

SimEngine 使用说明

一：编程基本方法

概念：

- 一：接口(Interface)，用于对外提供调用的类以及方法，最终封装成 sdk 包
- 二：对象(Object)，用于描述应用场景中的具体实例目标以及行为个体。
- 三：组件(Component)，主要用于描述对象的各种能力，一个对象可包含多个不同组件。
- 四：系统(System)，用于支撑对象与组件实现的底层服务，包括渲染引擎与物理引擎。

引擎的编程模型采用了组件对象系统，对象通过实现或者复用多个组件来实现功能组合，组件通过调用各种子系统(渲染/物理引擎)来实现具体的功能以及模块集成。

基础接口

基础接口指各类应用开发都通用的基础编程接口,用于支持一些通用的程序框架和流程

- 1: IReference, 用于支持接口与接口之间动态转换与扩展，以及接口引用计数管理
- 2: Factory, 用于支持对象的统一创建与销毁。
- 3: IDispatcher, 调度器，用于将执行过程按需求进行异步调度或者跨模块调度用于将操作命令与执行过程解耦，包括定时调度器，过程处理调度器，命令调度器。

接口使用

一：创建对象

假设引擎中支持对象 ObjectA, 且对外提供了接口 InterfaceA, 可通过如下方式创建对象：

InterfaceA::Create(); 该调用返回 ObjectA 的接口 InterfaceA.

注意：用户不能直接访问对象 ObjectA，只能通过对象的接口来访问对象，因为对象的实现可能会频繁更替，但是接口一般保持不变

二：接口管理

A: 接口创建, 一个对象可能同时拥有多个接口, 也即是一个对象同时支持多种功能, 即一个对象包含了一个接口集(不重复的接口组成的集合, 一般情况下, 一个对象的接口集中只有一个接口)。

B: 接口转换, 如果一个对象支持多个接口, 可以通过 `IReference->cast<>()` 来进行接口转换, 接口转换的行为与 c++ 的 `dynamic_cast` 类似, 但是 `dynamic_cast` 转换仅仅支持通过继承实现的对象和类, 且编译期就固定下来了, `IReference->cast<>()` 可以支持运行期动态组合的对象接口。且继承形式不限于语言继承, 也支持采用组合实现的接口集合。

C: 释放接口, 若使用者不需要继续持有某接口时, 则可以调用 接口的 `Release` 方法释放接口。对象工厂内部根据接口的使用情况负责管理对象的销毁, 也可以使用引擎提供的包装指针, 用户无需调用 `Release` 方法, 包装器自动处理接口的释放。

引擎接口介绍

ISimEngine

一：引擎接口, 用户开发应用程序需要首先创建引擎对象, 并获得 `ISimEngine` 接口, 通过 `ISimEngine` 接口, 用户可以直接或间接的创建与获得本引擎各种功能的所有对象与接口。

二：本文档只说明重点和关键接口, 详细的接口清单请参考:

<http://192.168.6.45:8080/doxygen/html/classes.html>

程序主流程：

一：创建引擎(C++)

```
_sim_engine = DF::ISimEngine::Create()->Init();// 创建引擎并初始化
```

二：创建场景

```
_scene = _sim_engine->CreateScene();创建场景
```

```
_scene->SetIBL(“光照文件”);
```

二：主循环

直接调用引擎的 Run 方法可启动引擎的默认主循环，用户也可以自定义主循环。

示例：

1：引擎默认主循环：_sim_engine->Run();

2：自定义主循环：

```
while(!exit)
{
    _sim_engine->Refresh(dt); // 刷新仿真引擎
    _sim_engine->Step(dt); // 物理仿真
    _sim_engine->Render(); // 渲染画面
}
```

可分别设置时间条件来控制_sim_engine->Render();和_sim_engine->Step(dt)的调用，实现渲染与仿真的帧率限制。

详细的主循环示例代码如下：

```
double      dt = 0; // 秒
double      render_escaped = 0;
double      render_tpf = _sim_engine->GetRenderFps()>0?
1.0/_sim_engine->GetRenderFps() : (1.0/60);

double      physic_tpf = _sim_engine->GetPhysicFps()>0?
1.0/_sim_engine->GetPhysicFps():(1.0/60);
double      physic_escaped = 0;
while( ! _sim_engine->GetRender()->IsClosed() )
{
    std::chrono::time_point<std::chrono::steady_clock> t_start =
std::chrono::high_resolution_clock::now();

    std::this_thread::sleep_for(std::chrono::milliseconds(1));
    _sim_engine->Refresh(dt); //此行必须调用
    if(physic_escaped>=physic_tpf)
```

```

    {
        _sim_engine->Step(physic_escaped); // 此行可自由决定何时调用
        physic_escaped = 0;
    }
    if(render_escaped>=render_tpf)
    {
        _sim_engine->Render(); // 此行可自由决定何时调用
        render_escaped = 0;
    }

    std::chrono::time_point<std::chrono::steady_clock> t_end =
std::chrono::high_resolution_clock::now();

    dt =
std::chrono::duration_cast<std::chrono::microseconds>(t_end-
t_start).count() /1000000.0; //转换成秒

    render_escaped += dt;
    physic_escaped += dt;

}

```

三：退出

应用程序退出的时候，需要关闭引擎，释放资源：

```

_sim_engine->UnInit();

```

二：引擎功能

功能概念：

一：场景(Scene)：若要对真实世界进行仿真，需要建立一个场景，场景是各种类型物体对象的集合，并负责维护各种对象的关系结构。

二：节点(Node)：是指具有空间属性的对象(包装位置与姿态属性)，支持空间属性设置，也可以设置对象之间的空间关系，凡是占据一定空间位置和体积的对象，都可以作为节点对象。

三：物体(Object)：物体对象继承自节点对象，物体与物体之间可以通过添加父子关系来实现挂接与组合关系。

四：角色(Actor)：独立的物体对象，可施加力/速度等物理操作。

五：链接对象(Link)：可以通过在对象之间施加关节约束的一类对象。

六：关节对象(Articulation)：拥有多个链接对象的链接集合体。比如机器人，机械臂，他们内部的每个活动零部件都可以看作是一个链接。

七：摄像机对象(Camera)：用于在场景中设置视点、视角、视野三个参数来决定渲染哪些场景内容。对于视野范围外的物体，将不参与渲染。

八：渲染模型 RenderBody, 包含 Mesh/Material

九：物理模型 PhysicBody.

场景接口 IScene

一：场景创建：

场景可以调用 ISimEngine 创建，ISimEngine::CreateScene。

当创建一个场景后，用户可以调用场景接口中的各种对象创建功能来创建对象。

创建一个角色：IScene::CreateActor

创建一个关节对象：IScene::CreateArticulation

创建物理平面：IScene::CreatePhysicPlane,支持创建一个具有物理遮挡功能的平面，比如创建地面，墙面等等。

二：外部内容载入

支持从文件导入场景内容：IScene::ImportFile，文件格式支持 urdf,glTF,fbx,obj,ply 等等。

三：对象检索

按名字查找一个对象 IScene::FindObject

根据射线的发射方向选择一个对象 IScene::RayCast()

遍历场景中的所有对象 IScene::Accept，此调用需要传入一个访问器，场景通过回调列出遍历的每个对象。

四：支持 IBL 光照。

IScene::SetIBL(),调用此接口，并指定一个光照文件目录，目录下包含一个辐照度描述文件 sh.txt，一个后缀名为_ibl.ktx 环境贴图，一个后缀名为_skybox.ktx 天空盒贴图。该目录的内容由工具 cmgen 处理.hdr 文件获得，相关素材可从 <https://polyhaven.com/hdris> 获取

五：仿真更新

默认情况下，程序每帧都会刷新场景的内容，也会刷新物理引擎的仿真状态。用户可以调用 IScene::ScaleTime,来调节物理引擎仿真的速度。

节点接口 INode

- 一：节点创建，通常情况下，用户无需直接创建节点，当调用场景的创建接口创建 Object、Articulation, Actor 等等空间功能对象的时候，会自动创建并拥有节点接口。
若用户想纯粹的创建一个独立的节点对象, 可调用 `INode::Create(scene)`, 并将场景传递给第一个参数。如果想创建一个独立的空间参考点，则可以这样使用，前提时你需要准确的理解 Node 的存在意义，以及自己的需求。
- 二：节点支持平移，缩放，旋转三种空间操作。当节点存在相对参考节点的时，节点的所有空间操作都是相对的，即相对于参考目标节点。
可以通过调用 `INode::World` 接口获取到自己在世界中的变换矩阵。
- 三：节点扩展资源，可以为节点分配渲染模型以及物理模型。通过调用 `INode::Init()` 创建。
也可以单独调用 `INode::CreatePhysicBody` 创建一个物理模型。当操作节点的空间变换时，节点下的所有模型均都产生空间变换效果。
- 四：节点支持轴对齐包围盒，可分别调用 `INode::LocalAABB` 以及 `INode::WorldAABB` 来获取物体局部空间的包围盒以及世界空间的包围盒

物体接口 IObject

- 一：物体的可以通过 `IScene` 中的各种具体对象的创建方法创建。
若用户想纯粹的创建一个独立的物体对象，可以调用 `IObject::Create(scene)`。并将场景指针传递给第一个参数。
- 二：物体继承自节点，拥有节点的全部功能。
物体包含父子关系，一个物体可以由多个物体组装而成，或者一个子物体可以挂载到另一个父物体对象上。通过调用 `IObject::AttatchParent()` 和 `IObject::DetachParent` 来实现父子物体之间的挂载与卸载。子物体可以继续挂载子物体，如此可以形成一棵物体层次树。
- 三：子物体的访问，通过调用 `IObject::Children` 可以获取该物体的子物体。

渲染模型接口 IRenderBody+ IMesh

一：IRenderBody 封装了各种渲染模型的访问操作。一个 RenderBody 包含一个或多个 3D 渲染模型 Mesh。通过 RenderBody 可以获取到其包含的 Mesh。RenderBody 负责加载卸载 Mesh，以及处理 Mesh 的渲染构建。

二：IMesh 封装了模型数据的深度操作，通过 IMesh 接口，用户可以访问模型数据的顶点数据，面片数据，支持给 IMesh 设置材质。一个 Mesh 对应可设置一个材质。参考相关接口：

```
IMesh::AttributeBuffer(attribID) // 属性缓冲
```

```
IMesh::BufferDesc() // 缓冲描述结构
```

```
IMesh::Indices() // 图元绘制索引缓冲
```

```
IMesh::PolygonList() // 多边形列表
```

IMesh::PolygonNormal() // 面法线列表，注意 Mesh 不维护面法线列表,每次调用此方法都会重新计算面法线，因此不推荐用户频繁调用。

```
IMesh::SetMaterial() // 设置材质
```

三：IRenderBody 也提供了设置材质的接口 SetMaterial。若 RenderBody 原来是由多个 Mesh 组成，调用 IRenderBody::SetMaterial。此调用会将所有 Mesh 的材质统一设置为同一个材质。

材质接口 IMaterialMgr+IMaterial+IMaterialInst

一：IMaterialMgr：材质管理器,通过 IMaterialMgr 可以从文件中加载材质

相关接口：IMaterialMgr::loadMaterial(材质目录)，返回材质模板 IMaterial
路径目录最后一个目录名作为材质包名，材质包目录下的文件应该是以材质包名开头，以材质模板(调用 IMaterial::materialTemplate()获得)的各种贴图类型作为后缀的贴图文件,名字与贴图后缀名直接用下划线分开

* 举例：假设材质所在的目录名叫 Metal022_2K,则 Metal022_2K 目录下应该包含 Metal022_2K_Color.jpg,Metal022_2K_Metallic.jpg,Metal022_2K_Normal.jpg,Metal022_2K_Roughness.jpg 等等作为贴图文件名.

材质管理器支持若干默认材质：

IMaterialMgr::loadMaterial,标准 PBR 材质，因此要求用户提供的材质目录是标准的 PBR 材质贴图文件。

IMaterialMgr::colorRGB，默认的 RGB 纯色材质

IMaterialMgr::colorRGBA，默认的 RGBA 纯色带透明的材质

IMaterialMgr::emissiveRGBA，默认的自发光纯色材质，此类材质在没有光照的场景中也能显示，而其他非自发光材质必须在有光照的场景中才能显示。

创建材质

IMaterialMgr::createFromScript,从材质脚本文本创建材质

IMaterialMgr::createFromCache，从缓存创建材质，缓存数据是从材质脚本通过材质工具 matc 编译获得，缓存材质是为了加快引擎的解析与渲染而设计的一种二进制格式材质。它可被 GPU 直接读取执行。而将材质脚本编译成缓存需要耗费比较长的时间，此过程一般离线编译处理。

IMaterialMgr::createFromConfig，根据指定的若干参数选项创建材质，可以指定一个材质选项配置，然后创建一个自定义材质。

二：IMaterial：从文件加载材质后得到一个 IMaterial 接口，该接口是与该材质目录关联的一个材质模板。IMaterialMgr 内部维护了一个文件目录与材质模板的一个映射表。

三：IMaterialInst: 材质实例

通过材质模板，可以创建多个材质示例，如下所示：

```
IMaterialInst* instM = IMaterial::createInst(name),
```

当用户需要从文件中读取一套材质，但是加载后需要定制化的去调整不同的参数配置，即可以如此创建多个材质实例，每个实例配置不同的参数。

材质模板内维护了一个名字与材质实例的映射表，用户可通过 IMaterial::Inst(name)按名 i 在获取材质，当不传递名字的时候，将获取到该材质模板的一个默认材质实例。

四：金属-粗糙度材质，前文 IMaterialMgr::loadMaterial 所加载的即是此类材质。

此类材质具备以下贴图：

baseColorMap: 基础纹理贴图

aoMap: 环境光遮挡贴图

roughnessMap 粗糙度贴图

metallicMap: 金属度贴图

normalMap: 法线贴图

其他参数：

baseColor: 基础颜色。

metallic: 金属度系数

emissive 自发光颜色

roughness 粗糙度系数

reflectance: 反射系数

sheenColor: 光泽颜色

五：布料材质，待补充

六：次表面材质，待补充

七：镜面-光泽工作流，与金属-粗糙度殊途同归的一类材质，待补充。

物理模型接口 IPhysicBody

一：IPhysicBody 封装了各种物理模型的访问操作。一个 PhysicBody 包含一个或多个物理形状。物理形状决定了物体之间的碰撞边界。

二：IPhysicBody 本身有刚体、软体、流体、气体等等类型。引擎目前的版本只支持刚体。

IPhysicBody 本节若没有明确说明，所有的物体模型默认指的是刚体。

三：IPhysicBody 支持创建静态和动态的物体，动态物体又分为动力学的和运动学的

静态物体(ACTOR_STATIC)，在整个场景仿真期间不会发生空间属性的变化。

运动学物体(ACTOR_KINEMATIC)，在仿真期间可被用户之间操作空间属性，用户可操作运动学物体的 INode::SetPosition 方法来控制物体的运动，并且与其他物体发生碰撞(可以挤压弹开其他物体)。如果两个运动学物体发生碰撞，则产生模型的叠加和穿越且不会反弹。若用户期望运动学物体之间不发生穿越，可以通过处理碰撞回调禁止物体继续前进来实现。

动力学物体(ACTOR_DYNAMIC)。在仿真期间只能通过施加初速度和力的物体,即遵守牛顿定律的物体。可设置质量，速度，加速度，动量，冲量，惯性，摩擦系数，反弹系数等等物理属性，这些属性均影响着物体的运动状态。用户无法通过 INode::SetPosition 来设置物体的位置。

当动力学物体与运动学物体发生碰撞时，相互都会存在反弹效果。

当动力学物体与运动学物体发生碰撞时，可以认为运动学物体拥有无限大质量，不受动力学物体影响，且动力学物体会被不可抗力的弹开。

通过 IPhysicBody::InitPhysicMesh(),以及 INode::CreatePhysicBody 可以创建以上三种类型的物体。

四：碰撞检测，对于高速运动的动力学物体之间的碰撞检测，需要启用连续碰撞检测功能，可调用 IPhysicBody::SetPysicFlag(ENABLE_CCD)，否则容易发生穿越而无法检测到碰撞。运动学物体不支持此功能。

角色接口 IActor

一：Actor 可以通过 IScene::CreateActor 来创建。

二：Actor 默认具备了物理功能，可以通过 IActor::setLinearVelocity

以及 IActor::setAngularVelocity 来设置线速度与角速度。

如果 Actor 是动力学物体，则调用 IActor::SetPosition 无法对物体产生影响(可以这样理解，即使当前成功设置了物体的位置属性，但是下一帧，物体的位置属性立即被物理引擎按照物体的牛顿运动规律重置了)

如果 Actor 是运动学物体，则无法 setLinearVelocity 与 setAngularVelocity 方法无效。

IArticulationV0

Articulation 先后有两个版本，早期版本称为 IArticulationV0，在该版本中，用户可自定义创建与组装关节。并且支持运动学与动力学混合控制。

机械臂仿真的动力学关节控制问题：

机械臂仿真实际需求是当给某一关节 Link 下达一个运动命令后，命令 Link 运动到目标角度。对于空操作，没有交互的情况，动力学机械臂往往没什么问题，但是如果存在碰撞和交互，比如机械臂碰到一个比较重的物体或者去抓一个比较重的物体，此时可能会对机械臂产生反推作用，这种反推的作用力不止作用域机械臂的夹爪末端，而且会因为从末端传递到机械臂的上臂，从而导致机械臂的整个姿态发生了变化。因此，为了保证机械臂的整体姿态不受其他外力的影响，应该把机械臂的 Link 设置为运动学的。

机械臂的夹爪控制问题，如果把夹爪设置为运动学的，当夹爪夹紧物体时，物体将被挤压，且由于运动学夹爪无视障碍物继续运动，将产生穿模现象。物体也可能因为压力太大被夹飞。因此，夹爪最好设置为动力学的，采用力矩控制。如果设置为运动学的，必须在夹住物体两侧时，通过接触回调指令，禁止夹爪继续夹紧。

IArticulation

一：Articulation 是多关节类物体，如机器人，机械臂等等。

Articulation 拥有多个 Link，Link 与 Link 之间具有物理关节约束。关节约束限制了每个 Link 的运动自由度(每个 Link 最多有 3 个方向移动的自由度以及 3 方向个旋转的自由度)。

在 Articulation 内部，所有的 Link 组成了一个树形结构。除了根 Link 之外，所有子 Link 拥有一个关节，该关节指向父 Link。根 Link 没有父 Link,所以没有关节。

二：IArticulation 提供了以下方法来控制关节和 Link 的运动：

IArticulation::setRootGlobalPose 设置根 Link 的位置

IArticulation::getDofs 获得 Articulation 总的自由度数量。

IArticulation::getJointsActivated 获得所有活动的关节

IArticulation::setJointPositions 设置所有活动关节的位置(或者旋转角度)，参数长度必须与

IArticulation::getDofs 相等

IArticulation::setJointLinearVelotities 设置所有活动关节的速度，参数长度必须与 getDofs 相等

IArticulation::setJointEfforts 设置所有活动关节的力，参数长度必须与 getDofs 相等

三：IArticulation 支持将另一个 IArticulation 挂接到自己的某个 Link 部位，比如机械臂更换抓手。

提供如下方法。

IArticulation::attachChild 装载一个 Articulation

IArticulation::detachChild 卸载一个 Articulation

四：将两个 Articulation 强制安装到一起时，以及关节的父子 Link 之间添加约束时， 一般都会产生模型重叠，对于动力学关节，此时一般会发生碰撞导致乱晃不止，此时应该在相邻两个连接的对象之间过滤碰撞。具体参考以下方法

IArticulation::filterCollision

ILink::filterCollision

对于通过文件加载的关节物体，一般在加载过程中已经自动设置好了碰撞过滤，若用户期望手动组装多关节物体，则需要处理好碰撞过滤问题。

