Gearman Java Message Service Interface (GJMSI)

v0.1

Adam Elston-Jones
Isaiah Van der Elst
John Ogle
Nick Harsh
Rogelio Hernandez
Scott Merz

Table of Contents

1

About Gearman Java Message Service Interface

Gearman Java Message Service Interface is a project that implements Sun Microsystems Java Messaging System's (JMS) API 1.1 specification. GJMSI is an API that supports the formal communication known as messaging between computers in a network. It provides a way for programs to create, send, receive and read messages in any distributed system.

Currently GJMSI is being developed as a Capstone project at Portland State University. The goal of this project is to get as close to the JMS API specifications within the time constrain of our Capstone project. The project is being hosted at *Launchpad* under the project name “Gearman JMS.”

https://launchpad.net/gearman-jms

GJMSI is using the Simplified BSD License. Please see the COPYING file in the project for further details.

The GJMSI project is currently working with a distributed system call *Gearman* as well as using *Gearman Java* which implements Gearman which is originally written in C.

2

Why Gearman Java Message Service Interface

Java messaging in today's applications often becomes big and complex. By using JMS, Java applications can improve in design by taking advantage of named message queues or by using anonymous publishing/subscribing messaging systems, both of which are supported by JMS. As it turns out, most JMS implementations come bundled together with a J2EE application server which is already customize to the customer's needs and makes it hard for companies to replace the application server if a better, cheaper options comes along. Enter GJMSI. GJMSI is a messaging service that leaves the configuration to the user and makes it easier to make decision such as swapping a new or cheaper application server if the need arises with as little work and hustle as possible.

JMS also bring advantages over RPC because JMS does not have to wait for the method to finish execution and return control back to the invoker. Hence, JMS clients do not have to wait for a response from the server and can happily continue with other work. JMS also bring a variety of types of messages such as Message, TextMessage, ByteMessage, StreamMessage, ObjectMessage, and MapMessage.

3

Gearman

Since GJMSI, is working with the distributed system *Gearman*. It is only natural to make a quick reference to it. As mention in the *About Gearman Java Message Service Interface* section, Gearman is located in *Launchpad*. The project can be downloaded by using the following Bazaar command:

```
bzr branch lp:gearmand
```

A deb version of Gearman can be downloaded from a Personal Package Archive (PPA). Located at:

https://launchpad.net/~gearman-developers/+archive/ppa

Once you have install Gearman we can start using the job server by calling it with the following command:

$ gearmand –d

This will start the job server and put it in the background. You can also call it in with arguments "-vv" to see it running in debug mode.

We will also be using *Gearman in Java* a pure Java implementation of Gearman. This will helps us to work with Gearman and to minimize the code needed. More information can be obtained by going to their Launchpad page at https://launchpad.net/gearman-java.

That is all we need to do for Gearman. More information about Gearman can be found at their Wiki which is located at **Gearman.org**. You can find more information about all the Gearman projects that are currently available by visiting *launchpad.net/gearman*.

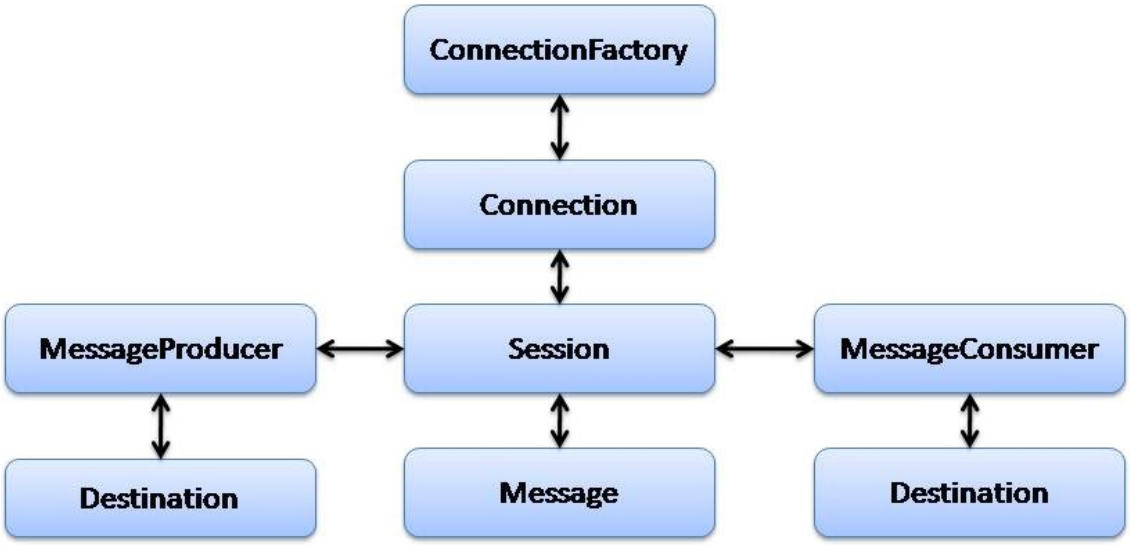
4

Gearman Java Message Service Interface Design

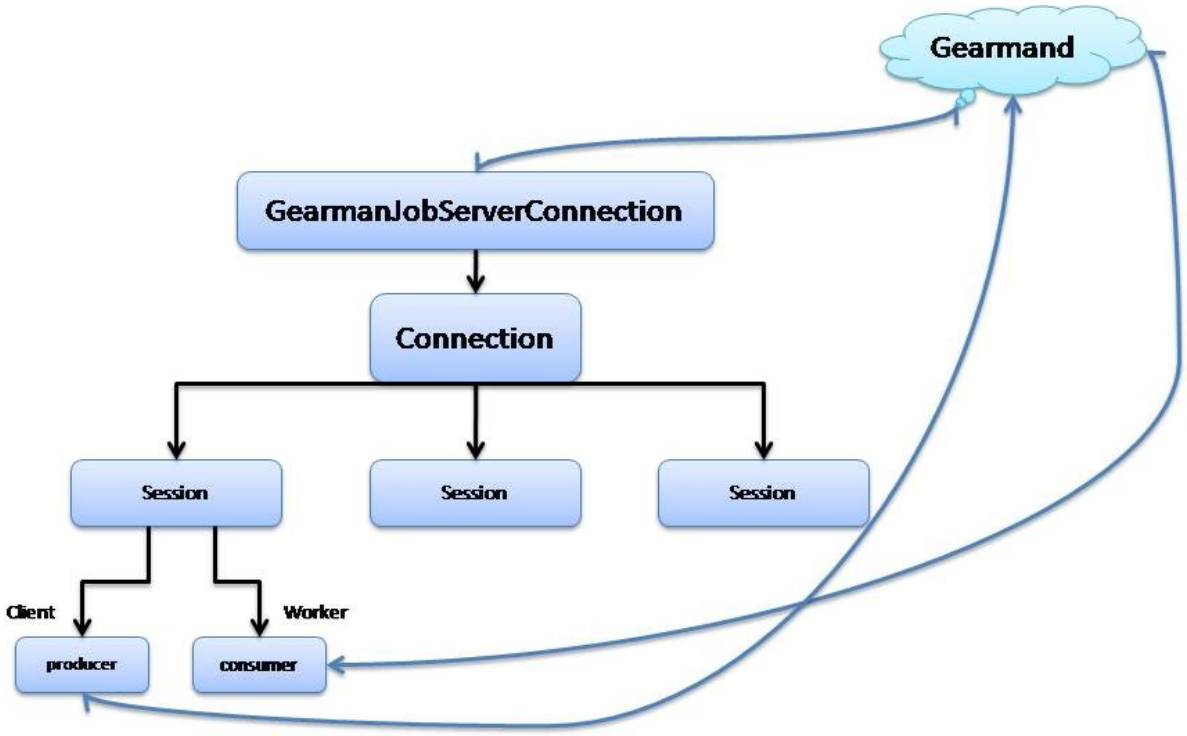
The top level overview of GJMSI starts with a ConnectionFactory which can create a Connection and Connection can create Sessions. Sessions can produce both MessageConsumers and MessageProducers and both can create a Destination.

ConnectionFacatory is an object that encapsulates a set of connection configuration parameters that has been defined by an administrator. A Session object is a single-threaded context for producing and consuming messages. MessageConsumer is used to receive messages from a destination and MessageProducer sends messages to a destination. Message is the root interface of all JMS messages. Message defines the header and the acknowledge method used for all messages. A diagram of this high level representation of GJMSI is as follows:

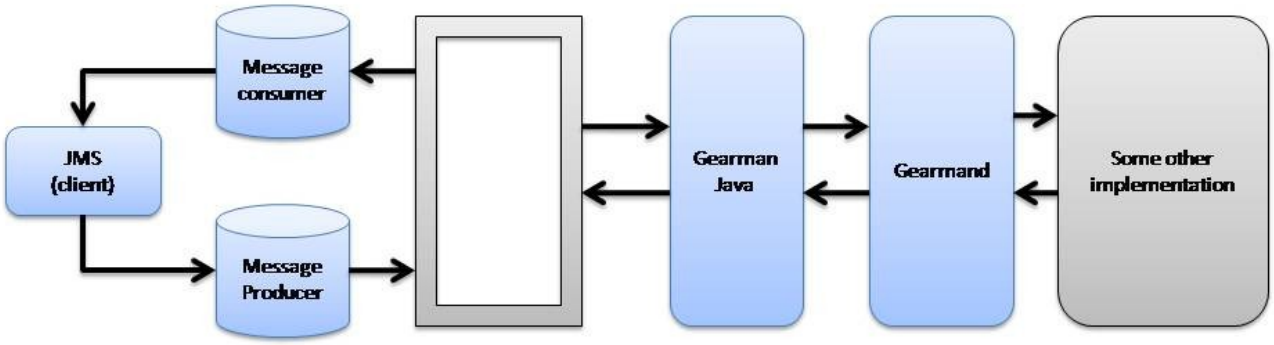
GJMSI High Level Design



Since we are working with Gearman, we will be using some of its functionality for example using Geaman's job server connection. An illustration of the interaction is as follows:



Another representation of what is happening:



As an extra encapsulation of our messages, we will use Google's Protobuf protocol, which is being done in the Message layer before being sent off to its destination. This is done because Protobuf gives us a way of encoding structured data in an efficient, flexible, smaller, and simpler way. More information can be obtained about Google's Protobuf from their site which is located at http://code.google.com/apis/protocolbuffers/docs/overview.html.

Furthermore, the JMS needs to have JNDI for the purpose of getting a ConnectionFatory or a Destination object but for the purpose of this project a fully functional JNDI is not possible and a fake JNDI is implemented instead that generates only the needed functionality. This does not mean that a fully working JNDI cannot be swap in for our JNDI.

5

Getting Started

The first step is to download the project from Launchpad. The project can be downloaded by using the bazaar command:

```
bzr branch lp:gearman-jms
```

This will download the current truck branch into the current directory you are working from. The project is being developed using Eclipse so you can import the project if you wish.

The project files can also be viewed on line if you go to GJMSI Launchpad page and go to the code section and select current truck you will see a link that will take you to the Source Code page where the files can be easily browse.

There is no configuration needed. You just need to compile the code and make sure you are running the Gearman job server. There are some code examples in the examples folder for a quick glance at how to setup a client and a worker. How to reverse a string is cover in the next section.

6

Reverse Example

In this section we cover a simple example that will reverse a given string. The example does not require any input since all its arguments are hard coded into the class but if you want to see all the options that GJMSI provides when creating a message a quick glance at the project will show all the options available to you.

The reverse example takes the string “Hello World,” packages it, and sends it off to get reserved. Then the worker that is designed to do work for the function “reverse” will work on the job and return the result to the client.

The first step is to make sure you have Gearman job server running.

```
$ gearmand –d
```

Secondly, we need to make a ConnectionFactory and save it to the JNDI repository. To do this, we need to run the “shell” command that will let you make Gearman-JMS administrator commands. To accomplish this type

```
$java shell
gearman> mkcf myCF
gearman> exit
```

in the command shell prompt. You can also type “help” for a small list of the options available. For complete documentation, see the docs folder in the project.

Once you have Gearman running and setup the ConnectionFactory, you can setup a worker that will be receiving the work that is designated for the function “reverse” or any function for that matter. We do this by calling *ReverseWorker*. I am assuming you are in the directory where you have compiled GJMSI or have configured the PATH so it know where the classes are located in.

```
$ java ReverseWorker
```

Now any client that wants to have a string reversed can call the server with the “Reverse” destination and there will be a worker waiting do the work. We call the reverse client by using the following:

```
$ java ReverseClient
```

This will send the request to the Gearman job server with the destination “reverse.” After the job is done, the following result is returned.

```
dlrow olleH
```

Congratulations. You have completed your first job running GJMSI.

7
Common Methods

Here we have a small outline of commonly used methods:

InitialContext & jndi.lookup are used for set-up
InitialContext() - get ConnectionFactory (vendor specific)
jndi.lookup("myCF") – returns ConnectionFactory

Example:
Context jndi = new InitialContext();
ConnectionFactory cf = (ConnectionFactory) jndi.lookup("myCF");

The following methods are used when creating a connection:
createConnection() - returns a connection (gearmand)
createSession(boolean , (int) Acknowledge mode) – creates a session
createQueue("Function_Name") – creates queue with given function name

Example:
m_conn = cf.createConnection();
m_session = m_conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
m_dest = m_session.createQueue("Reverse");

Constructing a Message:
createProducer(Destination) - creates a producer to send messages
createTemporaryQueue() - creates temporary queue
createConsumer(return_destination) - creates consumer for incoming messages

Example:
m_producer = m_session.createProducer(m_dest);
m_returndest = m_session.createTemporaryQueue();
m_consumer = m_session.createConsumer(m_returndest);

start listening for incoming messages:
m_conn.start();

Please look in the examples folder for a better look at how these methods are constructed and used.

8

Packages & project Hierarchy

The package structure for GJMSI is as follows:

- org.gearman.jms
- org.gearman.jms.Commands
- org.gearman.jms.Common
- org.gearman.jms.Connection
- org.gearman.jms.ConnectionFactory
- org.gearman.jms.Destination
- org.gearman.jms.Message
- org.gearman.jms.Message.MessageConsumer
- org.gearman.jms.Message.MessageProducer
- org.gearman.jms.Session
- org.gearman.jndi.naming

The project files structure:

- examples/src – GJMSI examples
- src – source of all classes use for GJMSI
- test – Junit test cases
- doc – javadocs for GJMSI
- examples – more examples
- lib – binaries need to run GJMSI

Note: Gearman Java, J2EE and Protobuf have been added to the project as jar files.