

# 字段注册与构建调度系统设计文档

目标：构建一套通用、声明式、可扩展的 Apidom Element 字段注册与构建系统，支持字段提取、Fold 处理、schema 比对、UI schema 导出等核心功能。









## 一、背景与动机

当前在构建 Apidom Element（如 Encoding、Info、Header 等）时，字段处理逻辑采用手工 match：

- 字段匹配硬编码；
- 缺乏字段清单；
- 无法与 OpenAPI Schema 校验一致性；
- 不便于生成 UI schema 或类型声明；
- 不利于扩展或模块化构建。

因此需要构建一套结构驱动的字注册与调度系统，实现解耦、规范、自动化。

## 二、目标能力清单

能力	说明
 字段结构提取	自动从 OpenAPI 3.1 schema AST 中提取字段定义
 字段注册声明	为每个 Element 注册字段名、处理函数
 构建调度	构建器中字段自动派发到处理器
 Fold 集成	字段值可通过 <b>Fold</b> 折叠器预处理
 Schema 比对	自动校验注册字段是否匹配 schema 定义
 Fallback 控制	根据 schema 中 <b>additionalProperties</b> 控制未知字段处理
 UI schema / DTO 生成	可导出字段结构用于 UI 或多语言类型声明
 模板生成	支持从 schema 自动生成字段注册模板代码

## 三、系统模块组成

### 1. Schema 解析模块

- **schema\_loader.rs**: 读取并解析 OpenAPI JSON Schema 为 Apidom **Element** AST
- **dereference\_pass.rs**: 解引用 **\$ref**，确保完整结构用于提取

### 2. 字段结构提取模块

- **extract\_fieldspecs(&Element) -> Vec<FieldSpec>**:
  - 解析 **properties**、**required**、**patternProperties**

- 生成 `FieldSpec { name, type_hint, required, from_pattern }`

### 3. 字段注册与宏模块

- `register_fixed_fields!` 宏：注册字段名与对应 handler
- `register_pattern_fields!` 宏：注册扩展字段（如 `^x-`）
- `FieldHandlerMap<T>` 类型：`HashMap<&str, FieldHandlerFn<T>>`

### 4. 构建器派发逻辑

- `builder_dispatch.rs`：接收一个字段名，根据 handler map 自动调用构建逻辑
- 支持 fallback 分支（可自定义控制策略）

### 5. Schema 比对与验证器模块

- `compare_field_specs(schema_fields, registered_fields)`
- 可检测：缺失字段、多余字段、类型不符、fallback 越权

### 6. 模板与导出模块（可选）

- `render_register_macro()`：生成 Rust 字段注册模板代码
- `render_ui_schema()`：生成 JSON schema for UI
- `render_dto()`：生成 TypeScript、Dart 或 Rust DTO 类型结构

---

## 四、字段处理开发流程

Step-by-step 工作流：

1. 使用 Apidom 加载 `2022-10-07` schema → Element AST
2. 解引用 schema 中所有 `$ref`
3. 提取某个类型（如 Encoding）的字段结构为 `Vec<FieldSpec>`
4. 自动生成 `register_fixed_fields!` 模板代码
5. 开发者填写每个字段的处理逻辑（handler 函数体）
6. 构建器中使用统一分发机制自动调用 handler
7. 可选比对字段一致性，或导出表单结构 / 类型声明

---

## 示例代码模板

```
register_fixed_fields!(EncodingElement, {
  // contentType: string
  "contentType" => |val, el, folder| {
    let v = fold_and_convert_string(val, folder)?;
    el.set_content_type(v);
    Some(())
  },
  // headers: object
  "headers" => |val, el, folder| {
```

```

        let v = fold_and_convert_object(val, folder)?;
        el.set_headers(v);
        Some(())
    },
});

```

## ✓ 五、设计优势与扩展性

特性	支持	说明
多 Element 复用	✓	所有构建器共用一套注册和调度机制
Fold 插件化	✓	所有字段支持统一折叠预处理
多语言支持	✓	可生成 TypeScript/Dart/Rust 类型
Schema 演化适配	✓	自动与 OpenAPI 3.1 保持一致
动态导出	✓	可用于 UI builder、代码生成、文档
字段追踪与元信息注入	✓	可插入 source、specPath 等 metadata

## 🚧 六、可选拓展模块

- `validate_required_fields!` 宏（在运行时检查是否遗漏必填字段）
- `define_element_builder!` 宏（统一生成某 Element 的 builder + 注册 + dispatch）
- `builder_test_suite!`（自动生成测试用例验证字段构建完整性）

## 🧭 七、典型使用路径

场景	动作
想实现一个新 Element 构建器	提取 schema → 自动生成字段模板 → 写 handler
想验证现有 builder 与 schema 一致	比对 schema 字段与注册字段
想生成 UI 配置表单结构	从 FieldSpec 输出 JSON schema
想支持 TS 类型导出	渲染 DTO 结构体模板
想插入 Fold AST 操作	所有 handler 统一接收 <code>&amp;mut Fold</code>

## ✓ 八、总结

本系统提供了一套可复用、结构驱动、完全可扩展的字段注册与构建器调度机制。它兼容 Apidom 的 Element AST 与 Fold 架构，同时具备：

- 自动结构提取
- 统一注册管理

- 构建逻辑分发
- Schema 对齐验证
- 元数据追踪注入
- UI 与代码生成支撑

未来可以作为 Apidom / Stepflow / OpenAPI 系统的核心组成部分，支持 DSL、AI Schema、插件系统等进一步扩展。