CFD-Уикенд 2020

Применение современных методов программирования в вычислительном комплексе NOISEtte

Краснов М.М., ИПМ им. М.В. Келдыша РАН

Москва, 1 декабря 2019 г.

Преимущества программных комплексов

- Готовая инфраструктура (сетки, OpenMP, MPI, GPU, графика);
- Готовые подсистемы, которые можно использовать непосредственно, а можно брать как образцы для создания своих объектов (солверы, начальные и граничные условия, источники, точные решения и пр.).

Проблемы

- Отсутствие объектно-ориентированного подхода;
- Сложность добавления нового функционала, нужно делать изменения в нескольких местах основного кода;
- При получении новой версии повторное внесение изменений;
- Сложность инициализации, порядок инициализации подсистем жёстко задан.

Цели

- Сделать код более объектно-ориентированным, чтобы данные были инкапсулированы в объекте, уйти от глобальных переменных.
- Создать расширяемый отчуждаемый код, сторонние разработчики должны иметь возможность добавлять новый функционал без изменения основного кода;

Цели (продолжение)

- Упростить инициализацию, не инициализировать ненужное, нужную инициализацию сделать управляемой, чтобы объекты могли сами определить порядок инициализации.

```
 MeanFields(); // Background fields
InitPuls(); // Initial data (pulsations of full)
ESInit(); // Init exact solution
InitSpongeLayerParams(); // Sponge layer
InitRKCoeff(); // Explicit time integration
InitImplParams(); // Implicit time integration
InitFlowState(); // Flow state report
AVGRecord.Init(VAR_CHECK_ON);   // Time averaging
RecoveryRec.InitParams(&AVGRecord); // Init recovery
VisuaInit(); // Output: visualization
HistoryInitParams(); // Output: time evolution
CheckParams(); // Param check before reading mesh

InitOMP_decompTopo();
AllocCompNcArrays(); // Allocation of main computational arrays
InitScheiben(); // Rotate frame initialization
BuildCVSegments(); // Building geometry
CheckMeshConformity(); // Comformity check
HElemMin(); // Find minimal height of mesh element
InitBC(); // Init boundary conditions
InitSemiStructure(); // Init of structure in Z-direction
InitTurb(); // Init turbulence
InitSour(); // Init sources
InitMovingMesh(); // Init of moving (deforming) mesh
InitLowOrderSwitcher();
BuildHOconstructs();
InitLOPolynomials(BaseGradients); // Element gradients (Stiefel)
```

```
enum tEStype{
  ES_NO                =0, // no exact solution
  ES_Zero              =4, // zero exact solution
  ES_Gaussian          =1, // Acoustic Gaussian impulse
  ES_Gaussian2D        =2, // Acoustic Gaussian impulse
…
};
EXTERNAL tEStype KeyExactSol DEFAULT(ES_NO);
#define ExactSolutionNames " \
 NO                  0, \
Gaussian             1, \
Gaussian2D           2, \
…";
switch(KeyExactSol){…}
if(KeyExactSol ==…){}
```

Модули (пул объектов)

- Регистрация объектов, имени (строке) ставится в соответствие функция, создающая объект (фабрика);
- Создание объекта по имени. Ищется и вызывается функция, создающая объект с данным именем;
- Пулов может быть несколько для разных типов объектов.

Модули (пул объектов)

```
template<class tObjType>
struct object_pool{
  typedef tObjType*(*factory_type)();
  typedef map<string, factory_type> registry_type;
  tObjType* GetObject(const char *name){
    typename registry_type::iterator it = registry.find(name);
    return it != registry.end() ? it->second() : 0;
  }
  void RegisterFactory(const char *name,
    factory_type factory){ registry[name] = factory; }
  template<typename T>
  static tObjType* factory() { return new T(); }
private: registry_type registry;
};
```

Реализация

- В качестве параметра шаблона класса object_pool передаётся базовый класс всех объектов данного типа;
- В этом классе задаётся общий функционал данного типа объектов (виртуальные функции);
- При проектировании важно тщательно продумать этот функционал.

Текущее состояние

В настоящее время в NOISEtte на модули переведены:

- Начальные условия;
- Источники;
- Вязкие потоки.

Потенциальные кандидаты:

- Точные решения;
- Граничные условия;
- Решатели СЛАУ (solvers).

Автоматическая инициализация (runtime_class)

- Цель – полуавтоматическое создание и инициализация объектов.
- Явно создаются только основные объекты. Вспомогательные объекты создаются автоматически.
- Инициализация происходит автоматически и в правильном порядке, задаваемом объектами.

В чём идея

- Каждому классу (конечному или промежуточному) ставится в соответствие статический объект времени исполнения (класса *runtime_class*);
- В этом объекте задаётся уникальное имя (имя класса), функция создания объекта (для конечных классов) и базовый класс (*runtime_class*);
- Все объекты класса *runtime_class* выстраиваются в единую цепочку.

Реализация

```
struct runtime_class {
  runtime_class(const char *class_name,
    object_base* (*pf_create_object)() = 0,
    const runtime_class *pbase_class = 0) :
    class_name(class_name),
    pf_create_object(pf_create_object),
    pbase_class(pbase_class), pnext(phead)
  { phead = this; }

const char *const class_name;
object_base* (*const pf_create_object)();
const runtime_class*const pbase_class,*const pnext;

static const runtime_class *phead;
};
```

Идея (продолжение)

- Все классы, для которых определён *runtime_class*, должны быть пронаследованы от единого базового класса *object_base*;
- Имеется дополнительный класс *object_container*, который хранит все созданные им объекты;
- Создание всех объектов и их инициализацию осуществляет этот контейнер (метод *create_all*). У объекта, созданного контейнером, имеется указатель на него (в переменной *pcontainer*).

- После создания объекта у него можно спросить, какие дополнительные объекты ему нужны. Эти объекты также создаются;
- Инициализация объектов проводится после создания ВСЕХ объектов;
- При инициализации объектов у каждого объекта запрашивается, какие объекты должны быть проинициализированы до него. Соответственно, они инициализируются раньше.

- Имеется три стадии инициализации: *pre_init*, *init* и *post_init*. На предварительной стадии (*pre_init*) объект может проинициализировать внутренние ссылки на другие объекты, взяв их из контейнера.
- У контейнера есть методы обхода объектов (*do_for_all_objects*, *do_for_all_objects_of_class*) и поиска объектов по имени (find_object_by_name) или по классу (*find_objects_by_class*, *find_object_by_class*).

Класс *object_base*

```
class object_base {
protected:
   object_base() : pcontainer(0){}
public:
  virtual ~object_base(){}
  virtual const runtime_class* get_runtime_class() const;
  virtual std::string required_objects() const
  { return std::string(); }
  bool is_kind_of(const runtime_class* pclass) const;
  static const runtime_class class_object_base;
 friend class object_container;
private: const object_container* pcontainer;
};
```

Будущее языка C++

- Цель – более безопасное программирование;
- В C++23 отмена сырых (raw) указателей и операторов new и delete, в C++20 они объявлены как устаревшие (deprecated).

Спасибо за внимание