# 概述

edu-async-request模块是异步请求模块，例如通过浏览器访问controller接口，接口层将参数封装到 CommonRequest实体中，使用RequestAsyncProcessService（spring注入）发送请求到线程池中。注意，使用此方式操作数据库时，没有事务。

此模块不仅限于浏览器访问接口，postman、mq、单元测试等任何可以主动发送请求的方式，都可以使用此模块异步执行请求。

**优势**：

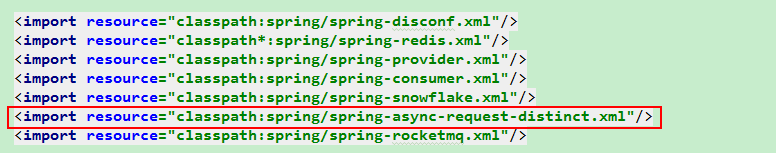
1. 一定程度上解决同一时刻的高并发
2. 一定程度上减轻数据库负担
3. 路由具有同一属性的请求到同一个队列，保证顺序消费
4. 不考虑请求丢失的情况下，去重请求在一定程度上可以减少数据库死锁的概率
5. 提供并发控制的接口，方便开发者测试并发请求（不仅限于测试此模块的异步请求）

**劣势**：

1. 服务宕机会导致内存消息丢失
2. 去重队列可能丢失最新的数据
3. 模块提供了通用型请求、自定义请求和去重请求3种，目前仅支持使用其中的一种，当然可以通过重写配置文件使用多种（尚未测试）
4. 去重请求虽然在一定程度上可以减少数据库死锁的概率，但是仅限于一个服务节点，如果多个服务节点同时存在，那么，需要采用外部手段解决请求路由问题，例如控制相同ip段的用户请求发送到同一个服务节点

**使用此模块的前提**：

1. Spring-application.xml中引入配置文件（有3种配置），如果应用在mq中，那么配置文件应该在mq的配置之前，下面是引入去重模块的配置



**注意事项**：

模块提供3种请求，由于代码设计的原因，3种请求中存在多个类名称相同，因此要以包名作为区分

clife.sleep.commons.async.request.common包下的为通用请求

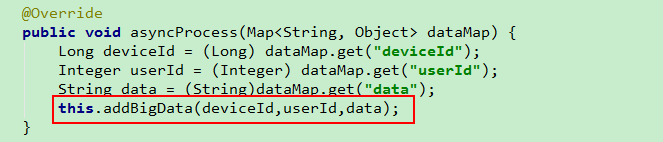
clife.sleep.commons.async.request.distinct包下的是去重请求，复用common包下的类

clife.sleep.commons.async.request.custom包下的是用户自定义请求

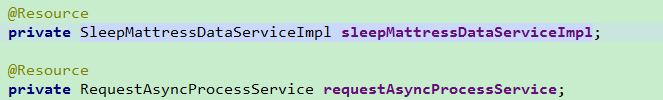
# 请求的使用方式

## Common请求的使用

1. 引入<import resource="classpath:spring/spring-async-request-common.xml"/>
2. 用户的service实现层实现clife.sleep.commons.async.request.common.interfaces.RequestAsyncService<D>接口，D为参数的类型，实现asyncProcess方法，在此方法中实现你的逻辑，并调用你真正的处理数据的方法。如下，D可以指定为Map类型，也可以指定为其他类型，或者自定义类型。



1. 注入用户service实现类和RequestAsyncProcessService



1. 封装CommonRequest实体



注意：此种请求，一个service只能处理一个方法。当然，你可以通过参数传递的方式，在asyncProcess实现中通过条件选择正确的方法

## Distinct请求的使用

此请求的第二步和第三步与common请求的步骤相同，第四步有差异。

如果你的工程中有多个地方要使用异步线程池，并且路由Id相同时，就要使用distinctKey指定去重规则

1. 引入<import resource="classpath:spring/spring-async-request-distinct.xml"/>
2. 封装DistinctRequest



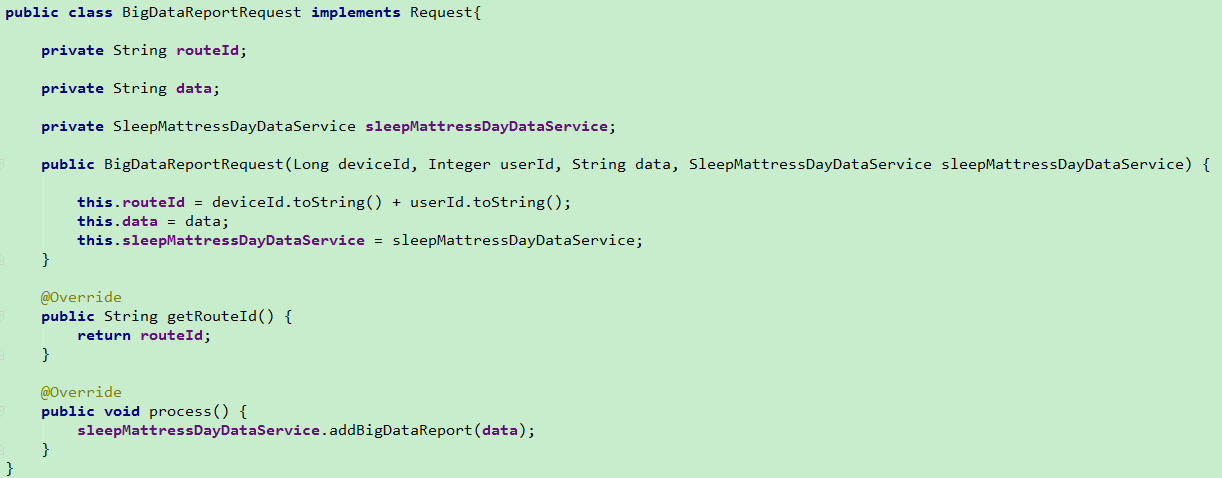
注意：同common请求。

另：我们可以通过条件判断封装DistinctRequest时，传递不同的参数和不同的distinctKey

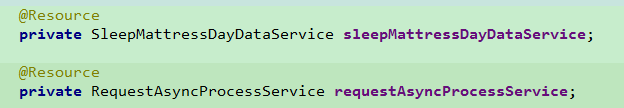
## Custom请求的使用

CommonRequest和DistinctRequest与service息息相关，但是Custom请求，用户不必关注service。Custom更灵活，用户可以使用在业务层、控制层

1. 引入<import resource="classpath:spring/spring-async-request-custom.xml"/>
2. 自定义request，实现clife.sleep.commons.async.request.custom.interfaces.Request，如下BigDataReportRequest



1. 注入要使用的service服务和clife.sleep.commons.async.request.custom.interfaces.RequestAsyncProcessService



1. 封装BigDataReportRequest



注意：custom没有实现去重

# AbstractConcurrentControl的使用

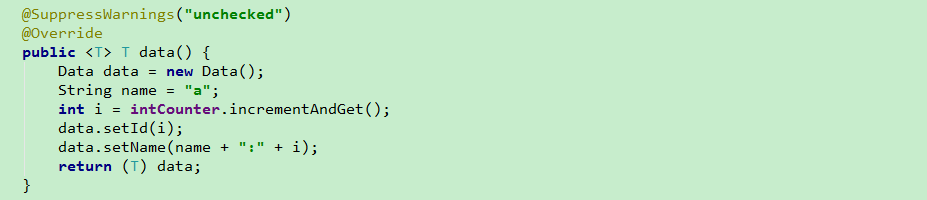
1. 单元测试继承AbstractConcurrentControl



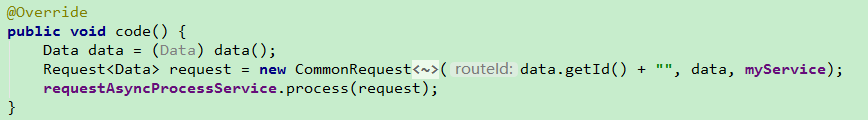
1. 单元测试提供构造函数，指定并发线程数，如果不提供构造函数，默认并发线程数为2000。自定义并发线程数时，要根据自己电脑配置来设定



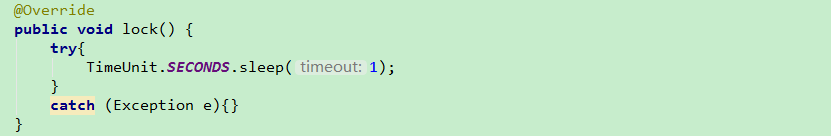
1. 实现AbstractConcurrentControl的data方法，提供并发数据，例如并发测试死数据，构造动态的主键



1. 实现AbstractConcurrentControl的code方法，提供并发代码，在此方法中调用你实现的data方法获取并发数据，在并发请求到达并发线程数之前，所有请求都阻塞在调用此方法的地方



1. 实现AbstractConcurrentControl的lock方法，阻塞主线程，防止JVM结束进程，不推荐使用wait方式，因为在maven打包时，会阻塞。推荐使用主线程休眠



1. 写单元测试方法，在此方法中调用process()方法即可



# 注：如果发现bug，及时沟通