# 目录

[目录 1](#_Toc477365579)

[1. 架构介绍 2](#_Toc477365580)

[1. 架构图 2](#_Toc477365581)

[2. 功能介绍 2](#_Toc477365582)

[3. 开发指南 2](#_Toc477365583)

[1. 引入依赖 2](#_Toc477365584)

[2. Spring配置 3](#_Toc477365585)

[3. 接口说明 4](#_Toc477365586)

[1. EventBusFactory 4](#_Toc477365587)

[2. EventListener<T> 5](#_Toc477365588)

[3. EventCallback <T> 5](#_Toc477365589)

[4. 例子 5](#_Toc477365590)

[4. 配置介绍 7](#_Toc477365591)

# 架构介绍

## 架构图



基于google的EventBus封装的同步事件框架，可以持久化事件。

# 功能介绍

**持久化事件框架：**可以选择是否持久化指定事件，并通过任务调度系统执行持久化的事件。如果将业务操作与事件发送置于同一个事务中，可确保业务操作和事件的一致性。

# 开发指南

### 引入依赖

<**dependency**>  
 <**groupId**>com.arto</**groupId**>  
 <**artifactId**>pergesa-event</**artifactId**>  
 <**version**>最新版本号</**version**>  
</**dependency**>

### Spring配置

*<!— 事件配置 -->*  
<**import resource="classpath:pergesa-event.xml"**/>

*<!-- 持久化事件恢复Job的ZK注册中心 -->*<**reg:zookeeper id="regCenter" serverLists="${serverLists}"  
 namespace="${namespace}"  
 baseSleepTimeMilliseconds="${baseSleepTimeMilliseconds}"  
 maxSleepTimeMilliseconds="${maxSleepTimeMilliseconds}"  
 maxRetries="${maxRetries}"** />  
  
*<!-- 持久化事件恢复Job -->*<**job:bean id="persistentEventRecoveryJob" class="com.arto.event.recovery.PersistentEventRecoveryJob" regCenter="regCenter"  
 shardingTotalCount="10"  
 cron="0 0/10 \* \* \* ? \*"  
 failover="true"  
 description="Persistent event recovery job"** />

*<!-- 持久化事件的jdbcTemplate，数据源必须与业务库一致 -->*

<**bean id="jdbcTemplate" class="org.springframework.jdbc.core.namedparam**

**.NamedParameterJdbcTemplate"**>

<**constructor-arg ref="dataSource"** />

</**bean**>  
  
<**bean id="transactionManager"  
class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>  
 <**property name="dataSource" ref="dataSource"** />  
</**bean**>  
  
<**tx:annotation-driven transaction-manager="transactionManager"**/>

需要确认与事务消息同一个事务的业务DB中存在如下表：

**CREATE TABLE** event\_storage  
(  
 **id** bigserial,  
 **tag smallint**,  
 **system\_id character varying**(64),  
 **business\_id character varying**(64),  
 **business\_type character varying**(64),  
 **EVENT\_TYPE character varying**(128),  
 **status smallint**,  
 **payload** text,  
 **retried\_count\_d smallint**,  
 **retried\_count\_c smallint**,  
 **next\_retry\_time timestamp** without **time zone**,  
 **memo** text,  
 **gmt\_created timestamp** without **time zone**,  
 **gmt\_modified timestamp** without **time zone**,  
 **CONSTRAINT** event\_storage\_pkey **PRIMARY KEY** (**id**, **tag**, **system\_id**)  
);  
  
**create** index idx\_event\_storage\_ts **on** event\_storage(status);  
**create** index idx\_event\_storage\_gm **on** event\_storage(gmt\_modified);  
  
**commit**;

### 接口说明

#### EventBusFactory

事件总线(单例模式)

|  |  |
| --- | --- |
| **方法** | **static** EventBusFactory getInstance() |
|  | 获取事件总线实例 |
| **方法** | **void** register(**final** Class group, **final** EventListener listener) |
|  | 注册事件监听器 |
| **方法** | **void** post(**final** Event event) |
|  | 发布事件 |
| **方法** | **void** clearListeners(**final** Class group) |
|  | 清除指定事件 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **分类** | **参数名** | **类型** | **必录** | **说明** |
| Event<T> | group | Class | 是 | 事件监听类 |
| businessId | String | 否 | 业务凭证流水号(持久化时必填) |
| businessType | String | 否 | 业务类型(持久化时必填) |
| payload | T | 是 | 具体事件 |
| callback | EventCallback<T> | 否 | 事件回调，请在事件处理逻辑中自行实现 |
| isPersistent | boolean | 否 | 为true时持久化 |
| retry | int | 否 | 默认重试5次，-1时无限重试，(无限重试默认间隔离10分钟) |
| eventContext | EventContext | 否 | 持久化事件恢复时使用，业务无需关心此属性 |

#### EventListener<T>

事件监听器

|  |  |
| --- | --- |
| **方法** | **void** listen(T event) |
|  | 监听指定事件 |

#### EventCallback <T>

异步发送时需实现此接口

|  |  |
| --- | --- |
| **方法** | **void** onCompletion(T t) |
|  | 事件完成时的回调 |

### 例子

以事务消息为例，首先需要生成指定事件：

@Setter  
@Getter  
@ToString(callSuper = **true**)  
**public class** KafkaProduceEvent **extends** MqEvent {  
 */\*\* 主键用来负载均衡 \*/* **private** String **key**;  
 */\*\* 分区 \*/* **private int partition** = -1;

**public** KafkaProduceEvent(){  
 **this**.setType(MqTypeEnum.***KAFKA***.getMemo());  
 **this**.setGroup(KafkaProduceEvent.**class**);  
 }  
}

再编定事件监听器：

@Component  
**public class** KafkaProduceEventListener **implements**

EventListener<KafkaProduceEvent> {  
  
 @Autowired  
 **private** KafkaMessageProducer **producer**;  
 @Autowired  
 **private** EventAdviceService **service**; *// 事件包装器*  
  
 @Subscribe  
 @AllowConcurrentEvents  
 @Override  
 **public void** listen(KafkaProduceEvent event) {  
 **try** {  
 *// 前处理* **service**.before(event);  
 *// 发送消息* **producer**.send(event);  
 *// 后处理* **service**.after(event);  
 } **catch** (Throwable e) {  
 *// 失败处理* **service**.fail(event, e);  
 }  
 }  
  
 @Override  
 **public** String getIdentity() {  
 **return** Constants.***KAFKA***; *// 标识事件*

}  
}

最后注册事件：

@Component  
**public class** KafkaEventListenerManager {  
  
 */\*\* 消息需发送事件监听器 \*/* @Autowired  
 **private** KafkaProduceEventListener **kafkaProduceEventListener**;  
  
 @PostConstruct  
 **public void** init() **throws** Exception {  
 *// 注册事件* EventBusFactory.*getInstance*().register(KafkaProduceEvent.**class**

, **kafkaProduceEventListener**);

}  
}

这样，就可以发送指定事件了：

Event e = **new** Event();  
e.setPayload(**"testEvent"**);  
EventBusFactory.*getInstance*().post(e);

# 配置介绍

非必填项请谨慎修改。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **属性名** | **类型** | **必填** | **默认值** | **描述** |
| sar.name | String | 是 | - | 系统唯一标识，不能与其它系统重复(恢复Job使用) |
| event.persistent.lock.optimistic | boolean | 否 | false | 在持久化事件的更新操作时，是否采用乐观锁，采用乐观锁会加大消息重复几率。默认为悲观锁 |
| event.recovery.start.day | int | 否 | 7 | 天，持久化事件最长恢复时间(恢复Job使用) |
| event.recovery.delay.second | int | 否 | 600 | 秒，新增持久化事件延迟多久恢复(恢复Job使用) |
| event.recovery.limit | int | 否 | 1000 | 条，持久化事件一次恢复数量(恢复Job使用) |
| event.infinite.retry.interval | int | 否 | 600 | 秒，持久化事件无限重试时的重试间隔(发送事件时retry设为-1为无限重试，同时持久化事件在上报错误时也会用到此值)。 |
| event.storage.tag | int | 否 | 10 | 持久化事件存储时的分片数 |
| event.retry.times | int | 否 | 5 | 持久化事件默认重试次数 |
| event.failed.report.dest | String | 否 | event.failed.report.topic | 持久化事件达到最大重试次数时上报监控平台的队列地址(通过MQ上报) |