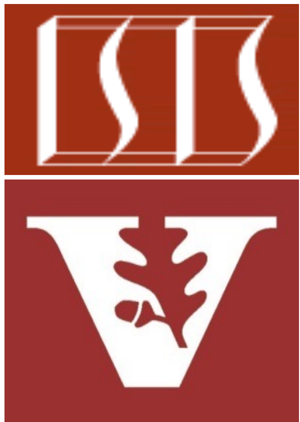
The Structure & Functionality of the RSocket ZippyQuotes Client

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

Professor of Computer Science

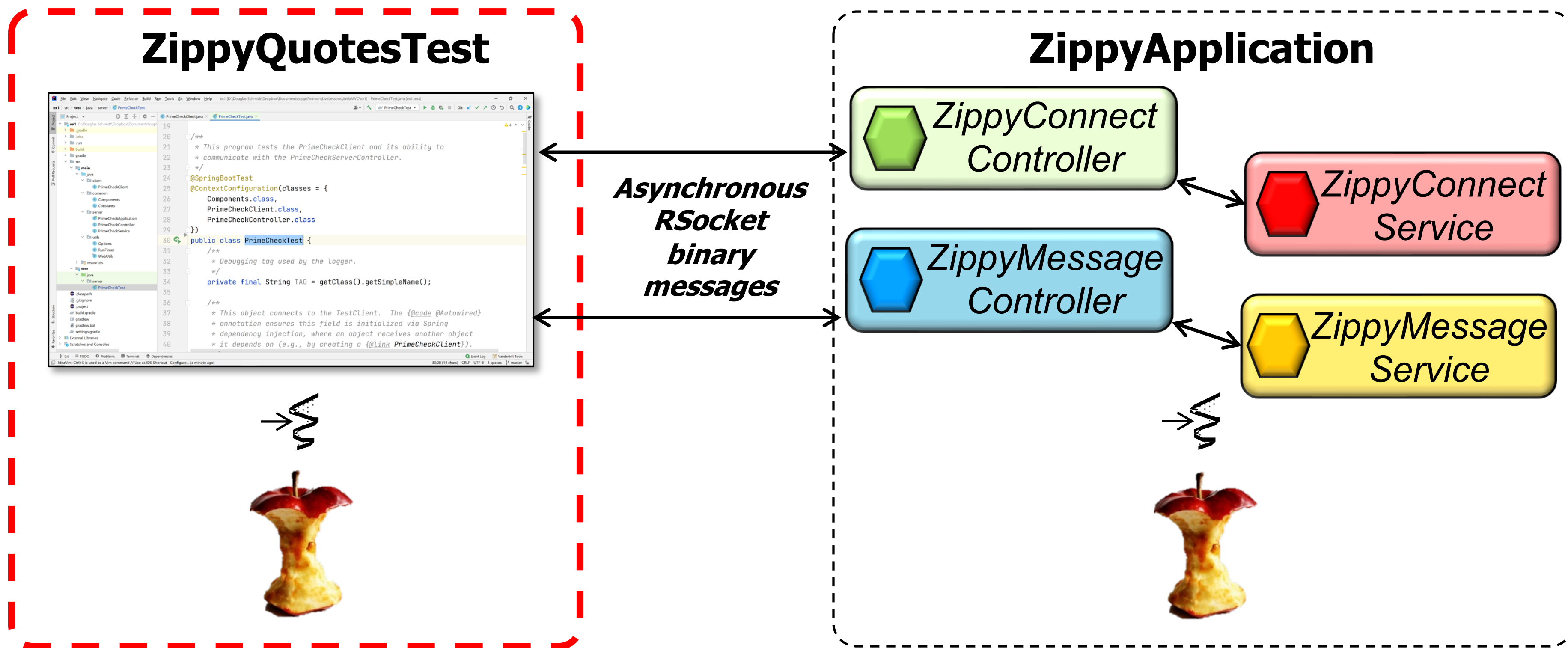
Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA



Learning Objectives in this Lesson

- Understand the structure & functionality of the RSocket ZippyQuotes client that exchanges messages asynchronously with the server via reactive types



See github.com/douglascraigschmidt/LiveLessons/tree/master/RSocket/ex1

Overview of the ZippyQuotes Client Connection Initiator

Overview of the ZippyQuotes Client Connection Initiator

- The ClientBeans class defines a @Bean that returns a Mono emitting an RSocketRequester connected to the server

ClientBeans	
mMimeType	MimeType
mCredentials	UsernamePasswordMetadata
connectorStrategies	Consumer<RSocketConnector>
mResponder	SocketAcceptor
mRsocketStrategies	RSocketStrategies
TAG	String
getRSocketRequester()	Mono<RSocketRequester>

See RSocket/ex1/src/test/java/zippyisms/client/ClientBeans.java

Overview of the ZippyQuotes Client Connection Initiator

- 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(...);
```

See en.wikipedia.org/wiki/Fluent_interface

Overview of the ZippyQuotes Client Connection Initiator

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

```
Mono<RSocketRequester>
                  getRequester(...) {
   return RSocketRequester
     .builder()
     ...
```

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

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

Overview of the ZippyQuotes Client Connection Initiator

- To obtain an RSocketRequester on the client side involves a fluent multi-step chain of calls
  - The mime type for data on the connection can be set

```
Mono<RSocketRequester>
                  getRequester(...) {
   return RSocketRequester
     .builder()

     .dataMimeType(MediaType
                    .APPLICATION_CBOR)
     ...
```

Use the concise binary object representation (CBOR) protocol

See cbor.io

Overview of the ZippyQuotes Client Connection Initiator

- RSocketRequester.Builder accepts RSocketStrategies to configure the requester

```
Mono<RSocketRequester>
                  getRequester(...) {
  return RSocketRequester
    .builder()
    ...
    .rsocketStrategies
      (rSsocketStrategies)
```

Select the Jackson CBOR encoder/decode as the protocol

```
new SimpleAuthenticationEncoder
```

```
var rSocketStrategies = RSocketStrategies
   .builder()
   .encoders(encoders -> encoders.add(new Jackson2CborEncoder()))
   .decoders(decoders -> decoders.add(new Jackson2CborDecoder()))
   .encoder(new SimpleAuthenticationEncoder())
   .build();
```

See springframework/http/codec/cbor/Jackson2CborEncoder.html

Overview of the ZippyQuotes Client Connection Initiator

- RSocketRequester.Builder provides a callback that exposes the under lying RSocketConnector for further configuration options

```
Mono<RSocketRequester>
                   getRequester(...) {
  ...
  return RSocketRequester
    .builder()
    ...
    .rsocketConnector(
       connectorStrategies::
       accept)
    ...
```

```
Consumer<RSocketConnector> connectorStrategies =
    connector -> connector
    .reconnect(Retry.fixedDelay(2, ofSeconds(2)))
    .acceptor(mResponder);
```

Overview of the ZippyQuotes Client Connection Initiator

- RSocketRequester.Builder provides a callback that exposes the underlying RSocketConnector for further configuration options, e.g.
  - Keepalive intervals, interceptors, session resumption, acceptors, etc.

```
Mono<RSocketRequester>
                  getRequester(...) {
  ...
  return RSocketRequester
    .builder()
    ...
    .rsocketConnector(
       connectorStrategies::
       accept)
    ...
```

Try to reconnect twice, with a delay of 2 seconds per retry

```
Consumer<RSocketConnector> connectorStrategies =
      connector -> connector
      .reconnect(Retry.fixedDelay(2, ofSeconds(2)))
      .acceptor(mResponder);
```

Overview of the ZippyQuotes Client Connection Initiator

- RSocketRequester.Builder provides a callback that exposes the underlying RSocketConnector for further configuration options, e.g.
  - Keepalive intervals, interceptors, session resumption, acceptors, etc.

```
SocketAcceptor mResponder =
  RSocketMessageHandler
    .responder(mRsocketStrategies,
               new
      ConnectResponseHandler());
```

Receives server's response to the connection request

```
Consumer<RSocketConnector> connectorStrategies =
    connector -> connector
    .reconnect(Retry.fixedDelay(2, ofSeconds(2)))
    .acceptor(mResponder);
```

See javadoc.io/static/io.rsocket/rsocket-core/1.1.1/io/rsocket/SocketAcceptor.html

Overview of the ZippyQuotes Client Connection Initiator

- There are also methods the client uses to connect with the server & authenticate itself

```
Mono<RSocketRequester>
                  getRequester(...) {
   ...
   return RSocketRequester
      .builder()
      ...
      .setupRoute(SERVER_CONNECT)
      ...
```

Set up the route to connect with the server

See earlier discussion about @ConnectMapping

Overview of the ZippyQuotes Client Connection Initiator

- There are also methods the client uses to connect with the server & authenticate itself

```
Mono<RSocketRequester>
                  getRequester(...) {
   ...
   return RSocketRequester
      .builder()
      ...
      .setupMetadata
          (mCreds,
           mMimeType)
      ...
```

```
UsernamePasswordMetadata mCreds
   = new UsernamePasswordMetadata
       ("d.schmidt@vanderbilt.edu",
        "you-shall-not-pass");

MimeType mMimeType = MimeTypeUtils
   .parseMimeType(MESSAGE_RSOCKET_AUTHENTICATION.getString());
```

Set up the metadata to pass the credentials

Overview of the ZippyQuotes Client Connection Initiator

- There are also methods the client uses to connect with the server & authenticate itself

```
Mono<RSocketRequester>
                getRequester(...) {
  ...
  return RSocketRequester
    .builder()
    ...
    .setupData(UUID
                .randomUUID()
                .toString())
    ...
```

Set up the data payload to send the server initially

Overview of the ZippyQuotes Client Connection Initiator

- Finally, the chain must connect with the server to obtain a Mono to an RSocketRequester

```
Mono<RSocketRequester>
                  getRequester(...) {
  ...
  return RSocketRequester
     .builder()
     ...
     .tcp(LOCAL_HOST,
          SERVER_PORT);
```

Build a requester that's connected to the localhost via TCP SERVER_PORT

Overview of the ZippyProxy Class

Overview of the ZippyProxy Class

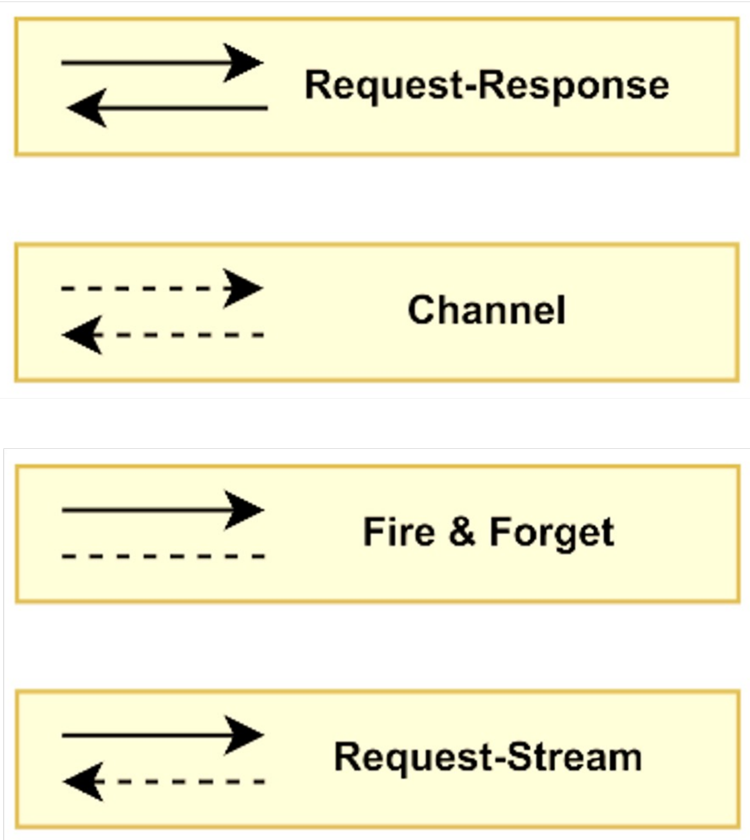
- This proxy defines methods used to send messages to endpoints provided by the ZippyApplication

ZippyProxy	
mZippyQuoteRequester	Mono<RSocketRequester>
getAllQuotes(Mono<Subscription>)	Flux<Quote>
getRandomQuotes(Mono<Integer[]>)	Flux<Quote>
getAllQuotes(Tuple2<RSocketRequester, Subscription>)	Flux<Quote>
getQuoteMax()	Mono<Integer>
cancelUnconfirmed(Mono<Subscription>)	Mono<Void>
subscribe(UUID)	Mono<Subscription>
cancelConfirmed(UUID)	Mono<Subscription>
closeConnection()	void
makeRandomIndices(int)	Mono<Integer[]>
cancelConfirmed(Mono<Subscription>)	Mono<Subscription>

See RSocket/ex1/src/test/java/zippyisms/client/ZippyProxy.java

Overview of the ZippyProxy Class

- This proxy defines methods used to send messages to endpoints provided by the ZippyApplication
  - These methods demo each of the four interaction models supported by RSocket



Overview of the ZippyProxy Class

- Message requests can be sent after an RSocketRequester is created

```
class ZippyProxy {
  @Autowired
  private Mono<RSocketRequester>
    mZippyQuoteRequester;
  ...
```

See RSocket/ex1/src/test/java/zippyisms/client/ZippyProxy.java

Overview of the ZippyProxy Class

- Message requests can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method

```
Mono<Subscription> subscribe
                    (UUID uuid) {
  return mZippyQuoteRequester
    .map(r -> r
         .route(SUBSCRIBE)
         .data(new Subscription
                    (uuid)))

    .flatMap(r -> r
           .retrieveMono
              (Subscription
               .class))

    .cache();
}
```

Create & return a Subscription that's confirmed by the server

Overview of the ZippyProxy Class

- Message requests can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method

```
Mono<Subscription> subscribe
                        (UUID uuid) {
  return mZippyQuoteRequester
    .map(r -> r
         .route(SUBSCRIBE)
         .data(new Subscription
                    (uuid)))

    .flatMap(r -> r
              .retrieveMono
                 (Subscription
                  .class))

    .cache();
}
```

Create metadata for a message containing data that will be sent to the server to subscribe for quotes

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

Overview of the ZippyProxy Class

- Message requests can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method

```
Mono<Subscription> subscribe
                          (UUID uuid) {
  return mZippyQuoteRequester
    .map(r -> r
         .route(SUBSCRIBE)
         .data(new Subscription
                    (uuid)))

    .flatMap(r -> r
             .retrieveMono
                (Subscription
                 .class))

    .cache();
}
```

Create data for a message containing a subscription id

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

Overview of the ZippyProxy Class

- Message requests can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method

```
Mono<Subscription> subscribe
                       (UUID uuid) {
  return mZippyQuoteRequester
    .map(r -> r
         .route(SUBSCRIBE)
         .data(new Subscription
                     (uuid)))


    .flatMap(r -> r
               .retrieveMono
                 (Subscription
                  .class))


    .cache();
}
```

Send a message to the server to subscribe w/ the given subscription & return a confirmed Subscription as a Mono<Subscription>

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

Overview of the ZippyProxy Class

- Message requests can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method

```
Mono<Subscription> subscribe
                      (UUID uuid) {
  return mZippyQuoteRequester
    .map(r -> r
         .route(SUBSCRIBE)
         .data(new Subscription
                      (uuid)))

    .flatMap(r -> r
             .retrieveMono
                (Subscription
                 .class))

    .cache();
}
```

Project Reactor operators are applied on Mono & Flux types to initiate & handle two-way message passing

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

Overview of the ZippyProxy Class

- Message requests can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method

```
Mono<Subscription> subscribe
                       (UUID uuid) {
  return mZippyQuoteRequester
    .map(r -> r
         .route(SUBSCRIBE)
         .data(new Subscription
                    (uuid)))

    .flatMap(r -> r
              .retrieveMono
                 (Subscription
                  .class))

    .cache();
}
```

Turn this Mono into a hot source & cache last emitted signals for future subscribers

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html#cache

Overview of the ZippyProxy Class

- Message requests can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method
  - The getRandomQuotes() proxy method

```
Flux<Quote> getRandomQuotes
  (Mono<Integer[]> randomIndices) {
  return mZippyQuoteRequester
    .zipWith(randomIndices)

    .map(tuple -> tuple.getT1()
          .route(GET_QUOTES)
          .data(Flux.fromArray
                   (tuple.getT2()))

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

Returns a Flux that emits random Zippy quotes

Overview of the ZippyProxy Class

- Message requests can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method
  - The getRandomQuotes() proxy method

```
Flux<Quote> getRandomQuotes
   (Mono<Integer[]> randomIndices) {
  return mZippyQuoteRequester
    .zipWith(randomIndices)

    .map(tuple -> tuple.getT1()
          .route(GET_QUOTES)
          .data(Flux.fromArray
                   (tuple.getT2()))

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

Combine the result from this Mono & the other Mono into a Tuple2

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html#zipWith

Overview of the ZippyProxy Class

- Message requests can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method
  - The getRandomQuotes() proxy method

```
Flux<Quote> getRandomQuotes
   (Mono<Integer[]> randomIndices) {
  return mZippyQuoteRequester
    .zipWith(randomIndices)

    .map(tuple -> tuple.getT1()
          .route(GET_QUOTES)
          .data(Flux.fromArray
                     (tuple.getT2()))

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

Get the RSocketRequester from the first tuple element

Overview of the ZippyProxy Class

- Message requests can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method
  - The getRandomQuotes() proxy method

```
Flux<Quote> getRandomQuotes
   (Mono<Integer[]> randomIndices) {
  return mZippyQuoteRequester
    .zipWith(randomIndices)

    .map(tuple -> tuple.getT1()
           .route(GET_QUOTES)
           .data(Flux.fromArray
                   (tuple.getT2()))

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

Create metadata for a message containing data that will be sent to the server to get the specified quotes

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

Overview of the ZippyProxy Class

- Message requests can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method
  - The getRandomQuotes() proxy method

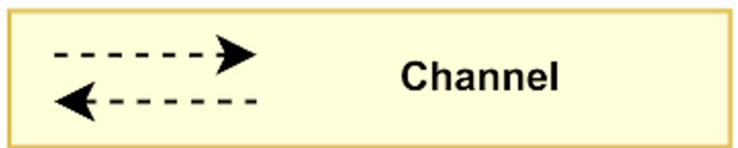
```
Flux<Quote> getRandomQuotes
   (Mono<Integer[]> randomIndices) {
  return mZippyQuoteRequester
    .zipWith(randomIndices)

    .map(tuple -> tuple.getT1()
          .route(GET_QUOTES)
          .data(Flux.fromArray
                  (tuple.getT2()))

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

Create data for a message thaqt contains a Flux of Integer quote IDs

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

Overview of the ZippyProxy Class

- Message requests can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method
  - The getRandomQuotes() proxy method




```
Flux<Quote> getRandomQuotes
   (Mono<Integer[]> randomIndices) {
  return mZippyQuoteRequester
    .zipWith(randomIndices)

    .map(tuple -> tuple.getT1()
         .route(GET_QUOTES)
         .data(Flux.fromArray
                  (tuple.getT2()))

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

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

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

Overview of the ZippyProxy Class

- Message requests can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method
  - The getRandomQuotes() proxy method

```
Flux<Quote> getRandomQuotes
   (Mono<Integer[]> randomIndices) {
  return mZippyQuoteRequester
    .zipWith(randomIndices)

    .map(tuple -> tuple.getT1()
          .route(GET_QUOTES)
          .data(Flux.fromArray
                 (tuple.getT2()))

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

Returns a Flux from Mono

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html#flatMapMany

End of Overview of the RSocket Connection & Messaging APIs