

21307190 潘沛国

```
classDiagram
    class Component {
        -icon
        -name
        -is_first
        -is_leaf
        -is_last
        +draw()
    }
    class Container {
        -level
        +add()
        +draw()
    }
    class Leaf {
        -value
        +add()
        +draw()
    }
    class AbstractFactory {
        -node_factory
        -style_factory
        +get_style()
        +get_node()
    }
    class NodeFactory {
        +create_root()
        +create_container()
        +create_leaf()
    }
    class StyleFactory {
        +get_style()
    }
    class TreeStyleFactory {
        +get_style()
    }
    class RectangleStyleFactory {
        +get_style()
    }
    class Style {
        -first
        -body
        -last
        +style_config()
        +set_style_config()
        +get_style_config()
    }
    class TreeStyle {
        +set_style_config()
    }
    class RectangleStyle {
        +set_style_config()
    }

    Component "1" *-- "1" Container
    Component "1" *-- "1" Leaf
    Container "1" o-- "*" children
    Container "1" <--> "1" Leaf : get_node().create
    AbstractFactory "1" <.. "1" NodeFactory : Use
    AbstractFactory "1" <.. "1" StyleFactory : Use
    AbstractFactory "1" <.. "1" TreeStyleFactory : Use
    AbstractFactory "1" <.. "1" RectangleStyleFactory : Use
    AbstractFactory "1" <.. "1" Style : Use
    NodeFactory "1" <|-- "1" TreeStyleFactory
    NodeFactory "1" <|-- "1" RectangleStyleFactory
    StyleFactory "1" <|-- "1" TreeStyleFactory
    StyleFactory "1" <|-- "1" RectangleStyleFactory
    Style "1" <|-- "1" TreeStyle
    Style "1" <|-- "1" RectangleStyle
```

工厂方法的使用:

- 1.StyleFactory 类是一个抽象工厂基类, 提供一个创建获取不同可视化风格的接口 `get_style`。`TreeStyleFactory` 和 `RectangleStyleFactory` 继承自 `StyleFactory` 并实现了 `get_style` 方法, 分别用于创建 `TreeStyle` 和 `RectangleStyle` 对象。这三个类提供了创建 `Style` 对象的不同子类的方法。`Style` 是抽象产品, 而 `TreeStyle` 和 `RectangleStyle` 是它的子类, 为具体产品。
- 2.NodeFactory 类, 其展示工厂方法的使用。这个类提供三个静态方法/接口 `create_root`、`create_container`、`create_leaf`, 三个方法/接口用于创建不同类型的 json 节点, 由这个工厂类决定要实例化(创建)的节点是哪种类型。

工厂方法的作用：

满足开闭原则，由此允许在不修改原有代码的情况下，通过引入新的具体产品类和对应的具体工厂类来扩展 style 的种类，或者是拓展其他的功能。

抽象工厂模式的使用：

这里由于底层设计不当,导致实际上并不能完全体现抽象工厂模式的思想。在实际实现中, AbstractFactory 提供两个接口,一个用于创建风格,一个用于创建节点。理论上 node_factory 和 style_factory 应该负责创建一组相互关联的产品。但由于底层设计的不当,实际上风格与节点这两个并不互相依赖和关联,导致到并不能很好地体现抽象工厂模式的思想。由于本次作业截止的时间快到了,我也没有时间再进行修正完善.会争取下一次完善。

抽象工厂模式的作用：

客户端与具体产品解耦隔离，客户端不必了解细节，有助于简化代码；同时客户端与具体产品的创建过程被解耦，提高了程序的灵活性和可扩展性。

组合模式的使用：

Component 类是抽象基类,定义了所有组件的公共接口，如方法 draw 。

Container 类是容器节点,为复合对象，可以包含其他 Component 对象，其重写的 draw 能够递归地绘制其子组件。

Leaf 类是叶子节点，其不会包含其他 Component 对象。

组合模式的作用：

便于以统一的方式处理组件对象，有助于简化客户端代码，有利于提高代码的层次结构，提高代码的可拓展性。

功能简介

FJE 可以快速切换**风格** (style)，包括：树形 (tree)、矩形 (rectangle)，

也可以指定**图标族** (icon family)，为中间节点或叶节点指定一套 icon。

如果你想增加风格,那你需要添加必须的抽象工厂。

如果你想增加图标族,请直接修改配置文件`./config/icon_family.json`。