GETTING STARTED

Messaging with Redis（与redis通信）

This guide walks you through the process of using Spring Data Redis to publish and subscribe to messages sent via Redis.

What you’ll build

You’ll build an application that uses StringRedisTemplate to publish a string message and has a [POJO](http://spring.io/understanding/POJO) subscribe for it using MessageListenerAdapter.

|  |  |
| --- | --- |
|  | It may sound strange to be using Spring Data Redis as the means to publish messages, but as you’ll discover, Redis not only provides a NoSQL data store, but a messaging system as well. |

What you’ll need

* About 15 minutes
* A favorite text editor or IDE
* [JDK 1.8](http://www.oracle.com/technetwork/java/javase/downloads/index.html) or later
* [Gradle 2.3+](http://www.gradle.org/downloads) or [Maven 3.0+](https://maven.apache.org/download.cgi)
* You can also import the code straight into your IDE:
  + [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts)
  + [IntelliJ IDEA](http://spring.io/guides/gs/intellij-idea/)
    - Redis server (installation instructions below)

How to complete this guide

Like most Spring [Getting Started guides](http://spring.io/guides), you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To **start from scratch**, move on to [Build with Gradle](http://spring.io/guides/gs/messaging-redis/#scratch).

To **skip the basics**, do the following:

* [Download](https://github.com/spring-guides/gs-messaging-redis/archive/master.zip) and unzip the source repository for this guide, or clone it using [Git](http://spring.io/understanding/Git): git clone <https://github.com/spring-guides/gs-messaging-redis.git>
* cd into gs-messaging-redis/initial
* Jump ahead to [Create a Redis message receiver](http://spring.io/guides/gs/messaging-redis/#initial).

**When you’re finished**, you can check your results against the code in gs-messaging-redis/complete.

Build with Gradle

First you set up a basic build script. You can use any build system you like when building apps with Spring, but the code you need to work with [Gradle](http://gradle.org/) and [Maven](https://maven.apache.org/) is included here. If you’re not familiar with either, refer to [Building Java Projects with Gradle](http://spring.io/guides/gs/gradle) or [Building Java Projects with Maven](http://spring.io/guides/gs/maven).

### Create the directory structure

In a project directory of your choosing, create the following subdirectory structure; for example, with mkdir -p src/main/java/hello on \*nix systems:

└── src

└── main

└── java

└── hello

### Create a Gradle build file

Below is the [initial Gradle build file](https://github.com/spring-guides/gs-messaging-redis/blob/master/initial/build.gradle).

build.gradle

buildscript {

repositories {

mavenCentral()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:1.5.2.RELEASE")

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'idea'

apply plugin: 'org.springframework.boot'

jar {

baseName = 'gs-messaging-redis'

version = '0.1.0'

}

repositories {

mavenCentral()

}

sourceCompatibility = 1.8

targetCompatibility = 1.8

dependencies {

compile("org.springframework.boot:spring-boot-starter")

compile("org.springframework.boot:spring-boot-starter-data-redis")

testCompile("junit:junit")

}

The [Spring Boot gradle plugin](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-tools/spring-boot-gradle-plugin) provides many convenient features:

* It collects all the jars on the classpath and builds a single, runnable "über-jar", which makes it more convenient to execute and transport your service.
* It searches for the public static void main() method to flag as a runnable class.
* It provides a built-in dependency resolver that sets the version number to match [Spring Boot dependencies](https://github.com/spring-projects/spring-boot/blob/master/spring-boot-dependencies/pom.xml). You can override any version you wish, but it will default to Boot’s chosen set of versions.

Build with Maven

Build with your IDE

Standing up a Redis server

Before you can build a messaging application, you need to set up the server that will handle receiving and sending messages.

Redis is an open source, BSD-licensed, key-value data store that also comes with a messaging system. The server is freely available at <http://redis.io/download>. You can download it manually, or if you use a Mac with homebrew:

brew install redis

Once you unpack Redis, you can launch it with default settings.

redis-server

You should see a message like this:

[35142] 01 May 14:36:28.939 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf

[35142] 01 May 14:36:28.940 \* Max number of open files set to 10032

\_.\_

\_.-``\_\_ ''-.\_

\_.-`` `. `\_. ''-.\_ Redis 2.6.12 (00000000/0) 64 bit

.-`` .-```. ```\/ \_.,\_ ''-.\_

( ' , .-` | `, ) Running in stand alone mode

|`-.\_`-...-` \_\_...-.``-.\_|'` \_.-'| Port: 6379

| `-.\_ `.\_ / \_.-' | PID: 35142

`-.\_ `-.\_ `-./ \_.-' \_.-'

|`-.\_`-.\_ `-.\_\_.-' \_.-'\_.-'|

| `-.\_`-.\_ \_.-'\_.-' | http://redis.io

`-.\_ `-.\_`-.\_\_.-'\_.-' \_.-'

|`-.\_`-.\_ `-.\_\_.-' \_.-'\_.-'|

| `-.\_`-.\_ \_.-'\_.-' |

`-.\_ `-.\_`-.\_\_.-'\_.-' \_.-'

`-.\_ `-.\_\_.-' \_.-'

`-.\_ \_.-'

`-.\_\_.-'

[35142] 01 May 14:36:28.941 # Server started, Redis version 2.6.12

[35142] 01 May 14:36:28.941 \* The server is now ready to accept connections on port 6379

Create a Redis message receiver

In any messaging-based application, there are message publishers and messaging receivers. To create the message receiver, implement a receiver with a method to respond to messages:

src/main/java/hello/Receiver.java

package hello;

import java.util.concurrent.CountDownLatch;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

public class Receiver {

private static final Logger LOGGER = LoggerFactory.getLogger(Receiver.class);

private CountDownLatch latch;

@Autowired

public Receiver(CountDownLatch latch) {

this.latch = latch;

}

public void receiveMessage(String message) {

LOGGER.info("Received <" + message + ">");

latch.countDown();

}

}

The Receiver is a simple POJO that defines a method for receiving messages. As you’ll see when you register the Receiver as a message listener, you can name the message-handling method whatever you want.

|  |  |
| --- | --- |
|  | For demonstration purposes, it is autowired（自动装配） by its constructor with a countdown latch. That way, it can signal when it has received a message. |

Register the listener and send a message

Spring Data Redis provides all the components you need to send and receive messages with Redis. Specifically, you need to configure:

* A connection factory
* A message listener container
* A Redis template

You’ll use the Redis template to send messages and you will register the Receiver with the message listener container so that it will receive messages. The connection factory drives both the template and the message listener container, enabling them to connect to the Redis server.

This example uses Spring Boot’s default RedisConnectionFactory, an instance of JedisConnectionFactory which is based on the [Jedis](https://github.com/xetorthio/jedis) Redis library. The connection factory is injected into both the message listener container and the Redis template.

src/main/java/hello/Application.java

package hello;

import java.util.concurrent.CountDownLatch;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.Bean;

import org.springframework.data.redis.connection.RedisConnectionFactory;

import org.springframework.data.redis.core.StringRedisTemplate;

import org.springframework.data.redis.listener.PatternTopic;

import org.springframework.data.redis.listener.RedisMessageListenerContainer;

import org.springframework.data.redis.listener.adapter.MessageListenerAdapter;

@SpringBootApplication

public class Application {

private static final Logger LOGGER = LoggerFactory.getLogger(Application.class);

@Bean

RedisMessageListenerContainer container(RedisConnectionFactory connectionFactory,

MessageListenerAdapter listenerAdapter) {

RedisMessageListenerContainer container = new RedisMessageListenerContainer();

container.setConnectionFactory(connectionFactory);

container.addMessageListener(listenerAdapter, new PatternTopic("chat"));

return container;

}

@Bean

MessageListenerAdapter listenerAdapter(Receiver receiver) {

return new MessageListenerAdapter(receiver, "receiveMessage");

}

@Bean

Receiver receiver(CountDownLatch latch) {

return new Receiver(latch);

}

@Bean

CountDownLatch latch() {

return new CountDownLatch(1);

}

@Bean

StringRedisTemplate template(RedisConnectionFactory connectionFactory) {

return new StringRedisTemplate(connectionFactory);

}

public static void main(String[] args) throws InterruptedException {

ApplicationContext ctx = SpringApplication.run(Application.class, args);

StringRedisTemplate template = ctx.getBean(StringRedisTemplate.class);

CountDownLatch latch = ctx.getBean(CountDownLatch.class);

LOGGER.info("Sending message...");

template.convertAndSend("chat", "Hello from Redis!");

latch.await();

System.exit(0);

}

}

The bean defined in the listenerAdapter method is registered as a message listener in the message listener container defined in container and will listen for messages on the "chat" topic. Because the Receiver class is a POJO, it needs to be wrapped in a message listener adapter that implements the MessageListener interface required by addMessageListener(). The message listener adapter is also configured to call the receiveMessage() method on Receiver when a message arrives.

The connection factory and message listener container beans are all you need to listen for messages. To send a message you also need a Redis template. Here, it is a bean configured as a StringRedisTemplate, an implementation of RedisTemplate that is focused on the common use of Redis where both keys and values are `String`s.

The main() method kicks everything off by creating a Spring application context. The application context then starts the message listener container, and the message listener container bean starts listening for messages. The main() method then retrieves the StringRedisTemplate bean from the application context and uses it to send a "Hello from Redis!" message on the "chat" topic. Finally, it closes the Spring application context and the application ends.

Build an executable JAR

You can run the application from the command line with Gradle or Maven. Or you can build a single executable JAR file that contains all the necessary dependencies, classes, and resources, and run that. This makes it easy to ship, version, and deploy the service as an application throughout the development lifecycle, across different environments, and so forth.

If you are using Gradle, you can run the application using ./gradlew bootRun. Or you can build the JAR file using ./gradlew build. Then you can run the JAR file:

java -jar build/libs/gs-messaging-redis-0.1.0.jar

If you are using Maven, you can run the application using ./mvnw spring-boot:run. Or you can build the JAR file with ./mvnw clean package. Then you can run the JAR file:

java -jar target/gs-messaging-redis-0.1.0.jar

|  |  |
| --- | --- |
|  | The procedure above will create a runnable JAR. You can also opt to [build a classic WAR file](http://spring.io/guides/gs/convert-jar-to-war/) instead. |

You should see the following output:

. \_\_\_\_ \_ \_\_ \_ \_

/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \

( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )

' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /

=========|\_|==============|\_\_\_/=/\_/\_/\_/

:: Spring Boot :: (v1.5.2.RELEASE)

2014-04-18 08:03:34.032 INFO 47002 --- [ main] hello.Application : Starting Application on retina with PID 47002 (/Users/gturnquist/src/spring-guides/gs-messaging-redis/complete/target/classes started by gturnquist)

2014-04-18 08:03:34.062 INFO 47002 --- [ main] s.c.a.AnnotationConfigApplicationContext : Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@7a53c84a: startup date [Fri Apr 18 08:03:34 CDT 2014]; root of context hierarchy

2014-04-18 08:03:34.326 INFO 47002 --- [ main] o.s.c.support.DefaultLifecycleProcessor : Starting beans in phase 2147483647

2014-04-18 08:03:34.357 INFO 47002 --- [ main] hello.Application : Started Application in 0.605 seconds (JVM running for 0.899)

2014-04-18 08:03:34.357 INFO 47002 --- [ main] hello.Application : Sending message...

2014-04-18 08:03:34.370 INFO 47002 --- [ container-2] hello.Receiver : Received <Hello from Redis!>

2014-04-18 08:03:34.379 INFO 47002 --- [ Thread-1] s.c.a.AnnotationConfigApplicationContext : Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@7a53c84a: startup date [Fri Apr 18 08:03:34 CDT 2014]; root of context hierarchy

2014-04-18 08:03:34.380 INFO 47002 --- [ Thread-1] o.s.c.support.DefaultLifecycleProcessor : Stopping beans in phase 2147483647

Summary

Congratulations! You’ve just developed a simple publish-and-subscribe application with Spring and Redis.

|  |  |
| --- | --- |
|  | [Redis support](http://gopivotal.com/products/redis) is available. |

Want to write a new guide or contribute to an existing one? Check out our [contribution guidelines](https://github.com/spring-guides/getting-started-guides/wiki).

|  |  |
| --- | --- |
|  | All guides are released with an ASLv2 license for the code, and an [Attribution, NoDerivatives creative commons license](https://creativecommons.org/licenses/by-nd/3.0/) for the writing. |