.NET 2.1.1 Driver for MongoDB

MongoDB3.0版本后，.NET的驱动更新了2.0版本，全面100%从底层支持异步（Not only is the API async on the surface, the driver is 100% async down to the lowest level.）。

新的API基于最上层的三个接口：[**IMongoClient**](http://mongodb.github.io/mongo-csharp-driver/2.0/reference/driver/connecting/#mongo-client),[**IMongoDatabase**](http://mongodb.github.io/mongo-csharp-driver/2.0/reference/driver/connecting/#mongo-database),[**IMongoCollection**](http://mongodb.github.io/mongo-csharp-driver/2.0/reference/driver/connecting/#mongo-collection)。新的应用模式是先创建一个MongoClient实例，调用GetDatabase得到IMongoDatabase实例，然后调用GetCollection得到IMongoCollection实例。大多数操作会用IMongoCollection实例，少数操作用IMongoClient 或者IMongoDatabase层的接口。

新的IMongoCollection接口比原来的MongoCollection 类的方法少很多，更方便使用.一方面这通过利用Options类来可选参数，这样我们不必要重载多个方法只为了提供不同参数的方法。另一方面，通过引进fluent API 来实现最常用的查找和聚合。

初始代码：

var client = new MongoClient("mongodb://localhost");

var database = client.GetDatabase("foo");

var collection = database.GetCollection<BsonDocument>("bar");

插入：

var document = new BsonDocument { { "\_id", 1 }, { "x", 2 } };

await collection.InsertOneAsync(document);

InsertOneAsync返回一个没有值的Task，插入完成后Task要么完成，要么错误状态。

查询：

var filter = new BsonDocument("x", new BsonDocument("$gte", 100));

var documents = await collection.Find(filter).ToListAsync();

如果查询返回很多文档，应该在流中处理他们而不是把他们转成一张大表。

var filter = new BsonDocument("x", new BsonDocument("$gte", 100));

await collection.Find(filter).ForEachAsync(async document =>

{

await ProcessDocumentAsync(document);

});

正如驱动是100%异步的，你的程序也应该100%异步，在上面的例子文档是异步处理的。

1.Find 和FindAsync

IMongoCollection<TModel> collection = // ...

FindOptions<TModel> options = new FindOptions<TModel> { Limit = 1 };

IAsyncCursor<TModel> task = await collection.FindAsync(x => x.Id.Equals(id), options);

List<TModel> list = await task.ToListAsync();

TModel result = list.FirstOrDefault();

return result;

IMongoCollection<TModel> collection = // ...

IFindFluent<TModel, TModel> findFluent = collection.Find(x => x.Id == id);

findFluent = findFluent.Limit(1);

TModel result = await findFluent.FirstOrDefaultAsync();

return result;

Find和FindAsync参数不一样，都允许以异步方式查询，性能相同。只是FindAsync返回的是cursor，这样不会一次加载所有文档，并且提供你接口通过数据库游标去遍历检索文档。当查询结果巨大的时候很有用。

Find提供更简单的参数通过方法ToListAsync，实际也是通过cursor检索文档，然后一次返回所有文档。

如果只检索单个文档，Find似乎效率更高，因为不会用到游标。

如果cursor没找到，先使用

await cursor.MoveNextAsync()；

没找到：cursor.Current==0；

找到：cursor.Current==n；

IFindFluent<JDBCEntity, JDBCEntity> findFluent = collection1.Find(x => x.Name == "exp3");

JDBCEntity entity = await findFluent.FirstOrDefaultAsync();

找到了返回实例，没找到entity==null;

如果要使用 ToListAsync()方法，就不用MoveNext了。找不到会返回0长度的list。

var cursor = await collectionPayload.FindAsync(filter);

// await cursor.MoveNextAsync(); 这句一定要注释掉

List<JDBCEntity> tm=await cursor.ToListAsync();

IFindFluent的用法：

IFindFluent<JDBCEntity, JDBCEntity> findFluent =

collection1.Find(x => x.Id == exp2.Id);

findFluent = findFluent.Limit(1);

JDBCEntity result1 = await findFluent.FirstOrDefaultAsync();

Console.WriteLine(result1.Id);

查询时要过滤相关属性时：

Find

var filter = Builders<JDBCEntity>.Filter.Eq<string>(m => m.Name, "exp2");

var projection = Builders<JDBCEntity>.Projection.Exclude(m => m.Id).Include(m => m.Name);

var doc = collection1.Find(filter).Project(projection).FirstOrDefaultAsync().Result;

foreach (var item in doc)

{

Console.WriteLine(item.Name);

Console.WriteLine(item.Value);

}

或者：

var filter = Builders<JDBCEntity>.Filter.Eq<string>(m => m.Name, "exp2");

var projection = Builders<JDBCEntity>.Projection.Exclude(m => m.Id).Include(m => m.Id);

var doc = collection1.Find(filter).Project(projection).ToListAsync().Result;

foreach (var item in doc)

{

Console.WriteLine(item["\_id"].AsGuid);

}

或者：

var doc = collection1.Find(filter).Project(projection).ForEachAsync((ww) => { Console.WriteLine(ww["\_id"].AsGuid); });

FindAsync

var filter = Builders<JDBCEntity>.Filter.Eq(m => m.Name,"exp2");

var projection = Builders<JDBCEntity>.Projection.Include(m => m.Id);

var options = new FindOptions<JDBCEntity, BsonDocument> { Projection = projection };

var cursor = await collection1.FindAsync(filter, options);

while (await cursor.MoveNextAsync())

{

var batch = cursor.Current;

foreach (BsonDocument b in batch)

{

Console.WriteLine(b["\_id"].AsGuid);

}

}

查询时要指定查询结果数量时：

var builder = Builders<JDBCEntity>.Filter;

var filters = new List<FilterDefinition<JDBCEntity>>();

filters.Add(builder.Eq(m => m.Name, "exp2"));

filters.Add(builder.Eq(m => m.Id, exp2.Id));

IAsyncCursor<JDBCEntity> cursor = collection1.FindAsync(builder.And(filters),

new FindOptions<JDBCEntity>() { BatchSize = 1 }).Result;

List<JDBCEntity> members = new List<JDBCEntity>();

while (cursor.MoveNextAsync().Result)

{

members.AddRange(cursor.Current);

}

ForEachAsync的用法：

var docs = new Dictionary<string, BsonDocument>();

var result = db.GetCollection<BsonDocument>("mycollection").Find(new BsonDocument());

result.ForEachAsync((bsonDoc) =>

{

string name = bsonDoc.GetValue("name").AsString;

if (!docs.ContainsKey(name))

{

docs[name] = bsonDoc;

}

});

var result2 = collection1.Find(new BsonDocument());

await result2.ForEachAsync((bsonDoc) =>

{

Console.WriteLine(bsonDoc.Name);

});

文档查询Slice

var filter = new BsonDocument();

var sot = Builders<JDBCEntity>.Sort.Descending("Num");

var options = new FindOptions<JDBCEntity, BsonDocument> { Sort=sot ,Skip=0,Limit=1,};

var cursor = await collection1.FindAsync(filter,options);

UpdataManyAsync

update时，如果用PUSH，它是每次把变量当做一个整体push进去，比如Dimensions是一个List<long>={1,2,4},使用push（tmp）后，变成{1,2,4，{15,18,16}}，push（12），变成{1,2,4，{15,18,16}，12}

如果用pusheach方法，{1,2,4，15,18,16}，则是按序push。

var filter = Builders<JDBCEntity>.Filter.Eq("Name", "exp2");

var up = Builders<JDBCEntity>.Update.Set ("Phone","1355");

var result= await collection1.UpdateManyAsync(filter, up);

var tmp=new List<long>{15,18,16};

// var up = Builders<JDBCEntity>.Update.Push("Dimensions",16);

// var up = Builders<JDBCEntity>.Update.Push("Dimensions",tmp);

var up = Builders<JDBCEntity>.Update.PushEach("Dimensions", tmp);

如果是用AddToSet方法，效果是一样的，但是会检查集合中，是否有元素相等，如果相等则不做任何操作，如果有个别不相等，则会插入末尾 。Push方法不会检查，会直接插入数据。

// var up = Builders<JDBCEntity>.Update.AddToSet("Dimensions", tmp);

// var up = Builders<JDBCEntity>.Update.AddToSetEach("Dimensions", tmp);

不做其他操作，只查询时，cursor这里会报错，不能创建抽象类的实例。

var client = new MongoClient();

var database = client.GetDatabase("JDBC");

var collection1 = database.GetCollection<JDBCEntity>("Experiment");

var filter = Builders<JDBCEntity>.Filter.Eq("Name", "exp2");

using (var cursor = await collection1.FindAsync(filter))

{

await cursor.MoveNextAsync();

var res = cursor.Current.FirstOrDefault();

}

需要在查询前插入一句：

BsonClassMap.LookupClassMap(typeof(Experiment));

var client = new MongoClient();

var database = client.GetDatabase("JDBC");

var collection1 = database.GetCollection<JDBCEntity>("Experiment");

var filter = Builders<JDBCEntity>.Filter.Eq("Name", "exp2");

using (var cursor = await collection1.FindAsync(filter))

{

await cursor.MoveNextAsync();

var res = cursor.Current.FirstOrDefault();

}

**查询实例，修改后再插入，不行。需要使用ReplaceOneAsync**

var filter = Builders<JDBCEntity>.Filter.Eq("Name", "exp2");

BsonClassMap.LookupClassMap(typeof(Experiment));

using (var cursor = await collection1.FindAsync(filter))

{

await cursor.MoveNextAsync();

var res = cursor.Current.FirstOrDefault();

res.Name = "exp10";

await collection1.ReplaceOneAsync(filter,res);

}

MongoDB单文件大小16MB，大约到16.5M会报错。

大约是1450000个Double数

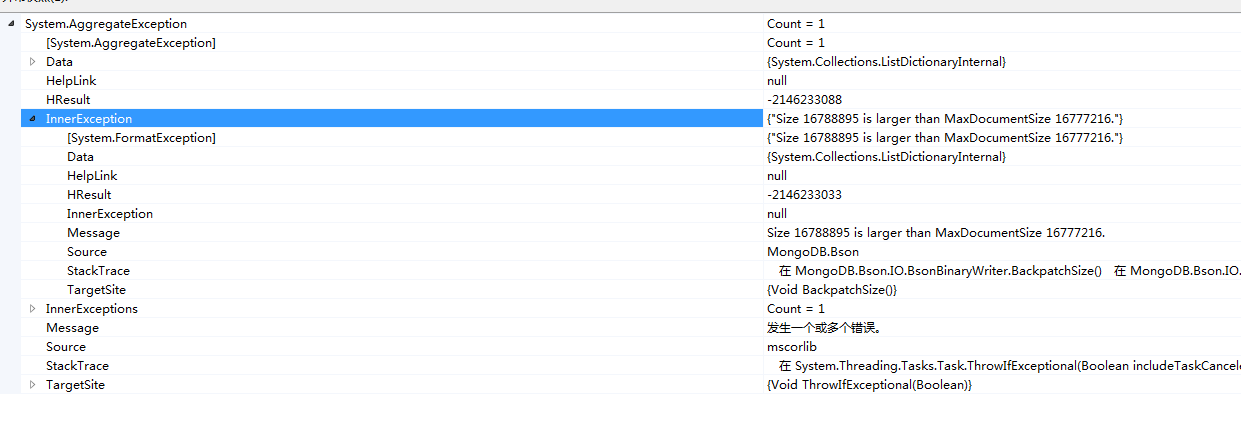
插入一个long，11个字符，插入一个double，12个字符，当

List<double>.Count()>1000000 时，再插入大量double就容易报错，尽管实际大小并没有达到16MB。

差不多1050000时，可以插入单个数据，插入数组时就报错。

这时文档大小差不多12015635B（11.5M）

{"A write operation resulted in an error.\r\n BSONObj size: 16793621 (0x1004015) is invalid. Size must be between 0 and 16793600(16MB) First element: Samples.130003: 0.06810414142352722"}



Payload的查询方法

var start = new List<long> { 2, 1, 3, 4 };

var filter = Builders<JDBCEntity>.Filter.Eq("Dimensions", start);

BsonClassMap.LookupClassMap(typeof(Payload<long>));

using (var cursor = await collectionPayload.FindAsync(filter))

{

await cursor.MoveNextAsync();

var batch = cursor.Current;

foreach (Payload<long> item in batch)

{

Console.WriteLine(item.Name);

}

查找Payload，并对End进行排序，找出最大的End

var filter = new BsonDocument();

var sot = Builders<JDBCEntity>.Sort.Descending("End");

var options = new FindOptions<JDBCEntity, Payload<long>> { Sort = sot };

BsonClassMap.LookupClassMap(typeof(Payload<long>));

using (var cursor = await collectionPayload.FindAsync(filter, options))

{

await cursor.MoveNextAsync();

var batch = cursor.Current;

//foreach (var item in batch)

//{

// Console.WriteLine(item["Name"].AsString+" "+item["End"].AsInt64);

//}

foreach (Payload<long> item in batch)

{

Console.WriteLine(item.Name + " " + item.End);

}

}