CHAPTER 17

Reactive Streams

QUESTIONS AND EXERCISES

1. What are reactive streams? What are pull and push models in reactive streams?

Answer:

Reactive Streams provide a standard for *asynchronous* stream processing with non-blocking backpressure. In the Reactive Streams model, the subscriber sends an asynchronous request to the publisher for `N` items. The publisher sends `N` or fewer items to the subscriber asynchronously .

In the push model, the publisher pushes items to the subscriber. In the pull model, the subscriber pulls items from the publisher. Reactive Streams dynamically switches between the pull model and the push models. It uses the pull model when the subscriber is slower and uses the push model when the subscriber is faster.

2. Describe the four components — publisher, subscriber, subscription, and processor — of reactive streams.

Answer:

A *publisher* is a producer of potentially an unbounded number of sequenced items. It publishes (or sends) items to its current subscribers based on the demands received from them.

A *subscriber* subscribes to a publisher to receive items.

A *subscription* represents a token of subscription of a subscriber to a publisher.

A *processor* represents a processing stage that acts as both a subscriber and a publisher.

3. List the fully qualified names of the four interfaces of the Reactive Streams API in Java?

Answer:

The four interfaces specified by the Reactive Streams Java API are included in the `java.util.concurrent.Flow` class as nested static interfaces. They are as follows:

- `Flow.Processor<T,R>`
- `Flow.Publisher<T>`
- `Flow.Subscriber<T>`
- `Flow.Subscription`

4. Is the following statement true or false?

   The Reactive Streams API supports asynchronous processing of stream of data.

Answer:

True

5. Which method on the subscriber is called when the subscriber's subscription with a publisher succeeds?

Answer:

When the subscription is successful, the publisher asynchronously calls the `onSubscribe (Flow.Subscription subscription)` method of the subscriber.

6. Which method on the subscriber is called when the subscriber's subscription with a publisher fails?

Answer:

When the attempt to subscribe to a publisher by a subscriber fails, the `onError(Throwable throwable)` method of the subscriber is called with an `IllegalStateException` and the publisher-subscriber interaction ends.

7. How does a subscriber requests 200 items from a publisher?

Answer:

Upon a successful subscription, a subscriber receives a `Flow.Subscription` that represents a token of the subscriber's subscription with a publisher. To request 200 items from the publisher, the subscriber calls the `request(200)` method of the `Flow.Subscription`.

8. What method on the subscriber is called when it receives an item from its publisher?

Answer:

The `onNext(T item)` method.

9. How does a subscriber cancel its subscription with a publisher? Is it possible for a subscriber to receive more items from its publisher after it cancels its subscription?

Answer:

The subscriber can cancel its subscription by calling the `cancel()` method of its `Flow.Subscription`. Once a subscription is cancelled, the publisher-subscriber interaction ends. However, it is possible for the subscriber to receive items after canceling its subscription if there were pending requests before requesting the cancellation.

10. What is the fully qualified name of the implementation class for the `java.util.concurrent.Flow.Publisher<T>` interface?

Answer:

`java.util.concurrent.SubmissionPublisher<T>` is the implementation class for the `java.util.concurrent.Flow.Publisher<T>` interface.