Database - P71

MongoDB

GridFS : solve distributed system lack of single file system
shared between all servers
mongofiles : 用于[redacted]实现与GridFS的交互
如 mongofiles -h <port> put <file_name>
用于上传指定的文件到<port>
mongofiles -h <port> list
用于列出当前<port>上管理的文件
GridFS 生成的 collection 与其他 collection 类似
can be replicated / querid from console

ODM (object data modeling), 对象数据建模
ORM (object relational modeling), 对象关系建模
allow for application-level enforcement of schema restriction for data
specify aspect of application design in one source repository
give control of how data in database organized to application

Mongoose : popular ODM library in Node.js
in Mongoose, everything derived by Schema
[redacted] 如 var mongoose = require('mongoose');
mongoose.connect(<URL>);
(schema with property name as String) var dbSchema = mongoose.Schema({name: String});
Model : class can create documents with
如 var Kitten = mongoose.model('Kitten', dbSchema)
var silence = new Kitten({name: 'Silence'});
console.log(silence.name) ⟶ 'Silence'
add function to one of document
如 dbSchema.methods.speak = function() { "I don't have name"
var greeting = this.name ? "Meow name is" + this.name :
console.log(greeting); }
var Kitten = mongoose.model('Kitten', dbSchema);
function added to 'methods' property of schema
get compiled into Model prototype
and exposed on each document instance

Database - P72

MongoDB

Mongoose : save : document can be saved to MongoDB

```
$o fluffy.save(function(err, fluffy) {
    if(err) return console.error(err);
    fluffy.speak(); })
```

can access all documents through model

```
$o Kitten.find(function(err, kittens) {
    if(err) return console.error(err);
    console.log(kittens); })
```

return result array to the callback

```
$o Kitten.find({name: /^fluff/}, callback);
```

API will return variable of type Query, when not sepitify callback
use query object to build up query
then execute with callback via exec()

```
$o var query = Athlete.find({'sport':'Tennis'});
query.select('name age');   等价于 SELECT name, age
query.limit(5);             等价于 LIMIT 5
query.sort({age:-1});       等价于 ORDER BY age DESC
query.exec(function(err, atheletes) {
    if(err) return handleError(err); })
```

can also use the where() function
and chain parts of our query together using dot operator (.)

```
Athlete.find()
  .where('sport').equals('Tennis')
  .where('age').gt(17).lt(50)
  .limit(5)
  .sort({age:-1})
  .select('name age')
  .exec(callback);
```

[SELECT name, age
[FROM Athelete
[WHERE sport='Tennis'
[AND age>17 AND age<50
[ORDER BY age DESC
[LIMIT 5

其中 callback is name of callbach function

Database - P73

Mongo DB

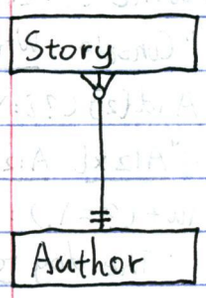
Mongoose : findById() / findByIdAndRemove() / findByIdAndUpdate()
find single document by Id and remove / update it
findOne() / findOneAndRemove() / findOneAndUpdate()
find single document by criteria and remove / update it

Schema in Mongoose can have any number of field of various SchemaType
ex var schema = new Schema({
name : String,
binary : Buffer,
living : Boolean,
updated : {type : Date, default : Date.now},
age : {type : Number, min : 18, max : [65, 'Too Old']},
mixed : Schema.Types.Mixed,
_someId : Schema.Types.ObjectId,
decimal : Schema.Types.Decimal128,
array : [],
arrayOf : [<Schema Type>],
nested : {stuff :
{type : String, lowercase : true, trim : true}},
map : Map,
mapOfString : {type : Map, of String} })
Mixed : arbitrary schema type
[] : array of items
can perform JavaScript array operation on these models
push / pop / unshift / ...
field <name> and field <type> as key-value pair
field <name> and <object> defining the type, as well as any other option
provide [redacted] built-in and custom validator (synchronous / asynchronous)
specify both acceptable range value and error message
default / required / min, max for number
String : automatically set to lowercase, uppercase, trimmed
enum : allowed value for String field
match : regular expression to match
minlength / maxlength

Database - P74

MongoDB

Mongoose : can create reference from one document/model instance to another
using schema field : ObjectId
or array of ObjectIds
field store id of related model
populate() : used in query to replace id with actual data
when need the actual content of associated document




```
eg var mongoose = require('mongoose');
   var Schema = mongoose.Schema;
   var authorSchema = Schema({
       name : String,
       stories : [{type : Schema.Types.ObjectId, ref:'Story'}]});
   var storySchema = Schema({
       author : {type : Schema.Types.ObjectId, ref: 'Author'},
       title : String });
   var Story = mongoose.model('Story', storySchema);
   var Author = mongoose.model('Author', authorSchema);
```

can save reference to related document by assigning the _id value

```
eg var bob = new Author({name : 'Bob Smith'});
   bob.save(function(err) {
     if (err) return handleError(err);
     var story = new Story({
       title : "Bob goes sledding",
       author : bob._id });
       story.save(function(err) {
         if (err) return handleError(err);});
   });
  Story.findOne({title : 'Bob goes sledding'})
    .populate('author')
    .exec(function(err, story) {
      if (err) return handleError(err);
      console.log('The author is %s', story.author.name); });
```

→ The author is Bob Smith

Database - P75

CouchDB

JSON and REST-based, document-oriented database
REST (representational state transfer, 表现层状态转换)
go on even when connectivity only rarely available
与MongoDB类似, CouchDB存储 document
JSON object consisting of key-value pairs
与MongoDB不同, no ad-hoc ability to query data
在CouchDB中, create indexed view
produced by incremental mapreduce operation
primary way to discover document in CouchDB

aim to support variety of deployment scenarios
written in Erlang, very resilient database
utilize append-only storage model
data virtually incorruptible
easy to replicate / backup / restore
call to CouchDB happen through RESTful interface
replication can be one-way / bidirectional, ad-hoc / continuous

using Fauxton web interface, go to http://localhost:5984/_utils/
use cURL to make REST call directly to CouchDB
all libraries and drivers for CouchDB ultimately send REST request

document in Fauxton consist of JSON object
containing key-value pair called field
_id field must be unique and can never be changed
can set _id explicitly / generate automatically
_rev field get new value every time given document change
format for revision string : integer | dash | psedo-random unique string
integer indicate numerical revision, start at 1
go {"_id": "2ac587...",
"_rev": "4-3d6295...",
"name": "somename",
...
}

Database - P76

CouchDB

no transaction / locking in CouchDB:
mechanism of requiring both _id and _rev is how CouchDB prevent conflict
ensure only most recent document revision modified
both _id and _rev fields reserved in CouchDB
both _id and the matching _rev must be provided
to update / delete existing document
CouchDB will reject operation if either missing
first read recording, take note of _id and _rev fields
then request update providing full document, plus _id and _rev
all operations fist come, first served
by requiring matching _rev, CouchDB ensure
document being modified not already altered
between time read and time updated

与MongoDB类似, CouchDB not limited to storing just string values
can handle JSON nested to any depth

all communication with CouchDB is REST-based, issuing command over HTTP
如: export COUCH_ROOT_URL = http://localhost:5984
用于定义一个可以在后续命令中调用的变量
curl -u admin:admin ${COUCH_ROOT_URL}
issue GET request (cURL default) to CouchDB
retrieving information about resource indicated in URL
root url of server, response information indicating CouchDB status/version
curl -u admin:admin "${COUCH_ROOT_URL}/<database>/<_id>"
retrieve information about specific document in specific database

method	path	description
GET, POST	/db/_local_docs	return list of all non-replicated documents
GET	/db/_local/id	return latest revision of non-replicated document
PUT	/db/_local/id	insert new version of non-replicated document
DELETE	/db/_local/id	delete non-replicated document
COPY	/db/_local/id	copy non-replicated document

Database - P77

CouchDB

GET request always safe in CouchDB
and any other correctly implemented RESTful service
CouchDB will not make any change to document as result of GET request

PUT / POST / DELETE : to make change to document , HTTP method

PUT : to update document / to create new document with specific _id
eg curl -u admin:admin -i -X PUT \
"${COUCH_ROOT_URL}/<db>/<_id>" \
-H "Content-Type : application/json" \ 设置 Content-Type
-H "If-Match: <_rev>" \
-d "<find_query>" , eg '{"name" : "Faker"}'

特别注意: MongoDB modify document in place
但 CouchDB force always overwrite entire document to make change
in the Fauxton UI, record whole document each time Save hit

DELETE: remove document from database
注意由于 CouchDB 实际上不在原有 document 上直接修改
DELETE operation will supply new revision number
even document "deleted", not actually removed from disk
new empty document appended, flagged as deleted

view : window into documents contained in database
consist of mapper / reducer function
used to generate list of ordered key-value pair
both key and value can be JSON document
_all_docs : simplest view created by default for all databases
contain entry for each document in database, keyed by _id
curl -u admin:admin "${COUCH_ROOT_URL}/<db>/_all_docs"
⟶ {"id" : "_design/random", "key" : "_design/random", "value" : {"..."}},
{"id" : "_design/test", "key" : "_design/test", "value" : {"rev" : "3-..."}},
...

Database - P78

CouchDB

view : cURL output array of row representing each document
- id : document's _id field
- key : JSON key produced by mapper function in mapreduce call
- value : associated JSON value produced through reducer function in mapreduce call

include_docs = true : to retrieve entire document
- by default only metadata for document in value field
- 如 curl -u admin:admin "${COUCH_ROOT_URL}/<db>/_all_docs?include_docs=true"
- can see all field for document returned in value object

emit(key, value) : key to output shown in the view
- map function may call emit() 0 or more time
- 如 默认的 map function : function(doc){ emit(doc._id, 1);}
- _all_docs 等价于 function(doc){ emit(doc._id, {rev: doc._rev}); }

design document : real document in the database
- can be replicated to other servers like any other data
- all design document start with _design/
- contain one or more view
  - index name : distinguish view from other
- 调用如 curl -u admin:admin "${COUCH_ROOT_URL}/<database_name>/_design/<design_doc>/_view/<index_name>

CouchDB ensure records retrieved from view
- alphabetically sorted by emitted key value (lexicographically)

can add query parameter key to URL, to select row contain exact key
- 如 curl -u admin:admin "... /<index_name>?key=<key>"

total_rows : represent count of total number of records in the view

offset : exactly how far into the full dataset the first record presented appear

Database - P79

CouchDB

import data : Ruby contain set of libraries make importing data easy
data structured XML format
easy to parse and import into CouchDB

couchrest : Ruby gem wrap all HTTP request of CouchDB
libxml-ruby to streaming parse XML input

query parameter	type	description
conflicts	boolean	include conflicts information in response default : false , ignored if include_docs : false
descending	boolean	return design document in descending by key order , default : false
(end_key) endkey	string	(optional) stop returning record when specified key reached
(end_key_doc_id) endkey_docid	string	stop returning record when specified design document ID reached
include_docs	boolean	include full content of design document in the return
inclusive_end	boolean	specify whether the specified end key included in the result default : true
key	string	return only document match specified key
keys	string	return only design document match specified keys
limit	number	limit number of returned design documents to specified number
skip	number	skip number of records before starting to return result
(start_key) startkey	string	return record starting with the specified key
(start_key_doc_id) startkey_docid	string	return record starting with specified design document ID
update_seq	boolean	response include update_seq value indicating sequence id of underlying database the view reflect

URL中特殊字符转义编码

space	"	#	%	&	(	)	+
%20	%22	%23	%25	%26	%28	%29	%2B
,	/	:	;	<	=	>	?
%2C	%2F	%3A	%3B	%3C	%3D	%3E	%3F
@	\	|	!	*	'	$	
%40	%5C	%7C	%21	%2A	%27	%24	
[	]						
%5B	%5D						

Database - P80

CouchDB

Step at high level when CouchDB build view

1. send document off to mapper function
2. collect all emitted values
3. sort emitted values by key
4. send chunk of rows with same key to reduce function
5. if too much data to handle all reduction in single call
   call the reduce function again with previously reduced values
6. repeat recursive call to reduce function
   as necessary until no duplicate key remain

reduce function : take three arguments : key, values, reduce

key : array of tuples

two element tuple : key emitted by the mapper

_id of the document produced

values : array of values corresponding to the key

reduce : Boolean value if the invocation is re-reduction

if true, key parameter will be null

call being sent result of previous reducer call

reduce call may be recursive

will go on as long as reduction to be done

most mapreduce system throw away output of mapper and reducer after work done

CouchDB will keep the intermediate mapper and reducer values

until change to document would invalidate the data

once view codified into design document

incremental mapreduce strategy : once change occur

CouchDB will incrementally run mapper and reducer

to correct for the updated data

will not start from scratch, recalculating everything each time

allow CouchDB to use view

as the primary indexing mechanism for data retrieval

Database - P81

CouchDB

Changes API : make CouchDB great candidate for "system of record"
CouchDB can act as central hub for all updates
providing single location systems can subscribe for changes to
make use of Changes API : Node.js
Node.js : event-driven, written in JavaScript
natural fit for integrating with CouchDB
curl -u admin:admin "${COUCH_ROOT_URL}/<db>/_changes"
can specify limit parameter to request subset of data
can add include_docs=true to cause full document returned
respond with everything when with no parameter
since : specify sequence ID to start from
can check peridically to find out change occured
polling (轮询) : fine solution if application accept delay between update
feed=longpoll: want update quicker without overhead of reopening connection
CouchDB leave connection open for period of time
waiting for change to happen before finishing response

Node.js : strongly event-driven system
driver watch for change and feed and emit change event
whenever CouchDB report changed document
can start off outputing record for each document in database
then eventually go silent waiting for future change

feed=continuous : send each change separately and keep connection open
instead of combining all changes into result array
and closing stream afterwards
return more JSON-serialized change notification object
as change become available
CouchDB eventually close connection if no change happened for a while
reduced overhead that accompany persistent HTTP connection
no time lost reestablishing HTTP connection
output more complex to parse since not straight JSON
communication style not well suited for web browser
web browser downloading feed asychronously may not receive any data until entire connection finish