

iOS 雲端服務開發

第三週

課程大綱

- MongoDB
- RESTful API
- Create a RESTful Service with MongoDB

MongoDB 特點

- Document oriented
- JSON-like 儲存方式 (BSON)
- javascript-like 操作方式
- High Availability / Scalability
- Map / Reduce

為何使用 MongoDB

- 操作一致性 (javascript-like)
- Big Data
- 彈性的資料儲存
- 支援地理位置儲存
- 擴充性 / 穩定性

MongoDB 的取捨

- atomic transaction只有在document層級
- 沒有join可使用，資料表的規劃方式與傳統RMDBS思維不同
- 資料佔用空間較RMDBS大，RAM的需求也較高

MongoDB 安裝

• Homebrew

```
#先更新系統  
brew update
```

```
#安裝  
brew install mongodb
```

```
#更新版本  
brew update  
brew upgrade mongodb
```

• MacPorts

```
port install mongodb
```

執行 MongoDB

```
#執行mongo  
~$ mongod
```

```
#進入mongo指令列  
~$ mongo
```

```
> db.test.save( { a: 1 } )  
> db.test.find()
```

```
離開  
> exit
```

資料庫架構

- db
- collection
- document
- field
- index
- cursor

CRUD

- Create: insert, update, save
- Read: find, findOne
- Update: update, save
- Delete: remove

連結資料庫

顯示目前資料庫

> db

顯示所有資料庫

> show dbs

切換資料庫

> use \$db

幫助

> help

新增資料

變數設定

```
> j = { name : "mongo" }
> k = { x : 3 }
```

插入

```
> db.things.insert( j )
> db.things.insert( k )
```

```
> db.things.find()
{ "_id" : ObjectId("51bb168a65132ef18fb767f6"), "name" : "mongo" }
{ "_id" : ObjectId("51bb184865132ef18fb767f7"), "x" : 3 }
```

迴卷

```
> for (var i = 1; i <= 40; i++) db.things.insert( { x : 4 , j : i } )
> db.things.find()
{ "_id" : ObjectId("51bb168a65132ef18fb767f6") , "name" : "mongo" }
{ "_id" : ObjectId("51bb184865132ef18fb767f7") , "x" : 3 }
{ "_id" : ObjectId("51bb1d7565132ef18fb767f8") , "x" : 4 , "j" : 1 }
{ "_id" : ObjectId("51bb1d7565132ef18fb767f9") , "x" : 4 , "j" : 2 }
{ "_id" : ObjectId("51bb1d7565132ef18fb767fa") , "x" : 4 , "j" : 3 }
{ "_id" : ObjectId("51bb1d7565132ef18fb767fb") , "x" : 4 , "j" : 4 }
{ "_id" : ObjectId("51bb1d7565132ef18fb767fc") , "x" : 4 , "j" : 5 }
{ "_id" : ObjectId("51bb1d7565132ef18fb767fd") , "x" : 4 , "j" : 6 }
{ "_id" : ObjectId("51bb1d7565132ef18fb767fe") , "x" : 4 , "j" : 7 }
{ "_id" : ObjectId("51bb1d7565132ef18fb767ff") , "x" : 4 , "j" : 8 }
{ "_id" : ObjectId("51bb1d7565132ef18fb76800") , "x" : 4 , "j" : 9 }
{ "_id" : ObjectId("51bb1d7565132ef18fb76801") , "x" : 4 , "j" : 10 }
{ "_id" : ObjectId("51bb1d7565132ef18fb76802") , "x" : 4 , "j" : 11 }
{ "_id" : ObjectId("51bb1d7565132ef18fb76803") , "x" : 4 , "j" : 12 }
{ "_id" : ObjectId("51bb1d7565132ef18fb76804") , "x" : 4 , "j" : 13 }
{ "_id" : ObjectId("51bb1d7565132ef18fb76805") , "x" : 4 , "j" : 14 }
{ "_id" : ObjectId("51bb1d7565132ef18fb76806") , "x" : 4 , "j" : 15 }
{ "_id" : ObjectId("51bb1d7565132ef18fb76807") , "x" : 4 , "j" : 16 }
{ "_id" : ObjectId("51bb1d7565132ef18fb76808") , "x" : 4 , "j" : 17 }
{ "_id" : ObjectId("51bb1d7565132ef18fb76809") , "x" : 4 , "j" : 18 }
Type "it" for more
```

查詢／刪除

```
> db.things.find( { name : "mongo" } )  
{ "_id" : ObjectId("51bb168a65132ef18fb767f6"), "name" : "mongo" }
```

移除所有資料

```
> db.things.remove()
```

移除collection

```
> db.things.drop()  
true
```

移除資料庫

```
> db.dropDatabase()
```

CREATE

- `insert`: 新增
- `save`: 找不到`_id`就新增
- `upsert`: 搜尋不到就新增

insert

- db.collection.insert(\$document)
- _id: 唯一，可自定或系統自動插入ObjectId，不可更動
- 欄位不可以以\$開頭，或含有.
- 一個document不可以超過16MB

insert

```
db.bios.insert({  
    _id: 1,  
    name: { first: 'John', last: 'Backus' },  
    birth: new Date('Dec 03, 1924'),  
    death: new Date('Mar 17, 2007'),  
    contribs: [ 'Fortran', 'ALGOL', 'Backus-Naur Form', 'FP' ],  
})  
  
db.bios.insert({  
    name: { first: 'John', last: 'McCarthy' },  
    birth: new Date('Sep 04, 1927'),  
    death: new Date('Dec 24, 2011'),  
    contribs: [ 'Lisp', 'Artificial Intelligence', 'ALGOL' ],  
})  
  
db.bios.insert(  
[  
    { _id:3, name: { first: 'Steve', last: 'Jobs' }},  
    { _id:4, name: { first: 'Bill', last: 'Gates' }}  
]  
)
```

save

- db.collection.save(\$document)
- 如果沒有_id欄位或是資料表裡沒有此筆資料則新增，否則執行update
- 注意：update操作是覆蓋原資料

```
db.bios.save({  
  _id: 1,  
  name: { first: 'Brendan', last: 'Eich' },  
  contribs: [ 'Javascript' ],  
})  
  
db.bios.save({  
  name: { first: 'Marc', last: 'Andreessen' },  
  birth: new Date('July 9, 1971'),  
  contribs: [ 'Mosaic', 'Netscape' ],  
})
```

upsert

- update選項找到資料時，執行update，若沒找到資料則執行insert

```
db.bios.update(  
  { name: { first: 'Dennis', last: 'Ritchie' } },  
  {  
    name: { first: 'Dennis', last: 'Ritchie' },  
    birth: new Date('Sep 09, 1941'),  
    death: new Date('Oct 12, 2011'),  
    contribs: [ 'UNIX', 'C' ],  
  },  
  { upsert: true }  
)
```

upsert

- 如果upsert的資料包含了update operators，則會把搜尋的部分也加入（找不到資料的情況下）

```
db.bios.update(  
  {  
    _id: 7,  
    name: { first: 'Ken', last: 'Thompson' }  
  },  
  {  
    $set: {  
      birth: new Date('Feb 04, 1943'),  
      contribs: [ 'UNIX', 'C', 'B', 'UTF-8' ]  
    }  
  },  
  { upsert: true }  
)
```

READ

- 📌 find: 搜尋
- 📌 findOne: 搜尋，但只回傳第一筆文件

find

- `db.collection.find($query, $projection)`
 - `query`: 搜尋條件
 - `projection`: 結果欄位

列出所有資料

> `db.things.find()`

列出所有資料，並作格式化

> `db.things.find().pretty()`

find 進階搜尋

```
> db.bios.find({  
    _id: 5  
})
```

搜尋多個值

```
db.bios.find({  
    _id: { $in: [ 5, ObjectId("507c35dd8fada716c89d0013") ] }  
})
```

find 進階搜尋

```
> db.students.find()
{ "_id" : 1, "score" : [ -1, 3 ] }
{ "_id" : 2, "score" : [ 1, 5 ] }
{ "_id" : 3, "score" : [ 5, 5 ] }

> db.students.find( { _id: { $gt: 1 } } )
{ "_id" : 2, "score" : [ 1, 5 ] }
{ "_id" : 3, "score" : [ 5, 5 ] }

> db.students.find( { score: { $gt: 0, $lt: 2 } } )
{ "_id" : 1, "score" : [ -1, 3 ] }
{ "_id" : 2, "score" : [ 1, 5 ] }

> db.students.find( { score: 5 } )
{ "_id" : 2, "score" : [ 1, 5 ] }
{ "_id" : 3, "score" : [ 5, 5 ] }
```

find 進階搜尋

• subdocuments

絕對相等（欄位順序需要相等）

```
db.bios.find(  
  {  
    name: {  
      first: 'Yukihiro',  
      last: 'Matsumoto'  
    }  
  }  
)
```

```
db.bios.find(  
  {  
    'name.first': 'Yukihiro',  
    'name.last': 'Matsumoto'  
  }  
)
```

find 進階搜尋

• 運用邏輯運算子

```
db.bios.find(  
  { $or: [  
    { 'name.first' : /^G/ },  
    { birth: { $lt: new Date('01/01/1945') } }  
  ]  
}
```

預設為AND

```
db.bios.find(  
  {  
    'name.first': /^K/,  
    contribs: 'UNIX'  
  }  
)
```

find 結果欄位

僅顯示name, contribs 欄位

```
db.bios.find(  
  { },  
  { name: 1, contribs: 1 }  
)
```

顯示name, contribs 不顯示_id

```
db.bios.find(  
  { },  
  { name: 1, contribs: 1, _id: 0 }  
)
```

find 結果欄位

顯示所有其他欄位

```
db.bios.find(  
  { contribs: 'OOP' },  
  { _id: 0, 'name.first': 0, birth: 0 }  
)
```

顯示name.last 以及contribs的前兩筆

```
db.bios.find(  
  { },  
  {  
    _id: 0,  
    'name.last': 1,  
    contribs: { $slice: 2 }  
  }  
)
```

find 結果操作

依照name升冪排序

```
db.bios.find().sort( { name: 1 } )
```

僅顯示前五筆

```
db.bios.find().limit( 5 )
```

從第六筆開始顯示

```
db.bios.find().skip( 5 )
```

從第六筆開始顯示五筆

```
db.bios.find().skip( 5 ).limit( 5 )
```

- 順序不影響結果

findOne

- 基本上是`find().limit(1)`
- 操作方式與`find`一樣
- 但回傳資料為`document`，故不能用`sort`, `limit`等

Update

- update
- save

update

- db.collection.update(\$query, \$update, \$options)
- query為搜尋條件，參照find
- update為更新資料，可全部替換或是部分替換
- 預設僅更新第一筆符合的文件
 - 透過\$options.multi來更改行為

update

全部替換

```
db.bios.update(  
  { '_id': 1 },  
  { 'name.middle': 'Warner' }  
)
```

部分更新

```
db.bios.update(  
  { '_id': 1 },  
  {  
    $set: { 'name.middle': 'Warner' }  
  }  
)
```



部分更新也可新增欄位

update

移除欄位

```
db.bios.update(  
  { _id: 3 },  
  { $unset: { birth: 1 } }  
)
```

更新多筆資料

```
db.bios.update(  
  { 'awards.award': 'Turing' },  
  { $set: { turing: true } },  
  { multi: true }  
)
```

update 陣列操作

針對陣列index:1元素更新

```
db.bios.update(  
  { _id: 1 },  
  { $set: { 'contribs.1': 'ALGOL 58' } }  
)
```

更新符合的元素

```
db.bios.update(  
  { _id: 3, 'contribs': 'compiler' },  
  { $set: { 'contribs.$': 'A compiler' } }  
)
```

- 如果陣列中有多個元素需要更新，只有第一個元素會更新

update upsert

```
db.bios.update(  
  { _id:11, name: { first: 'James', last: 'Gosling' } },  
  {  
    $set: {  
      born: new Date('May 19, 1955'),  
      contribs: [ 'Java' ],  
      awards: [  
        {  
          award: 'The Economist Innovation Award',  
          year: 2002,  
          by: 'The Economist'  
        },  
        {  
          award: 'Officer of the Order of Canada',  
          year: 2007,  
          by: 'Canada'  
        }  
      ]  
    },  
    { upsert: true }  
  }
```

新文件也會包含此欄位

update 常用功能

• \$inc

• db.collection.update(\$query,
{ \$inc: { field1: amount } });

```
db.collection.update( { age: 20 }, { $inc: { age: 1 } }, { multi:  
true } );  
db.collection.update( { name: "John" }, { $inc: { age: 2 } }, { multi:  
true } );
```

update 常用功能

- \$rename
 - db.collection.update(<query>, { \$rename: { <oldname>:<newname> } });

```
db.students.update( { _id: 1 }, { $rename: { 'nickname': 'alias', 'cell': 'mobile' } } )
```

DELETE

- remove
 - db.collection.remove(<query>, <justOne>)

```
db.bios.remove( { turing: true }, 1 )
```

map-reduce

用來做資料統計

```
var map = function() {
    var key = {};
    key.bigX = this.x > 10;
    key.bigJ = this.j > 10;
    emit(key, {count:1});
};
```

```
var reduce = function(key, values) {
    var sum = 0;
    values.forEach(function(value) {
        sum += value['count'];
    });
    return {count: sum};
};
```

執行

```
> db.things.mapReduce(map, reduce, {out: {inline:1}})
> db.things.mapReduce(map, reduce, {out: {'things_stats'}})
> db.things_stats.find()
```

index & 效能分析

```
> db.things.ensureIndex({x:1});
```

```
> db.things.find({x:{$gt:10}}).explain()
```

```
> db.things.dropIndex({x:1});
```

備份 & 回復

```
~ $ mongodump --collections #{coll} --db #{db} --out #{out_path}
```

```
~ $ mongorestore #{path_to_dump}
```

- 練習1： 將week3/dump資料夾下的資料還原回資料庫
- 練習2： 找出2000年後有得獎的人名
- 練習3： 針對2000年後有得獎的人，新增一個欄位millenium:1
- 練習4： 移除所有bios資料，刪除所有bios資料

mongoose

- mongo資料庫物件操作 for node.js
- 定義schema
- middleware
- 操作model
- validation
- 類join: population
- 文件操作

mongoose

- mongo資料庫物件操作 for node.js

安裝

```
$ npm install mongoose
```

- connect(uri, option, callback)

```
var mongoose = require('mongoose');

var cb = function(err) {
  if (err) {
    return console.log("Error connecting to: " + uristring + ".");
  } else {
    return console.log("Successfully connected to: " + uristring);
  }
});

mongoose.connect('mongodb://localhost/test', {db: {safe:true}}, cb);
```

Schemas

```
var blogSchema = new Schema({
  title: { type: String, index: true },
  author: String,
  body: String,
  comments: [{ body: String, date: Date }],
  date: { type: Date, default: Date.now },
  hidden: Boolean,
  meta: {
    votes: Number,
    favs: Number
  }
});
```

Schemas

- Types

- String
- Number
- Date
- Buffer
- Boolean
- Mixed
- Objectid
- Array

Schemas

```
var schema = new Schema({
  name:      String,
  binary:    Buffer,
  living:    Boolean,
  updated:   { type: Date, default: Date.now }
  age:       { type: Number, min: 18, max: 65 }
  mixed:    Schema.Types.Mixed,
  _someId:  Schema.Types.ObjectId,
  array:     [],
  ofString:  [String],
  ofNumber:  [Number],
  ofDates:   [Date],
  ofBuffer:  [Buffer],
  ofBoolean: [Boolean],
  ofMixed:   [Schema.Types.Mixed],
  ofObjectId: [Schema.Types.ObjectId],
  nested: {
    stuff: { type: String, lowercase: true, trim: true }
  }
})
```

Schemas

```
// example use

var Thing = mongoose.model('Thing', schema);

var m = new Thing();
m.name = 'Statue of Liberty'
m.age = 125;
m.updated = new Date();
m.binary = new Buffer(0);
m.living = false;
m.mixed = {[ any: { thing: 'i want' } ]};
m.markModified('mixed');
m._someId = new mongoose.Types.ObjectId();
m.array.push(1);
m.ofString.push("strings!");
m.ofNumber.unshift(1,2,3,4);
m.ofDate.addToSet(new Date());
m.ofBuffer.pop();
m.ofMixed = [1, [], 'three', { four: 5 }];
m.nested.stuff = 'good';
m.save(callback);
```

Schemas

```
userSchema.methods.comparePassword = function(candidatePassword, cb) {  
  if ( candidatePassword == this.password ) {  
    return cb(null, true);  
  } else {  
    return cb( new Error("Password does not match") );  
  }  
};
```

```
userSchema.statics.findByName = function(name, cb) {  
  this.find({ name: name }, cb);  
};
```

```
personSchema.virtual('name.full').get(function () {  
  return this.name.first + ' ' + this.name.last;  
});
```

middleware

- init, validate, save, remove 前後執行

```
userSchema.pre('save', function(next) {  
    //encrypt password and save to password field  
});
```

```
userSchema.post('save', function (user) {  
    console.log('%s has been saved', user._id);  
})
```

population

- 類似join的功能

```
var mongoose = require('mongoose')
, Schema = mongoose.Schema

var personSchema = Schema({
  _id      : Number,
  name    : String,
  age     : Number,
  stories : [{ type: Schema.Types.ObjectId, ref: 'Story' }]
});

var storySchema = Schema({
  _creator : { type: Number, ref: 'Person' },
  title    : String,
  fans     : [{ type: Number, ref: 'Person' }]
});

var Story  = mongoose.model('Story', storySchema);
var Person = mongoose.model('Person', personSchema);
```

population

```
Story
.findOne({ title: /timex/i })
.populate('_creator', 'name') // only return the Persons name
.exec(function (err, story) {
  if (err) return handleError(err);

  console.log('The creator is %s', story._creator.name);
  // prints "The creator is Aaron"

  console.log('The creators age is %s', story._creator.age);
  // prints "The creators age is null"
})
```

Models 新增

```
var schema = new mongoose.Schema({ name: 'string', size: 'string' })
var Tank = mongoose.model('Tank', schema);
```

```
var Tank = mongoose.model('Tank');

var small = new Tank({ name: 'T92', type: 'light' });
small.save(function (err) {
  if (err) return handleError(err);
  // saved!
})

// or

Tank.create({ name: 'M1A1', type: 'MBT' }, function (err, small) {
  if (err) return handleError(err);
  // saved!
})
```

Models 操作

```
var schema = new mongoose.Schema({ name: 'string', size: 'string' })
var Tank = mongoose.model('Tank', schema);
```

```
User.find({age: {$gte: 21, $lte: 65}}, callback);
User.where('age').gte(21).lte(65).exec(callback);
```

```
Tank.remove({ type: 'light' }, function (err) {
  if (err) return handleError(err);
  // removed!
});
```

Models 操作

```
var query = { name: 'borne' };
Model.update(query, { name: 'jason borne' }, options, callback)

// is sent as
Model.update(query, { $set: { name: 'jason borne' }}, options,
callback)
```

```
Tank.findByIdAndUpdate(id, { $set: { size: 'MBT' }}, function (err,
tank) {
  if (err) return handleError(err);
  res.send(tank); //return updated document
});
```

Models 操作

```
var query = { name: 'borne' };
Model.update(query, { name: 'jason borne' }, options, callback)

// is sent as
Model.update(query, { $set: { name: 'jason borne' }}, options,
callback)
```

```
Tank.findByIdAndUpdate(id, { $set: { size: 'MBT' }}, function (err,
tank) {
  if (err) return handleError(err);
  res.send(tank); //return updated document
});
```

Validation

- 內建validator
 - ALL: required
 - Numbers: min, max
 - Strings: enum

```
courseSchema = new Schema({  
    title: {  
        type: String,  
        required: true  
    },  
    description: String,  
    status: {  
        type: String,  
        "enum": ['on', 'off']  
    }  
});
```

自定 Validator

```
var toySchema = new Schema({
  color: String,
  name: String
});

var Toy = mongoose.model('Toy', toySchema);

Toy.schema.path('color').validate(function (value) {
  return /blue|green|white|red|orange|periwinkel/i.test(value);
}, 'Invalid color');

var toy = new Toy({ color: 'grease'});

toy.save(function (err) {
  // err.errors.color is a ValidationError object

  console.log(err.errors.color.message) // prints 'Validator "Invalid color"
  failed for path color with value `grease``
  console.log(String(err.errors.color)) // prints 'Validator "Invalid color"
  failed for path color with value `grease``
  console.log(err.errors.color.type) // prints "Invalid color"
  console.log(err.errors.color.path) // prints "color"
  console.log(err.errors.color.value) // prints "grease"
  console.log(err.name) // prints "ValidationError"
  console.log(err.message) // prints "Validation failed"
});
```

MessageBoard

- 把資料存在Mongodb
 - 留言 : Message
 - 欄位 : text, user(ref User)
 - 使用者 : User
 - name, email, password, description
 - 密碼需要加密 : pre save, comparePassword

MessageBoard

- 新增註冊功能 /register
 - /views/register.jade
 - 新增一筆user資料並回傳結果訊息
- /login
 - 於資料庫中找出該user
 - 比對密碼，成功則將user放入session中

MessageBoard

- 所有留言 /
 - messages中撈出所有留言
 - 記得populate，這樣user.name才會有值
- 我的留言 /mymessage
 - 找出collection中 user=req.session.user._id的留言

MessageBoard

- 修改package.json，增加額外兩個套件
 - bcrypt
 - mongoose

REST

- 軟體架構風格
 - 應用程式的狀態與功能為resources
 - 每個資源有獨立的URI
 - 資源透過 GET/POST/PUT/DELETE 實現CRUD操作
- 符合以上條件的系統稱作RESTful

RESTful 設計準則

● 資源

- 名詞為主：對應資料庫的collection
- URL： /資源名稱/資源id
 - 創建資源：POST /資源名稱/
 - 資源列表：GET /資源名稱/
 - 刪除所有資源：DELETE /資源名稱/
 - 批次修改資源：UPDATE /資源名稱/
 - 資源詳細資料：GET /資源名稱/資源ID
 - 修改資源： UPDATE /資源名稱/資源ID
 - 刪除資源： DELETE /資源名稱/資源ID
- 額外的filter加在query : paging, field等

RESTful 設計準則

Resource	POST create	GET read	PUT update	DELETE delete
/dogs	Create a new dog	List dogs	Bulk update dogs	Delete all dogs
/dogs/1234	Error	Show Bo	If exists update Bo If not error	Delete Bo

RESTful 設計準則

- 功能
 - 動詞為主：牽扯到許多資源的操作、不需要任何資源的操作
 - query裡包含功能操作所需的參數
 - 通常只需要支援GET即可
 - ex: /search?q=key
 - ex: /calculate?from=USD&to=TWD&amount=100

RESTful 設計準則

- 錯誤處理
 - 成功： 200
 - 客戶端資料有誤： 4xx
 - 400： 資料格式錯誤（新增或更新或操作功能時）
 - 401： 使用者沒有權限
 - 404： 無此資料
 - 同服端處理錯誤： 500
 - production環境時不需要讓使用者知道內部問題

重回 MessageBoard

- 資源
 - 使用者： users
 - 留言： messages
 - 我的留言： messages + filter
- 功能
 - 登入： login
 - 登出： logout

MessageBoard 使用者

- 使用者列表： GET /users
- 使用者資料： GET /users/:name
 - 404：無此使用者
- 修改使用者： PUT /users/:name
 - 404：無此使用者，400：格式錯誤
- 刪除使用者： DELETE /users/:name
 - 404：無此使用者

MessageBoard 留言

- 留言列表 : GET /messages
- 新增留言 : POST /messages
- 刪除留言 : DELETE /messages/:id
- 404 : 無此留言

MessageBoard 我的留言

- 留言列表： GET /messages
 - name=\$user.name
 - ex: /messages?name=fin
 - page=\$page
 - ex: /messages?name=fin&page=1

MessageBoard URL

- 功能
 - 登入 : /login
 - 登出 : /logout
 - 我的留言 : /mymessage

MessageBoard URL

- 功能
 - 登入 : /login
 - 登出 : /logout
 - 我的留言 : /mymessage