УДК 004

Создание модуля по регистрации запросов и ответов Rest API в SpringBoot

Еровлева Регина Викторовна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Еровлев Павел Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье рассмотрен процесс создания модуля для регистрации запросов Rest API в журнал событий. Произведена настройка для полноценной записи запросов на языке программирования Java. Итоговым результатом является настроенное приложение с подключенным журналированием.
Ключевые слова: Rest API, Java, Spring

Creation of a module for registration of requests and answers REST API in Springboot

Erovleva Regina Viktorovna
Sholom-Aleichem Priamursky State University
Student

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article will consider the process of creating a module for registering REST API requests to the event log. A configuration will be made for a full -fledged record of requests. The final result will be a configured application with connected journaling.
Keywords: Rest API, Java, Spring

Журналирование – один из немаловажных пунктов в создании программы. Оно помогает выявлять причины неисправностей и отслеживать ход событий, благодаря чему можно впоследствии устранить ошибку.

Цель данной статьи – настроить журналирование запросов и ответов Rest API.

А.А. Симаков описал разработку трассировки для JVM программ в использовании анализа ПО и обратного проектирования [1]. В своей работе В.П. Великов, К.С. Добрева рассмотрели проблемы автоматизированной генерации ПО [2]. В своей статье С.В. Мельников рассмотрел принципы для работы с отладочным интерфейсом Java [3]. А.А. Федоренко описала концепции часовых поясов, а также сравнила способы для работы с временем и датой на языке программирования Java [4]. М.К. Ермаков и С.П. Вартанов рассмотрели подход к поиску состояния гонки в многопоточных программах с использованием Java при помощи динамического анализа [5].

Чтобы реализовать ведение журнала, нужно перехватывать каждый запрос до вызова соответствующего контроллера, получать все необходимые данные и вносить их в журнал.

Для начала реализовываем «WebMvcConfigurer» и переопределяем метод «addInterceptors», добавив перехватчики в «InterceptorRegistry» (рис.1).

```
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.stereotype.Component;
5  import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
6  import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
7
8  @Component
9  public class CustomWebConfigurer implements WebMvcConfigurer {
10
11     @Autowired
12     private InterceptLog logInterceptor;
13
14     @Override
15     public void addInterceptors(InterceptorRegistry registry) {
16         registry.addInterceptor(logInterceptor);
17     }
18 }
```

Рисунок 1 – WebMvcConfigurer

HTTP-методы GET, DELETE и иногда PUT не будут иметь полезной нагрузки в запросах API. Для таких случаев необходимо реализовать метод «HandlerInterceptor» и переопределить его «preHandle» (рис.2).

```
3   import com.example.basiccrudAPIs.services.LoggingService;
4   import org.springframework.beans.factory.annotation.Autowired;
5   import org.springframework.http.HttpMethod;
6   import org.springframework.stereotype.Component;
7   import org.springframework.web.servlet.HandlerInterceptor;
8
9   import javax.servlet.http.HttpServletRequest;
10  import javax.servlet.http.HttpServletResponse;
11
12  @Component
13  public class InterceptLog implements HandlerInterceptor {
14
15      @Autowired
16      LoggingService loggingService;
17
18      @Override
19      public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
20          if(request.getMethod().equals(HttpMethod.GET.name())
21                  || request.getMethod().equals(HttpMethod.DELETE.name())
22                  || request.getMethod().equals(HttpMethod.PUT.name()))    {
23              loggingService.displayReq(request,null);
24          }
25          return true;
26      }
27  }
```

Рисунок 2 - HandlerInterceptor

Для запросов API с полезной нагрузкой необходимо расширить «RequestBodyAdviceAdapter», предоставляя реализацию для абстрактных методов и «afterBodyRead()» (рис.3).

```
import com.example.basiccrudAPIs.services.LoggingService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.MethodParameter;
import org.springframework.http.HttpInputMessage;
import org.springframework.http.converter.HttpMessageConverter;
import org.springframework.stereotype.Component;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.servlet.mvc.method.annotation.RequestBodyAdviceAdapter;

import javax.servlet.http.HttpServletRequest;
import java.lang.reflect.Type;

@ControllerAdvice
public class RequestBodyInterceptor extends RequestBodyAdviceAdapter {

    @Autowired
    LoggingService logService;

    @Autowired
    HttpServletRequest request;

    @Override
    public Object afterBodyRead(Object body, HttpInputMessage inputMessage, MethodParameter parameter,
        logService.displayReq(request,body);
        return super.afterBodyRead(body, inputMessage, parameter, targetType, converterType);
    }

    @Override
    public boolean supports(MethodParameter methodParameter, Type targetType, Class<? extends HttpMessa
        return true;
    }
}
```

Рисунок 3 - RequestBodyAdviceAdapter

«ResponseBodyAdvice» позволяет настраивать ответ после выполнения метода «@ResponseBody» или «ResponseEntity» контроллера, но до того, как тело будет записано с помощью файла «HttpMessageConverter». Таким образом, можно реализовать интерфейс, чтобы получить ответ каждого API (рис.4).

```
import com.example.basiccrudAPIs.services.LoggingService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.MethodParameter;
import org.springframework.http.MediaType;
import org.springframework.http.converter.HttpMessageConverter;
import org.springframework.http.server.ServerHttpRequest;
import org.springframework.http.server.ServerHttpResponse;
import org.springframework.http.server.ServletServerHttpRequest;
import org.springframework.http.server.ServletServerHttpResponse;
import org.springframework.stereotype.Component;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.servlet.mvc.method.annotation.ResponseBodyAdvice;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@ControllerAdvice
public class ResponseBodyInterceptor implements ResponseBodyAdvice<Object> {

    @Autowired
    LoggingService loggingService;

    @Override
    public boolean supports(MethodParameter returnType, Class<? extends HttpMessageConverter<?>> conver
        return true;
    }

    @Override
    public Object beforeBodyWrite(Object body, MethodParameter returnType,
                                  MediaType selectedContentType, Class<? extends HttpMessageConverter<
                                  ServerHttpRequest request, ServerHttpResponse response) {
        loggingService.displayResp(((ServletServerHttpRequest) request).getServletRequest(), ((Servlet
        return body;
    }
}
```

Рисунок 4 – ResponseBodyAdvice

Обе реализации «RequestBodyAdvice» и «ResponseBodyAdvice» снабжены аннотациями «@ControllerAdvice», поэтому они будут автоматически обнаруживаться «RequestMappingHandlerAdapter» или «ExceptionHandlerExceptionResolver».

Во всех вышеперечисленных классах будем получать запросы или ответы API. Чтобы регистрировать их в требуемом формате, необходима служба, принимающая запросы, ответы и тело в качестве входных данных(рис.5) и реализуем его (рис.6-7).

```
3  import javax.servlet.http.HttpServletRequest;
4  import javax.servlet.http.HttpServletResponse;
5
6
7  public interface LoggingService {
8
9      void displayReq(HttpServletRequest request, Object body);
10
11     void displayResp(HttpServletRequest request, HttpServletResponse response, Object body);
12 }
```

Рисунок 5 - LoggingService

```
3  import org.slf4j.Logger;
4  import org.slf4j.LoggerFactory;
5  import org.springframework.stereotype.Service;
6
7  import javax.servlet.http.HttpServletRequest;
8  import javax.servlet.http.HttpServletResponse;
9  import java.util.*;
10
11 @Service
12 public class LoggingServiceImpl implements LoggingService{
13
14     Logger logger = LoggerFactory.getLogger("LoggingServiceImpl");
15
16     @Override
17     public void displayReq(HttpServletRequest request, Object body) {
18         StringBuilder reqMessage = new StringBuilder();
19         Map<String,String> parameters = getParameters(request);
20
21         reqMessage.append("REQUEST ");
22         reqMessage.append("method = [").append(request.getMethod()).append("]");
23         reqMessage.append(" path = [").append(request.getRequestURI()).append("] ");
24
25         if(!parameters.isEmpty()) {
26             reqMessage.append(" parameters = [").append(parameters).append("] ");
27         }
28
29         if(!Objects.isNull(body)) {
30             reqMessage.append(" body = [").append(body).append("]");
31         }
32
33         logger.info("log Request: {}", reqMessage);
34     }
35
36     @Override
37     public void displayResp(HttpServletRequest request, HttpServletResponse response, Object body)
38         StringBuilder respMessage = new StringBuilder();
39         Map<String,String> headers = getHeaders(response);
40         respMessage.append("RESPONSE ");
41         respMessage.append(" method = [").append(request.getMethod()).append("]");
42         if(!headers.isEmpty()) {
43             respMessage.append(" ResponseHeaders = [").append(headers).append("]");
44         }
45         respMessage.append(" responseBody = [").append(body).append("]");
46
47         logger.info("logResponse: {}",respMessage);
48     }
```

Рисунок 6 – LoggingServiceImpl

```
50      private Map<String,String> getHeaders(HttpServletResponse response) {
51          Map<String,String> headers = new HashMap<>();
52          Collection<String> headerMap = response.getHeaderNames();
53          for(String str : headerMap) {
54              headers.put(str,response.getHeader(str));
55          }
56          return headers;
57      }
58
59      private Map<String,String> getParameters(HttpServletRequest request) {
60          Map<String,String> parameters = new HashMap<>();
61          Enumeration<String> params = request.getParameterNames();
62          while(params.hasMoreElements()) {
63              String paramName = params.nextElement();
64              String paramValue = request.getParameter(paramName);
65              parameters.put(paramName,paramValue);
66          }
67          return parameters;
68      }
69
70
71  }
```

Рисунок 7 – LoggingServiceImpl

Теперь при выполнении программой любых запросов и получения ответов все данные будут записываться в журнал (рис.8).

```
2022-04-16 09:22:31.591  INFO 5348 --- [nio-8080-exec-1]
LoggingServiceImpl                       : log Request: REQUEST method
= [GET] path = [/students]
2022-04-16 09:22:31.687  INFO 5348 --- [nio-8080-exec-1]
LoggingServiceImpl                       : logResponse: RESPONSE
method = [GET] responseBody = [[]]
2022-04-16 09:22:45.120  INFO 5348 --- [nio-8080-exec-3]
LoggingServiceImpl                       : log Request: REQUEST method
= [POST] path = [/students]  body = [Student{Id=null,
studentName='Vinay', studentAge='26', address='Bangalore'}]
Student:Student{Id=null, studentName='Vinay', studentAge='26',
address='Bangalore'}
2022-04-16 09:22:45.168  INFO 5348 --- [nio-8080-exec-3]
LoggingServiceImpl                       : logResponse: RESPONSE
method = [POST] responseBody = [Student{Id=1, studentName='Vinay',
studentAge='26', address='Bangalore'}]
2022-04-16 09:22:59.012  INFO 5348 --- [nio-8080-exec-4]
LoggingServiceImpl                       : log Request: REQUEST method
= [GET] path = [/students]
2022-04-16 09:22:59.035  INFO 5348 --- [nio-8080-exec-4]
LoggingServiceImpl                       : logResponse: RESPONSE
method = [GET] responseBody = [[Student{Id=1, studentName='Vinay',
studentAge='26', address='Bangalore'}]]
2022-04-16 09:23:08.618  INFO 5348 --- [nio-8080-exec-5]
LoggingServiceImpl                       : log Request: REQUEST method
= [PUT] path = [/students]  parameters = [{Address=Mumbai, Id=1}]
2022-04-16 09:23:08.659  INFO 5348 --- [nio-8080-exec-5]
LoggingServiceImpl                       : logResponse: RESPONSE
method = [PUT] responseBody = [Student{Id=1, studentName='Vinay',
studentAge='26', address='Mumbai'}]
2022-04-16 09:23:20.051  INFO 5348 --- [nio-8080-exec-6]
LoggingServiceImpl                       : log Request: REQUEST method
= [DELETE] path = [/students]  parameters = [{Id=1}]
2022-04-16 09:23:20.074  INFO 5348 --- [nio-8080-exec-6]
LoggingServiceImpl                       : logResponse: RESPONSE
method = [DELETE] responseBody = [null]
```

Рисунок 8 – Журналирование

В данной статье был рассмотрен процесс реализации разработки модуля, который регистрирует ответы и запросы Rest API в журнал.

Библиографический список

1. Симаков А.А. Java tracer. Программное средство для трассировки java программ // Заметки по информатике и математике. 2019. №3. С. 51-57.
2. Великов В.П., Добрева К.С. Генератор из диаграмм классов java в исходный код java // Информационные системы и технологии: управление и безопасность. 2014. №3. С. 14-23.
3. Мельников С.В. Обзор и применение отладочного интерфейса java (jdi) для обратимой модификации программных продуктов // Современные проблемы науки и образования. 2018. №8. С. 8-19.
4. Федоренко А.А. Средства работы с датой и временем в языке

программирования java // Актуальные научные исследования в современном мире. 2021. №7-1(51). С. 65-69.
5. Ермаков М.К. и Вартанов С.П. Подход к проведению динамического анализа java-программ методом модификации виртуальной машины java // Труды института системного программирования ран. 2015. №2. С. 39-52.