分布式计算框架(DC4C)

厉华

版本修订

版本	日期	修订人	内容
0.1.0	2015-05-28	厉华	创建文档 编写第四章 开发接口 编写附录 A 任务调度引擎 数据库表结构

目录索引

1 概述 5

1.1 简介 5

1.2 体系结构 5

1.3 功能和优势 5

2 架构原理 6

2.1 基础平台架构 6

2.2 任务调度引擎 6

3 安装部署 7

3.1 单机部署 7

3.2 集群部署 7

4 开发接口 8

4.1 用户节点接口 8

4.1.1 环境类 8

4.1.2 同步发起任务类 10

4.1.3 异步发起任务类 12

4.1.4 获取执行反馈类 13

4.1.5 低层函数类 15

4.1.6 其它类 16

4.2 计算节点接口 17

4.2.1 日志操作类 17

4.2.2 反馈信息类 18

4.3 任务调度引擎接口 18

4.3.1 高层函数 19

4.3.2 低层函数 21

4.4 示例 25

4.4.1 单任务 25

4.4.2 批量任务 27

4.4.3 多批量任务 30

4.4.4 DAG 调度多批量任务（从配置文件载入配置） 34
4.4.5 DAG 调度多批量任务（从数据载入配置） 37
5 附件 41
5.1 附件 A.任务调度引擎 数据库表结构 41
5.1.1 计划表 41
5.1.2 批量表 41
5.1.3 批量依赖关系表 42
5.1.4 批量任务表 42

1 概述

1.1 简介

杭州

1.2 体系结构

杭州

1.3 功能和优势

2 架构原理

2.1 基础平台架构

2.2 任务调度引擎

3 安装部署

3.1 单机部署

3.2 集群部署

4 开发接口

4.1 用户节点接口

用户节点部署的应用调用的函数接口，主要用于同步、异步发起任务、批量任务、多批量任务，并检查任务执行后的反馈信息等功能。

4.1.1 环境类

4.1.1.1DC4CInitEnv

函数原型： int DC4CInitEnv(struct Dc4cApiEnv **ppenv , char *rservers_ip_port);

函数描述： 初始化批量任务环境

函数说明： 申请 Dc4cApiEnv 批量任务环境结构所需内存，并初始化，让(*ppenv)指向该结构。

解析注册节点地址信息，保存到环境结构中。注册节点地址信息可以是”ip:port”格式，也可以带多个地址”ip1:port1,ip2:port2,…”。如果该参数为 NULL，尝试取系统环境变量”DC4C_RSERVERS_IP_PORT”。

函数返回值为 0 表示成功，非 0 表示失败。

代码示例：

```
struct Dc4cApiEnv *penv = NULL ;
...
nret = DC4CInitEnv( & penv , "127.0.0.1:12001" ) ;
```

4.1.1.2DC4CCleanEnv

函数原型： void DC4CCleanEnv(struct Dc4cApiEnv **ppenv);

函数描述： 清理批量任务环境

函数说明： 清理并释放 Dc4cApiEnv 环境结构所需内存。

代码示例：

```
struct Dc4cApiEnv *penv = NULL ;
...
```

```
nret = DC4CInitEnv( & penv , "127.0.0.1:12001" ) ;
...
DC4CCleanEnv( & penv );
```

4.1.1.3DC4CSetTimeout

函数原型： void DC4CSetTimeout(struct Dc4cApiEnv *penv , int timeout);

函数描述： 设置公共超时时间

函数说明： struct Dc4cApiEnv *penv 批量任务环境结构指针

int timeout 公共超时时间（单位：秒）。实际任务超时时间取公共超时时间与配置任务时间较大值

4.1.1.4DC4CGetTimeout

函数原型： int DC4CGetTimeout(struct Dc4cApiEnv *penv);

函数描述： 得到公共超时时间

函数说明： struct Dc4cApiEnv *penv 批量任务环境结构指针

返回公共超时时间

4.1.1.5DC4CSetOptions

函数原型： void DC4CSetOptions(struct Dc4cApiEnv *penv , unsigned long options);

函数描述： 设置公共选项

函数说明： struct Dc4cApiEnv *penv 批量任务环境结构指针

unsigned long options 公共选项

备　　注： 选项由如下宏组合而成

DC4C_OPTIONS_INTERRUPT_BY_APP 允许应用返回 status 非 0 时中断后续任务执行，缺省为不中断

DC4C_OPTIONS_BIND_CPU 尝试在任务程序开始执行前绑定 CPU

代码示例：

```
DC4CSetOptions( penv , DC4C_OPTIONS_INTERRUPT_BY_APP |
DC4C_OPTIONS_BIND_CPU );
```

4.1.1.6DC4CGetOptions

函数原型： unsigned long DC4CGetOptions(struct Dc4cApiEnv *penv);

函数描述： 得到公共选项

函数说明： struct Dc4cApiEnv *penv 批量任务环境结构指针

返回公共选项

4.1.2 同步发起任务类

4.1.2.1DC4CDoTask

函数原型： int DC4CDoTask(struct Dc4cApiEnv *penv , char *program_and_params , int timeout);

函数描述： 分发单任务，同步等待反馈

函数说明： struct Dc4cApiEnv *penv 批量任务环境结构指针

char *program_and_params 执行命令行

int timeout 配置超时时间（单位：秒）

函数返回值为 0 表示成功，非 0 表示失败

代码示例：

```
struct Dc4cApiEnv *penv = NULL ;
...
nret = DC4CInitEnv( & penv , "127.0.0.1:12001" ) ;
...
nret = DC4CDoTask( penv , "dc4c_test_worker_sleep 5" , 60 ) ;
...
DC4CCleanEnv( & penv );
```

4.1.2.2 DC4CDoBatchTasks

函数原型： int DC4CDoBatchTasks(struct Dc4cApiEnv *penv , int workers_count , struct Dc4cBatchTask *a_tasks , int tasks_count);

函数描述： 分发批量任务，同步等待全部反馈

函数说明： struct Dc4cApiEnv *penv 批量任务环境结构指针

int workers_count 并发数量。当为-1 时尽可能大并发

struct Dc4cBatchTask *a_tasks 任务集合，每个任务包含执行命令行和配置超时时间

int tasks_count 任务数量

函数返回值为 0 表示成功，非 0 表示失败

代码示例：

```
struct Dc4cApiEnv *penv = NULL ;
struct Dc4cBatchTask  *tasks_array = NULL ;
...
nret = DC4CInitEnv( & penv , "127.0.0.1:12001" ) ;
...
tasks_array = (struct Dc4cBatchTask *)malloc( sizeof(struct Dc4cBatchTask) * tasks_count ) ;
for( i = 0 , p_task = tasks_array ; i < tasks_count ; i++ , p_task++ )
{
    strcpy( p_task->program_and_params , "dc4c_test_worker_sleep -5" );
    p_task->timeout = DC4CGetTimeout(penv) ;
}
...
nret = DC4CDoBatchTasks( penv , workers_count , tasks_array , tasks_count ) ;
...
free( tasks_array );
...
DC4CCleanEnv( & penv );
```

4.1.3 异步发起任务类

4.1.3.1DC4CBeginBatchTasks

函数原型： int DC4CBeginBatchTasks(struct Dc4cApiEnv *penv , int workers_count , struct Dc4cBatchTask *a_tasks , int tasks_count);

函数描述： 开始批量任务

函数说明： （同同步发起批量任务 DC4CDoBatchTasks）

函数返回值为 0 表示成功，非 0 表示失败

4.1.3.2DC4CPerformMultiBatchTasks

函数原型： int DC4CPerformMultiBatchTasks(struct Dc4cApiEnv **ppenvs , int envs_count , struct Dc4cApiEnv **ppenv , int *p_remain_envs_count);

函数描述： 多路复用处理多批量任务

函数说明： struct Dc4cApiEnv **ppenvs 批量任务环境结构指针集合

int envs_count 批量任务环境数量

struct Dc4cApiEnv **ppenv 输出当前完成或失败批量任务

int *p_remain_envs_count 当前还剩下批量任务数量

函数返回值为 0 表示有批量任务结束了，DC4C_INFO_NO_UNFINISHED_ENVS 为所有批量任务都结束了，非 0 表示失败

代码示例：

```
struct Dc4cApiEnv **ppenvs = NULL ;
struct Dc4cBatchTask  *tasks_array = NULL ;
...
ppenvs = (struct Dc4cApiEnv**)malloc( sizeof(struct Dc4cApiEnv*) *
envs_count ) ;
...
for( i= 0 ; i < envs_count ; i++ )
{
    nret = DC4CInitEnv( & (ppenvs[i]) , NULL ) ;
    ...
    tasks_array = (struct Dc4cBatchTask *)malloc( sizeof(struct
```

```
Dc4cBatchTask) * tasks_count ) ;
    for( j = 0 , p_task = tasks_array ; j < tasks_count ; j++ , p_task++ )
    {
        strcpy( p_task->program_and_params , "dc4c_test_worker_sleep
-5" );
        p_task->timeout = DC4CGetTimeout(penv) ;
    }
    nret = DC4CBeginBatchTasks( ppenvs[i] , workers_count , tasks_array ,
tasks_count ) ;
    free( tasks_array ) ;
}
...
while(1)
{
    nret = DC4CPerformMultiBatchTasks( ppenvs , envs_count , & penv , &
remain_envs_count ) ;
    if( nret == DC4C_INFO_NO_UNFINISHED_ENVS )
        break;
    else if( nret )
        return nret;
}
...
for( i= 0 ; i < envs_count ; i++ )
{
    nret = DC4CCleanEnv( & (ppenvs[i]) ) ;
}
```

4.1.4 获取执行反馈类

4.1.4.1DC4CGetTask*

函数原型： int DC4CGetTask*(struct Dc4cApiEnv *penv , …);

函数描述： 得到单任务反馈信息

函数说明： struct Dc4cApiEnv *penv 批量任务环境结构指针

Ip 计算节点 IP

Port 计算节点 PORT

Tid 任务 ID

ProgramAndParams 执行命令行

Timeout 配置的超时时间（单位：秒）

Elapse 实际执行时间（单位：秒）

Error 分发和反馈任务中发生的错误，0 为没有错误

Status 程序执行返回状态，即 waitpid 得到的 status

Info 应用返回信息，最长 1024 字节

函数返回值为 0 表示成功，非 0 表示失败

代码示例：

```
    char        *ip = NULL ;
    long        port ;
    char        *tid = NULL ;
    char        *program_and_params = NULL ;
    int         timeout ;
    int         elapse ;
    int         error ;
    int         status ;
    char        *info = NULL ;
...

    DC4CGetTaskIp( penv , & ip );
    DC4CGetTaskPort( penv , & port );
    DC4CGetTaskTid( penv , & tid );
    DC4CGetTaskProgramAndParams( penv , & program_and_params );
    DC4CGetTaskTimeout( penv , & timeout );
    DC4CGetTaskElapse( penv , & elapse );
    DC4CGetTaskError( penv , & error );
    DC4CGetTaskStatus( penv , & status );
    DC4CGetTaskInfo( penv , & info );
    printf( "Task-[%s][%ld]-[%s][%s][%d][%d]-[%d][%d][%s]\n" , ip ,
port , tid , program_and_params , timeout , elapse , error ,
WEXITSTATUS(status) , info );
```

4.1.4.2 DC4CGetBatchTasks*

函数原型： int DC4CGetBatchTasks*(struct Dc4cApiEnv *penv , int index , ...);

函数描述： 得到批量任务反馈信息

函数说明： struct Dc4cApiEnv *penv 批量任务环境结构指针集合

int index 任务索引，从 0 开始

（其它同 DC4CGetTask*）

函数返回值为 0 表示成功，非 0 表示失败

4.1.5 低层函数类

4.1.5.1DC4CQueryWorkers

函数原型： int DC4CQueryWorkers(struct Dc4cApiEnv *penv);

函数描述： 向注册节点查询空闲计算节点信息

函数说明： struct Dc4cApiEnv *penv 批量任务环境结构指针

函数返回值为 0 表示成功，非 0 表示失败

4.1.5.2DC4CSetBatchTasksFds

函数原型： int DC4CSetBatchTasksFds(struct Dc4cApiEnv *penv , fd_set *read_fds , fd_set *write_fds , fd_set *expect_fds , int *p_max_fd);

函数描述： 从已连接计算节点会话和连接空闲计算节点中会话（发送执行命令请求），设置描述字集合

函数说明： struct Dc4cApiEnv *penv 批量任务环境结构指针

fd_set *read_fds 可读事件描述字集合

fd_set *write_fds （目前未用）

fd_set *expect_fds （目前未用）

int *p_max_fd 可读事件描述字集合中的最大描述字值

函数返回值为 0 表示成功，非 0 表示失败

4.1.5.3DC4CSelectBatchTasksFds

函数原型： int DC4CSelectBatchTasksFds(fd_set *p_read_fds , fd_set *write_fds ,

fd_set *expect_fds , int *p_max_fd , int select_timeout);

函数描述： 多路复用等待可读描述字集合事件

函数说明： struct Dc4cApiEnv *penv 批量任务环境结构指针

fd_set *read_fds 可读事件描述字集合

fd_set *write_fds （目前未用）

fd_set *expect_fds （目前未用）

int *p_max_fd 可读事件描述字集合中的最大描述字值

int select_timeout 等待事件最长时间

函数返回值为 0 表示成功，非 0 表示失败

4.1.5.4DC4CProcessBatchTasks

函数原型： int DC4CProcessBatchTasks(struct Dc4cApiEnv *penv , fd_set *p_read_fds , fd_set *write_fds , fd_set *expect_fds);

函数描述： 处理可读描述字集合事件，接收执行命令响应、接收分发程序请求和发送响应

函数说明： struct Dc4cApiEnv *penv 批量任务环境结构指针

fd_set *read_fds 可读事件描述字集合

fd_set *write_fds （目前未用）

fd_set *expect_fds （目前未用）

函数返回值为 0 表示成功，非 0 表示失败

4.1.6 其它类

4.1.6.1DC4CResetFinishedTasksWithError

函数原型： void DC4CResetFinishedTasksWithError(struct Dc4cApiEnv *penv);

函数描述： 重置批量任务中程序执行返回失败的任务为待处理

函数说明： struct Dc4cApiEnv *penv 批量任务环境结构指针

4.1.6.2 DC4CGetUnusedWorkersCount

函数原型： int DC4CGetUnusedWorkersCount(struct Dc4cApiEnv *penv);

函数描述： 得到上次向注册节点查询空闲计算节点还未使用信息

函数说明： struct Dc4cApiEnv *penv 批量任务环境

4.2 计算节点接口

用户节点分发过来到计算节点的应用调用的函数接口，主要信息处理完后的设置反馈信息等功能。

4.2.1 日志操作类

4.2.1.1 DC4CSetAppLogFile

函数原型： void DC4CSetAppLogFile(char *program);

函数描述： 如果计算节点应用使用 DC4C 日志函数库，可以调用该函数重定向日志文件

函数说明： char *program 应用名

备　　注： 重定向后的日志文件名格式为”dc4c_wserver_(序号)_(IP:PORT).(应用名).log”

4.2.2 反馈信息类

4.2.2.1DC4CFormatReplyInfo

函数原型： int DC4CFormatReplyInfo(char *format , ...);

函数描述： 用于计算节点应用反馈信息给用户节点，类型为字符串，最长 1024 字符

函数说明： char *format 格式化串

备　　注： （同 snprintf）

4.2.2.2DC4CSetReplyInfo

函数原型： int DC4CSetReplyInfo(char *str);

函数描述： 用于计算节点应用反馈信息给用户节点，类型为字符串，最长 1024 字符

函数说明： （同上，非格式化串直接复制版本）

4.2.2.3DC4CSetReplyInfoEx

函数原型： int DC4CSetReplyInfoEx(char *buf , int len);

函数描述： 用于计算节点应用反馈信息给用户节点，类型为字符串，最长 1024 字符

函数说明： （同上，非格式化串带长度版本）

4.3 任务调度引擎接口

任务调度引擎是用户节点任务流控制的函数接口，替代手工编写代码，实现根据任务依赖关系，自动、快速调度任务的功能。

任务调度引擎封装了原生的同步发起任务函数接口，读入外部配置文件或数据库中的配置，按序执行线性、树形等批量任务。

目前只实现了有向无环图（DAG）数据结构的任务流模型，几乎可以配置出任意流程的任务树。

4.3.1 高层函数

4.3.1.1DC4CLoadDagScheduleFromFile

函数原型： int DC4CLoadDagScheduleFromFile(struct Dc4cDagSchedule **pp_sched , char *pathfilename , int options);

函数描述： 从外部配置文件中载入 DAG 流程模型的任务树配置

函数说明： 申请 Dc4cDagScheduleDAG 流程模型环境结构所需内存，并初始化，从配置文件载入 DAG 流程模型任务数配置，让(*pp_sched)指向该结构。

char *pathfilename 带路径的外部配置文件名

int options 传递给用户节点接口的 DC4CSetOptions 用于设置全局选项

函数返回值为 0 表示成功，非 0 表示失败

备　　注： 最多配置 1000 个批量任务，1000 个批量任务关系，每个批量任务最多配置 1000 个任务

代码示例：

```
struct Dc4cDagSchedule     *p_sched = NULL ;
...
nret = DC4CLoadDagScheduleFromFile( & p_sched ,
"/home/calvin/etc/test.dag_schedule" ,
DC4C_OPTIONS_INTERRUPT_BY_APP ) ;
...
DC4CUnloadDagSchedule( & p_sched );
```

4.3.1.2DC4CLoadDagScheduleFromDatabase

函数原型： int DC4CLoadDagScheduleFromDatabase(struct Dc4cDagSchedule

**pp_sched , char *schedule_name , int options);

函数描述： 从数据库中载入 DAG 流程模型的任务树配置

函数说明： 申请 Dc4cDagSchedule DAG 流程模型环境结构所需内存，并初始化，从数据库载入 DAG 流程模型任务数配置，让(*pp_sched)指向该结构。

char *schedule_name 计划名

int options 传递给用户节点接口的 DC4CSetOptions 用于设置全局选项

函数返回值为 0 表示成功，非 0 表示失败

备　　注： （数据库配置表详见附件 A）

代码示例：

```
struct Dc4cDagSchedule     *p_sched = NULL ;
...
nret = DC4CLoadDagScheduleFromDatabase( & p_sched ,
"test_dag_schedule" , DC4C_OPTIONS_INTERRUPT_BY_APP ) ;
...
DC4CUnloadDagSchedule( & p_sched );
```

4.3.1.3 DC4CExecuteDagSchedule

函数原型： int DC4CExecuteDagSchedule(struct Dc4cDagSchedule *p_sched , char *rservers_ip_port);

函数描述： 从数据库中载入 DAG 流程模型的任务树配置

函数说明： struct Dc4cDagSchedule *p_sched　DAG 流程模型环境结构指针

char *rservers_ip_port 注册服务器地址

函数返回值为 0 表示成功，非 0 表示失败

代码示例：

```
struct Dc4cDagSchedule     *p_sched = NULL ;
...
nret = DC4CLoadDagScheduleFromDatabase( & p_sched ,
"test_dag_schedule" , DC4C_OPTIONS_INTERRUPT_BY_APP ) ;
...
nret = DC4CExecuteDagSchedule( p_sched , NULL ) ;
...
DC4CUnloadDagSchedule( & p_sched );
```

4.3.1.4DC4CUnloadDagSchedule

函数原型： void DC4CUnloadDagSchedule(struct Dc4cDagSchedule **pp_sched);

函数描述： 从数据库中载入 DAG 流程模型的任务树配置

函数说明： struct Dc4cDagSchedule **pp_sched　DAG 流程模型环境结构指针的地址

4.3.1.5DC4CLogDagSchedule

函数原型： void DC4CLogDagSchedule(struct Dc4cDagSchedule *p_sched);

函数描述： 把 DAG 流程模型的任务树配置输出到日志

函数说明： struct Dc4cDagSchedule *p_sched　DAG 流程模型环境结构指针

4.3.2 低层函数

4.3.2.1DC4CLoadDagScheduleFromStruct

函数原型： int DC4CLoadDagScheduleFromStruct(struct Dc4cDagSchedule **pp_sched , dag_schedule_configfile *p_config , int options);

函数描述： 从大配置结构体载入 DAG 流程模型的任务树配置

函数说明： 申请 Dc4cDagScheduleDAG 流程模型环境结构所需内存，并初始化，从大配置结构体载入 DAG 流程模型任务数配置，让(*pp_sched)指向该结构。

int options 传递给用户节点接口的 DC4CSetOptions 用于设置全局选项

函数返回值为 0 表示成功，非 0 表示失败

备　　注： 此函数被 DC4CLoadDagScheduleFromFile 、DC4CLoadDagScheduleFromDatabase 调用，实现载入配置。

4.3.2.2DC4CInitDagSchedule

函数原型： int DC4CInitDagSchedule(struct Dc4cDagSchedule *p_sched , char *schedule_name , char *schedule_desc);

函数描述： 初始化 DAG 流程模型任务树

函数说明： struct Dc4cDagSchedule *p_sched DAG 流程模型结构指针

char *schedule_name 计划名

char *schedule_desc 计划描述

函数返回值为 0 表示成功，非 0 表示失败

4.3.2.3DC4CCleanDagSchedule

函数原型： void DC4CCleanDagSchedule(struct Dc4cDagSchedule *p_sched);

函数描述： 清理 DAG 流程模型任务树

函数说明： struct Dc4cDagSchedule *p_sched DAG 流程模型结构指针

4.3.2.4DC4CAllocDagBatch

函数原型： struct Dc4cDagBatch *DC4CAllocDagBatch(struct Dc4cDagSchedule *p_sched , char *batch_name , char *batch_desc , int view_pos_x , int view_pos_y);

函数描述： 创建一个 DAG 批量树节点，并初始化

函数说明： struct Dc4cDagSchedule *p_sched DAG 流程模型结构指针

char *batch_name 批量名

char *batch_desc 批量描述

int view_pos_x 批量节点图形配置坐标 X

int view_pos_y 批量节点图形配置坐标 Y

4.3.2.5DC4CFreeDagBatch

函数原型： BOOL DC4CFreeDagBatch(void *pv);

函数描述： 清理一个 DAG 批量树节点，并释放之

函数说明： void *pv　DAG 流程模型结构指针

4.3.2.6DC4CLinkDagBatch

函数原型： int DC4CLinkDagBatch(struct Dc4cDagSchedule *p_sched , struct Dc4cDagBatch *p_parent_batch , struct Dc4cDagBatch *p_batch);

函数描述： 挂接一个批量节点到 DAG 流程模型树上

函数说明： struct Dc4cDagSchedule *p_sched　DAG 流程模型结构指针

struct Dc4cDagBatch *p_parent_batch 父级批量节点

struct Dc4cDagBatch *p_batch 要挂接的批量节点

4.3.2.7DC4CSetDagBatchTasks

函数原型： void DC4CSetDagBatchTasks(struct Dc4cDagBatch *p_batch , struct Dc4cBatchTask *a_tasks , int tasks_count);

函数描述： 设置批量任务集合到批量节点上

函数说明： struct Dc4cDagBatch *p_batch 要挂接的批量节点

struct Dc4cBatchTask *a_tasks 批量任务集合

int tasks_count 批量任务数量

4.3.2.8DC4CGetDagBatchApiEnvPPtr

函数原型： struct Dc4cApiEnv **DC4CGetDagBatchApiEnvPPtr(struct Dc4cDagBatch *p_batch);

函数描述： 返回批量节点里的批量任务环境结构指针的地址

函数说明： struct Dc4cDagBatch *p_batch 要挂接的批量节点

4.3.2.9DC4CGetDagBatchBeginDatetimeStamp

函数原型： void DC4CGetDagBatchBeginDatetimeStamp (struct Dc4cDagBatch *p_batch , char begin_datetime[19+1] , long *p_begin_datetime_stamp);

函数描述： 返回批量节点里的批量开始日期时间戳

函数说明： struct Dc4cDagBatch *p_batch 批量节点

char begin_datetime[19+1] 存放返回的日期时间字符串

long *p_begin_datetime_stamp 存放返回的日期时间戳的地址

4.3.2.10 DC4CGetDagBatchEndDatetimeStamp

函数原型： void DC4CGetDagBatchEndDatetimeStamp(struct Dc4cDagBatch *p_batch , char end_datetime[19+1] , long *p_end_datetime_stamp);

函数描述： 返回批量节点里的批量结束日期时间戳

函数说明： struct Dc4cDagBatch *p_batch 批量节点

char end_datetime[19+1] 存放返回的日期时间字符串

long *p_begin_datetime_stamp 存放返回的日期时间戳的地址

4.3.2.11 DC4CGetDagBatchProgress

函数原型： void DC4CGetDagBatchProgress(struct Dc4cDagBatch *p_batch , int *p_progress);

函数描述： 返回批量节点里的进度

函数说明： struct Dc4cDagBatch *p_batch 批量节点

int *p_progress 存放返回的进度的地址

4.3.2.12 DC4CGetDagBatchResult

函数原型： void DC4CGetDagBatchResult(struct Dc4cDagBatch *p_batch , int *p_result);

函数描述： 返回批量节点里的结果

函数说明： struct Dc4cDagBatch *p_batch 批量节点

int *p_progress 存放返回的结果的地址

4.4 示例

4.4.1 单任务

源代码

```
#include "dc4c_api.h"

/* for testing
time ./dc4c_test_master 192.168.6.54:12001,192.168.6.54:12002
"dc4c_test_worker_sleep 3"
*/

int main( int argc , char *argv[] )
{
    struct Dc4cApiEnv *penv = NULL ;

    char        *ip = NULL ;
    long        port ;
    char        *tid = NULL ;
    char        *program_and_params = NULL ;
    int         timeout ;
    int         elapse ;
    int         error ;
    int         status ;
    char        *info = NULL ;
```

```
	int		nret = 0 ;

	DC4CSetAppLogFile( "dc4c_test_master" );
	SetLogLevel( LOGLEVEL_DEBUG );

	if( argc == 1 + 2 )
	{
		nret = DC4CInitEnv( & penv , argv[1] ) ;
		if( nret )
		{
			printf( "DC4CInitEnv failed[%d]\n" , nret );
			return 1;
		}
		else
		{
			printf( "DC4CInitEnv ok\n" );
		}

		DC4CSetTimeout( penv , 60 );
		DC4CSetOptions( penv , DC4C_OPTIONS_INTERRUPT_BY_APP );

		nret = DC4CDoTask( penv , argv[2] , DC4CGetTimeout(penv) ) ;
		if( nret )
		{
			printf( "DC4CDoTask failed[%d]\n" , nret );
		}
		else
		{
			printf( "DC4CDoTask ok\n" );
		}

		DC4CGetTaskIp( penv , & ip );
		DC4CGetTaskPort( penv , & port );
		DC4CGetTaskTid( penv , & tid );
		DC4CGetTaskProgramAndParams( penv , & program_and_params );
		DC4CGetTaskTimeout( penv , & timeout );
		DC4CGetTaskElapse( penv , & elapse );
		DC4CGetTaskError( penv , & error );
		DC4CGetTaskStatus( penv , & status );
		DC4CGetTaskInfo( penv , & info );
		printf( "Task-[%s][%ld]-[%s][%s][%d][%d]-[%d][%d][%s]\n" , ip , port , tid ,
program_and_params , timeout , elapse , error , WEXITSTATUS(status) , info );

		DC4CCleanEnv( & penv );
```

```
		printf( "DC4CCleanEnv ok\n" );
	}
	else
	{
		printf( "USAGE : dc4c_test_master rserver_ip:rserver_port
program_and_params\n" );
		exit(7);
	}

	return 0;
}
```

4.4.2 批量任务

源代码

```
#include "dc4c_api.h"

/* for testing
time ./dc4c_test_batch_master 192.168.6.54:12001,192.168.6.54:12002 2 3
"dc4c_test_worker_sleep 1" "dc4c_test_worker_sleep 2" "dc4c_test_worker_sleep 3"
time ./dc4c_test_batch_master 192.168.6.54:12001,192.168.6.54:12002 4 -10
"dc4c_test_worker_sleep 10"
time ./dc4c_test_batch_master 192.168.6.54:12001,192.168.6.54:12002 -1 -100
"dc4c_test_worker_sleep -10"
*/

int main( int argc , char *argv[] )
{
	struct Dc4cApiEnv *penv = NULL ;
	struct Dc4cBatchTask	*tasks_array = NULL ;
	struct Dc4cBatchTask	*p_task = NULL ;
	int		tasks_count ;
	int		workers_count ;
	int		repeat_task_flag ;
	int		i ;

	char		*ip = NULL ;
	long		port ;
	char		*tid = NULL ;
	char		*program_and_params = NULL ;
	int		timeout ;
	int		elapse ;
```

```
int             error ;
int             status ;
char            *info = NULL ;

int             nret = 0 ;

DC4CSetAppLogFile( "dc4c_test_batch_master" );
SetLogLevel( LOGLEVEL_DEBUG );

if( argc >= 1 + 3 )
{
        nret = DC4CInitEnv( & penv , argv[1] ) ;
        if( nret )
        {
                printf( "DC4CInitEnv failed[%d]\n" , nret );
                return 1;
        }
        else
        {
                printf( "DC4CInitEnv ok\n" );
        }

        DC4CSetTimeout( penv , 60 );
        DC4CSetOptions( penv , DC4C_OPTIONS_INTERRUPT_BY_APP );

        workers_count = atoi(argv[2]) ;
        tasks_count = atoi(argv[3]) ;

        if( tasks_count < 0 )
        {
                repeat_task_flag = 1 ;
                tasks_count = -tasks_count ;
        }
        else
        {
                repeat_task_flag = 0 ;
        }

        if( workers_count < 0 )
        {
                workers_count = tasks_count ;
        }

        tasks_array = (struct Dc4cBatchTask *)malloc( sizeof(struct Dc4cBatchTask) *
```

```
tasks_count ) ;
	if( tasks_array == NULL )
	{
		printf( "alloc failed , errno[%d]\n" , errno );
		return 1;
	}
	memset( tasks_array , 0x00 , sizeof(struct Dc4cBatchTask) * tasks_count );

	for( i = 0 , p_task = tasks_array ; i < tasks_count ; i++ , p_task++ )
	{
		if( repeat_task_flag == 1 )
			strcpy( p_task->program_and_params , argv[4] );
		else
			strcpy( p_task->program_and_params , argv[4+i] );
		p_task->timeout = DC4CGetTimeout(penv) ;
	}

	nret = DC4CDoBatchTasks( penv , workers_count , tasks_array , tasks_count ) ;
	free( tasks_array );
	if( nret )
	{
		printf( "DC4CDoBatchTasks failed[%d]\n" , nret );
	}
	else
	{
		printf( "DC4CDoBatchTasks ok\n" );
	}

	printf( "tasks_count[%d] worker_count[%d] - prepare_count[%d]
running_count[%d] finished_count[%d]\n" , DC4CGetTasksCount(penv) ,
DC4CGetWorkersCount(penv) , DC4CGetPrepareTasksCount(penv) ,
DC4CGetRunningTasksCount(penv) , DC4CGetFinishedTasksCount(penv) );

	tasks_count = DC4CGetTasksCount( penv ) ;
	for( i = 0 ; i < tasks_count ; i++ )
	{
		DC4CGetBatchTasksIp( penv , i , & ip );
		DC4CGetBatchTasksPort( penv , i , & port );
		DC4CGetBatchTasksTid( penv , i , & tid );
		DC4CGetBatchTasksProgramAndParams( penv , i , & program_and_params );
		DC4CGetBatchTasksTimeout( penv , i , & timeout );
		DC4CGetBatchTasksElapse( penv , i , & elapse );
		DC4CGetBatchTasksError( penv , i , & error );
		DC4CGetBatchTasksStatus( penv , i , & status );
```

```
                DC4CGetBatchTasksInfo( penv , i , & info );
                printf( "Task[%d]-[%s][%ld]-[%s][%s][%d][%d]-[%d][%d][%s]\n" , i ,
ip , port , tid , program_and_params , timeout , elapse , error , WEXITSTATUS(status) , info );
            }

            DC4CCleanEnv( & penv );
            printf( "DC4CCleanEnv ok\n" );
        }
        else
        {
            printf( "USAGE : dc4c_test_batch_master rserver_ip:rserver_port,...
workers_count task_count program_and_params_1 ...\n" );
            exit(7);
        }

        return 0;
}
```

4.4.3 多批量任务

源代码

```
#include "dc4c_api.h"

/* for testing
time ./dc4c_test_multi_batch_master 192.168.6.54:12001,192.168.6.54:12002 -1 -100
"dc4c_test_worker_sleep -10"
*/

int main( int argc , char *argv[] )
{
        struct Dc4cApiEnv **ppenvs = NULL ;
        int             envs_count ;
        struct Dc4cApiEnv *penv = NULL ;
        int             remain_envs_count ;
        struct Dc4cBatchTask   task ;
        struct Dc4cBatchTask   *tasks_array = NULL ;
        struct Dc4cBatchTask   *p_task = NULL ;
        int             tasks_count ;
        int             workers_count ;
        int             repeat_task_flag ;
        int             i ;
```

```
	char		*ip = NULL ;
	long		port ;
	char		*tid = NULL ;
	char		*program_and_params = NULL ;
	int		timeout ;
	int		elapse ;
	int		error ;
	int		status ;
	char		*info = NULL ;

	int		nret = 0 ;

	DC4CSetAppLogFile( "dc4c_test_multi_batch_master" );
	SetLogLevel( LOGLEVEL_DEBUG );

	if( argc >= 1 + 3 )
	{
		envs_count = 2 ;
		ppenvs = (struct Dc4cApiEnv**)malloc( sizeof(struct Dc4cApiEnv*) *
envs_count ) ;
		if( ppenvs == NULL )
		{
			printf( "malloc failed[%d] , errno[%d]\n" , nret , errno );
			return 1;
		}
		memset( ppenvs , 0x00 , sizeof(struct Dc4cApiEnv*) * envs_count );

		nret = DC4CInitEnv( & (ppenvs[0]) , argv[1] ) ;
		if( nret )
		{
			printf( "DC4CInitEnv failed[%d]\n" , nret );
			return 1;
		}
		else
		{
			printf( "DC4CInitEnv ok\n" );
		}

		DC4CSetOptions( ppenvs[0] , DC4C_OPTIONS_INTERRUPT_BY_APP );

		nret = DC4CInitEnv( & (ppenvs[1]) , argv[1] ) ;
		if( nret )
		{
			printf( "DC4CInitEnv failed[%d]\n" , nret );
```

```
		return 1;
	}
	else
	{
		printf( "DC4CInitEnv ok\n" );
	}

	DC4CSetOptions( ppenvs[1] , DC4C_OPTIONS_INTERRUPT_BY_APP );

	workers_count = atoi(argv[2]) ;
	tasks_count = atoi(argv[3]) ;

	if( tasks_count < 0 )
	{
		repeat_task_flag = 1 ;
		tasks_count = -tasks_count ;
	}
	else
	{
		repeat_task_flag = 0 ;
	}

	if( workers_count < 0 )
	{
		workers_count = tasks_count ;
	}

	strcpy( task.program_and_params , "dc4c_test_worker_sleep 3" );
	task.timeout = 0 ;

	nret = DC4CBeginBatchTasks( ppenvs[0] , workers_count , & task , 1 ) ;
	if( nret )
	{
		printf( "DC4CBeginBatchTasks failed[%d] , errno[%d]\n" , nret , errno );
		return 1;
	}

	tasks_array = (struct Dc4cBatchTask *)malloc( sizeof(struct Dc4cBatchTask) *
tasks_count ) ;
	if( tasks_array == NULL )
	{
		printf( "alloc failed , errno[%d]\n" , errno );
		return 1;
	}
```

```
	memset( tasks_array , 0x00 , sizeof(struct Dc4cBatchTask) * tasks_count );

	for( i = 0 , p_task = tasks_array ; i < tasks_count ; i++ , p_task++ )
	{
		if( repeat_task_flag == 1 )
			strcpy( p_task->program_and_params , argv[4] );
		else
			strcpy( p_task->program_and_params , argv[4+i] );
		p_task->timeout = 0 ;
	}

	nret = DC4CBeginBatchTasks( ppenvs[1] , workers_count , tasks_array ,
tasks_count ) ;
	free( tasks_array );
	if( nret )
	{
		printf( "DC4CBeginBatchTasks failed[%d] , errno[%d]\n" , nret , errno );
		return 1;
	}

	while(1)
	{
		nret = DC4CPerformMultiBatchTasks( ppenvs , envs_count , & penv , &
remain_envs_count ) ;
		if( nret == DC4C_INFO_NO_UNFINISHED_ENVS )
		{
			printf( "DC4CPerformMultiBatchTasks ok , all is done\n" );

			break;
		}
		else if( nret )
		{
			printf( "DC4CPerformMultiBatchTasks failed[%d]\n" , nret );
		}
		else
		{
			printf( "DC4CPerformMultiBatchTasks ok\n" );

		}

		tasks_count = DC4CGetTasksCount( penv ) ;
		for( i = 0 ; i < tasks_count ; i++ )
		{
			DC4CGetBatchTasksIp( penv , i , & ip );
```

```
                    DC4CGetBatchTasksPort( penv , i , & port );
                    DC4CGetBatchTasksTid( penv , i , & tid );
                    DC4CGetBatchTasksProgramAndParams( penv , i , &
program_and_params );
                    DC4CGetBatchTasksTimeout( penv , i , & timeout );
                    DC4CGetBatchTasksElapse( penv , i , & elapse );
                    DC4CGetBatchTasksError( penv , i , & error );
                    DC4CGetBatchTasksStatus( penv , i , & status );
                    DC4CGetBatchTasksInfo( penv , i , & info );
                   printf( "Task[%d]-[%s][%ld]-[%s][%s][%d][%d]-[%d][%d][%s]\n" ,
i , ip , port , tid , program_and_params , timeout , elapse , error , WEXITSTATUS(status) ,
info );
                }

                if( nret == DC4C_ERROR_APP )
                    DC4CResetFinishedTasksWithError( penv );
                else if( nret )
                    break;
            }

            DC4CCleanEnv( & (ppenvs[0]) );
            DC4CCleanEnv( & (ppenvs[1]) );
            free( ppenvs );
            printf( "DC4CCleanEnv ok\n" );
        }
        else
        {
            printf( "USAGE : dc4c_test_multi_batch_master rserver_ip:rserver_port,...
workers_count task_count program_and_params_1 ...\n" );
            exit(7);
        }

        return 0;
}
```

4.4.4 DAG 调度多批量任务（从配置文件载入配置）

配置文件

```
/*
                                    _BEGIN_
                                  /         \
                            BATCH_A1          BATCH_B1
```

```
                "dc4c_test_worker_sleep 1"                      "dc4c_test_worker_sleep 3"
                   /                   \                                  |
             BATCH_A21                 BATCH_A22                          BATCH_B2
  "dc4c_test_worker_sleep 4" "dc4c_test_worker_sleep 5"       "dc4c_test_worker_sleep
6"
                   \                   /                                  |
                          BATCH_A3                                       /
                  "dc4c_test_worker_sleep 2"                            /
                  "dc4c_test_worker_sleep 1"                          /
                                        \                   /
                                                _END_
*/
{
    "schedule" : {
        "schedule_name" : "test_schedule_name" ,
        "schedule_desc" : "test_schedule_desc"
    } ,
    "batches" : {
        "batches_info" : [
            { "batch_name":"BATCH_A1", "batch_desc":"", "view_pos_x":0,
"view_pos_y":0, "tasks": [
                { "program_and_params":"dc4c_test_worker_sleep 1", "timeout":60" }
            ] } ,
            { "batch_name":"BATCH_A21", "batch_desc":"", "view_pos_x":0,
"view_pos_y":0, "tasks": [
                { "program_and_params":"dc4c_test_worker_sleep 4", "timeout":60" }
            ] } ,
            { "batch_name":"BATCH_A22", "batch_desc":"", "view_pos_x":0,
"view_pos_y":0, "tasks": [
                { "program_and_params":"dc4c_test_worker_sleep 5", "timeout":60" }
            ] } ,
            { "batch_name":"BATCH_A3", "batch_desc":"", "view_pos_x":0,
"view_pos_y":0, "tasks": [
                { "program_and_params":"dc4c_test_worker_sleep 2", "timeout":60" } ,
                { "program_and_params":"dc4c_test_worker_sleep 1", "timeout":60" }
            ] } ,
            { "batch_name":"BATCH_B1", "batch_desc":"", "view_pos_x":0,
"view_pos_y":0, "tasks": [
                { "program_and_params":"dc4c_test_worker_sleep 3", "timeout":60" }
            ] } ,
            { "batch_name":"BATCH_B2", "batch_desc":"", "view_pos_x":0,
"view_pos_y":0, "tasks": [
                { "program_and_params":"dc4c_test_worker_sleep 6", "timeout":60" }
            ] }
```

```
        ],
        "batches_direction" : [
            { "from_batch":"", "to_batch":"BATCH_A1" },
            { "from_batch":"", "to_batch":"BATCH_B1" },
            { "from_batch":"BATCH_A1", "to_batch":"BATCH_A21" },
            { "from_batch":"BATCH_A1", "to_batch":"BATCH_A22" },
            { "from_batch":"BATCH_A21", "to_batch":"BATCH_A3" },
            { "from_batch":"BATCH_A22", "to_batch":"BATCH_A3" },
            { "from_batch":"BATCH_B1", "to_batch":"BATCH_B2" },
            { "from_batch":"BATCH_A3", "to_batch":"" },
            { "from_batch":"BATCH_B2", "to_batch":"" }
        ]
    }
}
```

源代码

```
#include "dc4c_api.h"
#include "dc4c_tfc_dag.h"

/* for testing
time ./dc4c_test_tfc_dag_master rservers_ip_port *.dag_schedule"
*/

int main( int argc , char *argv[] )
{
    struct Dc4cDagSchedule    *p_sched = NULL ;

    int             nret = 0 ;

    DC4CSetAppLogFile( "dc4c_test_tfc_dag_master" );
    SetLogLevel( LOGLEVEL_DEBUG );

    if( argc == 1 + 2 )
    {
        nret = DC4CLoadDagScheduleFromFile( & p_sched , argv[2] ,
DC4C_OPTIONS_INTERRUPT_BY_APP ) ;
        if( nret )
        {
            printf( "DC4CLoadDagScheduleFromFile failed[%d]\n" , nret );
            return 1;
        }
        else
        {
            printf( "DC4CLoadDagScheduleFromFile ok\n" );
        }
```

```
        DC4CLogDagSchedule( p_sched );

        nret = DC4CExecuteDagSchedule( p_sched , argv[1] ) ;
        if( nret )
        {
                printf( "DC4CExecuteDagSchedule failed[%d]\n" , nret );
                return 1;
        }
        else
        {
                printf( "DC4CExecuteDagSchedule ok\n" );
        }

        DC4CUnloadDagSchedule( & p_sched );
        printf( "DC4CCleanEnv ok\n" );
    }
    else
    {
        printf( "USAGE : dc4c_test_tfc_dag_master rservers_ip_port .dag_schedule\n" );
        exit(7);
    }

    return 0;
}
```

4.4.5 DAG 调度多批量任务（从数据载入配置）

数据库配置

```
truncate table dag_schedule ;
truncate table dag_batches_info ;
truncate table dag_batches_tasks ;
truncate table dag_batches_direction ;

INSERT INTO dag_schedule VALUES ( 1 , 'test_schedule_name' , 'test_schedule_desc' );

INSERT INTO dag_batches_info VALUES ( 'test_schedule_name' , 'BATCH_A1' , '' , 0 , 0 );
INSERT INTO dag_batches_info VALUES ( 'test_schedule_name' , 'BATCH_B1' , '' , 0 , 0 );
INSERT INTO dag_batches_info VALUES ( 'test_schedule_name' , 'BATCH_A21' , '' , 0 , 0 );
INSERT INTO dag_batches_info VALUES ( 'test_schedule_name' , 'BATCH_A22' , '' , 0 , 0 );
INSERT INTO dag_batches_info VALUES ( 'test_schedule_name' , 'BATCH_A3' , '' , 0 , 0 );
INSERT INTO dag_batches_info VALUES ( 'test_schedule_name' , 'BATCH_B2' , '' , 0 , 0 );
```

```
INSERT INTO dag_batches_tasks VALUES ( 'test_schedule_name' , 'BATCH_A1' ,
'dc4c_test_worker_sleep 1' , 60 );
INSERT INTO dag_batches_tasks VALUES ( 'test_schedule_name' , 'BATCH_A21' ,
'dc4c_test_worker_sleep 4' , 60 );
INSERT INTO dag_batches_tasks VALUES ( 'test_schedule_name' , 'BATCH_A22' ,
'dc4c_test_worker_sleep 5' , 60 );
INSERT INTO dag_batches_tasks VALUES ( 'test_schedule_name' , 'BATCH_A3' ,
'dc4c_test_worker_sleep 2' , 60 );
INSERT INTO dag_batches_tasks VALUES ( 'test_schedule_name' , 'BATCH_A3' ,
'dc4c_test_worker_sleep 1' , 60 );
INSERT INTO dag_batches_tasks VALUES ( 'test_schedule_name' , 'BATCH_B1' ,
'dc4c_test_worker_sleep 3' , 60 );
INSERT INTO dag_batches_tasks VALUES ( 'test_schedule_name' , 'BATCH_B2' ,
'dc4c_test_worker_sleep 6' , 60 );

INSERT INTO dag_batches_direction VALUES ( 'test_schedule_name' , '' , 'BATCH_A1' );
INSERT INTO dag_batches_direction VALUES ( 'test_schedule_name' , '' , 'BATCH_B1' );
INSERT INTO dag_batches_direction VALUES ( 'test_schedule_name' , 'BATCH_A1' ,
'BATCH_A21' );
INSERT INTO dag_batches_direction VALUES ( 'test_schedule_name' , 'BATCH_A1' ,
'BATCH_A22' );
INSERT INTO dag_batches_direction VALUES ( 'test_schedule_name' , 'BATCH_A21' ,
'BATCH_A3' );
INSERT INTO dag_batches_direction VALUES ( 'test_schedule_name' , 'BATCH_A22' ,
'BATCH_A3' );
INSERT INTO dag_batches_direction VALUES ( 'test_schedule_name' , 'BATCH_B1' ,
'BATCH_B2' );
INSERT INTO dag_batches_direction VALUES ( 'test_schedule_name' , 'BATCH_A3' , '' );
INSERT INTO dag_batches_direction VALUES ( 'test_schedule_name' , 'BATCH_B2' , '' );
```

源代码

```
#include "dc4c_api.h"
#include "dc4c_tfc_dag.h"
#include "IDL_dag_schedule.dsc.ESQL.eh"

/* for testing
time ./dc4c_test_tfc_dag_master rservers_ip_port *.dag_schedule"
*/

int main( int argc , char *argv[] )
{
	struct Dc4cDagSchedule	*p_sched = NULL ;

	int		nret = 0 ;
```

```
	DSCDBCONN( "0.0.0.0" , 18432 , "calvin" , "calvin" , "calvin" );
	if( SQLCODE )
	{
		printf( "DSCDBCONN failed , SQLCODE[%d][%s][%s]\n" , SQLCODE , SQLSTATE ,
SQLDESC );
		return 1;
	}
	else
	{
		printf( "DSCDBCONN ok\n" );
	}

	DC4CSetAppLogFile( "dc4c_test_tfc_dag_master" );
	SetLogLevel( LOGLEVEL_DEBUG );

	if( argc == 1 + 2 )
	{
		nret = DC4CLoadDagScheduleFromDatabase( & p_sched , argv[2] ,
DC4C_OPTIONS_INTERRUPT_BY_APP ) ;
		if( nret )
		{
			printf( "DC4CLoadDagScheduleFromDatabase failed[%d]\n" , nret );
			return 1;
		}
		else
		{
			printf( "DC4CLoadDagScheduleFromDatabase ok\n" );
		}

		DC4CLogDagSchedule( p_sched );

		nret = DC4CExecuteDagSchedule( p_sched , argv[1] ) ;
		if( nret )
		{
			printf( "DC4CExecuteDagSchedule failed[%d]\n" , nret );
			return 1;
		}
		else
		{
			printf( "DC4CExecuteDagSchedule ok\n" );
		}

		DC4CUnloadDagSchedule( & p_sched );
```

```
            printf( "DC4CCleanEnv ok\n" );
        }
        else
        {
            printf( "USAGE : dc4c_test_tfc_dag_master rservers_ip_port .dag_schedule\n" );
            exit(7);
        }

        DSCDBDISCONN();
        printf( "DSCDBDISCONN ok\n" );

        return 0;
}
```

5 附件

5.1 附件 A.任务调度引擎 数据库表结构

5.1.1 计划表

TABLE NAME		dag_schedule	
TYPE	LENGTH	NAME	NOT NULL
INT	4	order_index	NOT NULL
STRING	64	schedule_name	NOT NULL
STRING	256	schedule_desc	
UNIQUE INDEX1 : order_index UNIQUE INDEX2 : schedule_name			

5.1.2 批量表

TABLE NAME		dag_batches_info	
TYPE	LENGTH	NAME	NOT NULL
STRING	64	schedule_name	NOT NULL
STRING	64	batch_name	NOT NULL
STRING	256	batch_desc	
INT	4	view_pos_x	
INT	4	view_pos_y	
UNIQUE INDEX1 : schedule_name , batch_name			

5.1.3 批量依赖关系表

TABLE NAME		dag_batches_direction	
TYPE	LENGTH	NAME	NOT NULL
STRING	64	schedule_name	NOT NULL
STRING	64	from_batch	NOT NULL
STRING	64	to_batch	
INDEX1 : schedule_name , from_batch			

5.1.4 批量任务表

TABLE NAME		dag_batches_tasks	
TYPE	LENGTH	NAME	NOT NULL
STRING	64	schedule_name	NOT NULL
STRING	64	batch_name	NOT NULL
INT	4	order_index	NOT NULL
STRING	256	program_and_params	NOT NULL
INT	4	timeout	NOT NULL
UNIQUE INDEX1 : schedule_name , batch_name , order_index			