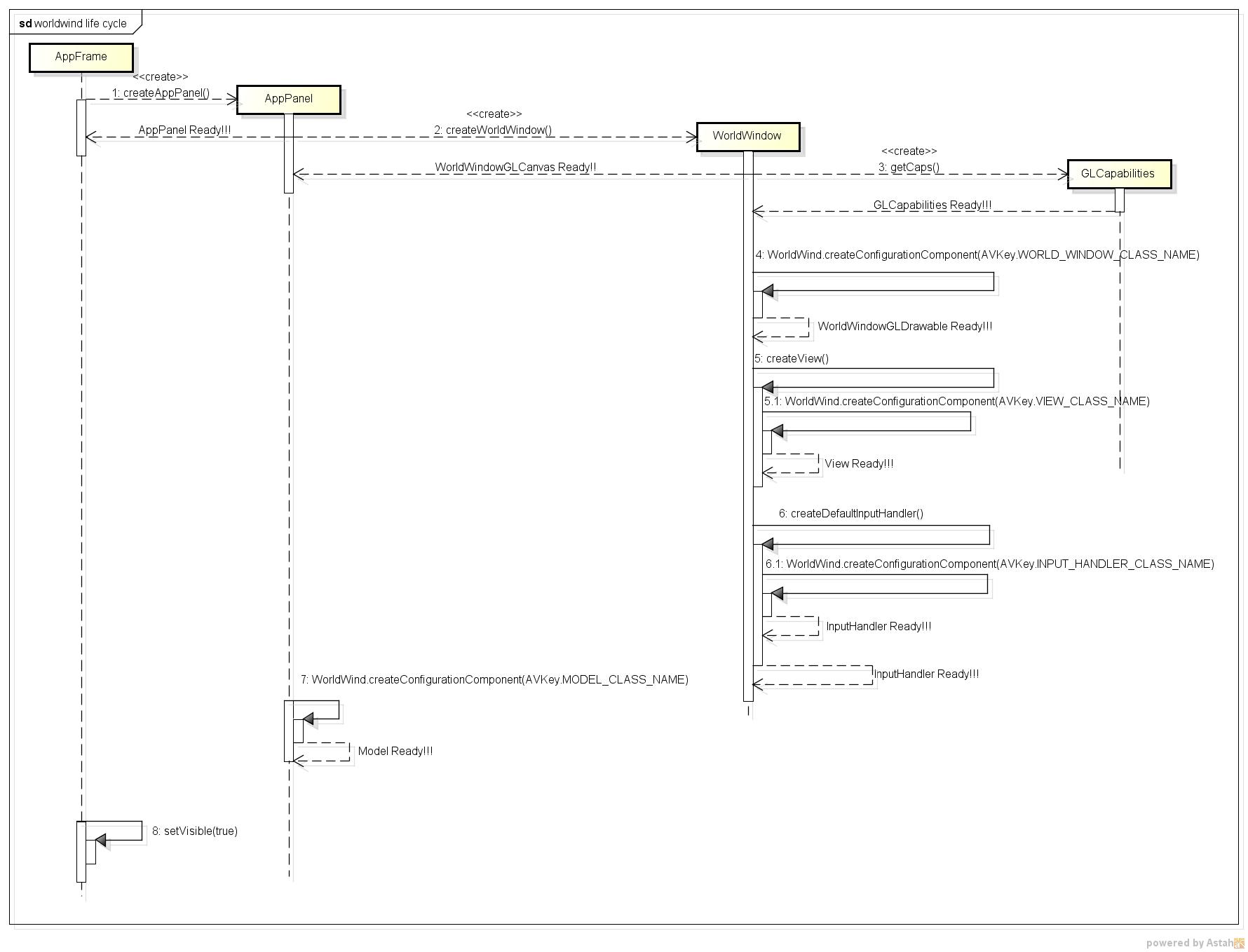
# 什么时候加载配置文件？从哪里加载？

为了能够找到WorldWind是如何加载配置文件的，我详细地跟踪了一下ApplicationTemplate的代码，绘制出如下一张序列图，描述了一个WorldWind窗口的创建过程：



图表 1 WorldWind窗口的创建过程

从图中可以看到，WorldWind.createConfigurationComponent方法在整个创建过程中被频繁调用，从字面上理解，这个方法大概就是用来创建配置文件中设置的组件的，下面就进入该方法了解一下。

## WorldWind.createConfigurationComponent(String classNameKey)

WorldWind的实现使用了“单例模式”，通过在WorldWind的私有构造函数中设置断点，查到了WorldWind首次被调用的地方是在WorldWindowGLCanvas类的构造函数中，也就是图表 1中的第4步。WorldWind的初始化都在该类的私有成员函数initialize()中进行，该函数的部分代码如下：

代码清单 1 WorldWind.initialize()

|  |
| --- |
| **private** **void** initialize()  {  **this**.wwo = **new** WWObjectImpl();  **this**.remoteRetrievalService = (RetrievalService) *createConfigurationComponent*(AVKey.*RETRIEVAL\_SERVICE\_CLASS\_NAME*);  ……  } |

在该函数中，WorldWind.createConfigurationComponent(String classNameKey)被首次调用。

该方法的主要代码如下：

代码清单 2 WorldWind.createConfigurationComponent(String classNameKey)

|  |
| --- |
| **public** **static** Object createConfigurationComponent(String classNameKey) **throws** IllegalStateException, IllegalArgumentException  {  ……  String name = Configuration.*getStringValue*(classNameKey);  ……  **try**  {  **return** WorldWind.*createComponent*(name.trim());  }  ……  } |

此方法的作用就是接受一个String类型的参数作为键值，从Configuration类中获取类名，然后再调用WorldWind的createComponent(String className)创建相关的类。由此可猜测，Configuration一定以某种方式存储了从配置文件加载的信息，下面就来进行一个验证。

## Configuration

Configuration同样使用了“单例模式”，通过将断点设置在它的私有构造函数中，定位到了Configuration被首次调用的地方在ApplicationTemplate的静态块中：

代码清单 3 ApplicationTemplate中出现Configuration的静态块

|  |
| --- |
| **static** {  ……  if (Configuration.isMacOS())  ……  } |

下面看看Configuration在构造函数中都做了什么与配置文件相关的工作。

代码清单 4 Configuration的私有构造函数

|  |
| --- |
| **private** Configuration()  {  this.properties = initializeDefaults();  ……  // Load the default configuration  **this**.loadConfigDoc(System.*getProperty*(*CONFIG\_WW\_DOCUMENT\_KEY*, *CONFIG\_WW\_DOCUMENT\_NAME*));  ……  } |

首先初始化了一个Properties类型的成员，经过对Configuration类其它细节的查看，该Properties成员里面即存储了所有从配置文件加载的信息。程序清单 4所摘录的下一行代码是调用该类的loadConfigDoc函数，这个方法便是加载配置文件的入口点了。loadConfigDoc函数在此接受了一个String类型的参数，该参数的值由其私有静态常量*CONFIG\_WW\_DOCUMENT\_NAME*提供，该常量的值是“config/worldwind.xml”，通过阅读loadConfigDoc的代码后可以确定，这便是WorldWind加载的第一个配置文件的路径了。

## 小结

位于程序运行目录config下的worldwind.xml文件是WorldWind加载的第一个配置文件，该文件在Configuration的构造函数中被加载，而第一次调用Configuration的地方在ApplicationTemplate的一个静态块中，由于Configuration使用了“单例模式”，程序运行期间只有一个实例存在，在程序接下来的运行过程中将始终使用该Configuration实例中的信息。

# config/worldwind.xml

代码清单 5 config/worldwind.xml配置文件部分内容

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  ……  <WorldWindConfiguration version=*"1"*>  ……  <LayerList href=*"config/worldwind.layers.xml"*/>  <!--Most configuration information is defined in attribute-value pairs-->  <Property name=*"gov.nasa.worldwind.avkey.ModelClassName"* value=*"gov.nasa.worldwind.BasicModel"*/>  <Property name=*"gov.nasa.worldwind.avkey.GlobeClassName"*  value=*"gov.nasa.worldwind.globes.Earth"*/>  <Property name=*"gov.nasa.worldwind.avkey.EarthElevationModelConfigFile"*  value=*"config/Earth/EarthElevationModelAsBil16.xml"*/>  <Property name=*"gov.nasa.worldwind.avkey.InitialLatitude"* value=*"38"*/>  ……  <Property name=*"gov.nasa.worldwind.avkey.ElevationTileCacheSize"* value=*"10000000"*/>  ……  <Property name=*"gov.nasa.worldwind.avkey.MilStd2525IconRetrieverPath"*  value=*"http://worldwind.arc.nasa.gov/milstd2525c/rev1/"*/>  </WorldWindConfiguration> |

这是WorldWind的默认配置文件，通过阅读Configuration的构造函数代码还可以知道，如果用户想指定其它的配置文件，可以通过设置Java Property“gov.nasa.worldwind.config.file”来达到目的。

下面详细分析一下，这个配置文件里对都哪些信息做了说明。

## LayerList

定义了在程序启动时加载的图层，具体的图层列表通过href指定的配置文件进行定义。

## Property

通过分析Configurationd的loadConfigProperties(Document doc)方法可以得知，所有Property节点所定义的“键-值”对都将添加到其properties成员中。这部分属性定义主要包括以下几部分：

### 加载类

比如“gov.nasa.worldwind.avkey.ModelClassName”的值被设置为“gov.nasa.worldwind.BasicModel”，说明从配置文件加载的模型类（Model）是gov.nasa.worldwind.BasicModel类；

再如“gov.nasa.worldwind.avkey.GlobeClassName”的值为“gov.nasa.worldwind.globes.Earth”，说明默认的球体类型是Earth（地球）。

### 路径

比如“gov.nasa.worldwind.avkey.EarthElevationModelConfigFile”的值为“config/Earth/EarthElevationModelAsBil16.xml”指示了地球高程模型的配置文件是位于config/Earth/目录下的EarthElevationModelAsBil16.xml文件。

再如“gov.nasa.worldwind.avkey.MilStd2525IconRetrieverPath”的值为“http://worldwind.arc.nasa.gov/milstd2525c/rev1/”，也是表达类似的意思。

### 全局值

比如“gov.nasa.worldwind.avkey.InitialLatitude”配置视图初始位置的纬度坐标值。

再如“gov.nasa.worldwind.avkey.ElevationTileCacheSize”配置了高程瓦片缓存大小。

# worldwind.layers.xml

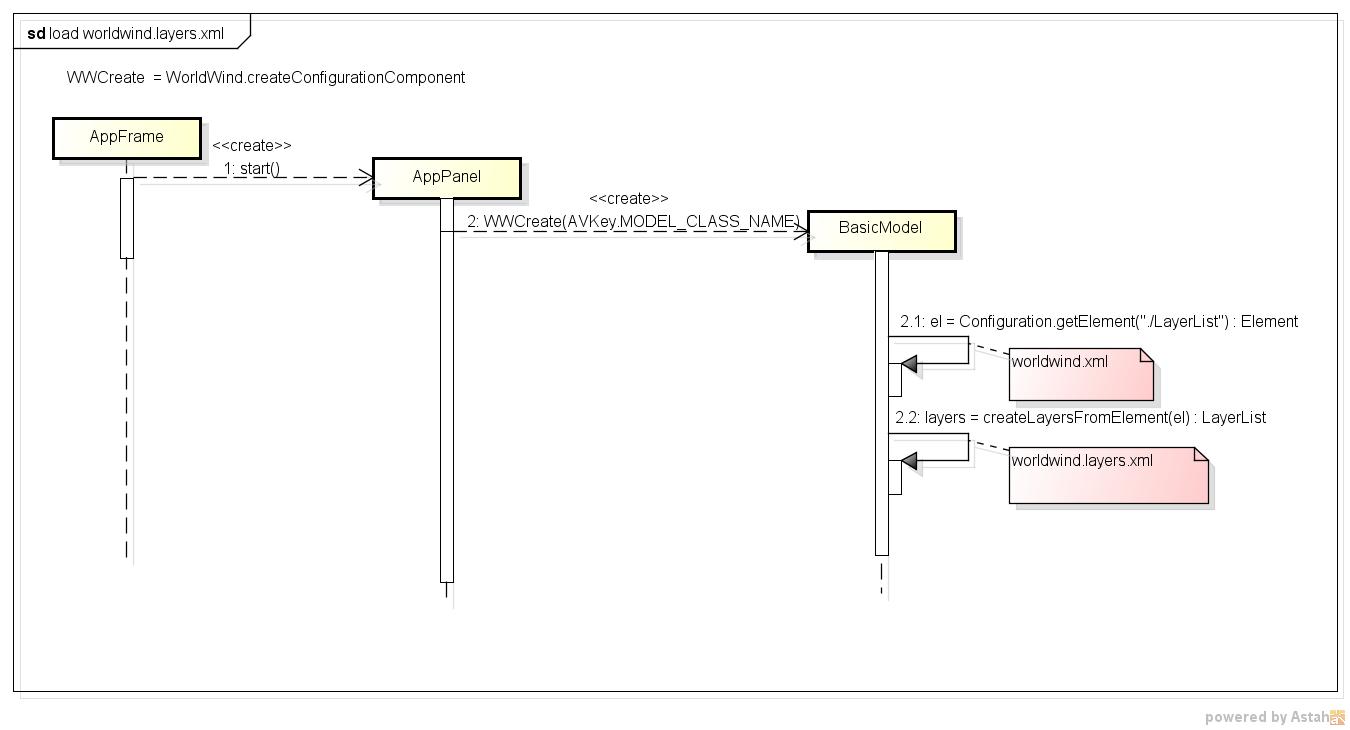
代码清单 6 worldwind.layers.xml配置文件部分内容

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  ……  <LayerList>  ……  <Layer className=*"gov.nasa.worldwind.layers.Earth.BMNGOneImage"*>  <Property name=*"MinActiveAltitude"* value=*"3e6"*/>  </Layer>  <Layer href=*"config/Earth/BMNGWMSLayer.xml"* actuate=*"onLoad"*/>  ……  <Layer href=*"config/Earth/USGSUrbanAreaOrthoLayer.xml"* actuate=*"onRequest"*/>  ……  </LayerList> |

该xml的结构比较容易理解，根节点<LayerList>下的每一个节点都定义了一种图层，而定义图层的形式一共有两种，一种是直接指定“类名”，那么该图层的相关详细配置就参考该类定义即可；另一种是指向另一个配置文件，关于该图层的详细配置都在这个目标配置文件里进行。

图层有一个属性“actuate”，如果属性值是“onLoad”则在系统启动后就会加载；如果属性值是“onRequest”则不会自动加载，需要用户在“图层面板”上或其它地方激活才会加载。

该配置文件初次使用的地方可以用下面的序列图来表达：



图表 2 worldwind.layers.xml的加载

# WMSLayerTemplate.xml

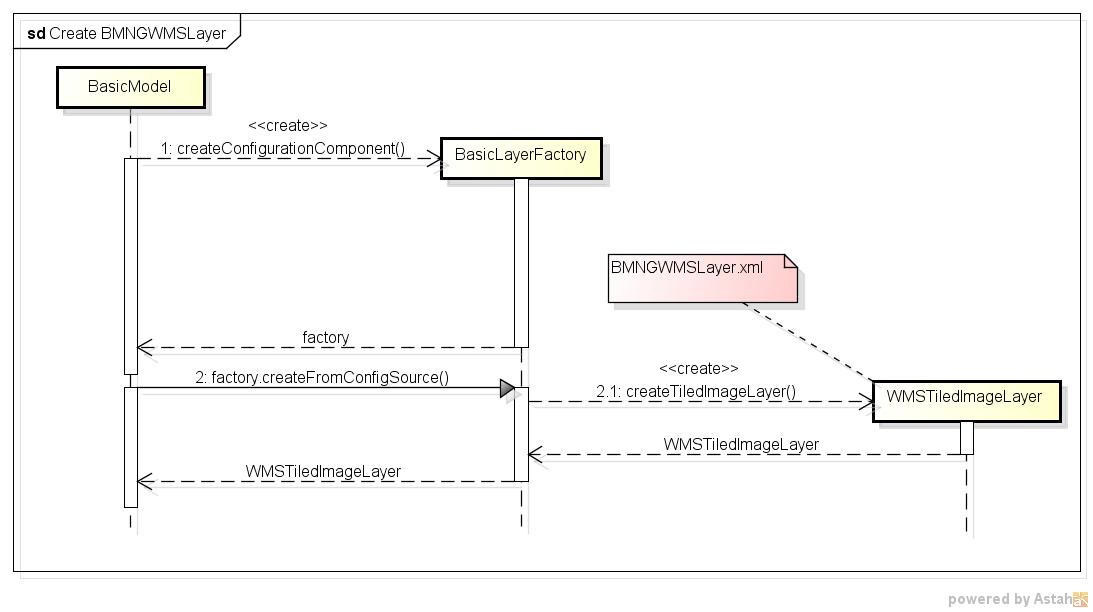
WMS图层配置文件的一个模版文件，仿照该模版配置WMS图层。对该文件的说明可参考代码清单 7中的注释。

代码清单 7 WMSLayerTemplate.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!--$Id: WMSLayerTemplate.xml 1 2011-07-16 23:22:47Z dcollins $-->  <!-- 这是一个WMS影像图岑的配置实例。这个文件用于指导开发者使用一些必要的参数去配置一个WMS影像图层。 -->  <!-- 跟节点定义了配置文件的版本和图层的类型。从2011年3月起，version的属性必须设置为“1”并且必须使用“TiledImageLayer”作为图层类型来定义一个WMS影像图层。 -->  <Layer version=*"1"* layerType=*"TiledImageLayer"*>  <!-- 在图层列表中将显示这个名字。 -->  <DisplayName>你的图层名</DisplayName>  <!-- 下面用来配置WMS数据服务的地址以及WMS图层名等参数。其中<LayerNames>的元素要么是一个单独的字符串，要么用逗号将几个字符串分割开来。 -->  <Service serviceName=*"OGC:WMS"* version=*"1.3"*>  <GetCapabilitiesURL>http://数据服务URL</GetCapabilitiesURL>  <GetMapURL>http://数据服务URL</GetMapURL>  <LayerNames>图层名\_1,图层名\_2,图层名\_3</LayerNames>  </Service>  <!-- 配置一个目录用来保存下载好的瓦片数据。这个目录必须是唯一的，目录名必须是以下符号的组合：a-z, A-Z, 0-9, "\_", 和 "/". -->  <DataCacheName>YOUR\_CACHE\_PATH</DataCacheName>  <!-- 缓存中保存的文件类型 -->  <FormatSuffix>.png</FormatSuffix>  <ImageFormat>image/png</ImageFormat>  <!-- WMS服务器所支持的影像类型列表， 该列表有优先级，排在前面的格式应该是你的WMS服务器优先支持的类型。-->  <AvailableImageFormats>  <ImageFormat>image/png</ImageFormat>  <ImageFormat>image/jpg</ImageFormat>  </AvailableImageFormats>  <!-- 配置影像金字塔的层数，对于WMS影像，19层数据的分辨率大概在0.1米/像素。 -->  <NumLevels count=*"19"* numEmpty=*"0"*/>  <!-- 配置WMS图层的范围，用一个Sector表示。 -->  <!-- 如果包含的区域内没有数据，在<UseTransparentTextures>属性设置为true的情况下会显示为透明的像素。 -->  <Sector>  <SouthWest>  <LatLon units=*"degrees"* latitude=*"西南角纬度"* longitude=*"西南角经度"*/>  </SouthWest>  <NorthEast>  <LatLon units=*"degrees"* latitude=*"东北角纬度"* longitude=*"东北角经度"*/>  </NorthEast>  </Sector>  <!-- 如果想要在没有数据的地方显示透明像素，该节点的属性必须设置为true -->  <UseTransparentTextures>true</UseTransparentTextures>  </Layer> |

# BMNGWMSLayer.xml

程序在根据worldwind.layers.xml创建图层时，如果图层的定义是由“href”的形式给出，即将定义链接到了另一个xml配置文件，则会根据所连接到的配置文件内容构建图层。下图便是根据BMNGWMSLayer.xml构建Blue Marble (WMS)图层的流程：



图表 3根据BMNGWMSLayer.xml构建Blue Marble (WMS)图层

# DataFileStore.xml

worldwind.xml中有这么一个配置：<Property name="gov.nasa.worldwind.avkey.DataFileStoreConfigurationFileName" value="config/DataFileStore.xml"/>。它指示了一个用来配置数据文件存储目录的文件。通过该配置文件将会创建DataFileStore的具体实例。对该文件的说明可参考代码清单 8中的注释。

代码清单 8 DataFileStore.xml

|  |
| --- |
| <?xml version=*"1.0"*?>  <!-- World Wind 数据文件存储的配置 -->  <!--$Id: DataFileStore.xml 1 2011-07-16 23:22:47Z dcollins $-->  <dataFileStore>  <readLocations>  <!--当World Wind需要加载本地数据或图像的时候，就会按照<readLocation>节点下配置的路径顺序搜索该文件 -->  <location property=*"gov.nasa.worldwind.platform.alluser.store"* wwDir=*"WorldWindData"*/>  <location property=*"gov.nasa.worldwind.platform.user.store"* wwDir=*"WorldWindData"*/>  <location property=*"user.dir"* wwDir=*"WorldWindData"*/>  <location property=*"user.home"* wwDir=*"WorldWindData"*/>  <!-- The standard install locations are searched after the standard read locations. -->  <location property=*"gov.nasa.worldwind.platform.alluser.store"* wwDir=*"WorldWindInstalled"* isInstall=*"true"* isMarkWhenUsed=*"true"*/>  <location property=*"gov.nasa.worldwind.platform.user.store"* wwDir=*"WorldWindInstalled"* isInstall=*"true"* isMarkWhenUsed=*"true"*/>  <location property=*"user.dir"* wwDir=*"WorldWindInstalled"* isInstall=*"true"* isMarkWhenUsed=*"true"*/>  <location property=*"user.home"* wwDir=*"WorldWindInstalled"* isInstall=*"true"* isMarkWhenUsed=*"true"*/>  </readLocations>  <writeLocations>  <!-- 下列路径中首个可写的目录将作为数据文件的路径。 -->  <!-- 如果路径不存在，并且该节点的create属性为true，World Wind将会尝试去创建该目录-->  <!-- 如果创建成功则改路径才能够被用作数据存储目录 -->  <!-- 注意，只要首个可以读写的目录被用作存储目录，随后列出的目录会被忽略掉。 -->  <!-- 被选作存储目录的目录同时也会作为一个读取目录在<readLocation>中列出的目录前被访问。 -->  <location property=*"gov.nasa.worldwind.platform.alluser.store"* wwDir=*"WorldWindData"* create=*"true"*/>  <location property=*"gov.nasa.worldwind.platform.user.store"* wwDir=*"WorldWindData"* create=*"true"*/>  <location property=*"java.io.tmpdir"* wwDir=*"WorldWindData"* create=*"true"*/>  </writeLocations>  </dataFileStore> |