

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;&#xA;import java.nio.channels.FileChannel;&#xA;import
java.nio.charset.StandardCharsets;&#xA;import java.nio.file.Files;&#xA;import
java.nio.file.Paths;&#xA;import java.nio.file.StandardOpenOption;&#xA;import
java.util.List;&#xA;import javax.activation.MimetypesFileTypeMap;&#xA;import
org.springframework.core.io.buffer.DataBuffer;&#xA;import
org.springframework.core.io.buffer.DataBufferFactory;&#xA;import
org.springframework.core.io.buffer.DataBufferUtils;&#xA;import
org.springframework.http.server.reactive.ServerHttpResponse;&#xA;import
org.springframework.web.bind.annotation.GetMapping;&#xA;import
org.springframework.web.bind.annotation.RequestMapping;&#xA;import
org.springframework.web.bind.annotation.RestController;&#xA;import
org.springframework.web.server.ServerWebExchange;&#xA;import
reactor.core.publisher.Flux;&#xA;@RestController&#xA;@RequestMapping(value =
&#34;/rest/file&#34;)&#xA;public class ReactiveFileService {&#xA;    public static final int
defaultBufferSize = 1 &lt;&lt; 12;&#xA;    public ReactiveFileService() {&#xA;        super();&#xA;
    }&#xA;    @GetMapping&#xA;    public Flux&lt;DataBuffer&gt; get(ServerWebExchange
webExchange) throws Exception {&#xA;        List pathList =
webExchange.getRequest().getQueryParams().get(&#34;path&#34;);&#xA;        ServerHttpResponse
serverHttpResponse = webExchange.getResponse();&#xA;        DataBufferFactory dataBufferFactory
= webExchange.getResponse().bufferFactory();&#xA;        if (pathList == null) {&#xA;
serverHttpResponse.getHeaders().add(&#34;Content-Type&#34;, &#34;text/html;
charset=UTF-8&#34;);&#xA;            DataBuffer replyDataBuffer =
dataBufferFactory.allocateBuffer(defaultBufferSize)&#xA;                .write(&#34;path is
null&#34;.getBytes(StandardCharsets.UTF_8));&#xA;            return Flux.just(replyDataBuffer);&#xA;
        }&#xA;        String path = pathList.get(0);&#xA;        String mimeType =
Files.probeContentType(Paths.get(path));&#xA;        if (mimeType == null) {&#xA;
MimetypesFileTypeMap mimeTypesMap = new MimetypesFileTypeMap();&#xA;            mimeType =
mimeTypesMap.getContentType(path);&#xA;        }&#xA;        serverHttpResponse.getHeaders().add(&#34;Content-Type&#34;, mimeType);&#xA;        Flux result
= DataBufferUtils&#xA;            .readByteChannel(() -&gt; FileChannel.open(Paths.get(path),
StandardOpenOption.READ), dataBufferFactory, defaultBufferSize)&#xA;            .onErrorResume(ex
-&gt; {&#xA;                serverHttpResponse.getHeaders().add(&#34;Content-Type&#34;,
&#34;text/html; charset=UTF-8&#34;);&#xA;                DataBuffer replyDataBuffer =
dataBufferFactory.allocateBuffer(defaultBufferSize).write(&#xA;                    (&#34;path &#34; + path +
&#34; not found, ex = &#34; + ex.getMessage()).getBytes(StandardCharsets.UTF_8));&#xA;
                return Flux.just(replyDataBuffer);&#xA;            });&#xA;            return result;&#xA;    }&#xA;}&#xA;}&#xA;Das
folgende Listing zeigt die Spring Boot Application:package ch.std.fileservice;&#xA;import
org.springframework.boot.SpringApplication;&#xA;import
org.springframework.boot.autoconfigure.SpringBootApplication;&#xA;@SpringBootApplication&#xA;
public class ReactiveFileServiceApplication {&#xA;    public static void main(String[] args) {&#xA;
SpringApplication.run(ReactiveFileServiceApplication.class, args);&#xA;    }&#xA;}&#xA;}
```

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.txt> Sofern 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;#xA;import
org.springframework.boot.SpringApplication;#xA;import
org.springframework.boot.WebApplicationType;#xA;import
org.springframework.boot.autoconfigure.SpringBootApplication;#xA;import
org.springframework.core.io.buffer.DataBuffer;#xA;import
org.springframework.core.io.buffer.DataBufferUtils;#xA;import
org.springframework.web.reactive.function.client.WebClient;#xA;import
reactor.core.publisher.Flux;#xA;@SpringBootApplication#xA;public class
ReactiveFileServiceClient implements ApplicationRunner {#xA;    public static void main(String[]
args) throws Exception {#xA;        SpringApplication app = new
SpringApplication(ReactiveFileServiceClient.class);#xA;
app.setWebApplicationType(WebApplicationType.NONE);#xA;        app.run(args);#xA;    }#xA;
@Override#xA;    public void run(ApplicationArguments args) throws Exception {#xA;        URL url
= null;#xA;        try {#xA;            url = new URL(args.getOptionValues("#4:url#34;").get(0));#xA;
        } catch (Exception e) {#xA;            System.err.println("#34;missing --url option
argument#34;);#xA;            this.help();#xA;            return;#xA;        }#xA;        String out =
null;#xA;        try {#xA;            out = args.getOptionValues("#34:out#34;").get(0);#xA;        } catch
(Exception e) {#xA;            }#xA;        String surl = url.getProtocol() + "#34://#34; + url.getHost() +
"#34:;#34; + url.getPort();#xA;        String path = url.getFile();#xA;
Flux&lt;DataBuffer&gt; data =
WebClient.create(surl).get().uri(path).retrieve().bodyToFlux(DataBuffer.class);#xA;        if (out !=
null) {#xA;            try (FileOutputStream fos = new FileOutputStream(out)) {#xA;
DataBufferUtils.write(data, fos).map(DataBufferUtils::release).blockLast();#xA;            }#xA;
System.out.println("#34;result written to file #34; + out);#xA;        }#xA;    }#xA;    private void
help() {#xA;        System.out.println("#34;usage java -jar reactivefileclient-0.0.1-SNAPSHOT.jar
--url= --out=#34;);#xA;        System.out.println("#34;example:#34;);#xA;
System.out.println("#34;java -jar reactivefileclient-0.0.1-SNAPSHOT.jar
--url=http://localhost:8080/rest/file?path=in/bigimage.jpg --out=out/bigimage.jpg#34;);#xA;
}#xA;}#xA;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

Das Copyright für sämtliche Inhalte dieser Website liegt bei Simtech AG, Schweiz.
Beachten Sie auch unsere Hinweise zum Urheberrecht, Datenschutz und Haftungsausschluss.
Jeder Hinweis auf Fehler nehmen wir gerne entgegen.

Copyright

2024 Simtech AG, All rights reserved, Powered by stack.ch written in Golang by Daniel Schmutz

<https://www.simtech-ag.ch/Projekten>