Spring Integration for VMware GemFire Documentation

Spring Integration for VMware GemFire 1.0

vmware by Broadcom

You can find the most up-to-date technical documentation on the VMware by Broadcom website at:

https://docs.vmware.com/

VMware by Broadcom
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

Spring Integration for VMware GemFire Documentation

Spring Integration for VMware GemFire is a Spring Integration project that enables users to use VMware GemFire within Spring Integration projects. Spring Integration for VMware GemFire provides Inbound and Outbound Channel Adapters, allowing Spring Integration flows to subscribe and publish data from/to VMware GemFire.

Requirements

Spring Integration for VMware GemFire 5.5 requires Java 8.0+, Spring Framework 5.3+,

Spring Integration for VMware GemFire 6.1 requires Java 17+, Spring Framework 6+,

VMware GemFire 9.15 or later.

Spring Integration for VMware GemFire supports:

- Spring Framework 5.3 or later
- Spring Integration 5.5 or later
- VMware GemFire 9.15 or later

Release Notes

This topic contains the release notes for Spring Data for VMware GemFire.

Spring Integration 6.1 & GemFire 10.0

1.0.0

- Initial release of Spring Integration For VMware GemFire, for Spring Integration 6.1 and VMware GemFire 10.0

Spring Integration 5.5 & GemFire 10.0

1.0.0

- Initial release of Spring Integration For VMware GemFire, for Spring Integration 5.5 and VMware GemFire 10.0

Spring Integration 5.5 & GemFire 9.15

1.0.0

- Initial release of Spring Integration For VMware GemFire, for Spring Integration 5.5 and VMware GemFire 9.15

Compatibility and Versions

This topic list Spring Integration for VMware GemFire compatibility and versions.

Compatibility

Spring Integration for VMware GemFire Artifact	Latest Versions	Compatible GemFire Versions	Compatible Spring Data Versions	Compatible Spring Framework Versions
spring-integration-5.5-gemfire-9.15	1.0.0	9.15.x	spring-data-2.7-gemfire-9.15:1.1.x	5.3.x
spring-integration-5.5-gemfire-10.0	1.0.0	10.0.x	spring-data-2.7-gemfire-10.0:1.0.x	5.3.x
spring-integration-6.1-gemfire-10.0	1.0.0	10.0.x	spring-data-3.1-gemfire-10.0:1.0.x	6.0.x

Getting Started

This topic explains how to download Spring Integration for VMware GemFire libraries to a project.

The Spring Integration for VMware GemFire libraries are available from the Pivotal Commercial Maven Repository. Access to the Pivotal Commercial Maven Repository requires a one-time registration step to create an account.

Add the Pivotal Commercial Maven Repository

1. In a browser, navigate to the Pivotal Commercial Maven Repository.
2. Click the **Create Account** link.
3. Complete the information in the registration page.
4. Click **Register**.
5. After registering, you will receive a confirmation email. Follow the instruction in this email to activate your account.
6. After account activation, log in to the Pivotal Commercial Maven Repository to access the configuration information found in gemfire-release-repo.
7. Add the repository to your project:
    - **Maven**: Add the following block to the `pom.xml` file:

```
<repository>
  <id>gemfire-release-repo</id>
  <name>Pivotal GemFire Release Repository</name>
  <url>https://commercial-repo.pivotal.io/data3/gemfire-release-repo/gemf
ire</url>
</repository>
```

    - **Gradle**: Add the following block to the `repositories` section of the `build.gradle` file:

```
repositories {
  mavenCentral()
  maven {
      credentials {
          username "$pivotalCommercialMavenRepoUsername"
          password "$pivotalCommercialMavenRepoPassword"
      }
      url = uri("https://commercial-repo.pivotal.io/data3/gemfire-release
-repo/gemfire")
  }
}
```

8. Add your Pivotal Commercial Maven Repository credentials.
    - **Maven**: Add the following to the `.m2/settings.xml` file. Replace `MY-USERNAME@example` and `MY-DECRYPTED-PASSWORD` with your Pivotal Commercial Maven Repository credentials.

      ```
      <settings>
        <servers>
            <server>
                <id>gemfire-release-repo</id>
                <username>MY-USERNAME@example.com</username>
                <password>MY-DECRYPTED-PASSWORD</password>
            </server>
        </servers>
      </settings>
      ```

    - **Gradle**: Add the following to the local (`.gradle/gradle.properties`) or project `gradle.properties` file. Replace `MY-USERNAME@example` and `MY-DECRYPTED-PASSWORD` with your Pivotal Commercial Maven Repository credentials.

      ```
      pivotalCommercialMavenRepoUsername=MY-USERNAME@example.com
      pivotalCommercialMavenRepoPassword=MY-DECRYPTED-PASSWORD
      ```

Add Libraries to a Project

After you have set up the repository and credentials, add the Spring Integration for VMware GemFire library to your application. To allow for more flexibility with multiple GemFire version, the Spring Integration for VMware GemFire library requires users to add an explicit dependency on the desired version of GemFire. The required dependencies differ depending on whether users a building a client application or a server application.

In the following examples:

- Update the `springIntegrationForGemFire.version` with the version of the library that your project requires.
- Update the `vmwareGemFire.version` with the version of GemFire that your project requires.

For **client** applications:

- **Maven**: Add the following to your `pom.xml` file.

  ```
  <properties>
      <springIntegrationForGemFire.version>1.0.0</springIntegrationForGemFire.ver
  sion>
      <vmwareGemFire.version>10.0.1</vmwareGemFire.version>
  </properties>

  <dependencies>
     <dependency>
         <groupId>com.vmware.gemfire</groupId>
         <artifactId>spring-integration-6.1-gemfire-10.0</artifactId>
         <version>{springIntegrationForGemFire.version}</version>
     </dependency>
     <dependency>
         <groupId>com.vmware.gemfire</groupId>
  ```

```
        <artifactId>gemfire-core</artifactId>
        <version>${vmwareGemFire.version}</version>
    </dependency>
    <dependency>
         <groupId>com.vmware.gemfire</groupId>
         <artifactId>gemfire-cq</artifactId>
         <version>${vmwareGemFire.version}</version>
    </dependency>
</dependencies>
```

- **Gradle**: Add the following to your `build.gradle` file.

```
ext {
    springIntegrationForGemFireVersion = '1.0.0'
    vmwareGemFireVersion = '10.0.1'
}

dependencies {
    implementation "com.vmware.gemfire:spring-integration-6.1-gemfire-10.0:$spr
ingIntegrationForGemFireVersion"
    implementation "com.vmware.gemfire:gemfire-core:$vmwareGemFireVersion"
    implementation "com.vmware.gemfire:gemfire-cq:$vmwareGemFireVersion"
}
```

Spring Integration for VMware GemFire

Background

GemFire is a distributed data management platform that provides a key-value data grid along with advanced distributed system features, such as event processing, continuous querying, and remote function execution. This guide assumes some familiarity with the commercial VMware GemFire.

Spring Integration provides support for GemFire by implementing inbound adapters for entry and continuous query events, an outbound adapter to write entries to the cache, and message and metadata stores and `GemfireLockRegistry` implementations. Spring Integration leverages the Spring Data for VMware GemFire project, providing a thin wrapper over its components.

To configure the XML `int-gfe` namespace, include the following elements within the headers of your XML configuration file:

```
xmlns:int-gfe="http://www.springframework.org/schema/integration/gemfire"
xsi:schemaLocation="http://www.springframework.org/schema/integration/gemfire
https://www.springframework.org/schema/integration/gemfire/spring-integration-gemfire.
xsd"
```

Adapters

Inbound Channel Adapter

The inbound channel adapter produces messages on a channel when triggered by a GemFire `EntryEvent`. GemFire generates events whenever an entry is `CREATED`, `UPDATED`, `DESTROYED`, or `INVALIDATED` in the associated region. The inbound channel adapter lets you filter on a subset of these events. For example, you may want to produce messages only in response to an entry being created.

In addition, the inbound channel adapter can evaluate a SpEL expression if, for example, you want your message payload to contain an event property such as the new entry value. The following example shows how to configure an inbound channel adapter with a SpEL language (in the `expression` attribute):

```
<gfe:cache/>
<gfe:replicated-region id="region"/>
<int-gfe:inbound-channel-adapter id="inputChannel" region="region"
cache-events="CREATED" expression="newValue"/>
```

The preceding configuration creates a GemFire `Cache` and `Region` by using Spring GemFire's 'gfe' namespace. The `inbound-channel-adapter` element requires a reference to the GemFire region on which the adapter listens for events.

Optional attributes include `cache-events`, which can contain a comma-separated list of event types for which a message is produced on the input channel.

By default, `CREATED` and `UPDATED` are enabled. If no `channel` attribute is provided, the channel is created from the `id` attribute. This adapter also supports an `error-channel`. The GemFire `EntryEvent` is the `#root` object of the `expression` evaluation.

The following example shows an expression that replaces a value for a key:

```
expression="new something.MyEvent(key, oldValue, newValue)"
```

If the `expression` attribute is not provided, the message payload is the GemFire `EntryEvent` itself.

NOTE: This adapter conforms to Spring Integration conventions.

Continuous Query Inbound Channel Adapter

The continuous query inbound channel adapter produces messages on a channel when triggered by a GemFire continuous query or `CqEvent` event. Spring Data introduced continuous query support, including `ContinuousQueryListenerContainer`, which provides a nice abstraction over the GemFire native API. This adapter requires a reference to a `ContinuousQueryListenerContainer` instance. The adapter will create a listener for a given `query`, and executes the CQ query against the GemFire cluster to receive events matching the query. The continuous query acts as an event source that fires whenever its result set changes state.

NOTE: GemFire queries are written in OQL (Object Query Language see GemFire Querying) and are scoped to the entire cache (not just one region).

Continuous Queries (CQ's) are a client-side construct and require access to a remote VMware GemFire.

See the GemFire Continuous Queries for more information on implementing continuous queries.

The following configuration creates a GemFire client cache (recall that a remote cache server is required for this implementation and its address is configured as a child element of the pool), a client region, and a `ContinuousQueryListenerContainer` that uses Spring Data:

```
<gfe:client-cache id="client-cache" pool-name="client-pool"/>

<gfe:pool id="client-pool" subscription-enabled="true" >
<!--configure server or locator here required to address the cache server -->
</gfe:pool>

<gfe:client-region id="test" cache-ref="client-cache" pool-name="client-pool"/>

<gfe:cq-listener-container id="queryListenerContainer" cache="client-cache"
pool-name="client-pool"/>

<int-gfe:cq-inbound-channel-adapter id="inputChannel"
cq-listener-container="queryListenerContainer"
query="select * from /test"/>
```

The continuous query inbound channel adapter requires a `cq-listener-container` attribute, which must contain a reference to the `ContinuousQueryListenerContainer`. Optionally, it accepts an

`expression` attribute that uses SpEL to transform the `CqEvent` or extract an individual property as needed.

The `cq-inbound-channel-adapter` provides a `query-events` attribute that contains a comma-separated list of event types for which a message is produced on the input channel. The available event types are `CREATED`, `UPDATED`, `DESTROYED`, `REGION_DESTROYED`, and `REGION_INVALIDATED`. By default, `CREATED` and `UPDATED` are enabled.

Additional optional attributes: * `query-name` (which provides an optional query name) * `expression` (which works as described in the preceding section) * `durable` (a boolean value indicating if the query is durable – it is false by default).

If you do not provide a `channel`, the channel is created from the `id` attribute. This adapter also supports an `error-channel`.

Outbound Channel Adapter

The outbound channel adapter writes cache entries that are mapped from the message payload. In its simplest form, it expects a payload of type `java.util.Map` and puts the map entries into its configured region. The following example shows how to configure an outbound channel adapter:

```
<int-gfe:outbound-channel-adapter id="cacheChannel" region="region"/>
```

Given the preceding configuration, an exception is thrown if the payload is not a `Map`. Additionally, you can configure the outbound channel adapter to create a map of cache entries by using SpEL. The following example shows how to do so:

```
<int-gfe:outbound-channel-adapter id="cacheChannel" region="region">
    <int-gfe:cache-entries>
        <entry key="payload.toUpperCase()" value="payload.toLowerCase()"/>
        <entry key="'thing1'" value="'thing2'"/>
    </int-gfe:cache-entries>
</int-gfe:outbound-channel-adapter>
```

In the preceding configuration, the inner element (`cache-entries`) is semantically equivalent to a Spring 'map' element. The adapter interprets the `key` and `value` attributes as SpEL expressions with the message as the evaluation context. Note that this can contain arbitrary cache entries (not only those derived from the message) and that literal values must be enclosed in single quotes. In the preceding example, if the message sent to `cacheChannel` has a `String` payload with a value `Hello`, two entries (`[HELLO:hello, thing1:thing2]`) are written (either created or updated) in the cache region. This adapter also supports the `order` attribute, which may be useful if it is bound to a `PublishSubscribeChannel`.

Gemfire Message Store

According to Enterprise Integration Patterns (EIP), a message store lets you persist messages. This can be useful when dealing with components that have a capability to buffer messages (`QueueChannel`, `Aggregator`, `Resequencer`, and others) if reliability is a concern. In Spring Integration, the `MessageStore` strategy interface also provides the foundation for the claim check pattern, which is described in EIP as well.

Spring Integration's Gemfire module provides `GemfireMessageStore`, which is an implementation of both the `MessageStore` strategy (mainly used by the `QueueChannel` and `ClaimCheck` patterns) and the `MessageGroupStore` strategy (mainly used by the `Aggregator` and `Resequencer` patterns).

The following example configures the cache and region by using the `spring-gemfire` namespace (not to be confused with the `spring-integration-gemfire` namespace):

```
<bean id="gemfireMessageStore" class="o.s.i.gemfire.store.GemfireMessageStore">
    <constructor-arg ref="myRegion"/>
</bean>

<gfe:cache/>

<gfe:replicated-region id="myRegion"/>


<int:channel id="somePersistentQueueChannel">
    <int:queue message-store="gemfireMessageStore"/>
<int:channel>

<int:aggregator input-channel="inputChannel" output-channel="outputChannel"
message-store="gemfireMessageStore"/>
```

Often, it is desirable for the message store to be maintained in one or more remote cache servers in a client-server configuration. In this case, you should configure a client cache, a client region, and a client pool and inject the region into the `MessageStore`. The following example shows how to do so:

```
<bean id="gemfireMessageStore"
class="org.springframework.integration.gemfire.store.GemfireMessageStore">
    <constructor-arg ref="myRegion"/>
</bean>

<gfe:client-cache/>

<gfe:client-region id="myRegion" shortcut="PROXY" pool-name="messageStorePool"/>

<gfe:pool id="messageStorePool">
    <gfe:server host="localhost" port="40404" />
</gfe:pool>
```

Note that the `pool` element is configured with the address of a cache server (you can substitute a locator here). The region is configured as a 'PROXY' so that no data is stored locally. The region's `id` corresponds to a region with the same name in the cache server.

The `GemfireMessageStore` supports the key `prefix` option to uniquely describe multiple instances linked to the same GemFire Region.

Gemfire Lock Registry

Spring Integration for VMware GemFire also has a `GemfireLockRegistry`. Certain components (for example, the aggregator and the resequencer) use a lock obtained from a `LockRegistry` to ensure thread safety within an integration flow.

By default, there is a `DefaultLockRegistry` performs this function within a component, but external lock registries can be configured.

When you use a shared `MessageGroupStore` with the `GemfireLockRegistry`, it can provide this functionality across multiple application instances, so that only one instance can manipulate the group at a time.

NOTE: One of the `GemfireLockRegistry` constructors requires a `Region` as an argument. It is used to obtain a `Lock` from the `getDistributedLock()` method. This operation requires `GLOBAL` scope for the `Region`. Another constructor requires a `Cache`, and the `Region` is created with `GLOBAL` scope and with the name, `LockRegistry`.

Gemfire Metadata Store

You can use the `GemfireMetadataStore` to maintain metadata state across application restarts. This new `MetadataStore` implementation can be used with adapters such as:

- Feed Inbound Channel Adapter
- Reading Files
- FTP Inbound Channel Adapter
- SFTP Inbound Channel Adapter

To get these adapters to use the new `GemfireMetadataStore`, declare a Spring bean with a bean name of `metadataStore`. The feed inbound channel adapter automatically picks up and use the declared `GemfireMetadataStore`.

NOTE: The `GemfireMetadataStore` also implements `ConcurrentMetadataStore`, letting it be reliably shared across multiple application instances, where only one instance can store or modify a key's value. These methods give various levels of concurrency guarantees based on the scope and data policy of the region. They are implemented in the peer cache and client-server cache but are disallowed in peer regions that have `NORMAL` or `EMPTY` data policies.

NOTE: The `GemfireMetadataStore` also implements `ListenableMetadataStore`, which lets you listen to cache events by providing `MetadataStoreListener` instances to the store, as the following example shows:

```
GemfireMetadataStore metadataStore = new GemfireMetadataStore(cache);
metadataStore.addListener(new MetadataStoreListenerAdapter() {

    @Override
    public void onAdd(String key, String value) {
         ...
    }

});
```