Spring Data Key-Value - Reference Documentation

1.0.0.M1

Costin Leau (SpringSource), Jon Brisbin (NPC International, Inc.)

Preface

The Spring Data Key-Value project applies core Spring concepts to the development of solutions using a key-value style data store. We provide a "template" as a high-level abstraction for sending and receiving messages. You will notice similarities to the JDBC support in the Spring Framework.

Part I. Introduction

This document is the reference guide for Spring Data - Key Value Support. It explains Key Value module concepts and semantics and the syntax for various stores namespaces.

For an introduction to key value stores or Spring, or Spring Data examples, please refer to Chapter 3, *Getting Started* - this documentation refers only to Spring Data Key Value Support and assumes the user is familiar with the key value storages and Spring concepts.

Chapter 1. Why Spring Data - Key Value?

The Spring Framework is the leading full-stack Java/JEE application framework. It provides a lightweight container and a non-invasive programming model enabled by the use of dependency injection, AOP, and portable service abstractions.

NoSQL storages provide an alternative to classical RDBMS for horizontal scalability and speed. In terms of implementation, Key Value stores represent one of the largest (and oldest) member in the NoSQL space.

The Spring Data Key Value (or SDKV) framework makes it easy to write Spring applications that use a Key Value store by eliminating the redundant tasks and boiler place code required for interacting with the store through Spring's excellent infrastructure support.

Chapter 2. Requirements

Spring Data Key Value 1.x binaries requires JDK level 6.0 and above, and Spring Framework 3.0.x and above.

In terms of key value stores, Redis 2.0.x and Riak 0.13 are required.

Chapter 3. Getting Started

Learning a new framework is not always straight forward. In this section, we (the Spring Data team) tried to provide, what we think is, an easy to follow guide for starting with Spring Data Key Value module. Of course, feel free to create your own learning 'path' as you see fit and, if possible, please report back any improvements to the documentation that can help others.

3.1. First Steps

As explained in Chapter 1, *Why Spring Data - Key Value?*, Spring Data Key Value (SDKV) provides integration between Spring framework and key value (KV) stores. Thus, it is important to become acquainted with both of these frameworks (storages or environments depending on how you want to name them). Throughout the SDKV documentation, each section provides links to resources relevant however, it is best to become familiar with these topics beforehand.

3.1.1. Knowing Spring

Spring Data uses heavily Spring framework's core functionality, such as the IoC container, resource abstract or AOP infrastructure. While it is not important to know the Spring APIs, understanding the concepts behind them is. At a minimum, the idea behind IoC should be familiar. These being said, the more knowledge one has about the Spring, the faster she will pick Spring Data Key Value. Besides the very comprehensive (and sometimes disarming) documentation that explains in detail the Spring Framework, there are a lot of articles, blog entries and books on the matter - take a look at the Spring framework home page for more information. In general, this should be the starting point for developers wanting to try Spring DKV.

3.1.2. Knowing NoSQL and Key Value stores

NoSQL stores have taken the storage world by storm. It is a vast domain with a plethora of solutions, terms and patterns (to make things worth even the term itself has multiple meanings). While some of the principles are common, it is crucial that the user is familiar to some degree with the stores supported by SDKV. The best way to get acquainted to this solutions is to read their documentation and follow their examples - it usually doesn't take more then 5-10 minutes to go through them and if you are coming from an RDMBS-only background many times these exercises can be an eye opener.

3.1.3. Trying Out The Samples

Unfortunately the SDKV project is very young and there are no samples available yet. However we are working on them and plan to make them available as soon as possible. In the meantime however, one can use our test suite as a code example (assuming the documentation is not enough) - we provide extensive integration tests for our code base.

3.2. Need Help?

If you encounter issues or you are just looking for an advice, feel free to use one of the links below:

3.2.1. Community Support

The Spring Data forum is a message board for all Spring Data (not just Key Value) users to share information and help each other. Note that registration is needed *only* for posting.

3.2.2. Professional Support

Professional, from-the-source support, with guaranteed response time, is available from SpringSource, the company behind Spring Data and Spring.

3.3. Following Development

For information on the Spring Data source code repository, nightly builds and snapshot artifacts please see the Spring Data home page.

You can help make Spring Data best serve the needs of the Spring community by interacting with developers through the Spring Community forums.

If you encounter a bug or want to suggest an improvement, please create a ticket on the Spring Data issue tracker.

To stay up to date with the latest news and announcements in the Spring eco system, subscribe to the Spring Community Portal.

Lastly, you can follow the SpringSource Data blog or the project team on Twitter (Costin)

Part II. Reference Documentation

Document structure

This part of the reference documentation explains the core functionality offered by Spring Data Key Value.

Chapter 4. Riak Support

Riak is a Key/Value datastore that supports Internet-scale data replication for high performance and high availability. Spring Data Key/Value (SDKV) provides access to the Riak datastore over the HTTP REST API using a built-in driver based on Spring 3.0's `RestTemplate`. In addition to making Key/Value datastore access easier from Java, the `RiakTemplate` has been designed, from the ground up, to be used from alternative JVM languages like Groovy or JRuby.

Since the SDKV support for Riak uses the stateless REST API, there are no connection factories to manage or other stateful objects to keep tabs on. The helper you'll spend the most time working with is likely the thread-safe `RiakTemplate` or `RiakKeyValueTemplate`. Your choice of which to use will depend on how you want to manage buckets and keys. SDKV supports two ways to interact with Riak. If you want to use the convention you're likely already familiar with, namely of storing an entry with a given key in a "bucket" by passing the bucket and key name separately, you'll want to use the `RiakTemplate`. If you want to use a single object to represent your bucket and key pair, you can use the `RiakKeyValueTemplate`. It supports a key object that is encoded using one of several different methods:

- *Using a `String`* - You can concatenate two strings, separated by a colon: "mybucket:mykey".
- *Using a `BucketKeyPair`* - You can pass an instance of `BucketKeyPair`, like `SimpleBucketKeyPair`.
- *Using a `Map`* - You can pass a `Map` with keys for "bucket" and "key".

4.1. Configuring the `RiakTemplate`

This is likely the easiest path to using SDKV for Riak, as the bucket and key are passed separately. The examples that follow will assume you're using this version of the the template.

There are only two options you need to set to specify the Riak server to use in your `RiakTemplate` object: "defaultUri" and "mapReduceUri". Encoded with the URI should be placeholders for the bucket and the key, which will be filled in by the `RestTemplate` when the request is made.

Important

You can also turn the internal, ETag-based object cache off by setting `useCache="false"`. It's generally recommended, however, to leave the internal cache on as the ETag matching will pick up any changes made to the entry on the Riak side and your application will benefit from greatly-increased performance for often-requested objects.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="riakTemplate" class="org.springframework.data.keyvalue.riak.core.RiakTemplate"
          p:defaultUri="http://localhost:8098/riak/{bucket}/{key}"
          p:mapReduceUri="http://localhost:8098/mapred"
          p:useCache="true"/>

</beans>
```

4.1.1. Advanced Template Configuration

There are a couple additional properties on the `RiakTemplate` that can be changed from their defaults. If you want to specify your own ConversionService to use when converting objects for storage inside Riak, then set it on the "conversionService" property:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="conversionService" class="com.mycompany.convert.MyConversionService"/>
    <bean id="riakTemplate" class="org.springframework.data.keyvalue.riak.core.RiakTemplate"
          p:defaultUri="http://localhost:8098/riak/{bucket}/{key}"
          p:mapReduceUri="http://localhost:8098/mapred"
          p:conversionService-ref="conversionService"/>

</beans>
```

Depending on the application, it might be useful to set default Quality-of-Service parameters. In Riak paralance, these are the "dw", "w", and "r" parameters. They can be set to an integer representing the number of vnodes that need to report having received the data before declaring the operation a success, or the string "one", "all", or (the default) "quorum". These values can be overridden by passing a different set of `QosParameters` to the set/get operation you're performing.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="qos" class="org.springframework.data.keyvalue.riak.core.RiakQosParameters"
          p:durableWriteThreshold="all"
          p:writeThreshold="all"/>
    <bean id="riakTemplate" class="org.springframework.data.keyvalue.riak.core.RiakTemplate"
          p:defaultUri="http://localhost:8098/riak/{bucket}/{key}"
          p:mapReduceUri="http://localhost:8098/mapred"
          p:defaultQosParameters-ref="qos"/>

</beans>
```

It might also be necessary to replace the default `ExecutorService` (by default a cached ThreadPoolExecutor) with an executor you've explicitly configured. Set your `ExecutorService` on the template's "executorService" property.

4.2. Working with Objects using the `RiakTemplate`

One of the primary goals of the SDKV project is to make accessing Key/Value stores easier for the developer by taking away the mundane tasks of basic IO, buffering, type conversion, exception handling, and sundry other logistical concerns so the developer can focus on creating great applications. SDKV for Riak works toward this goal by making basic persistence and data access as easy as using a `Map`.

4.2.1. Saving data into Riak

To store data in Riak, use one of the six different `set` methods:

```
import org.springframework.data.keyvalue.riak.core.RiakTemplate;
```

```
public class Example {

    @Autowired
    RiakTemplate riak;

    public void setData(String bucket, String key, String data) throws Exception {
        riak.set(bucket, key, data); // Set as Content-Type: text/plain
        riak.setAsBytes(bucket, key, data.getBytes()); // Set as Content-Type: application/octet-stream
    }

    public void setData(String bucket, String key, MyPojo data) throws Exception {
        riak.set(bucket, key, data); // Converted to JSON automatically, Content-Type: application/json
    }

}
```

Additionally, there is a `setWithMetaData` method that takes a `Map` of metadata that will be set as the outgoing HTTP headers. To set custom metadata, your key should be prefixed with `X-Riak-Meta-` e.g. `X-Riak-Meta-Custom-Header`.

4.2.2. Retrieving data from Riak

Retrieving data from Riak is just as easy. There are actually 13 different `get` methods on `RiakTemplate` that give the developer a wide range options for accessing and converting your data.

Assuming you've stored a POJO using an appropriate `set` method, you can retrieve that object from Riak using a `get`:

```
import org.springframework.data.keyvalue.riak.core.RiakTemplate;

  public class Example {

      @Autowired
      RiakTemplate riak;

      public void getData(String bucket, String key) throws Exception {
          // What you get depends on Content-Type.
          // application/json=Map, text/plain=String, etc...
          Object o = riak.get(bucket, key);

          // If your entry is Content-Type: application/json...
          // It will automatically be converted when retrieved.
          MyPojo s = riak.getAsType(bucket, key, MyPojo.class);

          // If your entry is Content-Type: application/octet-stream,
          // you can access the raw bytes.
          byte[] b = riak.getAsBytes(bucket, key); // No conversion at all
      }

  }
```

4.3. Working with streams

SDKV for Riak includes a couple of useful helper objects to make reading and writing plain text or binary data in Riak really easy. If you want to store a file in Riak, then you can create a `RiakOutputStream` and simply write your data to it (making sure to call the "flush" method, which actually sends the data to Riak).

```
import org.springframework.data.keyvalue.riak.core.RiakTemplate;
import org.springframework.data.keyvalue.riak.core.io.RiakOutputStream;

public class Example {
    @Autowired
    RiakTemplate riak;

    public void writeToRiak(String bucket, String key, String data) throws Exception {
        OutputStream out = new RiakOutputStream(riak, bucket, key);
        try {
            out.write(data.getBytes());
        } finally {
            out.flush();
            out.close();
        }
    }

}
```

Reading data from Riak is similarly easy. SDKV provides a `java.io.File` subclass that represents a resource in Riak. There's also a Spring IO Resource abstraction called `RiakResource` that can be used anywhere a `Resource` is required. There's also an `InputStream` implementation called `RiakInputStream`.

```
import org.springframework.data.keyvalue.riak.core.RiakTemplate;
import org.springframework.data.keyvalue.riak.core.io.RiakInputStream;

public class Example {

    @Autowired
    RiakTemplate riak;

    public String readFromRiak(String bucket, String key) throws Exception {
        InputStream in = new RiakInputStream(riak, bucket, key);
        String data;
        ...read data and work with it...
        return data;
    }

}
```