконагруженных сетей и анализа потока пакетов, проходящих через устройства данной сети. Достигается это совместной работой агентов, коллекторов и анализирующих устройств [2].

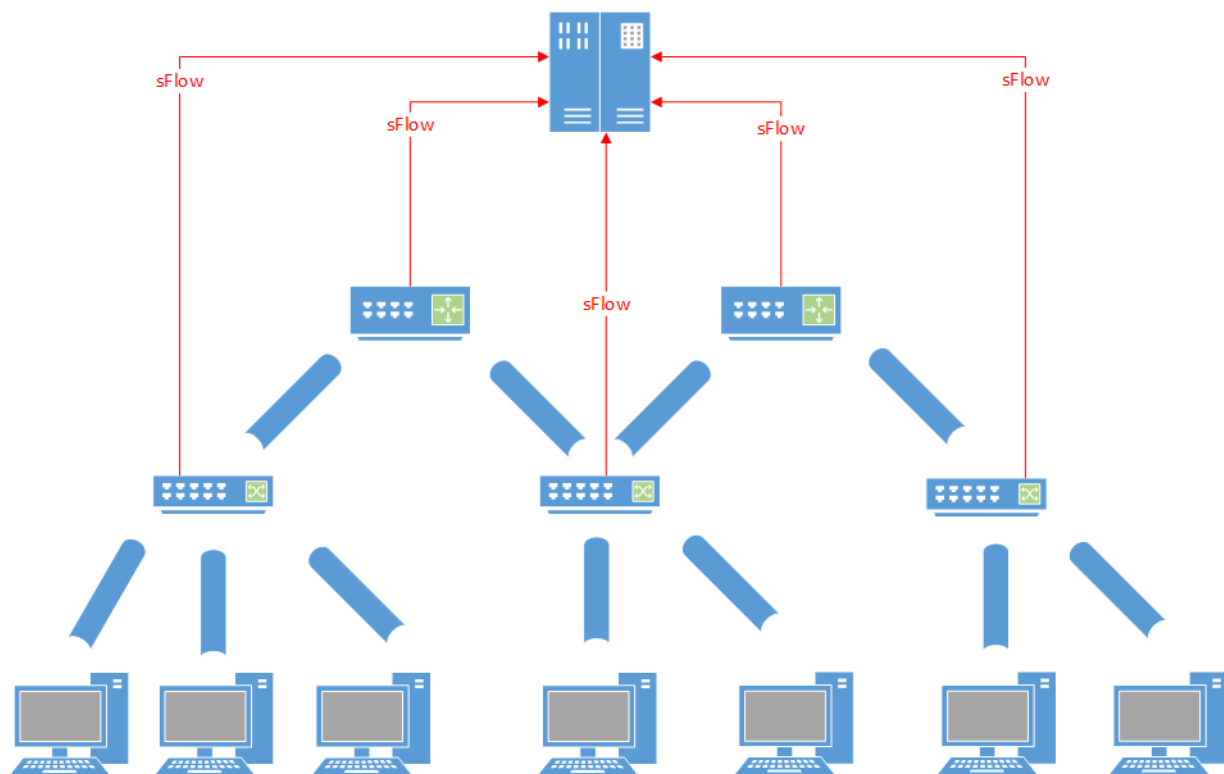


Рис. 1. Схематичная схема работы sFlow.

Агент sFlow – это программное обеспечение, которое устанавливается на коммутатор или маршрутизатор. В дальнейшем, агент, используя технологию выборки, собирает информацию о проходящем через данное устройство трафике и отправляет ее в коллектор данных. Этот поток данных на схеме изображен тонкими стрелками, ведущими к коллектору.

Коллектор предназначен для хранения данных sFlow, концептуально различают два подхода к устройству коллектора:

1. Коллектор представляет собой хранилище данных, а анализатор sFlow представлен в системе отдельным узлом.
2. Коллектор совмещен с анализатором sFlow.

Обычно коллектор представляет собой обычную реляционную базу данных. На данной схеме изображена версия коллектора, который совмещен с анализатором.

Анализатор предназначен для агрегации данных sFlow и составления отчетов на основании этих данных, а также визуальном представлении результатов. На основании данных отчетов можно определить характер проблемы, а также локализовать ее.

Вот несколько примеров типовых отчетов:

1. Адреса-лидеры по утилизации для данного сервера, то есть список тех адресов, с которыми происходит наибольший обмен информацией, сколько было получено/отправлено бит информации, какой процент составляет от общего потока.
2. Порты-лидеры по утилизации – список портов, по которым происходит наибольший обмен информацией. По номеру порта можно определить конкретное приложение, если этот порт был официально зарегистрирован в администрации адресного пространства интернет (IANA).

Данная работа посвящена разработке анализатора, а также созданию минимальной инфраструктуры для его тестирования.

Для обеспечения хорошей масштабируемости и добавления минимальной нагрузки на сеть в процессе работы, sFlow использует принцип статистической выборки, ее суть заключается в том, что агент собирает информацию не о каждом пакете, прошедшем через устройство, а об одном из N пакетов, где N – коэффициент выборки, обычно устанавливаемый администратором сети. Использование выборки незначительно портит статистическую картину ввиду больших скоростей (от 1 Гбит/с и выше).

Формат данных sFlow. Используя технологию sFlow, можно получить следующую информацию о каждом из пакетов:

1. Адрес агента, осуществившего сборку данных.
2. IP-адрес источника и получателя.
3. Тип транспортного протокола (TCP, UDP).
4. Порт источника и получателя.
5. Размер пакета.
6. Дата и время прохождения пакета через устройство.

7. MAC-адрес источника и получателя.
8. Тип Ethernet (v4, v6).
9. Номера виртуальных локальных сетей.
10. Частота выборки.
11. Предельный период времени, после которого пакет перестанет существовать.
12. Тип обслуживания (Ip TOS, type of service) – байт, содержащий набор критериев, определяющих тип обслуживания пакета, позволяет расставлять приоритеты по доставке трафика. На практике практически не используется.
13. Прочие tcp флаги.

Остается заметить, что использование технологии sFlow является промышленным стандартом, а почти все коммутаторы и маршрутизаторы на данный момент имеют поддержку данной технологии.

В качестве альтернативы sFlow существует технология NetFlow. В отличие от sFlow использование данной технологии ведет к большей нагрузке на сеть, потому что в ней не используется алгоритм выборки, по этой причине NetFlow недостаточно хорошо работает для высоконагруженных сетей.

2. Используемые технологии

Так как итоговая система должна представлять из себя веб-приложение, то необходимо определиться со стеком технологий, которые будут использованы для разработки.

При выборе веб-фреймворка в качестве основных критериев выбора были выделены:

1. Простота создания пользовательского интерфейса
2. Скорость разработки
3. Наличие удобных инструментов для отображения диаграмм и графиков
4. Степень документированности

Учитывая вышеперечисленные ориентиры, область выбора была сужена до следующих java веб-фреймворков:

- 1) GWT
- 2) Vaadin

Оба из них позволяют создавать пользовательский интерфейс в Swing-подобном стиле (Swing — библиотека, предназначенная для создания настольных приложений на платформе java), то есть императивно и на языке java. Каждый из них имеет удобные инструменты для отрисовки диаграмм, удовлетворяющие заданным требованиям. При детальном рассмотрении более удобным оказался Vaadin, который в итоге и был выбран.

В качестве инструмента для работы с базой данных был выбран Hibernate. Hibernate представляет из себя java-библиотеку, которая, помимо решения задач объектно-реляционного отображения баз данных, предоставляет удобные средства конфигурирования запросов и избавляет от написания большого количества однотипного шаблонного кода.

В качестве инструмента сборки используется Apache Maven. Он предоставляет удобную систему работы с зависимостями посредством декларативных описаний, а также автоматизирует сам процесс сборки и создания конечного файла.

2.1. Vaadin

Vaadin это веб-фреймворк, предназначенный для простого и удобного создания Rich Internet Application. Компонентно-ориентированность позволяет легко и быстро создавать большие сложные интерфейсы. Создание элементов интерфейса происходит полностью на языке java, например следующим образом:

```
Button button = new Button();  
  
button.setCaption("This is the button!");
```

Основное удобство заключается в том, что не надо задумываться о том, как это превратится в HTML, CSS и JavaScript страницы, этим занимается Google Web Toolkit, лежащий в основе создания пользовательского интерфейса данного веб-фреймворка. Для отображения приложения браузер не нуждается в установке каких-либо дополнительных плагинов, что обеспечивает высокий уровень кроссбраузерности приложения. Стилистическое изменение внешнего вида обеспечивается темами, они могут быть созданы вручную посредством Sass (расширение CSS3), или посредством графического конструктора на сайте.

Архитектура фреймворка представлена на следующем рисунке, темным цветом выделены классы, предоставляемые фреймворком, остальные – те, которые разрабатывает программист и характерны конкретному приложению [3].

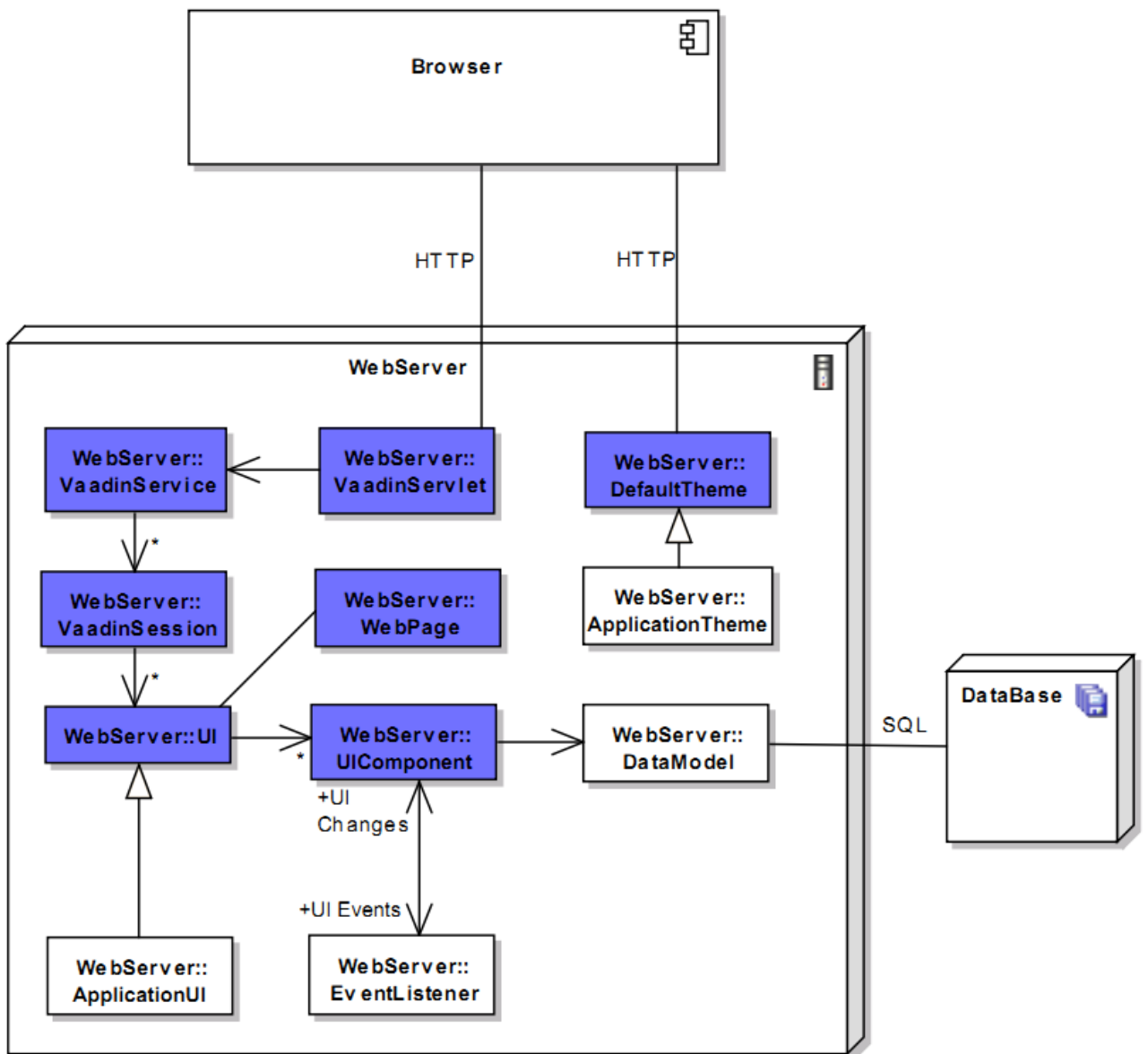


Рис. 2. Архитектура фреймворка Vaadin.

3. Разработка

3.1. Проектирование

3.1.1. MVP

Архитектуру приложения было решено построить в соответствии с паттерном Model-View-Presenter. Архитектурный шаблон Model-View-Presenter представляет из себя видоизмененный MVC:

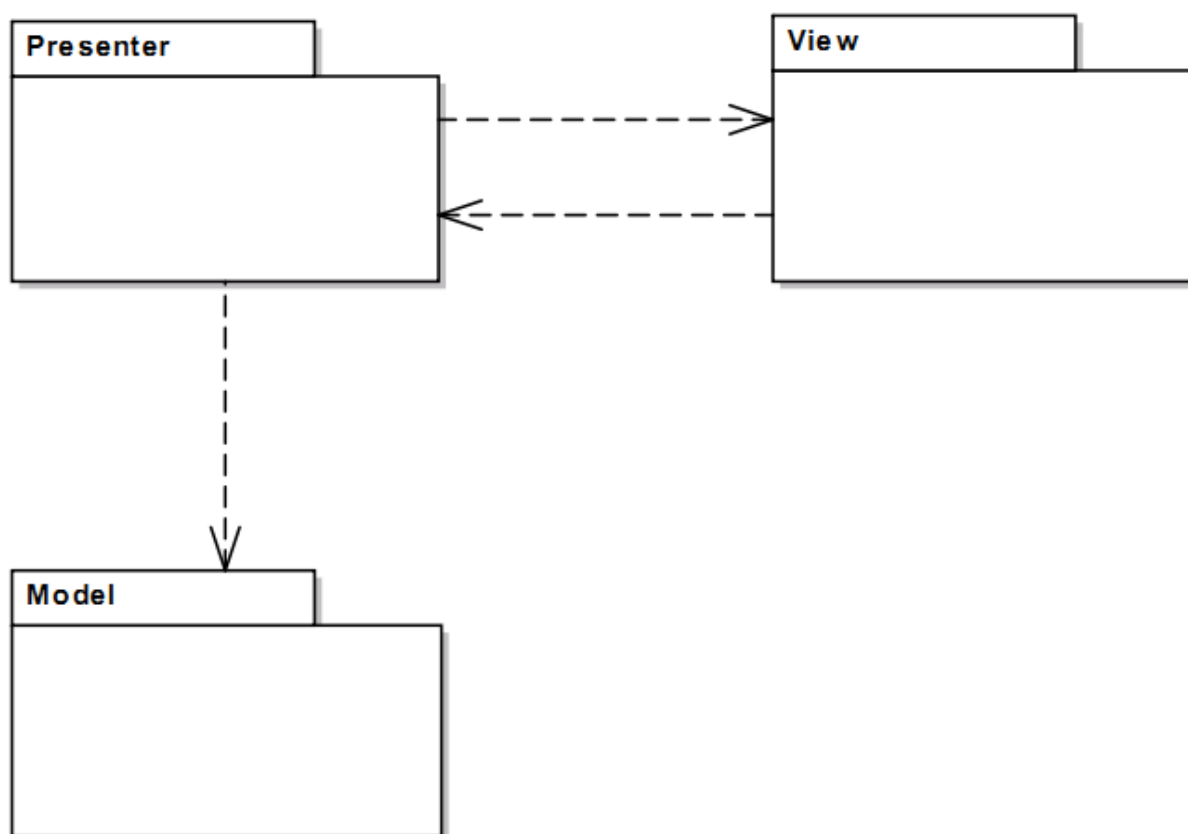


Рис. 3. Model-View-Presenter.

При использовании данного паттерна приложение делится на 3 концептуальные части:

1. Model (модель) - часть системы, представляющая из себя всё, что относится к бизнес логике приложения – классы сущностей, обработки и доступа к данным.
2. View (представление) - часть системы, отвечающая за отображение пользовательского интерфейса и данных, иными словами визуальное отображение модели.

3. Presenter - промежуточное звено между моделью и представлением, в его обязанности входит обработка пользовательских действий, работа с моделью и последующее форматирование и отображение данных в представлении. С точки зрения реализации Presenter в своем распоряжении имеет ссылки на модель и на представление, а также соответствующие методы доступа.

Паттерн имеет несколько вариаций в зависимости от того, какой объем работ предоставлен промежуточному звену:

1. Passive View – вся логика отрисовки и обновления представления сосредоточена в presenter'е. В случае использования Vaadin это означает, что представление лишь описывает существующие элементы интерфейса и предоставляет методы доступа к ним, дабы presenter мог их изменять.
2. Supervising Controller – представление содержит минимум логики, остальное сосредоточено на presenter'е.
3. Майк Потел (Mike Potel) предлагает организацию представления таким образом, что вся логика его обновления хранится в представлении, а presenter занимается обработкой пользовательских действий и отправлением данных из модели в представление [10, 11].

Для данной работы было решено использовать presenter последнего вида, так как представление имеет сложную логику отрисовки.

Веб-фреймворк Vaadin не имеет стандартной реализации данного архитектурного подхода. Поэтому он был реализован вручную.

3.1.2. Схема хранения данных

Данные, полученные по протоколу sFlow хранятся в одной таблице реляционной базы данных. В качестве реляционной СУБД используется MySQL. Для хранения в базе данных были выбраны самые важные и используемые атрибуты.

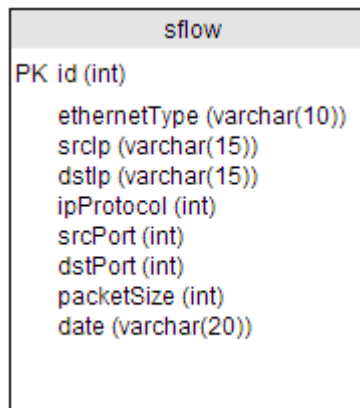


Рис. 4. Схема хранения данных.

3.2. Реализация

3.2.1. Создание и сборка проекта

Чтобы создать Vaadin веб-приложение с использованием Apache Maven, используется команда “mvn archetype:generate -DarchetypeGroupId=com.vaadin -DarchetypeArtifactId=vaadin-archetype-application -DarchetypeVersion=7.x.x -DgroupId=org.clayman -DartifactId=site -Dversion=0.1 -Dpackaging=war”. Так как Apache Maven поддерживает принцип Convention over Configuration (соглашения прежде конфигурации, то есть структура проекта фиксирована), то после выполнения команды будет создан готовый каркас приложения. На рисунке ниже можно увидеть структуру проекта.

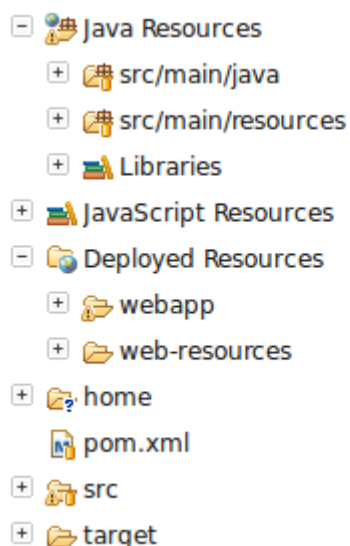


рис. 5. Структура проекта.

Основные папки и файлы проекта:

1. `src/main/java` – здесь хранятся java-классы.
2. `src/main/resources` – файлы всех настроек, ресурсов и конфигураций. Например, конфигурация библиотеки Hibernate.
3. `Libraries` – jar файлы всех библиотек проекта. Там хранятся как библиотеки автоматически загруженные maven'ом, так и добавленные вручную.
4. `Deployed resources` – все ресурсы, имеющие отношение к дальнейшему развертыванию веб-приложения; в случае если не используется конфигурация на основе java-аннотаций, там лежит дескриптор развертывания приложения `web.xml`, также там лежат sass файлы, характеризующие используемую в приложении визуальную тему. Она изменяет фон приложения, внешний вид и цветовую гамму всех элементов интерфейса.
5. `pom.xml` файл – основной файл любого maven проекта, содержит полную информацию о сборке приложения, используемых плагинах, зависимостях и используемых для них репозиториях.

Так как при создании проекта использовался архетип, соответствующий Vaadin веб-приложению, то основная масса зависимостей и плагинов связанных с развертыванием и сборкой приложения уже добавлены в `pom` файл.

Для добавления библиотеки в проект, необходимо добавить зависимость в `pom` файл, делается это с помощью следующего блока:

```
<dependencies>

  <dependency>

    <groupId> </groupId>

    <artifactId> </artifactId>

    <version> </version>

  </dependency>

</dependencies>
```

Параметры значений `groupId`, `artifactId`, `version` можно узнать воспользовавшись сайтом основного maven репозитория [15] или с использованием специального плагина для

IDE, так, например, для библиотеки Hibernate эти значения будут равны org-hibernate, hibernate-core и 4.3.5 Final соответственно.

После этого Maven сам загрузит данную библиотеку и добавит ее в проект. Для сборки проекта используется команда mvn package. Эта команда осуществляет сборку проекта в соответствии с тем, как это описано в pom-файле, тип собранного проекта зависит от параметра <packaging> </packaging>. Для веб-приложений используется тип упаковки Web Application File или сокращенно war. Такой же тип упаковки применяется в данной работе.

Файл с расширением .war представляет из себя файл, описывающий, как веб-приложение упаковывается в соответствии со спецификацией java-сервлетов. При правильно настроенной сборке является самодостаточным и может быть развернут на любом контейнере java-сервлетов [9].

Отправной точкой веб-приложения Vaadin является класс пользовательского интерфейса MainUI, у него имеется внутренний класс Servlet, который с помощью аннотации описывает как запускается приложение, пример конфигурации сервлета приложения –

```
public class MainUI extends UI {

    @WebServlet (value = "/*", asyncSupported = true)

    @VaadinServletConfiguration (ui = MainUI.class, widgetset =
"org.clayman.widgetset.MyWidgetSet")

    public static class Servlet extends VaadinServlet { }

    protected void init (VaadinRequest request) {

        <блок инициализации приложения>

    }

}
```

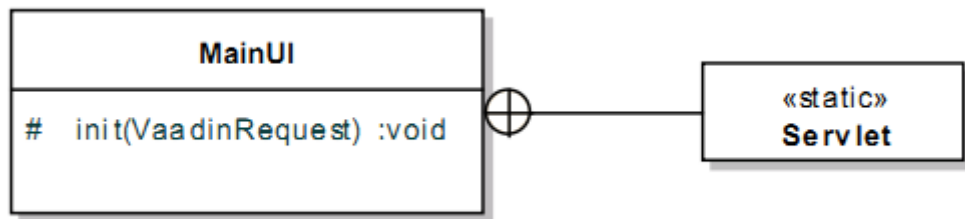


Рис. 6. Запуск приложения.

После того, как приложение будет запущено, будет вызван метод `init()`.

3.2.2. Пакетная обработка

Ввиду отсутствия полной инфраструктуры системы (агентов, коллекторов и прочих периферийных устройств) тесты производились на одной машине, а данные sFlow были получены в виде текстовых файлов и перенесены на локальный коллектор. Для переноса данных из текстовых файлов в локальный коллектор было написано вспомогательное приложение. При интеграцию в реальную систему основным отличием будет подключение к удаленной базе данных, а не к локальной. Также возможно использование другой СУБД, предназначенной для обработки большого количества данных. Благодаря гибкости и простоте библиотеки Hibernate это исправляется изменением пары строк конфигурационного файла.

В библиотеке Hibernate попытка поместить в базу данных больше количество записей (больше 50 000) в скором времени завершится ошибкой `OutOfMemoryException` – исключение, вызываемое, когда виртуальная java-машина не может создать объект из-за недостатка памяти и, в то же время, невозможно освободить достаточное количество памяти с помощью сборщика мусора. Данная ошибка возникает по причине того, что объект сессии кэширует все добавленные объекты.

Метод решения данной проблемы имеет название пакетная обработка (`batch processing`). Пакетная обработка в библиотеке Hibernate осуществляется следующим образом, в первую очередь в файле конфигурации необходимо добавить следующее свойство:

`<property name = “hibernate.jdbc.batch_size”> 50 </property>`, значение свойства – размер пакета, должно лежать в пределах от 10 до 50.

...

```
while (read(...)) {
```



```

if (count >= 50) {
    session.flush();
    session.clear();
    count = 0;
}
session.save(new Sflow(...));
count++;
}
transaction.commit();
...

```

Основная суть пакетной обработки в hibernate заключается в выделенных строчках, flush() – в принудительном порядке применяет еще невыполненные действия, которые хранятся в сессии, к текущей базе данных, clear() – полностью очищает текущую сессию[6].

3.2.3. MVP

Ввиду отсутствия реализации паттерна в фреймворке, он был реализован в приложении с нуля. Структура классов любого java-приложения основана на пакетах, пакет это такая группирующая сущность, объединяющая различные файлы (в данном случае java-классы) по какому-либо общему признаку [1, 12]. В рамках данной работы были выделены три пакета, концептуально и идеологически соответствующие трем компонентам паттерна Model-View-Presenter:

1. org.clayman.model:
 - a. PortUtil – занимается преобразованием порта в имя приложения, хранит хэш-карту с этими значениями.
 - b. QueryModel – класс, хранящий данные о параметрах запроса.
 - c. QueryResult – класс, описывающий формат результата запроса.
 - d. QueryService – класс, занимающийся конфигурацией и исполнением запроса.
 - e. Sflow – класс-сущность данных sFlow.

f. SflowManager – класс, отвечающий за конфигурацию работы механизма ORM и получение объекта SessionFactory.

2. org.clayman.presenter:

a. Presenter – промежуточное звено между моделью и представлением.

3. org.clayman.view:

a. MainUI – основной класс приложения, “точка запуска”, содержит в себе код инициализации всех компонент приложения, а также конфигурацию сервлета.

b. View – класс, описывающий графический интерфейс приложения.

c. ViewHandler – интерфейс обработчика пользовательских действий.

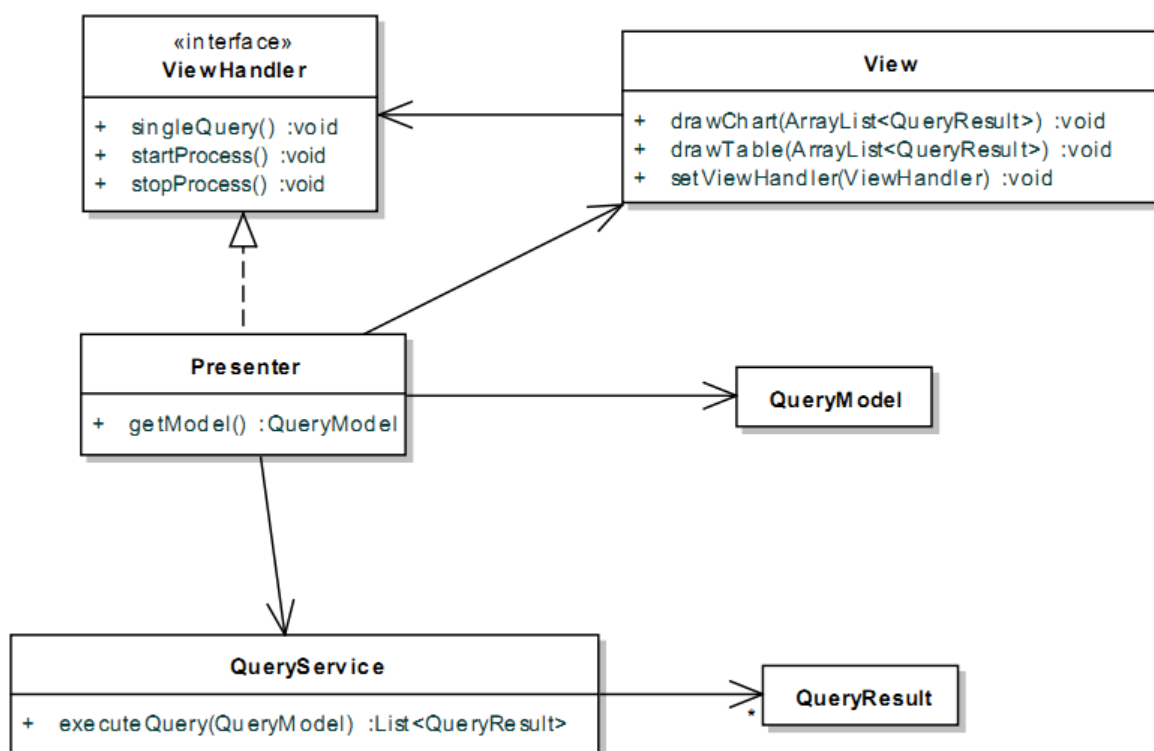


Рис. 7. Реализация MVP.

На рис.7 изображена реализация подхода Model-View-Presenter в приложении. Presenter реализует интерфейс ViewHandler. View использует обработчик ViewHandler, а если быть точнее, то его реализации Presenter для обработки пользовательских действий. На кнопки вешаются обработчики, при их нажатии, presenter вызывает соответствующие методы, переопределенные из ViewHandler:

```

executeButton.addClickListener(new ClickListener() {

    public void buttonClick(ClickEvent event) {

        viewHandler.startProcess();

    }

});

```

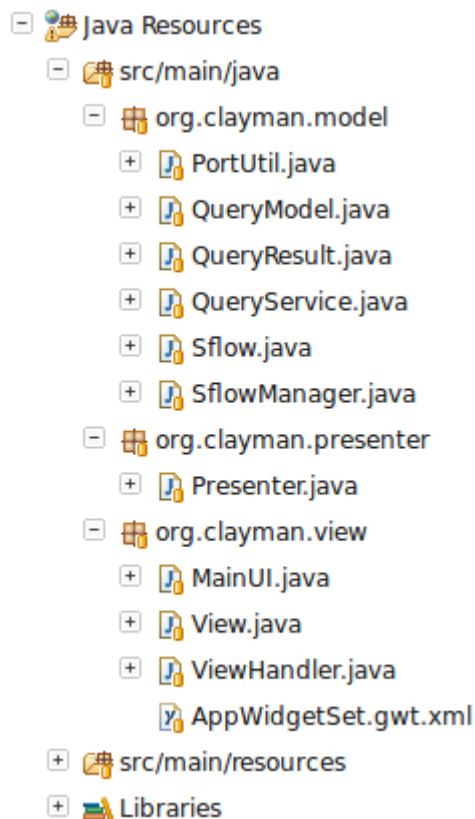


Рис. 8. Java-ресурсы проекта.

В блоке инициализации приложения создадим объект класса Presenter, добавим пользовательский интерфейс, установим обработчик пользовательских событий следующим образом [4, 5]:

```

protected void init(VaadinRequest request) {

    View view = new View();

    QueryService queryService = QueryService.getInstance();

```

```

        Presenter presenter = new Presenter(view, queryService); /* создание ссылок в
презентере на модель и на представление */

        view.setHandler(presenter); // добавление обработчика

        setContent(view); /* установление в качестве пользовательского интерфейса
объект класса View */

    }

```

3.2.4. Конфигурирование и выполнение запроса

Пользователь формирует свой запрос посредством графического интерфейса, на основании которого создается объект класса `QueryModel`, который хранит информацию о типе запроса и всех заданных значениях. Презентер формирует этот объект посредством метода `getModel()`, используя доступ ко всем элементам интерфейса, в дальнейшем отправляет этот объект на разбор, после чего конфигурируется запрос к базе данных.

Существует три способа создания запроса к базе данных с использованием библиотеки `Hibernate`:

1. `SQL (Structured Query Language)` – обыкновенный запрос на языке таблицы БД, конфигурация запроса в коде сводится к сборке одного большого выражения посредством конкатенация множества строк `sql`-кода. В случае большого запроса уменьшает читаемость кода и увеличивает его громоздкость, которая и без того присуща большим запросам.
2. `HQL (Hibernate Query Language)` – позволяет писать запросы на `SQL`-подобном языке, только вместо имен таблиц и атрибутов базы данных используются имена, которые фигурируют в объектной модели.
3. `Criteria` – объект запроса. Запрос создается посредством добавления объектов класса `Criterion` следующим синтаксисом – `criteria.add(Criterion criterion)`, где `Criterion` – объектно-ориентированное представление ограничения запроса. Данный подход позволяет добавлять ограничения в любом порядке, например вначале указать группировку, а потом равенство атрибута, дальнейшей

конфигурацией займется библиотекой, что позволяет программисту писать код, как ему удобно в соответствии с бизнес логикой приложения [6].

Ввиду большой гибкости и удобства, в реализации данной работы был выбран третий вариант.

Для выполнения любых действий над базой данных с использованием Hibernate, нужно сделать следующие вещи:

1. Создать файл конфигурации.
2. Создать класс сущности.
3. Инстанцировать фабрику сессий.

Рассмотрим каждый из пунктов подробнее:

Написание файла конфигурации. Файл конфигурации hibernate представляет собой xml-файл, в котором описана основная информация об используемой базе данных – диалект SQL, jdbc-драйвер для подключения к базе, url подключения к базе данных, имя пользователя и пароль этого пользователя. Помимо этого необходимо указать полные имена (полное имя пакета + имя класса) классов сущности, для которых будет осуществляться объектно-реляционное отображение. В соответствии с правилами хорошего тона и негласных конвенций принято хранить все конфигурации в папке /src/main/resources.

Конфигурация, используемая в данной работе:

```
<hibernate-configuration>
  <session-factory>
    <property name = "hibernate.dialect"> org.hibernate.dialect.MySQLDialect
  </property>
    <property name = "hibernate.connection.driver_class"> com.mysql.jdbc.Driver
  </property>
    <property name = "hibernate.connection.url"> jdbc:mysql://localhost/sflow
  </property>
    <property name = "hibernate.connection.username"> root </property>
    <property name = "hibernate.connection.password"> root </property>
    <mapping class = "org.clayman.model.Sflow"/>
  </session-factory>
</hibernate-configuration>
```

</session-factory>

</hibernate-configuration>

Создание класса сущности. Чтобы библиотека поняла, что данный класс является сущностью, его надо пометить с помощью двух аннотаций - @Entity и @Table(name = "tablename"). Класс должен иметь конструктор без параметров, а все поля класса быть объявлены приватными, иметь геттер и сеттер, а также помечены аннотациями @Column(name = "columnName"). Помимо этого, поле ID должно быть дополнительно помечено аннотациями @Id, @GeneratedValue(generator = "generatorName"), @GenericGenerator(name = "generatorName", strategy = "strategyType"). Параметр аннотации strategy определяет логику изменения поля Id, можно предоставить hibernate самому установить тип в соответствии с выбранной базой данных, тогда это поле должно принимать значение "native". В данной работе используется инкрементная стратегия, для нее параметр strategyType принимает значение "increment".

Для получения фабрики сессий, используется следующая цепочка вызовов:

```
SessionFactory sf = new Configuration().configure().buildSessionFactory();
```

Объект класса Configuration – объект, позволяющий приложению определить все необходимые свойства для создания фабрики сессий на основе файла конфигурации. Метод configure() без параметров ожидает, что конфигурация будет лежать в папке /src/main/resources с названием hibernate.cfg.xml. После создания фабрики сессий, можно открывать сессию. Сессия является фабрикой для создания объектов класса Criteria.

После всех этих действий, можно приступить к работе с базой данных, для этого необходимо создать сессию и запрос -

```
Session session = sessionFactory.openSession();
```

```
Criteria criteria = session.createCriteria(ClassName.class); // где classname – короткое имя класса сущности, например Sflow.
```

Существует один основной метод для работы с Criteria: criteria.add(Criterion criterion);

Основные реализации интерфейса Criterion и их назначение:

1. Restrictions.like(имя атрибута, шаблон поиска) – сравнение по шаблону.

2. `Restrictions.eq(имя атрибута, значение)` – проверка на равенство атрибута конкретному значению.
3. `Restrictions.or(Criterion, ...)` – логическое “или” между несколькими ограничениями.
4. `Restrictions.between(имя атрибута, нижнее значение, верхнее значение)` – проверка на принадлежность отрезку значений.
5. `Restrictions.in(имя атрибута, массив значений)` – проверка на принадлежность массиву значений.

Для создания группировки используются проекции – `criteria.setProjection(Projections.sum(имя атрибута));`

Для сортировки по атрибуту используется - `criteria.addOrder(Order.desc(имя атрибута));`

Для получения результата запроса необходимо вызвать метод `criteria.list()`, он вернет список записей, удовлетворяющих запросу [7].

Диаграмма последовательности, соответствующая исполнению запроса в системе изображена на рисунке 9.

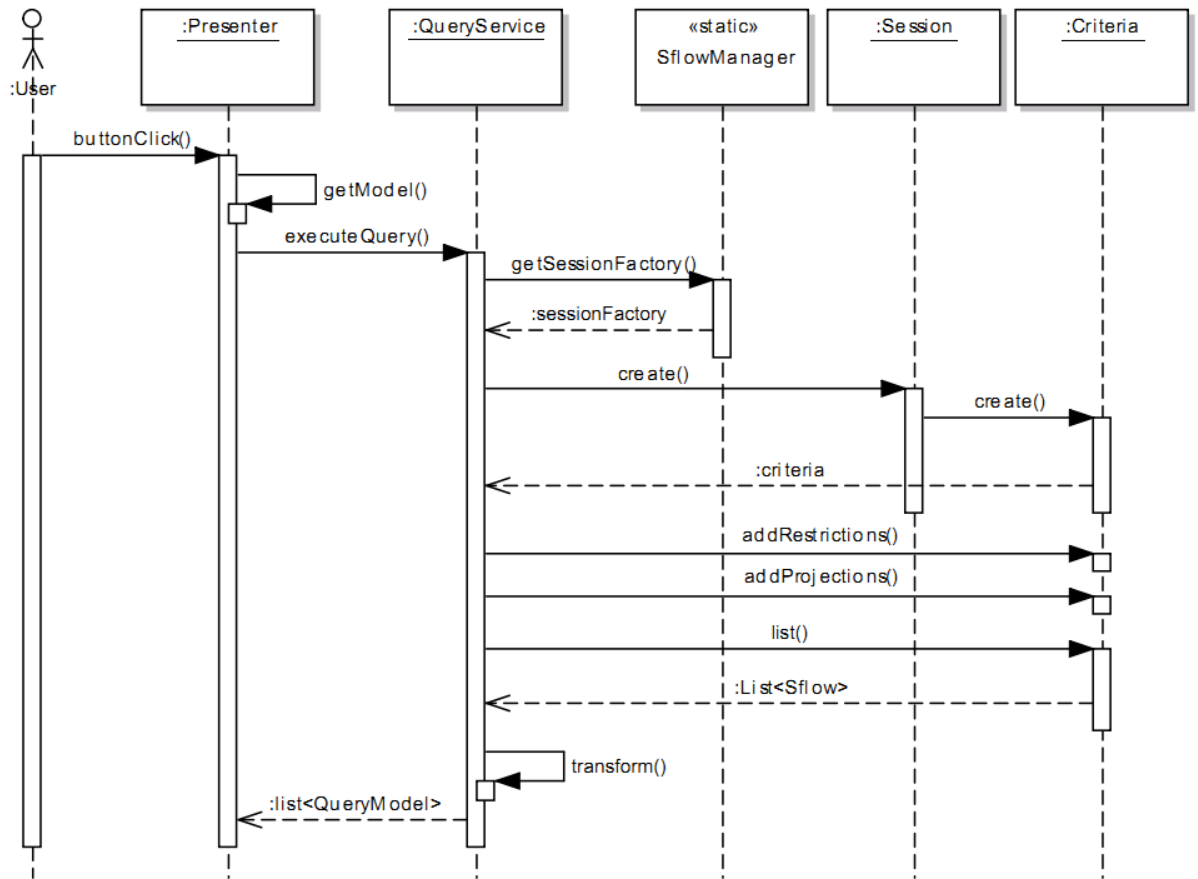


Рис. 9. Диаграмма последовательности выполнения запроса.

3.2.5. Обновление пользовательского интерфейса

По умолчанию пользовательский интерфейс не обновляется мгновенно при выполнении длительных действий в потоке. Для того, чтобы внешний вид приложения обновлялся сразу после выполнения запроса, используется подход именуемый server push.

Для активации этого подхода необходимо, чтобы класс UI, для которого необходимо обновление, был помечен аннотацией `com.vaadin.annotations.Push`. Иными словами описание класса приобретет следующий вид:

```

@Push

public class MainUI extends UI {...}
  
```

Теперь класс готов к мгновенному отправлению изменений на сторону клиента. Помимо этого необходимо правильным образом обновлять этот интерфейс, для этого существует специальный метод, который имеет следующую сигнатуру:


```
public java.util.concurrent.Future<java.lang.Void> access (java.lang.Runnable runnable)
```

Метод предоставляет возможность обновления графического интерфейса из потока, который не является обработчиком любой из его компонент. Передаваемый параметр `java.lang.Runnable` определяет каким именно образом будет изменяться внешний вид приложения [4,5]:

```
boolean volatile isRun = true;

while(isRun) {

    MainUI.getCurrent().access(new Runnable() {

        @Override

        public void run() {

            view.drawChart(result);

            view.drawTable(result);

        }

    });

}
```

Для остановки потока применяется подход с использованием логической переменной и проверка ее истинности в цикле. Для остановки потока, значение переменной изменяется на “ложь” и поток обновления останавливается. Стоит заметить, что поток не останавливается мгновенно, это происходит лишь после того, как будет выполнен текущий запрос.

3.2.6. Инструмент изображения диаграмм

Для изображения графиков используется аддон – `Vaadin Charts`, он позволяет быстро и удобно рисовать диаграммы, а также добавлять их в интерфейс приложения.

Для использования аддона необходимо добавить его в зависимости проекта с помощью внесения соответствующих изменений в `pom` файл:

```
<dependency>

    <groupId> com.vaadin.addon </groupId>
```

```
<artifactId> vaadin-charts </artifactId>
```

```
<version> 1.1.5 </version>
```

```
</dependency>
```

Для создания диаграммы необходимо инстанцировать объект класса `Chart`, а в конструкторе указать тип диаграммы посредством перечисления(enumeration) следующим образом:

```
Chart chart = new Chart(ChartType.PIE);
```

За отображаемые данные в первую очередь отвечает объект класса `Configuration`:

```
Configuration config = chart.getConfiguration();
```

Для добавления данных в диаграмму используется объект класса `DataSet`:

```
DataSet series = new DataSet();
```

```
series.setName("seriesName");
```

```
series.add(new DataSetItem(name, value));
```

Остается лишь добавить диаграмму в текущий компонент:

```
component.addComponent(chart);
```

3.2.7. Определение приложения по номеру порта и всех адресов подсети

Из источника [13] был получен список всех зарегистрированных портов в ассоциации IANA (администрация адресного пространства интернет). Далее этот файл был разобран и составлены пары ключ-значение вида номер_порта-имя_приложения. Данные записи были помещены в хэш-карту, которая хранится в классе `PortUtils`. В любой нужный момент для замены номера порта на имя приложения вызывается статический метод `PortUtils.transform(номер порта)`, в качестве возвращаемого значения получаем имя приложения или номер порта, если в хэш-карте отсутствует данный ключ.

Для определения всех адресов подсети используется библиотека `org.apache.commons.net`, для получения массива всех адресов подсети по известному адресу и количеству бит в маске подсети, необходимо инстанцировать объект класса `SubnetUtils`:

```
SubnetUtils utils = new SubnetUtils(ip + "/" + mask);
```

```
SubnetInfo info = utils.getInfo();
```

```
String[] addresses = info.getAllAddresses();
```

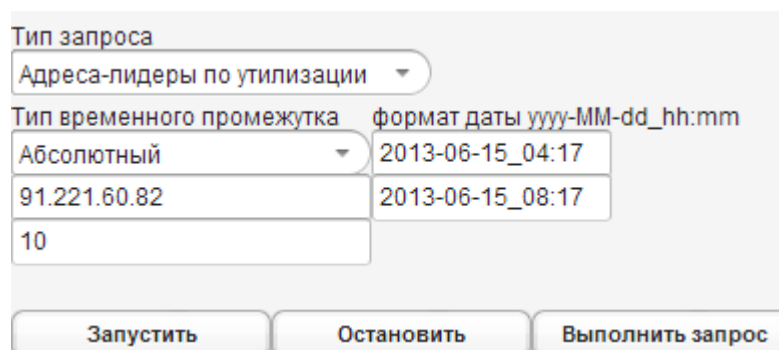
Теперь в массиве `addresses` лежат все адреса данной подсети, для проверки принадлежности массиву значений в запросе используется следующая конструкция:

```
Restrictions.in("ip", addresses);
```

 где `ip` – имя атрибута таблицы БД.

4. Руководство пользователя

На рисунке ниже изображена панель конфигурирования запроса, с ее помощью определяется тип выполняемого запроса и его основные параметры.



Тип запроса	
Адреса-лидеры по утилизации	
Тип временного промежутка	
Абсолютный	формат даты yyyy-MM-dd_hh:mm
91.221.60.82	2013-06-15_04:17
10	2013-06-15_08:17

Запустить Остановить Выполнить запрос

Рис. 10. Пример сконфигурированного запроса.

В первую очередь необходимо выбрать тип запроса, это можно сделать в первом выпадающем списке, каждый запрос характеризуется свойственными ему параметрами, всего существует три вида запросов.

Во втором выпадающем списке выбирается тип временного отрезка, он может принимать два значения:

1. Абсолютный, задается двумя значениями – нижней и верхней границей интервала, все записи подходящие под данный интервал будут добавлены в итоговый результат.
2. Относительный, может принимать два значения – за последний час или за последние 30 минут, итоговый временной промежуток вычисляется следующим образом [текущее время – 30(60) минут; текущее время].

Управляющие кнопки:

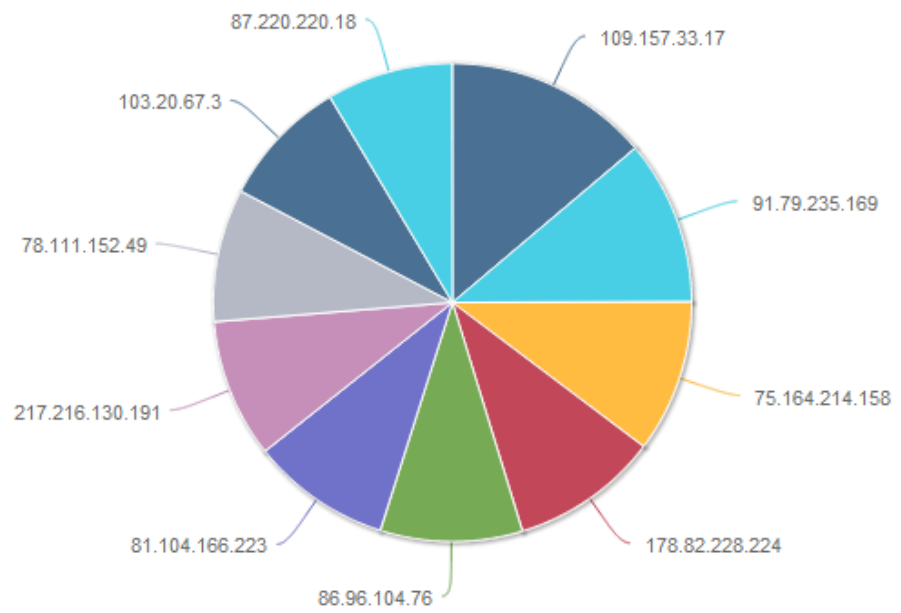
1. Запустить – выполняет запрос и обновляет результаты своей работы раз в 1 минуту, результат работы будет меняться, если в базу данных добавляются новые значения.
2. Остановить – останавливает работу предыдущего режима, следует отметить что мгновенной остановки не произойдет, работа остановится после того, как выполнится текущий запрос.

3. Выполнить запрос – единожды выполняет запрос и отображает полученный результат на экране.

Рассмотрим подробнее типы запросов:

1. Адреса-лидеры по утилизации – задаем временной интервал, предварительно выбрав его тип, выбираем адрес (подсеть) и количество строк результата к выводу. Данный запрос выводит адреса тех компьютеров, которые отправили на выбранный адрес максимальное количество бит информации, а также те адреса, которые получили наибольшее количество информации от выбранного сервера. Помимо этого можно искать по подсети, для этого необходимо после IP-адреса указать слэш и указать количество бит в маске подсети, например '192.168.11.10/21', в таком случае будут выведены компьютеры, которые получили и адресовали всем компьютерам данной подсети наибольшее количество информации.

Отправлено в



АДРЕС	ПАКЕТОВ ОТПРАВЛЕНО	БИТ ОТПРАВЛЕНО	%
ИТОГО	169 331	107 556 901	100
109.157.33.17	1 725	2 519 756	2
91.79.235.169	1 402	2 024 437	1
75.164.214.158	1 281	1 878 602	1
178.82.228.224	1 494	1 817 916	1
86.96.104.76	1 212	1 749 993	1
81.104.166.223	1 282	1 719 981	1
217.216.130.191	1 181	1 710 088	1
78.111.152.49	1 130	1 634 850	1
103.20.67.3	1 101	1 591 440	1

Рис. 11. Информация о пакетах, отправленных с выбранного адреса.

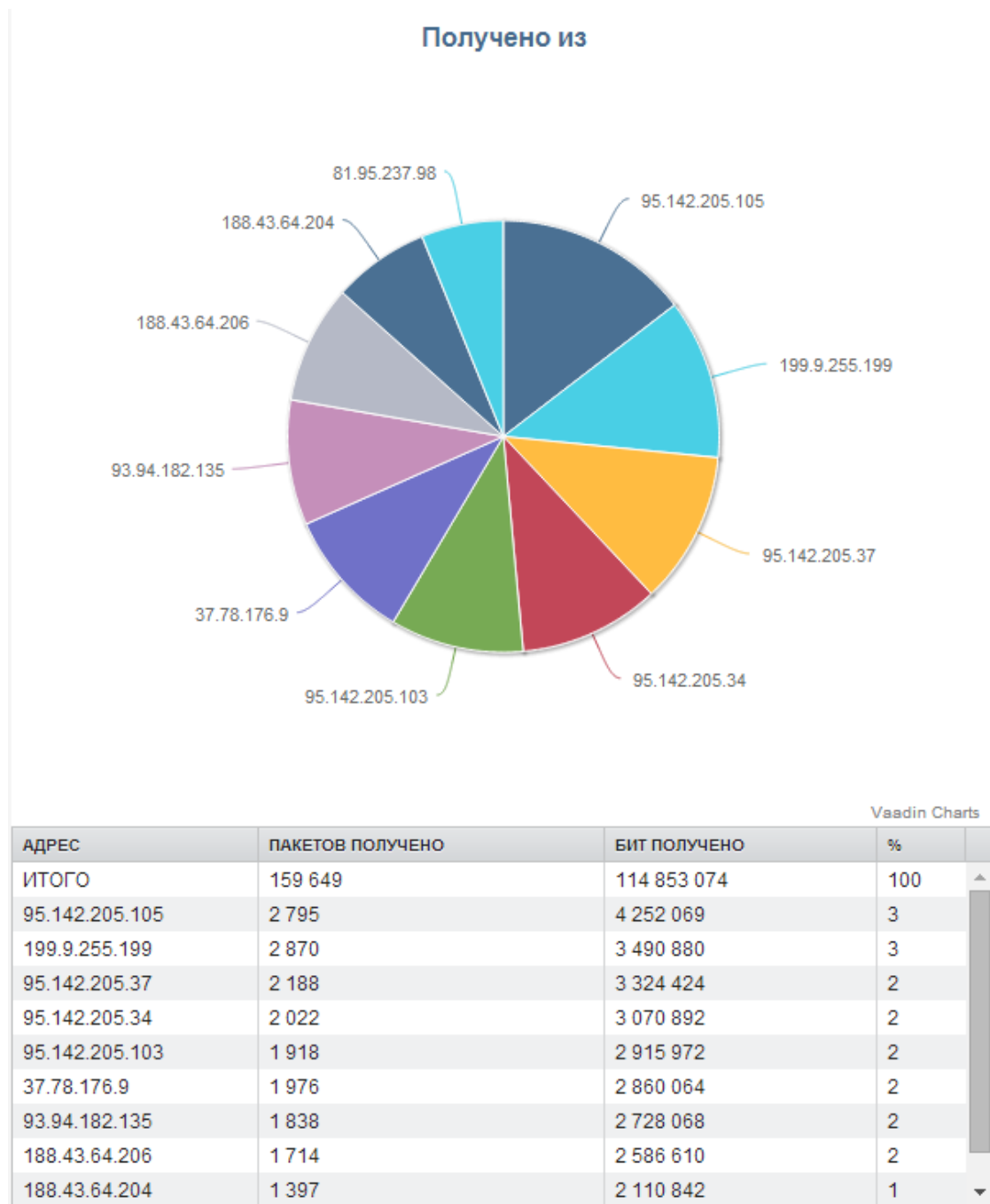
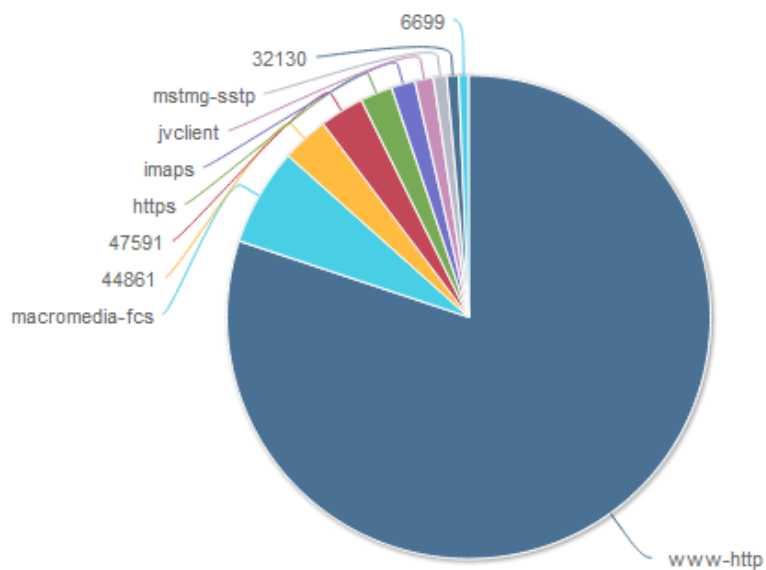


Рис. 12. Информация о пакетах, полученных данным ip-адресом.

- Порты-лидеры по утилизации – запрос аналогичен по входным параметрам, только вместо адресов выводятся порты, по которым было передано и получено наибольшее количество информации, все известные приложению номера портов преобразуются в имена приложений, которые производили обмен.

Получено из

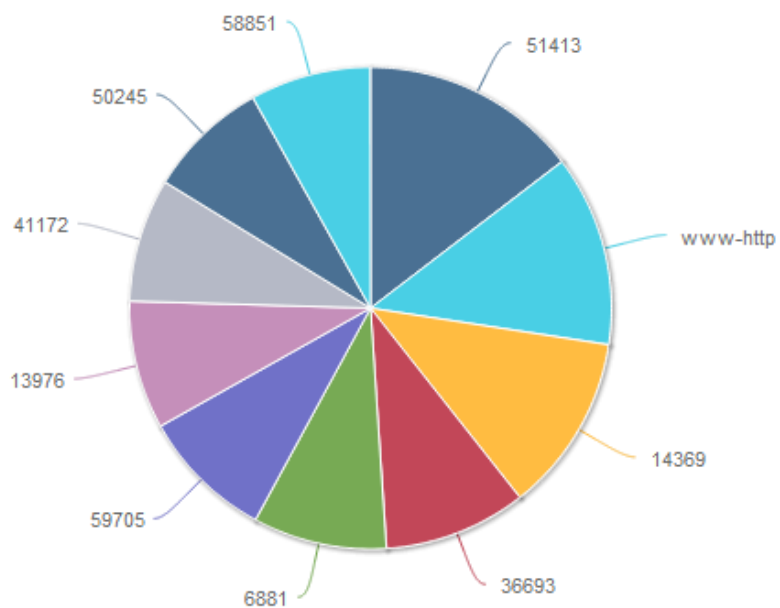


Vaadin Charts

ПРИЛОЖЕНИЕ	ПАКЕТОВ ПОЛУЧЕНО	БИТ ПОЛУЧЕНО	%
ИТОГО	159 649	114 853 074	100
www-http	50 887	73 628 121	64
macromedia-fcs	4 981	5 990 852	5
44861	1 979	2 860 250	2
47591	1 838	2 728 068	2
https	2 654	1 973 793	1
imaps	1 345	1 467 243	1
jvclient	968	1 137 247	0
mstmg-sstp	721	810 645	0
32130	511	723 378	0

Рис. 13. Порты, по которым было получено наибольшее количество пакетов.

Отправлено в



Vaadin Charts

ПРИЛОЖЕНИЕ	ПАКЕТОВ ОТПРАВЛЕНО	БИТ ОТПРАВЛЕНО	%
ИТОГО	169 331	107 556 901	100
51413	3 265	3 084 195	2
www-http	29 818	2 707 756	2
14369	1 725	2 519 756	2
36693	1 402	2 024 437	1
6881	3 496	1 896 099	1
59705	1 285	1 882 863	1
13976	1 494	1 817 916	1
41172	1 212	1 749 993	1
50245	1 282	1 719 981	1

Рис. 14. Порты, по которым было отправлено наибольшее количество пакетов.

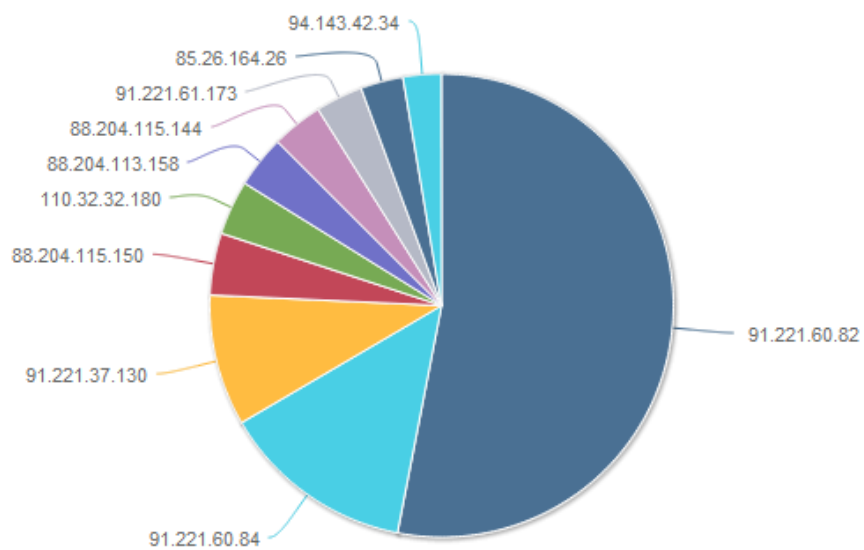
3. Лидеры по утилизации порта – выбирается порт, остальные параметры аналогичны тем, что были в предыдущих запросах:

Тип запроса	
Лидеры по утилизации порта	
Тип временного промежутка	формат даты yyyy-MM-dd_hh:mm
Абсолютный	2013-06-15_04:17
80	2013-06-15_08:17
10	
подсеть	
Запустить	
Остановить	
Выполнить запрос	

Рис. 15. Конфигурация запроса “лидеры по утилизации порта”.

Результат работы запроса представляет собой адреса, которые отправили или получили по данному порту наибольшее количество пакетов за выбранный промежуток времени. Если указать подсеть, то в утилизации порта будут учитываться только компьютеры данной подсети.

Отправлено в



Vaadin Charts

АДРЕС	ПАКЕТОВ ОТПРАВЛЕНО	БИТ ОТПРАВЛЕНО	%
ИТОГО	281 548	384 332 011	100
91.221.60.82	50 887	73 628 121	19
91.221.60.84	13 529	18 747 232	4
91.221.37.130	10 444	12 692 355	3
88.204.115.150	4 242	6 016 350	1
110.32.32.180	3 496	5 278 960	1
88.204.113.158	3 442	5 045 183	1
88.204.115.144	3 523	4 956 391	1
91.221.61.173	3 043	4 566 821	1
85.26.164.26	2 913	4 130 634	1

Рис. 16. Результат работы запроса “Лидеры по утилизации порта”.

Заключение

В рамках данной работы были выполнены следующие задачи:

1. Изучен подход к мониторингу высоконагруженных сетей с использованием технологии sFlow
2. Изучены средства разработки веб-приложений на платформе Java
3. Реализована архитектура приложения в соответствии с паттерном Model-View-Presenter
4. Разработаны основные модули системы, реализован основной функционал, система организована таким образом, что она легко способна наращивать функционал.

Таким образом, поставленная цель работы – разработка прототипа системы мониторинга – достигнута.

В дальнейшем планируются работы по трем основным направлениям:

1. Добавление возможности сохранять отчеты и предоставление постоянного доступа к ним по URL ссылке.
2. Возможность формирования отчетов на основе существующих отчетов.
3. Повышение дружелюбности интерфейса.

Список использованных источников

1. Эккель Б. Философия Java : пер. с англ. / Б.Эккель ; под ред. Д.В. Лощинина. – 4-е изд. – СПб. : Питер, 2010. – 465
2. sFlow [Электронный ресурс] – URL: <http://www.sflow.org/> (дата последнего обращения: 10.05.2014)
3. Vaadin book [Электронный ресурс] – URL: <https://vaadin.com/book> (дата последнего обращения: 6.05.2014)
4. Vaadin Wiki [Электронный ресурс] – URL: <https://vaadin.com/wiki/-/wiki/Main/Vaadin+7> (дата последнего обращения: 17.05.2014)
5. Vaadin Framework API Index [Электронный ресурс] – URL: Vaadin Wiki [Электронный ресурс] – URL: <https://vaadin.com/wiki/-/wiki/Main/Vaadin+7> (дата последнего обращения: 17.05.2014)
6. Hibernate Reference Documentation [Электронный ресурс] – URL: <http://docs.jboss.org/hibernate/orm/4.3/manual/en-US/html/> (дата последнего обращения: 24.04.2014)
7. Hibernate Api JavaDoc [Электронный ресурс] – URL: <http://docs.jboss.org/hibernate/orm/4.3/javadocs/> (дата последнего обращения: 24.04.2014)
8. Apache Tomcat Documentation [Электронный ресурс] – URL: <http://tomcat.apache.org/tomcat-7.0-doc/index.html> (дата последнего обращения: 16.05.2014)
9. Maven Documentation [Электронный ресурс] – URL: <http://maven.apache.org/guides/> (дата последнего обращения: 13.04.2014)
10. Graphical User Interface Architectures [Электронный ресурс] – URL: <http://martinfowler.com/eaDev/uiArchs.html> (дата последнего обращения: 10.05.2014)
11. Model-View-Presenter [Электронный ресурс] – URL: <http://www.wildcrest.com/Potel/Portfolio/mvp.pdf> (дата последнего обращения: 10.05.2014)

12. Хорстманн К. Java 2. Том 1. Основы : пер. с англ. Мухина Н.А. / К. Хорстманн, Г. Корнелл ; под редакцией Ю.Н. Артеменко. – 8-е изд. – МСК. : Вильямс, 2012. - 813
13. Service names port numbers [Электронный ресурс] – URL: <https://www.iana.org/assignments/service-names-port-numbers/> (дата последнего обращения: 25.05.2014)
14. Java Platform Standard Edition 7 API Specification [Электронный ресурс] – URL: <http://docs.oracle.com/javase/7/docs/api/> (дата последнего обращения: 25.05.2014)
15. Maven repository [Электронный ресурс] – URL: <http://mvnrepository.com/> (дата последнего обращения: 02.05.2014)

Приложение А. Руководство по установке.

Установка и подготовка контейнера сервлетов Apache Tomcat для *nix-подобных систем:

1. Для начала необходимо скачать последнюю версию дистрибутива сервера с официального сайта <http://tomcat.apache.org/download-70.cgi>. При отсутствии графической оболочки загрузка осуществляется с использованием команды `wget -'wget http://apache-mirror.rbc.ru/pub/apache/tomcat/tomcat-7/v7.0.53/bin/apache-tomcat-7.0.53.tar.gz'`.
2. Полученный архив распаковать посредством утилиты `tar` следующей командой - `'tar -xvf apache-tomcat-7.0.53.tar.gz'`.
3. С помощью команды `mv` переместить содержимое архива в рабочий каталог. Предварительно создав его - `'mkdir /opt/apache'`, команда для перемещения - `'mv apache-tomcat-7.0.53 /opt/`.
4. Добавить переменную окружения `CATALINA_HOME`. Для этого отредактировать файл — `'nano ~/.profile'` и добавить в конец файла следующие строки `'CATALINA_HOME=/opt/apache/apache-tomcat-7.0.53'` и `'export CATALINA_HOME'`, первая строка объявляет переменную и присваивает ей значение, а вторая делает ее переменной окружения и предоставляет возможность пользоваться ею. Добавление этих строчек в файл `.profile` обеспечит создание этой переменной при перезагрузке системы.
5. Для запуска сервера выполнить команду `$CATALINA_HOME/bin/startup.sh`. Сервер запустится и оповестит об этом соответствующим сообщением в консоли.
6. Для установки веб-приложения на сервере, необходимо поместить `.war` файл в папку `/opt/apache/apache-tomcat-7.0.53/webapps`. Если сервер запущен, то спустя некоторое время он обнаружит добавленный файл и развернет веб-приложение. Проверить работоспособность приложения извне можно по адресу *адрес_сервера:8080/имя_war_файла*. Или *localhost:8080/имя_war_файла*, если проверка осуществляется с этого же компьютера.
7. При наличии исходного кода приложения без готового `war` файла, необходимо, находясь в корневой папке проекта выполнить команду `mvn package`. `war` файл будет создан и располагаться в папке `/target`.

8. Отправить файл на удаленный сервер можно с помощью команды `scp`, если файл лежит в папке с проектом, удаленный сервер имеет адрес 192.168.5.172, а имя пользователя `administrator`, то команда будет выглядеть следующим образом - '`scp полный/путь/к/имени/файла administrator@адрес_сервера:/usr/apache/apache-tomcat-7.0.53/webapps'`, далее необходимо ввести пароль пользователя `administrator`. Файл поместится в данную папку и через некоторое время приложение будет развернуто [8].