## 测试文档

# 单元测试：CoreService

* **函数名称：AddNodeTest()**

增加节点测试： 测试CoreService增加Experiment和Signal

1. root下添加同名节点

测试root下能否添加同名的Experiment节点

预期结果：抛出异常

1. root下添加信号节点

测试root下能否添加Signal节点

预期结果：抛出异常

1. 信号节点下添加信号节点

测试Signal节点下能否再添加Signal节点

预期结果：抛出异常

1. experiment节点下添加同名信号节点

测试Experiment节点下能否添加同名的信号节点

预期结果：抛出异常

1. 父节点不存在时添加信号节点

测试在一个不在数据库里的父节点下添加信号节点

预期结果：抛出异常

1. 正确添加节点并测试

预期结果：正确

* **函数名称：DuplicateTest()**

复制节点：复制单个节点和级联复制节点

1：父节点下有同名节点

预期结果：抛出异常

2：复制节点

3：递归复制节点

* 函数名称：DeleteTest()

删除节点：删除单个节点和级联删除节点

1：有子节点时，需要指定递归才能删除此节点

预期结果：抛出异常

2：删除节点不存在

预期结果：抛出异常

3：普通删除测试

4：递归删除测试

* 函数名称：QueryByNameTest()

查询节点：

1：通过名字查找

2：父节点不存在时，报错

* 函数名称：ChangePathTest()

更改节点路径：

1：移动父节点到子节点时报错

2：正确更改节点路径检验

* 函数名称：RenameTest()

重命名：

1：重名测试

有相同名字时报错

2：正确测试

* 函数名称：UpdateTest()

修改节点：修改节点属性

1：修改id不可更改的属性

预期结果：抛出异常

2：修改信号节点的属性

3：修改实验节点的属性

.

# 单元测试：StorageEngine

* 函数名称：AppendData1DTest()

添加一维数据，并查找测试

数据格式：0-100,10个payload

维度： 单维数据：0-99 N为维度

读取起始位置：｛15｝ ，读取数量 ：｛10｝

实际读取请求数量：30

预期结果：expectedSample = new double[] {15,16,17,18,19,20,21,22,23,24 };

* 函数名称：AppendData2DTest()

添加二维数据，并查找测试

数据格式：

维度：0-100 单维数据：N.00-N.99 N为维度

读取起始位置：｛5,6｝ ，读取数量 ：｛4,4｝ 读取间隔：｛5,2｝

实际读取请求数量：100

预期结果：var expectedSample = new double[4, 4]

{

{ 5.06,5.08,5.10,5.12 },

{ 10.06,10.08,10.10,10.12 },

{ 15.06,15.08,15.10,15.12 },

{ 20.06,20.08,20.10,20.12 }

};

* 函数名称：AppendData3DTest() （需要更改）

添加三维数据，并查找测试

数据格式：

维度：｛0,0｝-｛7,7｝ 单维数据：0-7

读取起始位置：｛2,1,3｝ ，读取数量 ：｛2,2,3｝

实际读取请求数量：15

预期结果：var expectedSample = new double[2, 2, 3]

{

{ { 213,214,215 }, { 223,224,225 } },

{ { 313,314,315 }, { 323,324,325 } }

};

* 函数名称：DeleteDataTest()

删除数据：

1：级联删除

2：删除信号的数据

预期结果：查找时异常

* 函数名称：ReadData1DTest()

读取一维数据：

数据格式：

维度： 单维数据：0-99

1：读取数据

读取起始位置：｛46｝ ，读取数量 ：｛60｝

实际读取请求数量：10

预期结果：46-55

再次读取请求数量：10

预期结果：56-65

再次读取请求数量：30

2：超出边界测试

再次读取请求数量：10

预期结果：异常

3：读取数据并使用factor

读取起始位置：｛46｝ ，读取数量 ：｛5｝ 读取间隔：｛10｝

实际读取请求数量：10

预期结果：{ 46,56,66,76,86};

* 函数名称：ReadData2DTest()

读取二维数据：使用和不使用factor

数据格式：

维度：｛0｝，｛1｝ 单维数据：0-99

1：读取测试

读取起始位置：｛0,46｝ ，读取数量 ：｛2,60｝

实际读取请求数量：50

预期结果：46-95

2 : 跨维度测试

读取起始位置：｛0,46｝ ，读取数量 ：｛2,50｝

实际读取请求数量：60

预期结果：46-95，46-55

再次请求数量：50

预期结果：56-95

3：边界测试

读取起始位置：｛0,40｝ ，读取数量 ：｛2,60｝

实际读取请求数量：60

预期结果：40-99

再次请求数量：60

预期结果：40-99

4：跨维度读取数据并使用factor

读取起始位置：｛0,5｝ ，读取数量 ：｛2,10｝ 读取间隔：｛1,5｝

实际读取请求数量：5

预期结果：{ 5,10,15,20,25}

* 函数名称：ReadData3DTest()

读取三维数据：使用和不使用factor

数据格式：

维度：｛0,0｝-｛7,7｝ 单维数据：[0-7]\*10

1：跨纬度读取数据，使用factor

var cursor1 = storageEngine.GetCursorAsync<double>(sig11.Id, new List<long> { 2, 1, 3 }, new List<long> { 2, 2, 3 }, new List<long> { 2, 2, 2 }).Result;

List<double> result11 = cursor1.Read(3).Result.ToList();

double[] assertresult11 = { 213, 215, 217 };

CollectionAssert.AreEqual(assertresult11, result11);

List<double> result12 = cursor1.Read(4).Result.ToList();

double[] assertresult12 = { 233, 235, 237, 413 };

CollectionAssert.AreEqual(assertresult12, result12);

var cursor2 = storageEngine.GetCursorAsync<double>(sig11.Id, new List<long> { 2, 1, 3 }, new List<long> { 2, 2, 3 }, new List<long> { 2, 2, 3 }).Result;

List<double> result21 = cursor2.Read(2).Result.ToList();

double[] assertresult21 = { 213, 216 };

CollectionAssert.AreEqual(assertresult21, result21);

List<double> result22 = cursor2.Read(3).Result.ToList();

double[] assertresult22 = { 219, 233, 236};

CollectionAssert.AreEqual(assertresult22, result22);

var cursor3 = storageEngine.GetCursorAsync<double>(sig11.Id, new List<long> { 2, 1, 3 }, new List<long> { 2, 2, 10 }, new List<long> { 2, 2, 6 }).Result;

List<double> result31 = cursor3.Read(3).Result.ToList();

double[] assertresult31 = { 213, 219, 215 };

CollectionAssert.AreEqual(assertresult31, result31);

List<double> result32 = cursor3.Read(4).Result.ToList();

double[] assertresult32 = { 211, 217, 213, 219 };

CollectionAssert.AreEqual(assertresult32, result32);

List<double> result33 = cursor3.Read(4).Result.ToList();

double[] assertresult33 = { 215, 211, 217, 233 };

CollectionAssert.AreEqual(assertresult33, result33);

2：边界测试，使用factor

var cursor4 = storageEngine.GetCursorAsync<double>(sig11.Id, new List<long> { 2, 1, 2 }, new List<long> { 2, 2, 6 }, new List<long> { 2, 2, 2 }).Result;

List<double> result41 = cursor4.Read(5).Result.ToList();

double[] assertresult41 = { 212, 214, 216,218,210 };

CollectionAssert.AreEqual(assertresult41, result41);

List<double> result42 = cursor4.Read(5).Result.ToList();

double[] assertresult42 = { 212, 232, 234, 236,238 };

CollectionAssert.AreEqual(assertresult42, result42);

3：跨维度读取数据，不使用factor

var cursor5 = storageEngine.GetCursorAsync<double>(sig11.Id, new List<long> { 2, 1, 8}, new List<long> { 2, 2, 5 }).Result;

List<double> result51 = cursor5.Read(3).Result.ToList();

double[] assertresult51 = { 218, 219, 210 };

CollectionAssert.AreEqual(assertresult51, result51);

List<double> result52 = cursor5.Read(4).Result.ToList();

double[] assertresult52 = { 211, 212, 228, 229 };

CollectionAssert.AreEqual(assertresult52, result52);

* 函数名称：DuplicateDataTest()

复制数据：

1：新的节点不能和原来的子节点同名

2：递归拷贝节点测试

3：递归拷贝节点和数据测试

### 集成测试：CoreAPI

* 函数名称：LoadPlugInTest()

PlugIn的动态加载测试（IdQueryPlugIn，PathQueryPlugIn，FixedWaveDataTypePlugin）

1： 测试加载IDQueryPlugIn

2： 测试加载PathQueryPlugIn

3： 测试加载DataPlugIn

* 函数名称：IdQueryTest()

测试id的查询方法

* 函数名称：PathQueryTest()

测试Path的查询方法

* 函数名称：AppendDataTest ()

IntFixedIntervalWaveSignal的写入测试

* 函数名称：ReadDataTest ()

IntFixedIntervalWaveSignal的读取测试