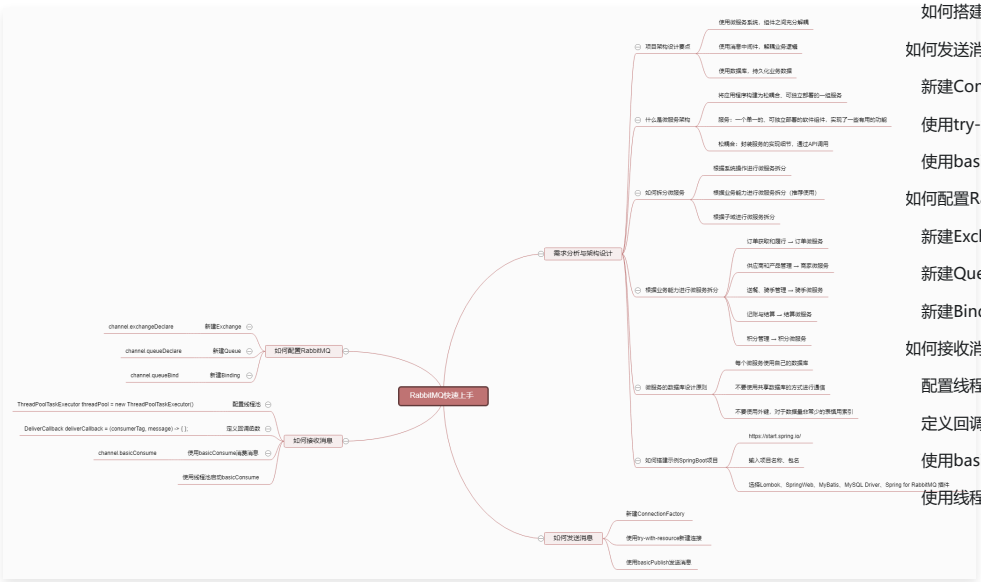
Moody • 更新于 2020-10-10

上一节 第二章RabbitM... 第四章RabbitM... 下一节



需求分析与架构设计

项目架构设计要点

- 使用微服务系统，组件之间充分解耦
- 使用消息中间件，解耦业务逻辑
- 使用数据库，持久化业务数据

什么是微服务架构

- 将应用程序构建为松耦合、可独立部署的一组服务
- 服务：一个单一的、可独立部署的软件组件，实现了一些有用的功能
- 松耦合：封装服务的实现细节，通过API调用

如何拆分微服务

- 根据系统操作进行微服务拆分
- 根据业务能力进行微服务拆分（推荐使用）
- 根据子域进行微服务拆分

根据业务能力进行微服务拆分

- 订单获取和履行 → 订单微服务
- 供应商和产品管理 → 商家微服务
- 送餐、骑手管理 → 骑手微服务

- 记账与结算 → 结算微服务
- 积分管理 → 积分微服务

微服务的数据库设计原则

- 每个微服务使用自己的数据库
- 不要使用共享数据库的方式进行通信
- 不要使用外键，对于数据量非常少的表慎用索引

如何搭建示例SpringBoot项目

- https://start.spring.io/
- 输入项目名称、包名
- 选择Lombok、SpringWeb、MyBatis、MySQL Driver、Spring for RabbitMQ 插件

如何发送消息

新建ConnectionFactory

代码块

```
ConnectionFactory connectionFactory = new ConnectionFactory();
connectionFactory.setHost("localhost");
connectionFactory.setHost("localhost");
```

使用try-with-resource新建连接

代码块

```
try (Connection connection = connectionFactory.newConnection();
      Channel channel = connection.createChannel()) {

          //业务逻辑

      }
```

使用basicPublish发送消息

代码块

```
channel.basicPublish("exchange.name", "key.name", null, messageToSend.getBytes());
```

如何配置RabbitMQ

新建Exchange

代码块

```
channel.exchangeDeclare(
                    "exchange.name",
                    BuiltinExchangeType.DIRECT,
                    true,
                    false,
                    null);
```

新建Queue

代码块

```
channel.queueDeclare(
                    "queue.name",
                    true,
                    false,
                    false,
                    null);
```

新建Binding

代码块

```
channel.queueBind(
        "queue.name",
        "exchange.name",
        "key.name");
```

如何接收消息

配置线程池

代码块

```
@Configuration
@EnableAsync
public class AsyncTaskConfig implements AsyncConfigurer {

    // ThredPoolTaskExcutor的处理流程
    // 当池子大小小于corePoolSize，就新建线程，并处理请求
    // 当池子大小等于corePoolSize，把请求放入workQueue中，池子里的空闲线程就去workQueue中取任
    // 当workQueue放不下任务时，就新建线程入池，并处理请求，如果池子大小撑到了maximumPoolSize，
    // 当池子的线程数大于corePoolSize时，多余的线程会等待keepAliveTime长时间，如果无请求可处理

    @Override
    @Bean
    public Executor getAsyncExecutor() {
        ThreadPoolTaskExecutor threadPool = new ThreadPoolTaskExecutor();
        //设置核心线程数
        threadPool.setCorePoolSize(10);
        //设置最大线程数
        threadPool.setMaxPoolSize(100);
        //线程池所使用的缓冲队列
        threadPool.setQueueCapacity(10);
        //等待任务在关机时完成--表明等待所有线程执行完
        threadPool.setWaitForTasksToCompleteOnShutdown(true);
        // 等待时间 （默认为0，此时立即停止），并没等待xx秒后强制停止
        threadPool.setAwaitTerminationSeconds(60);
        //  线程名称前缀
        threadPool.setThreadNamePrefix("Rabbit-Async-");
        // 初始化线程
        threadPool.initialize();
        return threadPool;
    }

    @Override
    public AsyncUncaughtExceptionHandler getAsyncUncaughtExceptionHandler() {
        return null;
    }
```

定义回调函数

代码块

```
    DeliverCallback deliverCallback = (consumerTag, message) -> {

        //业务逻辑

        };
```

使用basicConsume消费消息

代码块

```
    @Async
        public void handleMessage() {
            channel.basicConsume("queue.name", true, deliverCallback, consumerTag -> {
                        });
        }
```

使用线程池启动basicConsume

代码块

```
    @Autowired
    public void startListenMessage() throws IOException, TimeoutException, InterruptedExcept
        orderMessageService.handleMessage();
    }
```

第二章RabbitMQ快速入门回顾 上一节 下一节 第四章RabbitMQ高级特性总结

我要提出意见反馈