

# Funny JSON Explorer（进阶）设计文档

---

21307362 郑博铨

[Funny JSON Explorer（进阶）设计文档](#)

[设计要求](#)

[类图](#)

[类说明](#)

[迭代器实现](#)

[策略实现](#)

[设计模式](#)

[迭代器模式](#)

[策略模式](#)

## 设计要求

对已有的FJE软件设计进行设计重构，改用迭代器+策略模式。

## 类图



- `next(self)`: 返回当前索引位置的元素, 并将索引加1。如果没有更多元素, 抛出 `StopIteration` 异常。

## 策略实现

该策略主要是对原来 **Component** 类中的 `draw` 函数使用策略实现, 对于不同风格的 `draw` 使用不同策略实现

**StyleStrategy**类: 这是一个抽象基类, 用于定义风格策略的接口, 包含三个抽象方法:

- `execute(self, is_leaf)`: 执行绘制操作, 根据节点类型选择绘制方法。
- `draw_container(self)`: 绘制容器节点。
- `draw_leaf(self)`: 绘制叶子节点。

**TreeStyleStrategy**类: 这是具体的树形风格策略类, 继承自 `StyleStrategy`。

- `__init__(self, node, prefix)`: 构造方法, 接受一个节点和前缀参数。
- `execute(self, is_leaf)`: 根据节点类型调用相应的绘制方法。
- `draw_container(self)`: 绘制容器节点, 使用前缀和图标表示层次结构。
- `draw_leaf(self)`: 绘制叶子节点, 显示节点名称和值。

**RectangleStyleStrategy**类: 这是具体的矩形风格策略类, 继承自 `StyleStrategy`。

- `__init__(self, node, prefix)`: 构造方法, 接受一个节点和前缀参数。
- `execute(self, is_leaf)`: 根据节点类型调用相应的绘制方法。
- `draw_container(self)`: 绘制容器节点, 使用前缀和图标表示层次结构。
- `draw_leaf(self)`: 绘制叶子节点, 显示节点名称和值。

## 设计模式

### 迭代器模式

迭代器模式在上面的代码中通过抽象类 `Iterator` 和具体实现类 `ComponentIterator` 来实现。

`Iterator` 类定义了两个抽象方法 `has_next()` 和 `next()`, 用于遍历集合中的元素。

`ComponentIterator` 类实现了这些方法, 封装了遍历组件集合的逻辑。

在策略类 `TreeStyleStrategy` 和 `RectangleStyleStrategy` 中, 使用了 `ComponentIterator` 来遍历节点的子节点。例如, 在 `draw_container()` 方法中, 使用迭代器对象 `iterator` 遍历节点的子节点, 并调用 `child.draw(new_prefix)` 来绘制每个子节点。这种迭代方式使得代码可以一致地遍历和处理不同类型的组件集合, 而不必暴露集合的内部结构。

### 策略模式

策略模式在上面的代码中通过抽象类 `StyleStrategy` 和具体实现类 `TreeStyleStrategy` 及 `RectangleStyleStrategy` 来实现。`StyleStrategy` 定义了绘制方法的接口 `execute(is_leaf)`, `draw_container()` 和 `draw_leaf()`, 这些方法分别用于执行绘制操作和绘制容器及叶节点。

具体的策略类 `TreeStyleStrategy` 和 `RectangleStyleStrategy` 实现了这些方法, 分别定义了树形和矩形两种不同风格的绘制逻辑。在 `Component` 的子类中可以根据用户的选择动态地实例化不同的策略类, 并调用其 `execute()` 方法来绘制节点。通过这种方式, 代码实现了绘制风格的可扩展性和可替换性, 使得不同的绘制策略可以在运行时进行切换, 而无需修改客户端代码。

