Querying Using Procedures

Module Objectives

In this module, you will cover:

- Using Procedures for querying Reactor
- Creating a Procedure
- Implementing responses
- Handling errors

Querying Reactor

Procedures let you query the Reactor, its DataSets and retrieve results

Procedures:

- Allow synchronous calls into the Reactor from an external system
- Perform server-side processing on-demand
- Are similar to a stored procedure in a traditional database

Procedures are typically used to post-process data at query time:

- Filtering
- Aggregating
- Joins over multiple DataSets

Procedure Operations

A Procedure can perform all the same operations as a Flowlet

- With the same consistency and durability guarantees
- Deployed into the same pool of application containers as Flows
- Can run multiple instances to increase the throughput of requests

A Procedure implements and exposes a very simple API:

- A method name (String) and
- Arguments (map of Strings)

This implementation is then bound to a REST endpoint and can be called from any external system

Creating and Implementing a Procedure

To create a Procedure:

- Implement the `Procedure` interface, or
- More conveniently, extend the `AbstractProcedure` class

Configuration and Initialization

- Similar to a Flowlet
- Instead of a process method you'll define a handler method

Response

- Upon external call, the handler method receives the request and sends a response
- The most generic way to send a response is to obtain a `Writer` and stream out the response as bytes

Example Procedure

```
import static com.continuuity.api.procedure.ProcedureResponse.Code.SUCCESS;
...
class HelloWorld extends AbstractProcedure {

  @Handle("hello")
  public void wave(ProcedureRequest request,
                   ProcedureResponder responder) throws IOException {
    String hello = "Hello " + request.getArgument("who");
    ProcedureResponse.Writer writer =
      responder.stream(new ProcedureResponse(SUCCESS));
    writer.write(ByteBuffer.wrap(hello.getBytes())).close();
  }
}
```

- This uses the most generic way to create the response, which allows you to send arbitrary byte content as the response body
- Make sure to close the `Writer` when you are done

Responding with JSON

In many cases, you will actually respond with JSON

Reactor `ProcedureResponder` has convenience methods for returning JSON maps:

```
// Return a JSON map
Map<String, Object> results = new TreeMap<String, Object>();
results.put("totalWords", totalWords);
results.put("uniqueWords", uniqueWords);
results.put("averageLength", averageLength);
responder.sendJson(results);
```

Responding to Errors

Convenience method to respond with an error message:

```
@Handle("getCount")
public void getCount(ProcedureRequest request, ProcedureResponder responder)
                     throws IOException, InterruptedException{
  String word = request.getArgument("word");
  if (word == null) {
    responder.error(Code.CLIENT_ERROR,
                    "Method 'getCount' requires argument 'word'");
    return;
  }
```

Module Summary

You should now be able to:

- Using Procedures for querying Reactor
- Create a Procedure
- Implement a response
- Handle errors

Module Completed