我们不需要为每一个应用请求创建一个新的Db对象，也不需要关心多个用户同时操作数据库服务器的情况：mongodb模块使用连接池来同时处理多个请求。我们可以通过调整Server构造器中的poolSize参数的值来选择MongoDB服务器同时连接的个数，如上述代码所示。

想要创建或者使用某个已经存在的数据库，可以将数据库名称传递给Db对象的构造器。如果数据库不存在，则会自动创建。

8.3.2 创建集合

正如前文所说，MongoDB中的集合等价于关系型数据库系统中的表，我们可以使用Db对象中的collection或者createCollection方法创建集合。通过在第二个参数中指定{safe:true}选项，可以控制如何处理存在或不存在的集合（见表8.1）。

表 8.1 创建

函 数	{ safe: true }	功 能
collection	是	若集合存在，打开集合；反之，返回错误
collection	否	若集合存在，打开集合；反之，在插入第一条数据时创建集合
createCollection	是	若集合不存在，创建集合；反之，返回错误
createCollection	否	若集合不存在，创建集合；反之，返回一个已存在的集合

想要使用{safe:true}选项，代码如下所示：

```
db.collection("albums", { safe: true }, function (err, albums) {
    // ... do stuff
});
```

下面这段代码是最常见的创建或打开集合的方式，但是它也非常简单：

```
db.collection("albums", function (err, albums) {
    if (err) {
        console.error(err);
        return;
    }

    // albums can now be used to do stuff ...
});
```

8.3.3 向集合中插入文档

如果要在集合中插入新数据，可以使用insert方法，如下所示：

```
var album = { _id: "italy2012",
              name: "italy2012",
              title: "Spring Festival in Italy",
              date: "2012/02/15",
              description: "I went to Italy for Spring Festival." };

albums.insert(album, { safe: true }, function (err, inserted_doc) {
    if (err && err.name == "MongoError" && err.code == 11000) {
        console.log("This album exists already, sorry.");
        return;
    } else if (err) {
        console.log("Something bad happened.");
        return;
    }
    // continue as normal
});
```

可以看到，代码中指定了文档中的_id字段。如果不指定，MongoDB会自动生成。如果已经有文档包含相同的_id值，回调函数就会返回错误信息。我们可以从代码中看到在insert函数中传递了{safe:true}参数选项，以确保写数据成功。但是，如果是为数据分析应用编写存储，则允许偶尔丢失数据，所以可以省略该选项。

我们还可以同时向集合中插入多份文档，只需向insert函数中传递数组即可：

```
var docs = [{ _id: "italy2012",
              name: "italy2012",
              title: "Spring Festival in Italy",
              date: "2012/02/15",
              description: "I went to Italy for Spring Festival." },
            { _id: "australia2010",
              name: "australia2010",
              title: "Vacation Down Under",
              date: "2010/10/20",
              description: "Visiting some friends in Oz!" },
            { _id: "japan2010",
              name: "japan2010",
              title: "Programming in Tokyo",
              date: "2010/06/10",
              description: "I worked in Tokyo for a while."
            }];
albums.insert(docs, { safe: true }, callback);
```

8.3.4 更新文档内容

要更新文档，可以调用集合中的update方法。第一个参数用来匹配一个或一组文档，第二个参数是对象描述（object description），表明如何修改已匹配的文档。对象描述包含命令格

式和需要修改的一个或多个字段：

```
photos.update({ filename: "photo_03.jpg", albumid: "japan2010" },
              { $set: { description: "NO SHINJUKU! BAD DOG!" } },
              { safe: true },
              callback);
```

该对象描述中，使用了$set命令，MongoDB会使用提供的新值更新字段。上述代码中，使用新值更新了description字段。MongoDB提供了许多不同的命令，其中有趣的更新命令如表8.2所示：

表 8.2　更新命令

命　令	功　能
$set	设置指定字段值
$inc	增加指定字段值
$rename	修改指定字段名称
$push	若字段为数组，向数组尾部添加新值
$pushAll	若字段为数组，向数组尾部添加多个新值
$pop	移除数组字段的最后一个元素
$pull	移除数组字段中的指定值

8.3.5　删除集合中的文档

要想删除集合中的数据，可以使用集合对象中的remove方法。可以指定一组字段来匹配一个或一组文档：

```
photos.remove({ filename: "photo_04.jpg", albumid: "japan2010" },
              { safe: true },
              callback);
```

如果不需要确认是否删除成功，可以跳过safe:ture选项和回调函数。

```
photos.remove({ filename: "photo_04.jpg", albumid: "japan2010" });
```

当然，也可以简单地通过调用不带任何参数的remove函数，删除集合中的所有文档：

```
photos.remove();      // DANGER ZONE!!
```

8.3.6　查询集合

到目前为止，之所以MongoDB成为最流行的NoSQL数据库引擎，是因为它能够像传统的SQL数据库查询一样在集合中找到文档。

而这一切，只需要集合对象中的find函数就能完成。

基本查询

开始之前需要注意，find方法本身并不做任何与查询相关的工作；它只是设置了查询结果的游标（cursor）（即可以用来迭代查询结果的对象）。在调用nextObject、each、toArray和streamRecords中任何一个函数之前，查询不会真正执行，游标的内容也没有真正生成出来。

顾名思义，前三个操作分别为：调用nextObject可以获得一个文档；调用each可以迭代查询到的文档数组；toArray可以获取所有文档，并作为回调函数中的参数：

```
var cursor = albums.find();

cursor.nextObject(function(err, first_match) {});
cursor.each(function(err, document) {});
cursor.toArray(function(err, all_documents) {});
```

如果直接调用不带任何参数的find方法，就会匹配集合中所有的文档。

我们可以使用游标上的streamRecords方法来创建Stream对象，这样就可以像使用其他数据流一样使用它：

```
var stream = collection.find).streamRecords();
stream.on("data", function(document) {});     // why data and not readable? See text!
stream.on("end", function() {});
```

这是获取大量数据集的最佳方式，因为相较于toArray这些方法而言，它占用更少的内存空间，而toArray则会在一个数据块中返回所有文档。在编写本书的时候，mongodb模块还没有更新到Node.js中最新的Stream事件模型"readable"，而这极有可能在你使用的时候已经更新了。因此，还是双击鼠标，亲自去验证一下吧（最好的选择莫过于驱动的官网了：http://mongodb.github.com/mongodb/node-mongodb-native/）。

要想在集合中找到指定的文档，可以在find函数的第一个参数中

指定需要匹配的字段：

```
photos.find({ albumid: "italy2012" }).toArray(function (err, results));
```

我们还可以在find查询中使用类似于前面update函数中的操作符。例如，现在有一些关于个人信息的文档，想要找到所有年龄大于20岁的人，可以使用：

```
users.find({ age: { $gt: 20 } }).toArray(callback);
```

还可以使用逻辑操作符$and或者$or将这些操作符组合起来，以提供更强大的查询功能。例如，想要返回所有年龄在20岁到40岁（包括在内）之间的用户，可以使用

```
users.find({ $and: [ { age: { $gte: 20 } }, { age: { $lte: 40 } } ] });
```

其他常见操作符可见表8.3。想要获取所有的操作符列表，可以检索MongoDB查询文档。

表 8.3　查询操作符

操　作　符	含　义
$ne	不相等
$lt	小于
$lte	小于或等于
$gt	大于
$gte	大于或等于
$in	若字段值在给定的数组中，则匹配
$nin	若字段值不在给定的数组中，则匹配
$all	若给定的字段是数组，且包含所有给定的数组值，则匹配

查询MongoDB生成的ID

正如我在前文中提到的一样，如果插入的文档中没有提供_id字段，MongoDB会自动为我们生成一个。这些自动生成的_id字段都是ObjectID类型，并且是24位16进制字符串。

但问题是，如果使用这些字段，则根本无法通过指定ID字符串的值来找到它们。需要将字符串包到ObjectID类的构造器里，如下所示：

```
var ObjectID = require('mongodb').ObjectID;
var idString = '50da9d8d138cbc5da9000012';
collection.find({_id: new ObjectID(idString)}, callback);
```

如果执行下面的代码，则无法得到任何匹配的结果：

```
collection.find({ _id: idString }, callback);
```

在本书应用的开发过程中不会遇到这个问题，因为我们使用的是自己的_id字段。

进一步改进查询

要将页面进行分页和排序，需要操作或者修改find操作的结果。在调用任何生成游标的函数前，mongodb模块可以让我们在find操作后面链式调用其他函数来实现这些功能。

最常用的方法包括skip、limit和sort。第一个方法指定在返回数据集前有多少文档需要跳过；第二个方法用来控制执行跳过后返回的文档数；最后一个用来整理和排序——支持在多个字段上的排序。

因此，要想将相册中所有的照片文档按照日期进行升序排列，代码如下：

```
photos.find({ albumid: "italy2012" })
    .sort({ date: 1})      // 1 is asc, -1 is desc
    .toArray(function (err, results));
```

假设每个页面有20张照片，要获取第三页的照片，需要使用下述代码：

```
photos.find({ albumid: "italy2012" })
    .sort({ date: 1})      // 1 is asc, -1 is desc
    .skip(40)
    .limit(20)
    .toArray(function (err, results) { });
```

同样，任何在toArray函数之前调用的函数只是设置结果游标，在最后一个函数调用之前，这些查询是不会被执行的。

我们还可以在sort函数的多个字段上排序：

```
collection.find()
    .sort({ field1: -1, field2: 1 })
    .toArray(callback);
```

8.4 更新相册应用

学会Node.js中使用MongoDB的基本操作之后，可以进一步更新相册应用，从而使用数据库替代简单的文件系统来存储相册和照片信息。这里，我们依然将图片文件存储在硬盘上，但在真实的生产环境中，可以将它们存储到一些存储服务器或者内容分发网络（content delivery network，CDN）中。

8.4.1 编写基本操作

首先，我们需要向应用中添加一个新文件夹data/，可以将一些后端的基本操作放在该文件夹下。我们创建一个名叫backend_helpers.js的文件，包含错误处理、生成错误、验证参数和一些其他的常用后端操作。这些操作都非常简单实用，你可以在Github上查看其源代码。

创建配置文件

在应用根目录下，创建一个名叫local.config.js的新文件，包含以下JavaScript：

```
exports.config = {
    db_config: {
        host: "localhost",
        // use default "port"
        poolSize: 20
    },

    static_content: "../static/"
};
```

只要引用该文件，就能确保所有的配置选项都能保存在一起。这样，只需要在这个文件中稍做修改，而不需要在代码中翻来覆去，就能更新配置。

创建数据库和集合

接下来，在data/下创建db.js文件。该文件用来创建相册应用所需的数据库连接和集合。同时，在该文件中，创建PhotoAlbums数据库连接也使用到了local.config.js文件中的配置信息：

```
var Db = require('mongodb').Db,
    Connection = require('mongodb').Connection,
    Server = require('mongodb').Server,
    async = require('async'),
    local = require("../local.config.js");

var host = local.config.db_config.host
    ? local.config.db_config.host
    : 'localhost';
var port = local.config.db_config.port
    ? local.config.db_config.port
    : Connection.DEFAULT_PORT;
var ps = local.config.db_config.poolSize
    ? local.config.db_config.poolSize : 5;

var db = new Db('PhotoAlbums',
                new Server(host, port,

                        { auto_reconnect: true,
                          poolSize: ps}),
            { w: 1 });
```

现在，要获取相册和照片的集合，需要打开数据库连接，并确保这些集合都存在。向db.js中添加相关函数，如下所示：

```
exports.init = function (callback) {
    async.waterfall([
        // 1. open database connection
        function (cb) {
            db.open(cb);
        },
        // 2. create collections for our albums and photos. if
        //    they already exist, then we're good.
        function (opened_db, cb) {
            db.collection("albums", cb);
        },
        function (albums_coll, cb) {
            exports.albums = albums_coll;
            db.collection("photos", cb);
        },
        function (photos_coll, cb) {
            exports.photos = photos_coll;
            cb(null);
        }
    // we'll just pass results back to caller, so can skip results fn
    ], callback);
};

exports.albums = null;
exports.photos = null;
```

我们可以看到albums和photos已经成为db.js中的输出对象，因

此可以在任何时候获取到它们：

```
var db = require('./db.js');
var albums = db.albums;
```

最后，需要确保db.init函数在应用启动前已经被调用，因此，需要将server.js中的调用

```
app.listen(8080);
```

替换成

```
db.init(function (err, results) {
    if (err) {
        console.error("** FATAL ERROR ON STARTUP: ");
        console.error(err);
        process.exit(-1);
    }
    app.listen(8080);
});
```

创建相册

而创建新相册的基本代码如下所示：

```
exports.create_album = function (data, callback) {
    var final_album;
    var write_succeeded = false;
    async.waterfall([
        function (cb) {
            try {
                backhelp.verify(data,
                                [ "name", "title", "date", "description" ]);
                if (!backhelp.valid_filename(data.name))
                    throw invalid_album_name();
            } catch (e) {
                cb(e);
            }
            cb(null, data);
        },

        // create the album in mongo.
        function (album_data, cb) {
            var write = JSON.parse(JSON.stringify(album_data));
            write._id = album_data.name;
            db.albums.insert(write, { w: 1, safe: true }, cb);
        },

        // make sure the folder exists.
        function (new_album, cb) {
            write_succeeded = true;
            final_album = new_album[0];
            fs.mkdir(local.config.static_content
                     + "albums/" + data.name, cb);
        }
    ],
    function (err, results) {
        if (err) {
            if (write_succeeded)
                db.albums.remove({ _id: data.name }, function () {});
            if (err instanceof Error && err.code == 11000)
                callback(backhelp.album_already_exists());
            else if (err instanceof Error && err.errno != undefined)
                callback(backhelp.file_error(err));
            else
                callback(err);
        } else {
            callback(err, err ? null : final_album);
        }
    });
};
```

尽管async模块让代码看起来有点“长”，但我们能看出代码变得整洁了许多。所有的异步操作都被表示成任务序列，之后async会为我们打理所有的细节！

这里可以使用一个小技巧来克隆一个对象：

```
var write = JSON.parse(JSON.stringify(album_data));
```

可以看出，序列化后再反序列化一个对象是JavaScript中最快的克隆对象的方式之一。之所以在之前的代码中克隆对象，是因为我们不想修改“不属于我们”的对象本身。直接修改函数中的对象是不合适的（或者说是彻底错误的），因此在添加_id字段前会先快速克隆

该对象。注意，backend_helpers.js类似于前文中的helpers.js，只是简单的后端（在data/文件夹下）帮助函数。

查询相册

使用指定的名称来查询一个相册，非常简单：

```
exports.album_by_name = function (name, callback) {
    db.albums.find({ _id: name }).toArray(function (err, results) {
        if (err) {
            callback(err);
            return;
        }

        if (results.length == 0) {
            callback(null, null);
        } else if (results.length == 1) {
            callback(null, results[0]);
        } else {
            console.error("More than one album named: " + name);
            console.error(results);
            callback(backutils.db_error());
        }
    });
};
```

从代码中可以看出，我们一般会将更多的时间花在错误处理和验证上。

相册列表

类似的，列出所有的相册信息也非常简单：

```
exports.all_albums = function (sort_field, sort_desc, skip, count, callback) {
    var sort = {};
    sort[sort_field] = sort_desc ? -1 : 1;
    db.albums.find()
        .sort(sort)
        .limit(count)
        .skip(skip)
        .toArray(callback);
};
```

获取相册中的照片

获取指定相册下的所有照片信息，同样轻而易举：

```
exports.photos_for_album = function (album_name, pn, ps, callback) {
    var sort = { date: -1 };
    db.photos.find({ albumid: album_name })
        .skip(pn)
        .limit(ps)
        .sort("date")
        .toArray(callback);
};
```

向相册中添加照片

事实上，稍微有些复杂的操作是将照片添加到相册中。该功能会多花费一些时间，因为需要将上传的临时文件拷贝到最终的static/albums/文件夹下：

```
exports.add_photo = function (photo_data, path_to_photo, callback) {
    var final_photo;
    var base_fn = path.basename(path_to_photo).toLowerCase();
    async.waterfall([
        function (cb) {
            try {
                backhelp.verify(photo_data,
                                [ "albumid", "description", "date" ]);
                photo_data.filename = base_fn;
                if (!backhelp.valid_filename(photo_data.albumid))
                    throw invalid_album_name();
            } catch (e) {
                cb(e);
            }
            cb(null, photo_data);
        },

        // add the photo to the collection
        function (pd, cb) {
            pd._id = pd.albumid + "_" + pd.filename;
            db.photos.insert(pd, { w: 1, safe: true }, cb);
        },

        // now copy the temp file to static content
        function (new_photo, cb) {
            final_photo = new_photo[0];
            var save_path = local.config.static_content + "albums/"
                + photo_data.albumid + "/" + base_fn;
            backhelp.file_copy(path_to_photo, save_path, true, cb);
        }
    ],
    function (err, results) {
        if (err && err instanceof Error && err.errno != undefined)
            callback(backhelp.file_error(err));
        else
            callback(err, err ? null : final_photo);
    });
};
```

8.4.2 修改JSON服务器的API

接下来，需要为JSON服务器添加两个新的API函数，以便创建相册和添加照片：

```
app.put('/v1/albums.json', album_hdlr.create_album);
app.put('/v1/albums/:album_name/photos.json', album_hdlr.add_photo_to_album);
```

还好express让这个过程变得非常简单，添加这两行代码后，API就扩展了新功能，现在需要更新相册处理程序来支持这些新

特性。

由于API现在还不支持上传数据，包括文件和POST数据，因此需要添加一些其他的中间件来支持这个功能。于是，将这些代码添加到server.js文件顶部：

```
app.use(express.logger('dev'));
app.use(express.bodyParser({ keepExtensions: true }));
```

代码中，我们添加了日志功能和bodyParser功能，bodyParser可以将请求的数据保存到req.body和req.files对象中。注意，需要在bodyParser中间件中指定保留文件扩展名。默认情况下，它会移除扩展名。

8.4.3 更新处理程序

现在，我们已经为相册和照片操作添加了数据库功能，但是还需要修改相册处理程序，以替换现有的文件系统操作。

帮助类

首先，需要创建两个帮助类。其中Photo类如下所示：

```
function Photo (photo_data) {
    this.filename = photo_data.filename;
    this.date = photo_data.date;
    this.albumid = photo_data.albumid;
    this.description = photo_data.description;
    this._id = photo_data._id;
}
Photo.prototype._id = null;
Photo.prototype.filename = null;
Photo.prototype.date = null;
Photo.prototype.albumid = null;
Photo.prototype.description = null;
Photo.prototype.response_obj = function() {
    return {
        filename: this.filename,
        date: this.date,
        albumid: this.albumid,
        description: this.description
    };
};
```

其中最有趣的函数当属response_obj函数。因为理论上Photo类可以包含一个照片所有的信息，而当把它作为JSON数据传递给API调用者时，有些数据是不想包含在其中的。假设有一个User对象，我们一般都会剔除密码和其他敏感数据。

一个基本的Album对象应当如下所示：

```
function Album (album_data) {
    this.name = album_data.name;
    this.date = album_data.date;
    this.title = album_data.title;
    this.description = album_data.description;
    this._id = album_data._id;
}

Album.prototype._id = null;
Album.prototype.name = null;
Album.prototype.date = null;
Album.prototype.title = null;
Album.prototype.description = null;

Album.prototype.response_obj = function () {
    return { name: this.name,
             date: this.date,
             title: this.title,
             description: this.description };
};
```

接下来，让我们看下处理程序如何使用前文中所写的相册基本操作。

创建相册

编写应用过程中，检测和捕捉错误的代码量往往要大于正常操作的代码量，这才是良好的代码风格。很多书籍和教程都忽略了这点，也许这就是世界上有那么多糟糕代码的原因之一。

```
var album_data = require('../data/album.js');
// ... etc ...
exports.create_album = function (req, res) {
    async.waterfall([
        function (cb) {
            if (!req.body || !req.body.name || !helpers.valid_filename(req.body.name))
{
                cb(helpers.no_such_album());
                return;
            }
            cb(null);
        },
        function (cb) {
            album_data.create_album(req.body, cb);
        }
    ],
    function (err, results) {
        if (err) {
            helpers.send_failure(res, err);
        } else {
            var a = new Album(results);
```

```
            helpers.send_success(res, {album: a.response_obj() });
        }
    });
};
```

根据名称检索相册

再次强调，错误检测和处理占了所有工作量的百分之九十以上。这里，我高亮显示了调用后端获取相册数据的代码：

```
exports.album_by_name = function (req, res) {
    async.waterfall([
        function (cb) {
            if (!req.params || !req.params.album_name)
                cb(helpers.no_such_album());
            else
                album_data.album_by_name(req.params.album_name, cb);
        }
    ],
    function (err, results) {
        if (err) {
            helpers.send_failure(res, err);
        } else if (!results) {
            helpers.send_failure(res, helpers.no_such_album());
        } else {
            var a = new Album(album_data);
            helpers.send_success(res, { album: a.response_obj() });
        }
    });
};
```

相册列表

在相册应用中，每次只获取25条相册数据，这样页面不会太复杂。如果需要，可以将其改成通过查询参数进行设置：

```
exports.list_all = function (req, res) {
    album_data.all_albums("date", true, 0, 25, function (err, results) {
        if (err) {
            helpers.send_failure(res, err);
        } else {
            var out = [];
            if (results) {
                for (var i = 0; i < results.length; i++) {
                    out.push(new Album(results[i]).response_obj());
                }
            }
            helpers.send_success(res, { albums: out });
        }
    });
};
```

让处理程序和数据库代码分离开来会增加一些额外的工作（其实并不多），但是这样会让后端变得更加灵活。在下一章中，我们会将相册和照片的数据存储迁移到另一个数据库系统中，而处理程序却不

需要做任何修改！只需要修改data/文件夹下的类即可。

获取相册中所有照片

代码清单8.1中展示的是如何浏览相册中的照片。它包含两个新方法：exports.photos_for_album和Album对象中的photos函数。这些函数中最复杂的部分就是处理分页和切割照片数组。

代码清单8.1　获取相册中所有照片

```
Album.prototype.photos = function (pn, ps, callback) {
    if (this.album_photos != undefined) {
        callback(null, this.album_photos);
        return;
    }
    album_data.photos_for_album(
        this.name,
        pn, ps,
        function (err, results) {
            if (err) {
                callback(err);
                return;
            }
            var out = [];
            for (var i = 0; i < results.length; i++) {
                out.push(new Photo(results[i]));
            }
            this.album_photos = out;
            callback(null, this.album_photos);
        }
    );
};

exports.photos_for_album = function(req, res) {
    var page_num = req.query.page ? req.query.page : 0;
    var page_size = req.query.page_size ? req.query.page_size : 1000;

    page_num = parseInt(page_num);
    page_size = parseInt(page_size);
    if (isNaN(page_num)) page_num = 0;
    if (isNaN(page_size)) page_size = 1000;

    var album;
    async.waterfall([
        function (cb) {
            // first get the album.
            if (!req.params || !req.params.album_name)
                cb(helpers.no_such_album());
            else
```

```
            album_data.album_by_name(req.params.album_name, cb);
        },

        function (album_data, cb) {
            if (!album_data) {
                cb(helpers.no_such_album());
                return;
            }
            album = new Album(album_data);
            album.photos(page_num, page_size, cb);
        },
        function (photos, cb) {
            var out = [];
            for (var i = 0; i < photos.length; i++) {
                out.push(photos[i].response_obj());
            }
            cb(null, out);
        }
    ],
    function (err, results) {
        if (err) {
            helpers.send_failure(res, err);
            return;
        }
        if (!results) results = [];
        var out = { photos: results,
                    album_data: album.response_obj() };
        helpers.send_success(res, out);
    });
};
```

添加照片

最后，编写添加照片的API，如代码清单8.2所示。该API会向Album对象中添加一个新方法。

代码清单8.2　使用API添加照片

```
Album.prototype.add_photo = function (data, path, callback) {
    album_data.add_photo(data, path, function (err, photo_data) {
        if (err)
            callback(err);
        else {
            var p = new Photo(photo_data);
            if (this.all_photos)
                this.all_photos.push(p);
            else
                this.app_photos = [ p ];
            callback(null, p);
        }
    });
};
```

```
exports.add_photo_to_album = function (req, res) {
    var album;
    async.waterfall([
        function (cb) {
            if (!req.body)
                cb(helpers.missing_data("POST data"));
            else if (!req.files || !req.files.photo_file)
                cb(helpers.missing_data("a file"));
            else if (!helpers.is_image(req.files.photo_file.name))
                cb(helpers.not_image());
            else
                album_data.album_by_name(req.params.album_name, cb);
        },

        function (album_data, cb) {
            if (!album_data) {
                cb(helpers.no_such_album());
                return;
            }
            album = new Album(album_data);
            req.body.filename = req.files.photo_file.name;
            album.add_photo(req.body, req.files.photo_file.path, cb);
        }
    ],
    function (err, p) {
        if (err) {
            helpers.send_failure(res, err);
            return;
        }
        var out = { photo: p.response_obj(), album_data: album.response_obj() };
        helpers.send_success(res, out);
    });
};
```

8.4.4 为应用添加新页面

目前为止，JSON服务器已经使用MongoDB完成应用中相册和照片的存储功能。剩下需要做的就是添加几个新页面，可以让用户通过Web界面创建新相册或向相册中添加新照片。现在，我们着手解决这个问题。

定义页面URL

这里，将两个新页面分别放到文件夹/pages/admin/add_album和/pages/admin/add_photo下。幸运的是，我们不需要为此在express应用中修改URL处理程序。

创建相册

不要忘记，在使用Mustache模板的网站中，每个页面都需要两个文件：

- JavaScript加载器
- HTML模板文件

添加相册页面的加载器代码微不足道，因为只需要模板文件，而不需要从服务器加载任何JSON数据，如代码清单8.3所示。

代码清单8.3　admin_add_album.js

```
$(function(){

    var tmpl,   // Main template HTML
    tdata = {};  // JSON data object that feeds the template

    // Initialize page
    var initPage = function() {

        // Load the HTML template
        $.get("/templates/admin_add_album.html", function (d){
            tmpl = d;
        });

        // When AJAX calls are complete parse the template
        // replacing mustache tags with vars
        $(document).ajaxStop(function () {
            var renderedPage = Mustache.to_html( tmpl, tdata );
            $("body").html( renderedPage );
        })
    }();
});
```

添加相册的HTML页面代码有一点复杂，因为需要用JavaScript实现Ajax表单提交，如代码清单8.4所示。而代码中的dateString变量是为了确保时间格式一直是yyyy/mm/dd，而不会偶尔出现yyyy/m/d的情况。

代码清单8.4　admin_add_album.html

```
<form name="create_album" id="create_album"
      enctype="multipart/form-data"
      method="PUT"
      action="/v1/albums.json">

 <h2> Create New Album: </h2>
 <dl>
  <dt>Album Name:</dt>
  <dd><input type='text' name='name' id="name" size='30'/></dd>
  <dt>Title::</dt>
  <dd><input id="photo_file" type="text" name="title" size="30"/></dd>
  <dt>Description:</dt>
  <dd><textarea rows="5" cols="30" name="description"></textarea></dd>
 </dl>
 <input type="hidden" id="date" name="date" value=""/>
</form>
```

```
<input type="button" id="submit_button" value="Upload"/>

<script type="text/javascript">

  $("input#submit_button").click(function (e) {
      var m = new Date();
      var dateString =
          m.getUTCFullYear() +"/"+
          ("0" + (m.getUTCMonth()+1)).slice(-2) +"/"+
          ("0" + m.getUTCDate()).slice(-2) + " " +
          ("0" + m.getUTCHours()).slice(-2) + ":" +
          ("0" + m.getUTCMinutes()).slice(-2) + ":" +
          ("0" + m.getUTCSeconds()).slice(-2);

      $("input#date").val(dateString);

      var json = "{ \"name\": \"" + $("input#name").val()
          + "\", \"date\": \"" + $("input#date").val()
          + "\", \"title\": \"" + $("input#title").val()
          + "\", \"description\": \"" + $("textarea#description").val()
          + "\" }";

      $.ajax({
          type: "PUT",
          url: "/v1/albums.json",
          contentType: 'application/json',    // request payload type
          "content-type": "application/json",   // what we want back
          data: json,
          success: function (resp) {
              alert("Success! Going to album now");
              window.location = "/pages/album/" + $("input#name").val();
          }
      });
  });

</script>
```

向相册中添加照片

要想向相册中添加一张新照片，必须编写更加复杂的代码。在加载器中，需要一个所有相册的列表，这样用户可以选择添加照片的相册，如代码清单8.5所示。

代码清单8.5　admin_add_photo.js

```
$(function(){

    var tmpl,   // Main template HTML
    tdata = {};  // JSON data object that feeds the template

    // Initialize page

    var initPage = function() {

        // Load the HTML template
        $.get("/templates/admin_add_photos.html", function(d){
            tmpl = d;
        });

        // Retrieve the server data and then initialize the page
        $.getJSON("/v1/albums.json", function (d) {
            $.extend(tdata, d.data);
        });

        // When AJAX calls are complete parse the template
        // replacing mustache tags with vars
        $(document).ajaxStop(function () {
            var renderedPage = Mustache.to_html( tmpl, tdata );
            $("body").html( renderedPage );
        })
    }();
});
```

最后，需要花些时间看一下Github代码库中第8章

create_album/文件夹下的HTML页面代码，看下表单和上传文件到服务器的代码（admin_add_photo.html）。该文件的最大亮点就是使用了XmlHttpRequest对象的FormData扩展来实现Ajax文件上传功能，如下所示：

```
$("input#submit_button").click(function (e) {
    var m = new Date();
    var dateString = /* process m -- see GitHub */
    $("input#date").val(dateString);

    var oOutput = document.getElementById("output");
    var oData = new FormData(document.forms.namedItem("add_photo"));
      var oReq = new XMLHttpRequest();
      var url = "/v1/albums/" + $("#albumid").val() + "/photos.json";
      oReq.open("PUT", url, true);
      oReq.onload = function(oEvent) {
          if (oReq.status == 200) {
              oOutput.innerHTML = "\
Uploaded! Continue adding or <a href='/pages/album/"
                  + $("#albumid").val() + "'>View Album</a>";
          } else {
              oOutput.innerHTML = "\
Error " + oReq.status + " occurred uploading your file.<br \/>";
          }
      };

      oReq.send(oData);
  });
```

FormData非常强大和神奇，但在低版本的Internet Explorer（IE 10之前）浏览器中不支持。Firefox、Chrome和Safari都已经支持它了。如果想要在旧版本的IE浏览器中支持Ajax文件上传功能，可以尝试其他文件上传方法，如使用Flash或者传统的HTML表单上传。

8.5 应用结构回顾

至此，应用已经变得有些复杂了，需要花些时间回顾下整个应用是如何组织架构的。将所有的静态内容移至static/文件夹下，并将代码都放置到app/文件夹下，因此我们拥有如下基本结构：

static/文件夹包含以下几个子文件夹：

- albums/——包含所有相册及图片文件
- content/——包含样式表（CSS）和渲染页面模板所需的JavaScript加载器文件
- templates/——浏览器渲染页面所需的HTML模板

在app/文件夹下，拥有：

- ./——包含核心的服务器端脚本和package.json文件
- data/——和后端数据存储相关的所有代码种类
- handlers/——包含所有处理客户端请求的代码

从本章起，所有版本的应用都会使用以上所示代码结构。

8.6 小结

现在，我们不仅升级了相册应用的版本，它使用MongoDB存储相册和照片数据；还多了一些有趣的页面，可以用来创建相册和上传照片到服务器。

唯一的缺陷就是所有人都可以访问这些页面，并使用API处理相册和照片。所以，接下来我们需要把目光转向添加用户权限上，以确保用户登录后才能使用应用。

第9章　数据库II：SQL（MySQL）

虽然NoSQL数据库在迅速普及，但我们仍然有许多理由继续使用关系型数据库，它们一如既往地受欢迎，特别是最常用的两个开源数据库：MySQL和PostgreSQL。好消息是，在Web应用中，Node.js的异步特性能够与这些数据库完全吻合，同时还有一个优秀的npm模块能够支持它们。

在本章中，我们会介绍如何在Node中使用MySQL和npm中的mysql模块。因为在上一章中，我们已经学习了如何将相册和照片数据存储到数据库，现在可以将关注点转向应用中的用户注册，并要求用户在创建任何相册或者添加照片之前必须处于登录状态。

即使你不打算使用诸如MySQL这样传统的关系型数据库，但本章仍然值得一读，因为本章会介绍express的一系列重要特性，还会讨论资源池（resource pooling）——一种控制和限制系统宝贵资源的方法。本章还会升级相册示例，以便相册和照片可以和MySQL一起工作。

9.1 准备工作

在Node.js中使用MySQL需要先做两件事情：确保MySQL已经安装并且安装了mysql npm模块。

9.1.1 安装MySQL

如果还没有在开发机器上安装MySQL，可以访问dev.mysql.com/downloads/mysql并下载合适的社区版服务器。对于Windows和Mac OS X系统，可以直接下载安装包；而对于Linux和其他的Unix系统，则解压.tar.gz文件到合适的位置（通常是/usr/local/mysql）。

如果在Windows和Mac OS X上运行安装包，请注意安装的细节，而对于二进制发行包，则需要阅读INSTALL-BINARY文本文件并按照其中的步骤来k完整安装并运行MySQL。当完成之后，应该能够通过命令提示符或者终端启动并运行mysql命令：

```
Kimidori:Learning Node marcw$ /usr/local/mysql/bin/mysql -u root
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.5.29 MySQL Community Server (GPL)

mysql>
```

9.1.2 从npm添加mysql模块

要为Node.js安装mysql模块，可以修改package.json并添加以下所示依赖：

```
"dependencies": {
  "async": "0.1.x",
  "mysql": "2.x"
}
```

现在应该能在项目根目录的node_modules/下看到mysql/目录。请注意，2.x系列的mysql模块相较于0.x系列有了显著的性能提