Overview of the RSocket Connection & Messaging APIs

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the RSocket framework
- Recognize the RSocket interaction models
- Know the RSocketRequester APIs to connect & pass messages

Interface RSocketRequester

All Superinterfaces:

`reactor.core.Disposable`

```
public interface RSocketRequester
extends reactor.core.Disposable
```

A thin wrapper around a sending `RSocket` with a fluent API accepting and returning higher level Objects for input and for output, along with methods to prepare routing and other metadata.

See springframework/messaging/rsocket/RSocketRequester.html

Overview of RSocket Connection APIs

Overview of RSocket Connection APIs

- RSocketRequester
  - Provides a fluent API to perform RSocket requests

Interface RSocketRequester

All Superinterfaces:

`reactor.core.Disposable`

```
public interface RSocketRequester
extends reactor.core.Disposable
```

A thin wrapper around a sending `RSocket` with a fluent API accepting and returning higher level Objects for input and for output, along with methods to prepare routing and other metadata.

See springframework/messaging/rsocket/RSocketRequester.html

Overview of RSocket Connection APIs

- RSocketRequester
  - Provides a fluent API to perform RSocket requests

RSocketRequester	
dataMimeType()	MimeType
wrap(RSocket, MimeType, MimeType, RSocketStrategies)	RSocketRequester
builder()	Builder
metadataMimeType()	MimeType
metadata(Object, MimeType?)	RequestSpec
rsocket()	RSocket?
rsocketClient()	RSocketClient
route(String, Object[])	RequestSpec

Its methods prepare connection & message routing & other metadata for subsequent delivery to a server

Overview of RSocket Connection APIs

- RSocketRequester
  - Provides a fluent API to perform RSocket requests
    - It's not strictly part of the core RSocket API

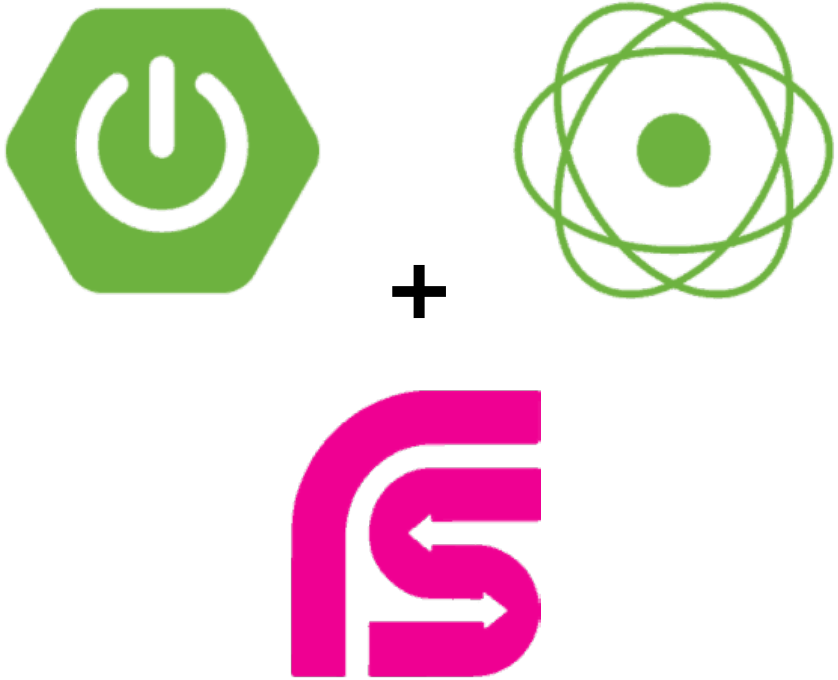
rsocket-core 1.1.3 API

Packages

Package	
io.rsocket	Contains key contracts of the RSocket pr ng model including **RSocket** for performing or handling RSocket interactions, **Acceptor** for declaring responders, **Payload** for access to the content o ad, and others.
io.rsocket.core	ntains **RSocketConnector** and **RSocketServer** ain classes for connecting tarting an RSocket server.
io.rsocket.e ons	A li of exceptions that represent RSocket protocol codes.
io.rsocket.	Support ding and decoding of RSocket frames to an **Payload**.
io.rsocket e.decoder	Support for enc nd decoding of RSocket frames to and **Payload**.
io.rsocket nal	Internal package and t be used outside this project.
io.rsocket. al.jctools.queues	
io.rsocket.k e	Support classes for sending and track of KEEPAL ames from the remote.
io.rsocket.leas	Contains support classes for the Leas of the RS protocol.
io.rsocket.loadba	Support client load-balancing in RSocket J
io.rsocket.metadata	Contains implementations of RSocket **proto nsions** related to the use of metadata.
io.rsocket.plugins	nnections, and requests in in RSocket Java.
io.rsocket.resume	Contains support classes for the RSocket resume capability.
io.rsocket.transport	Client and server transport contracts for pluggable transports.
io.rsocket.util	Shared utility classes and **Payload** implementations.

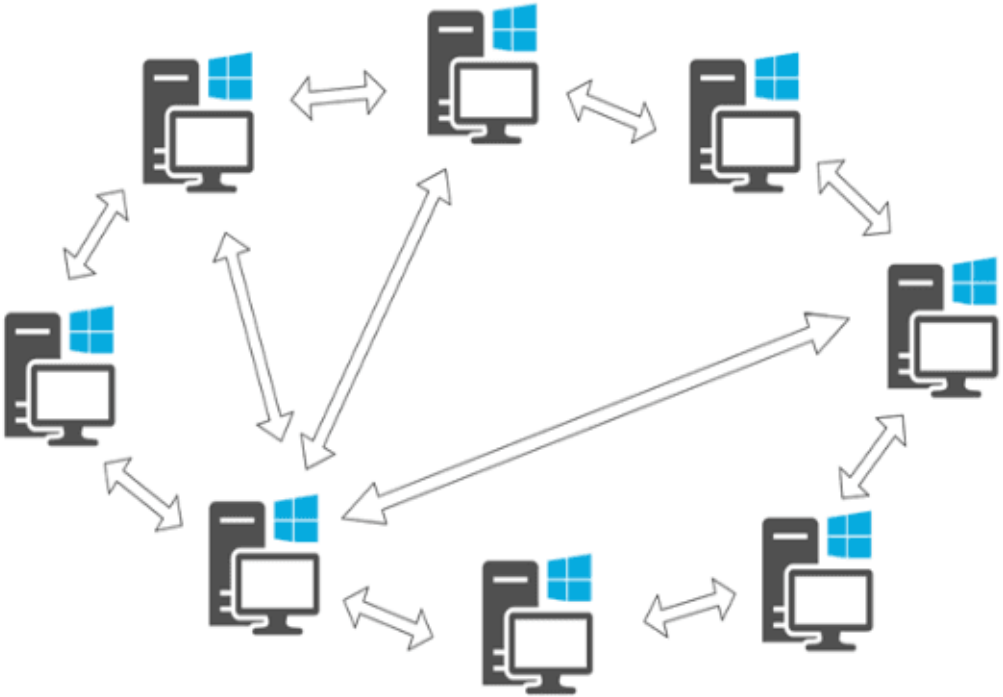
Overview of RSocket Connection APIs

- RSocketRequester
  - Provides a fluent API to perform RSocket requests
    - It's not strictly part of the core RSocket API
    - It's provided by Spring to build reactive microservices using RSocket



Overview of RSocket Connection APIs

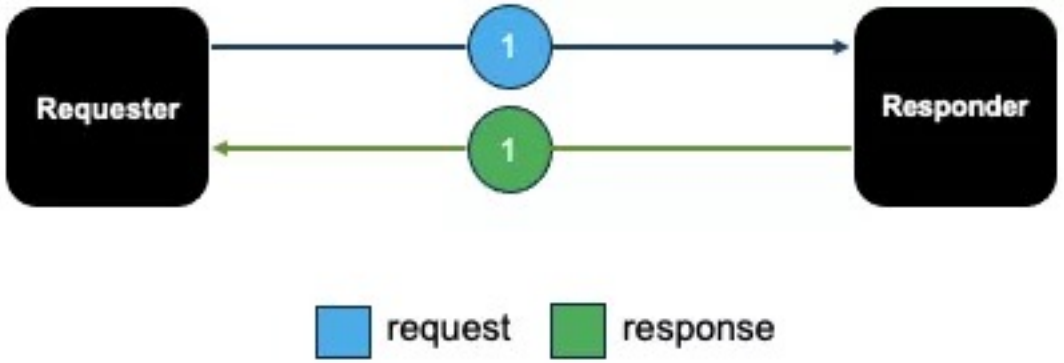
- RSocketRequester
  - Provides a fluent API to perform RSocket requests
  - It can make requests either from "clients" and/or from "servers" symmetrically



See spring.io/blog/2020/03/02/getting-started-with-rsocket-spring-boot-server

Overview of RSocket Connection APIs

- RSocketRequester
  - Provides a fluent API to perform RSocket requests
  - It can make requests either from "clients" and/or from "servers" symmetrically
    - The terms "requester" & "responder" are therefore often used



See www.alibabacloud.com/blog/a-brief-on-rsocket-and-reactive-programming_598219

Overview of RSocket Connection APIs

- To obtain an RSocketRequester on the client side involves a fluent multi-step chain of calls

```
Mono<RSocketRequester>
        getRequester(...) {
  return RSocketRequester
    .builder()
    .dataMimeType(...)
    .rsocketStrategies(...)
    .rsocketConnector(...)
    .setupRoute(...)
    .setupData(...)
    .setupMetadata(...)
    .tcp(...);
```

Fluent Interface

Functional | Reactive | About 2 min

Intent

A fluent interface provides an easy-readable, flowing interface, that often mimics a domain specific language. Using this pattern results in code that can be read nearly as human language.

Explanation

The Fluent Interface pattern is useful when you want to provide an easy readable, flowing API. Those interfaces tend to mimic domain specific languages, so they can nearly be read as human languages.

A fluent interface can be implemented using any of

- Method chaining - calling a method returns some object on which further methods can be called.
- Static factory methods and imports.
- Named parameters - can be simulated in Java using static factory methods.

See java-design-patterns.com/patterns/fluentinterface

Overview of RSocket Connection APIs

- To obtain an RSocketRequester on the client side involves a fluent multi-step chain of calls

```
Mono<RSocketRequester>
        getRequester(...) {
  return RSocketRequester
    .builder()
    .dataMimeType(...)
    .rsocketStrategies(...)
    .rsocketConnector(...)
    .setupRoute(...)
    .setupData(...)
    .setupMetadata(...)
    .tcp(...);
```

Interface RSocketRequester.Builder

Enclosing interface:

RSocketRequester

```
public static interface RSocketRequester.Builder
```

Builder to create a requester by connecting to a server.

Obtain a builder to create a client Rsocket Requester by connecting to an RSocket server

See springframework/messaging/rsocket/RSocketRequester.html#builder

Overview of RSocket Connection APIs

- To obtain an RSocketRequester on the client side involves a fluent multi-step chain of calls

```
Mono<RSocketRequester>
        getRequester(...) {
  return RSocketRequester
     .builder()
     .dataMimeType(...)
     .rsocketStrategies(...)
     .rsocketConnector(...)
     .setupRoute(...)
     .setupData(...)
     .setupMetadata(...)
     .tcp(...);
```

dataMimeType

```
RSocketRequester.Builder dataMimeType(@Nullable
                                      MimeType mimeType)
```

Configure the payload data MimeType to specify on the `SETUP` frame that applies to the whole connection.

If not set, this will be initialized to the MimeType of the first `non-default` `Decoder`, or otherwise the MimeType of the first decoder.

Sets the MIME type for data on the connection

See springframework/messaging/rsocket/RSocketRequester.Builder.html#dataMimeType

Overview of RSocket Connection APIs

- To obtain an RSocketRequester on the client side involves a fluent multi-step chain of calls

```
Mono<RSocketRequester>
        getRequester(...) {
  return RSocketRequester
     .builder()
     .dataMimeType(...)
     .rsocketStrategies(...)
     .rsocketConnector(...)
     .setupRoute(...)
     .setupData(...)
     .setupMetadata(...)
     .tcp(...);
```

rsocketStrategies

```
RSocketRequester.Builder rsocketStrategies
(@Nullable
 RSocketStrategies strategies)
```

Provide the `RSocketStrategies` to use.

This is useful for changing the default settings, yet still allowing further customizations via `rsocketStrategies(Consumer)`. If not set, defaults are obtained from `RSocketStrategies.builder()`.

Parameters:

`strategies` - the strategies to use

Select the Jackson CBOR encoder/decode as the protocol

See springframework/messaging/rsocket/RSocketRequester.Builder.html#rsocketStrategies

Overview of RSocket Connection APIs

- To obtain an RSocketRequester on the client side involves a fluent multi-step chain of calls

```
Mono<RSocketRequester>
        getRequester(...) {
  return RSocketRequester
     .builder()
     .dataMimeType(...)
     .rsocketStrategies(...)
     .rsocketConnector(...)
     .setupRoute(...)
     .setupData(...)
     .setupMetadata(...)
     .tcp(...);
```

rsocketConnector

`RSocketRequester.Builder rsocketConnector (RSocketConnectorConfigurer configurer)`

Callback to configure the `RSocketConnector` directly.

- The data and metadata mime types cannot be set directly on the `RSocketConnector` and will be overridden. Use the shortcuts `dataMimeType(MimeType)` and `metadataMimeType(MimeType)` on this builder instead.
- The frame decoder also cannot be set directly and instead is set to match the configured `DataBufferFactory`.
- For the `setupPayload` , consider using methods on this builder to specify the route, other metadata, and data as Object values to be encoded.
- To configure client side responding, see `RSocketMessageHandler.responder(RSocketStrategies, Object...)`.

Keepalive intervals, interceptors, re-connection policies, acceptors, etc.

See springframework/messaging/rsocket/RSocketRequester.Builder.html#rsocketConnector

Overview of RSocket Connection APIs

- To obtain an RSocketRequester on the client side involves a fluent multi-step chain of calls

```
Mono<RSocketRequester>
        getRequester(...) {
  return RSocketRequester
    .builder()
    .dataMimeType(...)
    .rsocketStrategies(...)
    .rsocketConnector(...)
    .setupRoute(...)
    .setupData(...)
    .setupMetadata(...)
    .tcp(...);
```

setupRoute

```
RSocketRequester.Builder setupRoute(String  route,
                                    Object ... routeVars)
```

Set the route for the setup payload. The rules for formatting and encoding the route are the same as those for a request route as described in `RSocketRequester.route(String, Object...)`.

By default this is not set.

Set up the route to connect with the server

See springframework/messaging/rsocket/RSocketRequester.Builder.html#setupRoute

Overview of RSocket Connection APIs

- To obtain an RSocketRequester on the client side involves a fluent multi-step chain of calls

```
Mono<RSocketRequester>
        getRequester(...) {
  return RSocketRequester
    .builder()
    .dataMimeType(...)
    .rsocketStrategies(...)
    .rsocketConnector(...)
    .setupRoute(...)
    .setupData(...)
    .setupMetadata(...)
    .tcp(...);
```

setupData

`RSocketRequester.Builder setupData(Object data)`

Set the data for the setup payload. The data will be encoded according to the configured `dataMimeType(MimeType)`. The data be a concrete value or any producer of a single value that can be adapted to a `Publisher` via `ReactiveAdapterRegistry`.

By default this is not set.

Set up the data payload to send the server initially

See springframework/messaging/rsocket/RSocketRequester.Builder.html#setupData

Overview of RSocket Connection APIs

- To obtain an RSocketRequester on the client side involves a fluent multi-step chain of calls

```
Mono<RSocketRequester>
        getRequester(...) {
  return RSocketRequester
     .builder()
     .dataMimeType(...)
     .rsocketStrategies(...)
     .rsocketConnector(...)
     .setupRoute(...)
     .setupData(...)
     .setupMetadata(...)
     .tcp(...);
```

setupMetadata

```
RSocketRequester.Builder setupMetadata(Object  value,
                                       @Nullable
                                       MimeType mimeType)
```

Add metadata entry to the setup payload. Composite metadata must be in use if this is called more than once or in addition to `setupRoute(String, Object...)`. The metadata value be a concrete value or any producer of a single value that can be adapted to a `Publisher` via `ReactiveAdapterRegistry`.

Set up the metadata to pass login credentials

See springframework/messaging/rsocket/RSocketRequester.Builder.html#setupMetadata

Overview of RSocket Connection APIs

- To obtain an RSocketRequester on the client side involves a fluent multi-step chain of calls

```
Mono<RSocketRequester>
        getRequester(...) {
  return RSocketRequester
     .builder()
     .dataMimeType(...)
     .rsocketStrategies(...)
     .rsocketConnector(...)
     .setupRoute(...)
     .setupData(...)
     .setupMetadata(...)
     .tcp(...);
```

setupMetadata

```
RSocketRequester.Builder setupMetadata(Object  value,
                                       @Nullable
                                       MimeType mimeType)
```

Add metadata entry to the setup payload. Composite metadata must be in use if this is called more than once or in addition to `setupRoute(String, Object...)`. The metadata value be a concrete value or any producer of a single value that can be adapted to a `Publisher` via `ReactiveAdapterRegistry`.

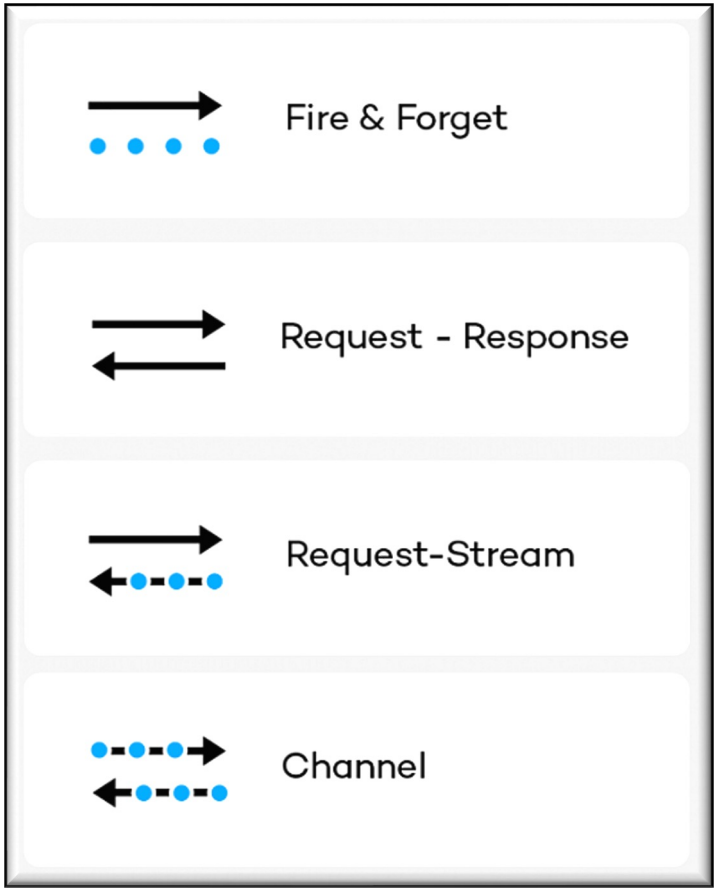
Build a requester that connects to the server host at a particular port

See springframework/messaging/rsocket/RSocketRequester.Builder.html#tcp

Overview of RSocket Messaging APIs

Overview of RSocket Messaging APIs

- Message requests can be sent after an RSocketRequester is created



See earlier lesson on "*Overview of RSocket Interaction Models*"

Overview of RSocket Messaging APIs

- Message requests can be sent after an RSocketRequester is created

```
Mono<Integer> results =
  requester
    .map(r -> r.route
        (GET_NUMBER_OF_QUOTES))

    .flatMap(r -> r
        .retrieveMono
            (Integer.class));
```

Create metadata for a message containing no data that will be sent to the server to get the # of quotes

route

```
RSocketRequester.RequestSpec route(String  route,
                                   Object ... routeVars)
```

Begin to specify a new request with the given route to a remote handler.

The route can be a template with placeholders, e.g. "`flight.{code}`" in which case the supplied route variables are formatted via `toString()` and expanded into the template. If a formatted variable contains a "." it is replaced with the escape sequence "%2E" to avoid treating it as separator by the responder.

If the connection is set to use composite metadata, the route is encoded as "`message/x.rsocket.routing.v0`". Otherwise, the route is encoded according to the mime type for the connection.

Parameters:

`route` - the route expressing a remote handler mapping

`routeVars` - variables to be expanded into the route template

Returns:

a spec for further defining and executing the request

See springframework/messaging/rsocket/RSocketRequester.html#route

Overview of RSocket Messaging APIs

- Message requests can be sent after an RSocketRequester is created

```
Mono<Integer> results =
  requester
    .map(r -> r.route
        (GET_NUMBER_OF_QUOTES))

    .flatMap(r -> r
        .retrieveMono
            (Integer.class));
```

Perform a two-way call to the server to get the max # of quotes, which is then returned as a Mono<Integer>

retrieveMono

`<T> reactor.core.publisher.Mono<T> retrieveMono (Class <T> dataType)`

Perform a `requestResponse` exchange.

If the return type is `Mono<Void>`, the `Mono` will complete after all data is consumed.

Note: This method will raise an error if the request payload is a multivalued `Publisher` as there is no many-to-one RSocket interaction.

Type Parameters:

`T` - parameter for the expected data type

Parameters:

`dataType` - the expected data type for the response

Returns:

the decoded response

See springframework/messaging/rsocket/RSocketRequester.RetrieveSpec.html#retrieveMono

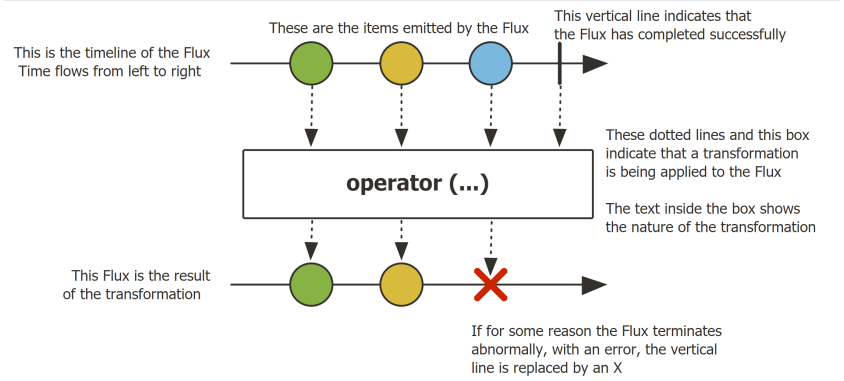
Overview of RSocket Messaging APIs

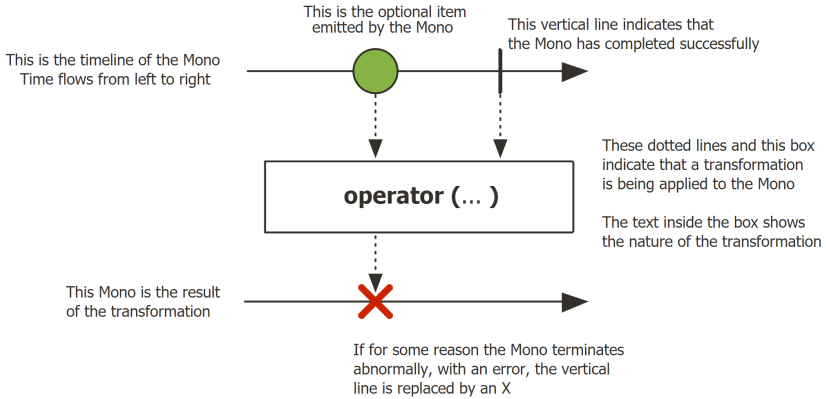
- Message requests can be sent after an RSocketRequester is created

```
Mono<Integer> results =
  requester
    .map(r -> r.route
        (GET_NUMBER_OF_QUOTES))

    .flatMap(r -> r
        .retrieveMono
            (Integer.class));
```

Project Reactor operators are applied on Mono & Flux types to initiate & handle messaging calls





See spring.io/blog/2016/04/19/understanding-reactive-types

Overview of RSocket Messaging APIs

- Message requests can be sent after an RSocketRequester is created

```
Flux<Quote> results = Mono
   .just(requester
      .map(r -> r
         .route(GET_ALL_QUOTES)
         .data(subscription))

   .flatMapMany(r -> r
      .retrieveFlux(Quote.class));
```

route

```
RSocketRequester.RequestSpec route(String  route,
                                   Object ... routeVars)
```

Begin to specify a new request with the given route to a remote handler.

The route can be a template with placeholders, e.g. "`flight.{code}`" in which case the supplied route variables are formatted via `toString()` and expanded into the template. If a formatted variable contains a "." it is replaced with the escape sequence "%2E" to avoid treating it as separator by the responder.

Create metadata for a message containing a confirmed subscription that's used get a Flux of all the Zippy quotes

See springframework/messaging/rsocket/RSocketRequester.html#route

Overview of RSocket Messaging APIs

- Message requests can be sent after an RSocketRequester is created

```
Flux<Quote> results = Mono
   .just(requester
      .map(r -> r
         .route(GET_ALL_QUOTES)
         .data(subscription))

   .flatMapMany(r -> r
      .retrieveFlux(Quote.class));
```

data

`RSocketRequester.RetrieveSpec data(Object data)`

Provide payload data for the request. This can be one of:

- Concrete value
- `Publisher` of value(s)
- Any other producer of value(s) that can be adapted to a `Publisher` via `ReactiveAdapterRegistry`

Parameters:

`data` - the Object value for the payload data

Returns:

spec to declare the expected response

Send the confirmed subscription as the data

See springframework/messaging/rsocket/RSocketRequester.RequestSpec.html#data

Overview of RSocket Messaging APIs

- Message requests can be sent after an RSocketRequester is created

```
Flux<Quote> results = Mono
   .just(requester
      .map(r -> r
         .route(GET_ALL_QUOTES)
         .data(subscription))

   .flatMapMany(r -> r
      .retrieveFlux(Quote.class));
```

Send a message to the server to obtain a Flux that emits all the Zippy quotes

retrieveFlux

`<T> reactor.core.publisher.Flux<T> retrieveFlux(Class <T> dataType)`

Perform an `requestStream` or a `requestChannel` exchange depending on whether the request input is single or multi-payload.

If the return type is `Flux<Void>`, the `Flux` will complete after all data is consumed.

Type Parameters:

`T` - parameterize the expected type of values

Parameters:

`dataType` - the expected type for values in the response

Returns:

the decoded response

See springframework/messaging/rsocket/RSocketRequester.RetrieveSpec.html#retrieveFlux

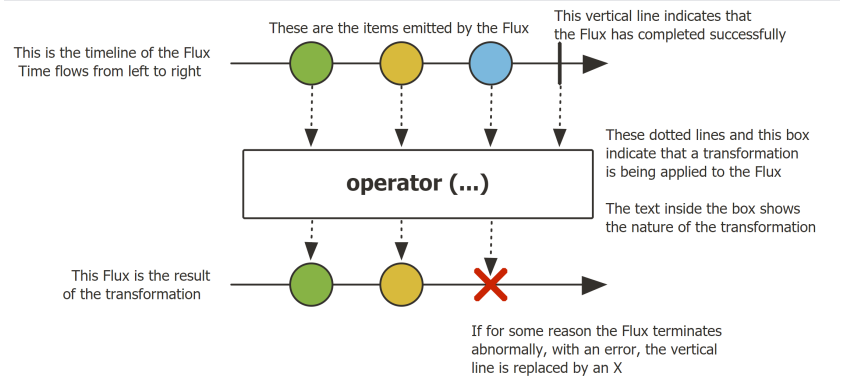
Overview of RSocket Messaging APIs

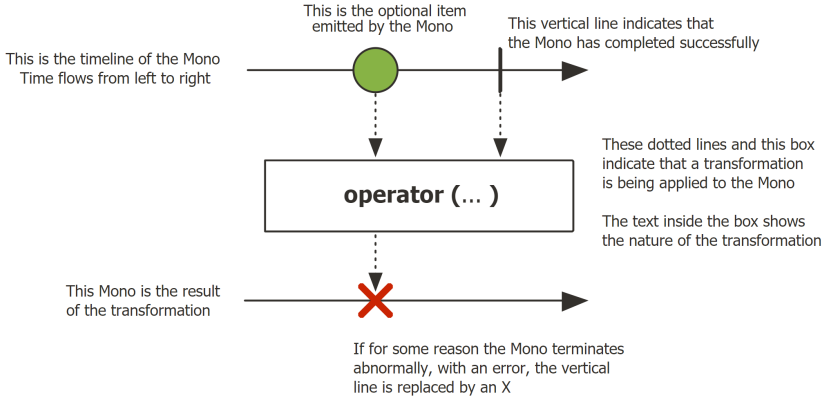
- Message requests can be sent after an RSocketRequester is created

```
Flux<Quote> results = Mono
    .just(requester
        .map(r -> r
            .route(GET_ALL_QUOTES)
            .data(subscription))

    .flatMapMany(r -> r
        .retrieveFlux(Quote.class));
```

Once again, Project Reactor operators are applied on Mono & Flux types to initiate & handle messaging calls





See spring.io/blog/2016/04/19/understanding-reactive-types

End of Overview of the RSocket Connection & Messaging APIs