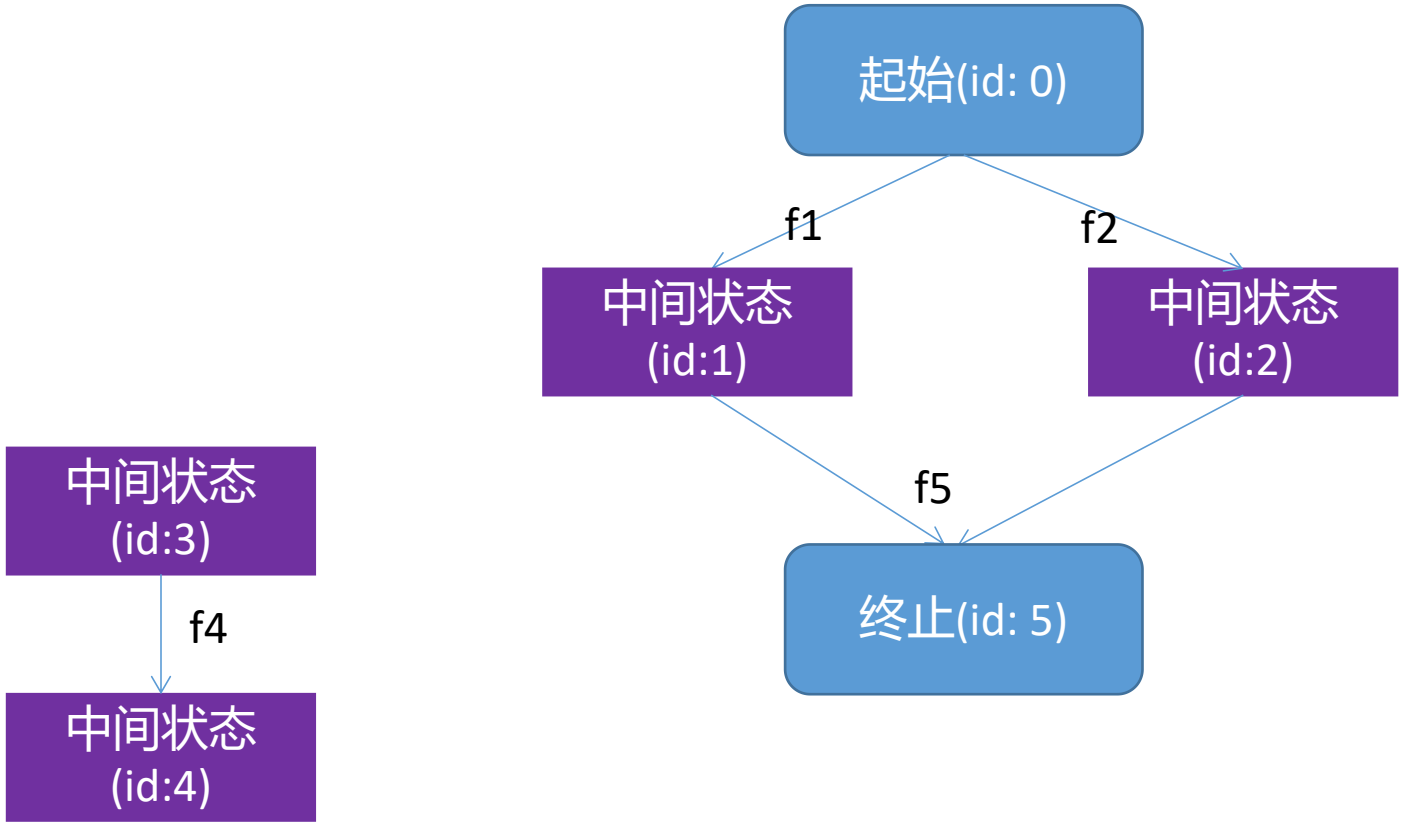
Pipeline设计

缪斯

C++部分设计框图



计算图抽象设计

我们称执行计算图(例如上页的图)的程序为计算引擎。计算引擎用一个C++类Engine来表示。

1. 初始时计算引擎利用拓扑排序算法检查图是否有环, 且除起始状态外是否每个状态都有唯一的函数和至少一个依赖状态。

2. 除了起始状态以外, 每个状态都有一个函数void f(std::vector<int>& pre, globalstate& gs)。函数f可以依赖一个或多个状态。例如上页f1, f2都依赖状态0, f3依赖状态1和2。pre就是依赖状态的id列表, 例如f1的参数里面pre={0}, f2的参数里面pre={0}, f3的参数里面pre={1, 2}。globalstate用于维护贯穿整个计算图的全局状态。

3. 优化:

(1)移除由初始状态不可达的点。例如上图中的状态3，4。事实上如果1中的检查通过, 不可能有初始状态不可达的状态(否则会有两个入度为0的点，与起始状态外其他状态入度至少为1矛盾)

(2)无依赖关系的函数(如: f1, f2)可以并行。

计算图抽象设计

4. 中间数据存储。不同状态的输入输出类型可能千变万化, 为了统一, 使用std::map<int, std::any> storage存储函数的输入输出。函数f在执行时, 根据pre里面的依赖的状态id, 从storage中取出any转化为对应的输入类型，运算结束后将结果转为std::any存入storage。注意，这里类型有一定耦合性, 例如上一个状态的输出类型由int修改为std::string, 那下一个函数在读入上一个函数的输出时, 要将代码里面的std::any_cast<int>转为std::any_cast<std::string>。类型是硬编码的, 但这样做硬编码的部分相当少。

5. 在Engine中增添std::map<std::string, std::funtion<void(std::vector<int>&, globalstate&)>> invoke, 用户可以通过string在invoke找到对应的函数。

6. 添加编译功能，将用户提供的json编译成计算图。例如第二页图的json可以为{“1”: {“pre”: [“0”], “fid”: “f1”}, “2”: {“pre”: [“0”], “fid”: “f2”}, “5”: {“pre”: [“1”, 2”], “fid”: “myf5”}}(省去不可达的3, 4状态, 没啥意义)。

通过5中的invoke, 可以将通过fid查找对应的函数并执行。

7. std::function也可以替换成仿函数(Functor, 即带有operator()的类), 差不多。Functor叙述起来比较麻烦，所以用std::function叙述。

8. AB平台是组合了{两个计算引擎(即class Engine)类的对象}和{一个裁判类(class Referee)的对象}的类(class AB)。AB平台可以自动生成两组json(即第6条的json), 并根据json分别编译运行两个计算图。

9. class Referee是业务强相关的, 我目前不知道怎么设计。但是我们可以搞一个基类和两个子类, 两个子类分别是SubjectiveReferee和ObjectiveReferee, 分别代表主观评价和客观评价, 但以我之前做图像处理的经验来看, 评价标准其实是一个公说公有理, 婆说婆有理的老大难问题。但我们可以先统一接口。