1. Запускаем плагин C:\Program Files\RabbitMQ Server\rabbitmq\_server-3.9.13\sbin server
2. Переходим на сайт рэббита <http://localhost:15672/#/>

СХЕМА РАБОТЫ В IDEA

(publisher/producer) Отправляем через постман запрос в контроллер -> контроллер принимает его и отправляет объект, согласно указанным условиям, в exchange -> exchange отправляет объект в связанную с ним очередь -> очередь отправляет объект прикрепленным слушателям

Publisher -> Exchange -> Queues -> Consumer

1. Привязать exchange к queue (на вкладке E вкладка bindings добавляем название очереди и всё)
2. Опубликовать сообщение в Е вкладка, вводим только текс сообщения. В Q он отобразится
3. Получить сообщение в Q get messages (Ask mode (возвращать или нет сообщение в очередь после получения его, кодировка, сколько сообщений получать за раз).

AMQP -  Advanced Message Queuing Protocol (AMQP)

Типы Е:

Ключи – набор слов, соединенных между собой точкой.

1) fanout – ключи маршрутизации не учитываются. Отправляет сообщения во все очереди, кт с ним связаны.

2) direct – когда совпадают ключи маршрутизации, т.е. сообщения попадают в те очереди, ключи которых полностью посимвольно совпадают с ключами маршрутизации сообщений (в сообщении и очереди задан явный ключ error.errorExt (просто пример). Если ключ не совпадает, то идёт потеря сообщения

3) Topic – проверяет ключи по шаблонам, т.е. если ключ соответствует какой-то логики (напр error.\* - сообщение отправляется в одни очереди, если #.info – в другие, где заданы такие же части ключа)

Сам пример <https://habr.com/ru/post/262069/>

Настройки:

1. В конфиг классе:

а) настраиваем подключение, amqp клиента и др:

@Bean  
public ConnectionFactory connectionFactory() {  
 return new CachingConnectionFactory("localhost"); }  
  
*/\*AmqpAdmin - это функциональный компонент системы управления Rabbit MQ, используемый  
для создания и удаления очереди, обмена и привязки.\*/*@Bean  
public AmqpAdmin amqpAdmin() {  
 return new RabbitAdmin(connectionFactory()); }  
  
*/\*Вспомогательный класс, упрощающий синхронный доступ к RabbitMQ (отправка и получение сообщений).\*/*@Bean  
public RabbitTemplate rabbitTemplate() {  
 return new RabbitTemplate(connectionFactory()); }

б) настраиваем очереди, присваиваем им имена и т.д.

@Bean  
public Queue myQueue1() {  
 return new Queue("new\_queue1"); }  
  
@Bean  
public Queue myQueue2() {  
 return new Queue("new\_queue2"); }

в) создаём обменники (exchange):

а.1) если хотим сделать обменник, который будет отправлять очередям сообщения без ключей

*/\*бесключевой обменник\*/*@Bean  
public FanoutExchange fanoutExchange(){  
 return new FanoutExchange("common\_Exchange"); }

а.2) если ключ указываем целиком или частично:

@Bean  
public TopicExchange topicExchange(){  
 return new TopicExchange("topic\_exchange1");}  
  
@Bean  
public DirectExchange directExchange(){  
 return new DirectExchange("direct\_exchange1"); }

г) настраиваем взаимосвязь между очередью и обменником

1)если связываем очередь с обменником без ключей

@Bean  
public Binding bindingFanoutExchange1(){  
 return BindingBuilder.*bind*(myQueue1()).to(fanoutExchange()); }

2) связываем со строгим ключом

@Bean  
public Binding bindingDirectExchange1(){  
 return BindingBuilder.*bind*(myQueue1()).to(directExchange()).with("error"); }

3) связываем с частью ключа

@Bean  
public Binding bindingTopic(){  
 return BindingBuilder.*bind*(myQueue1()).to(topicExchange()).with("error.\*"); }

1. Настраиваем слушателей очередей (связываем слушателя с очередью)

@EnableRabbit  
@Component  
public class Consumer {  
 @RabbitListener(queues = "new\_queue1")  
 public void processMyQueue1(String message) {делаем что.то с message} }

1. Настраиваем контроллер для работы с очередями

@RestController  
public class Controller {  
  
 */\*Задает базовый набор операций AMQP. Обеспечивает синхронные методы отправки и получения. Методы convertAndSend(Object) и receiveAndConvert() позволяют отправлять и получать объекты POJO. Ожидается, что реализации будут делегированы экземпляру MessageConverter для выполнения преобразования в и из типа полезной нагрузки AMQP byte[].\*/* private final RabbitTemplate rabbitTemplate;

public Controller(RabbitTemplate rabbitTemplate) {  
 this.rabbitTemplate = rabbitTemplate;}

контроллер для очередей без ключа  
 @PostMapping("/edit")  
 public String edit(@RequestBody String message) {  
 rabbitTemplate.setExchange("common\_Exchange");  
 rabbitTemplate.convertAndSend("new\_queue");  
 return "success default exchange";  
 }  
  
 @PostMapping("/edit1")  
 public String directExchange(@RequestBody Map<String, String> map) {  
 logger.info("edit to my Queue");  
 rabbitTemplate.setExchange("direct\_exchange1");  
 rabbitTemplate.convertAndSend(map.get("key"), map.get("message"));  
  
 return "success DirectExchange";  
 }  
  
 @PostMapping("/edit2")  
 public String topicExchange(@RequestBody Map<String, String > map){  
 logger.info("edit to my Queue");  
 rabbitTemplate.setExchange("topic\_exchange1");  
 rabbitTemplate.convertAndSend(map.get("key"), map.get("message"));  
 return "success TopicExchange";  
 }  
}