强类型 JSON 数据交换接口生成器的设计与实现

黄向平[1,2] 刘中一[1,2] 阎松柏[1,2]
HUANG Xiangping LIU Zhongyi YAN Songbai

摘 要 介绍了一个 C++ 开发环境中利用 JSON schema 与 C++ 数据类的动态映射工具（Mapper），所提出工具的特点是借鉴 WSDL2Code 原理，充分利用强类型高级面向对象语言的特性，消除了手工编写用于解析和生成 JSON 结构的数据交换接口代码的笨拙编程方式，不仅提高了开发效率而且数据交换接口统一规范，提升了代码可维护性。同时，借助开发平台通用的智能感知（Intellisenese）技术，实现了接口字段的便捷键入，进一步提高了开发的效率，降低了程序员发生低级失误的可能。所提出的框架由 Python 编码完成，无须第三方插件，适用于包含标准模板库（STL）的 C++ 开发环境，已经在实际项目中使用，反馈效果良好。

关键词 Mapper；数据交换接口；JSON schema；面向对象

doi：10.3969/j.issn.1672-9528.2024.05.017

0 引言

JSON[1] 作为一种轻量级的数据交换格式，使用了类似于 C 语言的结构，比 XML 更简洁 [2]，得到了 C、C++、JavaScript、Python 等主流语言的支持，正逐渐替换 XML 成为主流的网络数据传输交换格式 [3-5]。

所谓“强类型 [6]”通常指的是程序中表达的任何对象所从属的类型都必须在编译时刻确定。强类型语言在大规模信息系统开发中具有巨大优势，可以通过类型检查机制在编译过程中发现许多容易被人忽视的错误，从而保证软件的质量，使得大规模的软件集成成为可能。但强类型并非完美，其长期被人诟病的问题是结构过于僵化，适用于结构化数据，而半结构化数据在强类型语言中处理较为困难，造成程序臃肿、结构复杂，增加了开发和理解的难度。

这样的问题在强类型面向对象 C++ 语言与半结构化数据格式 JSON 的数据交换 [7] 中尤为突出。当前主流 JSON 解析工具库支持 DOM 风格的 API，可以很方便地解析 JSON 字符流为 DOM 树状数据结构。这种数据结构具有良好的跨平台、跨语言的通用性，但对于 C++ 语言而言缺少强类型支持，开发过程中无法借助编译器来检查数据类型，只能依赖数据定义 JSON schema 文件，以人工方式检查结构规范。自动化检查需延迟到程序运行期，在对大量 JSON 测试用例解析过程中验证接口代码，大大拉长了开发调试周期。

1. 中国民航信息网络股份有限公司 北京 101318
2. 北京市民航大数据工程技术研究中心 北京 101318

为了解决上述矛盾冲突，可以借助 JSON schema 文件自动生成 [8] 强类型数据结构类代码，开发人员面对的不再是 DOM 模式的数据结构，而是 C++ 对象类 [9]。本文提出的 JSON 与 C++ 数据类动态映射工具 Mapper，自动生成 JSON 数据交换接口代码，正是在这样的指导思想下展开设计与实现的。

1 一般 JSON 数据交换技术

JSON 数据交换技术包括 JSON schema 与 JSON 解析生成工具库。JSON schema 文件是数据交换双方约定的结构规范文件，数据交换双方的开发工作是并行的，可自由选择适合各自应用场景的编程语言。一些实时性要求较高的后台服务系统中，C++ 是一种常用的编程语言，C++ 对 JSON 字符串和对象的转换是用第三方的工具库来完成。

1.1 JSON schema

JSON schema 是一组特殊的 JSON 词汇，用来标记和校验 JSON 数据，也可以理解为一种对 JSON 数据格式定义的约定。JSON schema 使用一种人机都容易理解的方式来描述已有的数据格式， 定义了如何基于 JSON 格式描述 JSON 数据结构的规范，进而提供数据校验、文档生成和接口数据交互控制等一系列能力。

JSON schema 支持以下 7 种基本数据类型：string、number、integer、boolean、null、array、object。其文档本身符合 JSON 格式，均为形如（key:value）的键值对 kv 的集合。key 表示关键字，具有唯一性，数据类型只能是 string；value 表示原子值，可以是 7 种基本数据类型中的任意一种。

1.2 RapidJSON 工具库

RapidJSON 是一种常用的 JSON 解析生成器。其特点是同时支持 SAX 和 DOM 风格的 API，性能较高，不依赖于 STL、BOOST 等外部库，使用内存分配器来快速紧凑地分配内存。

每个 JSON 值都储存为 Value 类，而 Document 类则表示整个 DOM，它存储了一个 DOM 树的根 Value。RapidJSON 的所有公开类型及函数都在 rapidjson 命名空间中。

由于 DOM 树是一种非强类型的数据结构，开发人员在使用接口时必须严格按照 JSON schema 来选择字段，当接口字段繁杂时容易出错，在编译期无法得到检查，直至单元测试等环节才可能发现错误，造成开发调试周期过长。

2 Mapper 架构的分析与设计

JSON 数据交换接口生成器 Mapper（如图 1）实现了 JSON 文本到 C++ 接口数据类的自动映射，开发人员只需关注业务数据类接口，JSON 文本解析与生成逻辑代码由 Mapper 依据 JSON schema 文件自动产生。生成的代码是强类型数据类，程序员使用接口时在代码编辑与编译期间即可获得错误检查与报警。

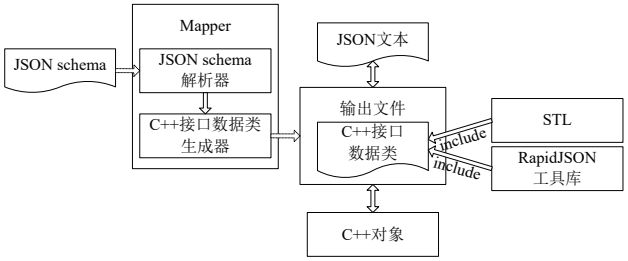

图 1 Mapper 架构

Mapper 工具开发采用了当前流行的 Python 脚本语言，其包括两个功能模块：JSON schema 解析器和 C++ 接口数据类生成器。Mapper 的输出是依赖于标准模板库 STL 与 RapidJSON 工具库的 C++ 数据类的头文件和源代码文件。

2.1 JSON schema 解析器

JSON schema 解析器的功能是解析 JSON schema 文件，生成有关联关系的 JSON 数据类图，解析器设计类图如图 2 所示。全局数据类存储在 key-value 结构的 DefinitionMap，key 为数据类名，value 是数据类定义。所有的数据类定义都继承了 JsonType 基类。JsonString、JsonBoolean、JsonInteger、JsonNumber 分别对应了 JSON 的 4 种内置类型，JsonArray 与 JsonObject 对应复杂结构 array 与 object；JsonProperty 表示 object 中的字段属性，从属于 JsonObject 对象；JsonEnum 对应 JSON schema 中的关键字 enum；JsonReference 代表 URI 引用，表示其数据定义在全局数据类字典 DefinitionMap 当中。数据类型在作用域上分为全局类与内部类，全局类可以由 JsonReference 引用，面向 JSON schema 内的所有数据类型，而内部类只作用于某个宿主数据类型之内，不能被外部数据类型所引用，宿主关系由内部数据类的 _parent 引用所表达，其解析流程如图 3 所示。

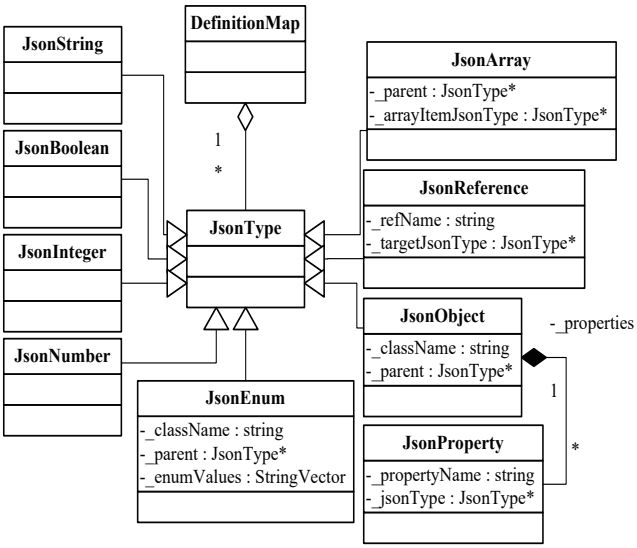

图 2 JSON schema 解析器类关系图

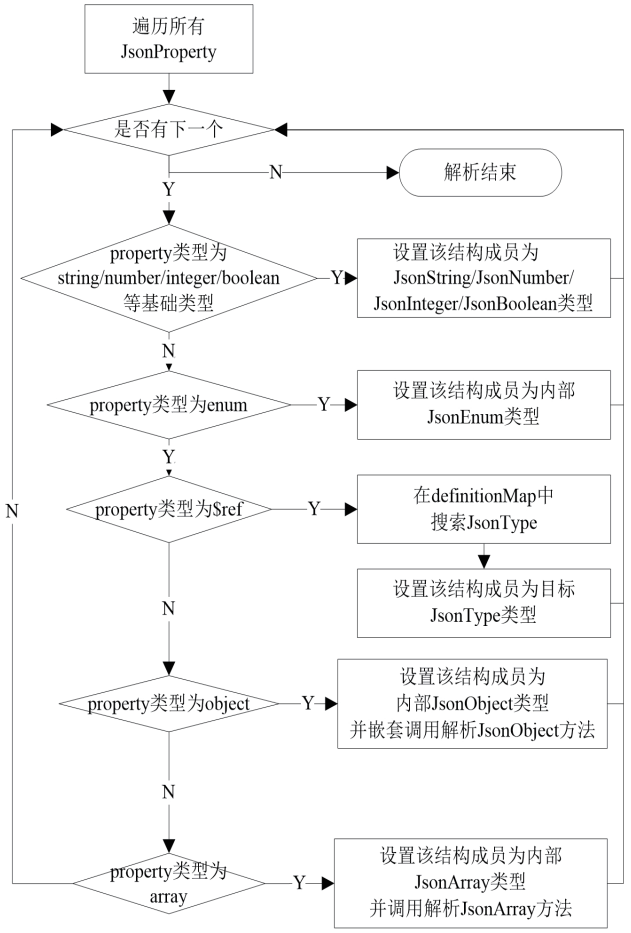

图 3 JSON schema 解析流程图

需要注意的是，JsonReference 可以连环引用，但最终数据定义是 JsonString、JsonBoolean、JsonInteger、JsonNumber、JsonArray、JsonObject、JsonEnum7 种基础类中的一个。

2.1.1 解析 JsonObject

JsonObject 表示 JSON schema 中的 object 类型，包含了一个或多个 Property（JsonProperty）。每个 JsonProperty 的数据类型是 7 种基础类中的任意一种。解析过程如图 4 所示。

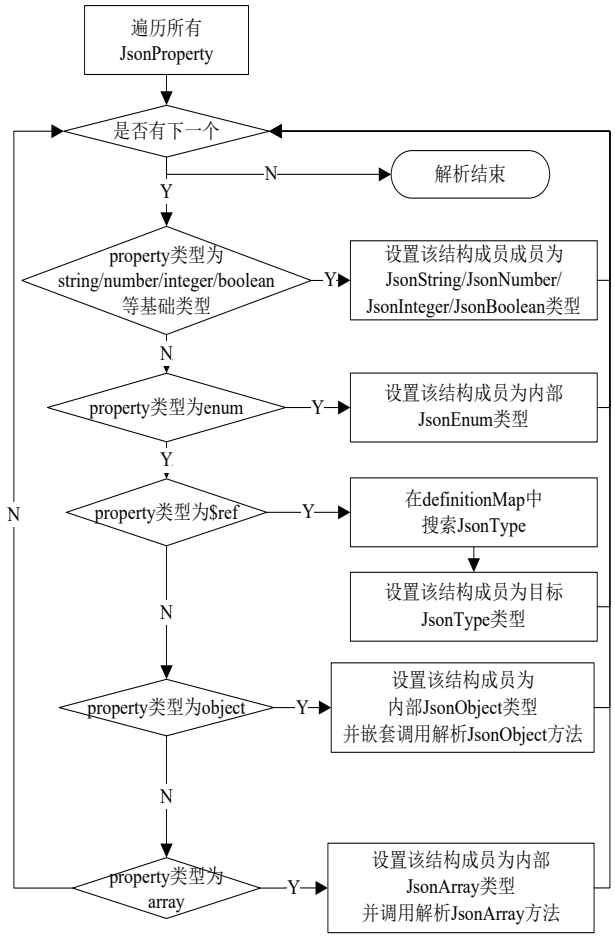

图 4 JsonObject 解析流程图

2.1.2 解析 JsonArray

JsonArray 表示 JSON schema 中的 Array 类型，数组成员类型由 schema items 字段指定，解析后是 7 种基础类中的一种。当 items 是 JsonArray 时，意味着是一个多维数组的结构。解析过程如图 5 所示。

2.1.3 解析 JsonEnum

enum 是 JSON schema 关键字，其 value 是一个 string list，表示 JSON 数据的取值只能是 list 中的某一个。对应到强类型 C++ 语言就是 enum 数据类型，但需要特殊处理的是 JSON 解析过程中数据取值可能超出 list 范围。在生成的 C++ 数据类中，不但存储了 C++ 枚举数值，还存储了完整的字符串，因此可以完全还原原始 JSON 文本内容。

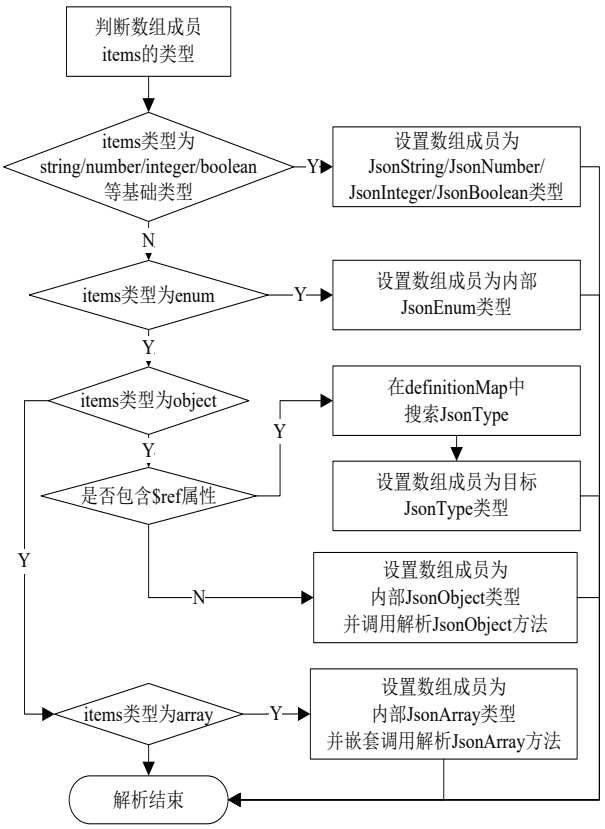

图 5 JsonArray 解析流程图

2.2 C++ 接口数据类生成器

C++ 接口数据类生成器依据上述生成的 JsonType 关系图，生成 C++ 头文件和源代码文件。其中每个全局数据类为一个独立的类文件，所有引用到该全局数据类的头文件都 include 该头文件。所有内部类不生成独立的文件，而处在宿主类文件中。每一个数据类都包含了 JSON 解析与生成等两个函数接口，其函数原型如下：

bool deserialize(const rapidjson::Value& jsonValue, std::ostream& out);

void serialize(rapidjson::Document::AllocatorType& allocator, rapidjson::Value* jsonValue) const;

deserialize 函数是解析函数，输入参数 jsonValue 是 RapidJSON 中的 JSON 数据对象，由 rapidjson::Document 的 Parse 函数解析 JSON 文本得到，输出参数 out 是标准模板库中的输出流基类，可以是字符串或文件，记录解析过程中的报警信息。

serialize 函数是 JSON 文本生成函数，输入参数 allocator 是 RapidJSON 内存分配器，输出参数 jsonValue 可以由 rapidjson::Writer 输出为 JSON 文本。

3 Mapper 应用实例

通过对 Mapper 的逻辑结构分析，可以对 Mapper 的基本

功能有一个比较细致的了解。

Mapper 工具使用方法为启动 Python 脚本并输入自定义参数，格式 为 Python mapper.py namespace prefix header_dir source_dir json_schema_paths。namespace 指定生成数据类所处的 namespace，prefix 是类名的前缀，header_dir 和 source_dir 分别指定头文件与源代码文件的存储目录，json_schema_paths 指定一个或多个 JSON schema 文件。

下面通过一个实例来展示 Mapper 是如何工作的。某民航客票改签服务系统中，改签输入输出接口采用结构复杂的 JSON 格式，包含了航班、运价、旅客、代理人、原客票、改签价格等信息，其改签结果主要类关系见图 6。

Mapper 生成的 C++ 文件 255 个，代码行数 103 946，包含全局类 121 个，内部类 223 个。自动生成代码的方式避免了大量人工编写代码和调试工作。在应用系统内部代码使用接口时，由于是强类型结构类，在诸如 eclipse 等集成开发环境中实现了接口字段的自动补全等提示，对于非法字段使用在编辑代码时期即有错误标识，较之在编译期间报警，进一步提高了编码工作的效率。

具体以 responseStatusBasic 结构为例，其 JSON schema 定义如下：

```
"responseStatusBasic": {
 "type": "object",
 "properties": {
    "liveDate": {"$ref": "#/dateTime"},
  "messages": {
   "type": "array",
    "items": {
     "type": "object",
      "properties": {
            "messageType": {"enum": ["WARNING", "ERROR",
```

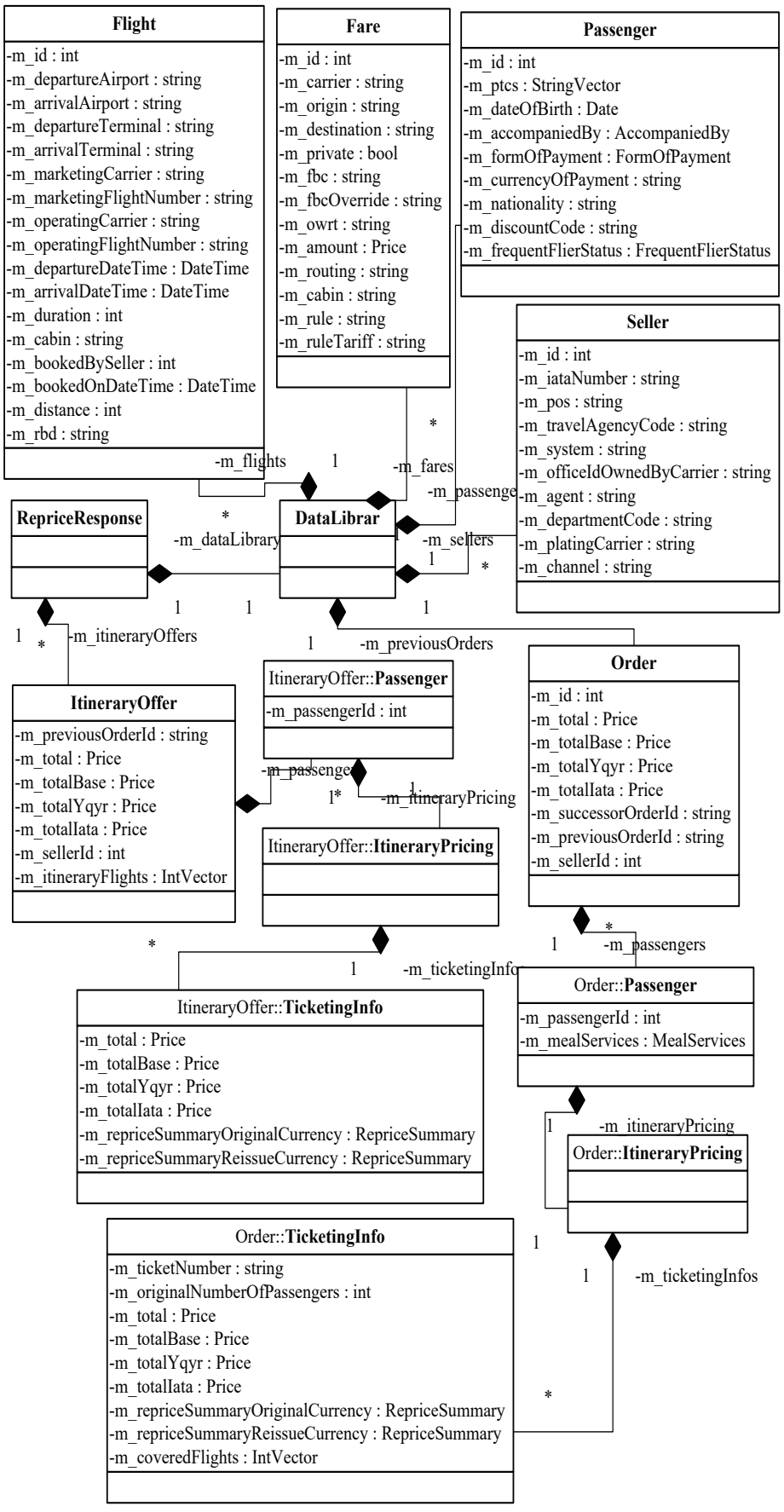

图 6 实例类关系图

```
"NOTE"]},
                "text": {"type": "string"}
            }
        }
    }
}
```

该定义包含了 JsonObject、JsonArray、JsonEnum、JsonString、JsonReference 等几种数据类型，Mapper 生成的 C++ 数据接口代码如下：

```
class Prefix_ResponseStatusBasic{
public:
  class Messages_ArrayItem{
  public:
    class MessageTypeEnum{
    public:
      enum{WARNING = 0,ERROR,NOTE};
    private:
      int m_enumNumber;
      std::string m_enumString;
    };
  private:
    MessageTypeEnum m_messageType;
bool m_messageType_Exist_;
    std::string m_text;
bool m_text_Exist_;
  };
private:
Prefix_DateTime m_liveDate;
bool m_liveDate_Exist_;
std::vector<Messages_ArrayItem> m_messages;
bool m_messages_Exist_;
};
```

代码隐去了非关键代码，如 getter、setter、serialize、deserialize、constructor 等类成员函数，重点介绍类成员变量。Messages_ArrayItem 与 MessageTypeEnum 是内部类，只能被内部引用，而 Prefix_ResponseStatusBasic 是全局类，可以被 schema 中其他结构引用，正如全局类 Prefix_DateTime 被当前类所引用一样。其他以 _Exist_ 结束的字段，如 m_liveDate_Exist_，当 m_liveDate 对应的 JSON 字段空缺或为 null 时，该 boolean 值为 false。

4 结语

经实测，Mapper 对于开发 JSON 数据交换接口具有重要作用，可以很好地支持快速程序开发。Mapper 的输出代码只是对于 RapidJSON 的一种轻量级封装，与开发人员手工编写接口代码没有本质的不同。因此，无需额外的学习成本就可以轻松掌握，并可以直接获得编辑期、编译期排错的便利。同时，借助集成开发环境的智能感知（Intellisenese）技术，实现了接口字段的便捷键入，进一步提高了开发的效率，并降低了程序员发生低级失误的可能。采用 JSON 技术的应用系统日益增多，Mapper 不仅适用于数据交换场景，同样也适用于数据序列化、反序列化 [10] 等应用场景，是一种高效、安全便捷的 JSON 解析生成技术。

参考文献：

[1] BRAY T. The JavaScript object notation(JSON) data interchange format[J]. RFC, 2014,8259;1-16.

[2] 高静 , 段会川 .JSON 数据传输效率研究 [J]. 计算机工程与设计 , 2011,32(7):2267-2270.

[3] 嵇亮亮 , 朱木 , 翟鹏 . 基于 JSON 和 Base64 的地理信息数据交互方法 [J]. 电子质量 ,2023(5):63-67.

[4] 黄秀丽 , 陈志 . 基于 JSON 的异构 Web 平台的设计与实现 [J]. 计算机技术与发展 ,2021,31(3):120-125.

[5] 胡章兵 , 左良利 . 时态 JSON 数据模型及查询语言处理 [J]. 计算机技术与发展 ,2019,29(10):141-145.

[6] 商陆军 . 面向对象的程序设计语言中的强类型问题 [J]. 计算机学报 ,1991,14(10):721-729.

[7] 张沪寅 , 屈乾松 , 胡瑞芸 . 基于 JSON 的数据交换模型 [J]. 计算机工程与设计 ,2015,36(12):3380-3384.

[8] 林美辰 , 冯志勇 , 陈世展 , 等 . 一种半自动化构建 Web 服务适配器的方法 [J]. 小型微型计算机系统 ,2017,38(2):323-327.

[9] 雷明涛 , 何赟 . 基于数据字典的自适应的对象化数据库访问技术研究 [J]. 微电子学与计算机 ,2015,32(10):21-25.

[10] TAO Z, QIANG H, LEI Y M. Algorithm of object serialization based on JSON[J]. Computer engineering and applications, 2007, 43(15): 98-100.

【作者简介】

黄向平（1982—），男，浙江金华人，硕士，高级工程师，研究方向：民航客票运价系统、高性能计算架构体系。

刘中一（1987—），男，山西大同人，硕士，高级工程师，研究方向：民航客票运价系统、密集计算系统。

阎松柏（1981—），男，重庆人，本科，研究方向：民航客票运价系统、国际国内的运价系统建设。

（收稿日期：2024-03-05）