```
;;      Jetton 代幣主合約
;;      繁體中文註解版 by Y.C.
;;      所有中文均經過人腦編譯，不存在 AI 製造的奇怪語法，並可直接複製貼上取代原有的 jetton-minter.fc 服用。



{-
      pragma version 指明運行此智能合約所需的 TVM 版本，為確保合約能支持整體合約邏輯能在適當的 TVM 上運行；
      大部份情況下，TON 都會使用最新版 TVM 運行，以確保安全性、效能和功能的最優化；然而，在測試和開發環境/某些節點或私有鏈的分叉
      ，仍可能在使用舊版本的TVM。
-}
#pragma version >=0.4.3;



#include "stdlib.fc";              ;; 讓文件調用 FunC 標準工具庫代碼
#include "op-codes.fc";            ;; 讓文件調用操作碼定義
#include "workchain.fc";           ;; 讓文件調用區塊鏈地址檢查工具
#include "jetton-utils.fc";        ;; 讓文件調用 Jetton 標準程序
#include "gas.fc";                 ;; 讓文件調用 gas fee 計算的相關函數



{-
      storage#_ ;; 這段 TL-B 說明了主合約在 c4 的資料存儲結構
            total_supply:Coins ;; 代幣的總供應量，使用 Coins 類型表示
            admin_address:MsgAddress ;; 合約管理員地址，有特權修改合約參數
            next_admin_address:MsgAddress ;; 更改管理員過渡時使用，將於當前管理員確認後成為新管理員地址
            jetton_wallet_code:^Cell ;; 存儲與代幣分支錢包相關的核心代碼
            metadata_uri:^Cell ;; 存儲本合約代幣的元數據，  包括描述及圖標URL等
      = Storage; ;; TL-B 本身只是高階的描述語法，合約結構的設計藍圖，而這裡結束了 Storage 的定義
-}



{-
      load_data() 用作讀取合約中的存儲數據，空括號代表不需要傳入參數使用；
      inline 代表這是一個內部函數，並不能從外來訊息直接調用，只能讓合約內的函數引用；
      (int, slice, slice, cell, cell) 表示這函數將回傳的五個數值類型，對應 data 內的回傳值；
-}
(int, slice, slice, cell, cell) load_data() inline {
         slice ds = get_data().begin_parse(); ;; 回傳智能合約的 c4 storage，並由 cell 轉換為 slice
         var data = (
             ds~load_coins(),    ;; 在 ds 這個 slice 上調用 load_coins() 函數並取得 total_supply
             ds~load_msg_addr(), ;; 同樣調用 stdlib.fc 當中的函數並取得 admin_address
             ds~load_msg_addr(), ;; 根據定義好的結合排列，這個就是 next_admin_address
             ds~load_ref(),      ;; 從剩餘的 slice 繼續載入下一個引用，這資料是 jetton_wallet_code
             ds~load_ref()       ;; 繼續載入儲存在合約中的 metadata_uri 所包含的特定資料格式 cell
         );
         ds.end_parse();  ;; 檢查 slice 是否為空，如果不為空則拋出異常，確認已讀完 ds 全部數據
         return data;
}
```

```
{-
    () save_data(int total_supply, slice admin_address, slice next_admin_address, cell jetton_wallet_code,
    cell metadata_uri) 用於更新智能合約中的存儲數據,
    前置的空括號表示此函數沒有回傳值, 右方括號則表示調用函數所需要輸入的五個數值以及其所屬的類型;
    impure 表示此函數會修改存儲數據、發送消息或消耗 gas, inline 表示這是內部函數。
-}
() save_data(int total_supply, slice admin_address, slice next_admin_address, cell jetton_wallet_code, cell
metadata_uri) impure inline {
        set_data( ;; 將指定的 cell 設置為合約的 c4 數據, 更新合約的持久存儲
            begin_cell() ;; 創建一個新 builder 構建和組裝資料並存儲數據
            .store_coins(total_supply)        ;; 將 total_supply 值存入並更新 builder 繼續儲存動作
            .store_slice(admin_address)       ;; 將 admin_address 值存入並更新 builder 繼續儲存動作
            .store_slice(next_admin_address) ;; 將 next_admin_address 值存入 builder 繼續動作
            .store_ref(jetton_wallet_code)    ;; 將 jetton_wallet_code 值存入 builder 繼續動作
            .store_ref(metadata_uri)          ;; 將 metadata_uri 值存入並更新 builder 繼續儲存動作
            .end_cell()  ;; 結束 builder 構建, 並封裝成 cell
        );
}



{-
    send_to_jetton_wallet 向指定的 Jetton 錢包地址發送信息操作並確保合約存儲狀態一致。
-}
() send_to_jetton_wallet(slice to_address, cell jetton_wallet_code, int ton_amount, cell master_msg, int
need_state_init) impure inline {

        raw_reserve(ONE_TON / 100, RESERVE_REGULAR);
            ;; 預先保留 0.01 TON 作為存儲費用, RESERVE_REGULAR 在 stdlib.fc 中定義為常用模式

        cell state_init = calculate_jetton_wallet_state_init(to_address, my_address(), jetton_wallet_code);
    ;; 以 cell 定義帳戶的起始狀態 @jetton-uilts

        slice to_wallet_address = calculate_jetton_wallet_address(state_init);
            ;; 以起始狀態定義帳戶的錢包地址 @jetton-uilts

            ;; 開始構建 cell 並使用 builder functions 加插資料
        var msg = begin_cell()
        .store_msg_flags_and_address_none(BOUNCEABLE) ;; 立 flag 標註信息可回彈
        .store_slice(to_wallet_address)       ;; 插入信息接收方地址
        .store_coins(ton_amount);             ;; 傳送的 TON 量
               if (need_state_init) { ;; need_state_init = 1 / 需要狀態初始化
                   msg = msg.store_statinit_ref_and_body_ref(state_init, master_msg);
                         ;; 包含狀態初始化信息和信息主體的完整信息
               } else {  ;; need_state_init = 0 / 不需要初始化
                   msg = msg.store_only_body_ref(master_msg);
                         ;; 僅包含信息主體
               }

        send_raw_message(msg.end_cell(), SEND_MODE_PAY_FEES_SEPARATELY | SEND_MODE_BOUNCE_ON_ACTION_FAIL);
    ;; 在 if / else 條件決定用什麼 builder 及插入什麼資訊後才結束構建 msg, 並發送已構建的消息 cell；發送者將會支付這條信息的費用, 如果發送失敗（例如目標合約拒絕消息或發生錯誤）, 則回彈信息。
```

```
}


;; 內部信息的唯一入口，統一接收並分流內部操作流程及指令
() recv_internal(int msg_value, cell in_msg_full, slice in_msg_body) impure {
    slice in_msg_full_slice = in_msg_full.begin_parse(); ;; 以 slice 載入完整信息
    int msg_flags = in_msg_full_slice~load_msg_flags(); ;; 從完整信息加載 flag 檢測信息種類

    if (is_bounced(msg_flags)) { ;; 如果信息是回彈的，表示操作失敗，並進行特定處理程序
        in_msg_body~skip_bounced_prefix(); ;; 跳過回彈信息前綴
            ifnot (in_msg_body~load_op() == op::internal_transfer) {
                return ();
            } ;; 如操作碼不是 op::internal_transfer（內部轉移操作）則退出函數結束處理
        in_msg_body~skip_query_id();
        ;; 跳過用於標識特定請求或操作的查詢 ID（query_id），因為回彈信息的主要處理失敗後的操作
        int jetton_amount = in_msg_body~load_coins(); ;; 加載回彈信息中的 jetton_amount
        (int total_supply, slice admin_address, slice next_admin_address,
            cell jetton_wallet_code, cell metadata_uri) = load_data(); ;; 加載合約當前數據
        save_data(total_supply - jetton_amount, admin_address, next_admin_address,
            jetton_wallet_code, metadata_uri); ;; 更新合約數據狀態
        ;; 不是常規轉帳動作，而是 mint，所以回彈操作需要從 total supply 減去未成功創建的 token 金額
        return (); ;; 收貔，合約操作流程結束（出口1）
    }
    slice sender_address = in_msg_full_slice~load_msg_addr(); ;; 從內部信息加載信息發送者地址
    int fwd_fee_from_in_msg = in_msg_full_slice~retrieve_fwd_fee(); ;; 獲取信息的轉發費用金額
    int fwd_fee = get_original_fwd_fee(MY_WORKCHAIN, fwd_fee_from_in_msg);
        ;; 根據當前工作鏈（MY_WORKCHAIN）計算準確的轉發費用，以特定工作鏈的費用結構，計算精確的轉發成本
    (int op, int query_id) = in_msg_body~load_op_and_query_id();
        ;; 從信息主體（in_msg_body）提取操作碼（op）和查詢 ID（query_id）
    (int total_supply, slice admin_address, slice next_admin_address, cell jetton_wallet_code, cell
metadata_uri) = load_data(); ;; 加載合約當前的狀態數據

     if (op == op::mint) {
        ;; 檢查管理員權限，發行並增加 Jetton token 的供應量
        throw_unless(error::not_owner, equal_slices_bits(sender_address, admin_address));
        ;; 如果發送者不是管理員，則拋出異常報錯
        slice to_address = in_msg_body~load_msg_addr(); ;; 從信息主體提取目標地址
        check_same_workchain(to_address); ;; 檢查目標地址是否位於當前工作鏈
        int ton_amount = in_msg_body~load_coins(); ;; 此信息所接收的 TON 量
        cell master_msg = in_msg_body~load_ref(); ;; 這個 "ref" 涉及 Jetton 代幣轉移細節
        in_msg_body.end_parse(); ;; 完結解讀，對應 recv_internal 第一行的 begin_parse

        slice master_msg_slice = master_msg.begin_parse(); ;; 解讀代幣轉移細節
        throw_unless(error::invalid_op, master_msg_slice~load_op() == op::internal_transfer); ;; 確保當中
的是 Jetton 內部轉帳操作
        master_msg_slice~skip_query_id(); ;; 這操作不用查詢 ID
        int jetton_amount = master_msg_slice~load_coins(); ;; 轉移的代幣數量
        master_msg_slice~load_msg_addr(); ;; 發送代幣的起始地址
        master_msg_slice~load_msg_addr(); ;; 接收代幣的目標地址，將作出回應
        int forward_ton_amount = master_msg_slice~load_coins(); ;; 涉及此代幣轉帳的 TON 量
        check_either_forward_payload(master_msg_slice);
        ;; 用以規限及管理 payload 轉發的格式與操作 @jetton-uilts
        check_amount_is_enough_to_transfer(ton_amount, forward_ton_amount, fwd_fee);
```

```
        ;; 確認收到得 TON 足以支付這次 Jetton 代幣的發行
        send_to_jetton_wallet(to_address, jetton_wallet_code, ton_amount, master_msg, TRUE); ;; 把附有代幣轉移細節的信息寄送到 Jetton 帳戶
        save_data(total_supply + jetton_amount, admin_address, next_admin_address, jetton_wallet_code, metadata_uri); ;; 更新合約數據狀態，更改代幣發行總量記錄
        return (); ;; 收貔，合約操作流程結束（出口2）
    }

    if (op == op::burn_notification) {
        ;; 接收內部信息通知，處理特定數量 tokens 被銷毀的情況
        int jetton_amount = in_msg_body~load_coins(); ;; 提取要銷毀的 Jetton tokens 數量
        slice from_address = in_msg_body~load_msg_addr(); ;; 發送銷毀通知的 Jetton 錢包地址
        throw_unless(error::not_valid_wallet,
            equal_slices_bits(calculate_user_jetton_wallet_address(from_address, my_address(), jetton_wallet_code), sender_address)
        ); ;; 驗證信息發送地址及銷毀 tokens 的地址是否對應，不匹配則操作無效及拋出錯誤
        save_data(total_supply - jetton_amount, admin_address, next_admin_address, jetton_wallet_code, metadata_uri); ;; 驗證地址後更新 total_supply 並存儲為新的合約狀態
        slice response_address = in_msg_body~load_msg_addr(); ;; 從信息主體提取回應地址
        in_msg_body.end_parse();
        ;; 完成信息主體的解析，對應 recv_internal 第一行的 begin_parse

        if (~ is_address_none(response_address)) {
            ;; 如果 response_address 不是 addr_none（代表是有效地址）則發送回應信息
            var msg = begin_cell()
            .store_msg_flags_and_address_none(NON_BOUNCEABLE) ;; 立 flag 標註不能回彈
            .store_slice(response_address) ;; 信息目標地址
            .store_coins(0) ;; 沒有牽涉資金轉移
            .store_prefix_only_body() ;; 佔用一個信息主體大小的空間，不附帶主體內容 @stdlib.fc
            .store_op(op::excesses) ;; 插入"額外處理"的操作碼，令接收方理解此信息操作目的
            .store_query_id(query_id); ;; 存儲用於標記這次操作的 ID
            send_raw_message(msg.end_cell(), SEND_MODE_IGNORE_ERRORS | SEND_MODE_CARRY_ALL_REMAINING_MESSAGE_VALUE); ;; 忽略任何錯誤並攜帶所有剩餘的金額發送以上構建信息
        }
        return (); ;; 收貔，合約操作流程結束（出口3）
    }

    if (op == op::provide_wallet_address) {
        ;; 回應內部信息的請求，以內部信息回傳 Jetton Wallet 地址
        slice owner_address = in_msg_body~load_msg_addr(); ;; 從信息主體提取主權地址
        int include_address? = in_msg_body~load_bool(); ;; 查閱指示是否包裹再轉發主權地址
        in_msg_body.end_parse(); ;; 完結解讀，對應 recv_internal 第一行的 begin_parse

        cell included_address = include_address?
        ? begin_cell().store_slice(owner_address).end_cell()
        : null(); ;; 依照 include_address 取得的指示，放進 owner_address 或空地址

        var msg = begin_cell()
        .store_msg_flags_and_address_none(NON_BOUNCEABLE) ;; 立 flag 標註不能回彈
        .store_slice(sender_address) ;; 回傳查詢結果給發起詢問的地址，所以插入 sender_address
        .store_coins(0) ;; 沒有牽涉資金轉移
        .store_prefix_only_body() ;; 佔用一個信息主體大小的空間，不附帶主體內容 @stdlib.fc
        .store_op(op::take_wallet_address) ;; 插入"提取錢包地址"操作碼，令接收方理解此信息操作目的
```

```
        .store_query_id(query_id); ;; 存儲用於標記這次操作的 ID

            if (is_same_workchain(owner_address)) { ;; 檢查主權地址與合約是否位於同一工作鏈上
                msg = msg.store_slice(calculate_user_jetton_wallet_address
                  (owner_address, my_address(), jetton_wallet_code));
                  ;; 是則計算 owner_address 對應的 Jetton 錢包地址，並加插在消息主體中
            } else {
                msg = msg.store_address_none(); ;; 否則加插空地址
            }

        cell msg_cell = msg.store_maybe_ref(included_address).end_cell();
              ;; 最後把 included_address 的取值加插 owner_address 或空值，再結束構建
        send_raw_message(msg_cell, SEND_MODE_CARRY_ALL_REMAINING_MESSAGE_VALUE |
SEND_MODE_BOUNCE_ON_ACTION_FAIL); ;; 攜帶所有剩餘價值發送以上信息，並回彈發送失敗的信息
        return (); ;; 收貔，合約操作流程結束（出口4）
    }

    if (op == op::change_admin) { ;; 發起更改管理員
        throw_unless(error::not_owner, equal_slices_bits(sender_address, admin_address));
        ;; 只有現任管理員的信息來源才會受理繼續操作
        next_admin_address = in_msg_body~load_msg_addr(); ;; 設定信息中提供的地址為準管理員
        in_msg_body.end_parse(); ;; 完結解讀，對應 recv_internal 第一行的 begin_parse
        save_data(total_supply, admin_address, next_admin_address, jetton_wallet_code, metadata_uri); ;;
把新的 next_admin_address 值存儲到新的合約狀態
        return (); ;; 收貔，合約操作流程結束（出口5）
    }

    if (op == op::claim_admin) { ;; 領取管理員權限
        in_msg_body.end_parse(); ;; 完結解讀，對應 recv_internal 第一行的 begin_parse
        throw_unless(error::not_owner, equal_slices_bits(sender_address, next_admin_address)); ;; 只有準
管理員的信息來源才會受理繼續操作
        save_data(total_supply, next_admin_address, address_none(), jetton_wallet_code, metadata_uri);
;; 把 next_admin_address 的值存儲到 admin_address 的位置，並把 next_admin_address 設為空白地址
        return (); ;; 收貔，合約操作流程結束（出口6）
    }

    if (op == op::drop_admin) { ;; 放棄管理員權限，讓合約真正去中心
        throw_unless(error::not_owner, equal_slices_bits(sender_address, admin_address));
        ;; 只有現任管理員的信息來源才會受理繼續操作
        in_msg_body.end_parse(); ;; 完結解讀，對應 recv_internal 第一行的 begin_parse
        save_data(total_supply, address_none(), address_none(), jetton_wallet_code, metadata_uri); ;; 把
next_admin_address 及 admin_address 都設為空白地址 （無主權）
        return (); ;; 收貔，合約操作流程結束（出口7）
    }

    if (op == op::change_metadata_uri) { ;; 更新 Jetton token 的描述或屬性
        throw_unless(error::not_owner, equal_slices_bits(sender_address, admin_address));
        ;; 只有現任管理員的信息來源才會受理繼續操作
        save_data(total_supply, admin_address, next_admin_address, jetton_wallet_code,
begin_cell().store_slice(in_msg_body).end_cell()); ;; 儲存新數據到合約
        return (); ;; 收貔，合約操作流程結束（出口7）
```

```
        }

        if (op == op::upgrade) { ;; 升級本合約代碼和數據
            throw_unless(error::not_owner, equal_slices_bits(sender_address, admin_address));
            ;; 只有現任管理員的信息來源才會受理繼續操作
            (cell new_data, cell new_code) = (in_msg_body~load_ref(), in_msg_body~load_ref());
            ;; 升級將涉及數據結構和邏輯變更，因此信息將提供兩組 ref cell，分別為新的數據和代碼
            in_msg_body.end_parse(); ;; 完結解讀，對應 recv_internal 第一行的 begin_parse
            set_data(new_data);
            set_code(new_code);
            ;; 將合約的數據和代碼更新為新的數據和代碼，新邏輯和狀態將在本次操作成功後生效
            return (); ;; 收貔，合約操作流程結束（出口8）
        }

        if (op == op::top_up) { ;; 單純向合約充值 TON （沒改變合約狀態）
            return (); ;; 收貔，合約純粹接收 TON，結束操作流程（出口9）
        }

        throw(error::wrong_op);
;; 當接收的操作碼無效，合約會拋出明確錯誤阻止任何未經授權的操作，確保合約不會執行任何未定義或潛在的危險操作。
}



;; 用在於 get_jetton_data 函數當中，用數據建構字典 cell 以供讀取
cell build_content_cell(slice metadata_uri) inline {
    cell content_dict = new_dict();
    content_dict~set_token_snake_metadata_entry("uri"H, metadata_uri);
    content_dict~set_token_snake_metadata_entry("decimals"H, "9");
    return create_token_onchain_metadata(content_dict);
}



;; 取得 Jetton 代幣數據
(int, int, slice, cell, cell) get_jetton_data() method_id { ;; 允許外部調用
    (int total_supply, slice admin_address, slice next_admin_address, cell jetton_wallet_code, cell
metadata_uri) = load_data();
    return (total_supply, TRUE, admin_address, build_content_cell(metadata_uri.begin_parse()),
jetton_wallet_code); ;; 回傳總供應量，操作成功標記，主合約管理員地址，代幣描述，錢包核心代碼
}



;; 查閱某地址所屬的 Jetton token 地址
slice get_wallet_address(slice owner_address) method_id { ;; 允許外部調用
    (int total_supply, slice admin_address, slice next_admin_address, cell jetton_wallet_code, cell
metadata_uri) = load_data();
    return calculate_user_jetton_wallet_address(owner_address, my_address(), jetton_wallet_code);
}
```

```
;; 查閱本合約現時的準管理員地址
slice get_next_admin_address() method_id { ;; 允許外部調用
   (int total_supply, slice admin_address, slice next_admin_address, cell jetton_wallet_code, cell
metadata_uri) = load_data();
   return next_admin_address;
}
```