

# Дополнительны е ВОЗМОЖНОСТИ

- Структура апплетов RemoteApp и создание своего апплета

# Структура апплетов RemoteApp и создание своего апплета

## Что такой апплет?

Апплет представляет из себя набор файлов, которые описывают процесс установки и запуска того или иного приложения на Microsoft RDS через RemoteApp. Это нужно для того, чтобы JumpServer мог запускать сессию доступа к этому приложению, автоматически авторизуясь в нем и скрывая параметры авторизации от пользователя.

## Структура апплета

Каждый апплет обязательно состоит из этих файлов:

```
AppletName
├── i18n.yml
├── icon.png
├── main.py
├── manifest.yml
└── setup.yml
```

main.py - скрипт запуска и авторизации в приложении

icon.png - значок апплета

manifest.yml - метаданные, то есть описание апплета

setup.yml - файл с описанием установки

i18n.yml - файл перевода на различные языки

## Файл manifest.yml

Пример на базе апплета MySQL Workbench

В файле manifest.yml находится общая информация об апплете и указание его типа и протокола.

```
# (обязательно)
name: mysql_workbench8
display_name: "{{ 'MySQL Workbench' | trans }}"
comment: "{{ 'A tool for working with MySQL, to execute SQL and design tables' | trans }}"
```

```

# (обязательно))
version: 0.1.1
# (обязательно)
exec_type: python
# (обязательно)
author: Eric
# general или web (обязательно)
type: general
update_policy: always
edition: community
# (обязательно)
tags:
  - database
# (обязательно)
protocols:
  - mysqlworkbench

# перевод на дрругие языки
i18n:
  MySQL Workbench:
    en: MySQL Workbench
    zh: MySQL Workbench
    ja: MySQL Workbench

A tool for working with MySQL, to execute SQL and design tables:
en: A tool for working with MySQL, to execute SQL and design tables
zh: MySQL MySQL SQL
ja: MySQLSQL

```

## Файл setup.yml

Файл setup.yml описывает параметры установки апплета на RDS сервер.

```

# тип установки ПО - msi,exe, zip, manual
type: msi
# адрес для загрузки дистрибутива ПО или имя файла, если дистрибутив вместе с апплетом в архиве
source: mysql-workbench-community-8.0.31-winx64.msi
# аргументы запуска установки
arguments:
  - /qn
  - /norestart
# директораия для установки
destination: C:\Program Files\MySQL\MySQL Workbench 8.0 CE
# путь и имя исполняемого файла
program: C:\Program Files\MySQL\MySQL Workbench 8.0 CE\MySQLWorkbench.exe
md5: d628190252133c06dad399657666974a

```

# Скрипт main.py

**main.py** - основной скрипт апплета

**Запуск приложения осуществляется вызовом команды:**

```
python main.py base64_json_data
```

То есть запускается скрипт **main.py** и в него передаются параметры запуска, структура **base64\_json\_data** выглядит примерно так:

```
{
  "app_name": "mysql_workbench8",
  "protocol": "mysql",
  "user": {
    "id": "2647CA35-5CAD-4DDF-8A88-6BD88F39BB30",
    "name": "Administrator",
    "username": "admin"
  },
  "asset": {
    "asset_id": "46EE5F50-F1C1-468C-97EE-560E3436754C",
    "asset_name": "test_mysql",
    "address": "192.168.1.1",
    "protocols": [
      {
        "id": 2,
        "name": "mysql",
        "port": 3306
      }
    ]
  },
  "account": {
    "account_id": "9D5585DE-5132-458C-AABE-89A83C112A83",
    "username": "root",
    "secret": "test"
  },
  "platform": {
    "charset": "UTF-8"
  }
}
```

## Содержимое main.py

```
import sys

from common import (block_input, unblock_input) # Импорт функций для блокировки/
разблокировки ввода
from common import convert_base64_to_dict # Импорт функции для конвертации Base64 строки
в словарь(массив)
from app import AppletApplication # Импорт основного приложения
```

```

def main():
    base64_str = sys.argv[1] # Получаем строку Base64 из аргументов командной строки
    data = convert_base64_to_dict(base64_str) # Конвертируем Base64 строку в словарь
    # словарь data содержит все параметры запуска приложения: учетную запись, имя сервера,
    имя базы данных и тд в зависимости от типа приложения
    applet_app = AppletApplication(**data) # Передаем данные словаря в функцию запуска
    приложения
    block_input() # Блокируем ввод пользователя
    applet_app.run() # Запускаем приложение
    unblock_input() # Разблокируем ввод пользователя
    applet_app.wait() # Ожидаем завершения работы приложения

if __name__ == '__main__':
    try:
        main() # Запускаем основную функцию
    except Exception as e:
        print(e) # Выводим ошибку, если она возникла

```

## Содержимое app.py

App.py обычно содержит весь основной код запуска приложения с нужными параметрами, поэтому это самая важная и самая сложная часть при разработке нового апплета. Проще всего не создавать его с нуля, а взять за основу один из скриптов апплетов, которые похожи по структуре\типу приложения на новый создаваемый апплет.

```

import sys # Импортирует модуль sys для работы с системными функциями

if sys.platform == 'win32': # Проверяет, если операционная система — Windows
    from pywinauto import Application # Импортирует библиотеку для автоматизации оконных
    приложений на Windows
    from pywinauto.controls.uia_controls import (ButtonWrapper, EditWrapper, MenuItemWrapper,
                                                MenuWrapper, ComboBoxWrapper, ToolbarWrapper)
    # Импортирует различные элементы управления для взаимодействия с интерфейсом
    приложения

from common import (BaseApplication, wait_pid, ) # Импортирует базовое приложение и
функцию ожидания процесса

_default_path = r"C:\Program Files\MySQL\MySQL Workbench 8.0 CE\MySQLWorkbench.exe"
# Определяет путь к приложению MySQL Workbench по умолчанию

class AppletApplication(BaseApplication): # Определяет класс приложения, наследующий от
BaseApplication

    def __init__(self, *args, **kwargs): # Инициализирует приложение
        super().__init__(*args, **kwargs) # Вызов конструктора родительского класса

```

```

self.path = _default_path # Устанавливает путь к приложению
self.username = self.account.username # Получает имя пользователя из учетной записи
self.password = self.account.secret # Получает пароль из учетной записи
self.host = self.asset.address # Получает адрес хоста из параметров актива
self.port = self.asset.get_protocol_port(self.protocol) # Получает порт по протоколу
self.db = self.asset.spec_info.db_name # Получает название базы данных
self.pid = None # Переменная для хранения ID процесса приложения
self.app = None # Переменная для хранения объекта приложения

def run(self): # Метод для запуска приложения
    app = Application(backend='uia') # Создает объект приложения для взаимодействия через
    UI Automation
    app.start(self.path) # Запускает приложение по указанному пути
    self.pid = app.process # Сохраняет ID процесса приложения
    if not all([self.username, self.password, self.host]): # Проверяет, заполнены ли необходимые
    параметры
        print(f'参数不全') # Выводит сообщение об ошибке на китайском ("Не хватает
    обязательных параметров")
        return

    # Доступ к главному окну MySQL Workbench и меню "Database"
    menubar = app.window(title="MySQL Workbench", auto_id="MainForm",
    control_type="Window") \
        .child_window(title="Database", control_type="MenuItem")
    menubar.wait('ready', timeout=10, retry_interval=5) # Ожидает готовности меню
    MenuItemWrapper(menubar.element_info).select() # Открывает меню "Database"

    # Выбирает пункт меню "Connect to Database"
    cdb = menubar.child_window(title="Connect to Database", control_type="MenuItem")
    cdb.wait('ready', timeout=10, retry_interval=5) # Ожидает готовности элемента
    MenuItemWrapper(cdb.element_info).click_input() # Нажимает на элемент "Connect to
    Database"

    # Вводит хост
    host_ele = app.top_window().child_window(title="Host Name", auto_id="Host Name",
    control_type="Edit")
    EditWrapper(host_ele.element_info).set_edit_text(self.host) # Вводит значение хоста

    # Вводит порт
    port_ele = app.top_window().child_window(title="Port", auto_id="Port", control_type="Edit")
    EditWrapper(port_ele.element_info).set_edit_text(self.port) # Вводит значение порта

    # Вводит имя пользователя
    user_ele = app.top_window().child_window(title="User Name", auto_id="User Name",
    control_type="Edit")
    EditWrapper(user_ele.element_info).set_edit_text(self.username) # Вводит имя пользователя

    # Вводит имя базы данных
    db_ele = app.top_window().child_window(title="Default Schema", auto_id="Default Schema",

```

```

control_type="Edit")
    EditWrapper(db_ele.element_info).set_edit_text(self.db) # Вводит имя базы данных

    # Нажимает на кнопку "OK" для подтверждения соединения
    ok_ele = app.top_window().child_window(title="Connection", auto_id="Connection",
control_type="Window") \
        .child_window(title="OK", control_type="Button")
    ButtonWrapper(ok_ele.element_info).click() # Нажимает на кнопку "OK"

    # Вводит пароль
    password_ele = app.top_window().child_window(title="Password", auto_id="Password",
control_type="Edit")
    password_ele.wait('ready', timeout=10, retry_interval=5) # Ожидает готовности поля для
ввода пароля
    EditWrapper(password_ele.element_info).set_edit_text(self.password) # Вводит пароль

    # Нажимает на кнопку "OK" для завершения подключения
    ok_ele = app.top_window().child_window(title="Button Bar", auto_id="Button Bar",
control_type="Pane") \
        .child_window(title="OK", control_type="Button")
    ButtonWrapper(ok_ele.element_info).click() # Нажимает на кнопку "OK"

    self.app = app # Сохраняет объект приложения для дальнейшего использования

def wait(self): # Метод для ожидания завершения работы приложения
    wait_pid(self.pid) # Ожидает завершения процесса с определенным ID (pid)

```