深入理解并实现基本的 JSON 解析器

黄京

Sep 21, 2025

1 副标题：抛开现成的库，亲手打造一个解析器，彻底掌握 JSON 的本质。

JSON（JavaScript Object Notation）作为现代数据交换的事实标准，几乎无处不在。从 Web API 的响应到配置文件的存储，JSON 以其轻量级和易读性赢得了广泛的应用。我们日常开发中经常使用如 Python 的 `json.loads()` 或 JavaScript 的 `JSON.parse()` 来解析 JSON 数据，但这些现成库的背后机制却往往被忽略。超越简单的导入和使用，亲手实现一个 JSON 解析器，不仅能帮助我们深入理解编译原理的基础知识如语法、词法和状态机，还能在遇到非标准或自定义数据格式时提供自主解决问题的能力。此外，这个过程极大地锻炼了编程技能、对细节的把握和调试能力。本文的目标是使用 Python 语言从零实现一个功能完备的 JSON 解析器，并通过官方的 JSONTestSuite 进行测试，以确保其正确性和健壮性。

2 背景知识：JSON 格式规范与解析器概述

JSON 的语法规范基于 RFC 7159，其基本结构包括对象（用花括号 {} 表示）和数组（用方括号 [] 表示），以及基本类型如字符串、数字、布尔值（`true` 或 `false`）和 `null`。重要规则包括键必须用双引号包裹、禁止尾部逗号等。解析 JSON 字符串通常涉及两个核心步骤：词法分析和语法分析。词法分析负责将字符流分解为有意义的词元（Token），类似于将句子拆分成单词；语法分析则根据 Token 流按照语法规则构建内存中的数据结构，如 Python 的字典或列表。整个解析器的架构可以概括为：JSON 字符串输入到词法分析器（Lexer）生成 Token 流，再传递给语法分析器（Parser）输出 Python 对象。

3 第一步：构建词法分析器（Lexer）

词法分析器的核心任务是识别和生成 Token。Token 类型包括结构字符如 {, }, [,], :, ,，以及值类型如 `STRING`, `NUMBER`, `TRUE`, `FALSE`, `NULL`。Lexer 的工作流程是循环遍历输入字符串，使用条件判断和状态机来识别这些 Token。实现过程中，需要处理空白字符（如空格、制表符、换行符）的跳过，以及解析字符串时的转义序列（例如 \\, \, \n, \uXXXX）。数字解析涉及识别负号、整数部分、小数部分和指数部分，策略通常是持续读取相关字符后统一用 `float()` 转换。字面量如 `true`, `false`, `null` 则通过匹配关键字来识别。
以下是一个简单的 Lexer 类框架代码示例：

```
class Lexer:
    def __init__(self, input_string):
        self.input = input_string
```

```
        self.position = 0
        self.current_char = self.input[self.position] if self.input else None

    def advance(self):
        self.position += 1
        if self.position < len(self.input):
            self.current_char = self.input[self.position]
        else:
            self.current_char = None

    def skip_whitespace(self):
        while self.current_char is not None and self.current_char.isspace():
            self.advance()

    def next_token(self):
        # 实现 Token 识别逻辑
        pass
```

在这个代码中，__init__ 方法初始化输入字符串和当前位置，advance 方法移动指针到下一个字符，skip_whitespace 用于跳过空白字符。next_token 方法是核心，它根据当前字符判断 Token 类型。例如，如果字符是 {，则返回一个表示左花括号的 Token；如果是双引号，则开始解析字符串。字符串解析需要处理转义序列，这是一个难点，因为必须正确识别和转换如 \n 为换行符。数字解析则涉及收集所有数字相关字符（包括符号、小数点、指数），然后使用 float() 进行转换，但需要注意错误处理，例如无效数字格式。

4 第二步：构建语法分析器（Parser）

语法分析器采用递归下降解析法，这是一种直观的方法，适合 JSON 这种上下文无关文法。每个语法规则对应一个解析函数。入口函数是 parse()，它根据第一个 Token 决定解析为对象或数组。parse_object() 函数处理花括号内的键值对，循环读取直到遇到右花括号；parse_array() 函数处理方括号内的值列表；parse_value() 是核心分发函数，根据当前 Token 类型调用相应的解析函数，如 parse_string 或 parse_number，或递归调用自身处理嵌套结构。

错误处理是关键部分，例如在预期冒号分隔键值对时却遇到其他 Token，应抛出清晰异常如“Expected ':' after key”。解析器需要与 Lexer 协同工作，确保在解析完一个结构后，Lexer 的位置正确指向下一个 Token。以下是一个 Parser 类的框架代码：

```
class Parser:
    def __init__(self, lexer):
        self.lexer = lexer
        self.current_token = self.lexer.next_token()

    def eat(self, token_type):
```

```
        if self.current_token.type == token_type:
            self.current_token = self.lexer.next_token()
        else:
            raise Exception(f"Expected␣{token_type},␣got␣{self.current_token.type}")

    def parse(self):
        if self.current_token.type == 'LBRACE':
            return self.parse_object()
        elif self.current_token.type == 'LBRACKET':
            return self.parse_array()
        else:
            return self.parse_value()

    def parse_object(self):
        # 解析对象逻辑
        pass
```

在这个代码中，`__init__` 方法接收 Lexer 实例并获取第一个 Token。`eat` 方法用于消耗预期类型的 Token，如果类型不匹配则抛出错误。`parse` 方法根据当前 Token 类型决定解析方向。`parse_object` 函数会循环读取键值对，每次读取一个字符串键、冒号、值，并处理逗号分隔。递归下降法的优势在于代码结构清晰，易于理解和调试，但需要 careful handling of recursive calls to avoid infinite loops。

5 整合与测试

将 Lexer 和 Parser 整合为一个函数 `my_json_loads(s)`，它接收 JSON 字符串并返回 Python 对象。基础测试用例包括简单 JSON 如 `{key: value}`，应解析为字典；数组如 `[1, true, null]`，应解析为列表。挑战性测试涉及嵌套结构，例如多层对象或数组，以验证递归处理的正确性。错误处理测试包括非法输入如缺少逗号或键名无双引号，解析器应提供有用的错误信息而非崩溃。最后，引入 JSONTestSuite 进行自动化测试，确保解析器符合官方标准，例如处理边缘情况如空字符串或超大数字。

通过实现 JSON 解析器，我们完整经历了从字符串到 Token 再到数据结构的解析过程，加深了对词法分析、语法分析和递归下降法的理解。未来优化方向包括性能提升（如使用生成器惰性产生 Token）、功能扩展（添加自定义参数如 `parse_float`）和增强鲁棒性（改进错误恢复机制）。鼓励读者以此为基础，探索更复杂格式如 XML 或 TOML 的解析，进一步提升编译原理技能。

6 附录 & 进一步阅读

完整代码可参考 GitHub 仓库示例。参考资源包括 RFC 7159 标准文档、JSONTestSuite 项目以及经典书籍《编译原理》（俗称龙书）的相关章节。这些资料有助于深入理解解析器设计和实现细节。