Integrating the RSocket APIs in Spring

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

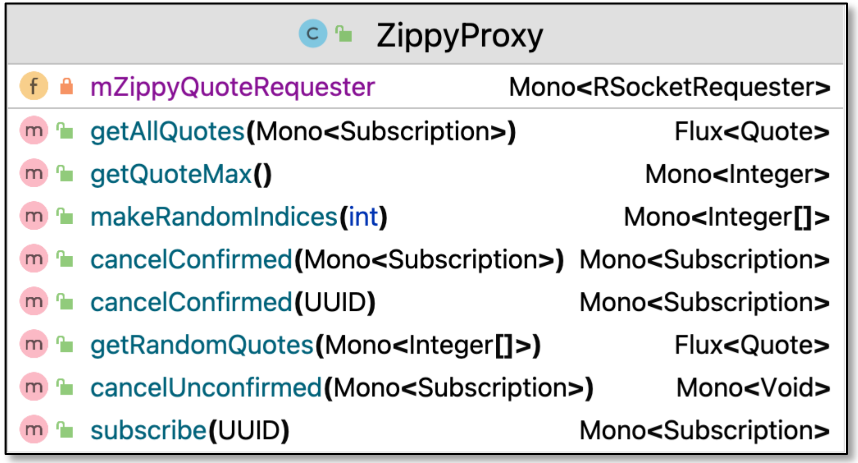
Professor of Computer Science

Institute for Software
Integrated Systems

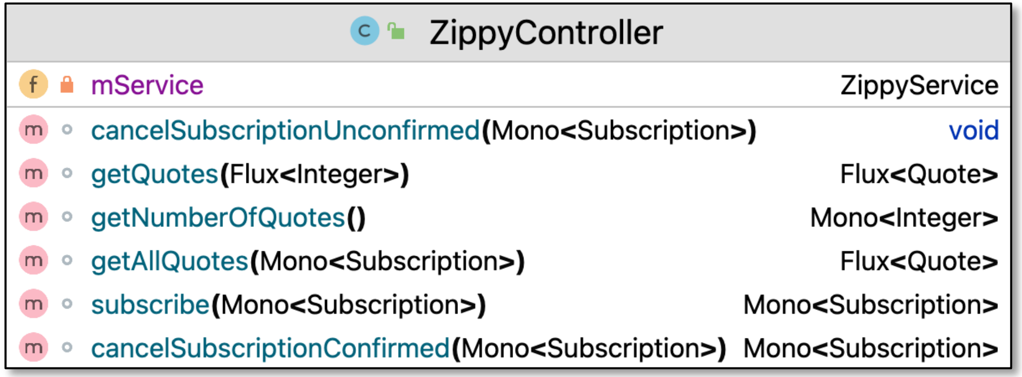
Vanderbilt University
Nashville, Tennessee, USA

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
- Recognize how RSocket can be integrated into Spring



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



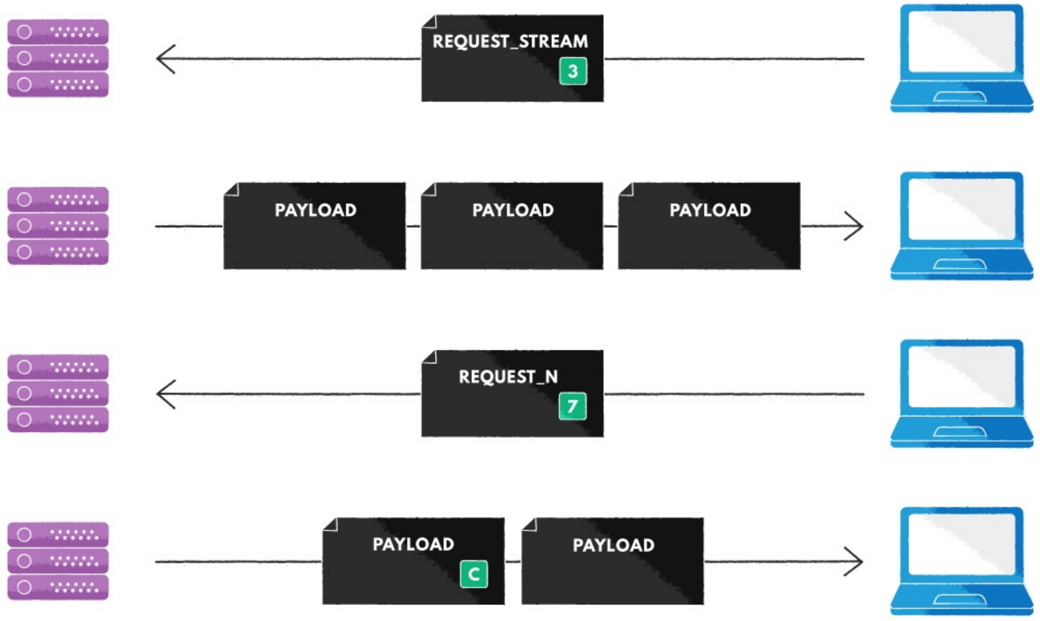
ZippyController	
mService	ZippyService
cancelSubscriptionUnconfirmed(Mono<Subscription>)	void
getQuotes(Flux<Integer>)	Flux<Quote>
getNumberOfQuotes()	Mono<Integer>
getAllQuotes(Mono<Subscription>)	Flux<Quote>
subscribe(Mono<Subscription>)	Mono<Subscription>
cancelSubscriptionConfirmed(Mono<Subscription>)	Mono<Subscription>

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

Integrating the RSocket APIs with Spring

Integrating the RSocket APIs with Spring

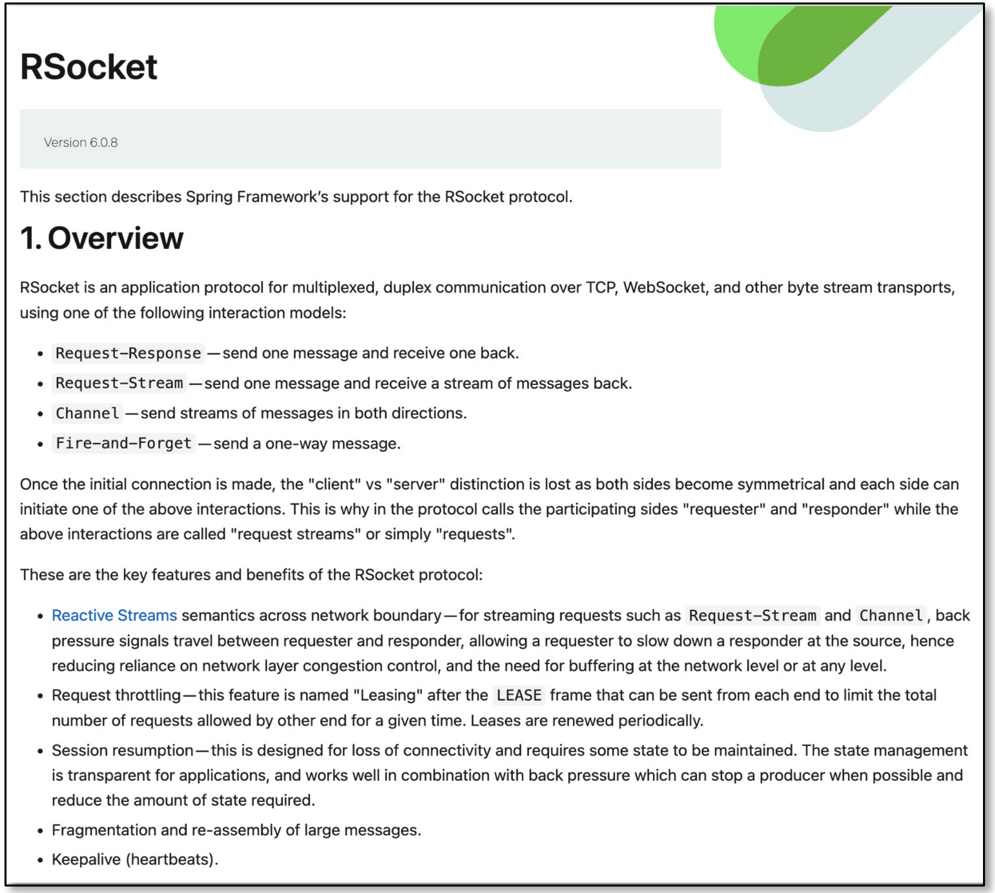
- RSocket can be used without any connection to Spring whatsoever



See rsocket.io

Integrating the RSocket APIs with Spring

- RSocket can be used without any connection to Spring whatsoever
  - However, it's generally more effective to leverage Spring's integration of RSocket



RSocket

Version 6.0.8

This section describes Spring Framework's support for the RSocket protocol.

1. Overview

RSocket is an application protocol for multiplexed, duplex communication over TCP, WebSocket, and other byte stream transports, using one of the following interaction models:

- `Request-Response` —send one message and receive one back.
- `Request-Stream` —send one message and receive a stream of messages back.
- `Channel` —send streams of messages in both directions.
- `Fire-and-Forget` —send a one-way message.

Once the initial connection is made, the "client" vs "server" distinction is lost as both sides become symmetrical and each side can initiate one of the above interactions. This is why in the protocol calls the participating sides "requester" and "responder" while the above interactions are called "request streams" or simply "requests".

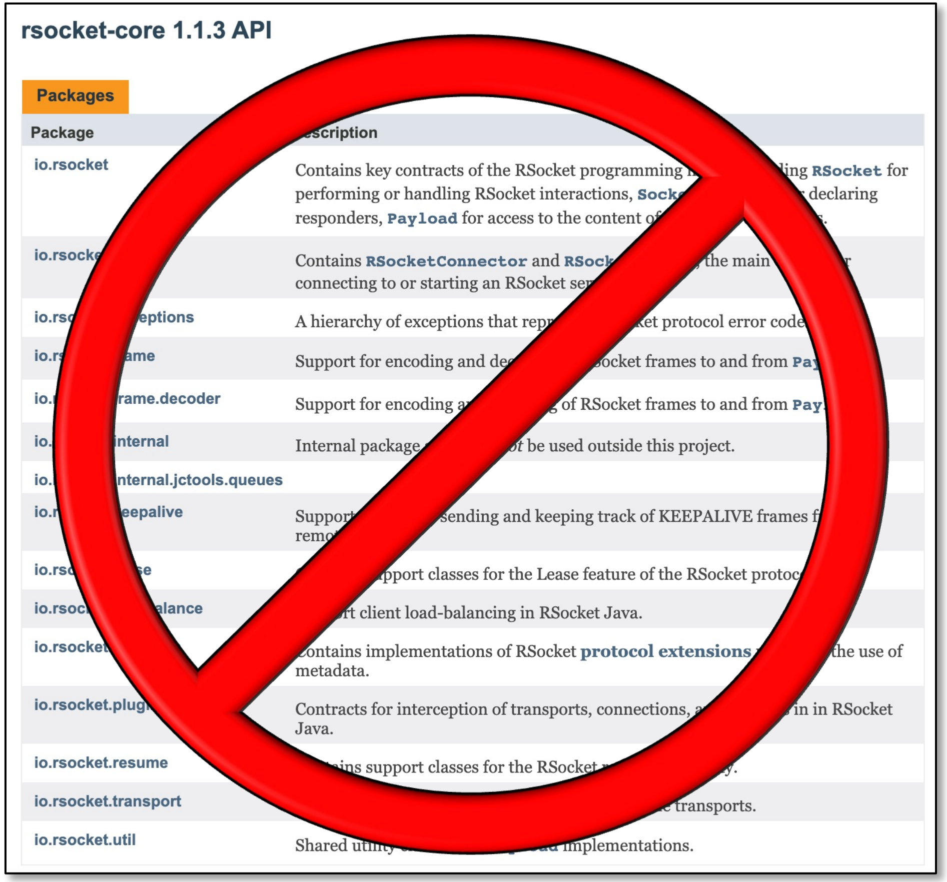
These are the key features and benefits of the RSocket protocol:

- Reactive Streams semantics across network boundary—for streaming requests such as `Request-Stream` and `Channel`, back pressure signals travel between requester and responder, allowing a requester to slow down a responder at the source, hence reducing reliance on network layer congestion control, and the need for buffering at the network level or at any level.
- Request throttling—this feature is named "Leasing" after the `LEASE` frame that can be sent from each end to limit the total number of requests allowed by other end for a given time. Leases are renewed periodically.
- Session resumption—this is designed for loss of connectivity and requires some state to be maintained. The state management is transparent for applications, and works well in combination with back pressure which can stop a producer when possible and reduce the amount of state required.
- Fragmentation and re-assembly of large messages.
- Keepalive (heartbeats).

See docs.spring.io/spring-framework/docs/current/reference/html/rsocket.html

Integrating the RSocket APIs with Spring

- RSocket can be used without any connection to Spring whatsoever
  - However, it's generally more effective to leverage Spring's integration of RSocket
  - Spring applications generally need not use (most of) the RSocket APIs directly



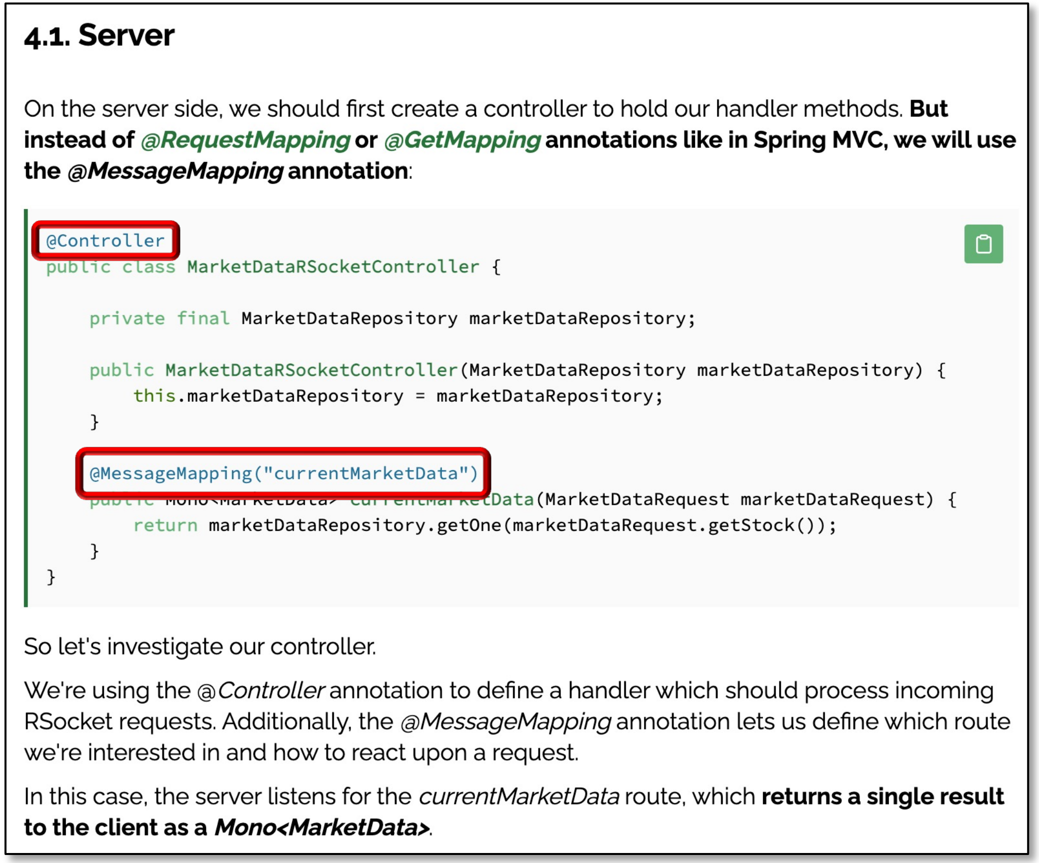
rsocket-core 1.1.3 API

Packages

Package	escription
io.rsocket	Contains key contracts of the RSocket programming ling RSocket for performing or handling RSocket interactions, Sock declaring responders, Payload for access to the content of s.
io.rsocke	Contains RSocketConnector and RSock , the main r connecting to or starting an RSocket ser
io.rs eptions	A hierarchy of exceptions that rep ket protocol error code
io.r ame	Support for encoding and de ocket frames to and from Pay
io. rame.decoder	Support for encoding a g of RSocket frames to and from Pay
io. internal	Internal package ot be used outside this project.
io. nternal.jctools.queues	
io.r eepalive	Support sending and keeping track of KEEPALIVE frames f remo
io.rs se	pport classes for the Lease feature of the RSocket protoc
io.rsoc alance	rt client load-balancing in RSocket Java.
io.rsocket	ontains implementations of RSocket protocol extensions the use of metadata.
io.rsocket.plug	Contracts for interception of transports, connections, a in in RSocket Java.
io.rsocket.resume	ins support classes for the RSocket y.
io.rsocket.transport	transports.
io.rsocket.util	Shared utility ad implementations.

Integrating the RSocket APIs with Spring

- Spring enables the integration of RSocket into a controller via various annotations



4.1. Server

On the server side, we should first create a controller to hold our handler methods. **But instead of *@RequestMapping* or *@GetMapping* annotations like in Spring MVC, we will use the *@MessageMapping* annotation**:

```
@Controller
public class MarketDataRSocketController {

    private final MarketDataRepository marketDataRepository;

    public MarketDataRSocketController(MarketDataRepository marketDataRepository) {
        this.marketDataRepository = marketDataRepository;
    }

    @MessageMapping("currentMarketData")
    public Mono<MarketData> currentMarketData(MarketDataRequest marketDataRequest) {
        return marketDataRepository.getOne(marketDataRequest.getStock());
    }
}
```

So let's investigate our controller.

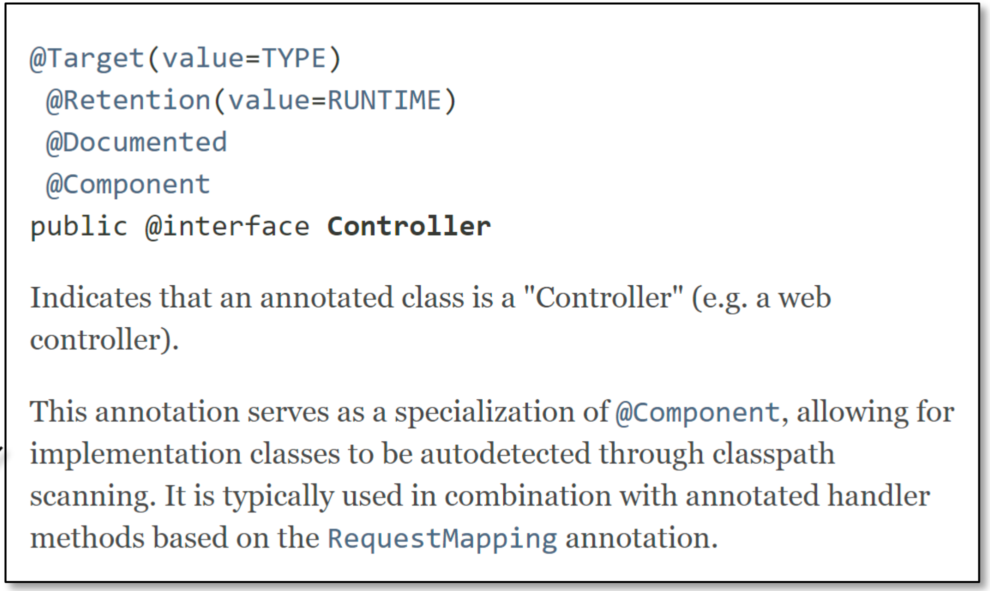
We're using the @*Controller* annotation to define a handler which should process incoming RSocket requests. Additionally, the *@MessageMapping* annotation lets us define which route we're interested in and how to react upon a request.

In this case, the server listens for the *currentMarketData* route, which **returns a single result to the client as a *Mono<MarketData>***.

See spring-framework/docs/current/reference/html/rsocket.html#rsocket-annot-responders

Integrating the RSocket APIs with Spring

- Spring enables the integration of RSocket into a controller via various annotations
  - @Controller
    - Enables the auto-detection of implementation classes via classpath scanning



```
@Target(value=TYPE)
 @Retention(value=RUNTIME)
 @Documented
 @Component
public @interface Controller
```

Indicates that an annotated class is a "Controller" (e.g. a web controller).

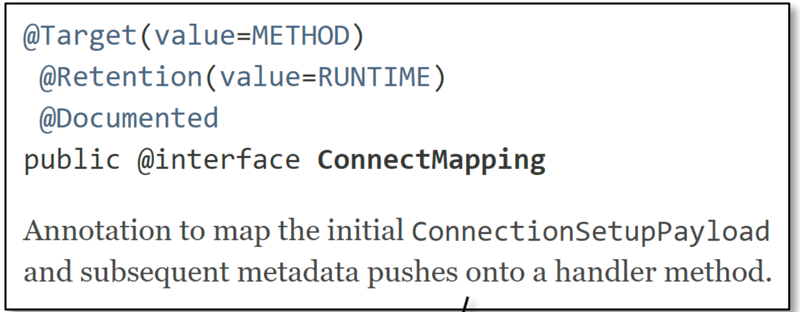
This annotation serves as a specialization of `@Component`, allowing for implementation classes to be autodetected through classpath scanning. It is typically used in combination with annotated handler methods based on the `RequestMapping` annotation.

*Note this is *not* @RestController!*

See www.baeldung.com/spring-controllers

Integrating the RSocket APIs with Spring

- Spring enables the integration of RSocket into a controller via various annotations
  - @Controller
  - @ConnectMapping
    - Handles connection-level events
      - e.g., setup & metadata push



```
@Target(value=METHOD)
 @Retention(value=RUNTIME)
 @Documented
public @interface ConnectMapping
```

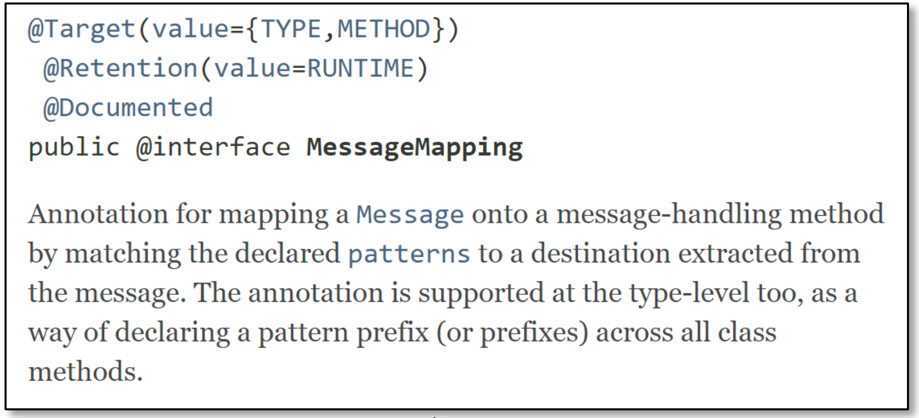
Annotation to map the initial `ConnectionSetupPayload` and subsequent metadata pushes onto a handler method.

Useful for authentication & other initialization operations

See [springframework/messaging/rsocket/annotation/ConnectMapping.html]()

Integrating the RSocket APIs with Spring

- Spring enables the integration of RSocket into a controller via various annotations
  - @Controller
  - @ConnectMapping
  - @MessageMapping
    - Maps a message to a message-handling method by matching the declared patterns to a destination extracted from the message



```
@Target(value={TYPE,METHOD})
 @Retention(value=RUNTIME)
 @Documented
public @interface MessageMapping
```

Annotation for mapping a `Message` onto a message-handling method by matching the declared `patterns` to a destination extracted from the message. The annotation is supported at the type-level too, as a way of declaring a pattern prefix (or prefixes) across all class methods.

Similar to the @GetMapping & @PostMapping annotations, but for RSocket messages instead

See springframework/messaging/handler/annotation/MessageMapping.html

Integrating the RSocket APIs with Spring

- RSocket endpoint handlers combine @Controller, @Message Mapping, & @ConnectMapping annotations

```
@Controller
class ZippyConnectController {
 ...
 @ConnectMapping(SERVER_CONNECT)
 void handleConnect(...) { ... }
 ...

@Controller
class ZippyMessageController {
 ...
 @MessageMapping(SUBSCRIBE)
 Mono<Subscription> subscribe
     (Mono<Subscription> request)
 { ... }
 ...
```

See [RSocket/ex1/src/main/java/zippyisms/controller/ZippyConnectController.java]

Integrating the RSocket APIs with Spring

- RSocket endpoint handlers combine @Controller, @Message Mapping, & @ConnectMapping annotations
  - Endpoints can handle connections

```
@Controller
class ZippyConnectController {
  ...
  @ConnectMapping(SERVER_CONNECT)
  void handleConnect(...) { ... }
  ...
```

Integrating the RSocket APIs with Spring

- RSocket endpoint handlers combine @Controller, @Message Mapping, & @ConnectMapping annotations
  - Endpoints can handle connections
    - Routes incoming connect requests to endpoint handlers

```
@Controller
class ZippyConnectController {
 ...
 @ConnectMapping(SERVER_CONNECT)
 void handleConnect(...) { ... }
 ...
```

Integrating the RSocket APIs with Spring

- RSocket endpoint handlers combine @Controller, @Message Mapping, & @ConnectMapping annotations
  - Endpoints can handle connections
  - Endpoints can also handle messages

```
@Controller
class ZippyMessageController {
 ...
 @MessageMapping(SUBSCRIBE)
 Mono<Subscription> subscribe
     (Mono<Subscription> request)
 { ... }

 @MessageMapping(GET_ALL_QUOTES)
 Flux<Quote> getAllQuotes
     (Mono<Subscription> request)
 { ... } ...
```

Integrating the RSocket APIs with Spring

- RSocket endpoint handlers combine @Controller, @Message Mapping, & @ConnectMapping annotations
  - Endpoints can handle connections
  - Endpoints can also handle messages
    - Routes incoming message requests to endpoint handlers

```
@Controller
class ZippyMessageController {
 ...
 @MessageMapping(SUBSCRIBE)
 Mono<Subscription> subscribe
     (Mono<Subscription> request)
 { ... }

 @MessageMapping(GET_ALL_QUOTES)
 Flux<Quote> getAllQuotes
     (Mono<Subscription> request)
 { ... } ...
```

Integrating the RSocket APIs with Spring

- RSocket endpoint handlers combine @Controller, @Message Mapping, & @ConnectMapping annotations
  - Endpoints can handle connections
  - Endpoints can also handle messages
  - Each endpoint can take a Mono or Flux parameter & can return a Mono or Flux result

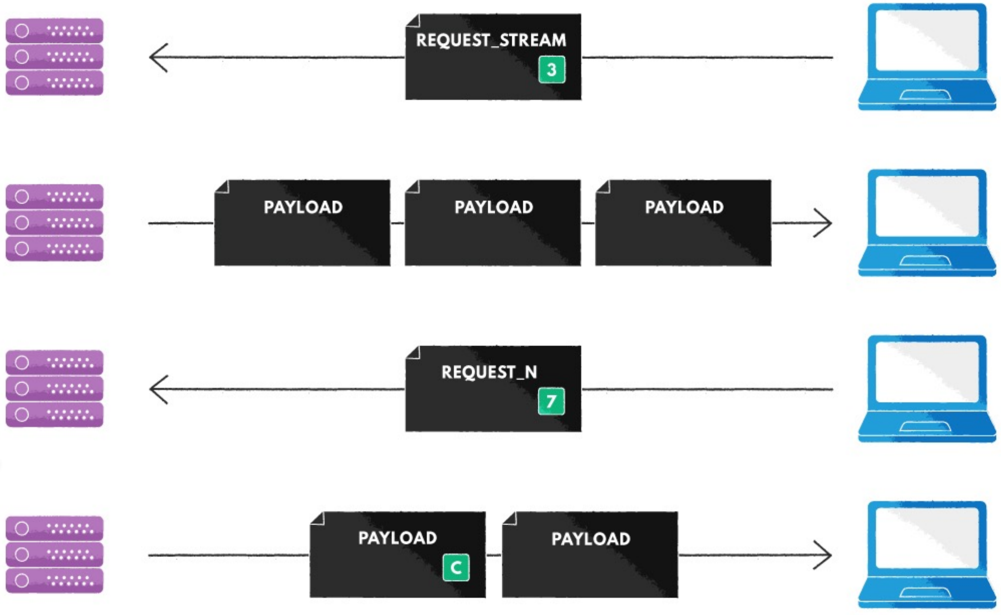
```
@Controller
class ZippyMessageController {
  ...
  @MessageMapping(SUBSCRIBE)
  Mono<Subscription> subscribe
      (Mono<Subscription> request)
  { ... }

  @MessageMapping(GET_ALL_QUOTES)
  Flux<Quote> getAllQuotes
      (Mono<Subscription> request)
  { ... } ...
```

It's also possible to use other Java reference types, including RSocketRequester

Integrating the RSocket APIs with Spring

- The use of Project Reactor Mono & Flux types enable client & server code to run reactively across host or process boundaries



Supports asynchronous message passing

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

Integrating the RSocket APIs with Spring

- Other Spring annotations are seamlessly integrated with RSocket

```
@Bean
public Mono<RSocketRequester>
getRSocketRequester() { ... }

...

@Autowired
private Mono<RSocketRequester>
  mZippyQuoteRequester;
```

Autowires the RSocketRequester with the client ZippyProxy class

End of Integrating the RSocket APIs in Spring