Spring Boot Reactive File Service

Das folgende Programmscript zeigt den REST Controller ReactiveFileService mit dem wir anhand des Path URL Parameters eine lokale Datei laden und reactive mit Flux<DataBuffer> an den HTTP Client transferieren.

package ch.std.fileservice.rest;
import java.nio.channels.FileChannel;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.List;
import javax.activation.MimetypesFileTypeMap;
import org.springframework.core.io.buffer.DataBuffer;
import org.springframework.core.io.buffer.DataBufferFactory;
import org.springframework.core.io.buffer.DataBufferUtils;
import org.springframework.http.server.reactive.ServerHttpResponse;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.server.ServerWebExchange;
import reactor.core.publisher.Flux;
@RestController
@RequestMapping(value = "/rest/file")
public class ReactiveFileService {
  public static final int defaultBufferSize = 1 << 12;
  public ReactiveFileService() {
    super();
  }
  @GetMapping
  public Flux<DataBuffer> get(ServerWebExchange webExchange) throws Exception {
    List pathList = webExchange.getRequest().getQueryParams().get("path");
    ServerHttpResponse serverHttpResponse = webExchange.getResponse();
    DataBufferFactory dataBufferFactory = webExchange.getResponse().bufferFactory();
    if (pathList == null) {
      serverHttpResponse.getHeaders().add("Content-Type", "text/html; charset=UTF-8");
      DataBuffer replyDataBuffer = dataBufferFactory.allocateBuffer(defaultBufferSize)
        .write("path is null".getBytes(StandardCharsets.UTF_8));
      return Flux.just(replyDataBuffer);
    }
    String path = pathList.get(0);
    String mimeType = Files.probeContentType(Paths.get(path));
    if (mimeType == null) {
      MimetypesFileTypeMap mimeTypesMap = new MimetypesFileTypeMap();
      mimeType = mimeTypesMap.getContentType(path);
    }
    serverHttpResponse.getHeaders().add("Content-Type", mimeType);
    Flux result = DataBufferUtils
      .readByteChannel(() -> FileChannel.open(Paths.get(path), StandardOpenOption.READ), dataBufferFactory, defaultBufferSize)
      .onErrorResume(ex -> {
        serverHttpResponse.getHeaders().add("Content-Type", "text/html; charset=UTF-8");
        DataBuffer replyDataBuffer = dataBufferFactory.allocateBuffer(defaultBufferSize).write(
          ("path " + path + " not found, ex = " + ex.getMessage()).getBytes(StandardCharsets.UTF_8));
        return Flux.just(replyDataBuffer);
      });
    return result;
  }
}
Das folgende Listing zeigt die Spring Boot Application:package ch.std.fileservice;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class ReactiveFileServiceApplication {
  public static void main(String[] args) {
    SpringApplication.run(ReactiveFileServiceApplication.class, args);
  }
}

Nach dem Start der Spring Boot Applikation ist der Reactive File Service aktiv und kann je nach Konfiguration wie folgt adressiert werden:http://localhost:8080/rest/file?path=/mypath/myfile.txtSofern das File existiert wird es zum Client reactive gestreamt.

Das folgende Listing zeigt den dazu passenden Spring Boot Reactive File Service Client entwickelt als Command Line Applikation:package ch.std.fileservice.client;
import java.io.FileOutputStream;
import java.net.URL;
import org.springframework.boot.ApplicationArguments;
import

org.springframework.boot.ApplicationRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.WebApplicationType;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.core.io.buffer.DataBuffer;
import org.springframework.core.io.buffer.DataBufferUtils;
import org.springframework.web.reactive.function.client.WebClient;
import reactor.core.publisher.Flux;
@SpringBootApplication
public class ReactiveFileServiceClient implements ApplicationRunner {
  public static void main(String[] args) throws Exception {
  SpringApplication app = new SpringApplication(ReactiveFileServiceClient.class);
 app.setWebApplicationType(WebApplicationType.NONE);
  app.run(args);
  }
 @Override
  public void run(ApplicationArguments args) throws Exception {
  URL url = null;
  try {
  url = new URL(args.getOptionValues("url").get(0));
  } catch (Exception e) {
  System.err.println("missing --url option argument");
  this.help();
  return;
  }
  String out = null;
  try {
  out = args.getOptionValues("out").get(0);
  } catch (Exception e) {
  }
  String surl = url.getProtocol() + "://" + url.getHost() + ":" + url.getPort();
  String path = url.getFile();
 Flux<DataBuffer> data = WebClient.create(surl).get().uri(path).retrieve().bodyToFlux(DataBuffer.class);
  if (out != null) {
  try (FileOutputStream fos = new FileOutputStream(out)) {
 DataBufferUtils.write(data, fos).map(DataBufferUtils::release).blockLast();
  }
 System.out.println("result written to file " + out);
  }
  }
  private void help() {
  System.out.println("usage java -jar reactivefileclient-0.0.1-SNAPSHOT.jar --url= --out=");
  System.out.println("example:");
 System.out.println("java -jar reactivefileclient-0.0.1-SNAPSHOT.jar --url=http://localhost:8080/rest/file?path=in/bigimage.jpg --out=out/bigimage.jpg");
 }
}
Das File bigimage.jpg kann ersetzt werden durch eine real existierende Datei analog kann der Output Pfad angepasst werden.

Client und Service sind am besten in 2 separaten Spring Boot Projekten zu programmieren z.B. mit der Eclipse IDE.

Feedback

War dieser Blog für Sie wertvoll. Wir danken für jede Anregung und Feedback

Kontakt

Simtech AG
Finkenweg 23
3110 Münsingen
Schweiz

Impressum

Copyright

https://www.simtech-ag.ch/serverwebexchange