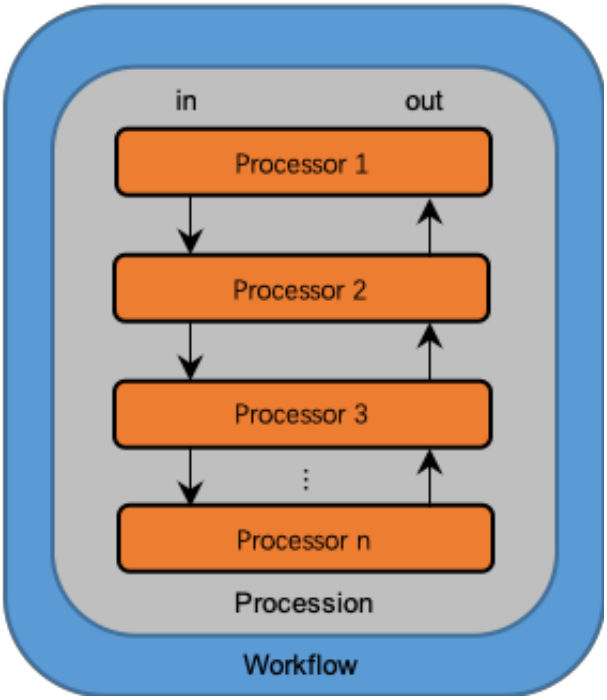
U型架构说明文档

1. U型架构图



如上图所示，U型架构中，通过Workflow完成特定任务，它包含一个Procession，并由该Procession管理一系列的Processor，每个Processor完成其中的某一部分。

U型架构是一个垂直的链式结构，数据流向如箭头所指，每个Processor有up和down两种数据流向(底部的Processor reflect除外，该Processor不对数据做任何处理，仅将输入数据反向，传到上层，目的是保证所有Processor的一致性，即确保每个Processor都有up和down两种数据流向，便于日后灵活配置)。

Procession的数据从顶端流入流出，数据进过每个Processor后，各个Processor自行决定数据流向，Processor之间可以构成环，即Procession包含的多个Processor可以构成一个状态机，实现复杂的业务逻辑。

该架构的优点是在实现各种业务时，可以把复杂的树形结构(以往通常使用大量的if else实现各种复杂的业务逻辑)通过简单的链式结构实现。仅需对Procession中的Processor进行组合配置，即可实现各种复杂功能。

2. 项目结构

```
u_shape_demo
├── requirements.txt
├── start_server.py                          程序入口
```

```
├── docker                                  docker打包相关文件
|   ├── Dockerfile
|   ├── build_docker_release.sh
|   ├── ci_docker.conf
|   ├── ci_docker_script.sh
|   ├── push_docker_release.sh
|   └── setup.py
├── project_demo                            源码目录
|   ├── __init__.py
|   ├── app                                 核心代码
|   |   ├── common                          公告代码包
|   |   |   ├── __init__.py
|   |   |   ├── driver.py                   添加系统目录及配置log等
|   |   |   ├── util.py                     工具类
|   |   |   └── ...
|   |   ├── __init__.py
|   |   ├── procession.py                   维护U型架构的运行逻辑
|   |   ├── server.py                       tornado服务端
|   |   ├── workflow.py                     持有procession对象,完成复杂任务
|   |   └── processors                      存储各个processor
|   |       ├── __init__.py
|   |       ├── processor_base.py           各个processor的基类
|   |       ├── processor_XXX.py            完成单一任务的processor
|   |       └── ...
|   ├── conf
|   |   ├── custom_conf                     自定义参数
|   |   |   ├── __init__.py
|   |   |   ├── demo.py                     demo builder 对应的override参数
|   |   |   └── ...
|   |   ├── procession_conf             u型框架结构及默认参数
|   |   |   ├── __init__.py
|   |   |   ├── demo.py                     demo builder 对应的override参数
|   |   |   └── ...
|   |   ├── __init__.py
|   |   ├── server_conf.py                  服务端参数等
|   |   ├── processor_conf_builder.py       processor配置项builder
|   |   └── log.conf                        配置log参数
|   └── log
|       ├── processor.log
|       ├── root.log
|       └── server.log
└── tests                                   单元测试
    ├── __init__.py
    ├── environment                         测试环境相关文件
    |   ├── Dockerfile
    |   ├── HTMLTestRunner.py
    |   ├── build_docker_dev.sh
    |   ├── push_docker_dev.sh
```

```
|   └── run_all_test_cases.py
├── test_data                          单元测试数据
|   └── ...
├── test_api.py                        api层面进行测试
├── test_wrokflow.py                   workflow层面进行测试
└── test_processor_XXX.py              processor层面进行测试
```

3. 使用说明

首先要说明几个概念：

1. 数据流：在U型框架中，数据通过数据流的形式，在各个processor间流动，实现各种功能。
2. processor：独立的处理单元，可以对数据流进行一种处理。
3. procession：对所有processor进行管理，维护整个数据流向。
4. workflow：一个具体任务的工作流，如合同比对。

每个Workflow实例为一个具体的任务，构造实例时，需要传入配置文件conf.py中的参数来配置Processor的组合。Workflow本身也支持重载配置参数以改变内部的Processor组合。

3.1 配置文件

u_shape_demo/u_shape_demo/procession_conf/demo.py示例如下：

```
from ..processor_conf_builder import ProcessorConfBuilder

builder = ProcessorConfBuilder('demo', ())

PROCESSION_CONF = [
    {
        'cmd': 'demo_a',
        'args': {}
    },
    builder.set_cmd('sub_package.demo_b').set_default({
        'enable': True,
    }),
]
```

该参数为列表，用于构造Workflow，决定了Processor组合的结构。列表中的每一个元素是一个字典，包含两个键，cmd和args：

- cmd: 键值数据类型为字符串，表示project/app/processors目录下的某个processor，如“diff”表示project/src/processors/processor_diff.py中的ProcessorUniform类。文件命名规则为processor_cmd.py，类命名规则如下：以Processor开头，然后将cmd以'_'分隔，且首字母大写，串联起来，代码看起来会更容易理解一点：

```
class_name = 'Processor' + ''.join([c.capitalize() for c in cmd.split('_')])
```

- args: 键值数据类型为字典，表示该Processor的使用参数

3.2 Procession

Procession仅完成一件任务，维护U型结构的数据流向，为了防止出现死循环，可以设置最大步数，默认为20。

3.3 Workflow

Workflow是完成一项具体工作的最小单元，包含了加载Processor，重载Processor和执行等操作。值得注意的是，一个Workflow实例和配置文件中的一个参数列表是一一对应的。

3.4 单元测试

使用PyUnit测试框架，测试代码在project/app/test目录下。

1. test_api.py 是对tornado服务这个层面上，程序整体的一个黑盒测试。
2. test_workflow.py 是在Workflow层面上进行的测试。
3. test*processor*XX.py 对某一个Processor进行测试。