中国农业信息 China Agricultural Informatics
第 36 卷第 1 期 2024 年 2 月 2024，36（1）：43-55 Vol.36，No.1 Feb.，2024

智慧农业

异态异构农业模型服务化封装方法研究*

陆宏伟[1]，潘瑜春[2※]
（1. 江苏大学计算机科学与通信工程学院，镇江 212000；2. 北京市农林科学院信息技术研究中心，北京 100097）

摘要：【目的】农业模型涉及多学科与多领域，是智慧农业的核心要素。屏蔽农业模型开发语言、调用运行方式等异态、异构特性，实现农业模型资源高效集成调用与共享，对于农业系统要素科学农业管理和决策，促进农业可持续、高效与安全发展具有重要意义。**【方法】**文章设计了模型描述接口、模型执行接口和模型部署接口等标准化封装接口，并在此基础上提出了针对COM组件、Jar包、R语言、Python语言以及DLL等不同形态模型运行体的标准化服务封装方法；以Python形态与DLL形态实现的马铃薯智能化推荐施肥模型为例，验证异态异构模型服务化封装方法的可行性与实用性。**【结果】**（1）封装后形成了模型标准化描述文档，完成了模型运行入口函数转换，增加了第三方依赖环境库，最终可通过网络请求直接运行模型，降低了模型使用成本；（2）以云南省为例，对封装后的标准模型服务进行调用运行，输入马铃薯目标产量，可获得该目标产量下最佳施肥量推荐，验证了异态异构模型服务化封装方法的可行性与实用性。**【结论】**该方法能够屏蔽模型在语义、数据、形态、运行环境等方面的差异，有效促进农业模型资源在网络环境下复用共享，增强智慧农业领域模型资源管理能力。

关键词：农业模型；异态异构模型；模型资源共享；标准化服务封装
DOI：10.12105/j.issn.1672-0423.20240104

0 引言

农业模型涉及生物、环境、技术及经济等农业领域，农业大数据及模型管理与应用是智慧农业的核心要素[1]。目前各学科、各领域开发了大量农业模型，要求智慧农业系统支持对各类已开发的模型进行有效管理和调用运行，以达到模型资源的重用与共享[2]。但由于模型开发者背景不同，其模型开发语言、实现方式及存在形态迥异，导致已有异态、异构模型资源难以灵活调用运行，无法复用与共享，从而形成模型孤岛效应[3, 4]，并已成为制约智慧农业发展的关键因素之一。如何标准化异构模型的语义、数据与调用方式，并将不同形态模型组件统一封装为Web服务，屏蔽模型的异态异构性，使得模型

收稿日期：2024-01-26
第一作者简介：陆宏伟（1997—），硕士研究生。研究方向：计算机应用技术。Email:18896670068@163.com
※ **通信作者简介：**潘瑜春（1971—），博士、研究员。研究方向：农业时空数据分析及服务技术。Email: panyc@nercita.org.cn
* **基金项目：**国家重点研发计划“黑土地耕地质量时空多维大数据预警系统研发”（2021YFD1500104）

资源能在网络下方便地复用与共享，是构建智慧农业系统亟须解决的关键问题。

目前，模型管理系统能管理的模型形态单一，如分别只针对exe应用程序、Jar组件或C\C++编译的组件，并利用python语言进行调用研究[5]；或者对Fortran语言编译的模型利用Java进行交互，并发布为服务[6]。更重要的是，这些模型与系统耦合度较高，无法动态扩展其他形态的模型组件，导致模型资源无法重用与共享，系统适用性、适应性和可扩展性受限[7-9]。针对这一问题，一些系统通过设定模型契约、模型运行接口等标准对模型进行规范标准化，对模型进行重新封装[10, 11]。在模型集成时，模型提供者需按系统接口标准进行模型的重新开发与改造[12]，使之符合其设定的接口标准，此类接口标准如基础模型接口（Basic model interface，BMI）、开放模型标准接口（Opening Modeling Interfacce，OpenMI）等，可促进模型的集成与复用[13]。但是，由于专业模型开发的困难性与模型建模的领域差异化，重新按照模型接口标准进行模型开发或改造模型实现困难，并且不同的框架标准相互独立，标准无法统一，它们之间的模型无法进行交互[14]，进一步加剧了模型的低水平重复开发，已有大量模型资源浪费严重。

鉴于此，文章对模型标准化封装接口与异态模型标准化服务封装方法进行研究，以期通过屏蔽模型形态结构差异特性，使得模型与系统解耦，方便模型资源集成，促进模型的复用与共享，也为增强智慧农业系统适用性和动态扩展能力提供支撑。

1 异态异构农业模型服务化封装策略设计

标准化是模型集成管理的基础。为标准化模型的语义、输入输出数据、运行环境、调用方式等诸多异构特性，该研究提出一种模型服务封装策略，包括模型标准化封装接口和异态模型标准化服务封装，整体架构如图1所示。

1.1 模型标准化封装接口

模型标准化封装接口包括模型描述接口、模型执行接口和模型部署接口3种类型。

（1）模型描述接口。用于对模型语义信息进行标准化描述，主要通过分析农业模型的共同属性，抽象出模型元数据描述标准，使得模型提供者能够以灵活和结构化的方式来描述模型相关的语义信息。模型描述接口主要包括模型名称、用途、适用条件、运行机理等描述信息，还包括模型使用文档，方便模型规范使用。通过该接口规范模型描述信息的定义、分类及原理表达，让使用者全面了解模型的属性与开发背景等信息，便于使用者查询与使用模型。

（2）模型执行接口。用于对模型输入、输出数据与运行行为进行标准化描述，对模型的输入、输出数据格式标准化转换，使其便于系统的前后端交互。不同形态模型的输入输出数据具有结构与语义上的异构性，其运行方式也同样相互异构，将模型原生运行接口与该执行接口进行映射，从而形成标准化模型服务。通过该接口规范多形态模型的输入输出语义、数据结构与执行行为信息的结构化描述，可为模型调用机制的研究提供信息支持。

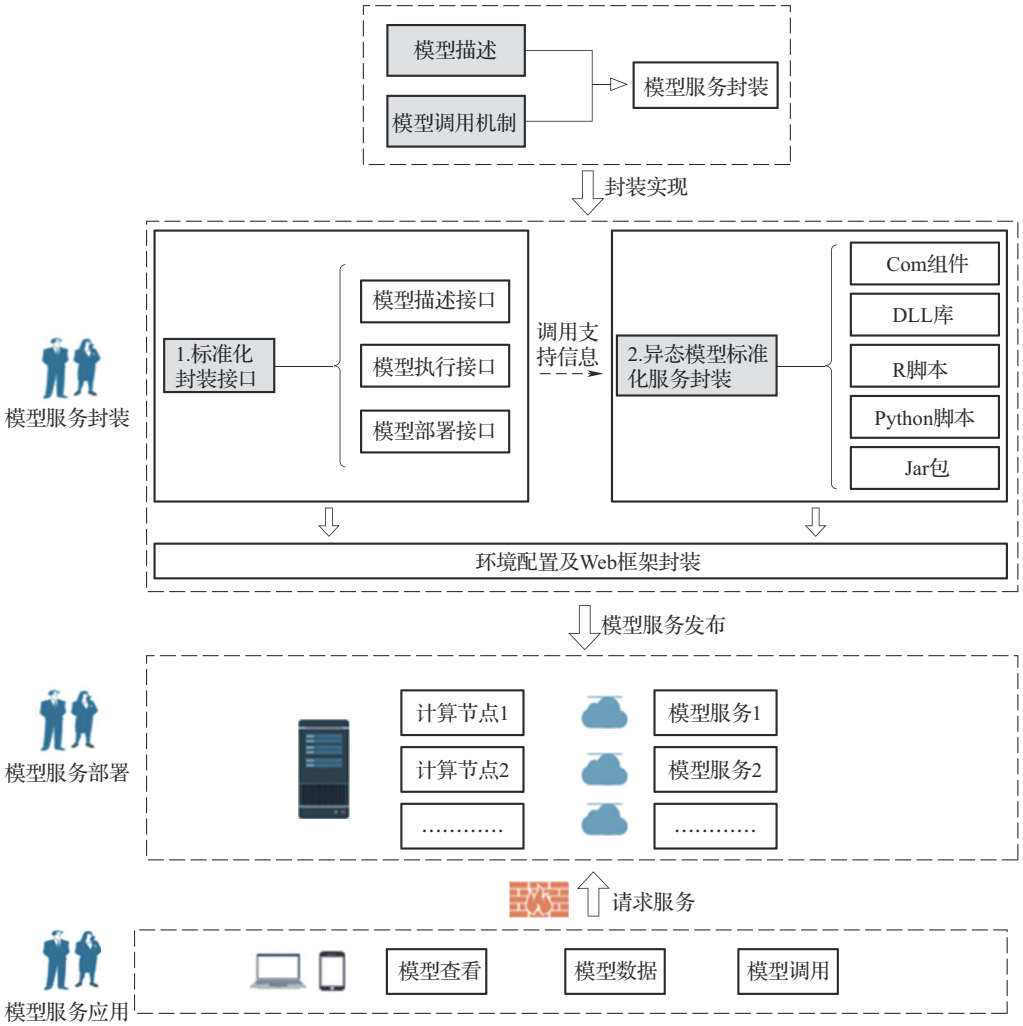

图 1　异态异构模型服务化封装方法架构

Fig. 1　Service-oriented encapsulation of diverse forms and structures in models

（3）模型部署接口。用于对异态异构模型服务的部署信息进行标准化描述，包括模型服务运行所需的依赖环境、第三方包、软硬件环境以及运行平台等部署信息，并可进一步对模型资源使用权限进行安全检查与配置。通过该接口规范描述的模型运行依赖标准信息来配置模型运行环境，最终实现将封装后的标准化模型服务部署与发布，为模型使用者暴露模型服务的接口，使用者可根据接口信息准备输入数据请求调用模型。

1.2　异态模型标准化服务封装

异态模型标准化服务封装主要利用Web框架将异态模型组件封装为标准化的模型服务，同时完成模型调用方式统一服务化转换，屏蔽模型的异构特性，以达到对异构模型有效管理和高效调用。其中，模型形态是指模型执行体的存在形态，如COM组件、Jar包和DLL动态链接库，以及R语言或Python语言等代码形态。通过异态模型标准化服务封装，可在模型标准化封装接口生成的模型标准化描述文档基础上，获取不同形态模型的

调用运行机制的关键支持信息，并对其进行相应的加载与配置，实现模型调用运行机制的统一化。

2 异态异构农业模型服务化封装策略实现

2.1 模型标准化封装接口设计

2.1.1 模型描述接口

将模型语义信息抽象为通用的模型元数据描述标准，形成模型描述接口（图2）。DescriptionTreeNode 为描述类的数据结构，由节点的属性和名称构成，CategoryObj、HelperObj 和 AttributeObj 3 个描述接口的父类遵循此结构，共同组成描述接口集合 DescriptionSet，实现对异构模型的标准化描述，并分别由 CategoryInfo、AttributeInfo 与 HelpInfo 3 个派生描述类承担异构模型语义信息规范化的具体工作。CategoryInfo 为模型的分类信息类，定义了模型关键字、模型应用领域和模型类型等属性，主要用于模型的查询与分类功能。AttributeInfo 为模型的基本信息类，定义了模型介绍、模型建模原理、模型版本、模型创建人、创建时间和使用场景等属性，主要用于模型的基本信息描述。HelpInfo 为模型的使用信息类，定义了模型的使用文档和示例等属性，对模型如何使用做出详细的说明指导。

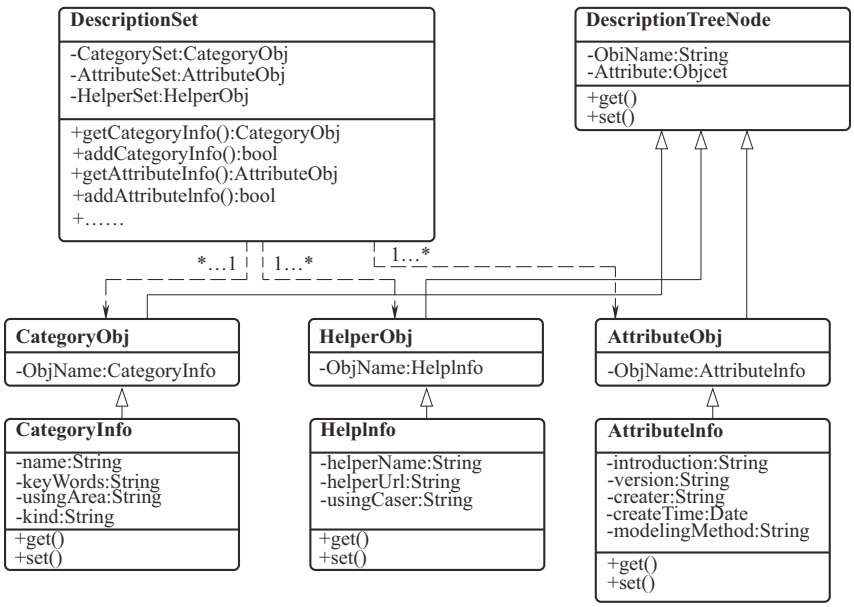

图2 模型描述接口类图

Fig. 2 Class diagram for model description interface

2.1.2 模型执行接口

模型执行接口依赖于 4 个描述类，分别为 InputInfo、OutputInfo、ParameterINfo 和 RunInfo（图3）。InputInfo 表示输入信息类，定义了模型输入的名称、类型、说明和范围等规范属性，用于标准化描述模型的输入信息。OutputInfo 表示模型输出信息类，定义了

模型输出的名称、说明和类型等属性，用于标准化描述模型的输出信息。ParameterINfo表示模型参数信息类，参数值根据模型不同的使用场景具有可调节性，定义了参数名称、参数类型、参数值和参数说明等属性。RunInfo表示模型的运行信息类，定义了不同形态模型运行的入口函数信息，如函数名、函数参数集等属性。通过模型执行接口，规范了模型的输入输出语义、结构信息与模型的运行信息，从而屏蔽模型运行过程中的数据与调用运行信息的异构特性。

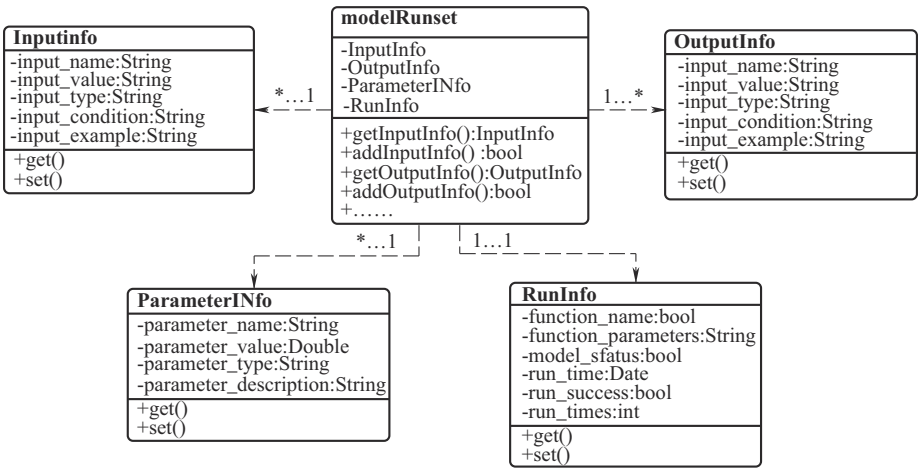

图3　模型执行接口类图

Fig. 3　Class diagram of model execution interface

2.1.3　模型部署接口

模型部署接口依赖于ProgramInfo、ModelFileInfo、EnvironmentInfo以及PermissionInfo4个模型配置类（图4）。ProgramInfo为模型的开发信息类，定义了模型开发语言、模型编译工具和目标程序类型等属性，负责描述模型的开发实现信息。ModelFileInfo为模型的文件信息类，定义了模型的依赖包、模型运行文件和第三方支持库等属性，负责描述模型运行的依赖信息。EnvironmentInfo为模型的运行环境类，定义了模型运行的软硬件需求、操作系统需求等属性，负责描述模型运行需要配置的环境。PermissionInfo为模型安全信息类，定义了模型禁用状态和模型使用权限标识等属性，负责描述模型资源的安全配置策略。通过模型部署接口，模型的运行条件规范化得到解决，可据此接口配置模型运行环境，将配置后的模型组件发布为模型服务。

2.2　异态模型标准化服务封装实现

模型标准化服务封装流程如图5所示。在模型标准化封装接口形成的模型标准描述文档基础上，通过类中公共方法get方法从模型部署接口中获取模型组件运行所需要的环境依赖属性信息，包括模型的运行文件、模型依赖库等，用于异态模型运行环境的封装。同样再获取模型执行接口中私有属性信息，包括模型的输入变量名、输入变量值、输入变量的约束条件、模型入口函数名和函数参数等，用于支持模型调用运行。最后配置好模型运行环境，通过Web框架进行模型服务化封装，从而形成标准化的模型服务。

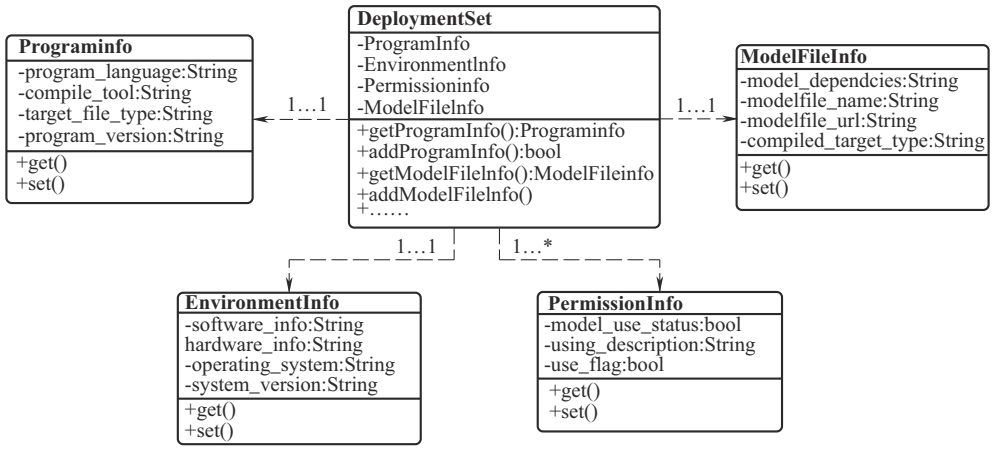

图 4　模型部署接口类图

Fig. 4　Model deployment interface class diagram

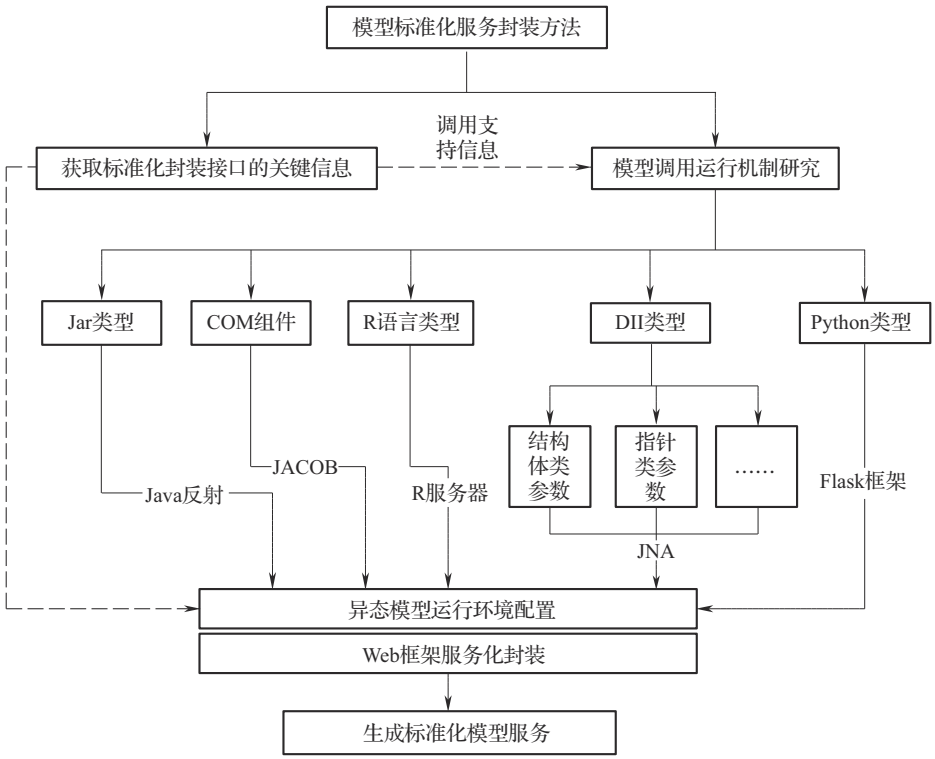

图 5　异态模型标准化服务封装实现流程

Fig. 5　Flowchart of standardized service encapsulation for heterogeneous model implementation

不同形态模型组件主要通过以下 4 个主要步骤实现封装。

（1）链接模型运行文件，并获取关键信息。加载模型运行文件，使得模型组件与开发环境建立连接，获取运行文件中的关键对象信息，如模型的入口函数信息，函数的参数信息等。

（2）获取模型组件的原输入参数并处理。对输入参数进行标准化操作，如获取参数，

对参数的数据结构进行转换等，以将多形态模型的原输入参数的数据结构与系统标准化后的相映射。

（3）模型调用，对模型输出结果进行处理。将映射后的标准输入传入模型运行文件的输入接口运行模型，通过上述不同的调用机制运行模型，如反射、服务等，并对模型运行后的返回结果进行标准化的格式转换。

（4）Web 框架封装，完成异态模型标准化服务封装。最后基于 SpringMVC、Flask 等 Web 框架，将多形态的模型组件标准化服务封装，同时生成模型调用服务，统一模型调用方式。

因为不同形态模型调用运行机制有所不同，需要针对性封装技术实现：针对 Jar 包的模型组件，基于 Java 反射技术对其自动调用运行封装；针对 COM 组件的模型，基于 Java-COM Bridge（JACOB）技术对其自动调用运行，使得 Java 可以调用 COM 组件封装好的功能；针对 R 语言形态的模型，主要通过 RServe 方法开启 R 服务，使得 Java 程序可远程调用 R 模型；针对 Python 语言类的模型，基于自身的 Web 框架 Flask，结合相关 Python 模型运行环境的配置，将 Python 模型直接封装为标准的模型服务；针对 DLL 形态的模型，基于 JNA 技术来实现调用封装。JNA 是一个允许 Java 程序与本地库进行交互的框架，使得 Java 可调用第三方语言如 C\C++等编译的本地库中函数接口。由于 DLL 形态模型通常包含除基本数据外，还包含特殊的数据类型如指针、结构体、引用、共用体等，针对这些特殊的数据类型，通过对 JNA 交互接口的运用与封装，系统自动完成数据类型映射，表 1 为 Java 与 DLL 形态模型中部分数据类型的对应关系映射表。

表 1　数据结构映射

Table 1　Data structure mapping

C\\C++ type	Java type	说明
int	int	整形
char	char\byte	字符
int *	Pointer	整形指针
struct	Structure	结构体
Int &	Poniter	整形引用
union	Union	共用体
enum	Enum类	枚举

3　异态异构农业模型服务化封装与验证

选取由 Python 与 DLL 两类形态实现的马铃薯智能化推荐施肥模型[15]，分别通过模型标准化服务封装，并展示模型组件封装前后关键映射项的对比，以及对封装后的模型服务进行调用运行，以验证封装方法的可行性。

3.1　异态异构农业模型服务化封装

首先分别解析两种形态模型的基础描述信息，输入输出与模型运行的执行描述信息

与运行环境相关的部署信息，再通过中间件 Mybatis 等获取解析的信息并生成规范化的相关模型描述文档，实现模型标准描述接口。由模型标准描述文档可知，马铃薯智能化推荐施肥模型是基于 QUEFTS 模型进行大量田间试验建模而成，通过模拟不同种植区马铃薯产量与养分吸收特征关系，实现推荐施肥和养分管理，并依据公式施氮量=产量反应/农学效率，施磷或施钾量=作物产量反应施磷或施钾量+维持土壤养分平衡，分别获得目标产量下氮肥、磷肥和钾肥推荐施用量。在输入马铃薯目标产量后，运行模型可自动输出氮、磷和钾肥的推荐施肥量。结果显示，我国马铃薯主产区氮、磷和钾肥的平均施用量分别为 8.1 t/hm^2、4.8 t/hm^{2}和 5.2 t/hm^2，施用氮、磷和钾肥的平均农学效率分别 45.2 kg/kg、51.1 kg/kg 和 35.1 kg/kg。接着从生成的标准模型描述文档中，获取支持模型调用运行机制研究的关键属性进行异态模型的调用运行机制研究，如模型运行的入口函数名，函数参数信息等，并通过配置模型运行环境与 Web 框架将模型组件封装为标准化模型服务。

3.1.1 Python 形态模型服务化封装

Python 形态模型服务化封装涉及的主要代码逻辑如图 6 所示。

（1）链接模型文件。在开发环境中链接模型，获取模型的模块对象与入口函数对象。将从模型标准化封装接口获取 Python 模型的入口函数名与模型存储地址，传入 inspect 库中的 getmodulename（）方法与 getattr（）方法获取模型的模块对象和入口函数对象。并通过给 Python 模型分配不同的端口号以支持模型的并行运行；

（2）获取模型组件的原输入参数并处理。在 Python 模型编译环境中自定义一个包装函数，在函数中利用第三方支持库 inspect 模块中 signature（）函数的 parameters（）方法，获取 Python 模型的入口函数参数信息对象；接着，利用第三方支持库 request 模块中的 data.get（）方法，传入入口函数参数信息对象参数，把参数转换为列表进行标准化处理。同时在函数中定义模型服务的请求地址，通过@app.route 函数定义 Python 模型的路由信息（如 http：//127.0.0.1：5000/PyWeb）。

（3）调用输出结果处理。通过 Json 库中的 jsonify（）方法将调用运行 Python 模型后得到模型计算的结果封装为 json 格式返回给请求者。

（4）标准化服务封装。对上述模型处理逻辑通过 Flask 框架封装与部署，完成模型标准化服务封装。

3.1.2 DLL 形态模型服务化封装

DLL 形态模型服务化封装涉及的主要代码逻辑如图 7 所示。

（1）链接模型文件。在开发环境中链接模型，获取模型的函数接口对象。将从模型标准化封装接口获取 DLL 形态模型的类名与模型存储地址，传入 Native 库中的 load（）方法加载 DLL 模型并获取接口对象。

（2）输入参数处理。自定义继承 Structure 类的 CropParams 类，用于构造 DLL 模型运行所需结构体参数的变量，并在类中重写 getFieldOrder（）方法，确保可按顺序正确地映射参数值，再通过 Map 集合调用 get 方法获取入口函数的参数值，将这些参数值分别赋值到创建的结构体 CropParams 中。

```
from flask import Flask, request, jsonify      -->导入依赖环境
import inspect
import importlib
import json
app = Flask(__name__)
def wrap_function(func):
    parameters = inspect.signature(func).parameters
    @app.route("/PyWeb", methods=["POST"])    -->服务地址发布
    def handle_request():
        data = request.get_json()
        args = []
        for param in parameters.values():    -->获取输入参数并处理
            arg = data.get(param.name)
            args.append(arg)
        result = func(*args)    -->调用模型，处理返回结果
        response = {
            "result": result
        }
        return jsonify(response)
    return handle_request
def load_and_wrap_function(file_path, function_name):    -->链接 Python 模型运行文件
    module_name = inspect.getmodulename(file_path)
    module_spec = importlib.util.spec_from_file_location(module_name, file_path)
    module = importlib.util.module_from_spec(module_spec)
    module_spec.loader.exec_module(module)
    if hasattr(module, function_name):
        function = getattr(module, function_name)
        return wrap_function(function)
    else:
        raise ValueError(f"Function '{function_name}' not found in the module.")
```

图 6　Python 形态模型服务化封装主要代码

Fig. 6　Python form model service encapsulation main code

```
    public interface cextends Library {
CropYield INSTANCE = Native.load("D:\\Debug\\CropYield.dll", CropYield.class);    ---->链接模型
    class CropParams extends Structure {            ---->参数构造与处理
    public double irrigation_volume
    public double black_soil_thickness
    public double temperature
    public double organic_matter_content
    public double crop_density
    public double pH
    @Override
    protected List<String> getFieldOrder() {      ---->确定参数赋值顺序
  return Arrays.asList("irrigation_volume" ,"black_soil_thickness temperature", "temperature"
,"organic_matter_content ", "crop_density ", "pH    ");} }
  double CropYield (CropParams params);
    }
     public double CropYield (Map<String, Double> paramMap) {                    ---->模型调用
     CropYield.CropParams params = new CropYield.CropParams();
     params.irrigation_volume    = paramMap.get("irrigation_volume");          ---->参数赋值
     params.black_soil_thickness    = paramMap.get("black_soil_thickness ");
     params.temperature = paramMap.get("temperature");
     params. organic_matter_content    = paramMap.get(" organic_matter_content ");
     params.crop_density = paramMap.get("crop_density");
     params.pH    = paramMap.get("pH");
     double result = CropYield.INSTANCE.CropYield (params);
     return result;                                ---->模型运行并返回结果
    }
```

图 7　DLL 形态模型服务化封装主要代码

Fig. 7　DLL form model service encapsulation main code

（3）调用输出结果处理。通过函数接口对象调用本地 DLL 模型中函数 CropYield（），函数参数为赋值过的结构体对象，得到模型运行结果，将结果转换为标准输出格式。

（4）标准化服务封装。对上述模型的处理逻辑通过SpringMVC框架封装与部署，完成模型标准化服务封装。

3.2 模型服务化封装效果验证

3.2.1 模型封装映射项对比

对Python形态模型和DLL形态模型封装前后主要映射关系进行对比分析（表2）。

（1）模型描述标准文档。封装前，两类形态的模型没有形成标准的模型描述文档，经过封装接口封装后形成了模型标准化描述文档。

（2）模型运行入口函数。两类模型运行的入口函数在封装后分别转换wrapFunction（path，funcName）与PotatoNutrient（Map<String，Double> variableMap，Integer mid），其中Python模型入口函数的参数由模型运行体的地址与函数名构成，DLL形态模型运行入口函数转换为由输入变量的名称与值的集合和模型唯一的标识符mid组成。

（3）模型运行依赖环境库。封装后的模型运行依赖库在原有模型的依赖库基础上新增了额外的第三方库，用于支持模型调用机制，如针对Python形态模型，新增Flask、Inspect、Json等库，针对DLL形态新增Jna、Native、Structure等类库。

（4）模型调用运行方式。封装前，模型调用运行的方式只能在各自形态的运行环境中去调用，而封装后可通过网络请求直接运行模型，如Python形态模型服务接口以POST方式进行请求，模型调用服务请求地址为http：//127.0.0.1：5000/PyWeb，DLL形态的模型调用服务以POST方式进行请求，请求地址为http：//localhost：8081/PotatoNutrient。

（5）模型使用成本。封装前，使用者需要学习不同形态模型的语言特性以及模型的调用方式才能使用模型，封装后使用者无需具备这些专业知识，也不需要手动配置模型运行环境。

表2 模型封装前后主要映射项对比

Table 2 Comparison of key mapping items before and after model encapsulation

映射项	Python形态模型	DLL形态模型	封装后模型
模型描述标准文档	未形成	未形成	形成模型标准化接口描述文档
模型运行入口函数	PotatoNutrient（yield，N，P，K）	PotatoNutrient（const CropYieldParameters& params）	Python：wrapFunction（path，funcName）；DLL：PotatoNutrient（Map<String，Double>，variableMap，Integer mid）
模型运行依赖环境库	numpy、math、scikit-learn库等	math、algorithm、iostream库等	Python：新增Flask、Inspect、Json等库 DLL：新增Jna、Native、Structure等库
模型调用运行方式	Pyhton环境调用	DLL环境调用	网络请求
模型使用成本	在特定环境下运行，需学习Pyhton语言	在特定环境下运行，需学习DLL环境的调用方式	无需学习第三方语言的调用方式，无操作系统、终端平台限制

3.2.2 模型服务调用验证

以云南省为例，利用Postman工具分别对封装后的Python形态模型和DLL形态模型进行服务调用验证。输入马铃薯目标产量25.1 t/hm^2，然后分别请求两类模型的调用运行服

务，以获取推荐施肥量数据。调用验证结果显示，云南省氮磷钾的最佳施肥量：氮肥为 145 kg/hm^2，磷肥为 82 kg/hm^2，钾肥为 130 kg/hm^2，Python 形态模型和 DLL 形态模型获得的结果一致（图 8、9）。

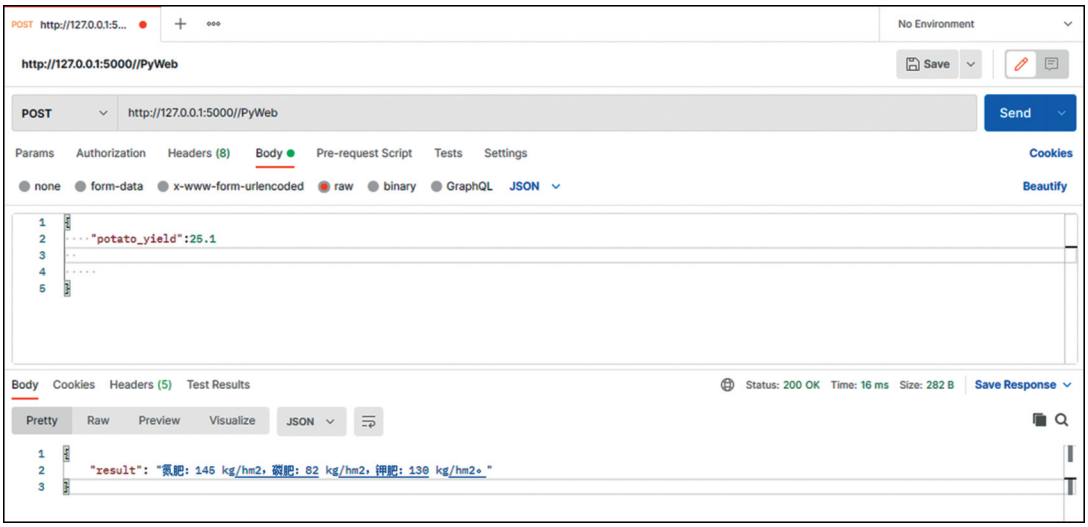

图 8　Python 形态模型服务调用验证

Fig. 8　Python form model service invocation validation

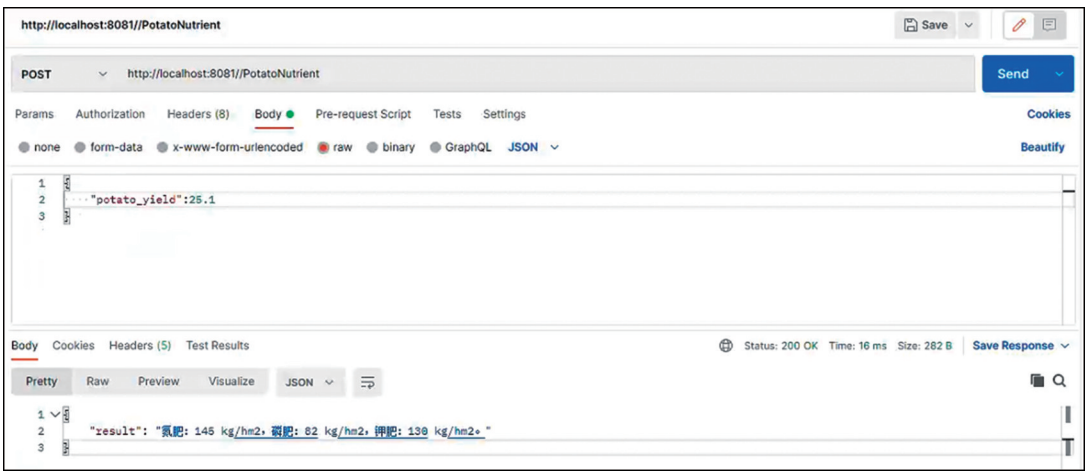

图 9　DLL 形态模型服务调用验证

Fig. 9　DLL form model service invocation validation

4　结论

该文通过设计异构模型标准化封装接口，包括模型描述接口、模型执行接口与模型部署接口，并在此基础上提出模型标准化服务封装方法，实现统一标准化模型语义、数据、运行环境与复杂的底层调用机制，为异态异构的农业模型资源集成管理与复用共享

提供了新的方法，增强了智慧农业系统的适应能力和可扩展能力，并可用于其他领域的模型资源管理与重用共享。

参考文献

[1] 韩巍，曹杰，石智峰，等. 智慧农业发展的现状、问题与对策. 中国农业信息，2023，35(2)：16-22.
[2] 曹宏鑫，葛道阔，张文宇，等. 农业模型发展分析及应用案例. 智慧农业(中英文)，2020，2(1)：147-162.
[3] 胡迪. 地理模型的服务化封装方法研究. 测绘学报，2015(11)：1298-1298.
[4] 谭羽丰，陈旻，张博文，等. 面向虚拟地理环境的Linux平台地理分析模型服务化封装方法. 测绘学报，2018，47(8)：1031-1042.
[5] 陈红燕. C/C++组件式地理分析模型的自动化封装方法研究[硕士论文]. 南京：南京师范大学，2015.
[6] 汪小林，邓浩，王海波，等. Fortran地理模型的拆分与服务化封装. 计算机科学与探索，2011，5(3)：221-228.
[7] 魏圆圆，王雪，王儒敬，等. 基于WebGIS的农场生产管理信息系统的设计与实现. 农业工程学报，2018，34(16)：139-147.
[8] 谭术魁，邹尚君，曾忠平，等. 基于RF-MLP集成模型的耕地生态安全预警系统设计与应用. 长江流域资源与环境，2022，31(2)：436-446.
[9] 陈栋，吴保国，王姗姗，等. 面向人工林经营的模型库和方法库服务平台. 林业科学，2020，56(1)：87-102.
[10] 余宇峰，胡健伟，严琳，等. 水文模型的服务化封装方法研究与应用. 河海大学学报(自然科学版)，2022，50(4)：34-41.
[11] 朱倩. 面向多领域服务的GIS模型多层级共享及封装方法研究[硕士论文]. 杭州：浙江大学，2020.
[12] 方艺辉，邹长忠，吴国祥. 面向复杂决策的异构水环境模型表示与一体化集成. 武汉大学学报(理学版)，2022，68(6)：635-643.
[13] Liu H，Liu X，Xu Z. Research on model component method based on OpenMI technology. *E3S Web of Conferences. EDP Sciences*，2021，260：03005.
[14] Zhang F，Chen M，Kettner A J，et al. Interoperability engine design for model sharing and reuse among OpenMI，BMI and OpenGMS-IS model standards. *Environmental Modelling & Software*，2021，144：105164.
[15] 宁琳懿睿，仇少君，徐新朋，等. 基于产量反应和农学效率的马铃薯智能化推荐施肥. 植物营养与肥料学报，2023，29(12)：2272- 2281.

Research on service-oriented encapsulation method for heterogeneous agricultural models*

Lu Hongwei[1]，Pan Yuchun[2※]

(1. School of computer science and communication engineering，Jiangsu University，Zhenjiang，212000，Jiangsu，China；2. Beijing academy of agriculture and forestry sciences information technology research center，Beijing 100097，China)

Abstract：[**Purpose**] Agricultural models，spanning across various disciplines and domains，constitute the cornerstone of smart agriculture. Shielding the heterogeneous attributes，such as development languages and invocation methods of agricultural models，is crucial for enabling efficient integration and sharing of agricultural model resources. This is paramount for facilitating scientific agricultural management and decision making，thereby fostering sustainable，efficient and secure agricultural development. [**Method**] This paper designed standardized encapsulation interfaces，including model description interfaces，model execution interfaces，

and model deployment interfaces. Based on this, it proposed standardized service encapsulation methods tailored for different forms of model runtime bodies such as COM components, Jar packages, R language, Python language, and DLLs. The feasibility and practicality of the heterogeneous model service encapsulation methods were validated using an example of a potato intelligent fertilization recommendation model implemented in both Python and DLL forms. [**Result**] (1) After encapsulation, the model generated standardized description documents, completed the transformation of model entry point functions, and integrated third-party dependency libraries. As a result, the model could be directly executed via network requests, reducing the cost of model usage. (2) Taking Yunnan Province as an example, the standardized model services post-encapsulation were invoked and operated. Inputting the target potato yield, optimal fertilizer recommendations for that specific yield were obtained, validating the feasibility and practicality of the heterogeneous model service encapsulation methods. [**Conclusion**] This method effectively shields discrepancies in semantics, data, form, and runtime environments of models, thereby promoting the efficient sharing and reuse of agricultural model resources in network environments. It enhances the management capability of model resources in the domain of smart agriculture.

Key words: agricultural models; heterogeneous and diverse models; model resource sharing; standardized service encapsulation