O'REILLY®

Python和HDF5大数据应用

Python and HDF5

[美] Andrew Collette 著
胡世杰 译

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

O'REILLY®

Python 和 HDF5 大数据应用

[美] Andrew Collette 著
胡世杰 译

人民邮电出版社
北京

图书在版编目（CIP）数据

Python和HDF 5大数据应用 / （美）科莱特(Collette,A.）著 ; 胡世杰译. -- 北京 : 人民邮电出版社, 2016.2
ISBN 978-7-115-41294-2

Ⅰ. ①P… Ⅱ. ①科… ②胡… Ⅲ. ①软件工具－程序设计 Ⅳ. ①TP311.56

中国版本图书馆CIP数据核字(2016)第012025号

版权声明

◆ 著 ［美］Andrew Collette
译 胡世杰
责任编辑 陈冀康
责任印制 张佳莹 焦志炜

◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路 11 号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：787×1000 1/16
印张：8.25
字数：143 千字 2016 年 2 月第 1 版
印数：1－2 500 册 2016 年 2 月河北第 1 次印刷

著作权合同登记号 图字：01-2013-8990 号

定价：39.00 元

读者服务热线：(010)81055410 印装质量热线：(010)81055316
反盗版热线：(010)81055315

内容提要

随着 Python 应用领域的拓展，越来越多的人将 Python 用于处理大型数值数据集，使用标准格式来进行数据的存储和通信也显得越来越重要，而 HDFS 也正迅速成为人们存储科学数据的选择。

本书向任何有 Python 数据分析基本背景的人介绍如何在 Python 下使用 HDF5。本书将侧重于 HDF5 的本地功能集，而不是 Python 的高层抽象。熟悉 Python 和 NumPy 的读者，更容易阅读和掌握本书的内容。

本书适合有一定基础的 Python 开发者，尤其适合要使用 Python 开发数据存储和处理等相关应用的读者阅读参考。

序

过去的几年里，Python 已经和 IDL 或 MATLAB 一样，成为科学分析领域值得信赖的选择。它拥有稳健的核心模块，可用于处理数值数组（NumPy）、分析（SciPy）以及绘图（matplotlib），同时还拥有十分丰富的专用模块。它能帮助人们减少编写科学代码的工作量，同时还能提高计算结果的质量。

已经有越来越多的人将 Python 用于大型数值数据集处理，使用标准格式来进行数据的存储和通信也显得越来越重要。国家超级电脑应用中心（National Center for Supercomputing Applications，NCSA）开发的“层次性数据格式”（Hierarchical Data Format，HDF）的最新版本 HDF5 正迅速成为人们储存科学数据的选择。很多使用（或有意使用）HDF5 的研究人员也被 Python 的易用性和快速开发能力吸引。

本书向所有有 Python 数据分析基本背景的人介绍如何在 Python 下使用 HDF5。本书假定读者只熟悉 Python 和 NumPy。本书将侧重于 HDF5 的本地功能集，而不是 Python 的高层抽象，以尽力确保在本书建议下创建的 HDF 文件可移植。

最后，本书将尽力同时支持 Python 2 和 Python 3 的用户。本书所有的示例都以 Python 2 写就，任何有可能导致误会的区别都将在文本中特别注明。

本书中用到的排版规范

本书使用下列排版规范：

斜体

用于新的名词，URL，E-mail 地址，文件名，文件扩展名。

等宽

用于程序列印以及正文内的程序元素，如变量或函数名、数据库、数据类型、环境变量、语句和关键字。

等宽加粗

用于命令或其他需要用户输入的文字。

等宽斜体

用于需要被替换成用户提供的值或根据上下文决定的值。

提示

该图标表示一个提醒、建议，或注意。

警告

该图标表明一个警告。

使用代码示例

本书帮助你完成你的任务。一般来说，只要是本书提供的示例代码，你都可以在你的程序和文档中使用它们，而不需要联系我们获得许可，除非你打算将大段代码重新发售。举例来说，你写一个程序用到了本书代码不需要得到许可。但如果你出售或发布一个包含 O'Reilly 书籍中示例代码的 CD-ROM 却需要得到许可。引用本书段落及书中例子回答问题不需要得到许可。将本书示例代码大量并入你的产品文档却需要得到许可。

我们会感激但不强求引用注明出处。通常包括书名、作者、出版商和 ISBN。比如："Book Title by Some Author (O'Reilly).Copyright 2012 Some Copyright Holder, 978-0-596-xxxx-x."

如果你觉得你对代码示例的使用不属于上述的免许可范围，请通过 permissions@oreilly.com 联系我们。

在线 Safari®书籍

 在线 Safari 书籍是一个云端数字图书馆，以书籍和视频的形式提供来自世界顶级的技术作者和商业作者的专业内容。

技术专家、软件开发者、网页设计者和商业创新专家将在线 Safari 书籍作为他们研究、解决问题、学习和培训认证的首选资源。

在线 Safari 书籍为组织、政府和个人提供一批产品组合和定价项目。订阅者可以通过一个全文搜索数据库访问出版商提供的成千上万的书籍、培训视频及正式出版前的手稿。这些出版商包括但不限于：O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional,Microsoft Press, Sams, Que, Peachpit Press, Focal Press,

Cisco Press, JohnWiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FTPress, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology。更多信息请访问我们的网站 https://www. safaribooksonline.com/。

如何联系我们

请将对本书的评论和问题发送给出版商：

美国：

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）

奥莱利技术咨询（北京）有限公司

我们为本书制作了一个网页，列出了勘误表、代码示例及各种附加信息。你可以通过这个网址访问：http://www.oreilly.com/catalog/<catalog page>。

评论或技术问题可发电子邮件到 bookquestions@oreilly.com。

更多信息请访问 http://www.oreilly.com 网站。

我们的 Facebook: http://facebook.com/oreilly

我们的 Twitter: http://twitter.com/oreillymedia

我们的 YouTube: http://www.youtube.com/oreillymedia

特别鸣谢

我要感谢 HDF 组织的 Quincey Koziol、Elena Pourmal、Gerd Heber 以及其他人在 Python 社区上对 HDF5 的支持。感谢来自 Eli Bressert 和 AnthonyScopatz 等评审者的评论和来自 O'Reilly 编辑 Meghan Blanchette 的指导，他们的奉献令本书得益匪浅。感谢 Darren Dale 以及其他很多人在 h5py 项目上的贡献。感谢 Francesc Alted，Antonio Valentino 和 PyTables 的共同作者们第一次将 HDF5 带入 Python 的世界。最后，我还要感谢 UCLA（加利福尼亚大学洛杉矶分校）基础等离子研究机构的 Steve Vincena 和 Walter Gekelman，我曾在那里跟他们一起共事，并第一次接触到大规模科学数据集。

目录

第 1 章

简介

我刚毕业那会遇到过一个严重的问题——一部国家公认的等离子体研究设备花了整整一周时间收集的上千万个数据的值不太对劲。

比正常情况小了约 *40 个数量级*。

我跟我的咨询师挤在他的办公室，在一台崭新的 G5 Mac Pro 上运行我们的可视化软件，试图搞明白哪里出了问题。从机器中获得的数据是正确的，实验所使用的数字转换器提交的原始数据看上去没有问题。我在 Thinkpad 笔记本上用 IDL 语言编写了一个巨大的脚本将原始数据转换成可视化软件能够识别的文件。这些文件的格式十分简单：一个简短的定长头部后面加上一堆二进制浮点数据。我还另外又花了一个小时写了一个程序来验证这些文件，它们也没问题。但当我将所有这些在 IDL 中看上去如此优雅的数据导入可视化软件以后，它们看上去就像是一锅粥，毫无特色、杂乱无章，所有的值大约都只有 10^{-41} 左右。

最后我们发现了问题所在：数字转换器和我的 Thinkpad 使用了“little-endian”格式，而 G5 Mac 使用了“big-endian”格式。一台机器输出的原始数据值无法被另一台机器正确地读入，反过来也一样。当时我所有想法中最有礼貌的一句是：*这也太笨了*。哪怕最后发现此类问题是如此司空见惯以至于 IDL 专门提供了一个 SWAP_ENDIAN 函数来处理也并没有令我的情绪变得更好。

在此之前我从不关心数据是如何存储的。这个事件以及其他一些类似事件改变了我的想法。作为一名科学家，我最终意识到，我们不仅需要选择数据的组织和存储，同时也需要选择数据的通信方式。设计优雅的标准格式不仅让每个人的生活变得简单（消除了上面愚蠢而又浪费时间的“endian”问题），而且也使得全世界都能共享这些数据。

1.1 Python 和 HDF5

在 Python 的世界里，人们在数值类型大数据的存储机制上进行选择时，迅速对层次性数据格式第 5 版（Hierarchical Data Format version 5，HDF5）达成了共识。当数据量越来越大的时候，数据的组织就变得越来越重要。命名数据集（第 3 章）、层次性分组（第 5 章）和用户自定义元数据“特征”（第 6 章）等 HDF5 特性对于数据分析的过程极为必要。

HDF5 这种结构化的自我描述格式跟 Python 相辅相成。目前 HDF5 已经有两大开发成熟、功能丰富的 Python 接口模块 h5py 和 PyTables，在两者之上还有许多为特定用途开发的小型封装模块。

1.1.1 数据和元数据的组织

这是一个利用 HDF5 的结构化能力帮助应用程序的简单例子。不要太担心文件结构和 HDF5 使用 API 等方面的细节，后续章节自会一一解释。就把这个当成是一次 HDF5 尝鲜。如果你想要运行这个例子，你需要 Python 2 并安装 NumPy（第 2 章）。

假设我们有一个 NumPy 数组，它代表了某次实验获取的一些数据：

```
>>> import numpy as np
>>> temperature = np.random.random(1024)
>>> temperature
array([ 0.44149738,  0.7407523 ,  0.44243584, ...,  0.19018119,
        0.64844851,  0.55660748])
```

假设这些数据来自某个气象站十秒一次的温度采样。为了让这些数据有意义，我们还需要记录采样的时间间隔“delta-T”。目前我们把它放到一个 Python 变量中。

```
>>> dt = 10.0
```

同时我们还需要记录第一个数据获取的时间起点，以及这些数据来自 15 号气象站：

```
>>> start_time = 1375204299  # in Unix time
>>> station = 15
```

我们可以用一个简单的 NumPy 内建函数 `np.savez` 来将这些数据存入磁盘，该函数将每个数据以 NumPy 数组的格式保存并打包进一个指定名字的 zip 文件：

```
>>> np.savez("weather.npz", data=temperature, start_time=start_time, station=station)
```

我们可以用 np.load 从文件中获取这些数据：

```
>>> out = np.load("weather.npz")
>>> out["data"]
array([ 0.44149738,  0.7407523 ,  0.44243584, ...,  0.19018119,
        0.64844851,  0.55660748])
>>> out["start_time"]
array(1375204299)
>>> out["station"]
array(15)
```

目前为止一切顺利。但如果每个气象站的数据不止一组怎么办？比如说还要记录一组风速数据？

```
>>> wind = np.random.random(2048)
>>> dt_wind = 5.0   # Wind sampled every 5 seconds
```

再假设我们有不止一个气象站。也许我们可以引入某种命名规范，比如“wind_15”作为 15 号气象站的风速数据，“dt_wind_15”作为采样时间间隔。又或许我们可以使用多个文件……

作为对比，让我们看看如果用 HDF5 来存储会是怎样：

```
>>> import h5py
>>> f = h5py.File("weather.hdf5")
>>> f["/15/temperature"] = temperature
>>> f["/15/temperature"].attrs["dt"] = 10.0
>>> f["/15/temperature"].attrs["start_time"] = 1375204299
>>> f["/15/wind"] = wind
>>> f["/15/wind"].attrs["dt"] = 5.0
---
>>> f["/20/temperature"] = temperature_from_station_20
---

(and so on)
```

这个例子演示了 HDF5 的两个杀手级特性：层次性分组和特征。组就像文件系统里的目录，使你可以将相关的数据集保存在一起。本例将来自同一个气象站的温度和风速分别保存在名为“/15”和“/20”的组里。特征允许你*在数据上直接附加*描述性的元数据。一旦你将这个文件给其他同事，他们可以轻易发现这些信息并明白这些数据的意义：

```
>>> dataset = f["/15/temperature"]
>>> for key, value in dataset.attrs.iteritems():
...     print "%s: %s" % (key, value)
dt: 10.0
start_time: 1375204299
```

1.1.2 大数据复制

人们正在越来越多地将 Python 用于大数据集的快速可视化项目，在大规模计算中将 Python 作为一种高层粘合性语言来协助那些编译型语言如 C 和 FORTRAN。现

在一个数据集动不动就是上千 GB 甚至 TB 的数据需要处理，HDF5 自身最大可以支持 EB 的规模。

大多数的机器不可能将如此大规模的数据集直接导入内存。HDF5 最大的优点之一在于支持子集分片和部分 I/O。让我们看一下之前创建的拥有 1024 个元素的“temperature”数据集：

```
>>> dataset = f["/15/temperature"]
```

这里的 dataset 对象是一个 HDF5 数据集的代理对象。它支持数组切片操作，NumPy 用户可能会觉得很熟悉：

```
>>> dataset[0:10]
array([ 0.44149738,  0.7407523 ,  0.44243584,  0.3100173 ,  0.04552416,
        0.43933469,  0.28550775,  0.76152561,  0.79451732,  0.32603454])
>>> dataset[0:10:2]
array([ 0.44149738,  0.44243584,  0.04552416,  0.28550775,  0.79451732])
```

记住，真正的数据保存在磁盘上，切片操作会去寻找合适的数据并读入内存。这种形式的切片利用了 HDF5 底层的子集分片功能，所以非常迅速。

HDF5 另一个伟大之处在于你可以控制存储的分配。当你创建了一个全新的数据集，除了一些元数据以外它不会占用任何空间，默认情况下仅当你真的需要写入数据时才会占用磁盘上的空间。

比如下面这个 2TB 的数据集，你可以在几乎任何计算机上创建出来：

```
>>> big_dataset = f.create_dataset("big", shape=(1024, 1024, 1024, 512),
dtype='float32')
```

虽然还没有为其真正分配任何存储，但整个数据集的空间对我们都是可用的。我们可以在数据集的任何地方写入，而磁盘上只会占用必要的字节：

```
>>> big_dataset[344, 678, 23, 36] = 42.0
```

在存储非常昂贵的情况下，你甚至可以对数据集进行透明压缩（第 4 章）：

```
>>> compressed_dataset = f.create_dataset("comp", shape=(1024,), type='int32',
compression='gzip')
>>> compressed_dataset[:] = np.arange(1024)
>>> compressed_dataset[:]
array([   0,    1,    2, ..., 1021, 1022, 1023])
```

1.2 HDF5 到底是什么

HDF5 是一种*存储相同类型数值的大数组*的机制，适用于可被*层次性组织*且数据集需要被*元数据*标记的数据模型。

它跟 SQL 风格的关系型数据库区别相当大，HDF5 在组织结构方面有一些特殊的技巧（第 8 章中有一个例子）。如果你需要在多个表上保持关系，或者想要在数据上进行 JOIN，那么一个关系型数据库可能更适合你。又或者你需要在一台没有安装 HDF5 的机器上读取一个小型的 1 维数据集，那么 CSV 这样的文本格式是更合理的选择。

但如果你需要处理多维数组，对性能有非常高的要求，需要在数据集上支持子集分片和部分 I/O，需要用特征来给数据集做标记，对关系型特性没有要求，那么 HDF5 就是完美的选择。

那么说到底，“HDF5”究竟是指什么？我确信它包含下面 3 点：

1. 一种文件规格及相关的数据模型；
2. 一个可被 C、C++、Java，Python 以及其他语言使用的 API 标准库；
3. 一个软件生态系统，由使用 HDF5 的客户程序以及 MATLAB、IDL 和 Python 等“分析平台”组成。

1.2.1 HDF5 文件规格

你已经在上面的例子见到 HDF5 数据模型的三大要素：

数据集：一种数组型对象，在磁盘上保存数值类型的数据；

组：层次性容器，可以包含数据集和子组；

特征：自定义元数据信息，可被附加在数据集（以及组!）上。

用户可以使用这些基本抽象构建适合自己问题域的应用格式。比如，我们之前的气象站代码为每个气象站分了一个组，为每个测量参数分配一个数据集，并附加了一些特征以描述数据集的额外信息。这种统一使用“格式内格式”来决定如何用组、数据集和特征来保存信息的方式在实验室或者其他机构中是非常普遍的。

既然 HDF5 处理一切如“endian”的跨平台问题，数据的分享就只需要对组、数据集和特征进行简单操作并获得结果。由于文件是*自我描述*的，你甚至不需要了解应用格式就可以从文件中获取数据。你只需打开文件并浏览其内容：

```
>>> f.keys()
[u'15', u'big', u'comp']
>>> f["/15"].keys()
[u'temperature', u'wind']
```