Distribution of Data Through OCAF Tree

OCAF 数据的分配

eryar@163.com

一、作者注 *Annotation*

本文档主要用于说明 ***OCAF***（***Open CASCADE Application Framework***）中数据模型的选择问题。另外，以一个例子来说明 ***OCAF*** 的标准属性的使用和创建 ***OCAF*** 新的属性。

作者假设读者已经熟悉 ***OCAF*** 的一些基础知识。

二、简介 *Introduction*

OCAF（***Open CASCADE Application Framework***）是为快速程序开发（***Rapid Application Development***）而提供的一些类。***OCAF*** 实现如下功能：撤销、重做、复制、剪切、粘贴、保存文档、打开文档等等。

OCAF 基于标签-属性（***Label-Attribute***）模型。标签组成树。在 ***OCAF*** 文档中，属性用于保存用户的数据，且属性绑定在标签上。

本文档描述了数据保存在 ***OCAF*** 文档中应考虑的注意事项：

1. 是使用标准属性还是创建自定义的新的属性？
2. 如何优化数据的存储来提高时间和空间上效率，和程序的运行速度？

三、概述 *Description*

当开始设计基于 ***OCAF*** 的程序时，对于数据的存储通常要面临这样的选择：是使用标准属性还是根据需要自己创建新的属性？在给出这个问题的答案之前，先简要回顾一下 ***OCAF*** 的标准属性：

所有基本数据类型都在 ***OCAF*** 中表示为标准属性：

- 整数 ***Integer***；
- 实数 ***Double***；
- 字符串 ***String***；
- 整数数组 ***Array of integer values***；
- 实数数组 ***Array of double values***；
- 字符串数组 ***Array of string values***；
- 拓扑形状 ***Topological shapes***；

除了拓扑形状属性外，其它属性都在 ***Toolkit TKLCAF*** 的 ***Package TDataStd*** 中：

- 整数属性：***TDataStd_Integer***；
- 实数属性：***TDataStd_Real***；
- 字符串属性：***TDataStd_AsciiString***、***TDataStd_Expression***；
- 整数数组：***TDataStd_IntegerArray***；
- 实数数组：***TDataStd_RealArray***；
- 字符串数组：***TDataStd_ExtStringArray***；

细心的读者可能发现没有 ***Boolean*** 类型，因为通常用 ***Integer*** 来代替了。

所以使用 ***OCAF*** 的标准属性可以用来定义任何模型。但是这样定义对内存的使用、程

序运行速度、及使用的方便性上来说是否合适？还是要根据特定的模型具体分析。

OCAF 只有一个限制：每种属性一个标签只能包含一个。即只能给一个标签定义一个整数属性，一个实数属性等。所以有必要考虑程序数据树的设计。例如：一个标签有几个实数值，是把这几个实数值分别存储到子标签中还是使用一个实数数组来存储，都是需要事先考虑周全。

考虑同一个模型在 ***OCAF*** 中不同的存储方法，并分析每种方法的优缺点。

四、不同方法的比较与分析 *Comparison and Analysis of Approaches*

这里描述了定义模型的两种方式：一种是基于 ***OCAF*** 标准属性；另一种是创建新的属性。

A load is distributed through the shape。量度由坐标（***x, y, z***）来定义。***Load*** 通过由每点的局部坐标系投影到 ***X***-、***Y***-和 ***Z***-轴上来表示。一个由局部坐标系转换到世界坐标系的矩阵也可能需要，但这是可选的。

所以，每个点上有 15 个数值需要被存储。若这样的点有 100000 个，则在 ***OCAF*** 文档中需要存储 1500000 个数值。

第一种方式是使用 ***OCAF*** 的标准属性。使用标准属性也有几种不同的方式：

1. 使用实数数组属性来存储这 1500000 个数，并将这个实数数组绑定到一个标签；如下图 1 所示。
2. 将每组 15 个值用一个实数数组属性存储，并将每个实数数组绑定到每个标签上；如下图 2 所示。
3. 使用实数数组属性来存储每个点的（***X***、***Y***、***Z***）3 个坐标值，并将其绑定到标签；局部坐标系到世界坐标系的投影轴的 3 个值也使用实数数组属性来存储，并将其绑定到子标签上；同样地，变换坐标系的 9 个值也用实数数组属性存储，并也将其绑定到子标签上；如下图 3 所示。
4. 当然，其它方法也是可行的。

图 1 使用一个数组属性来存储所有数据

第一种方法是把所有数据都用一个实数数组属性来存储。这种方法的优点是节省了初始内存，并且易于实现。但是数据的访问很不方便，还需要为取得相应数据来定个专门的类来处理。如果程序需要对这些值进行编辑，这就意味着整个数组在每次编辑时都需要备份，这会导致内存的使用快速增加。所以，这种方式可能只能用于数据不需要编辑的程序。

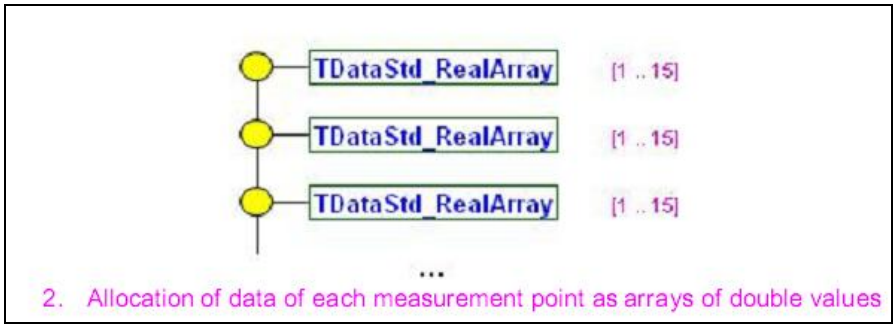

图 2 为每个数据点使用一个数组属性及一个标签

考虑每个数据点作为一个标签的第二种方式。这种情况需要创建 100000 个标签，每个标签上绑定了一个包含 15 个实数的实数数组属性。如图 2 所示。

现在数据的编辑要安全些，且内存的使用也在考虑之中。每个数据点的编辑（任意值：***point coordinate, load, and so on***）的备份也只是一个小的实数数组。但是这种结构需要更多的内存空间，因为使用很多的标签和属性。

另外，数据的访问还是不方便，也需要一个类来处理数据的访问。

图 3 存储数据到不同的数组属性中

第三种方式的数据存储如图 3 所示。在这种情况下引入了子标签，数据的访问变得容易了，不需要为此专门设计一个类来访问所需要的数据了。

这种方式一方面为属性分配更多的内存；另一方面，在数据编辑需要备份时节省了内存。所以，若程序中所有数据都可以修改，则这种方式会更好。

图 4 使用自定义属性

在下结论之前，考虑使用自定义属性来表示这个模型的方式。

例如，可以将每组 15 个值作为一个自定义属性绑定到标签上。与使用标准属性的第三种方式相比，自定义属性的方法使用更少的内存，因为只使用了一个属性而不是三个属性。

使用标准属性的第二种方式还有些不足：当编辑数据时，每个点的所有数据都被备份，不管有些数据有没有被修改。

当有些不可编辑的数据，最好将之与可编辑的数据分开存储。数据编辑备份时将不会对不可编辑的数据进行备份，这样在编辑时内存就会有所减少了。

五、结论 *Conclusion*

当确定选择使用哪种数据模型时，需要考虑程序响应时间、内存分配与内存在事务处理中的使用。大部分模型只使用 ***OCAF*** 的标准属性已经足够。其它需要特别处理的模型需要定义 ***OCAF*** 新属性。