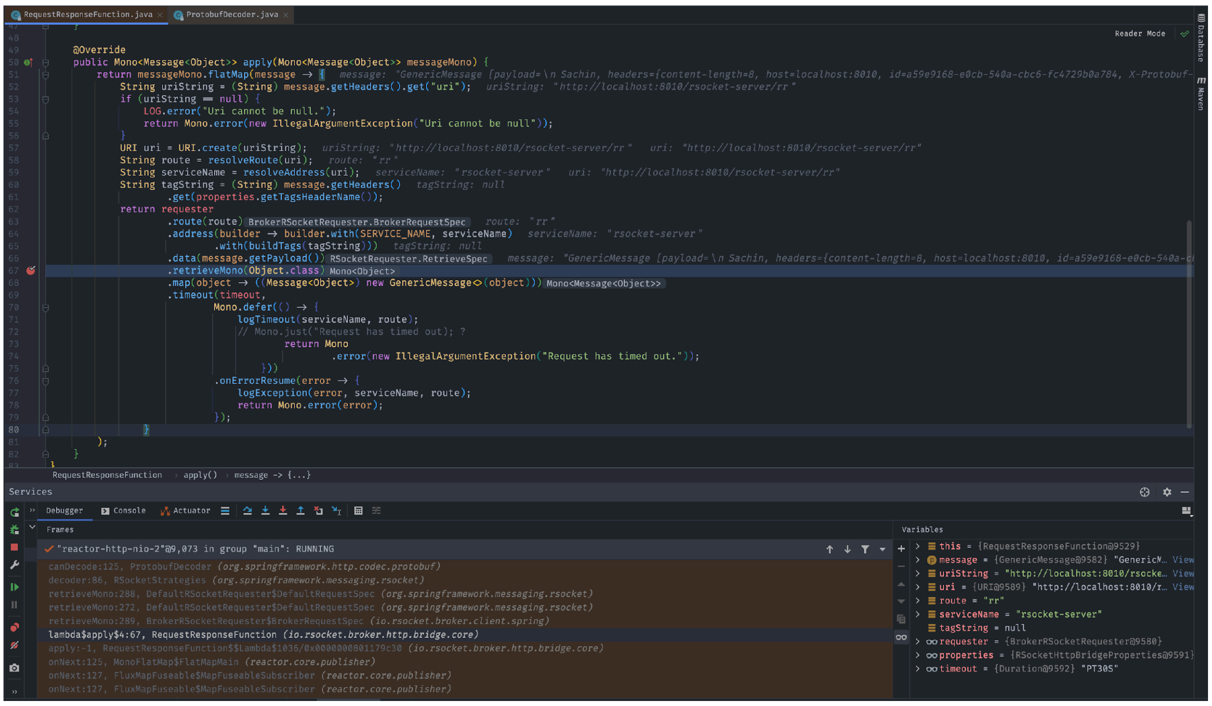
===================================
Problem area stackTrace
===================================

```
canDecode:125, ProtobufDecoder (org.springframework.http.codec.protobuf)
decoder:86, RSocketStrategies (org.springframework.messaging.rsocket)
retrieveMono:288, DefaultRSocketRequester$DefaultRequestSpec
(org.springframework.messaging.rsocket)
retrieveMono:272, DefaultRSocketRequester$DefaultRequestSpec
(org.springframework.messaging.rsocket)
retrieveMono:289, BrokerRSocketRequester$BrokerRequestSpec
(io.rsocket.broker.client.spring)
lambda$apply$4:67, RequestResponseFunction (io.rsocket.broker.http.bridge.core)
apply:-1, RequestResponseFunction$$Lambda$1036/0x0000000801179c30
(io.rsocket.broker.http.bridge.core)
```

On line 67 Object.class being passed which is causing Message.class.isAssignableFrom(elementType.toClass()) evaluation to false in org.springframework.http.codec.protobuf.ProtobufDecoder#canDecode() (Please, see below images)



```java
48
49      @Override
50      public Mono<Message<Object>> apply(Mono<Message<Object>> messageMono) {
51          return messageMono.flatMap(message -> {   message: "GenericMessage [payload=\n Sachin, headers={content-length=8, host=localhost:8010, id=a59e9168-e0cb-540a-cbc6-fc4729b0a784, X-Protobuf-
52              String uriString = (String) message.getHeaders().get("uri");   uriString: "http://localhost:8010/rsocket-server/rr"
53              if (uriString == null) {
54                  LOG.error("Uri cannot be null.");
55                  return Mono.error(new IllegalArgumentException("Uri cannot be null"));
56              }
57              URI uri = URI.create(uriString);   uriString: "http://localhost:8010/rsocket-server/rr"   uri: "http://localhost:8010/rsocket-server/rr"
58              String route = resolveRoute(uri);   route: "rr"
59              String serviceName = resolveAddress(uri);   serviceName: "rsocket-server"   uri: "http://localhost:8010/rsocket-server/rr"
60              String tagString = (String) message.getHeaders()   tagString: null
61                      .get(properties.getTagsHeaderName());
62              return requester
63                      .route(route) BrokerRSocketRequester.BrokerRequestSpec   route: "rr"
64                      .address(builder -> builder.with(SERVICE_NAME, serviceName)   serviceName: "rsocket-server"
65                              .with(buildTags(tagString)))   tagString: null
66                      .data(message.getPayload()) RSocketRequester.RetrieveSpec   message: "GenericMessage [payload=\n Sachin, headers={content-length=8, host=localhost:8010, id=a59e9168-e0cb-540a-c
67                      .retrieveMono(Object.class) Mono<Object>
68                      .map(object -> ((Message<Object>) new GenericMessage<>(object))) Mono<Message<Object>>
69                      .timeout(timeout,
70                              Mono.defer(() -> {
71                                  logTimeout(serviceName, route);
72                                  // Mono.just("Request has timed out"); ?
73                                      return Mono
74                                          .error(new IllegalArgumentException("Request has timed out."));
75                              }))
76                              .onErrorResume(error -> {
77                                  logException(error, serviceName, route);
78                                  return Mono.error(error);
79                              });
80          }
81      );
82  }
```

RequestResponseFunction › apply() › message -> {...}

Services

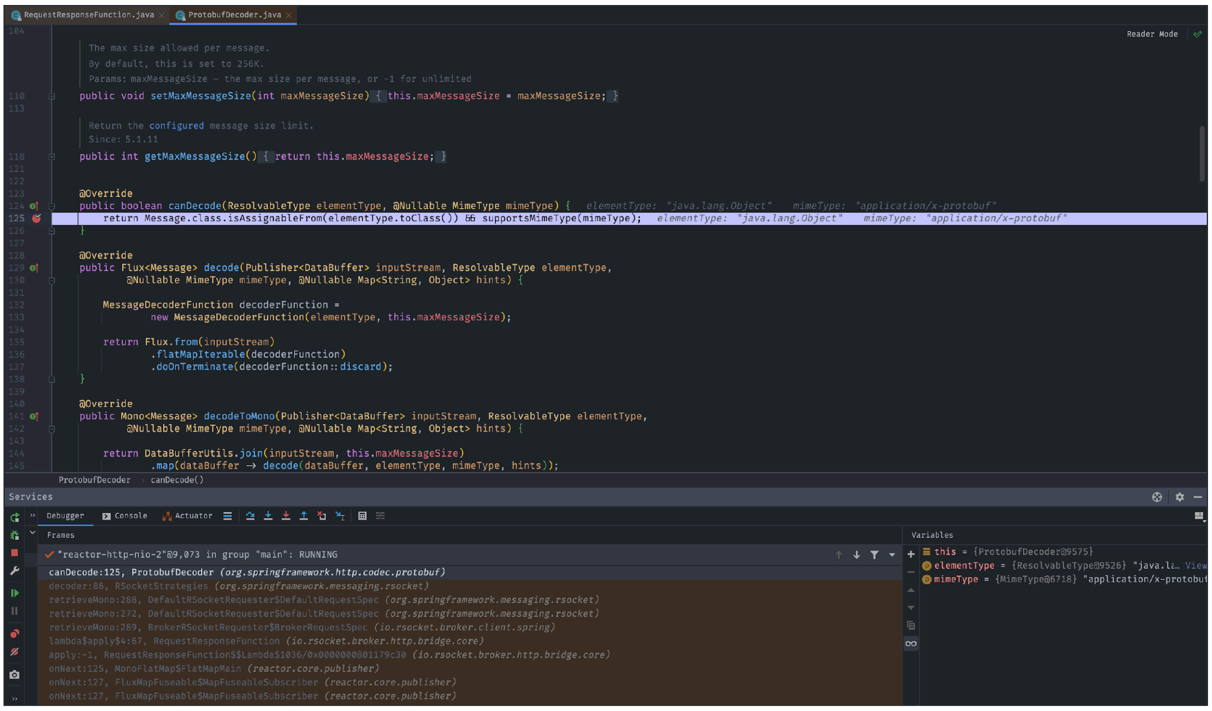
Debugger | Console | Actuator

Frames

```
"reactor-http-nio-2"@9,073 in group "main": RUNNING
canDecode:125, ProtobufDecoder (org.springframework.http.codec.protobuf)
decoder:86, RSocketStrategies (org.springframework.messaging.rsocket)
retrieveMono:288, DefaultRSocketRequester$DefaultRequestSpec (org.springframework.messaging.rsocket)
retrieveMono:272, DefaultRSocketRequester$DefaultRequestSpec (org.springframework.messaging.rsocket)
retrieveMono:289, BrokerRSocketRequester$BrokerRequestSpec (io.rsocket.broker.client.spring)
lambda$apply$4:67, RequestResponseFunction (io.rsocket.broker.http.bridge.core)
apply:-1, RequestResponseFunction$$Lambda$1036/0x0000000801179c30 (io.rsocket.broker.http.bridge.core)
onNext:125, MonoFlatMap$FlatMapMain (reactor.core.publisher)
onNext:127, FluxMapFuseable$MapFuseableSubscriber (reactor.core.publisher)
onNext:127, FluxMapFuseable$MapFuseableSubscriber (reactor.core.publisher)
```

Variables

```
> this = {RequestResponseFunction@9529}
> message = {GenericMessage@9582} "GenericM… View
> uriString = "http://localhost:8010/rsocke… View
> uri = {URI@9589} "http://localhost:8010/r… View
> route = "rr"
> serviceName = "rsocket-server"
  tagString = null
> requester = {BrokerRSocketRequester@9580}
> properties = {RSocketHttpBridgeProperties@9591}
> timeout = {Duration@9592} "PT30S"
```



```
The max size allowed per message.
By default, this is set to 256K.
Params: maxMessageSize – the max size per message, or -1 for unlimited
public void setMaxMessageSize(int maxMessageSize) { this.maxMessageSize = maxMessageSize; }

Return the configured message size limit.
Since: 5.1.11
public int getMaxMessageSize() { return this.maxMessageSize; }


@Override
public boolean canDecode(ResolvableType elementType, @Nullable MimeType mimeType) {   elementType: "java.lang.Object"   mimeType: "application/x-protobuf"
    return Message.class.isAssignableFrom(elementType.toClass()) && supportsMimeType(mimeType);   elementType: "java.lang.Object"   mimeType: "application/x-protobuf"
}

@Override
public Flux<Message> decode(Publisher<DataBuffer> inputStream, ResolvableType elementType,
        @Nullable MimeType mimeType, @Nullable Map<String, Object> hints) {

    MessageDecoderFunction decoderFunction =
            new MessageDecoderFunction(elementType, this.maxMessageSize);

    return Flux.from(inputStream)
            .flatMapIterable(decoderFunction)
            .doOnTerminate(decoderFunction::discard);
}

@Override
public Mono<Message> decodeToMono(Publisher<DataBuffer> inputStream, ResolvableType elementType,
        @Nullable MimeType mimeType, @Nullable Map<String, Object> hints) {

    return DataBufferUtils.join(inputStream, this.maxMessageSize)
            .map(dataBuffer -> decode(dataBuffer, elementType, mimeType, hints));
```

```
ProtobufDecoder › canDecode()
```

```
Frames
"reactor-http-nio-2"@9,073 in group "main": RUNNING
canDecode:125, ProtobufDecoder (org.springframework.http.codec.protobuf)
decoder:86, RSocketStrategies (org.springframework.messaging.rsocket)
retrieveMono:288, DefaultRSocketRequester$DefaultRequestSpec (org.springframework.messaging.rsocket)
retrieveMono:272, DefaultRSocketRequester$DefaultRequestSpec (org.springframework.messaging.rsocket)
retrieveMono:289, BrokerRSocketRequester$BrokerRequestSpec (io.rsocket.broker.client.spring)
lambda$apply$4:67, RequestResponseFunction (io.rsocket.broker.http.bridge.core)
apply:-1, RequestResponseFunction$$Lambda$1036/0x0000000801179c30 (io.rsocket.broker.http.bridge.core)
onNext:125, MonoFlatMap$FlatMapMain (reactor.core.publisher)
onNext:127, FluxMapFuseable$MapFuseableSubscriber (reactor.core.publisher)
onNext:127, FluxMapFuseable$MapFuseableSubscriber (reactor.core.publisher)
```

```
Variables
this = {ProtobufDecoder@9575}
elementType = {ResolvableType@9526} "java.la… View
mimeType = {MimeType@6718} "application/x-protobu
```

However protocol buffer payload is working correctly when I am not using ‘rsocket-broker-http-bridge’ rather directly hitting ‘rsocket-server’ via ‘rsocket-broker’.

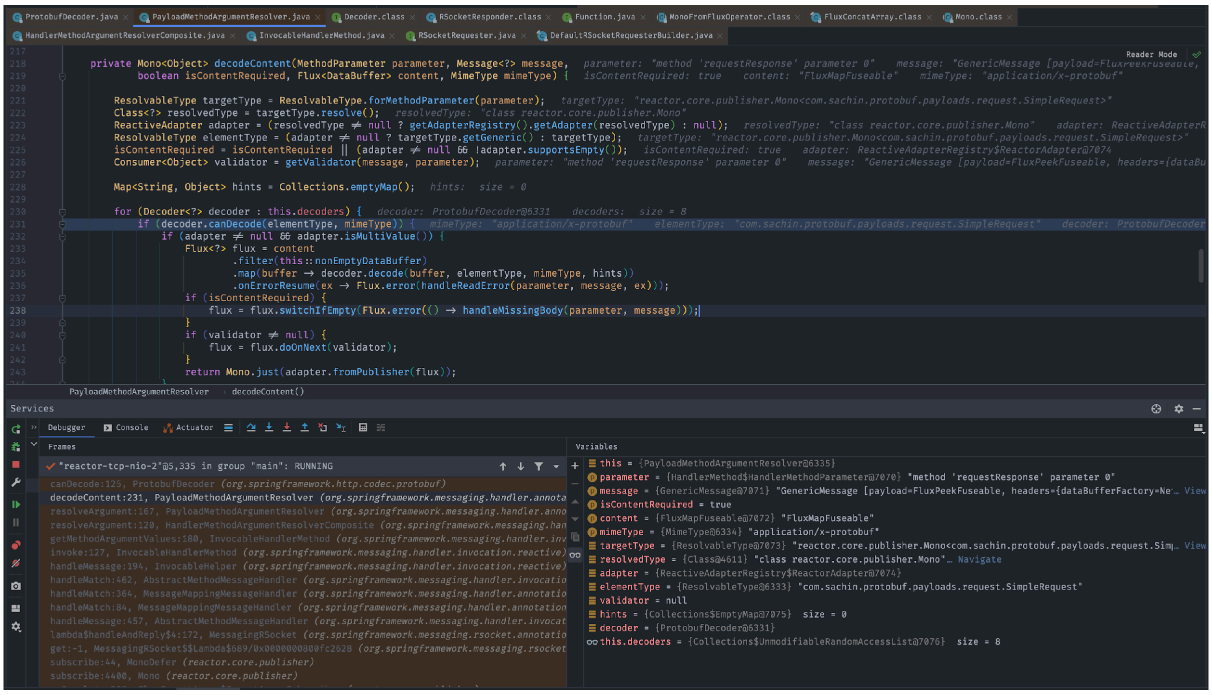
As per my analysis, the hardcoded argument Object.class (at line 67 e.g. .retrieveMono(Object.class)) is causing this issue, as in case of org.springframework.http.codec.protobuf.ProtobufDecoder#canDecode() the condition Message.class.isAssignableFrom(elementType.toClass()) evaluates to false and throws IllegalArgumentException with message Throws No decoder for java.lang.Object

Suggestion: The logic similar to below stackTrace might help in solving this issue, which is being called when I am not using ‘rsocket-broker-http-bridge’.

Suggestion reference stackTrace
canDecode:125, ProtobufDecoder (org.springframework.http.codec.protobuf)
decodeContent:231, PayloadMethodArgumentResolver (org.springframework.messaging.handler.annotation.reactive)
resolveArgument:167, PayloadMethodArgumentResolver (org.springframework.messaging.handler.annotation.reactive)
resolveArgument:120, HandlerMethodArgumentResolverComposite (org.springframework.messaging.handler.invocation.reactive)
getMethodArgumentValues:180, InvocableHandlerMethod (org.springframework.messaging.handler.invocation.reactive)
invoke:127, InvocableHandlerMethod (org.springframework.messaging.handler.invocation.reactive)
handleMessage:194, InvocableHelper (org.springframework.messaging.handler.invocation.reactive)

handleMatch:462, AbstractMethodMessageHandler
(org.springframework.messaging.handler.invocation.reactive)
handleMatch:364, MessageMappingMessageHandler
(org.springframework.messaging.handler.annotation.reactive)
handleMatch:84, MessageMappingMessageHandler
(org.springframework.messaging.handler.annotation.reactive)
handleMessage:457, AbstractMethodMessageHandler
(org.springframework.messaging.handler.invocation.reactive)
lambda$handleAndReply$4:172, MessagingRSocket
(org.springframework.messaging.rsocket.annotation.support)
get:-1, MessagingRSocket$$Lambda$689/0x0000000800fc2628
(org.springframework.messaging.rsocket.annotation.support)

Woking example protocol buffer as payload without using rsocket-broker-http-bridge


ProtobufDecoder.java
PayloadMethodArgumentResolver.java
Decoder.class
RSocketResponder.class
Function.java
MonoFromFluxOperator.class
FluxConcatArray.class
Mono.class
HandlerMethodArgumentResolverComposite.java
InvocableHandlerMethod.java
RSocketRequester.java
DefaultRSocketRequesterBuilder.java
Reader Mode

```
    private Mono<Object> decodeContent(MethodParameter parameter, Message<?> message,   parameter: "method 'requestResponse' parameter 0"    message: "GenericMessage [payload=FluxPeekFuseable,
            boolean isContentRequired, Flux<DataBuffer> content, MimeType mimeType) {   isContentRequired: true    content: "FluxMapFuseable"    mimeType: "application/x-protobuf"

        ResolvableType targetType = ResolvableType.forMethodParameter(parameter);   targetType: "reactor.core.publisher.Mono<com.sachin.protobuf.payloads.request.SimpleRequest>"
        Class<?> resolvedType = targetType.resolve();   resolvedType: "class reactor.core.publisher.Mono"
        ReactiveAdapter adapter = (resolvedType ≠ null ? getAdapterRegistry().getAdapter(resolvedType) : null);   resolvedType: "class reactor.core.publisher.Mono"    adapter: ReactiveAdapterR
        ResolvableType elementType = (adapter ≠ null ? targetType.getGeneric() : targetType);   targetType: "reactor.core.publisher.Mono<com.sachin.protobuf.payloads.request.SimpleRequest>"
        isContentRequired = isContentRequired || (adapter ≠ null && !adapter.supportsEmpty());   isContentRequired: true    adapter: ReactiveAdapterRegistry$ReactorAdapter@7074
        Consumer<Object> validator = getValidator(message, parameter);   parameter: "method 'requestResponse' parameter 0"    message: "GenericMessage [payload=FluxPeekFuseable, headers={dataBu

        Map<String, Object> hints = Collections.emptyMap();   hints:  size = 0

        for (Decoder<?> decoder : this.decoders) {   decoder: ProtobufDecoder@6331    decoders:  size = 8
            if (decoder.canDecode(elementType, mimeType)) {   mimeType: "application/x-protobuf"    elementType: "com.sachin.protobuf.payloads.request.SimpleRequest"    decoder: ProtobufDecoder
                if (adapter ≠ null && adapter.isMultiValue()) {
                    Flux<?> flux = content
                            .filter(this::nonEmptyDataBuffer)
                            .map(buffer → decoder.decode(buffer, elementType, mimeType, hints))
                            .onErrorResume(ex → Flux.error(handleReadError(parameter, message, ex)));
                    if (isContentRequired) {
                        flux = flux.switchIfEmpty(Flux.error(() → handleMissingBody(parameter, message)));
                    }
                    if (validator ≠ null) {
                        flux = flux.doOnNext(validator);
                    }
                    return Mono.just(adapter.fromPublisher(flux));
```

PayloadMethodArgumentResolver › decodeContent()

Services

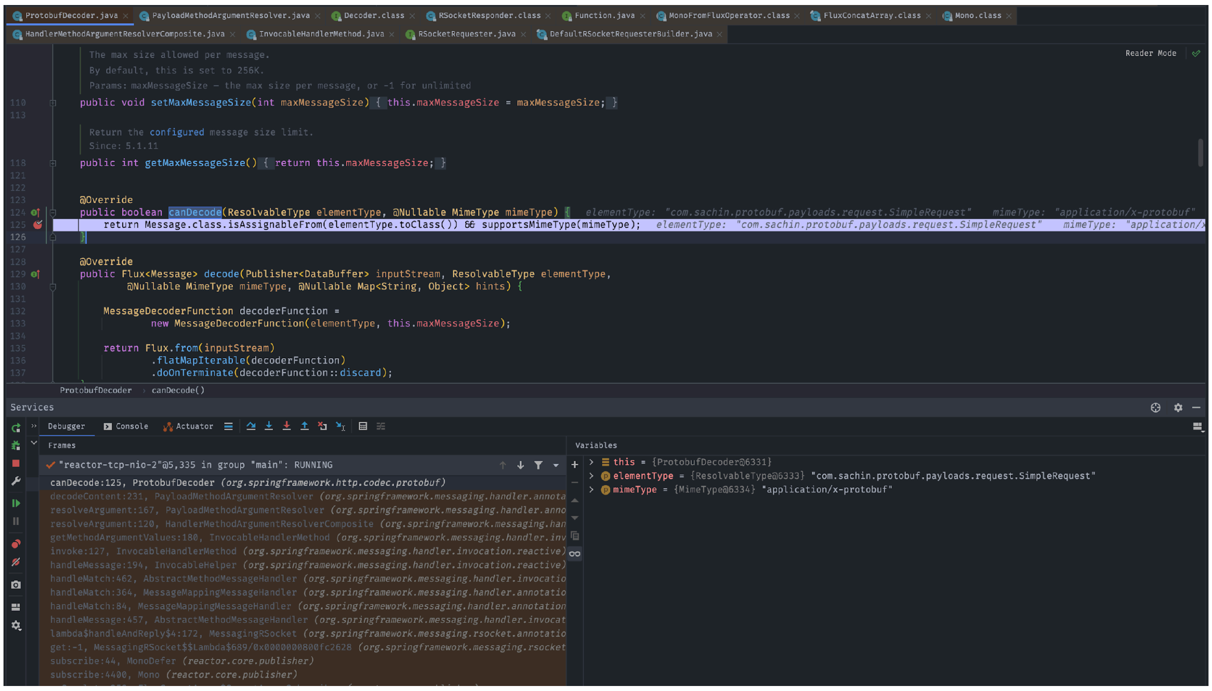
Debugger   Console   Actuator

Frames

```
"reactor-tcp-nio-2"@5,335 in group "main": RUNNING
canDecode:125, ProtobufDecoder (org.springframework.http.codec.protobuf)
decodeContent:231, PayloadMethodArgumentResolver (org.springframework.messaging.handler.annota
resolveArgument:167, PayloadMethodArgumentResolver (org.springframework.messaging.handler.anno
resolveArgument:120, HandlerMethodArgumentResolverComposite (org.springframework.messaging.han
getMethodArgumentValues:180, InvocableHandlerMethod (org.springframework.messaging.handler.inv
invoke:127, InvocableHandlerMethod (org.springframework.messaging.handler.invocation.reactive)
handleMessage:194, InvocableHelper (org.springframework.messaging.handler.invocation.reactive)
handleMatch:462, AbstractMethodMessageHandler (org.springframework.messaging.handler.invocatio
handleMatch:364, MessageMappingMessageHandler (org.springframework.messaging.handler.annotatio
handleMatch:84, MessageMappingMessageHandler (org.springframework.messaging.handler.annotation
handleMessage:457, AbstractMethodMessageHandler (org.springframework.messaging.handler.invocat
lambda$handleAndReply$4:172, MessagingRSocket (org.springframework.messaging.rsocket.annotatio
get:-1, MessagingRSocket$$Lambda$689/0x0000000800fc2628 (org.springframework.messaging.rsocket
subscribe:44, MonoDefer (reactor.core.publisher)
subscribe:4400, Mono (reactor.core.publisher)
```

Variables

```
this = {PayloadMethodArgumentResolver@6335}
parameter = {HandlerMethod$HandlerMethodParameter@7070} "method 'requestResponse' parameter 0"
message = {GenericMessage@7071} "GenericMessage [payload=FluxPeekFuseable, headers={dataBufferFactory=Ne… View
isContentRequired = true
content = {FluxMapFuseable@7072} "FluxMapFuseable"
mimeType = {MimeType@6334} "application/x-protobuf"
targetType = {ResolvableType@7073} "reactor.core.publisher.Mono<com.sachin.protobuf.payloads.request.Sim… View
resolvedType = {Class@4611} "class reactor.core.publisher.Mono"… Navigate
adapter = {ReactiveAdapterRegistry$ReactorAdapter@7074}
elementType = {ResolvableType@6333} "com.sachin.protobuf.payloads.request.SimpleRequest"
validator = null
hints = {Collections$EmptyMap@7075}  size = 0
decoder = {ProtobufDecoder@6331}
this.decoders = {Collections$UnmodifiableRandomAccessList@7076}  size = 8
```


ProtobufDecoder.java
PayloadMethodArgumentResolver.java
Decoder.class
RSocketResponder.class
Function.java
MonoFromFluxOperator.class
FluxConcatArray.class
Mono.class
HandlerMethodArgumentResolverComposite.java
InvocableHandlerMethod.java
RSocketRequester.java
DefaultRSocketRequesterBuilder.java
Reader Mode
ProtobufDecoder › canDecode()
Services
Debugger
Console
Actuator
Frames
Variables

```
    The max size allowed per message.
    By default, this is set to 256K.
    Params: maxMessageSize – the max size per message, or -1 for unlimited
public void setMaxMessageSize(int maxMessageSize) { this.maxMessageSize = maxMessageSize; }

    Return the configured message size limit.
    Since: 5.1.11
public int getMaxMessageSize() { return this.maxMessageSize; }


@Override
public boolean canDecode(ResolvableType elementType, @Nullable MimeType mimeType) {   elementType:  "com.sachin.protobuf.payloads.request.SimpleRequest"    mimeType:  "application/x-protobuf"
    return Message.class.isAssignableFrom(elementType.toClass()) && supportsMimeType(mimeType);   elementType:  "com.sachin.protobuf.payloads.request.SimpleRequest"    mimeType:  "application/x
}

@Override
public Flux<Message> decode(Publisher<DataBuffer> inputStream, ResolvableType elementType,
        @Nullable MimeType mimeType, @Nullable Map<String, Object> hints) {

    MessageDecoderFunction decoderFunction =
            new MessageDecoderFunction(elementType, this.maxMessageSize);

    return Flux.from(inputStream)
            .flatMapIterable(decoderFunction)
            .doOnTerminate(decoderFunction::discard);
```

```
"reactor-tcp-nio-2"@5,335 in group "main": RUNNING
canDecode:125, ProtobufDecoder (org.springframework.http.codec.protobuf)
decodeContent:231, PayloadMethodArgumentResolver (org.springframework.messaging.handler.annota
resolveArgument:167, PayloadMethodArgumentResolver (org.springframework.messaging.handler.anno
resolveArgument:120, HandlerMethodArgumentResolverComposite (org.springframework.messaging.han
getMethodArgumentValues:180, InvocableHandlerMethod (org.springframework.messaging.handler.inv
invoke:127, InvocableHandlerMethod (org.springframework.messaging.handler.invocation.reactive)
handleMessage:194, InvocableHelper (org.springframework.messaging.handler.invocation.reactive)
handleMatch:462, AbstractMethodMessageHandler (org.springframework.messaging.handler.invocatio
handleMatch:364, MessageMappingMessageHandler (org.springframework.messaging.handler.annotatio
handleMatch:84, MessageMappingMessageHandler (org.springframework.messaging.handler.annotation
handleMessage:457, AbstractMethodMessageHandler (org.springframework.messaging.handler.invocat
lambda$handleAndReply$4:172, MessagingRSocket (org.springframework.messaging.rsocket.annotatio
get:-1, MessagingRSocket$$Lambda$689/0x0000000800fc2628 (org.springframework.messaging.rsocket
subscribe:44, MonoDefer (reactor.core.publisher)
subscribe:4400, Mono (reactor.core.publisher)
```

```
> this = {ProtobufDecoder@6331}
> elementType = {ResolvableType@6333} "com.sachin.protobuf.payloads.request.SimpleRequest"
> mimeType = {MimeType@6334} "application/x-protobuf"
```