qtask

提供稳定，可靠的任务管理机制，直到任务被成功处理。

实时任务：将任务存入 DB,并放入消息队列。业务系统订阅消息，处理逻辑，根据需要结束任务。未结束的任务超时后自动放入队列，继续处理。

延时任务：将任务存入 DB,超时后放入消息队列。业务系统订阅消息，处理逻辑，根据需要结束任务。未结束的任务超时后自动放入队列，继续处理。

特性:

√ 支持实时任务

√ 支持延时任务

√ 任务自动存储，防丢失，支持 mysql,oracle

√ 定时将任务放入队列

√ 过期任务定时清理

√ 一行代码安装任务表

√ 基于 hydra 构建

一、准备

创建任务表

1. 编译 `qtask`

```
~/work/bin$ go install github.com/micro-plat/qtask #mysql

或

~/work/bin$ go install -tags "oracle" github.com/micro-plat/qtask # oracle
```

2. 运行命令

`qtask [注册中心地址] [平台名称]` 即可将 `qtask` 需要的表创建到`/平台/var/db/db` 配置对应的数据库中。

```
~/work/bin$ qtask zk://192.168.0.109 mall #根据/mall/var/db/db创建数据库

或
```

```
~/work/bin$ qtask zk://192.168.0.109 mall mdb #根据/mall/var/db/mdb创建数据库
```

二、编码

1. 绑定服务

将定时扫描和清除路由动态注册到引用qtask任务的hydra框架服务中,并设置定时扫描任务时间

1.定时清除任务 凌晨定时清除达到删除时间的任务

2.定时扫描任务 将达到执行期限的任务处理失败,并扫描需要执行的任务,发送消息队列(例如redis)

```
app.Initializing(func(c component.IContainer) error {
    qtask.Bind(app,10)  //每隔10秒将未完成的任务放入队列,app:*hydra.MicroApp
}
```

2. 创建任务

任务主要分为实时任务和延时任务两种

1.实时任务
需要立即执行的任务.支持事物,需要注意当传参为事物时,需要在事物提交之后,再执行发送消息队列函数

```
// 业务逻辑

// 创建实时任务，将任务保存到数据库(状态为等待处理)并放入消息队列
// taskID:任务编号
// fcallback:发送当前任务消息队列函数
taskID, fcallback, err:=qtask.Create(c,"订单绑定任务",map[string]interface{}{
    "order_no":"8973097380045"
},3600,"GCR:ORDER:BIND")

err:=fcallback(c)
```

2.延迟任务 需要延迟处理的任务,先将任务创建在数据库中,达到首次执行任务时间,再执行.

```
//创建延时任务，将任务保存到数据库(状态为等待处理),超时后放入消息队列
// taskID:任务编号
taskID, err:=qtask.Delay(c,"订单绑定任务",map[string]interface{}{
    "order_no":"8973097380045"
},60,3600,"GCR:ORDER:BIND", qtask.WithDeadline(86400))
```

3.可选择传入参数

> qtask.WithDeadline 秒数，设置任务执行截止时间,默认为604800(7天) qtask.WithDeleteDeadline 秒数，设置任务删除截止时间,默认为0.如果没有设置该参数,当任务执行成功,删除时间为当前时间,如果未执行成功,删除时间为当前时间加604800(7天), qtask.WithMaxCount 次数，设置任务最多执行次数,默认为100 qtask.WithOrderNO 外部业务单号，设置外部业务单号,便于查询同一业务单号任务,默认空

3. 处理 `GCR:ORDER:BIND` 消息

接收到消息后,先处理任务,再执行业务逻辑,最后结束任务,如果业务逻辑处理失败,会重试5次

```
func OrderBind(ctx *context.Context) (r interface{}) {
    //检查输入参数...

    //业务处理前调用，修改任务状态为处理中(超时前未调用qtask.Finish，任务会被重新放入队列)
    qtask.Processing(ctx,ctx.Request.GetInt64("task_id"))


    //处理业务逻辑...


    //业务处理成功，修改任务状态为完成(任务不再放入队列),并修改删除截止时间
    qtask.Finish(ctx,ctx.Request.GetInt64("task_id"))
}
```

三、其它

1. 自定义数据库名，队列名

```
qtask.Config("order_db","rds_queue") //配置数据库名，队列名
```

2. 使用不同的数据库

使用 mysql 数据库

```
 go install 或 go install -tags "mysql"
```

使用 oracle 数据库

```
 go install -tags "oracle"
```

完整示例