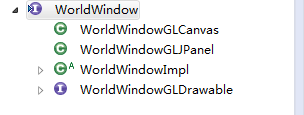
WorldWind一览

WorldWind Java SDK（有时候简写做WWJ）是用Java开发的一个三维地理信息系统的雏形。请原谅我叫它“雏形”，因为比起GIS界一些较成熟的三维平台，它目前还是一个实验室产品，不过这丝毫不会影响它在三维地理信息领域的地位，尤其是在开源GIS领域。正如NASA为人类太空探索所做出的卓越成就一样，它也是NASA给予GISer们最好的礼物。

利用WorldWind开发三维地理信息系统，你仅仅需要将gov.nasa.worldwind.WorldWindow类以组件的形式嵌入到你的界面里即可。形如下面的代码：

1. WorldWindow wwd = new WorldWindowGLCanvas();
2. // 设置wwd
3. ……
4. JFrame frame = new JFrame();
5. // 设置frame
6. ……
7. frame.add((Component)wwd, BorderLayout.CENTER);

在以上代码中，wwd是WorldWindowGLCanvas类型，WorldWIndowGLCanvas类型实现了WorldWindow接口，同时它还实现了GLCanvas接口，而GLCanvas是Component的子类，所以第7句中对wwd的Component强转是合法的。事实上WorldWind里实现了WorldWindow的类型大都间接地实现了Component接口。



其中WorldWindowGLCanvas和WorldWindowGLJPanel都实现了间接地Component接口，而WorldWindowImpl抽象类和WorldWindowGLDrawable接口没有实现Component接口，他们实现了基本的WorldWindow功能，开发者可以从这二者入手开发自己的Component。

WorldWind的组件是非常易于扩展的，它的API都是用接口来定义的，所以很多组件可以有选择的用另一个可替换的组件代替。

关于如何使用WorldWind来构建Applet，可以查看gov.nasa.worldwindx.examples.applet包下的例子。

WorldWind具有多种实现接口，可以直接与Swing或者AWT对接，将来也会增加对SWT的支持。

除了WorldWindow类，WorldWind中还有五个主要的接口，它们是：

* gov.nasa.worldwind.globes.Globe：用来表现一个诸如“地球”这样的星体的形状和地形起伏特征。
* gov.nasa.worldwind.layers.Layer：用于展示 “Globe”上的影像或其它信息。更专业点的叫法就是“图层”。
* gov.nasa.worldwind.**M**odel：组合Globe和Layer，使它们成为一体。
* gov.nasa.worldwind.Scene**C**ontroller：控制Model的渲染。
* gov.nasa.worldwind.**V**iew：是用户与Model进行交互的界面，也是Model渲染的载体。

作为一种典型的应用，通常一个基于WorldWind的应用程序会关联一个由一个Globe和多个Layers组成的Model，然后通过一个SenceController将Model显示在一个WorldWindow中。SenceController叫做场景控制器，它的主要作用就是将Globe和Layers组合在一起然后显示在用户界面上，而且管理着与用户的交互（[《两种worldwind开发环境的搭建方式》](两种worldwind开发环境的搭建方式.docx)一文中有这一典型应用的例子）。

任何实现了以上接口的对象，不管是NASA在worldwind里已经实现的还是应用程序开发者自己创建的，都可以被用在任何这些接口被调用的地方（多态）。在NASA已经发行的开发包里已经提供了多个Globe的实现，比如Earth、Mars、Moon，而且对Model、SenceController和View都有一些基本的实现。

再啰嗦一下，大多数的WorldWind组件都被定义为接口，这就极大程度的方便了开发者将自己的具体实现轻而易举的集成到WorldWind中。

# 数据检索（Data Retrieval）

WorldWind的运行是伴随着巨大的数据量的，而几乎所有主要的数据都存储在远程服务器上，那么，对数据进行检索和对这些数据建立本地缓存便是WorldWind的一个主要特征。负责该工作的是这两个类：gov.nasa.worldwind.retrieve.Retriever和gov.nasa.worldwind.retrieve.RetrievalService。

Retriever负责将检索的请求进行封装，不过gov.nasa.worldwind.retrieve.Retriever只是一个接口，最长使用的具体Retriever是它的一个实现类gov.nasa.worldwind.retrieve.HTTPRetriever，它通过http协议来检索数据。检索请求通常是由下面两个类发出的，一个是gov.nasa.worldwind.layers.Layer，它会创建Retriever来检索该图层显示所需要的数据；一个是gov.nasa.worldwind.globes.ElevationModel，它创建Retriever是用来检索高程数据的。

gov.nasa.worldwind.retrieve.RetrievalService通过一个线程池来管理检索任务。需要检索数据的对象将创建出来的Retriever对象传递给RetrievalService，RetrievalService会为每一个Retriever创建一个单独的线程。RetrievalService是gov.nasa.worldwind.WorldWind的成员，可以通过WorldWind类去访问RetrievalService。WorldWind使用了单例模式，在整个应用程序运行期间是独一份的，可以在全局访问它。

当检索的数据到达的时候，RetrievalService会通知gov.nasa.worldwind.retrieve.RetrievalPostProcessor，这个类在具体的Retriever被构造时传递到了Retriever里，它的作用是在检索到数据时，传递或者决定如何来将下载的数据进行持久化。具体的持久化和任何持久化之前的操作是由具体的对象决定的，比如gov.nasa.worldwind.layers.TiledImageLayer会将非DDS类型的数据转换为DDS格式。但也可能是简单的将数据按照原样储存在文件里，比如gov.nasa.worldwind.terrain.BasicElevationModel就是将数据不经任何加工的保存起来了。RetrievalPostProcessor不是在UI线程和要素渲染的线程里执行，而是和Retriever线程工作在一起。

之前检索到的或者其它原因存储在硬盘上的数据是在一个单独的线程里被载入内存的，也就是说这个线程和UI线程以及渲染线程是隔离开的。一条WorldWind的惯例就是：在渲染线程里，不以任何方式去访问硬盘上的数据。基于此，从硬盘加载数据的线程被指派到了另一个线程池内。管理这个线程池的类是gov.nasa.worldwind.util.ThreadedTaskService，它和RetrievalService拥有比较类似的接口。它负责执行的任务是从硬盘读取数据并添加到一个全局缓存里。

在渲染的时候，肯定要检测一下需要的数据是否在磁盘上或缓存里，如果是则可以直接加载，但如果不是就必须去远程服务器上检索。这些工作如果和渲染放在一个线程里一定会影响渲染的流畅度，所以如上所述的这种磁盘访问限制就显得非常有意义了。依照这个惯例，如果对象需要加载数据就会首先检测一下缓存里是否有需要的数据，如果没有的话就单独创建一个线程去检测需要从哪里获取数据，是磁盘还是网络。如果在磁盘上找到了数据，就会正常读取它并立即放入缓存。如果数据在远程服务器上，该线程就创建一个Retriever去向服务器请求数据。

# 缓存（Memory Cache）

为了让可缓存的对象之间共享数据，WorldWind中的大多数缓存数据都存在gov.nasa.worldwind.cache.MemeryCache中。MemoryCache使得一个打开的应用程序实例中的所有WorldWindWindow可以共享一套数据（在内存中）。也就是说，在两个独立的窗口内个显示一个Earth的时候，它们能够同时共享所有的影像或高程瓦片等数据。不过这样做有个要求就是，被共享的数据的equals()方法和hashCode()方法所基于的实例不能是特定的缓存对象。

# 文件缓存（File Cache）

所有持久话的或者从本地机器得到的数据都是由FileStore来管理的，所有的对象对不亲自管理自己的存储。文件缓存负责多个磁盘存储并负责统一的访问这些数据。文件缓存是一个单例，从另一个单例类WorldWind访问得到。