课程目标

1. 学习 MongoDB，掌握 MongoDB 的基本使用方法；
2. 学习 PyMongo，掌握使用 PyMongo 连接 MongoDB，并学会使用相应 API 在 Tornado 上进行简单动态网页的开发；

使用工具

Notepad++ or sublime 与插件;
Chrome browser;
python2.7; pip 1.4; pylint;
MongoDB 2.2及以上版本; PyMongo;
Tornado 3.1;

预备知识：

1. Database

   http://en.wikipedia.org/wiki/Database

   对于二年级还没有上过数据库这门课的同学，需要你们对数据库有一个基本的了解。三年的同学可以略过。

2. NoSQL

   http://en.wikipedia.org/wiki/Nosql

   由于本课程将要使用的 MongoDB 是 NoSQL 数据库，对于三年级的同学认真理解这个概念很重要；对于二年级的同学，看不懂没关系，本门课程将会带你们入门，随着你们在后续高年级课程的学习中你们总会理解。

一、 MongoDB 入门

1. MongoDB 介绍

http://docs.mongodb.org/manual/core/introduction/

MongoDB 是一个高性能、开源的文档型数据库，使用 C++开发，是当前 NoSQL（Not Only SQL，非关系型数据库）数据库产品中最热门的一种，旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。它支持的数据结构非常松散，是类似 JSON 的 BSON 格式，因此可以存储比较复杂的数据类型。

MongoDB 是一个面向集合的、模式自由的文档型数据库。

1） 面向集合（Collection-Oriented）

数据被分组存储在数据集中，称为一个集合（collection）。每个集合在数据库中都有一个唯一的标识，并且可以包含无限数目的文档。集合的概念类似关系型数据库里的表，不同的是它不需要定义任何模式（schema）。

2） 模式自由（Schema-Free）

对于存储在 MongoDB 数据库中的文件，不需要知道它的任何结构定义。例如，下面两条记录可以存在于同一个集合里面：

{ “welcome” : “Beijing” }

{ “age” : 25 }

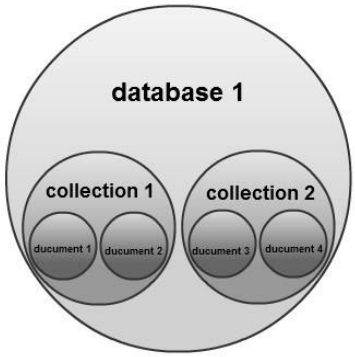
3） 文档型

http://docs.mongodb.org/manual/core/document/

MongoDB 存储的数据是“键-值”对的集合，键是字符串，值可以是数据类型集合里面的任意类型，包括数组和文档。这个数据格式称作 BSON。

2. MongoDB 数据逻辑组织结构

MongoDB 数据逻辑结构是一种层次结构，主要有文档（document）、集合（collection）、数据库（database）组成。如下图所示：



文档、集合、数据库三部分的关系如下：

a) MongoDB 的文档相当于关系数据库中的一条记录。
b) 多个文档组成一个集合，相当于关系数据库的表。
c) 多个集合逻辑上组织在一起就是数据库。
d) 一个 MongoDB 实例支持多个数据库。

3. BSON

http://bsonspec.org/

MongoDB 使用 BSON 来存储文档。BSON（Binary Serialized Document Format）是一种类 json 的一种二进制形式的存储格式，简称 Binary JSON，它和 JSON 一样，支持内嵌的文档对象和数组对象，但是 BSON 有 JSON 没有的一些数据类型，如 Date 和 BinData 类型。认识更多 BSON 支持的数据类型，可以参看链接：
http://docs.mongodb.org/manual/reference/bson-types/

4. MongoDB 的安装

选择合适自己操作系统的版本，进行 MongoDB 的安装，可以参考以下链接：
http://docs.mongodb.org/manual/installation/
课程实验室中的机器已经安装好了 MongoDB，请大家直接使用即可。

5. 使用 mongo shell 管理 MongoDB

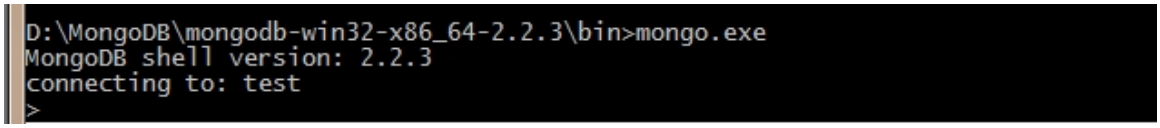
安装完成 MongoDB 后，就可以启动 MongoDB 了。以在 Windows 系统下为例，请参看以下链接：http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/

参照链接中 Run MongoDB/Start MongoDB 下的步骤，即可启动 MongoDB 数据库了。

```
D:\MongoDB\mongodb-win32-x86_64-2.2.3\bin>mongod.exe --dbpath D:\MongoDB\data\db
Thu Dec 19 13:12:37 [initandlisten] MongoDB starting : pid=10136 port=27017 dbpath=D:\MongoDB\data\db 64-bit host=CCmaster-PC
Thu Dec 19 13:12:37 [initandlisten] db version v2.2.3, pdfile version 4.5
Thu Dec 19 13:12:37 [initandlisten] git version: f570771a5d8a3846eb7586eaffcf4c2f4a96bf08
Thu Dec 19 13:12:37 [initandlisten] build info: windows sys.getwindowsversion(major=6, minor=1, build=7601, platform=2, service
Thu Dec 19 13:12:37 [initandlisten] options: { dbpath: "D:\MongoDB\data\db" }
Thu Dec 19 13:12:37 [initandlisten] journal dir=D:/MongoDB/data/db/journal
Thu Dec 19 13:12:37 [initandlisten] recover : no journal files present, no recovery needed
Thu Dec 19 13:12:37 [websvr] admin web console waiting for connections on port 28017
Thu Dec 19 13:12:37 [initandlisten] waiting for connections on port 27017
```

如上图所示，启动后的 MongoDB 在默认端口 27017 下等待链接。

继续参照链接中 Run MongoDB/Connect to MongoDB 下的步骤，另外再开启一个 Command Prompt（Windows 下就是 cmd.exe），完成对 MongoDB 的连接，此时即可进入 mongo shell 对 MongoDB 进行管理了。



如上图所示，MongoDB 默认已经创建了一个 test 数据库。

大多数情况下，我们对数据库的操作不外乎就是 CRUD(create, read, update, and delete) 操作，现在你可以练习在 mongo shell 使用 MongDB 的命令对其进行操作啦~ 请参考以下链接自己动手练习：http://docs.mongodb.org/manual/applications/crud/

Have a try and enjoy yourself!

二. PyMongo 入门

1. 什么是 PyMongo

MongoDB 支持很多编程语言接口对其进行集成应用开发，包括我们熟知的 C、C++、Java、C#、PHP、Perl、Ruby、Javascript 等等，当然也包括本门课程所学习的服务器端语言 Python。

PyMongo 是 Python 语言对 MongoDB 的一个驱动程序包，使用其给我们提供的 API，可以实现用 Python 语言访问 MongoDB。

2. PyMongo 的安装

PyMongo 的安装与其他 Python 包的安装没有不同，你可以使用 pip 或者 easy_install 对其进行安装，请参考链接：http://api.mongodb.org/python/current/installation.html

3. 学习使用 PyMongo API

安装完成 PyMongo 后，现在你可以使用 PyMongo 的 API 进行开发啦~ 作为入门，你可以参看下面这个链接，自己动手编写一些测试程序，熟悉使用 PyMongo：

http://api.mongodb.org/python/current/tutorial.html

当然在练习过程中，你可以随时查询 PyMongo 的 API 文档，以详细了解相应 API 的使用：http://api.mongodb.org/python/current/api/pymongo/index.html

Now, have a try and enjoy yourself!

三. 操作指导

#导入 pymongo 库

```
import pymongo
```

#创建一个到 mongodb 数据库的连接

```
conn = pymongo.Connection("localhost", 27017)
```

#使用 Connection 对象 conn 创建数据库，返回数据库对象

1) 对象属性 `db = conn.example` or

2) 字典 `db = conn['example']`

如果数据库不存在，则被自动建立

#数据库对象 db，调用 collection_names()方法获得数据库中的集合(Table)列表

```
db.collection_names()
```

输出 [ ] 为空，因为还没有添加任何集合

#在数据库对象 db 上通过访问集合名字的属性来获取代表【集合的对象 widgets】

```
widgets = db.widgets  or  widgets = db['widgets']
```

#调用集合对象的 insert 方法，制定一个 python dict 字典，插入文档 item

```
widgets.insert( {"foo":"bar"} )
```

因为 widgets 集合不存在，所以会在文档被添加的时候自动创建

这时再调用 `db.collection_names()`

得到`[u'widgets', u'system.indexes']`

Hint:（**`system.indexes`** 集合是 MongoDB 内部使用的。处于本章的目的，你可以忽略它。）

MongoDB 以“文档”的形式存储数据，相对自由的数据结构，是一个“无模式”数据库（同一个集合中的文档通常拥有相同的结构，但是 MongoDB 并不强制要求使用相同结构）。

MongoDB 以一种称为 BSON 的类似 JSON 的二进制形式存储文档。

pymongo 允许我们以 python 字典的形式写和取出文档

#创建一个新文档

使用字典作为参数调用集合的 insert 方法

```
widgets.insert( {"name":"flibnip", "description":"grade-a industrial flibnip", "quantiy":3} )
```

#取出文档 (获取 name=flibnip 的文档)

```
widgets.find_one( {"name":"flibnip"} )
```

结果=> { u'description': u'grade-A industrial flibnip',
u'_id': ObjectId('4eada3a4136fc4aa41000001'),
u'name': u'flibnip', u'quantity': 3 }

Hint: _id 域是自动添加的，它的值是一个 ObjectID，一种保证文档唯一的 BSON 对象

#更新文档

```
doc = db.widgets.find_one({"name": "flibnip"})
```

命令$：type(doc) =>输出 <type 'dict'> 那你知道 find_one 的返回类型了吧?

#修改指定值

```
doc['quantity'] = 4

db.widgets.find_one( {"name":"flibnip"} )
```

输出=> { u'_id': ObjectId('4eb12f37136fc4b59d000000'),
u'description': u'grade-A industrial flibnip',
u'quantity': *4*, u'name': u'flibnip' }

#遍历集合 widgets 的所有文档

```
for doc in widgets.find():
    print doc
```

#获得文档的一个子集 (获取 quantity=4 的文档)

```
for doc in widgets.find({"quantity":4})
    print doc
```

#删除文档

```
widgets.remove({"name":"flibnip"})
```

Ex1:

我们将要创建的应用是一个基于 Web 的简单字典。你发送一个指定单词的请求，然后返回这个单词的定义。一个典型的交互看起来是下面这样的：

```
$ curl http://localhost:8000/oarlock
{definition: "A device attached to a rowboat to hold the oars in place",
"word": "oarlock"}
```

这个 Web 服务将从 MongoDB 数据库中取得数据。具体来说，我们将根据 **word** 属性查询文档。在我们查看 Web 应用本身的源码之前，先让我们从 Python 解释器中向数据库添加一些单词。

```
>>> import pymongo
```

```
>>> conn = pymongo.Connection("localhost", 27017)
>>> db = conn.example
>>> db.words.insert({"word": "oarlock", "definition": "A device attached to
        a rowboat to hold the oars in place"})
ObjectId('4eb1d1f8136fc4be90000000')

>>> db.words.insert({"word": "seminomadic", "definition": "Only partial
        ly nomadic"})
ObjectId('4eb1d356136fc4be90000001')

>>> db.words.insert({"word": "perturb", "definition": "Bother, unsettle
        , modify"})
ObjectId('4eb1d39d136fc4be90000002')
```

代码清单 **definitions_readonly.py** 是我们这个词典 Web 服务的源码，在这个代码中我们查询刚才添加的单词然后使用其定义作为响应。

在命令行中像下面这样运行这个程序：

```
$ python definitions_readonly.py
```

现在使用 curl 或者你的浏览器来向应用发送一个请求。

```
$ curl http://localhost:8000/perturb
=> {"definition": "Bother, unsettle, modify", "word": "perturb"}
```

如果我们请求一个数据库中没有添加的单词，会得到一个404错误以及一个错误信息：

```
$ curl http://localhost:8000/snorkle
=> {"error": "word not found"}
```

那么这个程序是如何工作的呢？

让我们看看这个程序的主线。开始，我们在程序的最上面导入了 **import pymongo** 库。然后我们在我们的 TornadoApplication 对象的**__init__**方法中实例化了一个 pymongo 连接对象。我们在 **Application** 对象中创建了一个 **db** 属性，指向 MongoDB 的 **example** 数据库。下面是相关的代码：

```
conn = pymongo.Connection("localhost", 27017)
self.db = conn["example"]
```

一旦我们在 **Application** 对象中添加了 **db** 属性，我们就可以在任何 **RequestHandler** 对象中使用 **self.application.db** 访问它。实际上，这正是我们为了取出 pymongo 的 **words** 集合对象而在 **WordHandler** 中 **get** 方法所做的事情。

```
def get(self, word):
```

```
coll = self.application.db.words
word_doc = coll.find_one({"word": word})
if word_doc:
   del word_doc["_id"]
   self.write(word_doc)
else:
   self.set_status(404)
   self.write({"error": "word not found"})
```

在我们将集合对象指定给变量 **coll** 后，我们使用用户在 HTTP 路径中请求的单词调用 **find_one** 方法。如果我们发现这个单词，则从字典中删除 **_id** 键（以便 Python 的 **json** 库可以将其序列化），然后将其传递给 RequestHandler 的 **write** 方法。**write** 方法将会自动序列化字典为 JSON 格式。

如果 **find_one** 方法没有匹配任何对象，则返回 **None**。在这种情况下，我们将响应状态设置为404，并且写一个简短的 JSON 来提示用户这个单词在数据库中没有找到。

end Ex1:

Ex2:

我们例子的下一步是使 HTTP 请求网站服务时能够创建和修改单词。

它的工作流程是：发出一个特定单词的 POST 请求，将根据请求中给出的定义修改已经存在的定义。如果这个单词并不存在，则创建它。例如，创建一个新的单词：

```
$ curl -d definition=a+leg+shirt http://localhost:8000/pants
{"definition": "a leg shirt", "word": "pants"}
```

我们可以使用一个 GET 请求来获得已创建单词的定义：

```
$ curl http://localhost:8000/pants
{"definition": "a leg shirt", "word": "pants"}
```

我们可以发出一个带有一个单词定义域的 POST 请求来修改一个已经存在的单词（就和我们创建一个新单词时使用的参数一样）：

```
$ curl -d definition=a+boat+wizard http://localhost:8000/oarlock
{"definition": "a boat wizard", "word": "oarlock"}
```

代码清单 **definitions_write.py** 是我们的词典 Web 服务的读写版本的源代码。

除了在 **WordHandler** 中添加了一个 **post** 方法之外，这个源代码和只读服务的版本完全一

样。让我们详细看看这个方法吧：

```
def post(self, word):
    definition = self.get_argument("definition")
    coll = self.application.db.words
    word_doc = coll.find_one({"word": word})
    if word_doc:
        word_doc['definition'] = definition
        coll.save(word_doc)
    else:
        word_doc = {'word': word, 'definition': definition}
        coll.insert(word_doc)
    del word_doc["_id"]
    self.write(word_doc)
```

我们首先做的事情是使用 **get_argument** 方法取得 POST 请求中传递的 **definition** 参数。然后，就像在 **get** 方法一样，我们尝试使用 **find_one** 方法从数据库中加载给定单词的文档。如果发现这个单词的文档，我们将 **definition** 条目的值设置为从 POST 参数中取得的值，然后调用集合对象的 **save** 方法将改变写到数据库中。如果没有发现文档，则创建一个新文档，并使用 **insert** 方法将其保存到数据库中。无论上述哪种情况，在数据库操作执行之后，我们在响应中写文档（注意首先要删掉 **_id** 属性）。

end Ex2:

操作实践（不做你会后悔的，师兄温馨提醒）：

A)根据随堂测试二，完成注册功能（把注册消息写入数据库 Users 数据表）
B)修改登录功能，读取数据表 Users 数据，判断用户是否合法
C)用 python 脚本往数据库添加一个评论的数据表 Comments，往这个数据表 Comments 中添加一些评论，评论内容包括：评论者，评论时间，评论正文。
D)用 ajax 提交评论，提交评论不刷新全部页面；若提交成功，用 js 把添加的评论加到已显示的评论列表后面（使用部分刷新）