# A qui va dirigit

Aquest how-to va dirigit als perfils tècnics (desenvolupadors i arquitectes) que desenvolupin aplicacions que realitzin connexions a serveis REST amb pujada o descàrrega de fitxers.

# Introducció

En aquest HowTo s’explica com realitzar crides per pujar i descarregar fitxers de un servei REST.

Es realitza tant la pujada com la baixada de les dades del fitxer en blocs, sense carregar tot el contingut en memòria.

# Objecte a pujar

Primer de tot s’ha de crear l’objecte a enviar amb l’esquema que espera el servei REST. En el nostre exemple el servei espera rebre una petició JSON com la següent:

El servei Rest d’exemple espera la següent petició:

*{*

*"Document":{*

*"nom":"prova\_complerta.txt",*

*"data":<<prova\_complerta.txt en base64>>*

*}*

*}*

Generem la següent classe de Java:

*public class Document {*

*private String data;*

*private String nom;*

*/\*Getters & Setters\*/*

*public String getData() {*

*retorn data;*

*}*

*@JsonProperty("data")*

*public void setData(String data) {*

*this.data = data;*

*}*

*public String getNom() {*

*return nom;*

*}*

*@JsonProperty("nom")*

*public void setNom(String nom) {*

*this.nom = nom;*

*}*

*}*

# Pujar Fitxer

Per a realitzar la crida al servei REST utilitzem RestTemplate que es pot trobar a la llibreria spring-web. (org.springframework.web.client.RestTemplate)

Primer s’ha de crear el bean restTemplate:

*<bean id="restTemplate" class="org.springframework.web.client.RestTemplate">*

*<property name="requestFactory" ref="clientHttpRequestFactory" />*

*<property name="messageConverters">*

*<list>*

*<bean id="jsonMessageConverter"*

*class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter">*

*<property name="supportedMediaTypes">*

*<list>*

*<bean id="jsonMediaTypeTextPlain" class="org.springframework.http.MediaType">*

*<constructor-arg value="text" />*

*<constructor-arg value="plain" />*

*</bean>*

*<bean id="jsonMediaTypeApplicationJson" class="org.springframework.http.MediaType">*

*<constructor-arg value="application" />*

*<constructor-arg value="json" />*

*</bean>*

*</list>*

*</property>*

*</bean>*

*</list>*

*</property>*

*</bean>*

*<bean id="clientHttpRequestFactory" class="org.springframework.http.client.BufferingClientHttpRequestFactory" >*

*<constructor-arg>*

*<bean class="org.springframework.http.client.SimpleClientHttpRequestFactory">*

***<property name="bufferRequestBody" value="false" />***

*</bean>*

*</constructor-arg>*

*</bean>*

És necessari indicar la propietat bufferRequestBody a false per a poder enviar la petició amb streaming.

A la classe on realitzarem la petició REST s’ha d’injectar el bean creat:

*@Autowired*

***private*** *RestTemplate restTemplate;*

Es crea l’objecte a enviar i el InputStream del qual es llegirà el contingut del fitxer:

*Document document =* ***new*** *Document();*

*Document.setNom("prova\_complerta");*

*InputStream data =* ***new*** *FileInputStream(****new*** *File("/data/prova\_complerta.txt"));*

I la crida a la petició REST:

*restTemplate.execute(url, HttpMethod.POST, requestCallback, responseExtractor);*

A requestCallback s’ha de fer els següents passos:

* Obtenir el canal d’escriptura de la request.
* Escriure els bytes corresponents a la crida de upload a realitzar fins arribar al punt on s’ha d’enviar el contingut del fitxer
* Escriure el contingut del fitxer en base64 de tal forma que aquest contingut no arribi a estar en memòria al servidor en cap moment mentre s’escriu
* Escriure els bytes corresponents a la resta de la crida

***final*** *RequestCallback requestCallback =* ***new*** *RequestCallback() {*

*@Override*

***public******void*** *doWithRequest(****final*** *ClientHttpRequest request)* ***throws*** *IOException {*

*request.getHeaders().add("Content-type", "application/octet-stream");*

*//*Obtenir el canal d’escriptura de la request de la connexió:

*OutputStream output = request.getBody();*

*//objecte que utilitzarem per passer a JSON el nostre objecte*

*ObjectMapper mapper =* ***new*** *ObjectMapper();*

*String jsonInString = mapper.writeValueAsString(document);*

*//Tallem el String per on hem d’enviar el fitxer*

***int*** *split = jsonInString.indexOf("data");*

*//Escrivim la primera part de la nostra petició*

*String begin = jsonInString.substring(0, split + 6);*

*output.write(begin.getBytes("UTF-8"));*

*// Codifiquem I escrivim el nostre fitxer*

*BASE64Encoder encoder =* ***new*** *BASE64Encoder();*

*encoder.encode(data, output);*

*// Acabem d’escriure la nostra petició*

*String end = jsonInString.substring(split + 9, jsonInString.length());*

*output.write(end.getBytes("UTF-8"));*

}

};

# Objecte a rebre

Primer de tot s’ha de crear l’objecte a rebre amb l’esquema que utilitza el servei REST. En el nostre exemple el servei REST retorna una resposta com la següent:

*{*

*"Resposta":{*

*"codi":"OK",*

*"descripcio":"Correcte",*

*"id":"AA12",*

*"nom":" prova\_complerta.txt",*

*"mimeType":"text/plain",*

*"dataSize":43,*

*"dataEncoding":"Base64",*

*"dataEncodedSize":60,*

*"data":<<prova\_complerta.txt en base64>>*

*}*

*}*

Generem la següent classe Java:

**public** **class** Resposta {

**private** String codi;

**private** String descripcio;

**private** String id;

**private** String nom;

**private** String mimeType;

**private** Integer dataSize;

**private** String dataEncoding;

**private** Integer dataEncodedSize;

**private** ByteArrayOutputStream data;

/\*Getters & Setters\*/

**public** String getCodi() {

**return** codi;

}

@JsonProperty("codi")

**public** **void** setCodi(String codi) {

**this**.codi = codi;

}

**public** String getDescripcio() {

**return** descripcio;

}

@JsonProperty("descripcio")

**public** **void** setDescripcio(String descripcio) {

**this**.descripcio = descripcio;

}

**public** String getId() {

**return** id;

}

@JsonProperty("id")

**public** **void** setId(String id) {

**this**.id = id;

}

**public** String getNom() {

**return** nom;

}

@JsonProperty("nom")

**public** **void** setNom(String nom) {

**this**.nom = nom;

}

**public** String getMimeType() {

**return** mimeType;

}

@JsonProperty("mimeType")

**public** **void** setMimeType(String mimeType) {

**this**.mimeType = mimeType;

}

**public** Integer getDataSize() {

**return** dataSize;

}

@JsonProperty("dataSize")

**public** **void** setDataSize(Integer dataSize) {

**this**.dataSize = dataSize;

}

**public** String getDataEncoding() {

**return** dataEncoding;

}

@JsonProperty("dataEncoding")

**public** **void** setDataEncoding(String dataEncoding) {

**this**.dataEncoding = dataEncoding;

}

**public** Integer getDataEncodedSize() {

**return** dataEncodedSize;

}

@JsonProperty("dataEncodedSize")

**public** **void** setDataEncodedSize(Integer dataEncodedSize) {

**this**.dataEncodedSize = dataEncodedSize;

}

**public** ByteArrayOutputStream getData() {

**return** data;

}

@JsonProperty("data")

**public** **void** setData(ByteArrayOutputStream data) {

**this**.data = data;

}

/\* Fi Getters & Setters\*/

}

# Descarregar Fitxer

Al objecte responseExtractor de la crida restTemplate.execute és on es tracta la resposta rebuda.

ResponseExtractor<ResponseEntity<Resposta>> responseExtractor = **new** ResponseExtractor<ResponseEntity<Resposta>>() {

@Override

**public** ResponseEntity<Resposta> extractData(ClientHttpResponse response) **throws** IOException {

Resposta resposta = **null**;

**if** (response.getStatusCode().is2xxSuccessful()){

//Obtenim la resposta

InputStream is= response.getBody();

**final** ByteArrayOutputStream os = **new** ByteArrayOutputStream();

StreamingJsonParser parser = **new** StreamingJsonParser(os);

**try**{

parser.parse(is);

String jsonResponse=parser.getResponseBuffer().toString();

jsonResponse=jsonResponse.replace(",\"\"data\":\"", "");

ObjectMapper mapper = **new** ObjectMapper();

resposta= mapper.readValue(jsonResponse, Resposta.**class**);

resposta.setData(os);

}**catch** (Exception e){

}

}

**return** **new** ResponseEntity<Resposta>(resposta,response.getStatusCode());

}

};

Per tractar la resposta s’han de crear dos classes. “StreamingJsonParser” que s’utilitza per a llegir la resposta obtinguda, obtenir el valor de cada camp i setejar-lo al nostre objecte Resposta. Quan ha de llegir el camp data, que té els bytes del fitxer en Base64 ho fa amb streaming sense deixar el seu contingut en memòria. I per altra banda la classe “InnerInputStream” per a realitzar aquesta lectura.

Per últim mostrem el codi d’aquestes dues classes:

StreamingJsonParser.java

**public** **class** StreamingJsonParser {

**private** **long** expectedBase64Size;

**private** OutputStream os;

**private** StringBuilder responseBuffer;

**public** StreamingJsonParser(OutputStream os) {

**this**.os = os;

}

**public** **long** getExpectedBase64Size() {

**return** expectedBase64Size;

}

**public** **void** onDataEncodedSize(String previousTextContent) **throws** Exception {

expectedBase64Size = parseLong(previousTextContent);

}

**public** **void** onData(InputStream inputStream) **throws** Exception {

readBase64Content(inputStream, expectedBase64Size);

}

**public** **void** parse(InputStream inputStream) **throws** Exception {

responseBuffer= **new** StringBuilder();

StringBuilder buffer = **new** StringBuilder();

**int** rb = inputStream.read();

**while** (rb != -1) {

responseBuffer.append((**char**) rb);

**if** (rb == '"') {

**if** (!readMoreRequestContent(buffer, inputStream)) {

rb = '"';

**continue**;

}

} **else**

buffer.append((**char**) rb);

rb = inputStream.read();

}

}

**private** String parseString(InputStream inputStream) **throws** Exception {

StringBuilder sb = **new** StringBuilder();

sb.append('"');

**int** rb = -1;

**boolean** isScaping = **false**;

**while** ((rb = inputStream.read()) != -1) {

responseBuffer.append((**char**) rb);

**if** (rb == '"' && !isScaping)

**break**;

**else** **if** (rb == '\\')

isScaping = !isScaping;

**else** {

sb.append((**char**) rb);

isScaping = **false**;

}

}

sb.append('"');

**return** sb.toString();

}

**private** **long** parseLong(String text) **throws** Exception {

**int** indBegin = text.indexOf(':');

**if** (indBegin < 0)

**throw** **new** Exception("expected long value not found");

**int** indEnd = text.indexOf(',');

**if** (indEnd < 0)

indEnd = text.indexOf('}');

**if** (indEnd < 0)

indEnd = text.indexOf(']');

**if** (indEnd < 0)

**throw** **new** Exception("expected long value not found");

**return** **new** Long(text.substring(indBegin + 1, indEnd));

}

**private** **boolean** readMoreRequestContent(StringBuilder buffer, InputStream inputStream) **throws** Exception{

Method onPropertyCallback=**null**;

**boolean** incomingInputStreamBeingRead=**true**;

String previousTextContent=buffer.toString();

buffer.delete(0,buffer.length());

// reset buffer

**if**(previousTextContent.startsWith("\"")){

String onPropertyCallbackName="on"+previousTextContent.substring(1, previousTextContent.indexOf('"',1));

**try**{

onPropertyCallback=**this**.getClass().getMethod( onPropertyCallbackName,InputStream.**class**);

}**catch**(NoSuchMethodException nsex){

**try**{

onPropertyCallback=**this**.getClass().getMethod(onPropertyCallbackName,String.**class**);

incomingInputStreamBeingRead=**false**;

}**catch** (NoSuchMethodException nsex2){}

}

}

**if** (onPropertyCallback!=**null**){

**try**{

onPropertyCallback.invoke(**this**, incomingInputStreamBeingRead?inputStream:previousTextContent);

}**catch** (Throwable t){

**throw** **new** Exception("Error invoking "+onPropertyCallback.getName()+": "+ (t.getCause()!=**null**?t.getCause().getMessage():t.getMessage()));

}

}**else** buffer.append(parseString(inputStream));

**return** incomingInputStreamBeingRead;

}

**private** **void** readBase64Content(InputStream parentInputStream, **long** expectedInputSize) **throws** Exception{

InnerInputStream innerInputStream= **new** InnerInputStream (parentInputStream,expectedInputSize);

BASE64Decoder decoder = **new** BASE64Decoder();

decoder.decodeBuffer(innerInputStream, os);

**if** ('"'!=(**char**)parentInputStream.read())

**throw** **new** Exception("unexpected huge string ending character");

}

**public** StringBuilder getResponseBuffer() {

**return** responseBuffer;

}

}

InnerInputStream.java

**public** **class** InnerInputStream **extends** InputStream {

**private** InputStream parentInputStream;

**private** **long** numBytesLeft;

**public** InnerInputStream(InputStream parentInputStream, **long** maxBytesToRead) **throws** Exception {

**this**.parentInputStream = parentInputStream;

**this**.numBytesLeft = maxBytesToRead;

}

@Override

**public** **int** read() **throws** IOException {

**if** (numBytesLeft > 0) {

numBytesLeft--;

**int** b = parentInputStream.read();

**if** (b == -1)

numBytesLeft = 0;

// end of stream reached before expected max size.

**return** b;

}

**return** -1;

}

@Override

**public** **int** available() **throws** IOException {

**int** avail = parentInputStream.available();

**if** (numBytesLeft < avail)

**return** (**int**) numBytesLeft;

// avail to expected max size.

**return** avail;

}

@Override

**public** **int** read(**byte**[] b) **throws** IOException {

**return** read(b, 0, b.length);

}

@Override

**public** **int** read(**byte**[] b, **int** off, **int** len) **throws** IOException {

**int** rb = -1;

**if** (numBytesLeft > 0) {

**int** bmax = ((b.length - off) < len) ? b.length - off : len;

**if** (bmax <= numBytesLeft)

rb = parentInputStream.read(b, off, bmax);

**else**

rb = parentInputStream.read(b, off, (**int**) numBytesLeft);

**if** (rb == -1)

numBytesLeft = 0;

// end of stream reached before expected max size.

**else**

numBytesLeft -= rb;

}

**return** rb;

}

}