Компонент SrsProxy

Руководство администратора и пользователя

ДИСКЛЕЙМЕР:

Программное обеспечение является встраиваемым компонентом и не имеет графического пользовательского интерфейса. Конечный пользователь ПО – разработчики программного обеспечения, системные администраторы.

Содержание

Аннотация

Настоящий документ содержит руководство для администратора по работе с программным обеспечением «Прокси» (далее, компонент прокси, прокси или ПО). Назначением документа является описание назначения, архитектуры, доступа, настроек и диагностирования/устранения ошибок ПО.

Назначение ПО

Назначение программы

Изделие реализует мини-платформу для перехвата, анализа и модификации потоков данных в протоколе HTTP, под-протоколах SOAP (XML), REST (JSON) и HTML.

Программное обеспечение является встраиваемым компонентом и не имеет графического пользовательского интерфейса. Конечный пользователь ПО – разработчики, системные администраторы.

Функции программы

ПО реализует мини-платформу для перехвата, анализа и модификации потоков данных в протоколе HTTP, включая SOAP (XML), REST (JSON) и HTML.

ПО принимает потоки данных на порты, заданные в конфигурации.

При получении входного пакета данных, ПО, пользуясь набором правил, заданных в конфигурации, определяет компоненты стратегии, отвечающие за анализ и модификацию запроса и ответа полученного пакета данных, а также адрес и порт назначения пакета. Стратегии могут указывать протокол, URL, заголовки HTTP и прочие характеристики для определения типа входных пакетов, которые эти стратегии в состоянии обрабатывать. Дав выбранным компонентам стратегии запроса возможность обработать входной пакет данных, ПО направляет пакет данных запроса по адресу и порту назначения, после чего ожидает ответа. Получив ответ, прокси предоставляет компоненту стратегии ответа возможность обработать пакет данных ответа и затем отправляет обработанный ответ к источнику запроса.

В случае, когда прокси не может определить стратегии, отвечающие за анализ и модификацию запроса и ответа полученного пакета данных, ПО пересылает запросы и ответы между источником и назначением без какого-либо анализа и модификации, используя при этом наиболее эффективный алгоритм для пересылки (streaming).

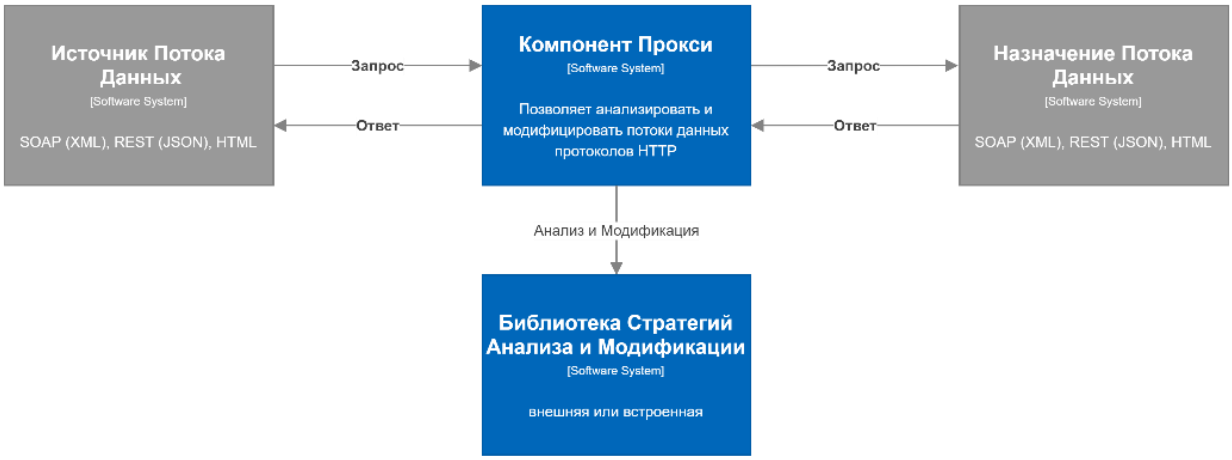
Компоненты стратегии, отвечающие за анализ и модификацию запроса и ответа, не являются частью ПО, но могут быть загружены в ПО из библиотеки в формате JAR в процессе запуска прокси. Альтернативно, разработчики могут использовать компонент прокси как базу для разработки ПО, которое объединяет прокси и стратегии в одном модуле с целью простоты развертывания и эксплуатации результирующего ПО - программное обеспечение.

Для кратких дополнительных сведений по стратегиям, см. Приложение C – Сведения о стратегиях анализа и модификации потоков данных.

Архитектура ПО

Назначение (контекст)

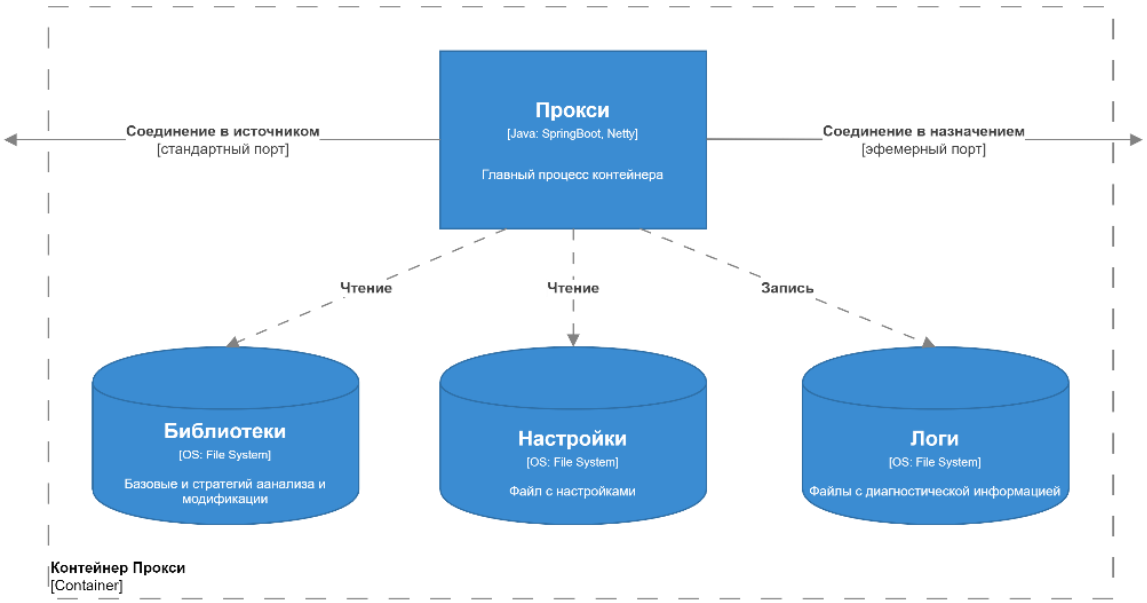
В бизнес-контексте ПО реализует мини-платформу для перехвата, анализа и модификации потоков данных в протоколе HTTP, под-протоколах SOAP (XML), REST (JSON) и HTML.



Системы потребители (источники потоков данных) взаимодействуют с системами обслуживания (назначение потоков данный) через ПО вместо того, чтобы взаимодействовать с друг другом непосредственно. Наличии ПО в этой цепочки коммуникаций делает возможными анализ и модификацию потока данных. Конкретные методы вышеупомянутых анализа и модификации производится стратегиями, которые могут быть встроены в ПО или, альтернативно, ПО может загружать из внешних библиотек.

Архитектура на уровне контейнера

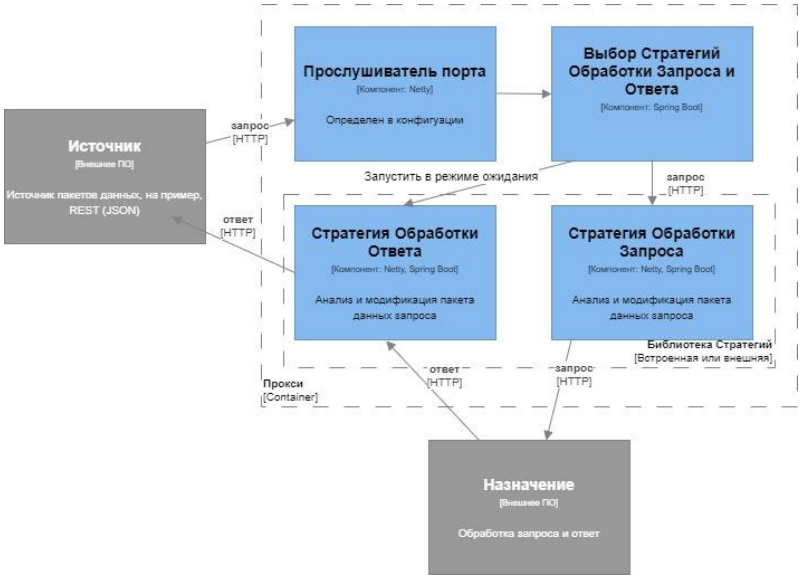
В контексте контейнера ПО состоит из главного процесса прокси, который использует базовые (системные) библиотеки наряду с библиотеками стратегий, файл конфигурации параметров ПО и систему логирования с ее собственным под-модулем конфигурации.



Архитектура на уровне компонентов

Главный компонент ПО состоит из следующих модулей:

Модуля прослушивателя портов и модуля выбора стратегий обработки запросов и ответов. Оба модуля используют SpringBoot для сборки и управления зависимостями, в то время как Netty используется для реализации эффективных сетевых функций.



Требования к аппаратным средствам

- Два виртуальных процессора с частотой 2.7 ГГц+,
- ОЗУ – не менее 2 Гб,
- ЖМД – не менее 10 Гб или в зависимости от конфигурации подсистемы логирования.

Установка, выполнение и доступ в ПО в среде JRE

Для выполнения ПО в среде JRE, в дополнении к необходимым техническим средствам, требуется наличие[1]:

1. Операционной системы CentOS 7.2, Red Hat Enterprise Linux Server 7.2 или Ubuntu 18.04 LTS
2. Open JDK версия 11 доступная в пути среды операционной системы. Например, на операционной системе Ubuntu 18.04 LTS, установка Open JDK 11 производится посредством выполнения: `sudo apt install default-jdk`.
3. Компонент прокси **http-proxy-${version}.jar** предоставленной производителем ПО в виде библиотеки JAR содержащую все необходимые библиотеки поддержки,
4. Файл **override.yaml** с настройками ПО которые отличаются от настроек поставляемых с компонентом прокси по умолчанию (см. Настройка Параметров ПО)

Само выполнение ПО осуществляется при помощи интерфейса командной строки (или с помощью скрипта) посредством следующей команды:

```
java -jar http-proxy-${version}.jar --spring.config.additional-location=file:${path-to-config}/override.yaml
```

где,

${version} – версия библиотеки компонента прокси, в настоящее время 1.0.0
${path-to-config} – путь к файлу с настройками ПО

При успешном старте компонент прокси выведен на консоль следующее сообщение:

```
Started Application in X.Y seconds (JVM running for X.Y)
```

После этого компонент прокси готов к получению запросов на порты, открытые во время запуска. В процессе работы компонент будет выводить всю отладочную информацию на консоль.

Настройка параметров ПО

ПО хранит все базовые настройки в файле **application.yaml,** который находится в корневом директории библиотеки JAR компонента. Пользователи ПО могут запросить полную версию **application.yaml** с каталогом всех настроек и их допустимых значений у производителя ПО. Для изменения параметров ПО пользователи могут создать дополнительный файл в формате YAML (например **override. yaml**) и поместить все желаемые настройку в данный файл, например:

```
anonymizer:
 routes:
  - name: 'SOAP'
    ingressPort: 2220
    resolver: 'Generic'
    services:
      - name: 'SOAP Service'
        baseUrl: 'https://some-service.com:443'
        path: '/custom'
```

[1] Для упрощения процесса установки Производитель ПО предоставляет скрипт для установки все необходимых системных компонентов.

Затем, пользователи могут указать на дополнительный файл настроек через опцию компонента `--spring.config.additional-location` как объяснено в разделе Установка, Выполнение и Доступ к ПО в Среде JRE.

Для логирования ПО использует библиотеку SLF4J, и настройка уровней логирования осуществляется способом который является стандартным для Spring Boot: https://www.baeldung.com/spring-boot-logging. Производитель предоставляет базовую конфигурацию компонента логирования по отдельному запросу.

Процедуры диагностики и устранения ошибок ПО

Подсистема логирования служит основным источником для диагностики ошибок. По умолчанию, подсистема логирования выводит основную информацию о потоках данных и стратегий по их обработкам:

Неправильная конфигурация, нерабочее состояние систем назначения поток данных и проблемы со стратегиями обработки потоков данных являются основными источниками проблем данного ПО. Во всех этих случаях ПО выводит отладочную информацию на уровне ERROR или WARNING. Дополнительно, лог содержит трассировки стека для всех исключительных ситуаций на уровне JVM и ПО.

В случае, когда необходима дополнительная отладочная информация, рекомендуется поменять уровень логирования с INFO на DEBUG.

Приложения

Приложение А – Изменение настроек ПО

Как было объяснено в разделе Установка ПО и Средств Доступа к ПО, базовые настройки ПО могут быть переопределены с командной строки с помощью файла **override.yaml** который имеет следующую структуру:

```
anonymizer:
  routes:
    - name: 'SOAP'
      ingressPort: 2220
      resolver: 'Generic'
      services:
        - name: 'SOAP Service'
          baseUrl: 'https://some-service.com:443'
          path: '/custom'
```

Основные изменения и добавления к **override.yaml** обычно состоят в определение новых или модификации существующих маршрутов (routes).

Приложение Б – Файл базовых параметров логирования

Нижеприведенный базовый файл настроек логирования показывает ряд логгеров, уровень которых можно менять в случае необходимости получения более детальной отладочной информации.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration scan="false" scanPeriod="10 seconds">
    <!-- use Spring default values -->
    <include resource="org/springframework/boot/logging/logback/defaults.xml"/>

    <appender name="Console"
              class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>${CONSOLE_LOG_PATTERN}</pattern>
            <charset>utf8</charset>
        </encoder>
    </appender>

    <conversionRule conversionWord="pcim"
                    converterClass="com.sabre.dae.anonymizer.logger.AnonymizerLogbackConverter"/>

    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%clr(%d{${LOG_DATEFORMAT_PATTERN:-yyyy-MM-dd HH:mm:ss.SSS}}){faint} %clr(${LOG_LEVEL_PATTERN:-%5p}) %clr(${PID:- }){magenta} %clr(---){faint} %clr([%15
.15t]){faint} %clr(%-40.40logger{39}){cyan} %clr(:){faint} %pcim%n${LOG_EXCEPTION_CONVERSION_WORD:-%wEx}</pattern>
            <charset>utf8</charset>
        </encoder>
    </appender>

    <appender name="ASYNC_CONSOLE"
              class="ch.qos.logback.classic.AsyncAppender">
        <discardingThreshold>0
        </discardingThreshold> <!-- default 20, means drop lower event when has 20% capacity remaining -->
        <appender-ref ref="STDOUT"/>
        <queueSize>1000</queueSize> <!-- default 256 -->
        <includeCallerData>false</includeCallerData><!-- default false -->
        <neverBlock>true</neverBlock><!-- default false, set to true to cause the
            Appender not block the application and just drop the messages -->
    </appender>

    <!-- LOG everything at INFO level -->
    <root level="INFO">
        <appender-ref ref="ASYNC_CONSOLE"/>
    </root>

    <logger name="anonymizer.in" level="INFO"/>
    <logger name="anonymizer.out" level="INFO"/>
    <logger name="anonymizer.proxy.in" level="INFO"/>
    <logger name="anonymizer.proxy.out" level="WARN"/>
    <logger name="anonymizer.response.content" level="DEBUG"/>
    <logger name="com.sabre.dae.anonymizer" level="INFO"/>

    <!-- logs exact replacements and personal data content, change to DEBUG for details -->
    <logger name="anonymizer.replacement" level="INFO"/>

    <!-- logs request content before and after processing (DEBUG), note that it won't log passthrough request contents -->
    <logger name="anonymizer.request.content" level="INFO"/>

    <!-- logs response content before and after processing (DEBUG), note that it won't log passthrough response contents -->
    <logger name="anonymizer.response.content" level="INFO"/>


    <logger name="io.netty" level="INFO"/>
    <logger name="org.springframework" level="INFO"/>
</configuration>
```

Наиболее полезные логгеры и их уровни:

Логгер	Уровень	Краткое описание диагностической информации
anonymizer.in, anonymizer.proxy.in	INFO	Детали части начала запроса (метод, URI, протокол, заголовки)
	DEBUG	Трассирование частей содержания и концовки запроса
anonymizer.out, anonymizer.proxy.out	INFO	Детали части начала ответа (метод, URI, протокол, заголовки)
	DEBUG	Трассирование частей содержания и концовки ответа
anonymizer.request.content	DEBUG	Содержание запроса
anonymizer.response.content	DEBUG	Содержание ответа

Приложение С – Сведения о стратегиях анализа и модификации потоков данных

ПО осуществляет поиск и подключение стратегий анализа и модификации потоков данных двумя способами:

Поиск Java классов реализующий интерфейсы **RequestStrategy** для запросов и **ResponseStrategy** для ответов. В случае запросов:

```
public interface RequestStrategy {
default String getName() { return this.getClass().getSimpleName(); }
boolean isApplicable(String url, @Nullable HttpHeaders headers);
void process(Object request, Queue<Consumer<SharedContextReference>> executionQueue);
}
```

ПО определяет использовать запрос или нет путем вызова метода **isApplicable()**.

Поиск с параметрами **classpath:templates/**/*.yaml** файлов типа YAML, которые содержат логику анализа и модификации в формате DSL (Domain-Specific Language). В случае протокола SOAP, если содержание элемента **Action** в заголовке сообщения совпадает с именем файла, то ПО использует этот файл для обработки сообщений. В случае протокола REST, ПО сравнивает путь к файлу с путем запроса и метод запроса с именем файла (например запрос **GET** к **/examples/simple** попытается использовать **classpath:templates/rest/examples/simple/GET.yaml**).

Для подробного описания языка DSL, см. документацию по программному обеспечению «Анонимайзер».

Приложение Д – Пример Функционирования ПО (DSL)

Данный раздел приводит пример функционирования ПО с использованием встроенного тестового файла в формате DSL.

Для реализации тестирования ПО предоставляет следующие файлы:

- ПО (http-proxy-1.0.0.jar),
- Файл override.yaml с настройками для тестирования ПО

```
anonymizer:
 routes:
  - name: 'Test'
    ingressPort: 2220
    resolver: 'Generic'
    services:
      - name: 'Hello World'
        baseUrl: 'http://localhost:9000'
        path: '/'
```

- Файл в формате DSL (встроенный в ПО для упрощения тестирования),

```
ResponseTemplate:
 content: custom GoodBye
```

- Java класс GoodBye (встроенный в ПО), реализующий акцию для файла DSL,

```
public class GoodBye implements Action {

   GoodBye(String... args) { }

   @Override
   public Object execute(String value, Variables variables, SharedContextReference scr) {
      return value.replace("Hello", "Good Bye");
   }
}
```

- Библиотека hello-service.jar с простой службой для тестирования (https://github.com/spring-guides/gs-actuator-service).

Для выполнения примера необходимо осуществить следующие шаги:

1. Использовать операционную систему Ubuntu 18.04 LTS,
2. Убедиться, что Open JDK 11 установлен и доступен (см. раздел Установка, выполнение и доступ в ПО в среде JRE),
3. Убедиться, что порты 2220, 8080 и 9000 операционной системы не используются другими приложениями,
4. Поместить все необходимые файлы (http-proxy-1.0.0.jar, hello-service.jar, override.yaml) в папку, в которой будет производиться тестирование,
5. Открыть три терминальных окна и в каждом из них перейти в папку, в которой будет производиться тестирование.
6. Запустить тестовую службу в первом окна:

```
java -jar hello-service.jar
```

7. Запустить ПО во втором терминальном окне:

```
java -jar http-proxy-1.0.0.jar --spring.config.additional-location=file:$(pwd)/override.yaml
```

8. С третьего окна, вызвать тестовую службу напрямую:

```
curl http://localhost:9000/hello-world?name=Proxy
```

```
{"id":2,"content":"Hello, Proxy!"}
```

9. В продолжение, с третьего окна, вызвать тестовую службу через ПО:

```
curl http://localhost:2220/hello-world?name=Proxy
```

```
{
  "id" : 1,
  "content" : "Good Bye, Proxy!"
}
```

Как видно из шага №9, ПО находит стратегию по обработке запроса и ответа, вызывает её. Стратегия модифицирует ответ на основании логики, указанной в файле DSL.