# HDF文件结构及基于Java的读写

## HDF简介和概述

HDF是用于存储和分发科学数据的一种自我描述、多对象文件格式。HDF是由美国国家超级计算 应用中心NCSA(全称：National Center for Supercomputing Application）创建的，为了满足各种领域研究需求而研制的一种能高效存储和分发科学数据的新型数据格式。一个HDF文件中可以包含多种类型的数据，如栅格图像数据，科学数据集，信息说明数据。这种数据结构，方便了我们对于信息的提取。例如，当我们打开一个HDF图像文件时，除了可以读取图像信息以外，还可以很容易的查取其地理定位，轨道参数，图像噪声等各种信息参数。

## HDF的数据结构特点

HDF是一个能够自我描述、多目标、用于科学数据存储和分发的数据格式。它针对存储和分发科学数据的各种要求提供解决方法。HDF设计特点为：

自我描述：一个HDF文件中可以包含关于该数据的全面信息。

多样性：一个HDF文件中可以包含多种类型的数据。例如，可以通过利用适当的HDF 文

件结构，在某个HDF文件中存储符号、数值和图形数据。

灵活性：可以让用户把相关数据目标集中一个HDF文件的某个分层结构中，并对其加以描述。同时可以给数据目标记上标记，方便查取。用户也可以把科学数据存储到多个HDF文件中。

可扩展性：在HDF中可以加入新数据模式，增强了它与其它标准格式的兼容性。

独立性：HDF是一种同平台无关的格式。HDF文件在不同平台间传递而不用转换格式。

## 建立HDF文件的原因

人们通常将不同的数据存于不同的文件中，这样在数据共享的过程中就会造成许多不必要的麻烦，HDF通过提供“总体目录结构”来处理这类问题：

1）为程序提供一种机制，使它能够直接从嵌套的文件中获得信息。

2）可以将不同类型的数据源存于同一个文件中，而这些数据源又可以同时包含其数据信息和和其它相关信息。

3）对常用数据集的格式和描述标准化。

4）鼓励所有机器和程序使用标准数据格式，产生包括具体数据的文件。

## HDF数据结构

HDF有6种主要数据类型：栅格图象,调色板，科学数据库，注释，Vdata和Vgroup。

1）栅格图象: 数据模式提供一种灵活方式存储、描述栅格图象数据，包括8bit栅格图象。

2）调色板: 也叫作彩色查对表，它提供图像的色谱。

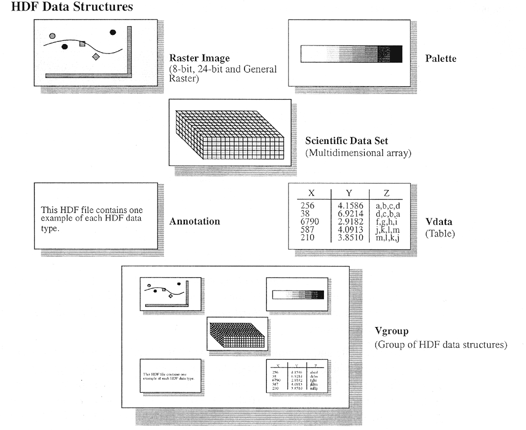
3）科学数据集: 用来存储和描述多维科学数据陈列。

4）Vdata: 是一个框架，用于存储和描述数据表。

5）HDF注释: 是文字串，用来描述HDF文件或HDF数据目标。

6）Vgroup: 是用来把相关数据目标联系起来。一个Vgroup可以含有其它Vgroup，以及数据目标。任一个HDF目标均可以包括进某个Vgroup中。

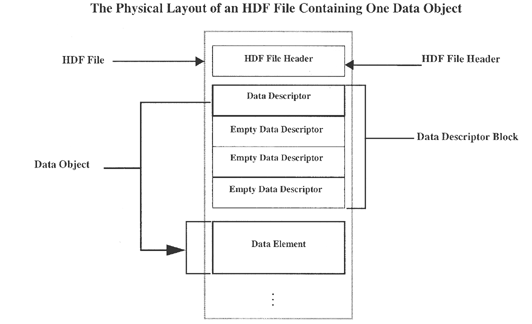
描述如下图：



## HDF文件格式

HDF格式可以被看成一本带目录的多章节书。HDF文件作为“数据书”，每一章包含不同类型的数据元素。HDF文件用“数据目录”列出数据元素。

一个HDF 文件应包括一个文件头，一个或多个描述符块，若干个数据元素。数据头是用来确定一个文件是否为HDF文件，描述符块存有数据描述符的序号。一个数据描述符和数据元素一起组成了数据对象。数据对象是HDF 的基本结构。



### 文件头

文件头占用文件的前4个字节，它由4个ASCII码形式的控制字符组成，每个字符占用一个字节。第一个控制字符是‘N’，第二个是‘C’， 第三个是‘S’，第四为‘A’，即(^N^C^S^A).

### 数据对象

一个数据对象是由一个数据描述符和一个数据元素组成。数据描述符包含了数据元素的类型、位置、尺度等信息。数据元素是实际的数据资料。HDF 这种数据组织方式可以实现HDF 数据的自我描述。

（1）数据描述符

所有的数据描述符都为12个字节长，它包含4个区域：标识符(占16-bit)，参照数（16-bit），数据偏移量（32-bit），数据长度（32-bit）。

标识符：数据标识符（tag）是用来确定数据元素区数据类型的。它是16位无符号整型值（1~65535），如果没有与其相对应的数据元素，则tag为DFTGA\_NULL(或0)。tag值的说明：1~32767——HDF 结构专用；32768~64999——可由用户定义；65000~65535——HDF 规格扩展使用。

参照数：对于HDF 文件中的每个标记符，都有唯一的一个参照数与其相对应。参照数是一个16 bit无符号整型数，在数据对象中一般是不可变的。标记符和参照数相结合确定唯一的数据对象。

数据偏移量和长度：数据偏移量是指从文件开始位置到数据元素的起始位置所包含的字节数。长度是指整个数据元素区占用字节数。数据偏移量和长度均为32-bit 无符号整型数。

（2）数据元素

数据元素是数据对象中的裸数据部分。

（3）在HDF 文件中将数据对象分组

在HDF 文件中通常将含有相关数据的数据对象分为一组。这些数据对象组称为数据集。HDF 用户采用应用界面来处理这些数据集。例如：一套8-bit的图像数据集一般有3个数据对象：一组对象用来描述数据集的成员，一组对象是图像数据，另一组对 象则用来描述图像的尺度大小。一个数据对象可以同时属于多个数据集，例如包含在一个栅格图像中的调色板对象，如果它的标识号和参照值也同时包含在另一个数 据集描述符中，那么则可以被另一个栅格图像调用。

## 读出或写入数据集的步骤

1. 获取数据集标识号

2. 指定内存数据类型

3. 指定内存数据空间

4. 指定文件数据空间

5. 指定传递特性

6. 完成对数据及需要的操作

7. 关闭数据集

8. 如有必要，关闭数据空间、数据类型和特性列表

下面介绍几个基于java读写HDF文件的方法：

（1）创建一个HDF文件

public static int H5Fcreate（java.lang.String name, int flags, int

create\_id, int access\_id）

(2)创建一简单数据空间

public static int H5Screate\_simple(int rank, long[] dims, lang[] maxdims)

(3)创建数据集

public sstatic int H5Dcreate(int loc\_id, java.lang.String name, int type\_id, int space\_id, int lcpl\_id, int dcpl\_id, int dapl\_id)

(4)向数据集中写入数据

public static int H5Dwrite(int dataset\_id, int mem\_type\_id, int mem\_space\_id, int file\_space\_id, int xfer\_plist\_id, java.lang.Object obj)

(5)打开一个已存在的HDF5文件

public static int H5Fopen(java.lang.String name, int flags, int access\_id)

(6)从数据集中读取数据

public static int H5Dread(int dataset\_id, int mem\_type\_id, int mem\_space\_id, int file\_space\_id, int xfer\_plist\_id, java.lang.Object obj)

(7)关闭数据集

H5Dclose(int dataset\_id)

(8)释放数据空间

H5Sclose(int space\_id)

(9)关闭文件

H5Fclose(int file\_id)

## 基于java的HDF文件读写实例

### 1.在myeclipse中新建Java Project HDF

File | new | Java Project，写入项目名HDF，点击Finish即可。

### 2.在项目中添加放置jar包的文件夹tools

在工程中，右击工程名，新建folder文件夹，命名为tools。将所需要的jar包jhdf5.jar拷贝到此文件夹下，然后为jar包创建路径，在jar包上右击，选择Build Path | Add to Build Path,即可在项目下看到多了一项Referenced Libraries

### 3.编辑代码实现功能

此实例展示如何从一个数据集中读写数据，首先新建一个数据集，并向其文件中写入

个数字，然后关闭文件。下一操作是打开文件，再将数字读出来。

新建一个 包Package，命名为examples.datasets，在此包中新建一Class文件，

命名为H5Ex\_D\_ReadWrite。代码如下：

**package** examples.datasets;

**import** ncsa.hdf.hdf5lib.H5;

**import** ncsa.hdf.hdf5lib.HDF5Constants;

**public** **class** H5Ex\_D\_ReadWrite {

**private** **static** String *FILENAME* = "D:/hdf5\_test.h5";

**private** **static** String *DATASETNAME* = "DS1";

**private** **static** **final** **int** *DIM\_X* = 4;

**private** **static** **final** **int** *DIM\_Y* = 7;

**private** **static** **final** **int** *RANK* = 2;

**private** **static** **void** WriteDataset() {

**int** file\_id = -1;

**int** filespace\_id = -1;

**int** dataset\_id = -1;

**long**[] dims = { *DIM\_X*, *DIM\_Y* };

**int**[][] dset\_data = **new** **int**[*DIM\_X*][*DIM\_Y*];

// Initialize data.

**for** (**int** indx = 0; indx < *DIM\_X*; indx++)

**for** (**int** jndx = 0; jndx < *DIM\_Y*; jndx++)

dset\_data[indx][jndx] = indx \* jndx - jndx;

// Create a new file using default properties.

**try** {

file\_id = H5.*H5Fcreate*(*FILENAME*, HDF5Constants.*H5F\_ACC\_TRUNC*, //创建文件，指定文件名，文件存取模式，文件创建特性列表，文件存取特性列表

HDF5Constants.*H5P\_DEFAULT*, HDF5Constants.*H5P\_DEFAULT*);

}

**catch** (Exception e) {

e.printStackTrace();

}

// Create dataspace. Setting maximum size to NULL sets the maximum

// size to be the current size.

**try** { //H5Screate\_simple新建一个数据空间并打开它等待接收数据

filespace\_id = H5.*H5Screate\_simple*(*RANK*, dims, **null**);

}

**catch** (Exception e) {

e.printStackTrace();

}

// Create the dataset. We will use all default properties for this example.

**try** {

**if** ((file\_id >= 0) && (filespace\_id >= 0)) //H5Dcreate 新建一个数据集file\_id

dataset\_id = H5.*H5Dcreate*(file\_id, *DATASETNAME*,

HDF5Constants.*H5T\_STD\_I32LE*, filespace\_id,

HDF5Constants.*H5P\_DEFAULT*, HDF5Constants.*H5P\_DEFAULT*, HDF5Constants.*H5P\_DEFAULT*);

}

**catch** (Exception e) {

e.printStackTrace();

}

// Write the data to the dataset.

**try** {

**if** (dataset\_id >= 0) //H5.H5Dwrite将dset\_data中的数据对象写入文件中

H5.*H5Dwrite*(dataset\_id, HDF5Constants.*H5T\_NATIVE\_INT*,

HDF5Constants.*H5S\_ALL*, HDF5Constants.*H5S\_ALL*,

HDF5Constants.*H5P\_DEFAULT*, dset\_data);

}

**catch** (Exception e) {

e.printStackTrace();

}

// End access to the dataset and release resources used by it.

**try** {

**if** (dataset\_id >= 0)

H5.*H5Dclose*(dataset\_id); //关闭数据集

}

**catch** (Exception e) {

e.printStackTrace();

}

**try** {

**if** (filespace\_id >= 0)

H5.*H5Sclose*(filespace\_id); //释放数据空间

}

**catch** (Exception e) {

e.printStackTrace();

}

// Close the file.

**try** {

**if** (file\_id >= 0)

H5.*H5Fclose*(file\_id);//关闭文件，写操作结束

}

**catch** (Exception e) {

e.printStackTrace();

}

}

**private** **static** **void** ReadDataset() {

**int** file\_id = -1;

**int** dataset\_id = -1;

**int**[][] dset\_data = **new** **int**[*DIM\_X*][*DIM\_Y*];

// Open file using the default properties.

**try** { //H5.H5Fopen打开一个已存在的文件

file\_id = H5.*H5Fopen*(*FILENAME*, HDF5Constants.*H5F\_ACC\_RDWR*,

HDF5Constants.*H5P\_DEFAULT*);

}

**catch** (Exception e) {

e.printStackTrace();

}

// Open dataset using the default properties.

**try** {

**if** (file\_id >= 0) //H5.H5Dopen打开数据集

dataset\_id = H5.*H5Dopen*(file\_id, *DATASETNAME*, HDF5Constants.*H5P\_DEFAULT*);

}

**catch** (Exception e) {

e.printStackTrace();

}

// Read the data using the default properties.

**try** {

**if** (dataset\_id >= 0) //H5Dread读数据到dset\_data中

H5.*H5Dread*(dataset\_id, HDF5Constants.*H5T\_NATIVE\_INT*,

HDF5Constants.*H5S\_ALL*, HDF5Constants.*H5S\_ALL*,

HDF5Constants.*H5P\_DEFAULT*, dset\_data);

}

**catch** (Exception e) {

e.printStackTrace();

}

// Output the data to the screen.

System.*out*.println(*DATASETNAME* + ":");

**for** (**int** indx = 0; indx < *DIM\_X*; indx++) {

System.*out*.print(" [ ");

**for** (**int** jndx = 0; jndx < *DIM\_Y*; jndx++)

System.*out*.print(dset\_data[indx][jndx] + " ");

System.*out*.println("]");

}

System.*out*.println();

// Close the dataset.

**try** {

**if** (dataset\_id >= 0)

H5.*H5Dclose*(dataset\_id);

}

**catch** (Exception e) {

e.printStackTrace();

}

// Close the file.

**try** {

**if** (file\_id >= 0)

H5.*H5Fclose*(file\_id);

}

**catch** (Exception e) {

e.printStackTrace();

}

}

**public** **static** **void** main(String[] args) {

H5Ex\_D\_ReadWrite.*WriteDataset*();

H5Ex\_D\_ReadWrite.*ReadDataset*();

}

}