本科毕业论文（设计）

题目：基于 osgEarth 的三维数字地球通用接口设计与实现

姓　　名：罗浩　　学号：20111000604

所在院系：信工学院　　专业：软件工程

指导老师：张剑波　　职称：副教授

评 阅 人：　　职称：

2015 年 06 月

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名：　　年　月　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保障、使用学位论文的规定，同意学校保留并向有关学位论文管理部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权省级优秀学士学位论文评选机构将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

1．保密 □，在_____年解密后适用本授权书。

2．不保密 □。

（请在以上相应方框内打“√”）

作者签名：　　年　月　日

导师签名：　　年　月　日

基于 osgEarth 的三维数字地球通用接口设计与实现

本科生：罗浩　　指导教师：张剑波

摘　要

空间信息技术研究和应用越来越注重多角度、多方式。传统二维的方式无法完全描述所有空间信息，三维技术的应用丰富扩展了空间信息技术领域。三维空间信息技术越来越受到追捧，应用领域仍在不断扩大，应用层面不断加深，拥有广阔的技术发展空间和前景。

本论文分析和研究了国内外三维数字地球软件开发包现状，学习研究基于 OpenSceneGraph、osgEarth 三维数字地球的技术应用。基于 osgEarth 设计开发出一套跨语言、跨平台、接口通用的三维数字地球软件开发包 HsEarth，提出一套实用、高效的三维数字地球应用开发解决方案。

技术学习与研究的内容：

1) 三维场景渲染引擎 OpenSceneGraph 和三维地图引擎 osgEarth
2) 跨平台多线程库 OpenThreads
3) COM 与 ATL 技术
4) 跨浏览器插件构建工具 FireBreath
5) 跨平台项目构建工具 CMake
6) Linux 和 QT 编程知识
7) Android NDK 应用程序开发知识

项目开发的内容：

1) 收集项目需求，设计项目总体架构，编写项目开发计划和项目开发文档。
2) 修改 OpenSceneGraph 和 osgEarth 的源码。
3) 设计开发核心基础库 HS_Libs、HS_OSG、HS_Threads。
4) 设计开发封装 osgEarth 和基础库的功能的 HS_Earth_Core。
5) 设计开发实现符合业务逻辑的通用接口的 HS_Earth。
6) 设计开发 COM 组件 HS_Earth_COM，以及开发使用 HS_Earth_COM 组件的示例。
7) 设计开发跨浏览器插件 HS_Earth_Web，以及开发使用 HS_Earth_Web 跨浏览器插件的示例。
8) 开发 Linux 和 Android 平台下的演示示例。

当前版本的三维数字地球软件开发包 HsEarth 完成了总体框架的设计和开发，核心库和组件具备了基本功能和接口。经过后续开发和发展，该三维数字地球软件开发包将会免费发布出来，并提供源码开放。

关键词：三维数字地球，OpenSceneGraph，osgEarth

The Designment and Implementation of 3D Digital Earth's Unified Interface

Bachelor Candidate: Hao Luo Supervisor: Jianbo Zhang

ABSTRACT

The study and application of spatial information technology are increasingly foused on multi-dimension and multi-ways. Using traditional 2D technology can not describe all kinds of spatial information, and the field of Spatial information Technology is extanding because of the application of 3D technology. 3D Spatial Information Technology, which has an vast technical development space and prospect, is being more and more popular, and the application field keeps extending and deepening.

This graduation thesis studied and analyzed the status of 3D digital earth software development kit at home and abroad, and studied the application of OpenSceneGraph and osgEarth. On the top of osgEarth, an cross-language, cross-platform and one-size-fits-all 3D digital earth SDK HsEarth was designed and developed, and an development solution of pratical and efficient 3D digital earth application was came up.

Here are the technologies need to be studied :

1) 3D scene rendering engine OpenSceneGraph and 3D mapping SDK osgEarth
2) Cross-platform multithread library OpenThreads
3) COM and ATL technologies
4) FireBreath, the tool of building cross-browser plugin
5) CMake, the tool of building cross-platform projects.
6) Programming knowledge of Linux and QT
7) Knowledge of developing Android NDK application

Here is the content of the project development :

1) Collecting project requirements, designing project's overall structure, and writing project development plan and documents.
2) Modifying the source code of OpenSceneGraph and osgEarth.
3) Designing and developing core libraries HS_Libs, HS_OSG and HS_Threads.
4) Designing and developing library HS_Earth_Core that encapsulates osgEarth and core libraries.
5) Designing and developing library HS_Earth that implements unified interface.
6) Designing and developing an COM component HS_Earth_COM, and developing typical examples.

7) Designing and developing a cross-browser plugin HS_Earth_Web, and developing typical examples.
8) Developing typical examples for Linux and android.

The current version of 3D digital earth SDK HsEarth has accomplished the designment and development of overall architecture, and core libraries and software components have the basic function and interface.After follow-up development, this 3D digital earth SDK will be released for free, and will also open the source.

Keywords: 3D digital earth; OpenSceneGraph; osgEarth

目　录

第一章 绪论

§1.1 选题来源

空间信息技术研究和应用越来越注重多角度、多方式。传统二维的方式无法完全描述所有空间信息，三维技术的应用丰富扩展了空间信息技术领域。

从上世纪 80 年代开始，三维空间信息技术研究和应用的热度从未停息。发展至今，三维空间信息技术已经发展成熟，成就突出，得到了广泛的应用。我国非常注重空间信息技术的研究和应用，虽然起步比国外晚，但经过多年发展，特别是近几年，国内三维空间技术研究和应用正逐渐追赶到队伍前列来。

三维空间信息技术越来越受到追捧，应用领域仍在不断扩大，应用层面不断加深。三维空间技术在航空航天、地图制作、地址勘探、资源开发、抢险救灾、土地资源管理、定位导航等领域被广泛应用。谷歌地球 Google Earth、World Wind 这些三维数字地球软件真是令人印象深刻，用户可以查看全球每一个角落的景观，了解每天的气象数据。三维空间技术应用已经普及到人们的学习、工作和生活的各个方面。

在信息时代，三维空间信息技术的普及和后续发展同样是一个重要的话题，只有不断进行技术更新才能跟上时代的步伐，满足研究和应用的需要。三维空间信息技术已经走向多领域、多应用层，三维空间信息技术还有广阔技术发展空间和前景。

§1.2 选题目的与意义

1.2.1 目的

选题是为了学习和研究三维数字地球应用开发技术和不同平台组件开发技术，基于 osgEarth 三维数字地球的技术应用，设计开发出一套跨语言、跨平台、接口通用的三维数字地球软件开发包，提出一套实用、高效的三维数字地球应用开发解决方案，还有发布和推广该免费的三维数字地球软件开发包。

1.2.2 意义

选题的学习研究成果为基于 osgEarth 的三维数字地球的研究和应用提供经验；设计开发的免费的软件开发包，降低了学习和开发的成本，利于相关技术的发展、应用和推广；提供的开发解决方案不仅是面向个人和小规模项目应用，而且还是面向企业和大规模项目应用，可以开发出不同需求的三维空间信息应用。

§1.3 国内外三维数字地球软件开发包现状

1.3.1 简述

三维空间信息技术日趋成熟，国内外先后推出了多款三维数字地球软件开发包，这些软件开发包在产品定位、产品架构、产品功能等方面都有独特之处。在三维空间信息技术发展过程中，国外的 Google Earth、Skyline 等三维空间信息技术产品最先出现，国内经过近十年的研究和发展，也先后推出了例如 EV-Globe、GeoGlobe 等三维地理信息技术产品。

Skyline Globe 是美国 Skyline 公司推出的行业领先的三维数字地球平台软件。Skyline Globe 生产线齐全，具有强大的空间信息分析、管理、展示能力，支持多种数据源，具备海量数据集成、大型场景构建和面向网络运营的能力。Skyline Globe Enterprise Solutions 是 Skyline 的企业级解决方案，具备 Skyline 完整的工具集和开发流程，客户可基于此解决方案构建出单机运行或者网络运营的三维地理信息系统。Skyline Globe 产品线主要包括场景构建平台 TerraBuilder、场景综合展示平台 TerraExplorer 和地形数据服务平台 TerraGate 三种，Skyline Globe 软件开发包支持多种语言、多种平台的三维地理场景应用的构建。Skyline Globe 产品应用广泛，目前在国内被大量用于构建三维地理场景。

EV-Globe 是北京国遥新天地研发的拥有自主知识产权的三维空间信息平台[1]。EV-Globe 面向海量数据、多源数据一体化管理的管理平台，支持二三维数据的一体化管理、分析、操作、显示， 其服务器产品支持多种服务发布、基于 Hadoop 的分布式部署等。EV-Globe 采用全平台统一的开发框架，软件开发包支持多语言、多平台的三维空间信息软件的构建。目前在国内三维空间信息软件产品中，EV-Globe 处于领先地位。

GeoGlobe 是武汉武大吉奥公司推出的地理信息平台软件。GeoGlobe 支持二三维地理数据操作、管理、存储、显示，支持多平台应用构建，GIS 应用集成，地理服务发布等。GeoGlobe 软件产品包括二三维 GIS 软件产品、地理信息服务器、移动设备应用 。GeoGlobe 软件开发包支持多平台、多功能 GIS 应用构建。

Google Earth 是 Google 公司推出的三维数字地球浏览器，是全球用户最多的三维数字地球产品。Google Earth 中整合了大量商业卫星影像、航拍图像数据和模拟三维图像，用户可以使用其客户端从不同角度浏览全球的景象、搜索景点资源、查看交通信息等等。Google Earth 提供多个版本客户端，提供不同级别的企业解决方案，提供组件式的二次开发接口。相比前面所提的 Skyline Globe 和 EV-Globe，Google Earth 缺乏空间分析和海量数据管理能力，二次开发广度、灵活度也不前两者。

InfoEarth iTelluro 是武汉地大信息工程股份有限公司推出的网络三维地理信息系统平台[2]。InfoEarth iTelluro 面向网络、面向分布式、面向海量三维空间数据的可视化、分析和服务发布。InfoEarth iTelluro 基于.NET 技术构建，可以以组件方式集成到已有的地理信息服务系统。

FreeEarth 是一个西安恒歌数码科技有限责任公司基于 OpenSceneGraph 研发的基于多行业应用的三维数字地球研发平台[3]。FreeEarth 以付费方式开发源码，支持跨平台，支持天空、陆地、海洋等多种渲染效果，支持主流格式数据的组织和三维可视化。

World Wind 是 NASA 推出的三维地理科普软件，它是免费开源的。World Wind 利用了 NASA 的卫星和航天飞机资源，集合了多颗卫星影像数据和航天飞机遥感数据，并且提供每天的气象、环境监测数据。World Wind 提供 C#和 Java 两个版本 SDK，因为 World Wind 本身是开放的，开发人员可以利用 World Wind 源码或 SDK 开发三维地球应用。基于 World Wind，国内也衍生出几种版本的三维数字地球开发包。

Cesium 是一个基于 WebGL 的跨平台、跨浏览器的虚拟地球引擎，它是免费开源的。Cesium 支持 2D、2.5D、3D 地图显示，支持多种数据源和模型的可视化，支持对卫星资源进行管理和任务规划，可以运行在支持 HTML5 规范的浏览器和移动设备环境中。

Ossim 是一个功能强大的地理信息处理工具集，它是免费开源的。Ossim 可用于处理影像、地图、地形和矢量数据。OssimPlanet 是 Ossim 的子集，是一个基于 OpenSceneGraph 的三维地理显示器，支持多种类型、多种数据源的数据显示。

osgEarth 是一个基于 OpenSceneGraph 的三维地图软件开发包，它是免费开源的。osgEarth 可以运行在多个平台，支持多种类型原始地理信息数据的分析和可视化，可扩展，开发人员可以通过编写 xml、C++代码构建三维场景和使用 Python、Lua 等脚本语言控制场景。

总之，目前的三维空间信息技术的研究和开发领域一片欣欣向荣，处于百花齐放，百家争鸣的境况，相关的技术和产品共同细分三维空间信息技术应用的市场，推动三维空间信息技术的发展。

1.3.2 对比

表 1-1 三维数字地球软件开发包对比

序号	SDK 名称	是否免费	是否开源	二次开发语言
1.	Skyline Globe	否	否	多语言
2.	EV-Globe	否	否	多语言
3.	GeoGlobe	否	否	多语言
4.	Google Earth	否	否	多语言
5.	InfoEarth iTelluro	否	否	.NET
6.	FreeEarth	否	付费开源	C++
7.	World Wind	是	是	Java、C#
8.	Cesium	是	是	JS
9.	OssimPlanet	是	是	C++
10.	osgEarth	是	是	XML、C++

1.3.3 评价

相比免费开源的三维数字地球软件开发包，商用三维数字地球软件开发包功能更多，提供更详细、周全的企业解决方案。像 Skyline Globe 这类商用三维数字地球软件开发包，面向企业应用，体系完整，设计开发工具齐全，应用领域广泛。

但是从另一角度来说，商用三维数字地球软件开发包不像免费开源的软件开发那样可以被任意修改和再发布，不一定能灵活适配不同的开发需求，不利于三维数字地球技术的推广

和应用。现有的像 osgEarth 等三维数字地球软件开发包，技术成熟，渲染效果好，数据格式支持齐全，而且还在稳步成长中。无论是技术学习还是应用开发，开源的 osgEarth 等三维数字地球软件开发包都是很好的选择。

§1.4 选题目标与研究内容

1.4.1 选题目标

设计一套符合业务逻辑的通用接口，并设计开发出满足该接口规范的链接库和组件，编写接口调用示例，编写整理项目开发文档和用户手册，提出一套跨语言、跨浏览器的三维数字地球应用开发的解决方案。

1.4.2 研究内容

技术学习与研究的内容：

1) OpenSceneGraph 三维场景渲染引擎、osgEarth 三维地图引擎
2) OpenThreads 多线程库
3) COM 与 ATL 技术
4) FireBreath 跨浏览器插件技术
5) CMake 跨平台构建工具
6) Linux 编程知识，QT 编程知识
7) Android NDK 应用程序开发

项目开发的内容：

1) 收集项目需求，设计项目总体架构，编写项目开发计划和项目开发文档。
2) 修改 OpenSceneGraph 和 osgEarth 的源码。
3) 设计开发核心基础库 HS_Libs、HS_OSG、HS_Threads。
4) 设计开发封装 osgEarth 和基础库的功能的 HS_Earth_Core。
5) 设计开发实现符合业务逻辑的通用接口的 HS_Earth。
6) 设计开发 COM 组件 HS_Earth_COM，以及开发使用 HS_Earth_COM 组件的示例。
7) 设计开发跨浏览器插件 HS_Earth_Web，以及开发使用 HS_Earth_Web 跨浏览器插件的示例。
8) 开发 Linux 和 Android 平台下的演示示例。

§1.5 论文组织结构

本论文主要分为六章：

第一章 简要介绍选题的来源、目的、意义、目标、工作内容，概要介绍、分析、比较目前国内外主流的三维数字地球软件开发包。

第二章 综合分析、介绍 OpenSceneGraph、osgEarth 的架构、核心技术和流程。

第三章 概要描述项目需求规定和总体架构设计。

第四章 详细介绍基础库 HS_Libs、HS_OSG、HS_Threads 的开发核心思想，简要介绍 OpenSceneGraph、osgEarth 的源码修改内容。

第五章 详细介绍核心库 HS_Earth_Core、HS_Earth、HS_Earth_COM、HS_Earth_Web 的开发核心思想，展望 HS_Earth_Android，还简要介绍了不同语言、不同平台的三维数字地球应用的构建过程。

第六章 总结选题研究和项目开发过程的收获与不足，展望未来的改进和发展。

论文最后为致谢内容、参考文献列表。

第二章 osgEarth 相关技术介绍

§2.1 OSG 综述

2.1.1 OSG 简介

OpenGL 是跨平台、高性能的图形硬件编程接口，OpenGL 是一套图形硬件接口实现的工业标准。使用 OpenGL，开发人员可以使用不同语言在不同平台、不同图形显卡硬件上开发出高性能的交互式图形应用。

OSG（全称: OpenSceneGraph）是基于 OpenGL 的高性能、面向对象、跨平台的三维场景渲染引擎，使用 C++语言编写而成，它是免费开源的。OSG 提供了出色的场景管理和图形渲染优化，被广泛用于虚拟仿真、虚拟现实、科学和工程可视化等领域，例如地理信息系统（GIS）、计算机辅助设计（CAD）等[4]。

OSG 具备快速开发、面向对象、高性能、可扩展、可移植、跨平台等特点，是一个成熟的三维场景渲染引擎。

OSG 封装了大部分 OpenGL 底层接口，抽象出场景类和图形渲染类，开发人员可以在不关心底层硬件和渲染方式的情况下，使用面向对象的思想组织、渲染场景。

OSG 除了封装了 OpenGL 底层，支持所有 OpenGL 功能之外，还提供了多种计算机图形学技术的面向对象实现，例如场景剪裁、层次细节、粒子系统、场景动态调度、多线程渲染等技术。

OSG 内部科学应用了多种设计模式的思想，充分考虑了三维场景设计的灵活性、扩展性要求，提供了强大的可扩展能力。开发人员可以基于现有的面向对象的 OSG 类，扩展出特定的场景类、功能、模块，包括数据文件类型、场景结点、事件处理器、回调等。

由于 OSG 是基于 OpenGL 构建，应用了多种设计模式，充分考虑了引擎可移植的要求，所以 OSG 可以运行在 Windows、Unix /Linux、Mac OS X 等操作系统和 Android、IOS 等移动操作系统上，OSG 支持 Python、Lua 等脚本操作。开发人员开发的基于 OSG 的应用程序可以轻松移植到各个平台和环境。

OSG 拥有日益完备的开发文档、活跃的社区，加之免费开源，应用领域不断扩大，全球用户量不断增加。

2.1.2 OSG 架构

OSG 作为图形中间件，构建于工业标准图形硬件接口 OpenGL 之上，抽象、封装了 OpenGL 的大部分接口和功能，基于 OSG 可以开发出更高层次的三维场景应用。

OSG 主要可分为五个部分，它们分别是：

1) OSG 核心库。OSG 核心库是 OSG 最基础的部分，封装了 OpenGL 大部分功能，提供最基础同时是最重要的场景组织管理和场景渲染的功能。OSG 核心库包括 osg 基础库、osgDB 文件插件注册器库、osgUtil 实用工具库、osgGA 交互库等。
2) 节点扩展库。OSG NodeKits 是 OSG 核心库的扩展，提供了特殊结点和渲染特效。节点扩充库包括 osgAnimation 动画库、osgFX 特效库、osgManipulator 场景操控库、osgParticle 粒子系统库、osgShadow 阴影特效库、osgTerrain 地形处理库、osgText 文字处理库、osgVolume 体渲染实现库、osgWidget 用户界面控件库等。
3) 文件读写插件。OSG 对不同格式场景数据文件的读写的支持使用了插件机制，开发人员可以通过自定义插件来扩充 OSG 对特定格式数据文件的支持。OSG 插件机制是基于责任链的，只有当需要读写某种格式数据时，OSG 才会去查找并加载支持该格式读写的插件。
4) 互操作库。OSG 互操作库用于对集成其他开发环境进行支持，互操作库提供一个与语言无关、可在运行时被调用的接口，使其他像 Java、Tcl、Lua 和 Python 等支持反射式和自省式编程的语言可以对 OSG 进行交互。
5) 工具集和示例集。OSG 提供了辅助场景创建、管理、调试等的工具集和演示 OSG 基本改变和功能的示例集，利用这些资源，开发人员可以减少开发时间，提高开发效率。

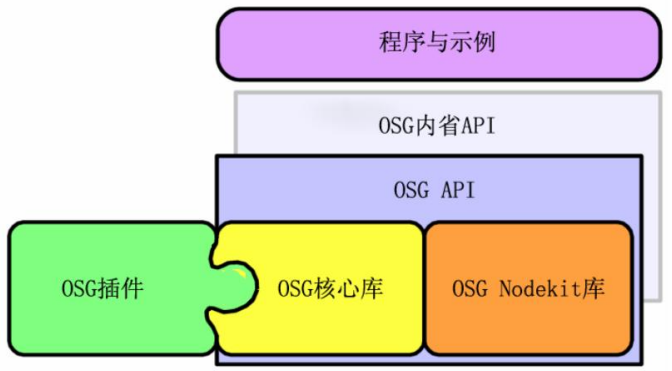

图 2-1 OSG 架构图

2.1.3 内存管理机制

OSG 内部使用引用计数的策略来进行内存管理，OSG 中大多数类继承于 osg::Object，而 osg::Object 继承于 osg::Referenced。引用计数基类 osg::Referenced 内部保存着对象的引用计数信息，智能指针模板类 osg::ref_ptr 管理着 osg::Referenced 及其派生类的引用计数。

当 osg::Referenced 对象被一个 osg::ref_ptr 对象引用时，osg::Referenced 对象内部的引用计数会增加 1；当一个 osg::ref_ptr 对象不再引用 osg::Referenced 对象时，osg::Referenced 对象内部的引用计数会减少 1；当 osg::ref_ptr 对象要去除对 osg::Referenced 对象的引用，并且发现该 osg::Referenced 对象的引用计数只有 1 时，说明 osg::Referenced 对象没有被其他智能指针引用，当前 osg::ref_ptr 对象会自动销毁 osg::Referenced 对象所占用的内存。

在 OSG 的内存管理机制中，为了 osg::Referenced 派生类对象能够被智能指针正确管理，

osg::Referenced 的派生类的构造函数必须声明为 virtual protected，osg::Referenced 派生类对象必须是通过 new 关键字创建的。如果将 osg::Referenced 的派生类析构函数声明为 virtual public，也就是允许直接定义 osg::Referenced 派生类对象，在后续的操作中会出现内存泄漏或者崩溃的风险，osg::Referenced 派生类对象没有被智能指针正确释放。

假设 A 是 osg::Referenced 的派生类，A 将析构函数声明为 virtual public。请看下面演示内存操作错误的示例代码：

```
//代码示例
A a;
{
    osg::ref_ptr< A > refA = &a;
}
```

因为 a 在进入局部作用域之前没有被智能指针引用，所以 a 将地址赋值给 refA 后，refA 将 a 的引用计数加 1，但是在退出局部作用域之前，refA 对象会执行析构函数，判断到 a 的引用计数为 1，知道 a 没有被其他智能指针引用，就会删除掉内部对象指针所指向的内存块，造成崩溃。所以为了去除隐患，osg::Referenced 派生类应将析构函数声明为 virtual protected。

2.1.4 场景组织结构

OSG 的场景图形管理采用的是包围体层次（Bounding Volume Hierachy）结构，使用包围体将一组场景物体包围于一个简单的空间三维几何体中，可以提高各种场景运算的效率[5]。OSG 中兼顾性能提升和简易场景组织，对物体采用包围球和包围盒两种形式。

像大部分采用包围层次的场景引擎一样，OSG 也是采用树结构来组织和管理三维场景信息，通过树结构自顶向下、分层组织场景数据。一个场景树从根节点开始，可以添加不同的场景组节点或者叶子节点，组节点可以继续添加场景节点。场景节点保存着几何信息和渲染状态信息等信息，场景组节点的属性和状态影响其子节点的属性和状态。

OSG 基本的场景结点有：

1) 场景节点基类 osg::Node。osg::Node 是场景树结构组织的基础，osg::Node 定义了场景节点的基础属性和方法，提供了外部访问节点的接口。
2) 场景组节点 osg::Group。osg::Group 可以添加其他场景节点，包括 osg::Group 类型的节点，osg::Group 组织同类结点，osg::Group 及其派生类保存了自身及其子类的几何和渲染状态信息，形成自定向下、分层的树结构，使整个场景树结构结构清晰明了。
3) 场景叶子节点 osg::Geode。osg::Geode 是场景叶子节点，不能继续添加其他子节点，osg::Geode 保存和管理着一个或者多个 osg::Drawable 可绘制物体的几何信息和渲染状态信息，这些 osg::Drawable 物体可以是几何物体、图片、文字等。

除了基本场景节点以外，OSG 还提供其他具有特定功能的场景节点。例如空间转换节点 osg::Transform、开关节点 osg::Switch、细节层次节点 osg::LOD 等等。

2.1.5 场景图形渲染

OSG 的场景图形渲染主要经历三个过程：更新、拣选、绘制。这些过程都涵盖了对场景树进行遍历的过程，通过使用节点访问器对场景树中的节点进行遍历访问，最终将正确的 OpenGL 指令发送到图形硬件上进行绘制。场景渲染的三个遍历过程发生在场景每一祯画面的渲染过程中。

场景更新遍历过程中，更新的操作通过执行更新回调来完成，这些更新包括人机交互事件、数据的动态加载和管理、自定义的场景更改操作。场景更新遍历是动态场景的保证。

场景拣选遍历，通过遍历计算，剔除场景中不可见的物体，剪裁不可见面，生成要绘制出来的几何体列表。场景拣选剔除提高了渲染性能。

场景绘制遍历，遍历场景树，根据场景更新信息、场景拣选信息，将场景以 OpenGL 指令的方式传送到图形硬件进行绘制出来。

§2.2 osgEarth 综述

2.2.1 osgEarth 简介

osgEarth 是一个基于 OSG 的功能丰富、跨平台的三维地图软件开发包，使用 C++语言编写而成，它是免费开源的。osgEarth 是一个在稳步成长中的三维地图软件开发包，除了具备其他三维地图软件开发包的一般功能之外，它在功能方面、性能方面都有过人之处，越来越受到开发者的青睐。

使用 osgEarth 可快速开发、显示三维地图场景，开发门槛低。osgEarth 支持 xml 和 C++代码两种方式构建三维地图场景。osgEarth 用户可以通过编写少量的结构清晰明了的 xml 代码，即可快速构建一个三维地图场景文件，然后使用 osgEarth 提供的 osgearth_viewer 将三维地图场景显示出来；osgEarth 代码风格与 OSG 的代码风格差不多，OSG 开发者可以轻松用代码构建、显示一个三维地图场景。

osgEarth 支持多种类型地理数据、地理数据服务的显示和操作。osgEarth 可以通过读取原始地理数据，并动态加载到三维地图场景中进行显示，而无需提前构建三维模型；osgEarth 支持读取多种不同的地图服务；osgEarth 支持不同类型数据的整合显示；osgEarth 支持脚本访问、操作场景中的数据。

osgEarth 支持多种三维场景节点。由于 osgEarth 是构建在 OSG 之上的，osgEarth 支持将 OSG 支持的模型、特效等集成到三维地图场景中。osgEarth 内建了丰富的三维场景节点，例如，多种类型天空、海洋和有地理特色的模型结点。开发人员可编写不同的场景节点加入到三维地图场景中。

osgEarth 性能优越。osgEarth 使用了层次细节、瓦片数据图层四叉树管理、自动剪裁等技术；使用了类模板和函数模板来管理参数，使得类对象的属性配置可根据需要动态创建。得益于科学合理高效的数据结构组织、管理，osgEarth 的数据加载效率、渲染效率非常高。

2.2.2 osgEarth 架构

osgEarth 构建于 OpenSceneGraph 三维场景渲染引擎之上。osgEarth 继承了 OpenSceneGraph 的工程构建风格、代码风格，osgEarth 可以集成到其他 OpenSceneGraph 场景中。

osgEarth 主要由以下三个部分组成：

1) osgEarth 基本库。osgEarth 基本的类库包括定义了基本地理数据类型和结构、基本三维地图场景类和相关属性的 osgEarth 库、定义标注类型和相关属性的 osgEarthAnnotation 库、定义矢量要素和相关属性的 osgEarthFeatures 库、定义绘制地理数据风格和相关属性的 osgEarthSymbology 库、定义开发辅助工具类的 osgEarthUtil 工具库。
2) osgEarth 插件库。osgEarth 插件库是 osgEarth 的重要组成部分。osgEarth 大部分的数据读写、功能实现都是以插件的形式提供，充分发挥了 OpenSceneGraph 插件机制的可选择、可扩展、不影响核心运行的优势，使 osgEarth 可以更灵活地被更改和扩展。
3) osgEarth 工具集和示例集。osgEarth 工具集用于分析、处理、显示、调试三维地图，osgEarth 示例集包含 osgEarth 功能演示，接口调用代码示例等。

2.2.3 基本场景节点类型

osgEarth 支持多种地图场景渲染节点，例如图层、标注、天空、海洋等。下面是 osgEarth 基本场景类。

osgEarth::Map 类是 osgEarth 的最主要的类，osgEarth::Map 代表场景三维地图模型类，是影像图层 osgEarth::ImageLayer、高程图层 osgEarth::ElevationLayer、矢量模型图层 osgEarth::ModelLayer 等图层的容器，保存着三维地图的地理数据信息。

osgEarth::MapNode 是 osgEarth 中 osgEarth::Map 对象在三维场景中的节点，是 osg::Group 的派生类，它根据 osgEarth::Map 对象进行构建，需要添加到 OSG 场景中进行渲染。

osgEarth::TerrainEngineNode 是地形引擎节点，位于 osgEarth::MapNode 的节点树下。osgEarth::TerrainEngineNode 主要用于处理影像、高程图层的显示。

osgEarth::ImageLayer 是影像图层类。osgEarth::ImageLayer 通过读取栅格影像数据，作为纹理，被 GPU 合成到三维地图几何体的表层。

osgEarth::ElevationLayer 是高程图层类。osgEarth::ElevationLayer 通过读取高程数据，建立高度图合成到三维地图中。

osgEarth::ModelLayer 是矢量模型图层类。osgEarth::ModelLayer 通过读取地理数据文件中的矢量数据信息建立 OSG 三维几何模型或者直接读取三维模型，作为一个图层加入到三维地图场景中进行渲染。

osgEarth::MaskLayer 是地形覆盖图层。osgEarth::MaskLayer 用于特别标出地形的某个部位，你可以将一个已经建好的三维地形模型插入到已有场景地形中的某个部位。

其他三维地图场景类还有各种类型的天空、海洋、标注、几何形状等等。

2.2.4 插件类型

1. 插件分类

osgEarth 中，存在着各式各样的驱动，驱动是以插件的方式进行管理的，驱动即是插件，因此驱动是可选择、可扩展的。osgEarth 插件可分为数据处理插件和功能提供插件两种。

2. 数据处理插件

osgEarth 支持主流的开放标准的地理数据、三维模型数据的读取，支持使用多种类型驱动进行数据读取。

osgEarth 中，一种数据类型代表着一种类型的数据源，一种类型的数据源可以被不同的方式进行处理。例如，GeoTIFF 格式的影像数据可以用于构建 osgEarth::ImageLayer，GeoTIFF 格式的高程数据可以用于构建 osgEarth::ElevationLayer，Shapefile 格式的矢量要素可以用于构建 osgEarth::ImageLayer，Shapefile 格式的矢量要素可以用于构建 osgEarth::ModelLayer。

数据处理驱动（插件）中，一种驱动代表着一种数据读取、产生、处理的方式或者工具，一种驱动支持一种或者多种类型数据的处理，例如 GDAL 驱动可以读取 TIFF 影像数据、TMS 影像数据等。有些类型驱动可以自身创建场景数据进行处理，例如 Debug 驱动可以根据 LOD 层次，产生不同层次瓦片的可绘制的几何形状，用于调试图层的 LOD 信息。

数据处理驱动有 TileSource 瓦片数据驱动、FeatureSource 矢量要素数据驱动、ModelSource 矢量模型数据驱动三种。以下是主要数据处理驱动的列表和功能描述。

表 2-1 TileSource 驱动表

序号	名称	描述
1	AGGLite	将矢量要素数据栅格化成影像瓦片，用于将矢量数据以影像图层的方式渲染。
2	ArcGIS	从 ESRI ArcGIS 服务器的 REST API 服务中读取影像瓦片。
3	Bing	从 Bing 地图中读取 REST API 服务。
4	ColorRamp	读取 ColorRamp 瓦片服务。
5	Debug	根据瓦片的 LOD 和位置信息渲染瓦片轮廓，用于调试、显示瓦片 LOD 层次信息。
6	GDAL	GDAL 是一个功能强大的开源地理数据抽象库，可从数据库或者 Web 服务中读取数据，支持多种数据格式，例如最常用的 GeoTIFF、JPEG 和 ECW 等。
7	MBTiles	读取 MBTiles 数据库中的数据。
8	Noise	使用 Perlin noise 生成器产生不规则的地形。
9	OSG	使用 OSG 的图片读写插件读取图片文件。
10	TileCache	读取 TileCache 瓦片。
11	TileService	读取 NASA WorldWind 瓦片。
12	TMS	读取 OSGeo 的 TMS 标准格式的数据。
13	VPB	VPB(Virtual Planet Builder)是一个用于构建分页地形模型的应用程序。osgEarth 中的 VPB 驱动读取 VPB 模型，转换成可以在 osgEarth 引擎中渲染出来的形式。
14	WCS	读取 OGC Web Coverage Service 标准的高程网格数据瓦片。
15	WMS	读取 OGC Web Map Service 标准的影像数据资源。

16	XYZ	读取 X/Y/LOD 标准的 Web 地图瓦片数据仓库。
17	Yahoo	读取雅虎地图服务。

表 2-2 ModelSource 驱动表

序号	名称	描述
1	Geom	调用定义的样式，将矢量要素数据转换成 OSG 支持的可绘制几何形状 osg::Geometry。
2	Simple	加载外部三维模型，并正确放置在地图上。
3	Stencil	使用模板缓存技术，将矢量要素数据合成到地形上。

表 2-3 FeatureSource 驱动表

序号	名称	描述
4	OGR	读取满足 OGR Simple Feature Library 格式的任意格式的矢量文件，常见的包括 ESRI Shapefiles 和 GML。
5	TFS	读取 Tiled Feature Service 数据仓库中的矢量数据。
6	WFS	读取 OGC Web Feature Service 资源中矢量数据。

3 功能提供插件

由于应用了插件机制，osgEarth 中的部分功能是可以组合的。在实现例如大地基准面、地形引擎、天空、海洋等功能的时候，有不同类型的实现、不同的算法和不同的内部数据对象组织结构，osgEarth 将每一种功能的实现都封装成插件，可动态从插件责任链中加载。下面列出地形引擎插件。

表 2-4 地形引擎插件表

序号	名称	描述
1	MP	osgEarth 默认的地形引擎。MP 地形引擎使用瓦片多管道混合技术（tile-level multipass blending technique）渲染一定数目的影像图层。
2	BYO	BYO 意为 Bring Your Own Engine，是自定义地形引擎接口，你可通过重新设置地形引擎，然后将其接入到 osgEarth 体系中。

2.2.5 数据加载及组织

与 OSG 一样，osgEarth 也是使用插件对文件进行读写操作的。由于 osgEarth 是基于 XML 需求开发的，所以 osgEarth 在读取 XML 格式的.earth 地图场景文件时，会先加载.earth 文件的读写插件，解析 XML 内容，提取需要构建的场景对象以及场景对象的参数 ConfigOptions，再根据每个对象的 ConfigOptions 构建该场景对象，然后组合形成一个三维地球场景。

作为三维地球场景的重要组成，地形引擎负责瓦片地形数据的组织管理。从 osgEarth 默认的 MP 地形引擎分析，MP 地形引擎将瓦片地形数据以四叉树的数据结构管理起来。MP 地形引擎的四叉树中，四叉树节点 osgEarth::TileNode 对象的位置的信息被保存在 osgEarth::TileKey 对象中，osgEarth::TileKey 对象保存着 LOD 层次、瓦片的 x 坐标、瓦片的

y 坐标等信息，每个 TileNode 包含着在当前 TileKey 下，osgEarth::Map 的影像、高程图层的瓦片信息。

2.2.6 地图场景更新与渲染

osgEarth 使用 osgEarth::MapFrame 对 osgEarth::Map 的更新进行管理，一个 osgEarth::MapFrame 对象是 osgEarth::Map 对象在某一个阶段的状态的副本，一个场景中有多个 osgEarth::MapFrame 对象。一些需要对场景进行访问的场景操作对象直接对 osgEarth::MapFrame 对象进行访问，而不是直接对 osgEarth：：Map 对象进行访问，这样就避免了场景操作类对象更新 osgEarth::Map 对象时会对其他类访问 osgEarth::Map 对象造成冲突。

osgEarth::Map 和 osgEarth::MapFrame 对象内部保存着一个版本号 osgEarth:: Revision 对象，当 osgEarth::Map 对象的场景模型或者状态发生改变的时候，不会立即反映到渲染器，osgEarth::Map 对象内部会将版本号加一，当场景操作类访问 osgEarth::MapFrame 对象时，osgEarth::MapFrame 对象会先比较自身版本号和 osgEarth::Map 对象的版本号，如果发现两者不一致，osgEarth::MapFrame 对象将会按照 osgEarth::Map 对象对自身进行更新。

正如前文所述，地形引擎是 osgEarth 的重要组成部分。作为 osgEarth 默认的地形引擎，MP 负责瓦片数据的组织、更新和管理，osgEarth::Drivers::MPTerrainEngine::MPTerrainEngineNode 是地图场景节点类。当 osgEarth::Map 对象中的场景模型发生更改时，会将更改的信息循环传递给回调函数，其中 MP 地形引擎会通过回调函数接收到这些更改信息。MP 地形引擎会根据更改信息，更改四叉树的数据、更改底层 OpenGL 的对象、更改渲染状态等等。

MP 地形引擎对瓦片进行渲染时，会根据当前场景中的 LOD 层次，查找定位到与该 LOD 层次相对应的四叉树层次。对该层次的在场景中可视范围内的 TileNode 进行访问，使用纹理数组传递 TileNode 对象中的所有影像、高程图层的影像瓦片给 GPU 进行合成。

2.2.7 不足之处

osgEarth 目前仍有很多有待完善的地方：

1) osgEarth 旨在加速三维地图可视化的过程，弱化了动态地图数据的加载、删除等功能。不完全支持动态改变三维地球场景模型。
2) osgEarth 目前对移动设备平台的支持不足，官方目前仅支持 IOS 平台，尚未支持 Android 平台。
3) 基于 XML 驱动的需求构建起来的三维地球，在 XML 快速构建三维地图这方面见长，但是在其他方面面临着不足，缺乏灵活的编程接口支持，这也限制了 osgEarth 的应用领域扩展。
4) 代码开发文档少。

§2.3 本章小结

本章第一节概要介绍了三维场景渲染引擎 OpenSceneGraph，并分析研究 OpenSceneGraph 的架构、内存管理机制、场景组织结构、场景图形渲染等内容。第二节概要介绍基于 OpenSceneGraph 构建起来的三维地图引擎 osgEarth，并分析研究了 osgEarth 的架构、场景节点类型、数据与驱动类型、数据加载及组织流程、地图 引擎渲染流程，并最后分析了目前版本 osgEarth 的不足之处。

第三章 项目分析与设计

§3.1 项目概述

本项目旨在开发一个跨开发语言、跨平台的三维数字地球软件开发包。该开发包面向二次开发人员，可被用于构建不同语言、不同平台下的三维数字地球应用程序。

项目输出产品有：

1) 基础链接库：HS_Libs、HS_OSG、HS_Threads、HS_Earth_Core、HS_Earth
2) COM 组件 HS_Earth_COM
3) 跨浏览器插件 HS_Earth_Web
4) Android NDK 链接库 HS_Earth_Android
5) 开发文档以及用户手册

§3.2 需求规定

3.2.1 功能需求

1) 三维地球场景物体。三维数字地球场景中所有的可绘制的组成部分都是以物体的形式构成的，像图层、天空、模型、图片、标注等，抽象成场景物体，它们有共同的物体基类。

2) 三维地球场景组。三维地球场景中的物体可以被场景组分组管理起来，场景组派生自物体基类，场景组也是场景物体，所以场景组对象可以加入到另外一个场景组对象中。

3) 三维地球场景动态管理。三维场景内的物体可以被动态更改，例如物体的增加、隐藏、删除；三维场景的渲染状态可以被动态更改；三维场景可动态交互。

4) 多数据源支持。封装 osgEarth 中所支持的数据接口，封装成数据源，扩充自有数据源。

5) 多线程多任务管理。程序操作应减少阻塞渲染线程时间，将耗费时间长的操作封装成任务，这些任务被独立于渲染线程的多线程执行。

6) 多 GUI 窗口集成支持。核心代码支持与多个 GUI 窗口集成，例如 Win32、QT、Android 等。

3.2.2 性能需求

1) **加载速度快，加载阻塞时间短**。阻塞渲染线程的操作所占用的阻塞时间短，不应在阻塞的时候进行大量繁杂的运算，不长时间妨碍其他操作，不影响交互操作。
2) **渲染速度快**。场景组织结构清晰明了，场景物体访问高效，场景渲染速度快。
3) **渲染线程与 GUI 窗口线程分离**。渲染线程独立于 GUI 窗口线程，两者的刷新及操作应该分离，互不干扰阻碍。
4) **状态更改反应时间短**。对于场景或者场景物体的状态更改，能够快速被渲染出来，之间的延迟时间短。
5) **事件、回调反应时间短**。事件触犯能迅速传递出来，回调操作能迅速被执行，延迟时间短，反应时间快。

3.2.3 接口需求

1) **统一的接口关系**。针对不同语言、不同平台的应用程序开发的组件，实现符合统一接口规范的接口关系。
2) **统一的调用接口**。不同平台、不同语言的应用程序可以通过调用统一命名规则的接口访问底层组件。
3) **统一的事件接口**。组件接口层为上层应用程序提供相同的事件触发和处理机制。

3.2.4 故障处理需求

1) **提供错误处理接口**。组件为应用程序层提供错误处理接口，当内部发生内部错误，这些错误能够被顶层应用程序接收并处理。
2) **内部容错处理能力高**。除了为顶层应用程序提供错误处理接口之外，底层也应充分考虑错误处理的情况，提高容错处理能力。

3.2.5 其他需求

1) **可维护性**。软件开发文档编写规范，文档结构清晰，代码风格统一、明了，接口设计合理。
2) **可扩展性**。底层功能封装和接口实现要充分考虑可扩展性，尽量减少层次之间的耦合性。
3) **易学性**。学习、开发成本低，简单易学，接口关系清晰，接口命名规范易懂。
4) **健壮性**。充分考虑、测试，保证各单元正确、高效运行，能适应多数环境和平台，能适应外部条件变化。
5) **可转换性**。开发环境可轻松转移，受平台牵制小。

§3.3 运行环境规定

3.3.1 设备

开发、运行基于 HsEarth 三维数字地球软件开发包的应用程序，需要支持以下硬件条件：

表 3-1 硬件设备要求

	PC	**移动设备**
芯片架构	x86、x64	arm
OpenGL 支持	OpenGL 1、OpenGL 2	OpenGL ES 2

3.3.1 支持开发环境

1) 三维数字地球软件开发包核心代码支持跨平台、跨编译器编译。
2) 软件开发包输出产品支持 Windows、Unix/Linux、Android 的三维数字地球应用开发应用。
3) 软件开发包在 Windows 下支持多语言开发应用，在 Unix/Linux 下支持 C++开发应用，在 Android 平台支持 C/C++和 Java 语言开发应用。

§3.4 总体架构设计

3.4.1 总体架构

三维数字地球软件开发包体系是从下至上一层层封装扩展而架构起来的。三维数字地球软件开发包从下至上大致可以分为三个层次：核心层、组件接口层、应用程序层（见图 3-1）。

核心层由 C++开发，主要封装、扩展了 OSG、osgEarth 的功能和接口，定义了符合业务逻辑的统一接口。

组件接口层建立在核心层之上，根据不同平台和语言，开发出相应的组件，为目标语言、平台的提供符合统一接口规范的接口。

应用程序层建立在组件接口层之上，通过调用不同平台下的组件，开发出基于这些组件的应用程序。

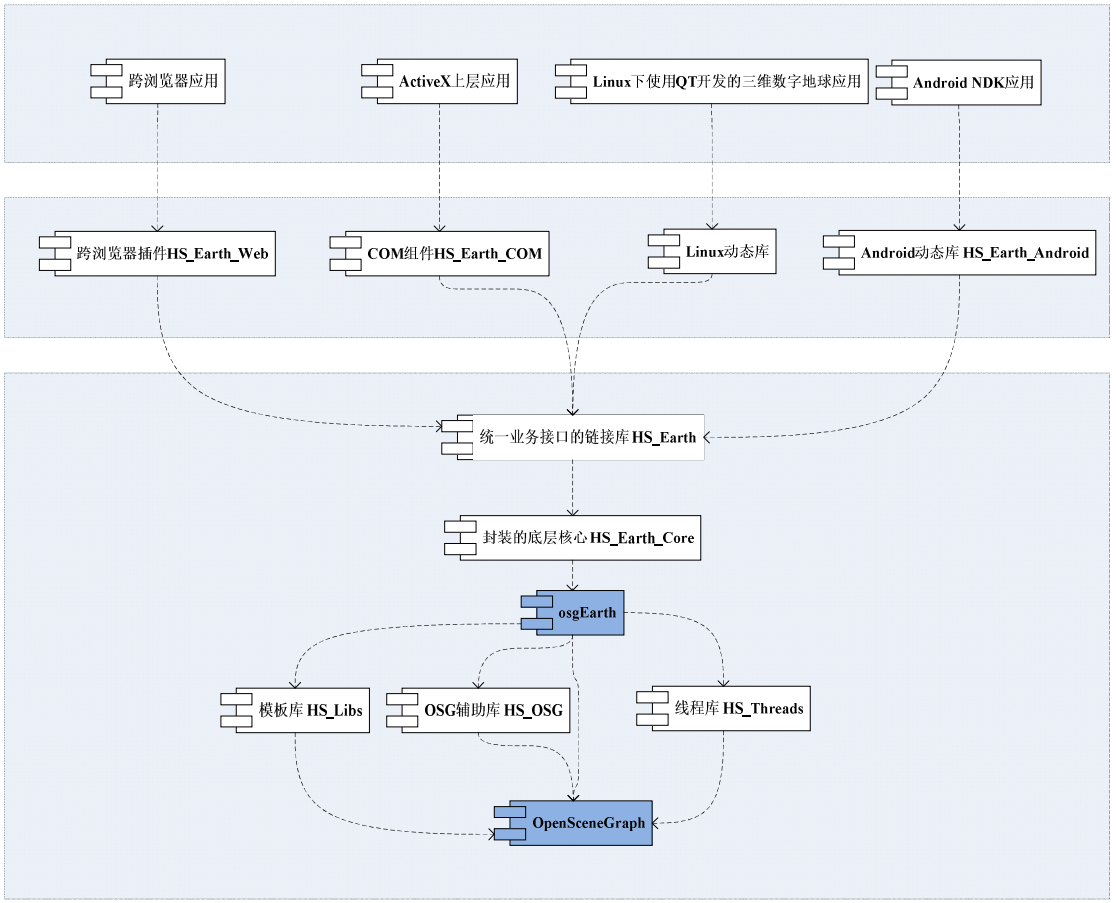

图 3-1 总体架构图

3.4.2 各层组件概述

核心层包括：

表 3-2 核心库表

序号	名称	简介
1	OpenSceneGraph	修改后的 OpenSceneGraph 链接库
2	HS_Libs	辅助开发的模板库，包含数据结构模板
3	HS_OSG	辅助 OpenSceneGraph 开发的辅助库
4	HS_Threads	扩充了 OpenThread 功能的线程库
5	osgEarth	修改后的 osgEarth 链接库
6	HS_Earth_Core	封装了 OSG、osgEarth 功能和接口的链接库
7	HS_Earth	封装了 HS_Earth_Core 接口和功能，定义符合业务逻辑的功能和接口的链接库

组件接口层包括：

表 3-3 组件表

序号	名称	简介
1	HS_Earth_COM	转换 HS_Earth 定义的接口成 COM 接口的组件
2	HS_Earth_Web	转换 HS_Earth 定义的接口成 FireBreath 跨浏览器接口的插件
3	HS_Earth_Android	转换 HS_Earth 定义的接口成 Android NDK 接口的 Android 链接库

3.4.3 运行流程

顶层应用程序对接口的调用和对事件的处理，会从上往下逐步传递到核心层代码进行处理。由于是定义了统一的接口规范，顶层应用程序的接口调用方式和事件处理方式是类似的，顶层应用开发人员无需关心底层实现，可以以相同的开发流程为不同平台、不同语言开发应用程序。

§3.5 本章小结

本章简要介绍了三维数字地球软件开发包的项目开发内容，列出了项目开发的需求规定和运行环境规定，最后介绍了项目总架构的设计和各层组件的功能。

第四章 核心库实现

§4.1 HS_Libs 模板库

4.1.1 项目范围

项目开发 HS_Libs 模板库。HS_Libs 库包含着一些开发 OSG 或者 osgEarth 项目需要的类和模板。目前 HS_Libs 主要有数据组织模板，用于高效组织数据结构，辅助数据自动管理，动态创建及释放数据对象等。

4.1.2 数据组织模板

1) 动态列表模板 DynamicList

DynamicList< T >的模板参数 T 为 osg::Referenced 及其派生类，DynamicList< T >内部拥有一个 std::List< osg::ref_ptr< T > >的数据成员指针，指向在需要时才动态创建的 std::List< osg::ref_ptr< T > >的对象。DynamicList 模板包含了通常操作列表项的成员函数，也提供访问内部列表数据成员的函数。

2) 动态数组向量模板 DynamicVector

DynamicVector< T >的模板参数 T 为 osg::Referenced 及其派生类，DynamicVector< T >内部拥有一个 std::vector< osg::ref_ptr< T >* >的数据成员指针，指向在需要时才动态创建的 std::vector< osg::ref_ptr< T >* >的对象。DynamicVector< T >内部数组向量对智能指针类 osg::ref_ptr< T >对象的指针进行操作，减少了移动内部成员造成的损耗，在删除 osg::ref_ptr< T >* 所指向的对象的时候，会释放 osg::ref_ptr< T >对象，因为 osg::ref_ptr 具备内存管理的功能，会自动检查 T 对象的引用计数，并释放引用计数为 0 的 T 对象。

§4.2 HS_OSG 链接库

4.2.1 项目范围

HS_OSG 是开发 OSG 应用程序的辅助库，用于协调场景更新与渲染之间的操作，避免两者冲突。

4.2.2 场景更新操作冲突避免

1. 功能介绍

在编写 OSG 程序时，对于场景更新与渲染的冲突避免有两种解决方法。

第一种方法就是为你的场景节点设置读写锁（有关 OSG 中线程锁概念请看第 4.3.1 节）。在场景更新遍历到所操作的场景节点时将写锁锁上，在绘制结束后解开写锁（关于 OSG 更新渲染流程请看第 2.1.5 节），这样通过线程锁控制资源的读写访问，避免了更新和渲染冲突。第一种方法操作起来要非常小心，不恰当应用一样会出现冲突。

第二种方法是使用 OSG 内部提供的自定义操作机制。osg::Operation 是一个自定义操作抽象类，它有一个自定义操作纯虚函数 virtual void operator()(Object*) 。如果需要自定义对场景模型的更改，可以通过继承 osg::Operation，并实现其自定义操作纯虚函数，添加具体的自定义场景操作，并调用渲染器基类 osgViewer::ViewerBase 对象或者其派生类对象的 addUpdateOperation（osg::Operation*）函数添加到渲染器中。OSG 渲染器会将 osg::Operation 派生类对象记录保存在队列中，并在每一帧遍历的更新遍历前，将当前队列中的所有 osg::Operation 派生类对象弹出并执行这些对象的自定义操作。因为是在更新遍历前就把当前所要做的更改执行完了，所以不会出现更新操作与渲染操作的冲突。第二种方法是 OSG 作者提倡的一种简单而且肯定不会出现冲突的方法。

在操作 osgEarth 场景时，使用上述第一种方法是无法完全避免冲突的，所以本功能模块使用的是第二种方法。

2. 类描述

HsOsg::StaticOSGAssistant 静态辅助类

HsOsg::StaticOSGAssistant
-m_pViewerBase : osgViewer::ViewerBase* -m_ViewerBaseMutex : OpenThreads::ReadWriteMutex
+setViewerBase(in pViewerBase : osgViewer::ViewerBase*) : void +dealWithOperation(in refOperation : osg::ref_ptr< osg::Operation >) : void

图 4-1 HsOsg::StaticOSGAssistant 类图

HsOsg::StaticOSGAssistant 是一个静态类，包含着静态数据成员和静态函数成员。m_pViewerBase 指针指向场景渲染器，这个属性可以被函数 setViewerBase 设置；m_pViewerBaseMutex 为多线程下访问 m_pViewerBase 指针的读写锁；dealWithOperation 函数接收自定义操作的 osg::Operation 派生类对象，并不冲突场景渲染地执行所添加的 osg::Operation 派生类对象中的自定义操作。

3. 执行流程

当用户将含有自定义操作的 osg::Operation 派生类对象传入 StaticOSGAssistant::dealWithOperation(osg::ref_ptr< osg::Operation >)中时，函数会判断是否已经指定了渲染器，如果指定了渲染器则添加到渲染器中，如果没指定渲染器，则直接执行

该 osg::Operation 派生类对象的操作。

§4.3 HS_Threads 链接库

4.3.1 OpenThreads 简介

OpenThreads 是一个用 C++编写的面向对象、跨平台的线程库，它是免费开源的。OpenThreads 隐藏了与平台相关的代码，定义了与平台无关的线程定义和操作类。OpenThreads 结构简单明了，开发过程简单高效。OpenThreads::Threads 和 OpenThreads::Mutex 是 OpenThreads 最基本的类。

OpenThreads::Thread，线程类，一个 Thread 对象代表一个线程。Thread 具有基本的线程开始接口 start 和线程结束接口 cancel，可通过重写 run 函数自定义线程运行时的操作，通过重写 cancel 函数自定义线程结束操作。

OpenThreads::Mutex，互斥类，用于同步多线程下对资源的操作。OpenThreads::Mutex 具有加锁函数 lock 和解锁函数 unlock，其子类 OpenThreads::ReadWriteMutex 派生出了分别对读、写操作的不同加锁和解锁函数。

另外 OpenThreads 还提供了满足条件才会允许线程运行的 OpenThreads::Condition 类、阻塞线程运行的 OpenThreads::Block 类和控制多线程同时到达的 OpenThreads::Barrier 类等线程操作类。

从 OSG 的 2.x 版本开始，OpenThreads 被集合到了 OSG 中，成为了 OSG 基本库之一。在 OSG 中，OSG 使用 OpenThreads 进行线程管理，同时 OpenThreads 也使用了 OSG 的内存管理技术，两者相互依存。

4.3.2 项目范围

利用 OpenThreads 线程库，HS_Threads 设计出更高级的多线程应用场景，扩充 OpenThreads 线程库的功能。

HS_Threads 中细化了线程运行与线程中执行的操作，抽象出更高级的线程类和任务类，并设计出任务管理类和线程管理类对多线程下的线程、任务等资源进行管理。

4.3.3 任务与工作线程

1. 功能介绍

OpenThreads 中，OpenThreads::Threads 对象在执行完 run 函数的代码后线程对象会进入停止运行的状态，如果你有一定数量的不同操作需要动态被多线程环境执行，就需要为每一类型的操作定义一个 OpenThreads::Threads 的派生类。使用这样方式不易于线程管理，而且如果一个进程启用了太多的线程，会消耗大量计算机时间片的同时也不会带来更大的速度提升。

HS_Threads 库根据实际项目开发需求，扩展了 OpenThreads::Thread 的应用范围，允许

OpenThreads::Thread 按照要求不断执行不同的线程任务。

2. 结构设计

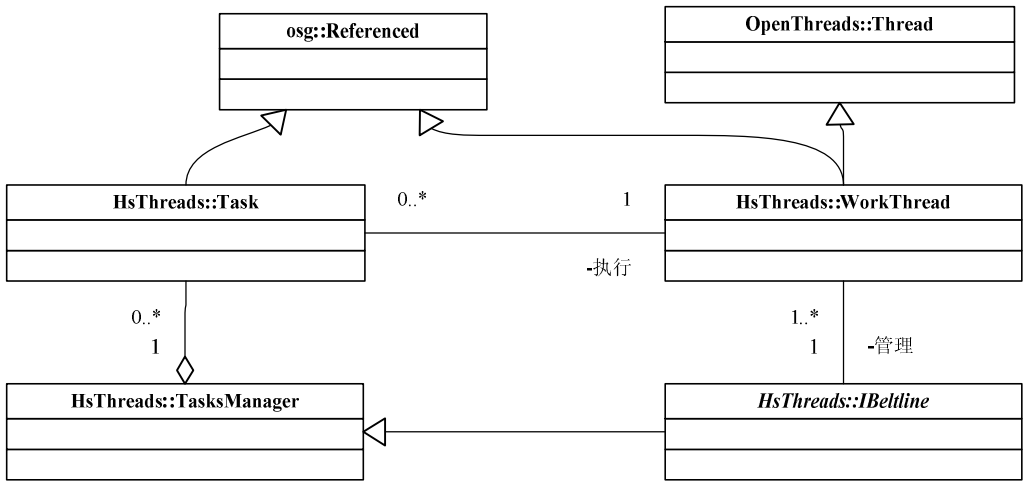

图 4-2 任务与工作线程结构图

HS_Threads 中，将需要运行在线程中的操作定义成任务类 HsThreads::Task，将执行 HsThreads::Task 对象中操作的线程定义为工作线程类 HsThreads::WorkThread，一个 HsThreads::WorkThread 对象可以顺序执行 0 到多个指定的 HsThreads::Task 对象中的操作。HsThreads::Task 和 HsThreads::WorkThread 都继承于 osg::Referenced，适用于不同线程不断执行不同的任务，自动释放任务对象和线程对象。

HS_Threads 中，HsThread::TasksManager 负责 HsThreads::Task 资源的管理，继承自 HsThreads::TasksManager 的生产线接口 HsThreads::IBeltline 扩充了对一到多条工作线程 HsThreads::WorkThread 的管理接口。一条生产线 HsThreads::IBeltline 具备一到多条工作线程 WorkThread 按需执行多个任务 HsThreads::Task 的能力。HsThreads::IBeltline 的实现类管理着 HsThreads::Task 任务资源和 HsThreads::WorkThread 工作线程资源，并保证两者有效正确运行。

3. 主要接口及功能描述

1) 任务类 HsThreads::Task

HsThreads::Task::TaskState 是 HsThreads::Task 的内部镶嵌枚举类型，代表着当前任务是否加入到任务队列中。HsThreads::Task 类拥有控制线程读写锁对象 m_TaskMutex 和任务状态对象 m_TaskState，成员函数 doTask 是一个纯虚函数，具体的 HsThreads::Task 派生类需要实现该虚函数，添加自定义的线程任务操作。

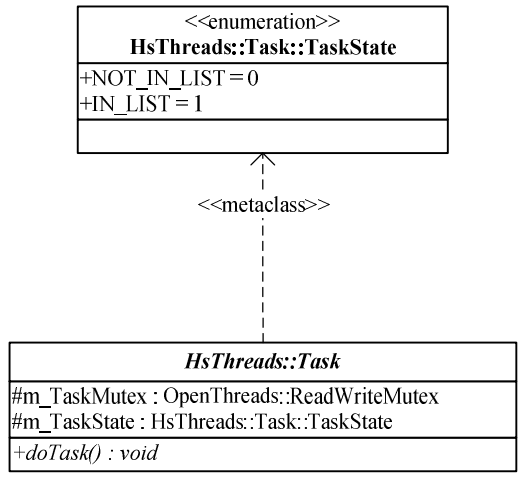

图 4-3 HsThreads::Task 类图

2) 工作线程类 HsThreads::WorkThread

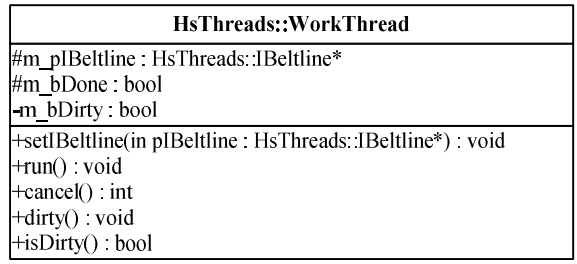

图 4-4 HsThreads::WorkThread 类图

HsThreads::WorkThread 保存着管理该对象的 HsThreads::IBeltline 对象指针 m_pIBeltline，m_pIBeltline 可以通过 setIBeltline 函数进行设置。run 和 cancel 成员函数派生于 OpenThreads::Thread，run 内部可以循环不断执行来自 m_pIBeltline 的任务队列中任务，当执行任务队列中的任务，或者调用 cancel 函数时，run 函数才会退出。当管理 HsThreads::WorkThread 对象的 m_pIBeltline 不再对该对象进行管理时，会将 m_bDirty 设置为 true，标识该工作线程处于空闲待销毁的状态。

3) 任务管理类 HsThreads::TasksManager

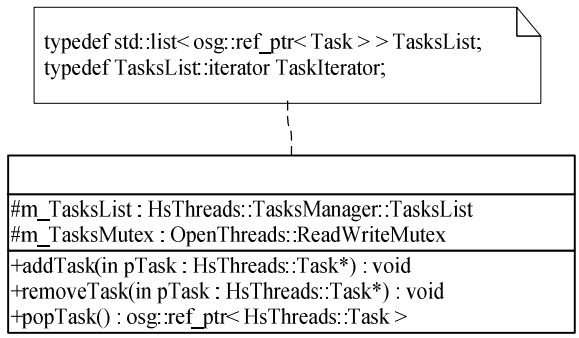

图 4-5 HsThreads::TasksManager 类图

HsThreads::TasksManager 内部拥有一个 HsThreads::Task 对象的任务列表，通过调用 addTask 往列表中添加对象，调用 removeTask 从列表中删除某个对象，调用 popTask 从列表头部弹出一个最早进入任务列表的任务。

4) 生产流水线接口 HsThreads::IBeltline

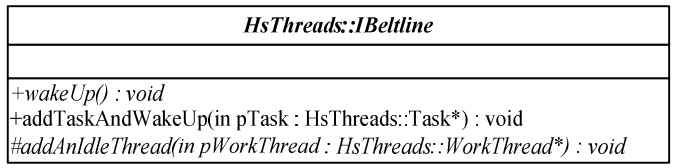

图 4-6 HsThreads::IBeltline 类图

HsThreads::IBeltline 继承于 HsThreads::TasksManager，具备管理任务的能力的同时，也提供了管理工作线程的能力，HsThreads::IBeltline 的实现类可以管理 1 到多个工作线程 HsThreads::WorkThread 对象用于处理内部任务队列中的任务。

wakeUp 函数用于唤醒流水线的工作线程开始工作，addTaskAndWakeUp 是往任务队列中添加一个任务后唤醒工作线程开始工作。addAnIdleThread 是 protected 的成员函数，当流水线内部的工作线程完成工作任务后处于空闲状态，内部工作线程会调用此函数通知流水线，以便流水线对空闲工作线程做安排。

4.3.4 单工作线程多任务流水线

1. 功能介绍

HsThreads::SingleThreadBeltline 是 HsThreads::IBeltline 的实现类，HsThreads::SingleThreadBeltline 是只有一条工作线程的生产线。HsThreads::SingleThreadBeltline 内部管着任务队列和一条工作线程，所有加入任务队列的任务会被这条工作线程顺序执行，直到任务队列为空或者人为终止。

2. 结构设计

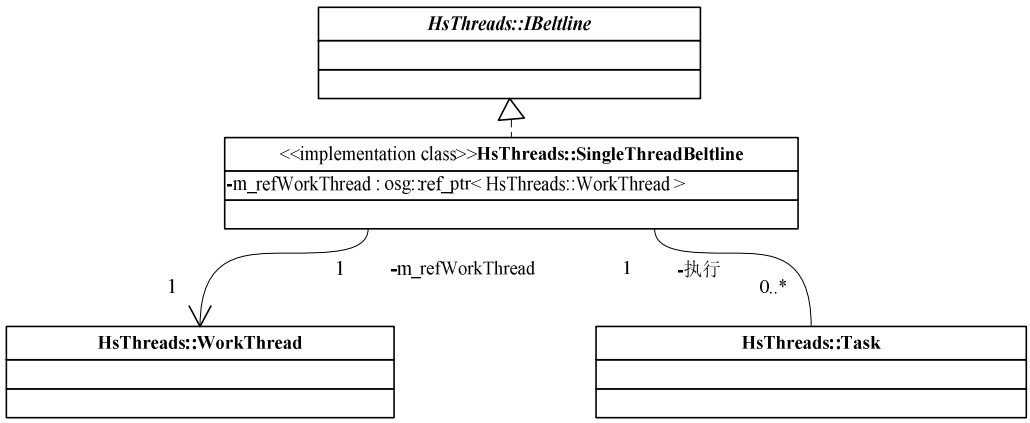

图 4-7 HsThreads::SingleThreadBeltline 类关系图

HsThreads::SingleThreadBeltline 继承自 HsThreads::IBeltline 抽象类，拥有一条工作线程 HsThreads::WorkThread，能使用内部工作线程顺序执行 0 到多个 HsThreads::Task 任务。

4.3.5 多工作线程多任务线程池

1. 功能介绍

HsThreads::ThreadPool 是 HsThreads::IBeltline 的实现类，HsThreads::ThreadPool 是多工作线程多任务的线程池类。HsThreads::ThreadPool 内部管理着工作线程列表和任务队列，允许多条工作线程一起执行任务队列中的任务，直至任务队列为空或者人为终止。

2. 结构设计

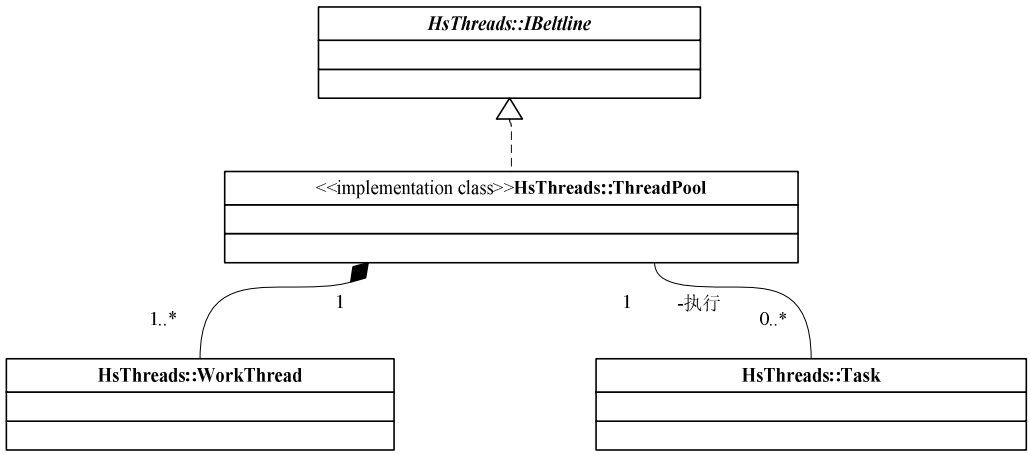

图 4-8 HsThreads::ThreadPool 类关系图

HsThreads::ThreadPool 继承于 HsThreads::IBeltline，内部可动态增加、减少 HsThreads::WorkThread 数目，可执行 0 到多个 HsThreads::Task。

3. 主要接口及功能描述

1) 线程池类 HsThreads::ThreadPool

HsThreads::ThreadPool 实现了基类 HsThreads::IBeltline 的纯虚函数，HsThreads::ThreadPool 内部拥有一个任务列表的同时，还拥有一个工作线程列表 m_ThreadRefList，保存着 1 到多个工作线程。使用 setMaxThreadsNum 函数动态伸缩 m_ThreadRefList 的容量，并将数目记录到 m_uiMaxThreadsNum，并在 wakeUp 的时候动态增加容量范围内的工作线程数，可动态根据任务数增加适当数量的工作线程，运行时，工作线程数为 1，最大线程数为 m_uiMaxThreadsNum。当一个工作线程处于空闲状态，可调用保护型成员函数 addAnIdleThread 将此工作线程对象加入到 m_IdleThreadPtrsQueue。getAnIdleThread 允许从 m_IdleThreadPtrsQueue 空闲线程队列中获取队头的一条空闲队列。

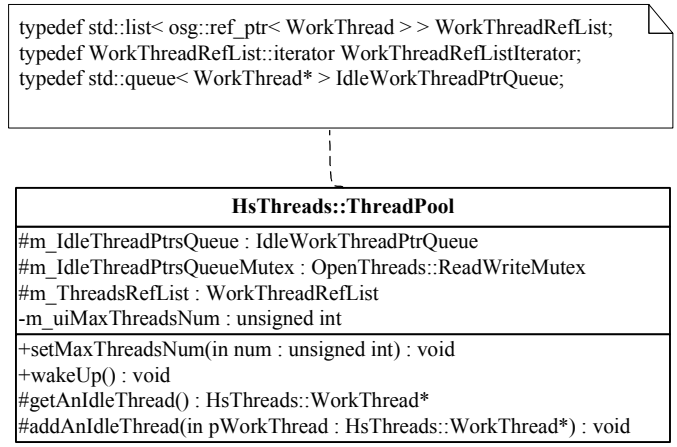

图 4-9 HsThreads::ThreadPool 类图

4.3.6 任务顺序纠正

1. 功能介绍

在多工作线程执行多种任务的情况下，本来是顺序添加到任务队列中的任务，会先后被不同的线程从队列中抽取出来进行处理，由于执行不同任务所需的时间是不一样的，所以这些任务被执行完成的时间点也是不一样的，原先的顺序也就被打乱了。用赛跑来比喻，赛道上的运动员被顺序编号的，比赛开始后，最终到达终点的运动员的编号就不一定是顺序编号的。

有一类需求，就是要求顺序进入任务队列的任务，执行完成的时候，能够按照之前的顺序插入排队，这样就保证了在所有的任务在完成后也保持之前的顺序信息，拿上述赛跑示例来说，要求先到达的运动员按照顺序大小排列好。例如，要加载一组不同的影像图层数据，每份数据的读取都在一个任务中加载，这些人物顺序加入到线程池 HsThreads::ThreadPool 的任务队列中，要求到达的时候能够按照之前的顺序组织起来，正确显示到三维地图上。

HsThreads 中的顺序纠正器模块就是为了满足上述需求的。在 HsThreads 中，使用 HsThreads::Order Holder 记录分配的顺序和到达（完成）信息，使用 HsThreads::OrderHoldersContainer 模板纠正 HsThreads::OrderHolder 的次序。需要纠正次序的某类任务，只需继承 HsThreads::OrderHolder 和 HsThreads::Task，然后构建该类的 HsThreads::OrderHoldersContainer 的模板实现类对象，并调用适当函数来管理此类任务即可。

2. 顺序纠正设计

派生自 HsThreads::OrderHolder 和 HsThreads::Task 的任务类对象，由该类的 HsThreads::OrderHoldersContainer 顺序纠正器模板实现类对象保存在 DynamicVector 数据成员 m_Original 中并分配顺序号，当任务类对象被 HsThreads::WorkThread 对象执行完成后，任务类对象会通知顺序纠正器进行顺序纠正，通过在已到达的任务列表 m_pArrival 中比较原先分配的序号进行插入，这样保证了所有需要顺序纠正的任务对象在到达后一样可以保持原先顺序。

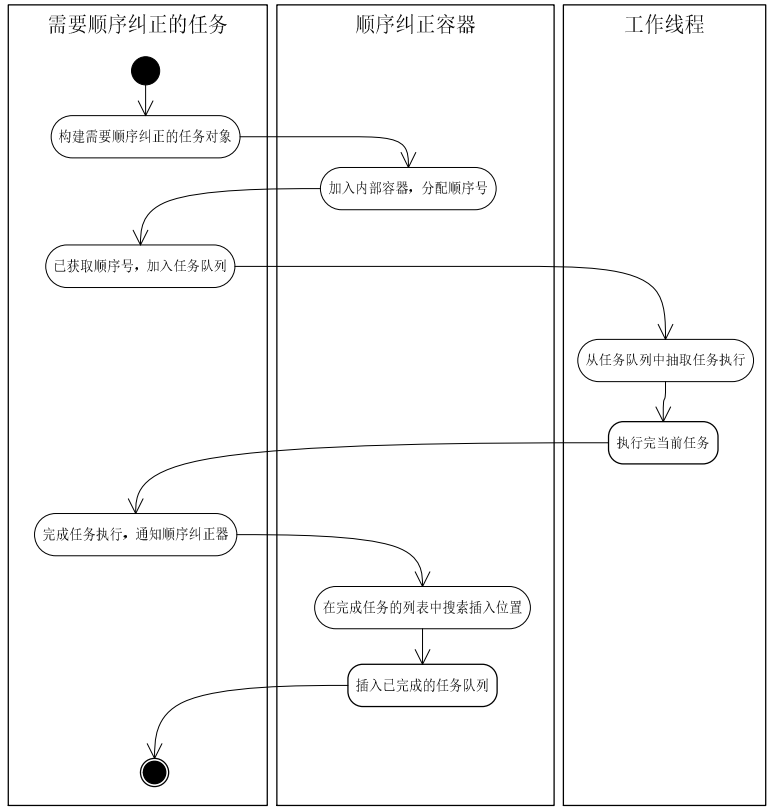

图 4-10 顺序纠正活动图

除了为 HsThreads::OrderHolder 分配顺序和调整到达顺序，还要考虑对顺序纠正器内部的对象的插入、移动、删除等操作所造成的次序混乱。当顺序纠正器内部对象的顺序结构发生改变，需要重新调整 m_Original 中对象顺序和 m_pArrival 中对象顺序。下面是当发生 move 移动操作时所发生的活动。

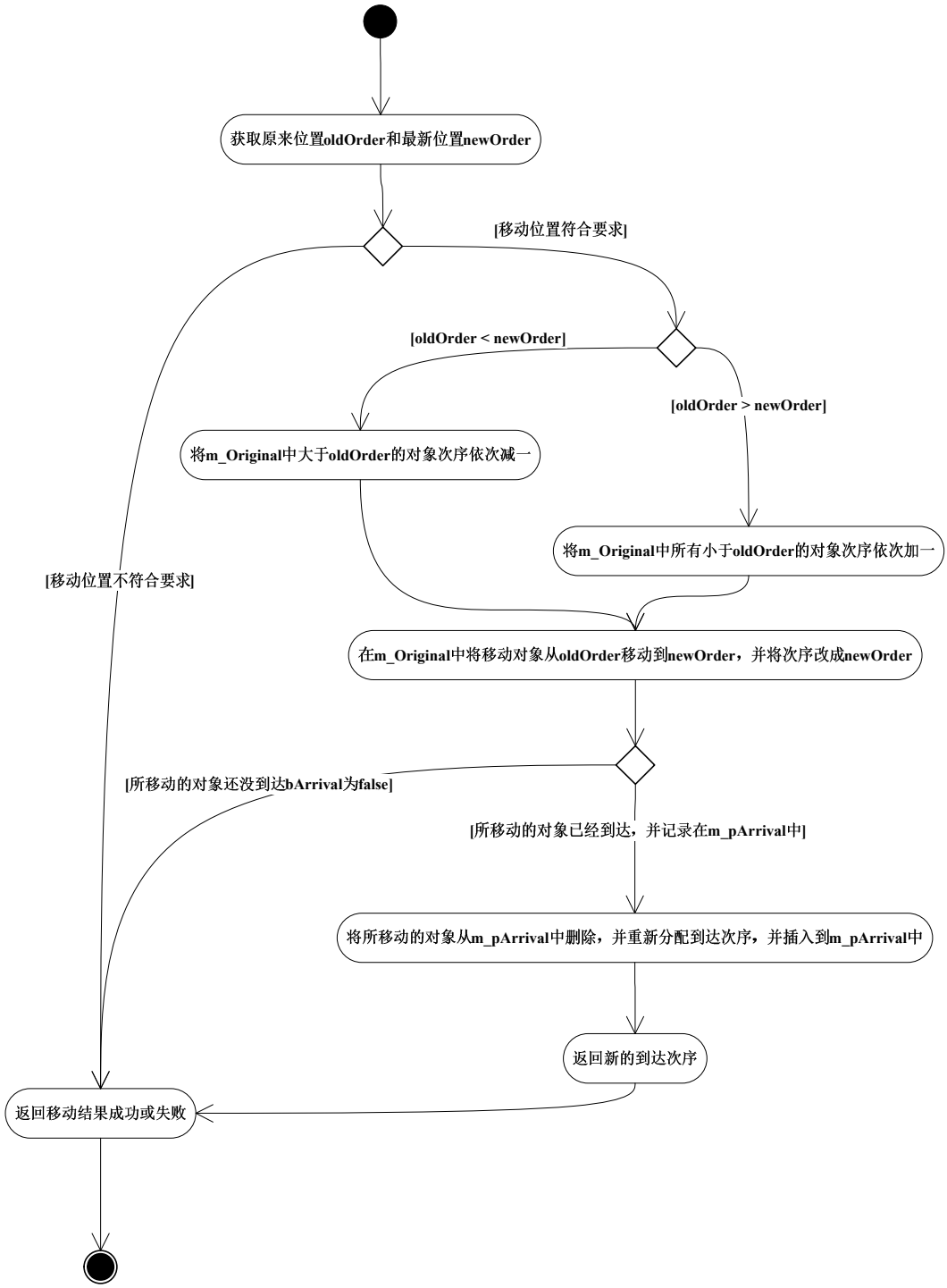

图 4-11 改变顺序活动图

3. 主要接口及功能描述

1) 顺序保持者类 OrderHolder

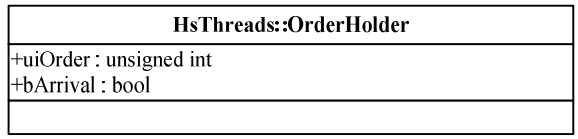

图 4-12 HsThreads::OrderHolder 类图

HsThreads::OrderHolder 是一个记录分配顺序和是否已经到达的类。该类及其派生类的对象可以被顺序纠正器分配顺序，当 bArrival 为 true，意味着此 HsThreads::OrderHolder 对象可以插入到顺序纠正器中的已经完成任务的列表中。

2) 顺序纠正容器模板 OrderHoldersContainer

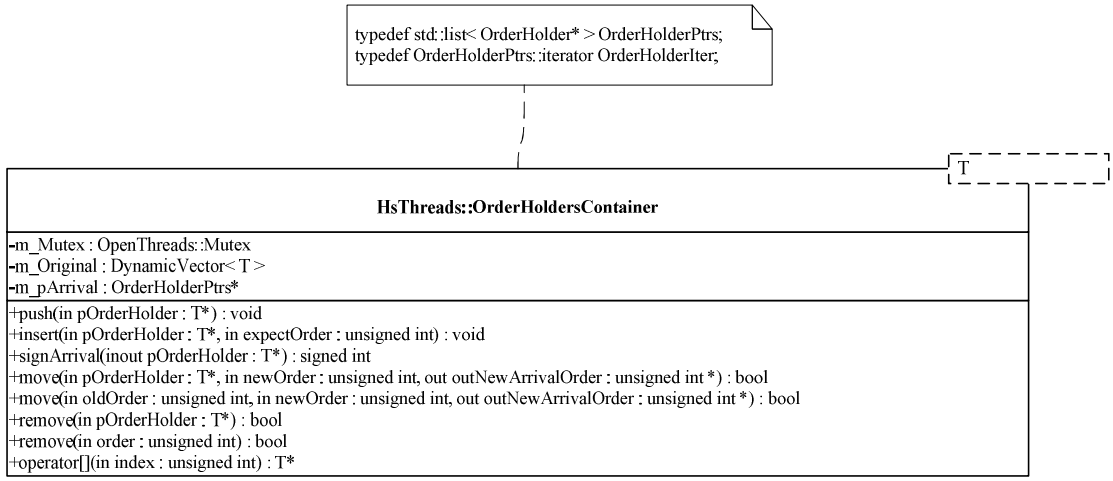

图 4-13 HsThreads::OrderHoldersContainer 类图

顺序纠正器 HsThreads::OrderHoldersContainer 的模板参数 T 为 HsThreads::OrderHolder 及其派生类，顺序纠正器内部拥有的数据成员有：保存 HsThreads::OrderHolder 及其派生类对象的动态矢量数组 m_Original，用于指向已完成任务的列表 m_pArrival，以及控制 HsThreads::OrderHoldersContainer 对象内部状态的线程锁 m_Mutex。

顺序纠正器通过 push 函数将 T 对象保存到内部动态矢量数组并为其分配顺序，或者使用 insert 将 T 对象插入到你想要插进的顺序中。加入到顺序纠正器里的 T 对象，不管是否还没被执行、正在被执行还是已经被执行完，都可以调用 move 方法改变次序，或者调用 remove 方法从顺序纠正器中删除，这些操作都不会影响在顺序纠正器中的 T 对象顺序排列。signArrival 函数的作用是，当顺序纠正器内的一个 T 对象已经被完成，调用 signArrival 并传入该 T 对象可返回一个当前 T 对象在 m_pArrival 列表中的序号。

§4.4 OSG、osgEarth 源码修改

4.4.1 源码修改需求说明

在项目开发过程中，会遇到一些错误和问题，需要修改 OSG、osgEarth 源码以适应项目开发需求。修改过后的源码与原来接口兼容，并增加了一些额外的功能、接口，修改了部分设计不足的地方。

4.4.2 OSG 源码修改

1) 引用计数函数重写

OSG 中的 osg::Referenced 类的引用函数 ref()和解除引用函数 unref()不是虚函数，osg::Referenced 派生类无法重写这两个函数。这会给派生类的引用调试带来麻烦，无法就未合理引用的 osg::Referenced 派生类对象进行跟踪。所以为上述两个函数添加 virtual 关键字，允许 osg::Referenced 派生类重写这两个函数，添加自定义代码。

2) osg::Object 虚继承 osg::Referenced

osg::ref_ptr< T >智能指针无法指向多次非虚继承于 osg::Referenced 的派生类 T，因为 T 的内存结构里有多份不同 osg::Referenced 基类的内存块，osg::ref_ptr< T >会因指向不明确而报错。如果 T 是多次虚继承于 osg::Referenced，osg::Referenced 的内存块始终只有一份，osg::ref_ptr< T >是有效的。

osg::Object 类公有继承（public）于 osg::Referenced 类，osg::Object 类是 OSG 中的基础类，大多数类都继承自 osg::Object。由于 osg::Object 没有虚继承于 osg::Referenced，限制了 osg::Object 的派生方式，无法满足派生类继承于 osg::Object 的同时继承其他 osg::Referenced 的派生类。通过将 osg::Object 从 osg::Referenced 的继承方式修改为虚公有继承（virtual public），解除 osg::Object 的派生限制。

4.4.3 osgEarth 源码修改

1) MP 地形引擎瓦片缓存清除

osgEarth 的 MP 引擎节点 osgEarth::Drivers::MPTerrainEngine::MPTerrainEngineNode 对象在加载瓦片数据的时候会建立缓存四叉树，并根据当前的 LOD 信息遍历四叉树进行瓦片合成并渲染到三维地图上（见第 2.2.5 节、第 2.2.6 节）。但是当前版本的 osgEarth 的 MP 地形引擎不会在地图模型发生更改后，调整四叉树的内部数据，例如，当从地图 osgEarth::Map 对象中删除一个图层，即使它不会再被渲染出来了，但是这个图层依然被 MP 地形引擎的四叉树每一层 LOD 的节点以及其他 osgEarth::MapFrame 对象引用着，导致这个图层引用计数不正确，不能立即自动释放，只能等到 MP 地形引擎节点析构时才会被释放。

在 MP 引擎的地形模型发生改变的时候，会调用刷新函数 refresh()，并重新创建地形 createTerrain()，通过在创建新地形过程中，对已经删除了的图层的瓦片进行清除，并更新版

本号与 osgEarth::Map 不一致的 osgEarth::MapFrame 对象。调试过程需要用到上一节中所提到的引用计数输出来监测图层对象的引用情况，通过对 MP 引擎的瓦片进行更新清除和 osgEarth::MapFrame 的更新同步，在删除图层的时候，引用计数会减少到正确的引用数。

2) MP 地形引擎异步创建地形节点

MP 地形引擎节点 osgEarth::Drivers::MPTerrainEngine::MPTerrainEngineNode 会在第一次初始化的时候或者地形模型发生改变的时候创建新地形节点 osgEarth::Drivers::MPTerrainEngine::TerrainNode 对象，创建地形节点的过程是同步操作的，会阻塞渲染线程。当往 osgEarth::Map 对象添加瓦片类的图层时，MP 地形引擎会消耗很长时间用于重新创建地形节点并阻塞渲染线程，严重影响其它操作。

在 MP 地形引擎源码项目中添加 HS_OSG 以及 HS_Threads 的链接库引用，为其设计派生自 HsThreads::Task 的用于创建地形节点的多线程任务 osgEarth::Drivers::MPTerrainEngine::RefreshTerrainTask 和派生自 osg::Operation 的用于将新地形节点添加到 MP 地形引擎节点中去的渲染器自定义操作 osgEarth::Drivers::MPTerrainEngine::RefreshTerrainOperation，为 MP 地形引擎节点添加 HsThreads::SingleThreadBeltline 对象。当地形模型发生更改并需要重建地形节点时，创建 osgEarth::Drivers::MPTerrainEngine::RefreshTerrainTask 对象，并添加到 HsThreads::SingleThreadBeltline 数据成员中进行执行，当在线程中创建地形节点完成后，创建 osgEarth::Drivers::MPTerrainEngine::RefreshTerrainOperation 对象并加入到 HsOsg::StaticOSGAssistant 中执行，最终完成将新地形节点添加到地形引擎节点中去，整个过程阻塞渲染线程的时间片非常小。

3) osgEarth 对象延迟初始化

osgEarth 大部分地图场景对象，例如图层、天空节点、标注节点等，在构造的时候传入参数选项集合 osgEarth::ConfigOptions 或者其派生类对象进行初始化，而且这种初始化是必须的而且仅此一次，有的初始化过程会消耗很长时间，阻塞渲染线程。通过修改或者新增构造函数、新增初始化函数、修改所继承类中的初始化过程，允许上述类在构造对象后再初始化内部。因为使用 osgEarth 对象延迟初始化，使得开发人员能够将对象初始化过程放在多线程环境中执行，并在初始化完成后正确显示出来。

4) 更改不合理的类设计

osgEarth 内部存在着一些设计不合理的代码，不影响运行，但是会有隐患。拿 osg::Referenced 的继承来说，osgEarth 内部部分继承于 osg::Referenced 的类，例如各种图层类，违背了 osg::Referenced 的使用原则（见第 2.1.3 节），将类虚析构函数声明为了 virtual public，应将源码中这些类的析构函数声明为 virtual protected。

§4.5 HS_Earth_Core 链接库开发

4.5.1 项目范围

HS_Earth_Core 封装底层 osgEarth 的功能和接口，扩展 osgEarth 的功能，定义新使用逻辑，简化调用接口。HS_Earth_Core 是三维数字地球核心层中非常重要的一个库，其他更高

逻辑的组件基于 HS_Earth_Core 库进行构建。

4.5.2 项目架构

HS_Earth_Core 主要由渲染器，地图以及场景物体提供类组成，这些组成部分封装 osgEarth 的功能和接口并扩展新的功能和接口的同时，支持回调机制，允许监听内部事件状态。

4.5.3 渲染器

1. 功能介绍

HS_Earth_Core 内部设置了多种不同功能、支持不同平台的渲染器，因为这些渲染器不是直接绑定内部场景节点的，所以可以根据不同需求选择不同的渲染器来渲染三维地图场景。HS_Earth_Core 项目利用 CMake 构建系统进行项目管理（见第 4.7 节），HS_Earth_Core 中的 CMake 项目构建脚本可以根据不同平台、不同环境将相应渲染器的源码与项目工程进行绑定，进而为不同平台、环境生成相应的渲染器。

2. 结构设计

HS_Earth_Core 中的渲染器是对原有 OSG 支持的渲染器进行功能的封装，使用方法更简单灵活，根据不同需求扩展了专有功能。HS_Earth_Core 中的渲染器体系主要分为两大部分，包括与底层窗口平台有关的渲染器和与平台无关、包含特定功能的特定渲染器提供者（见图 4-14）。

根据不同平台可以使用不同类型的渲染器模板，例如一般的渲染器 HsEarthCore::CommonViewerT、Windows 平台下的 HsEarthCore::Win32ViewerT、用于 Qt 窗口的 HsEarthCore::QtViewerT，这些模板继承于模板参数 T，模板参数 T 是 OSG 原有渲染器类。

与平台无关的渲染器提供者的模板类，其模板参数可以是 OSG 原有渲染器类，也可以是上述与底层平台有关的渲染器模板实现类。这些渲染器提供者对内部渲染器对象做平台无关的业务逻辑操作，增加特定的功能需求，这些并不会影响最终的渲染方式。

4.5.4 场景物体提供类

1. 功能介绍

HS_Earth_Core 中，有不同的场景物体提供类，这些场景物体提供类转换或者继承原有的 osgEarth 的场景物体类，提供不一样的加载方式或者操作接口。这些场景物体提供类对象加入到三维地图场景中后，最终还是会转换成原来 osgEarth 的场景物体类对象。

2. 结构设计

场景物体提供类大概可分为三种：

1) 派生自 osgEarth 原有物体类的物体提供类。
2) 延迟初始化的物体提供类。
3) 在多线程任务中初始化的物体提供类。

第一种主要是派生原有 osgEarth 物体类，扩展接口和功能。图 4-15 是第二种及第三种场景物体提供类的结构。

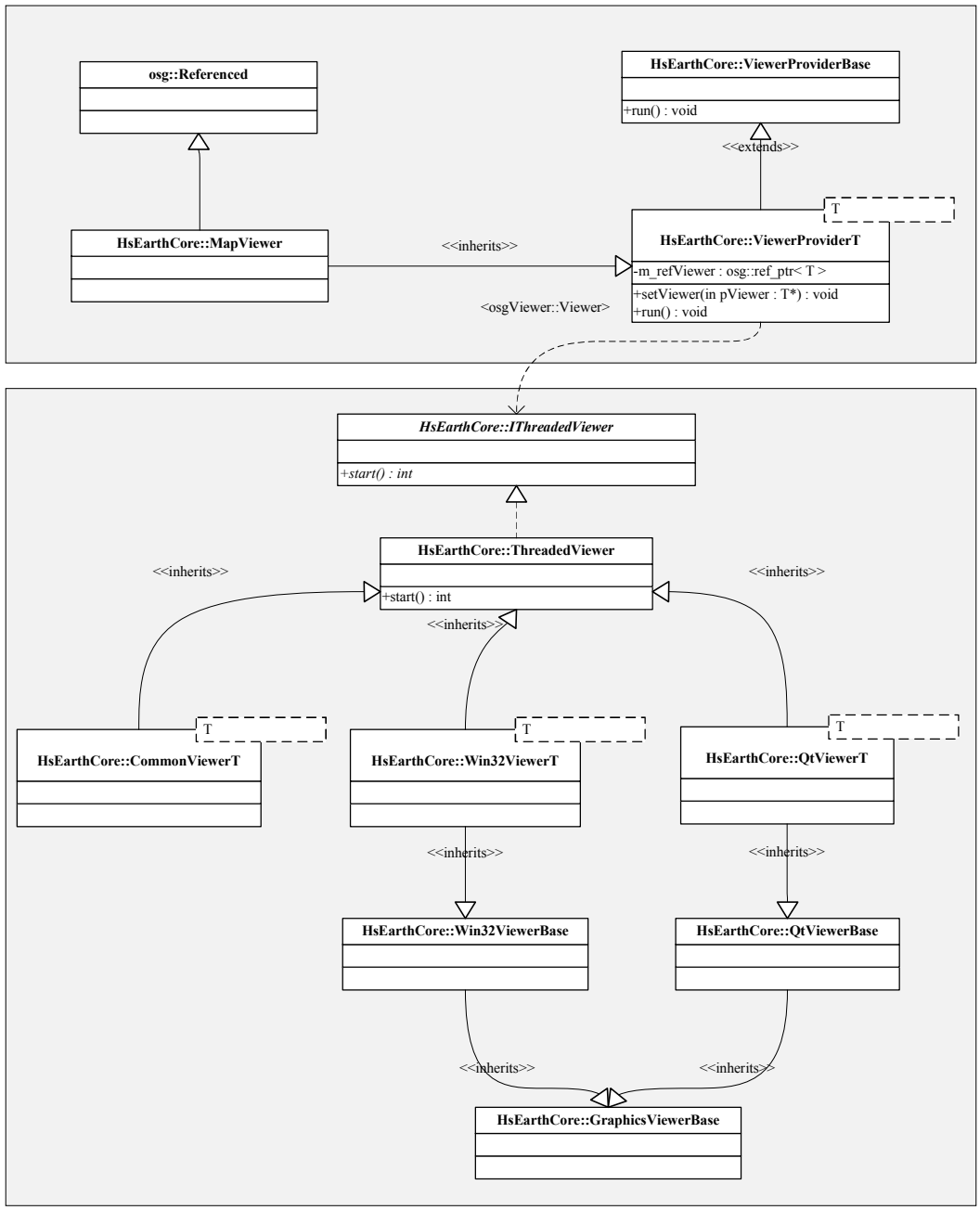

图 4-14 渲染器结构图

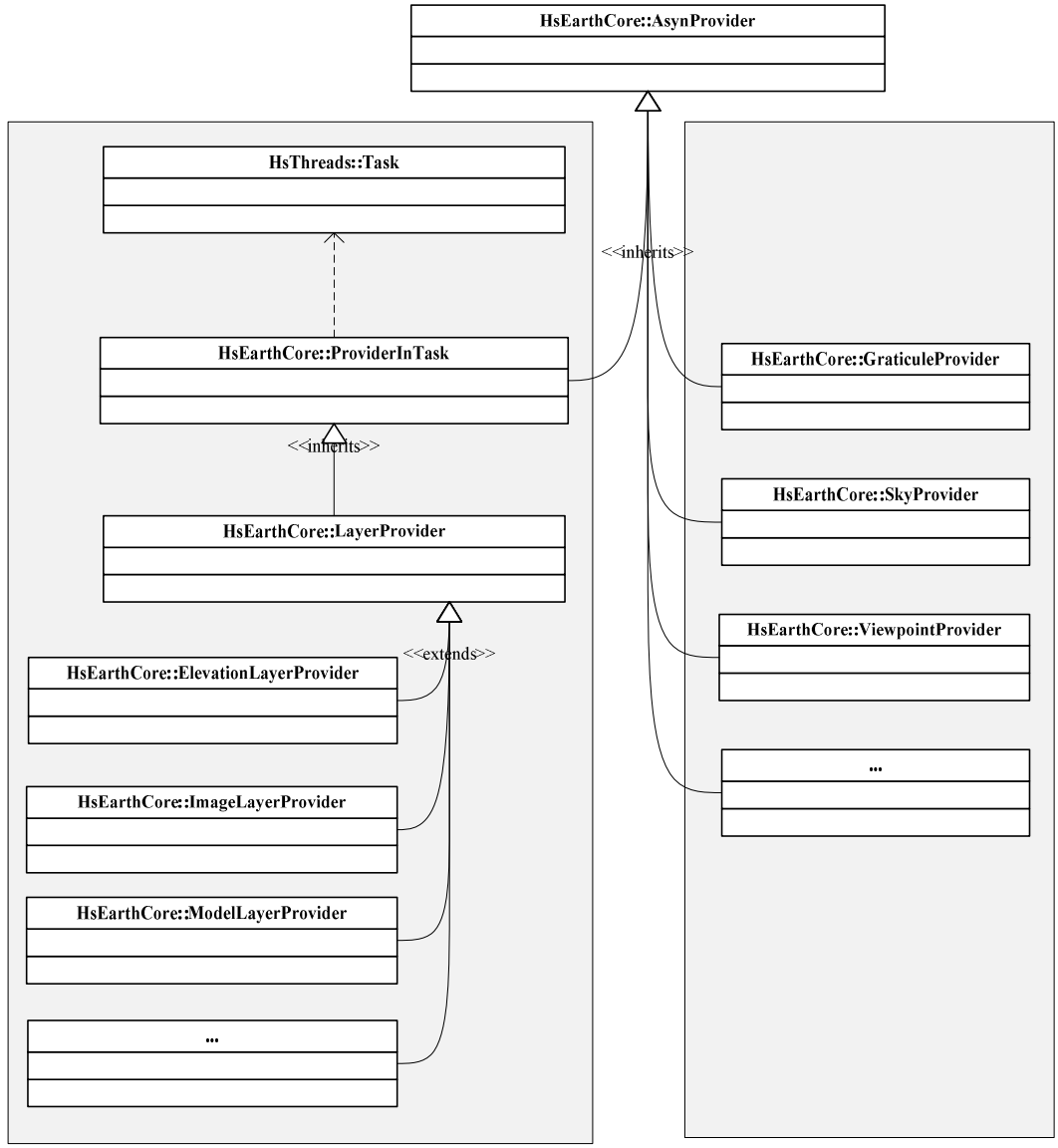

图 4-15 场景提供类结构图

4.5.5 地图

1. 功能介绍

HS_Earth_Core 中的地图是三维地图场景的管理类，地图保存着场景根节点，osgEarth 地图对象、不同类型的地图场景物体的对象容器以及用于执行多线程任务的线程池。

2. 结构设计

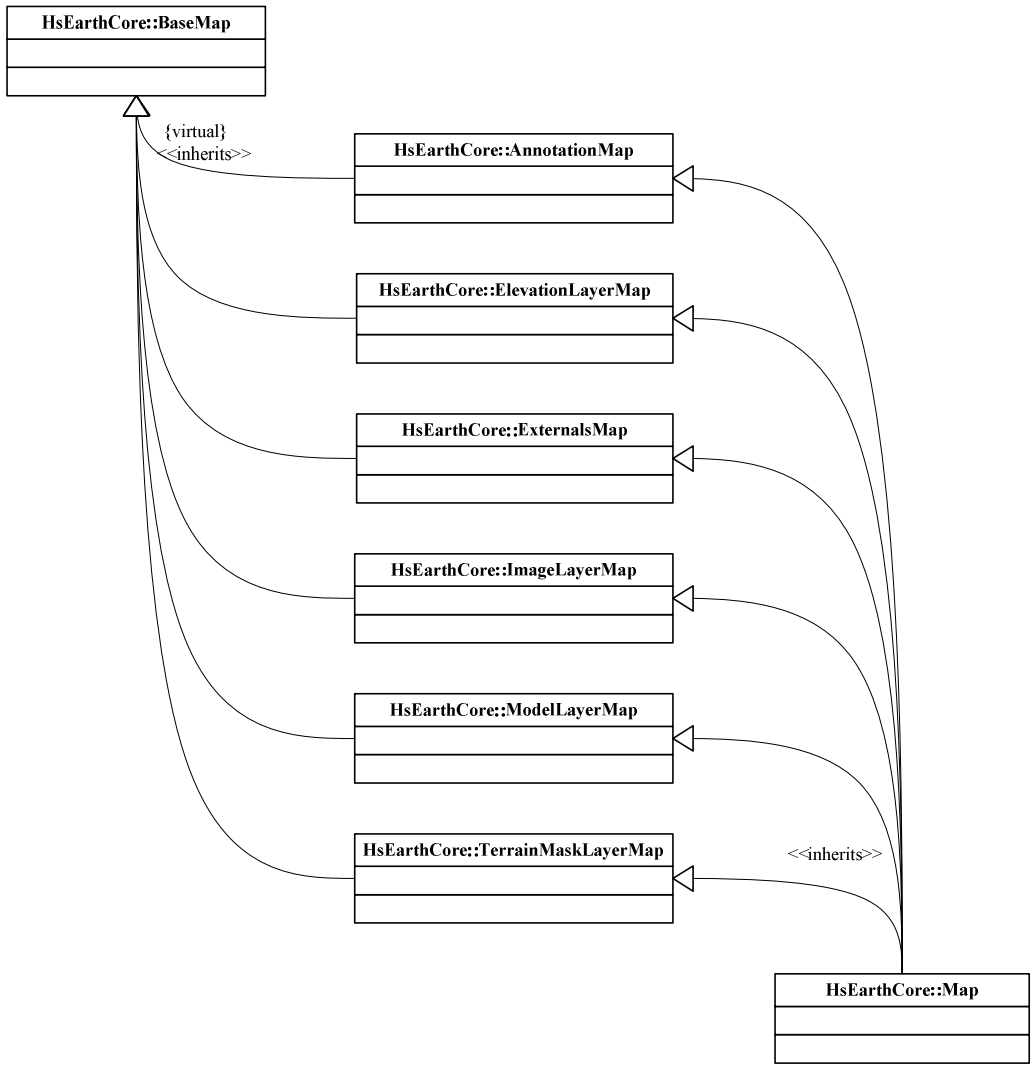

图 4-16 地图结构图

HsEarthCore::BaseMap 是地图基类，分别派生出专用于标注的 HsEarth::AnnotationMap、专用于地图外部场景的 HsEarthCore::ExternalsMap 、专用于高程图层的 HsEarthCore::ElevationLayerMap、专用于影像图层的 HsEarthCore::ImageLayerMap、专用于矢量图层的 HsEarthCore::ModelLayerMap 、专用于地形覆盖图层的 HsEarthCore::TerrainMaskLayerMap。HsEarthCore::Map 继承于上述专用图层，是功能最全的三维场景地图类型。

4.5.6 场景数据任务加载流程

1. 功能介绍

场景中的某些物体类需要读取数据进行构建对象，例如图层类对象通常需要读取文件比

较大的数据进行初始化，而原来 osgEarth 中对物体加载是需要阻塞渲染线程的，这样会严重影响其他操作。对于此类物体，HS_Earth_Core 中为其构建可在多线程任务中进行初始化的物体提供类，这些物体提供类会异步、正确地初始化，而不长时间阻塞渲染。

在多线程任务中初始化的物体提供类有相应的物体初始化任务，通过将物体初始化任务放入三维场景地图中的线程池进行多线程下的初始化操作，例如 HsEarthCore::ImageLayerProvider 的加载任务类为 HsEarthCore::ImageLayerTask。

2. 流程设计

下面用 HsEarthCore::ImageLayerMap 对象中添加 HsEarthCore::ImageLayerProvider 对象为例讲述多线程下的数据加载流程。HS_Earth_Core 在通过任务的方式加载物体时，对象之间的操作是异步的。

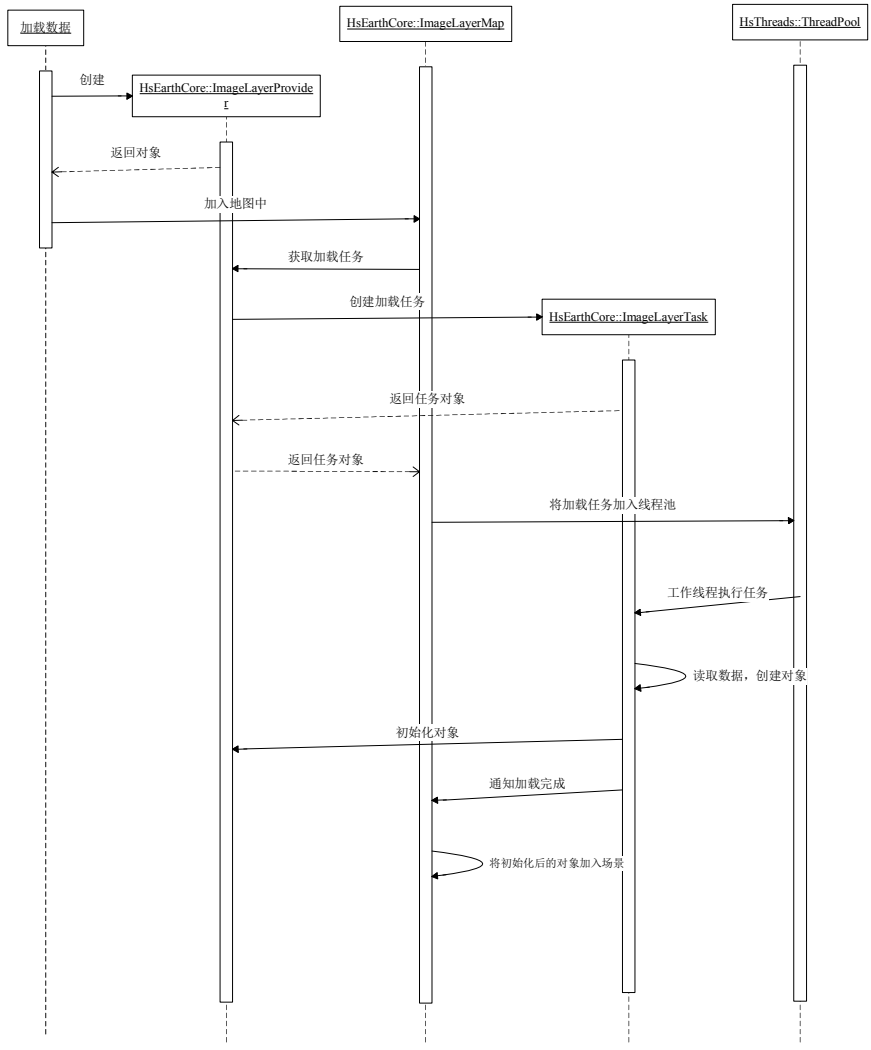

图 4-17 场景数据加载顺序图

4.5.7 回调机制

1. 功能介绍

HS_Earth_Core 支持使用回调来监听特定的类对象内部状态。回调机制在 HS_Earth_Core 更多被用于处理事件的发生、状态的改变等。

2. 结构设计

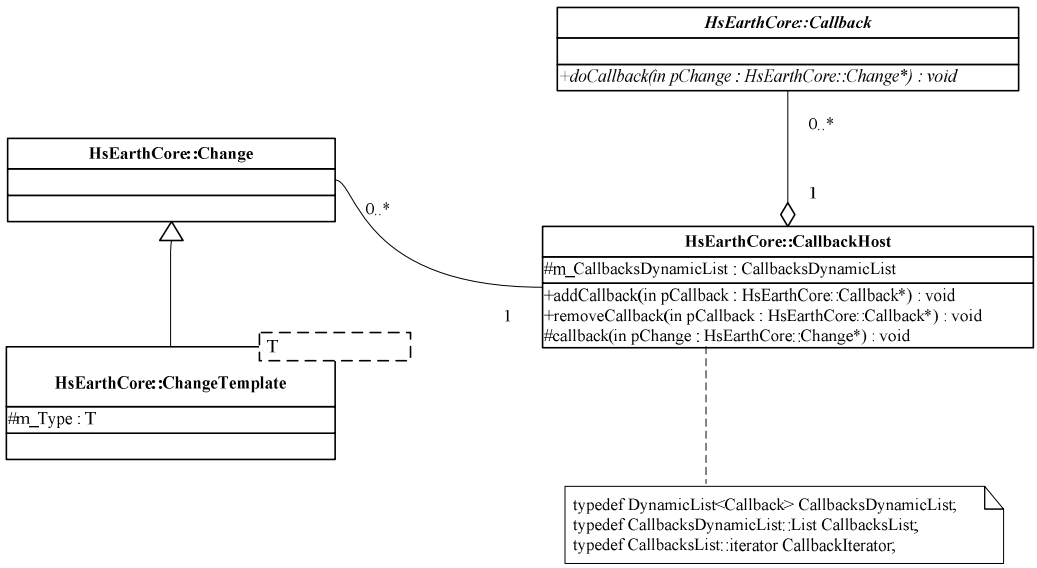

图 4-18 回调机制结构图

HsEarthCore::CallbackHost 回调宿主管理着各种需要监听该对象内部状态变化的 HsEarthCore::Callback 对调对象。HsEarthCore::Change 是代表一种状态更改类型的基类，HsEarthCore::ChangeTamplate 内部含有一个模板参数类型对象 m_Type，这个 m_Type 可以是任意类型，用于鉴别状态更改的类别和传递状态更改参数。

假设有继承于 HsEarthCore::CallbackHost 的类 Weather，Weather 有自己的状态更改类别类型 WeatherType，WeatherType 是一个含有成员 sunny、rainy、snowy 的枚举。当天气开始转变，天气变成 sunny（晴朗）时，Weather 对象 weather 会构造 HsEarthCore::ChangeTemplate< WeatherType >对象 weatherChange 传递给内部保护型成员函数 callback，callback 内部会将 weatherChange 传递给内部回调对象列表 m_CallbackDynamicList 的每一个回调的 doCallback。当人们需要知道天气状况，只需增加一个实现 HsEarthCore::Callback 的类对象，然后再内部将 HsEarthCore::Change 回调变化类型指针转换成 HsEarthCore::ChangeTemplate< WeatherType >类型指针进行操作，就可以获取天气变化信息，然后做出行程、穿衣等安排。

§4.6 HS_Earth 链接库开发

4.6.1 项目范围

HS_Earth 封装了 HS_Earth_Core，并实现了符合业务逻辑的接口，这些接口作为规范被顶层的不同的组件进行转换。

4.6.2 项目架构

HS_Earth 主要由地球总体类 Earth、场景物体、场景组组成。

4.6.3 物体类型

HS_Earth 中像图层、标注、天空等可往场景中的物体继承于 HsEarth::Object 和 HS_Earth_Core 或者 osgEarth 中对应的场景物体类，HS_Earth 目前支持以下场景物体。

表 4-1 场景物体表

序号	名称	ObjectType 枚举类型	描述
1.	HsEarth::Group	GROUP	场景组
图层			
2.	HsEarth::ImageSurfaceLayer	IMAGE_SURFACE_LAYER	影像图层
3.	HsEarth::VectorSurfaceLayer	VECTOR_SURFACE_LAYER	矢量图层
4.	HsEarth::ElevationSurfaceLayer	ELEVATION_SURFACE_LAYER	高程图层
标注			
5.	HsEarth::Circle	CIRCLE	圆形几何标注
6.	HsEarth::ImageOverlay	IMAGE_OVERLAY	图片标注
7.	HsEarth::Place	PLACE	地点标注
外部物体			
8.	HsEarth::Graticule	GRATICULE	经纬线
9.	HsEarth::Sky	SKY	天空
10.	HsEarth::Viewpoint	VIEWPOINT	视点

4.6.4 物体与组关系

1. 功能介绍

HS_Earth 中允许场景物体分组，HsEarth::Group 就是场景组，像其他场景物体一样，它也继承于 HsEarth::Object，所以 HsEarth::Group 对象除了能够添加其他场景物体，还可以添加到另一个 HsEarth::Group 对象中，进而形成符合业务需求的场景树。

2. 结构设计

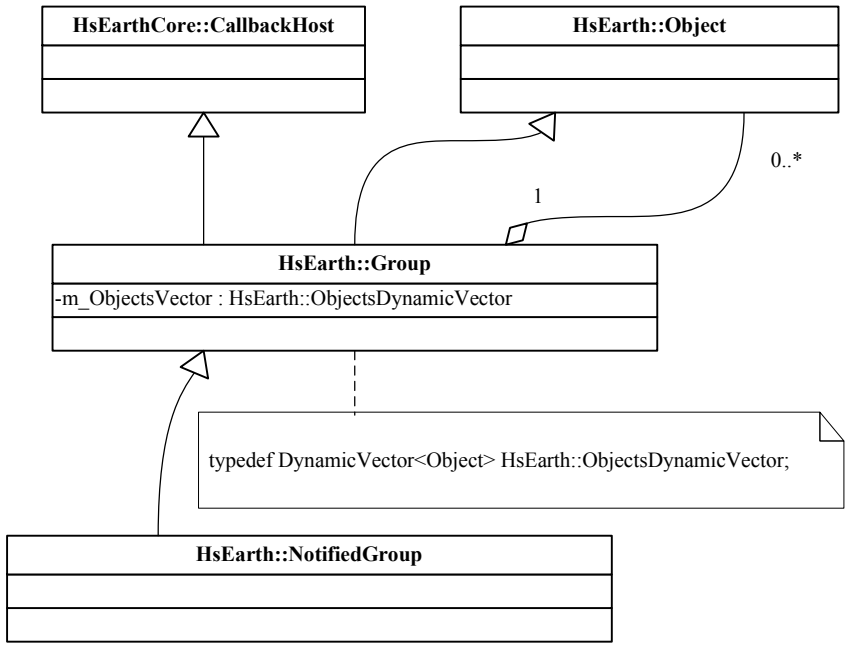

图 4-19 组关系结构图

HsEarth::Group 继承于 HsEarth::Object，内部添加 0 到多个其他 HsEarth::Object 对象。HsEarth::Group 派生出多种专有功能的场景组类，例如可监听子组内执行回调时也进行回调的 HsEarth::NotifiedGroup。

3. 主要接口及功能描述

回调通知组 HsEarth::NotifiedGroup

HsEarth::NotifiedGroup 继承于 HsEarth::Group，是一个可以能够监听子组的回调时也执行自身回调的组。

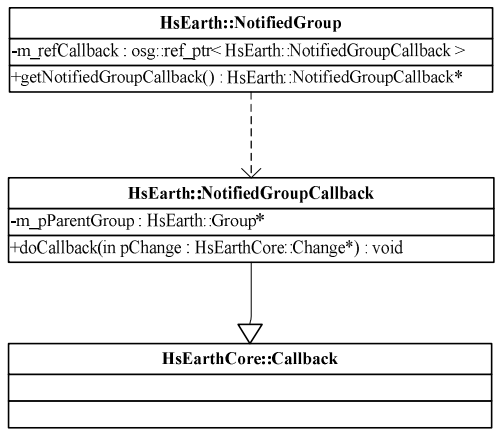

图 4-20 HsEarth:NotifiedGroup 类关系图

HsEarth::NotifiedGroup 依赖于 HsEarth::NotifiedGroupCallback 来实现功能，HsEarth::NotifiedGroup 自身的数据成员 m_refCallback 中的 m_pParentGroup 指向该 HsEarth::NotifiedGroup 对象，HsEarth::NotifiedGroup 对象将 m_refCallback 加入到子组的回调数组中，当子组执行回调时，会执行到 HsEarth::NotifiedGroup::m_refCallback，m_refCallback 内部的 doCallback 函数会执行 m_pParentGroup 中的 doCallback 函数，并传递改变类型。这样就实现 HsEarth::NotifiedGroup 对象能监听子组的回调状态。

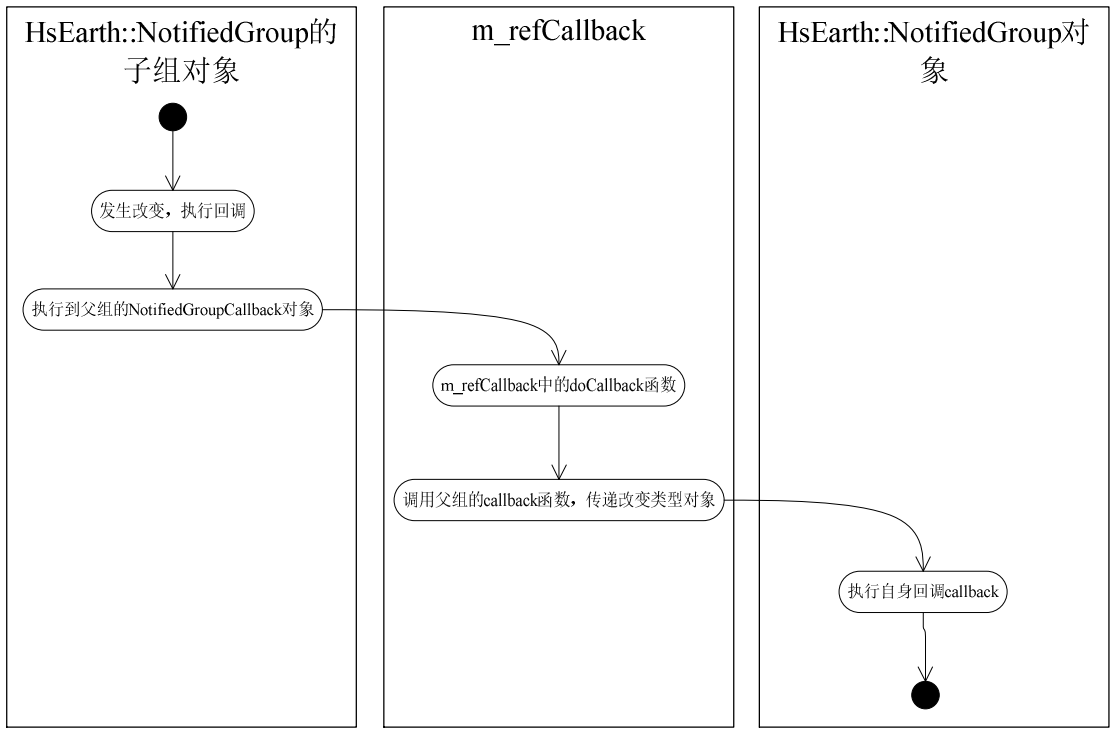

图 4-21 HsEarth::NotifiedGroup 通知活动图

4.6.5 地球总体类

1. 功能介绍

HS_Earth 中最中心的一个类是 HsEarth::Earth，它管理着整个三维地图场景，包括三维地图、渲染器和场景物体组。可以通过访问 HsEarth::Earth 对象或者其子数据成员对象进行场景设置和添加场景物体。

2. 结构设计

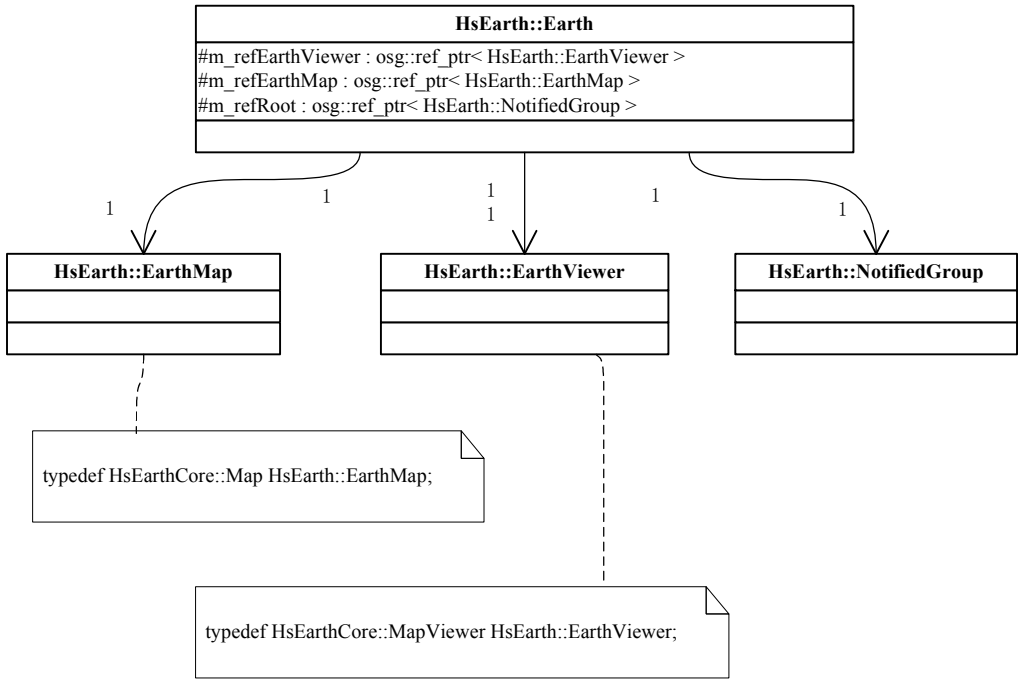

图 4-22 HsEarth::Earth 结构图

3. 主要接口及功能描述

HsEarth::Earth
#m_refEarthViewer : osg::ref_ptr< HsEarth::EarthViewer > #m_refEarthMap : osg::ref_ptr< HsEarth::EarthMap > #m_refRoot : osg::ref_ptr< HsEarth::NotifiedGroup >
+run() : void +initialize() : void +getEarthViewer() : HsEarth::EarthViewer* +getEarthMap() : HsEarth::EarthMap* +getEarthRoot() : HsEarth::NotifiedGroup* +setEarthViewer(in pEarthViewer : HsEarth::EarthViewer*) : void +setEarthMap(in pEarthMap : HsEarth::EarthMap*) : void

图 4-23 HsEarth::Earth 类图

HsEarth::Earth 中的 m_refEarthMap、m_refEarthViewer 可以通过调用相应的 get 和 set 函数进行获取、设置，也可以使用默认构建的对象。initialize 函数是将内部渲染器与场景节点进行绑定，在调用 run 函数的时候，run 函数会执行 initialize，然后将绑定后的场景在渲染器绘制出来。

4.6.6 物体处理流程

1. 功能介绍

HS_Earth 中所定义的物体关系虽然是建立在 HS_Earth、osgEarth 中所定义的物体关系之上的，但是 HS_Earth 的物体不能直接添加到场景中，像组等这些新增的关系必须经过正确处理，HS_Earth 中的场景物体才可以显示到 osgEarth 的三维地图场景中。

2. 结构设计

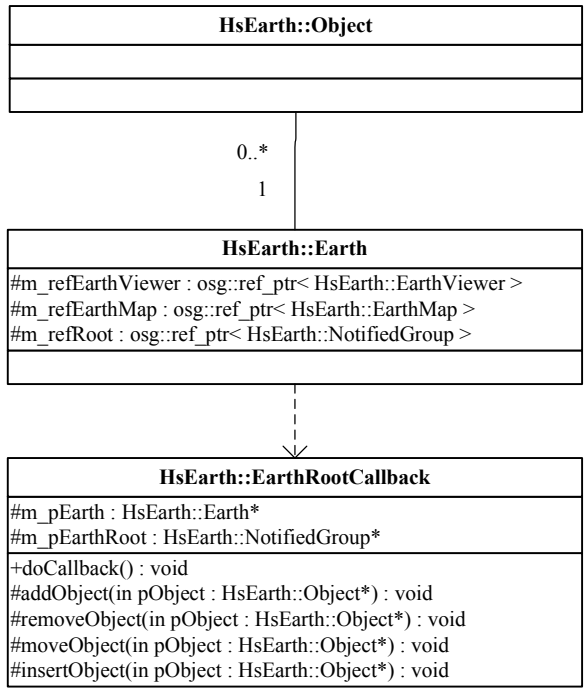

图 4-24 HsEarth::Object 与 HsEarth::Earth 类关系图

HsEarth::EarthRootCallback 对象添加到 HsEarth::Earth 对象中的 m_refEarthRoot 中，HsEarth::EarthRootCallback 对象内部指向 m_pEarth 指向该 HsEarth::Earth 对象，m_pEarthRoot 指向 m_refEarthRoot。当往 HsEarth::Earth::m_refEarthRoot 中对场景物体进行添加、插入、移动、删除等操作时，会执行回调，传入更改类型和场景物体对象，在 HsEarth::EarthRootCallback 中根据更改类型和场景物体对象，将场景物体对象正确绑定到 osgEarth 场景中，这些物体可能是绑定到 m_pEarth 指向对象的 m_refEarthMap 或者 m_refEarthViewer。

4.6.7 事件机制

1. 功能介绍

在不同的组件的实现中，事件是通过底层传递到组件顶层应用程序中去的。组件的调用

接口是被顶层应用程序主动调用的，而与组件的调用接口不同，事件是满足某个条件产生的，顶层应用程序是被动等待所触发的事件的。组件在触发事件的时候是在组件层发生的，不同的组件触发事件的方式是不一样的。为了满足相同的底层事件能够传递给不同的组件上的应用程序，HS_Earth 为需要触发事件的类设计相应的事件触发器。

2. 结构设计

事件触发器是包含着一组虚（virtual）事件函数的基类，上层组件接口只需继承这些事件触发器基类，然后派生基类中相应事件的函数，并在相应事件触发函数添加具体的组件事件触发代码。这样顶层应用程序就可以通过编写事件监听器监听到来自底层触发的事件消息。

以影像图层 HsEarth::ImageSurfaceLayer 物体事件触发初始化完成消息 onInitializeCompleted 为例，下面是触发流程。

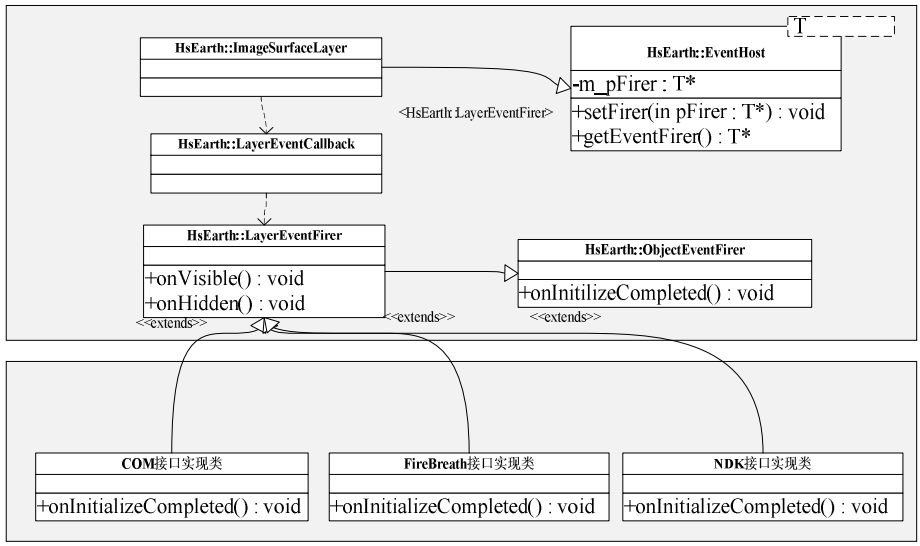

图 4-25 事件机制结构图

HsEarth::ImageSurfaceLayer 继承于 HsEarth::EventHost< HsEarth::LayerEventFirer >，内部保存着一个 m_pFirer 事件触发器指针。HsEarth::LayerEventCallback 对象监听着 HsEarth::ImageSurfaceLayer 对象的内部状态，假设当发生了初始化完成的改变（change）时，会执行 HsEarth::LayerEventCallback 回调，回调内部会判断是否为初始化完成，如果是，则调用 HsEarth::ImageSurfaceLayer 中 m_pFirer 的 onInitializeCompleted 函数，向外部触发初始化完成的事件，派生了 onInitializeCompleted 的 COM 接口实现类会以 COM 方式向上触发事件。

§4.7 跨平台构建

4.7.1 CMake 简介

CMake 是一个跨平台项目管理和构建工具，它是免费开源的，具有 CMake 的跨平台、

简易、可扩展等特点。通过编写少量的 CMake 脚本，CMake 就可以为你的代码生成不同平台、不同环境、不同编译器下的工程。

CMake 具备了例如 AutoConf、JAM、qmake、SCons、ANT 等构建工具的特点的同时，CMake 还添加了许多特性、功能。不像其他工具，CMake 是一个独立完备的构建体系，它在不同的平台环境无需依赖其他第三方工具，运行的是自有的脚本，扩展性强。

CMake 适用于多开发者共同开发、多目标平台、在多台电脑编译的项目。上文所述的 OSG、osgEarth、FireBreath 都是利用 CMake 进行项目构建的。

4.7.2 跨平台项目管理与构建

使用 CMake 对项目核心源码进行管理，使项目核心代码允许跨平台编译。因为修改了 osgEarth 源码，需要在 osgEarth 原来的 CMake 脚本文件中添加新的脚本代码段，以保证修改后的 osgEarth 通过 CMake 工具进行构建时不会出现错误。同时也为 HS_Libs、HS_OSG、HS_Threads、HS_Threads、HS_Earth_Core、HS_Earth 这些项目编写正确 CMake 代码，使这些项目能够在不同平台和编译环境中被 CMake 工作正确生成，并成功编译和正确部署。

4.7.3 Ubuntu 下使用 QT 开发三维数字地球应用示例

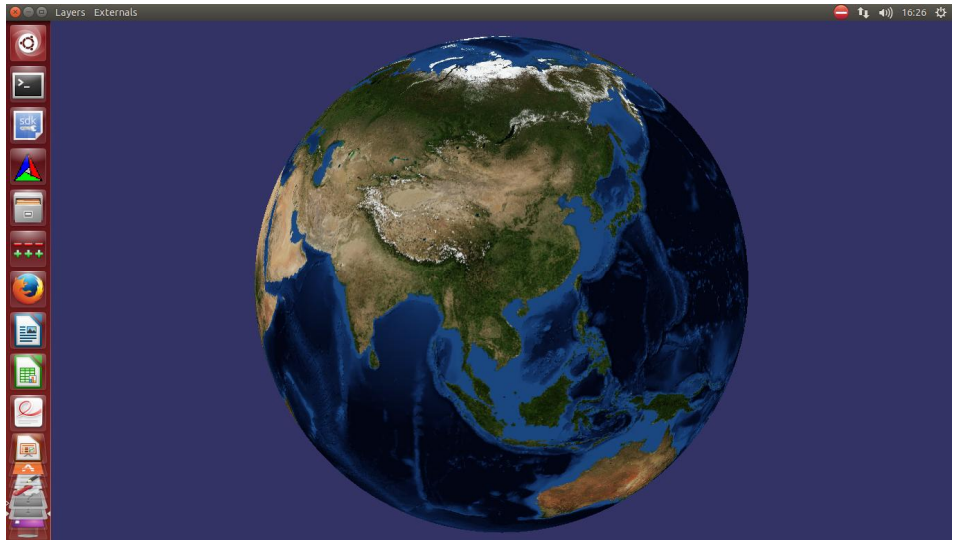

图 4-26 Ubuntu 应用示例图

§4.8 本章小结

本章是一个大章节，涵盖了三维数字地球开发核心层链接库的开发内容。分别详细描述了模板库 HS_Libs、OSG 辅助库 HS_OSG、线程库 HS_Threads、封装 osgEarth 功能的三维数字地球核心库 HS_Earth_Core、三维数字地球接口实现库 HS_Earth 的功能、架构以及内部实现。还介绍了 OSG、osgEarth 源码修改和利用 CMake 进行跨平台项目的管理和构建。

第五章 组件接口实现

§5.1 HS_Earth_COM 组件开发

5.1.1 COM 技术简介

COM（全称: Component Object Model，组件对象模型）是微软公司（Microsoft）推出的一种开发软件组件的规范。COM 与语言、平台无关，体现的是一种开发动态软件组件的思想。利用 COM 框架，使应用程序和操作系统可定制化，开发人员可以开发出各种单独、功能专一的 COM 组件，然后根据开发需求连接形成更复杂的应用程序和操作系统；COM 组件可在不影响其他组件和环境的情况下被动态地更新、插入、卸载，可被多个应用程序链接使用；COM 组件可由不同的语言、不同工具开发；使用 COM 框架，开发人员可以更快速开发出功能强大、扩展性强、维护性强的应用程序。

ActiveX 是微软公司对于一系列策略性面向对象程序技术和工具的称呼，其主要技术是组件对象模型（COM）[6]。ActiveX 控件由 COM 组件或者对象组成，可以被插入到应用程序和网页中，ActiveX 技术在 Windows 下被广泛应用。

ATL（全称: Active Template Library，活动模板库）是微软公司推出的一个用于加速构建 COM 组件（包括 ActiveX 控件）的 C++模板库。COM 组件编写要求高、难度大、过程繁琐，ATL 就是为了简化 COM 组件开发，提高开发效率而推出的。由于 ATL 是一个 C++模板库，所以使用 ATL 可以编写出高效、简洁的代码，无需编写大量繁杂冗余的代码即可自动化生成 COM 组件。ATL 涵括了 COM 组件开发的各个流程，保证了 COM 组件原有的灵活性和强大的功能，还加入了自动化生成和可视化支持，使 COM 组件开发变得简单、高效。

5.1.2 项目范围

HS_Earth_COM 将 HS_Earth 的接口转换成 COM 支持的接口，将 HsEarth::Earth 封装成 ActiveX 控件。

5.1.3 项目架构

HS_Earth_COM 封装了 HS_Earth 的接口，还扩展了数据源、基础单位、基础地理数据等接口。

5.1.4 组件绑定

1. 功能介绍

HS_Earth_COM 组件需要对 HS_Earth、osgEarth 的接口提供绑定，并转换成 COM 支持的接口形式，ATL 中使用 IDL 定义调用和事件的接口。

2. 结构设计

COM 组件将底层接口转换成外部应用程序能访问的接口。

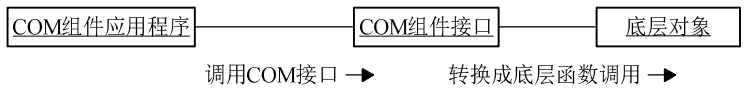

图 5-1 COM 接口转换图

5.1.5 调用接口

1. 功能介绍

在 COM 组件中以 COM 接口的编写方式重新构建 HS_Earth 中定义的类关系，并且将两者绑定起来。

2. 结构设计

HS_Earth_COM 中所有接口都继承于 IEarthDispath，然后根据 HS_Earth 中定义的类关系继承接口，扩展出具体的 COM 接口，例如图层接口和数据源接口等（见图 5-2）。

ATL 中支持接口的多继承，但是不支持接口实现类之间的多继承，一个接口实现类实现了多个接口，虚要为每个继承接口的方法提供实现。HS_Earth_COM 定义了类模板，用于支持上述的类关系，基础接口模板中定义了基础接口的实现函数模板，实现功能的同时减少了重复代码的编写量。

图 5-3 是 HS_Earth_COM 中的 IImageSurfaceLayer 接口实现类 CImageSurfaceLayer 的类结构（模板类省略了部分模板参数）。

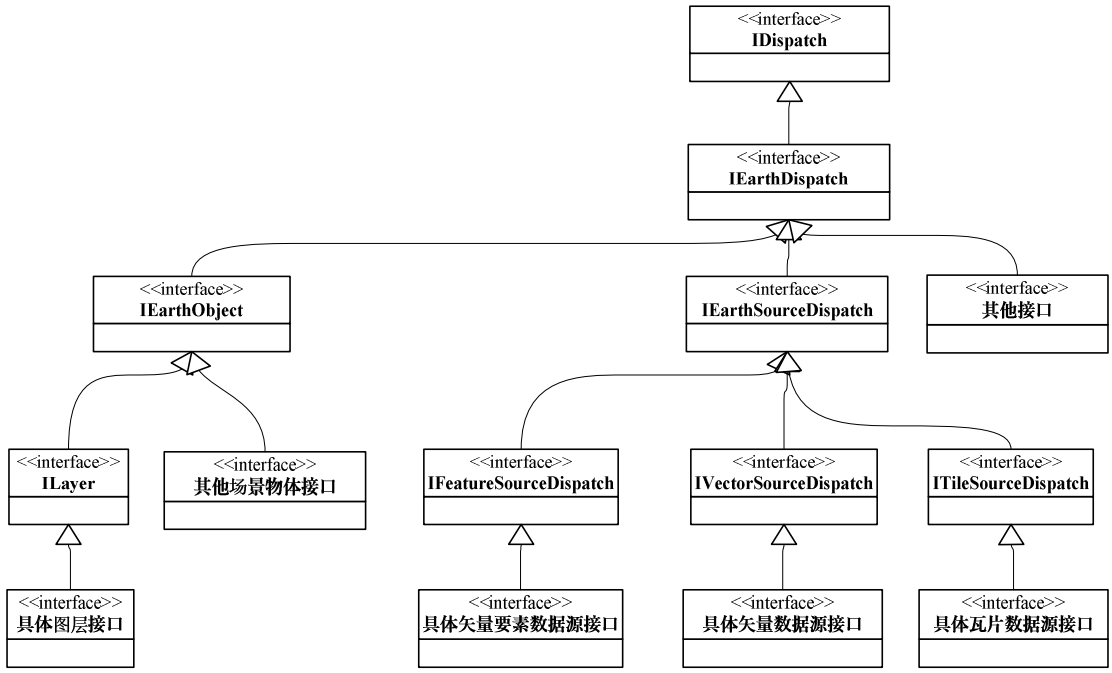

图 5-2 COM 接口继承关系图

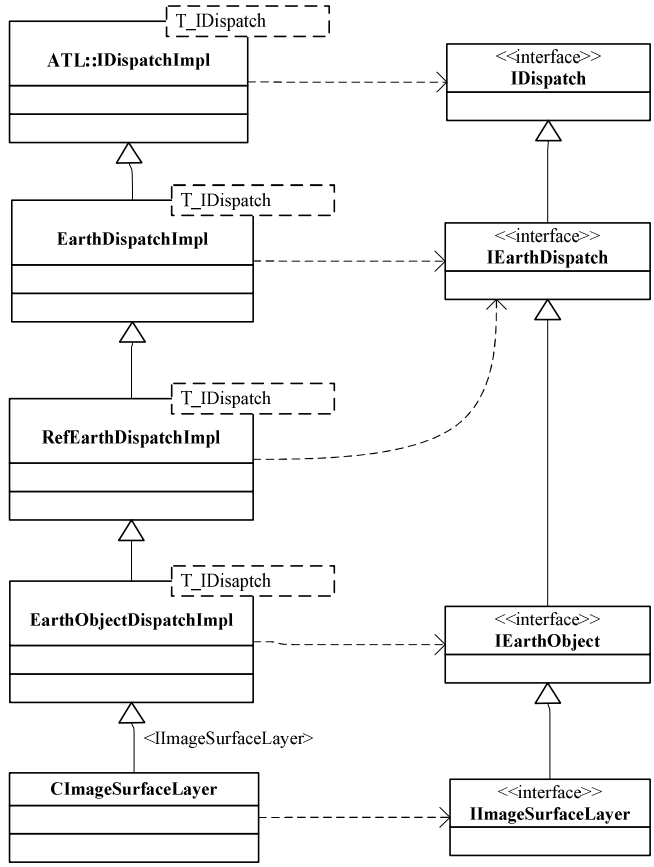

图 5-3 CImageSurfaceLayer 接口实现类关系图

IImageSurface 的父接口有 IEarthObject、IEarthDispatch、IDispatch，CImageSurfaceLayer 继承的各层接口实现模板对应着实现某个接口，例如 ATL::IDispatchImpl 专门针对 IDispatch 接口进行实现，EarthDispatchImpl 专门对 IEarthDispatch 接口进行实现。

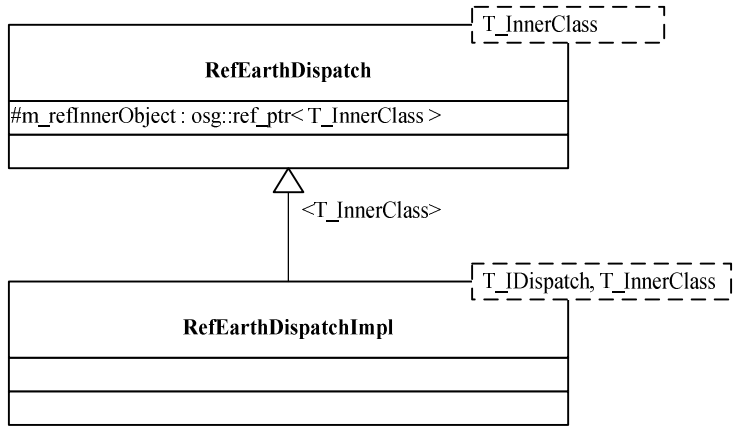

图 5-4 COM 接口与内部对象绑定图

每个 COM 接口实现类中都有一个对应于 HS_Earth 或者 osgEarth 中的类的内部对象，这些 COM 接口最终调用的是内部对象方法。COM 接口实现类与具体内部对象类之间可通过传入模板参数进行绑定，例如 CImageSurfaceLayer 在 RefEarthDispatchImpl 中的内部对象类的参数为 HsEarth::ImageSurfaceLayer，这样 CImageSurfaceLayer 就能够将 IImageSurfaceLayer 接口和 HsEarth::ImageSurfaceLayer 类进行绑定。除了内部对象是引用型的 RefEarthDispatchImpl 之外，还有内部对象是拥有型的接口实现模板类。

5.1.7 事件接口

1. 功能介绍

子接口的事件包含着所继承的父接口的事件，父接口的事件接口可以被子接口继承，HS_Earth_COM 通过事件触发模板来实现事件继承关系，减少重复代码的编写量。HS_Earth_COM 接口实现类通过派生 HS_Earth 中相应的事件触发器，并在派生的事件触发函数中调用定义的 COM 事件触发函数。

2. 结构设计

HS_Earth_COM 中的事件接口与调用接口的关系类似，如果一个 COM 接口实现类想要支持某种类型的事件，只需对实现特定事件接口的模板实现类进行继承即可。图 5-5 是 CImageSurfaceLayer 支持图层事件接口的类结构。

IDL 中定义了 IImageSurfaceLayerEvents 事件接口，在接口实现类中，CImageSurfaceLayer 继承实现事件接口的模板类。当 CImageSurfaceLayer 对象需要触发一个 onInitializeCompleted 的事件时，只需调用继承于 CProxyIEarthObjectEvents 的 Fire_onInitializeCompleted 函数即可。

HsEarth 中已经考虑到如何传递底层消息给上层组件，进而转发到顶层应用程序（见第 4.6.7）。以 CImageSurfaceLayer 向顶层应用程序传递 onInitializeCompleted 事件为例，当

CImageSurfaceLayer 继承了 HsEarth::LayerEventFirer，并派生 onInitializeCompleted 虚函数，在函数内部调用了 CProxyIEarthObjectEvents 的 Fire_onInitializeCompleted 函数，这样，底层的事件会以 COM 的方式传递出去。

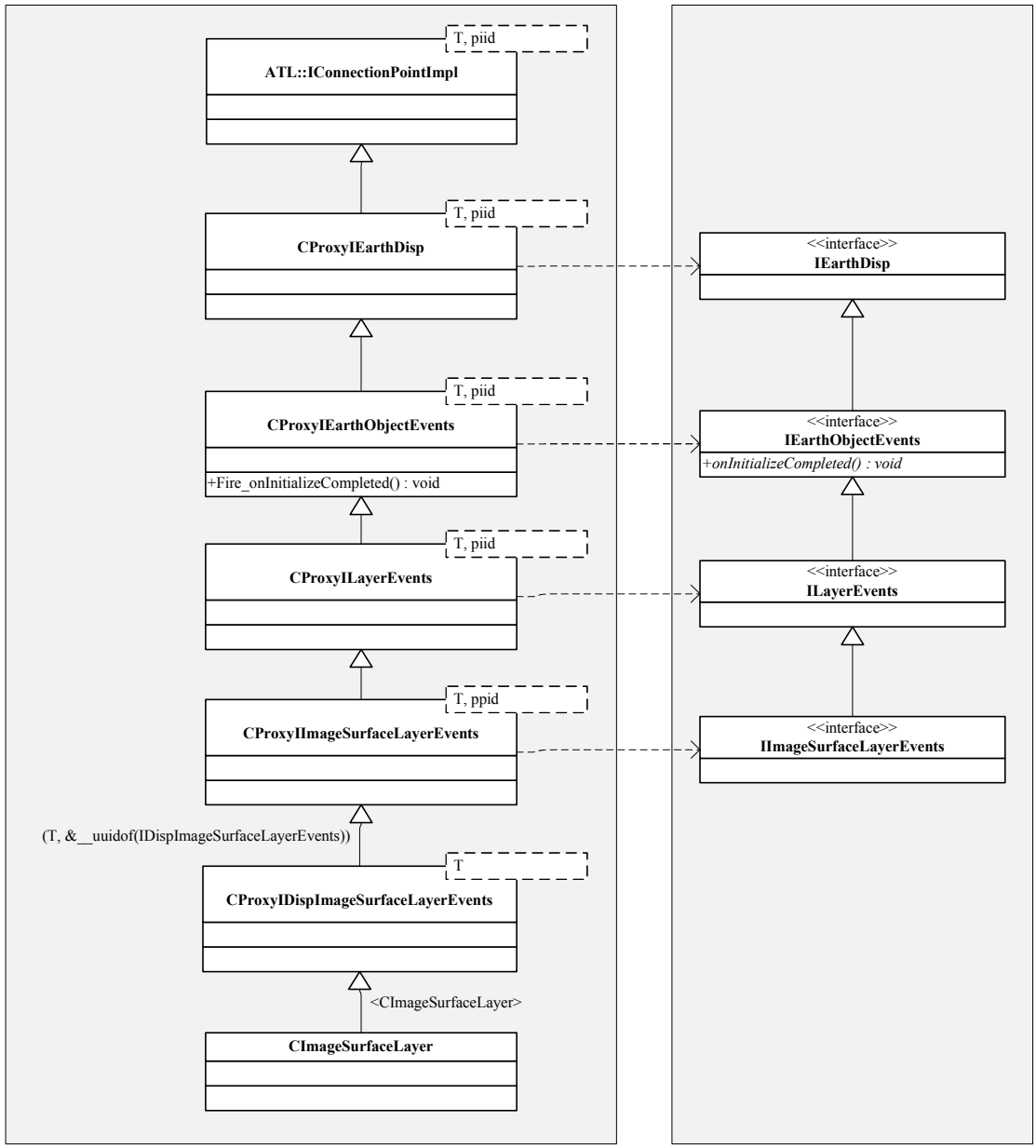

图 5-5 CImageSurface 事件接口继承关系实现图

5.1.9 工具类

1. 功能介绍

HS_Earth_COM 提供了静态工具类，这些工具类用于接口对象创建、辅助开发等。

2. 结构设计

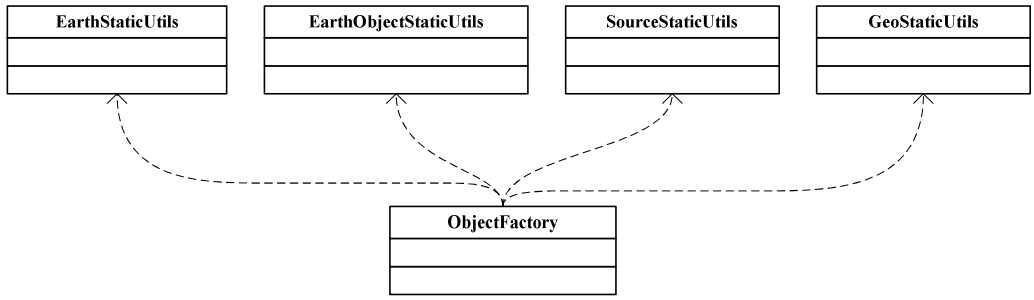

图 5-6 ObjectFactory 类关系图

静态工具类的功能主要是创建 COM 接口对象，或者访问 COM 接口实现类的内部成员。例如，ObjectFactory 是一个 COM 接口实现类，它可根据接口对象类型创建接口对象，在创建具体的 COM 接口对象时，会调用相应的静态类工具创建，然后返回给上层调用。

5.1.10 HS_Earth_COM 组件应用示例

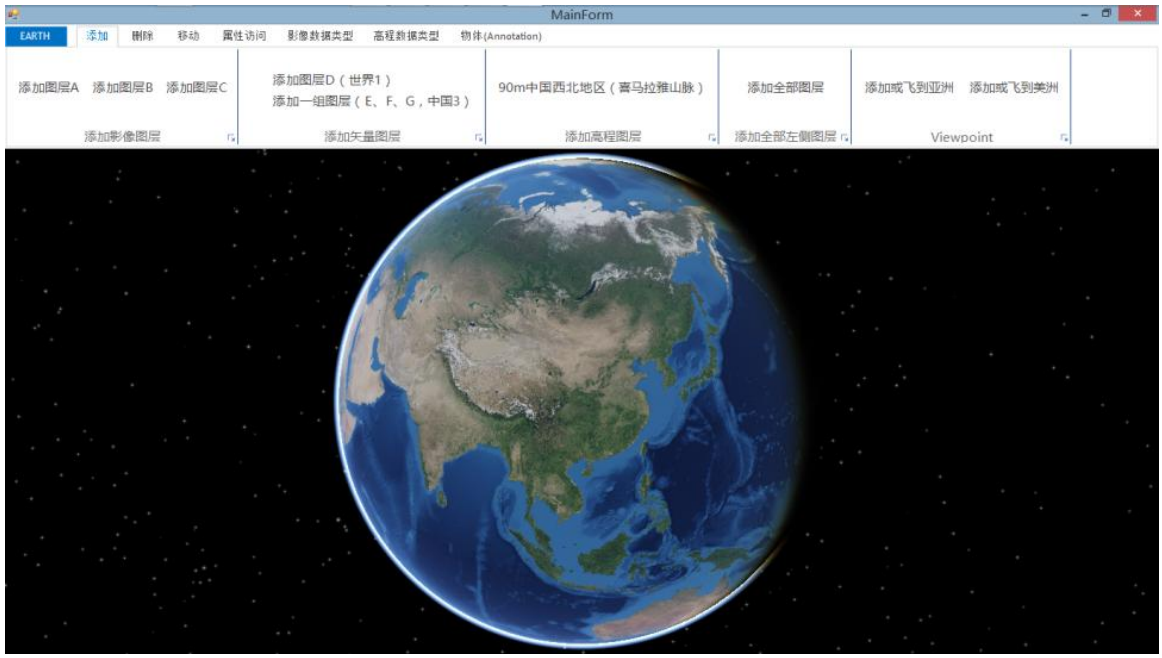

图 5-7 HS_Earth_COM 影像图层应用示例图

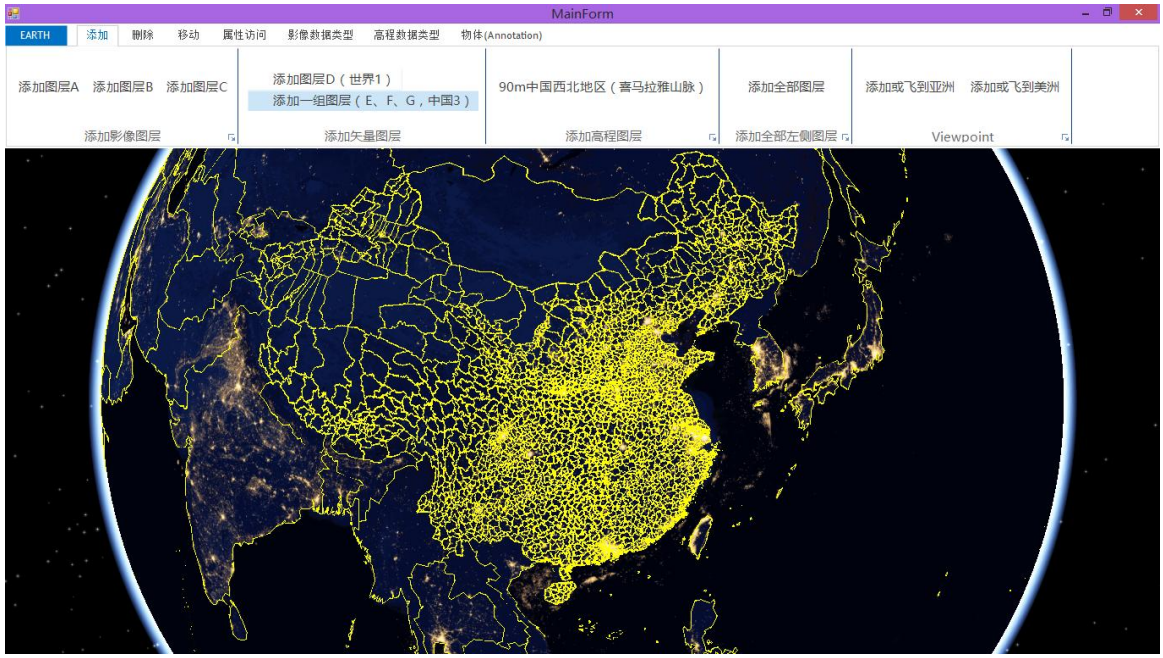

图 5-8 HS_Earth_COM 矢量图层应用示例图

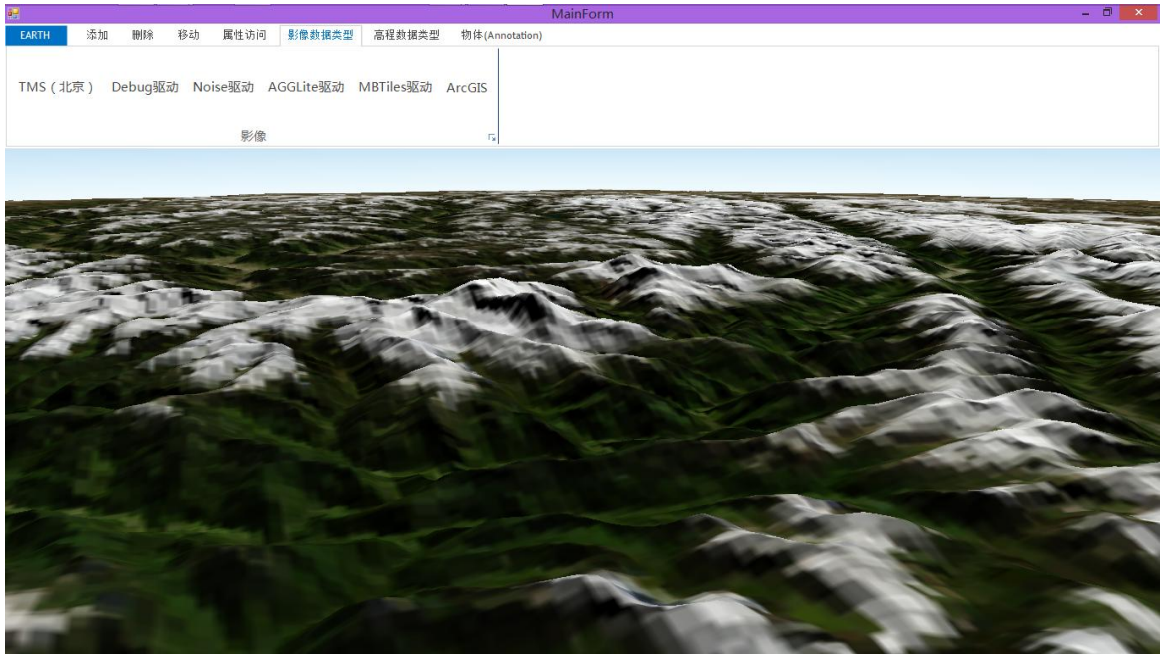

图 5-9 HS_Earth_COM 高程图层应用示例图

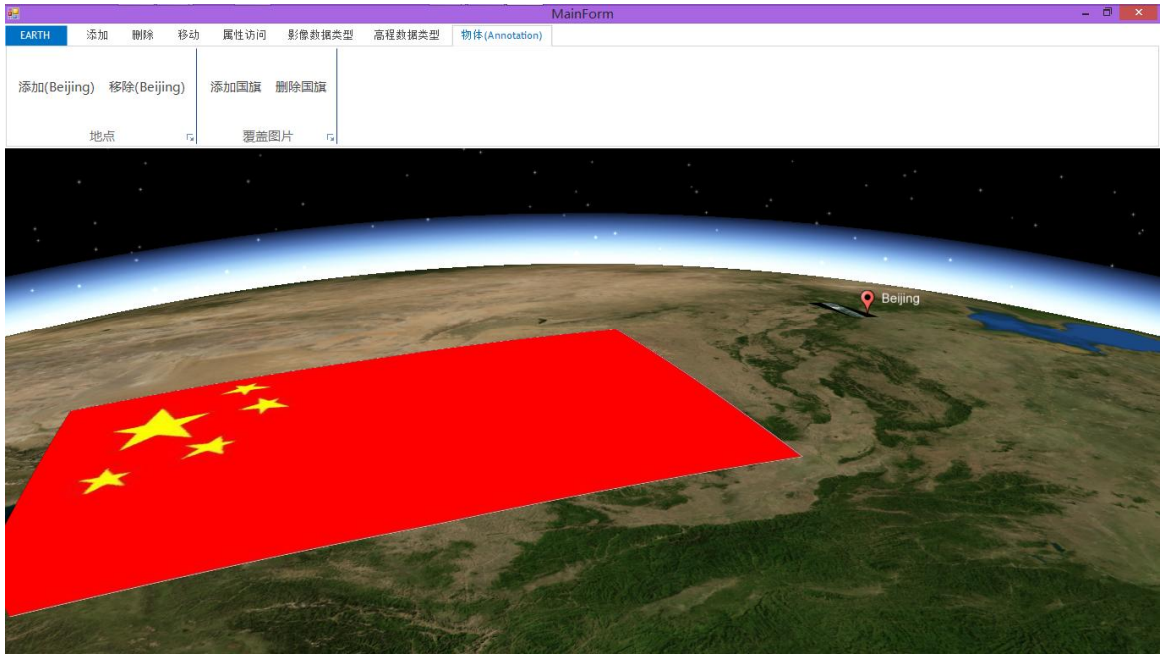

图 5-10 HS_Earth_COM 其他物体应用示例图

§5.2 HS_Earth_Web 插件开发

5.2.1 FireBreath 简介

目前常用的浏览器有 Internet Explorer、Chrome、Safari、Firefox、Opera 等，但它们的插件实现机制却是不一样的。IE 的浏览器插件使用的是 ActiveX 技术，而其他使用的是 NPAPI 技术。

FireBreath 是用于开发跨浏览器插件的跨平台开发框架，它是免费开源的。你可以使用 FireBreath 在 Windows、Mac OS X、Unix/Linux 下开发出功能强大、性能高的跨浏览器插件。

FireBreath 定义了简单、统一的接口编写方式，隐藏了内部具体的实现细节。开发者无需掌握各种浏览器插件的编写规则，使用 FireBreath 即可轻松构建跨浏览器插件，大大简化开发流程、提高开发效率。

FireBreath 接口的数据类型支持 int、double、std::string 等基本数据类型，还支持自定义的数据类型，例如可以装载任意类型数据的 FB::variant 类型。

5.2.2 项目范围

HS_Earth_Web 将 HS_Earth 的接口转换成 FireBreath 支持的接口，将 HsEarth::Earth 封装成跨浏览器插件。

5.2.3 项目架构

HS_Earth_Web 中的架构与 HS_Earth_COM 的架构、接口关系、类关系类似。

5.2.5 接口继承与实现

不像 HS_Earth_COM 调用接口和事件接口需要分别声明和实现，HS_Earth_Web 中的事件和方法可以在同一个基类中声明或者实现，然后被子类继承或者派生。HS_Earth_Web 中的接口继承于 FB::JSAPIAuto，函数和事件都可在接口类内进行声明和绑定。

以 HS_Earth_Web 中的 ImageSurfaceLayer 的接口继承和实现为例来分析。

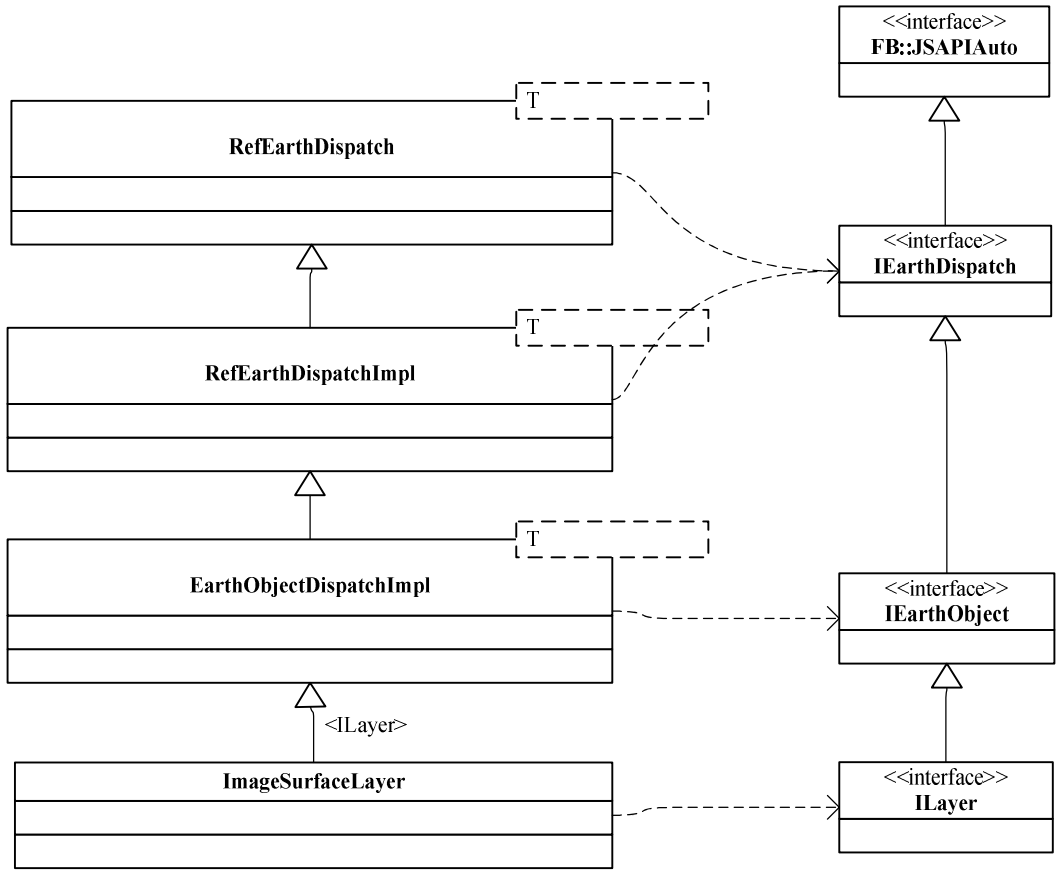

图 5-11 HS_Earth_Web 接口继承图

顶层 RefEarthDispatch 模板公有继承于模板参数 T。所以，ImageSurfaceLayer 继承于 ILayer，各层模板分别实现 ILayer 中的不同接口，最终 ImageSurfaceLayer 继承实现了 ILayer 接口。这结构与图 5-3 很像，但是 FB::JSAPIAuto 支持在同一个接口中声明调用函数和事件，而 COM 是分开的。

5.2.8 工具类

1. 功能介绍

除了与提供与 HS_Earth_COM 功能类似的静态工具类，为了暴露接口给 JS，HS_Earth_Web 中提供类接口工厂 ClassFactory，ClassFactory 的功能是将允许通过 JS 代码暴露 HS_Earth_Web 的接口。

2. 结构设计

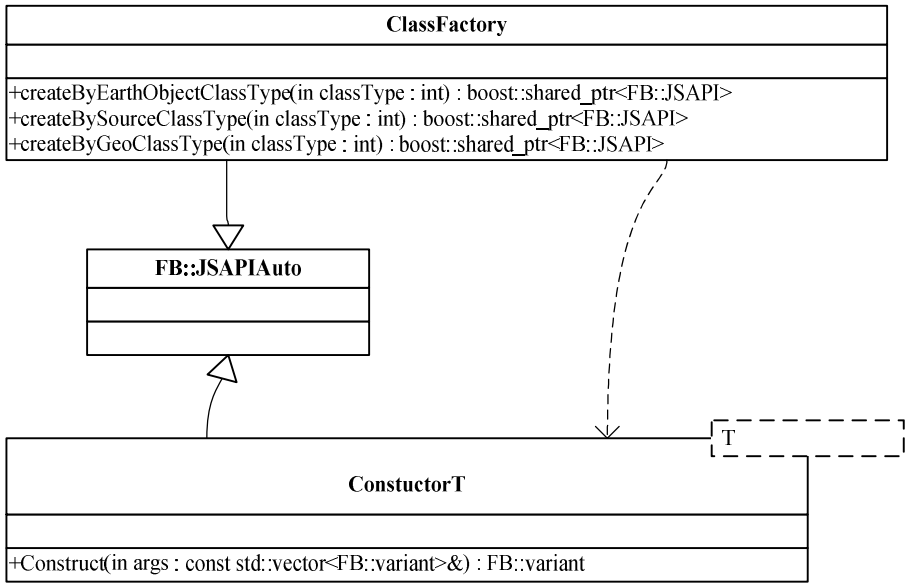

图 5-12 ClassFactory 类图

FB::JSAPIAuto 内部存在一个 Construct 函数，这个函数可以当 JS 接口中使用 new 关键字操作 FB::JSAPIAuto 对象时被调用，利用这个这个特性，可以在 Construct 函数中构建另外一个 FB::JSAPIAuto 对象。ConstructT< T >就是用于构建 T 对象的接口类，例如有继承于 FB::JSAPIAuto 的类 A，FireBreath 对外暴露了 Construct< A >对象 ConstructorA 给 JS 对象 JS_ConstructorA，当对 JS_ConstructorA 使用 new 关键字时，即 var jsA = new JS_ConstructorA()，会调用 ConstructorA 的 Construct 函数并传递构造参数进去，返回的对象是 A 的对象，这样就可以通过 new 的方式构造 A 类对象。

ClassFactory 接口可以根据要创建的接口对象的类型创建 JS 对象。

5.2.9 HS_Earth_Web 辅助库 Earth.js 开发

ClassFactory 可以根据接口对象类型创建 JS 对象。Earth.js 库在插件加载后，将内部的 JS 接口暴露出来。

Earth.js 中定义了代表 HS_Earth_Web 中接口类型的整数，当 HS_Earth_Web 插件加载完成后就会执行 Earth.js 的初始化函数。在初始化函数中，从插件中获取 ClassFactory 的接口

对象，然后再根据各个接口类型整数，将 HS_Earth_Web 各个接口暴露出来。例如，ImageSurfaceLayer 就是从 ClassFactory 对象暴露出来的用于创建 HS_Earth_Web 中 ImageSurfaceLayer 的 JS 接口，可以通过 var imageSurfaceLayer = new ImageSurfaceLayer()来创建 JS 对象。

5.2.10 HS_Earth_Web 应用示例

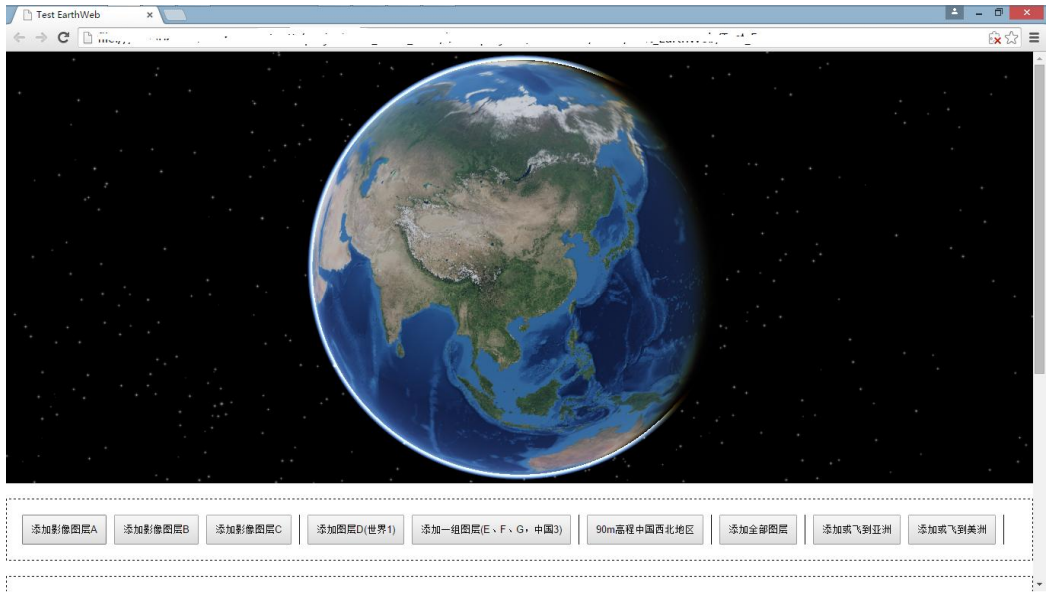

图 5-13 Chrome 应用示例

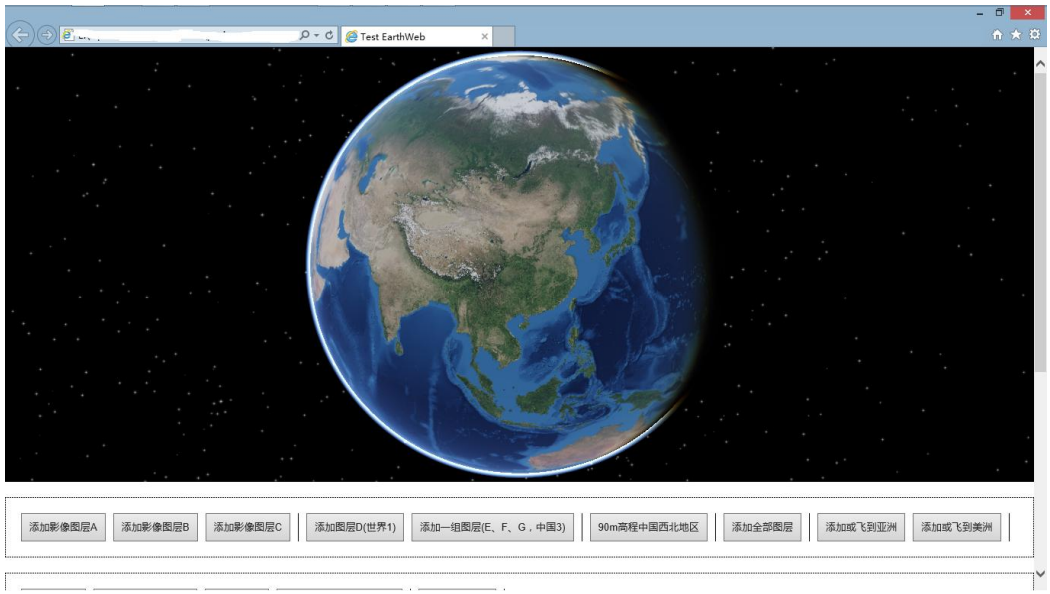

图 5-14 IE 应用示例

§5.3 HS_Earth_Android 链接库展望

5.3.1 Android 软件开发包简介

Android 是目前全球使用人数最多的移动设备操作系统，它是免费开源的。Android 系统架构从底层到顶层，由四个层次构成：Linux 操作系统及驱动层，系统运行库层、应用程序框架层、应用程序层，其中操作系统及系统运行库是由 C/C++编写的，应用程序框架层是由 Java 编写的（见图 5-15）。

Android SDK（全称: Android Software Development Kit），是使用 Java 语言开发 Android 应用的软件开发包。利用 Java 语言和 Android SDK，开发人员更快速地开发出 Android 顶层应用。

JNI（全称: Java Native Interface），是 Java 原生(本地)编程接口。JVM（Java Virtual Machine）虚拟机上的 Java 代码可以通过 JNI 机制调用 C/C++、汇编等语言编写而成的原生代码。

Android NDK（全称: Android Native Development Kit），是使用 C/C++语言开发 Android 应用运行库的软件开发包。利用 C/C++和 Android NDK，可以开发出对性能要求高、比较底层的 Android 应用运行库。NDK 开发出来的应用运行库可以被 Java 通过 JNI 机制调用，开发人员可以使用 Android SDK 开发出基于应用运行库的 Android 顶层应用。

虽然 C/C++代码运行效率比运行在虚拟机的 Java 代码的运行效率要高，但是由于 NDK 使用的是 JNI 机制，使用 NDK 开发应用并不一定会带来性能的提升。相比 Android SDK 开发一般应用，使用 Android NDK 开发应用开发要求高、开发流程繁琐，所以不建议选择 Android NDK 来开发一般应用。只有你的应用程序需要做复杂计算、大量底层操作、图形渲染的时候或者一定要用到 C++的时候，才考虑使用 Android NDK 来开发应用。

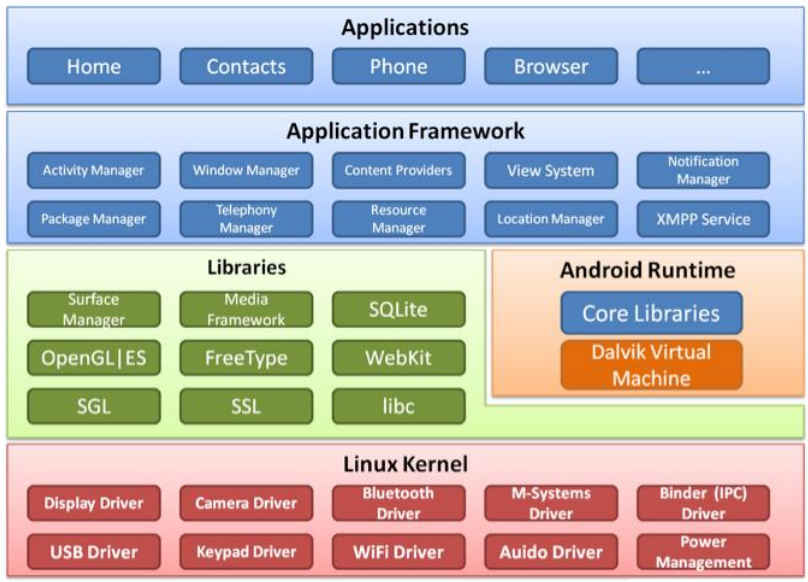

图 5-15 Android 架构图

5.3.2 项目概述

HS_Earth_Android 将 HS_Earth 的接口转换成 Android NDK 支持的接口，将 HsEarth::Earth 封装成 Android 链接库。HS_Earth_Android 总体架构与 HS_Earth_COM 相同，主要区别在于接口关系构建于 JNI 的 Java 端、调用接口和事件接口的分派方式、对象内存的管理。

HS_Earth_Android 现阶段仍处于设计阶段，但其接口调用流程与 HS_Earth_COM、HS_Earth_Web 类似，都是应用了组件设计的原理。

5.3.3 开发 Android 三维数字地球应用示例

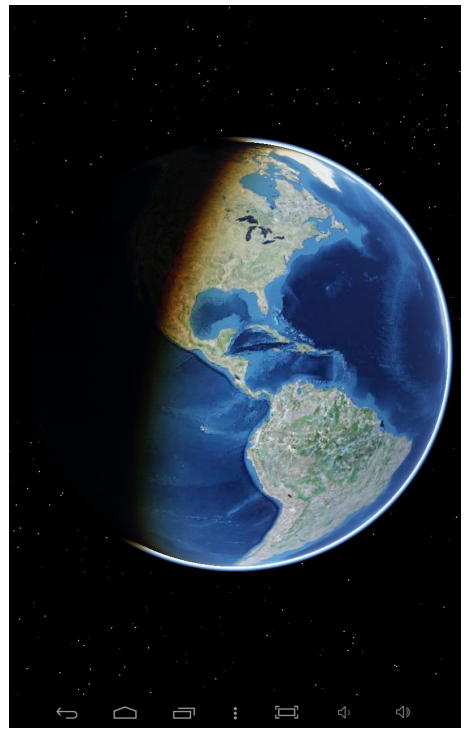

图 5-16 HS_Earth_Android 示例图

§5.4 本章小结

本章详细讲述三维数字地球软件开发包组件接口层的相关技术和组件实现细节。分别介绍了 COM 组件 HS_Earth_COM、跨浏览器插件 HS_Earth_Web 的项目架构、主要流程设计和接口实现，最后展望、介绍了 Android 开发库 HS_Earth_Android 的项目开发要点。

第六章 总结与展望

§6.1 总结

本毕设选题的目标是学习和研究三维数字地球应用开发技术和不同平台组件开发技术，基于 osgEarth 设计出一套通用的三维数字地球库接口，开发出一个跨平台、跨语言的三维数字地球软件开发包 HsEarth，提出一套实用、高效的三维数字地球应用开发解决方案，最后发布、开源该免费的三维数字地球软件开发包。

毕设研究和项目开发过程中，先后学习了 OSG、osgEarth、COM、ATL、FireBreath、CMake、Linux 编程、QT、Android NDK 等相关知识，逐步了解和掌握了三维场景渲染、三维地图应用开发、COM 组件设计和开发、跨浏览器插件设计和开发、不同平台项目构建、Linux 和 Android 共享库开发等技术要点。

经过前期进行资料查询和可行性实验后得出，可以基于 OSG 和 osgEarth 设计出一套与语言、平台无关的通用接口，这套接口可以被其他技术实现，进而开发出跨语言、跨平台的三维数字地球软件开发包。该软件开发包发布给二次开发人员用于开发三维数字地球应用，所以该软件开发包不仅仅考虑了真实客户业务需求，还要考虑到二次开发人员学习的便捷性、开发的便捷性和编写高质量、可维护性的代码等要求。

成功设计和开发出一套调用接口统一的链接库和组件，包括核心库 HS_Libs、HS_OSG、HS_Threads，COM 组件 HS_Earth_COM，跨浏览器插件 HS_Earth_Web。由链接库、组件和示例程序等组成的软件开发包能够根据不同开发需求跨语言、跨平台使用，可以用多种语言开发出不同平台的三维数字地球应用程序。

三维数字地球软件开发包 HsEarth 与原版 osgEarth 对比，支持了多种开发语言，增强二次开发能力，简化了调用接口，增加了功能，提高了性能。

§6.2 展望

目前该项目已经完成了毕设选题要求的所有任务，完成了总体框架的设计和开发，具备了基本功能和接口。但是，当前阶段的成果还存在一些已发现的问题和有待优化的地方：

1) osgEarth 源码在删除瓦片数据源时，仍未立即清除干净缓存瓦片所占用的全部内存。
2) 有时 osgEarth 天空盒会失去效果，这种现象在 HS_Earth_Web 和 HS_Earth_Android 刷新时和 Ubuntu 下开发的应用中更有可能出现。
3) 目前 HS_Earth_COM、HS_Earth_Web 实现大部分主要的功能和接口，但是仍未实现所有已支持的功能。
4) 因为时间的关系，HS_Earth_Android 还处于项目设计阶段，还没进入具体编码实现

的阶段。

5) 由于缺乏开发设备，实现统一接口规范的 IOS 版本 HS_Earth_IOS 的研究和开发工作还没开始。

计划在下一阶段的三维数字地球项目开发的工作内容：

1) 完善功能，修改、完善 osgEarth 源码。
2) 优化核心代码，提高性能。
3) 具体设计、实现 HS_Earth_Android。
4) 开发 HsEarth 面向 IOS 系统的版本 HS_Earth_IOS。
5) 三维数字地球软件开发包发布和开源。

本三维数字地球软件开发包将免费发布出来，并开放源码，最新进展请关注 https://github.com/haozzzzzzzz/HsEarth 中的更新。相信经过不断完善和发展后，该套跨语言、跨平台的三维数字地球软件开发包功能更完整，性能更高效。

致谢

真心感谢大学中每一位授予我知识的老师和陪伴我前行的同学，大学时光有趣而难忘。

特别感谢毕设指导老师张剑波的悉心指导。

向 OpenSceneGraph、osgEarth、CMake、FireBreath、Android 以及其他开源项目的开发人员致敬，感谢他们的辛勤劳动和无私付出。

参考文献

[1]http://www.ev-image.com/products/l-3852968750166.html [OL].2015
[2]http://www.infoearth.com/Product/jichugispingtai/327.html [OL].2015
[3]http://www.osgchina.org/index.php?option=com_content&view=category&id=128&Itemid=525 [OL].2015
[4]王锐,钱学雷.OpenSceneGraph 三维渲染引擎设计与实现.北京:清华大学出版社,2009.5-16
[5]王锐,钱学雷.OpenSceneGraph 三维渲染引擎设计与实现.北京:清华大学出版社,2009.56-60
[6] http://baike.baidu.com/view/28141.htm [OL].2015