MongoDB 的使用场景及主要优势

1.Web 应用程序。 文档能表示丰富的数据结构， 建模相同数据库所需的集合数量通常会比使用完全正规化关系型数据库的数据表数量要少。 动态查询和二级索引能让你轻松的实现 SQL 开发者所熟悉的大多数查询。作为一个成长中的 Web 应用程序， MongoDB 提供了清晰的扩展路线。

2.敏捷开发。 MongoDB 没有固定的 Schema，所有花在提交、沟通和实施 Schema变更的时间都生下来了。

3.分析和日志。 MongoDB 的目标原子更新和固定集合。

4.缓存

5.可变 Schema

MongoDB 提示与局限：大多数是由于 MongoDB 使用内存映射文件导致的。

1.MongoDB 应该运行于 64 位机器上。 32 为系统只能对 4GB 内存做寻址。

2.数据占用的内存会自动按需分配。 这样一来在共享环境中运行数据库会变得更加麻烦。所以最好能让 MongoDB 运行在一台专门的服务器上。

3.运行带复制功能的 MongoDB 是十分重要的，尤其是没有开启 Journaling 日志的时候。

1.mongodb介绍

MongoDB (名称来自 "humongous") 是一个可扩展的高性能，开源，模式自由，面向文档的数据库。它使用 C++编写。 MongoDB 特点:

a.面向集合的存储：适合存储对象及 JSON 形式的数据。

b.动态查询： mongo 支持丰富的查询表达方式，查询指令使用 JSON 形式的标记，可轻易查询文档中的内嵌的对象及数组。

c.完整的索引支持 :包括文档内嵌对象及数组。 mongo 的查询优化器会分析查询表达式，并生成一个高效的查询计划。

d.查询监视： mongo 包含一个监视工具用于分析数据库操作性能。

e.复制及自动故障转移： mongo 数据库支持服务器之间的数据复制，支持主-从模式及服务器之间的相互复制。复制的主要目的是提供冗余及自动故障转移。

f.高效的传统存储方式：支持二进制数据及大型对象（如照片或图片）。

g.自动分片以支持云级别的伸缩性： 自动分片功能支持水平的数据库集群，可动态添加额外的机器。

2.mongo 使用场合

mongodb 的主要目标是在键 /值存储方式（提供了高性能和高度伸缩性）以及传统的 RDBMS 系统（丰富的功能）架起一座桥梁，集两者的优势于一身。mongo 适用于以下场景：

a.网站数据： mongo 非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。

b.缓存：由于性能很高， mongo 也适合作为信息基础设施的缓存层。在系统重启之后，由 mongo 搭建的持久化缓存可以避免下层的数据源过载。

c.大尺寸、低价值的数据：使用传统的关系数据库存储一些数据时可能会比较贵，在此之前，很多程序员往往会选择传统的文件进行存储。

d.高伸缩性的场景： mongo 非常适合由数十或者数百台服务器组成的数据库。

e.用于对象及 JSON 数据的存储： mongo 的 BSON 数据格式非常适合文档格式化的存储及查询。

不适合的场景：

a.高度事物性的系统：例如银行或会计系统。传统的关系型数据库目前还是更适用于需要大量原子性复杂事务的应用程序。

b.传统的商业智能应用： 针对特定问题的 BI 数据库会对产生高度优化的查询方式。对于此类应用，数据仓库可能是更合适的选择。

c.需要 SQL 的问题。

开发中用 MongoDB 的爽快之处 MongoDB 是一个非常适合 PHP 开发的 #NoSQL# 数据库，尤其支持高并发和 schema-free（自由结构）特性，使得 PHP 开发变得更灵活，更高效。大家都来试试吧！或许很多时候，记录在 mysql 里面的自由字段（如序列化后的数组），都很难去回忆或者找到在什么地方。而 mongodb 却可以让你轻松找到需要的数组，而且很有规律（条件查询）。这种轻松，对比起来就知道了。 mysql 在大数据量情况下， orderby 和 groupby 的效率会非常的低。从目前的实践来看， mongodb 是很有希望和很有必要替代 orderby 和 groupby，因其有灵活的数据结构模型，可作为一个可查找和排序、归类的数据中间缓存层，而且效率会比通常的 mysql 查询高很多。

思考 MongoDB 的使用场景

重要数据： mysql，一般数据： mongodb，临时数据： memcache对于关系数据表而言， mongodb 是提供了一个更快速的视图 view ；而对于 PHP 程序而言， mongodb 可以作为一个持久化的数组来使用， 并且这个持久化的数组还可以支持排序、条件、限制等功能。

将 mongodb代替 mysql 的部分功能，主要一个思考点就是： 把 mongodb 当作 mysql 的一个 view（视图）， view 是将表数据整合成业务数据的关键。比如说对原始数据进行报表，那么就要先把原始数据统计后生成 view，在对 view 进行查询和报表。从这个意义上， mongodb 提供了一个更快速，更使用的 view

Mongodb 等 noSQL 技术，作为 memcache和 mysql 的一个中间 view 是比较好的

选择。而 noSQL 进行持久化是非常不好的事情，除非有 SSD 硬盘，否则普通硬盘条件下，持久化会令系统负担极大，并且效率甚至有可能比 mysql 还要低。MongoDB 遇到“ Caught exception: not utf8 string: ”的异常，这是插入非 UTF8 编码文字的时候就会抛出

2 MongoDB 主要优势

2.1Schema Free MongoDB 是一个面向文档的数据库内部数据存储格式为 BSON(可以认为是二进制的 JSON)。MongoDB 中，一个数据库可以有多个 Collection，每个 Collection 是 Document 的集合。Collection 和 Document 和传统数据库的 Table 和 Row 并不对等。数据库和 Collection 都无需预先定义，随时可以创建。Collection 中可以包含具有不同 schema的文档记录，即 Schema Free。简单来说，上一条记录中的文档有 3 个属性，而下一条记录的文档可以有 10 个属性，属性的类型既可以是基本的数据类型（如数字、字符串、日期等），也可以是数组或者散列，甚至还可以是一个嵌入式文档（ embed document ）。Schema Free 意味着应用数据改动的时候不需要对 MongoDB 进行修改，换而言之， mongoDB 是启动即可用的，可以方便的处理需求变化剧烈的场景。这省去了传统关系数据库基于表结构变化的繁琐 DDL 操作（比如 DBA 定期备份脚本，对正式环境数据库需要不断部署新的 sql 脚本等等）。

2.2

丰富查询功能 MongoDB 的主要目标是在键 /值存储方式（提供了高性能和高度伸缩性）以及传统的 RDBMS 系统（丰富的功能）架起一座桥梁，集两者的优势于一身。 MongoDB 中的查询基于 JSON 的语法，没有 SQL 难记的语法，相当的直观。对不同的开发语言， 你可以使用它最基本的数组或散列格式进行查询。配合附加的 operator， MongoDB 支持范围查询，正则表达式查询，对子文档内属性的查询，可以取代原来大多数任务的 SQL 查询。

下面详细介绍一下 MongoDB 的查询语法：

1.子集查询（ Projection）

```
// select z from things where x="john" db.things.find( { x : "john" }, { z : 1 } );
//查询没有 comments 字段的 posts 记录 db.posts.find( { tags : 'tennis' }, { comments : 0 } );
```

2.Slice 操作

```
//查询前 5 个 comments db.posts.find({}, {comments:{$slice: 5}})
//查询倒数第 20 个到倒数第 30 个 comments db.posts.find({}, {comments:{$slice: [-20, 10]}})
```

3.条件查询

```
// 算术关系（ <, <=, >, >=,!=）
```

```
查询 field > value 的记录
db.collection.find({ "field" : { $gt: value } } );
// all 查询
a 数组包含 2 和 3 的记录 db.things.find( { a: { $all: [ 2, 3 ] } } );
// in select * from things where j in (2,4,6) db.things.find({j:{$in: [2,4,6]}});
// exists 查询有
a 属性的记录 db.things.find( { a : { $exists : true } } );
// 取模
查询 a%10==1 的记录 db.things.find( { a : { $mod : [ 10 , 1 ] } } )
// 逻辑关系 or 操作
select * from things where a=1 or b=2
db.foo.find( { $or : [ { a : 1 } , { b : 2 } ] } )
// 逻辑关系 not 操作
db.things.find( { a : { $not : { $mod : [ 10 , 1 ] } } } );
4.正则查询
//使用 PCRE 表示
db.customers.find( { name : /acme.*corp/i } );
5.嵌套文档查询
//使用 dot 运算表示嵌套文档，其中 name:joe 是 author 属性的嵌套文档
db.postings.find( { "author.name" : "joe" } );
6.Cursor 操作
//limit
db.students.find().limit(10).forEach( function(student) { print(student.name + "<
p>"); } );
//skip,
指定开始位置，用于分页

db.students.find().skip((pageNumber-1)*nPerPage).limit(nPerPage)
 // sort ,
即 order by, 根据 ts 字段降序
db.myCollection.find().sort( { ts : -1 } );
// count
db.students.find({'address.state' : 'CA'}).count();
// group,非 shard 环境下，用 group 函数可以直接完成 group 操作
// shard
```

环境下需要使用 MapReduce任务完成具体的 group 操作

// 统计 2009 年 9 月的 http_action 的次数，总时间，平均时间

db.test.group({ cond: {"invoked_at.d": {$gte: "2009-11", $lt: "2009-12"}} , key : {http_action: true} , initial: {count: 0, total_time:0} , reduce: function(doc, out){ out.count++; out.total_time+=doc.response_time } , finalize: function(out){ out.avg_time = out.total_time / out.count } });

//map reduce的 group count

例子如下

> db.things.insert({ _id : 1, tags : ['dog', 'cat'] });

> db.things.insert({ _id : 2, tags : ['cat'] });

> db.things.insert({ _id : 3, tags : ['mouse', 'cat', 'dog'] });

> db.things.insert({ _id : 4, tags : []

});

> // map function

> m = function(){ this.tags.forEach(function(z){ ... emit(z , { count : 1 }); ... }

...); ...};

> // reduce function

> r = function(key , values){

...

var total = 0;

...

for (var i=0; i<values.length; i++)

...

total += values[i].count;

...

return { count : total };

...};

> res = db.things.mapReduce(m,r);

> res

{"timeMillis.emit" : 9 , "result" : "mr.things.1254430454.3" , "numObjects" : 4 , "timeMillis" : 9 , "errmsg" : "" , "ok" : 0}

> db[res.result].find()

{"_id" : "cat" , "value" : {"count" : 3}}

{"_id" : "dog" , "value" : {"count" : 2}}

```
{"_id" : "mouse" , "value" : {"count" : 1}}
```

2.3 数据可扩展性

MongoDB 提供基于 Range的 Auto Sharding 机制：一个 collection 可按照记录的范围，分成若干个段，切分到不同的 Shard上。 MongoDB 会根据各个 Shards 的网络延迟和系统负载情况调整数据的分布，实现各个 Shard的负载均衡，同时 MongoDB 支持动态添加 Shard，并重新调整数据分布，使新 Shard充满数据。 MongoDB 的 Shard机制实现了数据规模的可扩展性，随着数据规模增加，可以通过增加 Shard来满 足 需 求。同 时根据测试表明，随着 Shard增加，性能拐点的出现也会延长到更大的数据规模， 也就是说 Shard机制还实现了性能的可扩展性。 尽管 MongoDB 集群中数据会分布到不同的 Shard中，但查询是对客户端却是高度透明的。客户端执行查询， MapReduce等操作，这些会被 MongoDB 自动路由到后端的数据节点。在应用程序看来， MongoDB 集群整体就是一个数据库。 MongoDB 的 Sharding设计能力最大可支持大约 20 PB，足以支撑一般应用。总的来说， MongoDB 可以让我们关注于自己的业务，适当的时候可以无痛的升级。

2.4 容错机制

MongoDB 在 1.6 版本后实现了新的复制方式 Replica Set。通过 Relpica Set， MongoDB 实现了故障自动切换和故障自动恢复。 该模式类似分布式系统的结构，在管理节点（ 1 个或者 3 个）的管 理 下，数 据 节点的关系都是均等的，同过投票选出主节点，只有主节点可以读写，其余节点只读。当 Relpica Set中某一节点甚至主节点故障时，可以由管理节点控制选出替换节点，该过程是自动进行的，无须人工干预，即故障自动切换。当故障节点修复后，重新加入 Relpica Set， 该节点会自动从其他正常节点同步数据，直到完全一致，即故障自动恢复。通过 Relpica Set技术, MongoDB 具备了生产环境下的容错能力，能最大限度的保证数据完整性，并降低运维的难度。