# 父文档引用

# 父文档引用 模式用一个文档来表示树的一个节点。每一个文档除了存储节点的信息，同时也保存该节点父节点文档的id值。

# Tree data model for a sample hierarchy of categories.

db.parent.insert( { \_id: "MongoDB", parent: "Databases" } )

db.parent.insert( { \_id: "dbm", parent: "Databases" } )

db.parent.insert( { \_id: "Databases", parent: "Programming" } )

db.parent.insert( { \_id: "Languages", parent: "Programming" } )

db.parent.insert( { \_id: "Programming", parent: "Books" } )

db.parent.insert( { \_id: "Books", parent: null } )

查询父节点

db.parent.findOne( { \_id: "MongoDB" } ).parent

对parent字段建立索引，方便查询。

db.parent.ensureIndex({‘parent’:1})

db = connect2MongoDB().test

table = db.parent

def query(books, level):

boks = table.find({'parent':books})

if boks is not None:

print '|-' \* level, books

level = level + 1

for b in boks:

query(b.get('\_id'), level)

遍历

# 子文档应用

db.categories.insert( { \_id: "MongoDB", children: [] } )

db.categories.insert( { \_id: "dbm", children: [] } )

db.categories.insert( { \_id: "Databases", children: [ "MongoDB", "dbm" ] } )

db.categories.insert( { \_id: "Languages", children: [] } )

db.categories.insert( { \_id: "Programming", children: [ "Databases", "Languages" ] } )

db.categories.insert( { \_id: "Books", children: [ "Programming" ] } )

# 查询：

db.children.findOne( { \_id: "Databases" } ).children

# 深度遍历：

def **query**(books,level):

boks = table.find\_one({*'\_id'*:books}).get(*'children'*)

length = len(boks)

print *'|-'*\*level,books

for i in range(length):

query(boks[i],level = level+1)

# 同时在children 上面建立索引。

# db.children.ensureIndex({children:1})

# 祖先数组

# 祖先数组 模式用一个文档来表示树的一个节点。每一个文档除了存储节点的信息，同时也存储了对父文档及祖先文档的id值。

db.categories.insert( { \_id: "Databases",

ancestors: [ "Books", "Programming" ],

parent: "Programming" } )

db.categories.insert( { \_id: "Programming",

ancestors: [ "Books" ],

parent: "Books" } )

db.categories.insert( { \_id: "Books", ancestors: [ ], parent: null } )

# 物化路径

# 物化路径（Materialized Paths） 模式用一个文档保存树结构中的一个节点。除此之外, 这些文档也会保存一个从根节点到当前节点完整路径节点的id组成的一个字符串，称之为 物化路径。虽然 物化路径 本身需要一些额外的字符串处理，但是这种模式在和路径相关的操作方面会更加灵活，如基于部分路径的节点搜索。

db.categories.insert( { \_id: "MongoDB",

path: ",Books,Programming,Databases," } )

# 使用正则表达式查询：

 你可以在 path 字段上使用正则表达式来查询 Programming 的所有子代节点： db.path.find( { path: /,Programming,/ } )

# 嵌套集合

# 嵌套集合 模式对整个树结构进行一次深度优先的遍历。遍历时候对每个节点的压栈和出栈作为两个不同的步骤记录下来。然后每一个节点就是一个文档，除了节点信息外，文档还保存父节点的id以及遍历的两个步骤编号。压栈时的步骤编号保存到 left 字段里， 而出栈时的步骤编号则保存到 right 字段里。

# Example of a hierarchical data. The numbers identify the stops at nodes during a roundtrip traversal of a tree.

db.categories.insert( { \_id: "Books", parent: 0, left: 1, right: 12 } )

db.categories.insert( { \_id: "Programming", parent: "Books", left: 2, right: 11 } )

db.categories.insert( { \_id: "Languages", parent: "Programming", left: 3, right: 4 } )

db.categories.insert( { \_id: "Databases", parent: "Programming", left: 5, right: 10 } )

db.categories.insert( { \_id: "MongoDB", parent: "Databases", left: 6, right: 7 } )

db.categories.insert( { \_id: "dbm", parent: "Databases", left: 8, right: 9 } )