日日是Oracle APEX

Oracle APEXを使った作業をしていて、気の付いたところを忘れないようにメモをとります。

2022年2月18日金曜日

AQを使ってバックグランドでジョブを実行する - 通知の代わりにジョブを使う

以前に書いた記事でAQの通知を使ってバックグランド処理を行うようにしていのたですが、やはりバックグランド処理の並列度は制御したいので、通知ではなくDBMS_SCHEDULERでジョブを作成して処理する方法を試してみました。

特にAlways Freeのインスタンスはスレッドが２本なので、バックグラウンドで長時間実行されるジョブが何本もあると、画面操作が進まなくなってしまいます。

以下、作業手順になります。作成済みの通知、キュー、キュー表は削除しておきます。また、アプリケーションは作成済みのものを改修します。

最初にキュー表とキューを作成します。通知は使わないので、シングル・コンシューマで十分です。タイプjob_message_tが作成済みであれば、キュー表とキューの作成と開始を行うPL/SQLブロックのみ実行します。

```
/*
 * タイプの作成
 * 単純にJSONをキューに投入する。
 * 21cからJSON型が使用できる。
 */
create or replace type job_message_t as object(
    body varchar2(4000)
);
/

/*
 * キュー表 JOB_Q_TAB、キューJOB_IN_Q、JOB_OUT_Qの作成と開始。
 */
begin
    -- 通知を使わない場合は、multiple_consumersはFALSEとする。
    dbms_aqadm.create_queue_table(
        queue_table => 'job_q_tab'
        ,queue_payload_type => 'job_message_t'
        ,multiple_consumers => FALSE
    );
    dbms_aqadm.create_queue(
        queue_name => 'job_in_q'
        ,queue_table => 'job_q_tab'
```

```
    );
    dbms_aqadm.create_queue(
        queue_name => 'job_out_q'
        ,queue_table => 'job_q_tab'
    );
    dbms_aqadm.start_queue(
        queue_name => 'job_in_q'
    );
    dbms_aqadm.start_queue(
        queue_name => 'job_out_q'
    );
end;
/
```

prepare_q_qt_single.sql hosted with ❤️ by GitHub view raw

通知で呼び出されるプロシージャ**exec_job**を、DBMS_SCHDULERのジョブとして呼び出すためのラッパーとなるプロシージャ**exec_job0**を作成します。DBMS_SCHEDULER.CREATE_PROGRAMを呼び出して、**exec_job0**を呼び出すプログラム**MEX_SAMPLE_PROC**を作成します。続いてDBMS_SCHEDULER.CREATE_JOBを呼び出して、ジョブ**MEX_WORKER_JOB01**を作成し開始します。

```
/*
 * 引数なしでexec_jobを呼び出すためのラッパー
 */
create or replace procedure exec_job0
as
begin
    exec_job(null,null,null,null,null);
end;
/

/*
 * スケジューラー・ジョブより呼び出すプログラムを作成する。
 */
begin
    dbms_scheduler.create_program(
        program_name     => 'MEX_SAMPLE_PROC'
        , program_type   => 'STORED_PROCEDURE'
        , program_action => 'APEXDEV.EXEC_JOB0'
        , number_of_arguments => 0
        , enabled        => true
        , comments       => 'Run exec_job without params.'
  );
end;
/

/*
 * ジョブを作成し、開始する。
```

```
 */
begin
    dbms_scheduler.create_job(
        job_name        => 'MEX_WORKER_JOB01'
        , program_name  => 'MEX_SAMPLE_PROC'
        , job_style     => 'LIGHTWEIGHT'
        , start_date    => systimestamp
        , repeat_interval => 'FREQ=SECONDLY;INTERVAL=10'
        , enabled         => true
    );
end;
/
```

create_program_and_job.sql hosted with ❤ by GitHub view raw

作成済みのアプリケーションに含まれる、ジョブのリクエストをエンキューするコードを変更します。変更するコードの全体が以下になりますが、変更点は１行だけです。

dequeue_options.consumer_nameの指定をコメント・アウトしています。

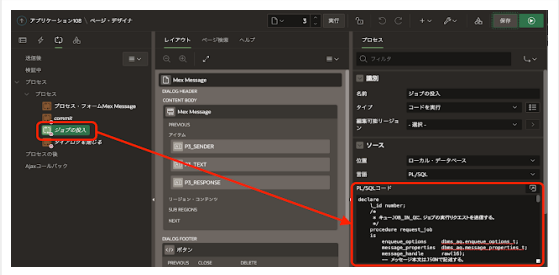
```
declare
    l_id number;
    /*
     * キューJOB_IN_Qに、ジョブの実行リクエストを送信する。
     */
    procedure request_job
    is
        enqueue_options     dbms_aq.enqueue_options_t;
        message_properties  dbms_aq.message_properties_t;
        message_handle      raw(16);
        -- メッセージ本文はJSONで記述する。
        message             apexdev.job_message_t;
        l_job_request       json_object_t;
    begin
        -- メッセージは送信時にコミットする。
        -- そうしないとページ・プロセスがすべて終了するまでエンキューされない。
        enqueue_options.visibility := dbms_aq.immediate;
        -- 通常はFIFO、dbms_aq.topを指定するとLIFO。
        enqueue_options.sequence_deviation := dbms_aq.top;
        -- バックグラウンド・ジョブのパラメータとなる値をJSONで記述する。
        -- 21cであれば、JSON型を直接扱うことができる。
        l_job_request := new json_object_t;
        l_job_request.put('sender', :P3_SENDER);
        l_job_request.put('text', :P3_TEXT);
        message := apexdev.job_message_t(
            l_job_request.to_string()
        );
        -- キューに投入するメッセージを、表の主キーIDに関連づける。
```

```
    l_id := :P3_ID;
    message_properties.correlation := l_id;
    -- ジョブの実行要求を送信する。
    dbms_aq.enqueue(
        queue_name          => 'apexdev.job_in_q'
        ,enqueue_options    => enqueue_options
        ,message_properties => message_properties
        ,payload            => message
        ,msgid              => message_handle
    );
end;
/*
 * キューJOB_OUT_Qから、ジョブの完了リクエストを受信する。
 */
procedure confirm_job_completion
is
    dequeue_options     dbms_aq.dequeue_options_t;
    message_properties  dbms_aq.message_properties_t;
    message_handle      raw(16);
    message             apexdev.job_message_t;
    l_job_response      json_object_t;
    l_response varchar2(200);
    -- デキューのタイムアウトの例外を定義する。
    dequeue_timeout exception;
    pragma exception_init(dequeue_timeout, -25228);
begin
    -- ジョブの完了を最長5秒待つ。
    dequeue_options.navigation := dbms_aq.first_message;
    /*
     * 通知を使わずキューがシングル・コンシューマの場合は
     * consumer_nameは指定しない。
     */
    -- dequeue_options.consumer_name := 'APEXDEV';
    dequeue_options.correlation := l_id;
    /*
     * ジョブに時間がかかることが分かっている場合は、画面からは
     * ジョブを投入するだけで、完了を待機する必要はない。
     * つまりJOB_OUT_Qを使った処理自体が不要になる。
     *
     * dbms_aq.NO_WAITを指定すると、エンキューはされていない
     * ため、dequeue_timeoutが発生する。
     */
    dequeue_options.wait := 5; -- 5秒だけ待つ。
    l_response := '';
    begin
        :AI_SUCCESS_MESSAGE := 'JOB_SUCCESS_MESSAGE';
        dbms_aq.dequeue(
```

```
                queue_name          =>      'apexdev.job_out_q',
                dequeue_options     =>      dequeue_options,
                message_properties  =>      message_properties,
                payload             =>      message,
                msgid               =>      message_handle
            );
            l_job_response := json_object_t.parse(message.body);
            l_response := l_job_response.get_string('response');
        exception
            when dequeue_timeout then
                :AI_SUCCESS_MESSAGE := 'JOB_TIMEOUT_MESSAGE';
        end;
        :P3_RESPONSE := l_response;
    end;
begin
    request_job;
    confirm_job_completion;
end;
```

enqueue_dequeue_msg_single.sql hosted with ❤ by GitHub view raw

ページ番号３にあるプロセス**ジョブの投入**の**PL/SQLコード**を置き換えます。



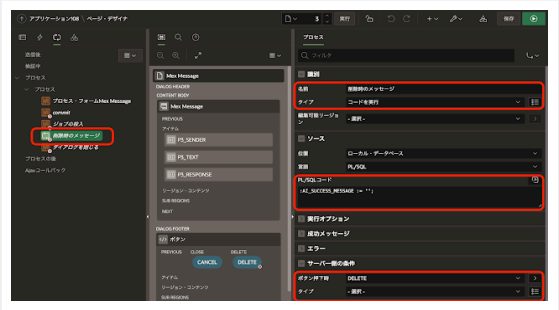
以上で変更は完了です。

同じジョブを同時に実行することで、並列度を上げることができます。

すこしアプリケーションに不具合がありました。行の削除時に表示されるメッセージを抑制するためのプロセスを追加します。

プロセス**ジョブの投入**の直前で実行されるプロセスメッセージの初期化を作成します。PL/SQLコードには、以下を記述します。

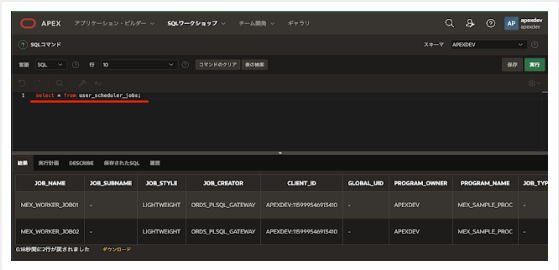
```
:AI_SUCCESS_MESSAGE := '';
```



以上で、バックグラウンドの処理が通知ではなくジョブで行われるようになりました。アプリケーションの動作に違いはありません。(ジョブの処理に時間がかかるようになっています)。

作成されたジョブは、ビューUSER_SCHEDULER_JOBSより確認できます。

select * from user_scheduler_jobs;



作成したDBMS_SCHEDULERジョブやプログラムを削除するコードを書いています。

```
/*
 * ジョブの停止。
 */
begin
    dbms_scheduler.stop_job(
        'MEX_WORKER_JOB01'
    );
end;
/

/*
 * ジョブの削除。
 */
begin
    dbms_scheduler.drop_job(
        'MEX_WORKER_JOB01'
    );
end;
/

/*
 * プログラムの削除。
 */
begin
```

```
    dbms_scheduler.drop_program(
        'MEX_SAMPLE_PROC'
    );
end;
/
```

cleanup_scheduler_job.sql hosted with ❤️ by GitHub view raw

以上になります。

Oracle APEXのアプリケーション作成の参考になれば幸いです。

完

Yuji N. 時刻: 13:40

共有

自己紹介

Yuji N.

日本オラクル株式会社に勤務していて、Oracle APEXのGroundbreaker Advocateを拝命しました。
こちらの記事につきましては、免責事項の参照をお願いいたします。

詳細プロフィールを表示