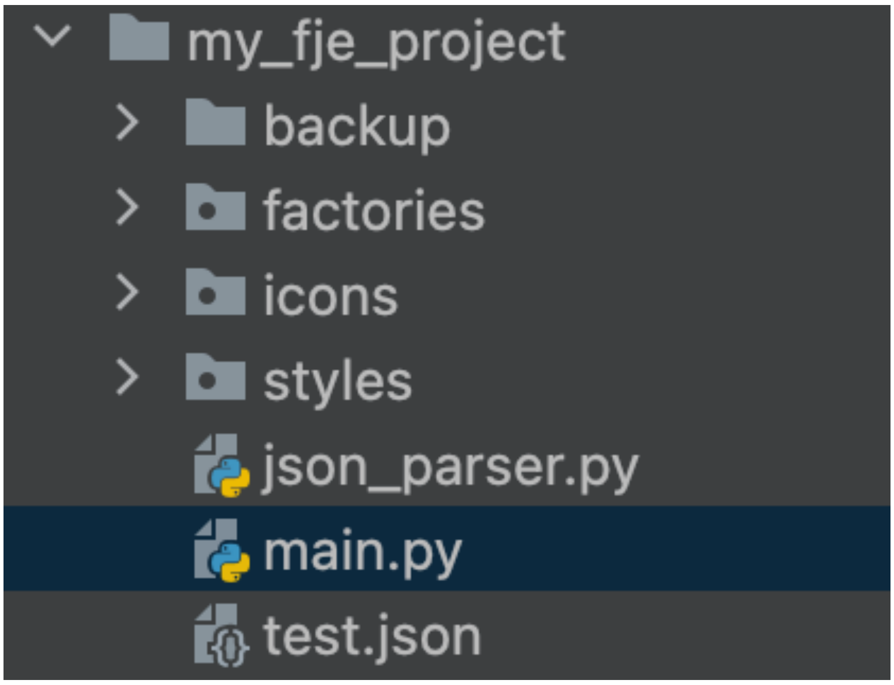
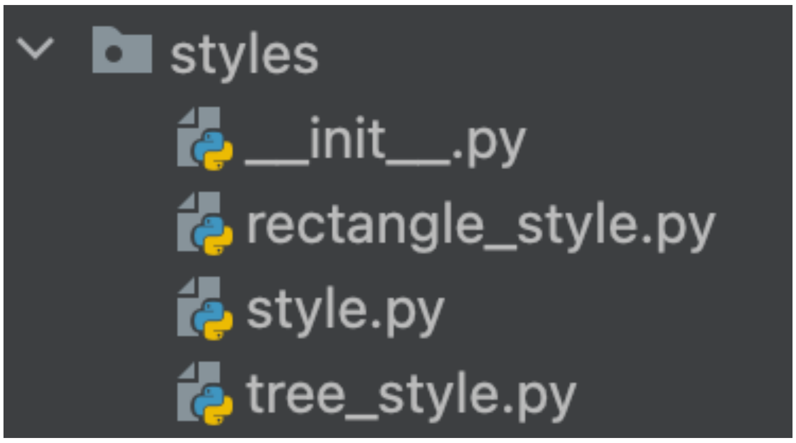
介绍代码框架与模式

21307420 庄钰堃



```
my_fje_project
  backup
  factories
  icons
  styles
  json_parser.py
  main.py
  test.json
```

首先分析一下代码结构，上图有用的文件夹有3个—— factories、icons 和 styles



```
styles
  __init__.py
  rectangle_style.py
  style.py
  tree_style.py
```

1 . style.py：采用的是抽象工厂模式，详情见下面

```
class Style(ABC):
    @abstractmethod
    def display(self, data, icon_family):
        pass
```

1.1 tree_style.py: 使用的是组合模式， 而接口TreeStytle作为抽象工厂的具体工厂的接口

```python
import json
from styles.style import Style

class TreeStyle(Style):
    def display( self, data, icon_family):
        icon = icon_family.get_icon()
        print_tree(data, icon)


class Component:
    def display_node(self, icon, prefix="", is_last=True):
        raise NotImplementedError("You should implement this method")


class Composite_Node(Component):
    def __init__(self, name, syntax="null", children=None):
        self.name = name
        self.syntax = syntax
        self.children = children if children is not None else []

    def display_node(self, icon, prefix="", is_last=True):
        connector = "└─" if is_last else "├─"
        line = prefix + connector + icon + self.name
        if self.syntax != "null":
            line += ": " + self.syntax
        print(line)

        if self.children:
            new_prefix = prefix + ("    " if is_last else "│   ") # 只有最后一个不用加 横杠，其他都需要，并且基础prefix是不断类加的
            for i, child in enumerate(self.children):
                is_child_last = i == len(self.children) - 1
                child.display_node(icon, new_prefix, is_child_last)


class Leaf_Node(Component):
    def __init__(self, name, syntax="null"):
        self.name = name
        self.syntax = syntax

    def display_node(self, icon, prefix="", is_last=True):
        connector = "└─" if is_last else "├─"
        line = prefix + connector + icon + self.name
        if self.syntax != "null":
            line += ": " + self.syntax
        print(line)
```

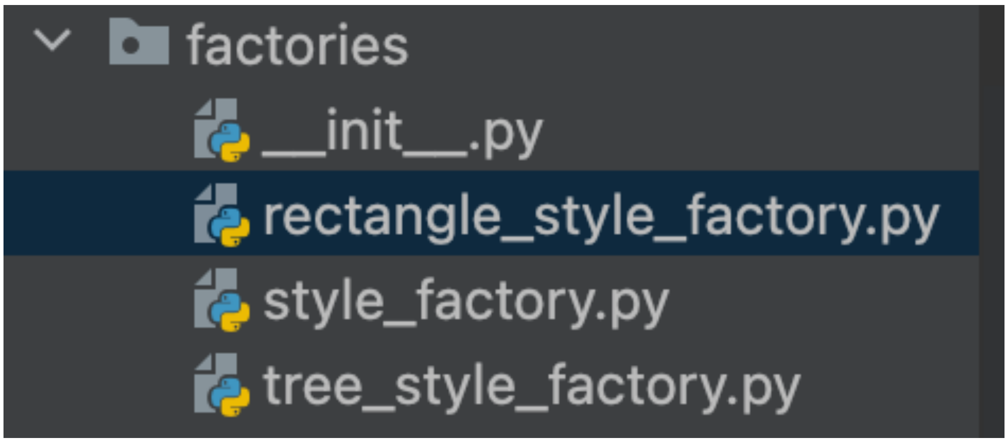
```
def json_to_nodes(name, data):
    # 叶子节点
    if not isinstance(data, dict):
        return Leaf_Node(name, data if data  else "null")
    # 复合节点
    else:
        children = []
        for key, value in data.items():
            child_node = json_to_nodes(key, value)
            children.append(child_node)  # children列表存的是 “结点”, eg: composite-
>composite->leaf
        return Composite_Node(name, syntax="null", children=children)


def print_tree(data, icon):
    # 转换JSON数据为树形结构
    nodes = [json_to_nodes(k, v) for k, v in data.items()]


    # 打印树形结构
    for i, node in enumerate(nodes):
        is_last = i == len(nodes) - 1
        node.display_node(icon, "", is_last)
```

1.2 rect_style.py: 用的是建造者模式，而接口RectangleStyle作为抽象工厂的具体工厂的接口

```
import json
from styles.style import Style

class RectangleStyle(Style):
    def display(self, data, icon_family):
        icon = icon_family.get_icon()
        print_rectangle(data, icon)

class NodeBuilder:
    def set_name(self, name):
        raise NotImplementedError

    def set_syntax(self, syntax):
        raise NotImplementedError

    def add_child(self, child):
        raise NotImplementedError

    def build(self):
        raise NotImplementedError

class RectNodeBuilder(NodeBuilder):
    def __init__(self):
        self.name = None
        self.syntax = "null"
        self.children = []
    def set_name(self, name):
        self.name = name
        return self

    def set_syntax(self, syntax):
        self.syntax = syntax
        return self

    def add_child(self, child):
        self.children.append(child)
        return self

    def build(self):
        return Rect_Node(self.name, self.syntax, self.children)

class Rect_Node:
    def __init__(self, name, syntax="null", children=None):
        self.name = name
        self.syntax = syntax
        self.children = children if children is not None else []

    def display_node(self, prefix="", is_last=False, is_bottom=False, count=0, icon=""):
        if is_bottom:
            connector = "└──┴──" + icon
        else:
            connector = "└─" + icon if is_last else "├─" + icon
        line = prefix + connector + self.name
        if self.syntax != "null":
            line += ": " + self.syntax

        if is_bottom:
            if count == 1:
                line += " " + "─" * (45 - len(prefix) - len(connector) - len(self.name) -
(len(self.syntax) + 2 if self.syntax != "null" else 0)) + "┘"
            else:
                line += " " + "─" * (45 - len(prefix) - len(connector) - len(self.name) -
(len(self.syntax) + 2 if self.syntax != "null" else 0)) + "┤"
        else:
            line += " " + "─" * (45 - len(prefix) - len(connector) - len(self.name) -
(len(self.syntax) + 2 if self.syntax != "null" else 0)) + "┤"
        print(line)

        if self.children:
            new_prefix = prefix + ("    " if is_last else "│   ")
            for i, child in enumerate(self.children):
                is_child_last = i == len(self.children) - 1
                child.display_node(new_prefix, is_child_last, is_bottom, 0, icon)

def json_to_nodes(name, data):
    builder = RectNodeBuilder().set_name(name)
    if isinstance(data, dict):
```

```python
        for k, v in data.items():
            child_node = json_to_nodes(k, v)
            builder.add_child(child_node)
        return builder.build()
    else:
        return builder.set_syntax(data if data is not None else "null").build()

def print_rectangle(data, icon_family):
    icon = icon_family

    nodes = [json_to_nodes(k, v) for k, v in data.items()]

    if nodes:
        top_node = nodes[0]
        print("┌─" + icon + top_node.name + " " + "─" * (44 - len(top_node.name) - 3) +
"┐")
        for child in top_node.children:
            child.display_node("│  ", False, False, 0, icon)

        for node in nodes[1:-1]:
            print("├─" + icon + node.name + " " + "─" * (44 - len(node.name) - 3) + "┤")
            for child in node.children:
                child.display_node("│  ", False, 0, icon)

        bottom_node = nodes[-1]
        print("├─" + icon + bottom_node.name + " " + "─" * (44 - len(bottom_node.name) -
3) + "┤")
        for i, child in enumerate(bottom_node.children):
            is_last_child = (i == len(bottom_node.children) - 1)
            child.display_node("", is_last_child, True, 1 if is_last_child else 0, icon)

if __name__ == "__main__":
    with open("test2.json", "r") as file:
        data = json.load(file)
    print_rectangle(data, '*+')
```

2. 工厂目录



```
factories
    __init__.py
    rectangle_style_factory.py
    style_factory.py
    tree_style_factory.py
```

2.1 sytle_factory.py

```
class StyleFactory(ABC):
    @abstractmethod
    def create_style(self):
        pass
```

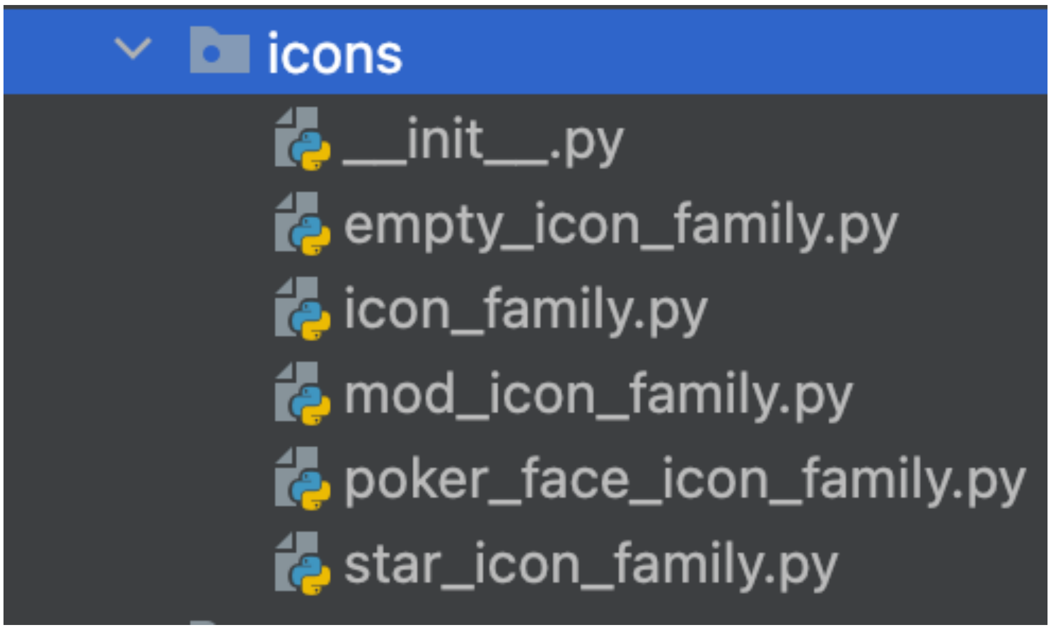
2.2 rectangle_style_factory.py

```
class RectangleStyleFactory(StyleFactory):
    def create_style(self):
        return RectangleStyle()
```

2.3 tree_style_factory.py

```
class TreeStyleFactory(StyleFactory):
    def create_style(self):
        return TreeStyle()
```

3. 图标类 用的是工厂模式



```
icons
    __init__.py
    empty_icon_family.py
    icon_family.py
    mod_icon_family.py
    poker_face_icon_family.py
    star_icon_family.py
```

3.1 icon_family.py

```
from abc import ABC, abstractmethod

class IconFamily(ABC):
    @abstractmethod
    def get_icon(self, value):
        pass
```

3.2 poker_face_icon_family

```python
from icons.icon_family import IconFamily

class PokerFaceIconFamily(IconFamily):
    def get_icon(self,):
        return "♤"
```

3... 剩下几个类同上，只是为了实现不同icon