CS544 EA

Integration

Messaging: Spring RabbitMQ

RabbitMQ

- **RabbitMQ** is a popular message-oriented middleware server using the **AMQP** protocol
  - Plugin support for STOMP, MQTT, and many others
  - Written in Erlang, client libraries for all major langs.
  - 2010 acquired by **SpringSource** (division of VMware)
  - Source code released under Mozilla Public License

AMQP Terminology

- **Producers**: send messages
- **Consumers**: receive messages
- **Broker**: the middleware server
- **Queue**: where messages are stored on the broker
- **Exchange**: what receives the messages on the broker and routes them to queues

RabbitMQ config

- Exchanges and Queues are not configured through a config file on the broker
  - Created with the Broker API by producer/consumer

- **Only need to create a queue**
  - There is a default Exchange (does direct delivery)
  - Give the name of the queue and it send it there

Hello World: Spring Boot Sender

```
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

Spring boot automatically configures a RabbitTemplate bean for this dependency

```
package edu.mum.cs544.message;

import org.springframework.amqp.core.Queue;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class Application {

  @Bean
  public Queue hello() {
    return new Queue("hello");
  }

  public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
  }
}
```

Only thing we need to add / configure is a Queue bean to be created on the broker

```
package edu.mum.cs544.message;

import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class Sender implements CommandLineRunner {
      @Autowired
      private RabbitTemplate template;

      @Override
      public void run(String... args) throws Exception {
            String queue = "hello";
            String msg = "Hello World!";
            template.convertAndSend(queue, msg);
            System.out.println("Sent: " + msg +" to: " + queue);
      }
}
```

Send to our queue

Hello World: Spring Boot Receiver

```
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

Separate Receiver application has same dependency

```
package edu.mum.cs544.message;

import org.springframework.amqp.core.Queue;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class Application {

  @Bean
  public Queue hello() {
    return new Queue("hello");
  }

  public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
  }
}
```

Again just configure the queue

```
package edu.mum.cs544.message;

import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
@RabbitListener(queues = "hello")
public class Receiver {

      @RabbitHandler
      public void receive(String msg) {
            System.out.println("Received: " + msg);
      }
}
```

Our queue

@Rabbit annotations register our Listener and Handler

Optional application.properties

- These are the **default values**
  - Leaving them off gives the same result
  - Both for sender and receiver project
  - Only need to specify them if different values needed

application.properties

```
spring.rabbitmq.host=localhost
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
spring.rabbitmq.port=5672
```