基于 C++ 构建的图片服务器

重要知识点

1. 简单的Web服务器设计能力
2. C/C++ 操作 MySQL 数据库
3. 数据库设计
4. Restful 风格 API
5. json 和 jsoncpp 的使用
6. 强化 HTTP 协议的理解
7. cpp-httplib 的使用和原理
8. 正则表达式
9. 基于 md5 进行校验
10. Postman 工具的使用
11. 软件测试的基本思想和方法

整体架构

核心就是一个 HTTP 服务器, 提供对图片的增删改查能力.

同时搭配简单的页面辅助完成图片上传/展示.

数据库设计

```
create database if not exists image_system;
use image_system;
```

创建图片表

```
drop table if exists `image_table`
create table `image_table`(image_id int not null primary key auto_increment,
                           image_name varchar(50),
                           size bigint,
                           upload_time varchar(50),
                           md5 varchar(128),
                           content_type varchar(50) comment '图片类型',
                           path varchar(1024) comment '图片所在路径')
```

> 什么是 md5?
>
> 这是一种常见字符串 hash 算法, 具有三个特性:
>
> 1. 不管源字符串多长, 得到的最终 md5 值都是固定长度

> 2. 源字符串稍微变化一点点内容, md5 值会变化很大(降低冲突概率)
> 3. 通过原字符串很容易计算得到 md5 值, 但是根据 md5 推导出原字符串很难(几乎不可能).

使用 MySQL C API 操作数据库

安装 MySQL C API

```
yum install mysql-devel
```

代码中使用时需要链接上 MySQL 提供的库

```
-L /usr/lib64/mysql -lmysqlclient
```

数据库插入数据

mysql_insert.cc

```
// 1. 初始化句柄
// 2. 建立连接
// 3. 设置编码格式
// 4. 拼装 SQL 语句
// 5. 执行 SQL 语句
// 6. 关闭句柄
#include <cstdio>
#include <cstdlib>
#include <memory>
#include <mysql/mysql.h>

int main() {
  // 1. 初始化句柄
  MYSQL* connect_fd = mysql_init(NULL);
  // 2. 建立链接
  // mysql_init 返回的指针
  // 主机地址
  // 用户名
  // 密码
  // 数据库名
  // 端口号
  // unix_socket
  // client_flag
  if (mysql_real_connect(connect_fd, "127.0.0.1", "root", "",
                                                "image_system", 3306, NULL, 0) == NULL) {
    printf("连接失败! %s\n", mysql_error(connect_fd));
    return 1;
  }
  // 3. 设置编码格式
  mysql_set_character_set(connect_fd, "utf8");
  // 4. 拼装 SQL 语句
  char sql[4096] = {0};
  char image_name[] = "滑稽.jpg";
```

```
  int size = 16 * 1024;
  char upload_time[] = "2019/05/14";
  char md5[] = "123456";
  char content_type[] = "jpg";
  char path[] = "./滑稽.jpg";
  sprintf(sql, "insert into image_table values(null, '%s', %d, '%s', '%s', '%s', '%s')",
      image_name, size, upload_time, md5, content_type, path);
  // 5. 执行 SQL 语句
  int ret = mysql_query(connect_fd, sql);
  if (ret != 0) {
    printf("执行 sql 失败! %s\n", mysql_error(connect_fd));
    return 1;
  }
  // 6. 关闭句柄
  mysql_close(connect_fd);
  printf("执行成功!\n");
  return 0;
}
```

数据库查找数据

mysql_select.cc

```
// 1. 初始化句柄
// 2. 建立连接
// 3. 设置编码格式
// 4. 拼装 SQL 语句
// 5. 执行 SQL 语句
// 6. 遍历查询结果
// 7. 释放结果集
// 8. 关闭句柄
#include <cstdio>
#include <cstdlib>
#include <mysql/mysql.h>

int main() {
  // 1. 初始化句柄
  MYSQL* connect_fd = mysql_init(NULL);
  // 2. 建立链接
  // mysql_init 返回的指针
  // 主机地址
  // 用户名
  // 密码
  // 数据库名
  // 端口号
  // unix_socket
  // client_flag
  if (mysql_real_connect(connect_fd, "127.0.0.1", "root", "",
                                                "image_system", 3306, NULL, 0) == NULL) {
    printf("连接失败! %s\n", mysql_error(connect_fd));
    return 1;
  }
  // 3. 设置编码格式
```

```
  mysql_set_character_set(connect_fd, "utf8");
  // 4. 拼装 SQL 语句
    char sql[1024 * 4] = {0};
    sprintf(sql, "select * from image_table");
  // 5. 执行 SQL 语句
  int ret = mysql_query(connect_fd, sql);
  if (ret < 0) {
    printf("执行 sql 失败! %s\n", mysql_error(connect_fd));
    return 1;
  }
  // 6. 遍历查询结果
  MYSQL_RES* result = mysql_store_result(connect_fd);
  if (result == NULL) {
    printf("获取结果失败! %s\n", mysql_error(connect_fd));
    return 1;
  }
  //  a) 获取行数和列数
  int rows = mysql_num_rows(result);
  int fields = mysql_num_fields(result);
  printf("rows: %d, fields: %d\n", rows, fields);
  //  b) 打印结果
  for (int i = 0; i < rows; ++i) {
    MYSQL_ROW row = mysql_fetch_row(result);
    for (int j = 0; j < fields; ++j) {
      printf("%s\t", row[j]);
    }
    printf("\n");
  }
  // 7. 释放结果集
  mysql_free_result(result);
  // 8. 关闭句柄
  mysql_close(connect_fd);
  printf("执行成功!\n");
  return 0;
}
```

服务器 API 设计

新增图片

```
请求:
POST /image
Content-Type: application/x-www-form-urlencoded

------WebKitFormBoundary5muoelvEmAAVUyQB
Content-Disposition: form-data; name="filename"; filename="图标.jpg"
Content-Type: image/jpeg

......[图片正文].....
```

```
响应:
HTTP/1.1 200 OK
{
    "ok": true,
}
```

查看所有图片元信息

```
请求:
GET /image/

HTTP/1.1 200 OK
[
    {
        "image_id": 1,
        "image_name": "1.png",
        "content_type": "image/png",
        "md5": "[md5值]"
    }
]
```

查看指定图片元信息

```
请求:
GET /image/:image_id

响应:
HTTP/1.1 200 OK
{
    "image_id": 1,
    "image_name": "1.png",
    "content_type": "image/png",
    "md5": "[md5值]"
}
```

查看图片内容

```
请求:
GET /image/show/:image_id

响应:
HTTP/1.1 200 OK
content-type: image/png

[响应 body 中为 图片内容 数据]
```

删除图片

```
请求:
DELETE /image/:image_id

响应:
HTTP/1.1 200 OK
{
    "ok": true
}
```

服务端实现

获取常用操作库

https://gitee.com/HGtz2222/cpp_util

封装数据库操作1

接口设计

db.hpp

```
namespace image_system {
    static MYSQL* MySQLInit() {}
    static void MySQLRelease(MYSQL* mysql) {}

class ImageTable {
    ImageTable(MYSQL* mysql) {  }
    bool Insert(const Json::Value& image);
    bool SelectAll(Json::Value* images);
    bool SelectOne(int32_t image_id, Json::Value* image);
    bool Delete(int image_id);
};
}
```

使用 JSON 作为数据交互格式

json 出自 JavaScript, 是一种非常方便的键值对数据组织格式, 目前被业界广泛使用.

C++ 中可以使用 jsoncpp 这个库来解析和构造 json 数据

```
yum install jsoncpp-devel
```

封装数据库操作2

各个接口代码实现

db.hpp

```
//////////////////////////////////////////////////
// 这个文件相当于 model 层.
// 只进行数据的基本 CURD , 不涉及到更复杂的数据加工
```

```
//////////////////////////////////////////////////

#pragma once
#include <cstdlib>
#include <cstring>
#include <stdint.h>
#include <string>
#include <memory>
#include <mysql/mysql.h>
#include <jsoncpp/json/json.h>

namespace image_system {

static MYSQL* MySQLInit() {
  MYSQL* connect_fd = mysql_init(NULL);
  if (mysql_real_connect(connect_fd, "127.0.0.1", "root", "",
                                                "image_system", 3306, NULL, 0) == NULL) {
    printf("连接失败! %s\n", mysql_error(connect_fd));
    return NULL;
  }
  mysql_set_character_set(connect_fd, "utf8");
  return connect_fd;
}

static void MySQLRelease(MYSQL* mysql) {
  mysql_close(mysql);
}

class ImageTable {
public:
  ImageTable(MYSQL* mysql) : mysql_(mysql) { }

  bool Insert(const Json::Value& image) {
    char sql[4096] = {0};
    sprintf(sql, "insert into image_table values(null, '%s', %d, '%s','%s', '%s', '%s')",
        image["name"].asCString(), image["size"].asInt(), image["upload_time"].asCString(),
        image["md5"].asCString(), image["content_type"].asCString(),
image["path"].asCString());
    int ret = mysql_query(mysql_, sql);
    if (ret != 0) {
      printf("执行 sql 失败! sql=%s, %s\n", sql, mysql_error(mysql_));
      return false;
    }
    return true;
  }

  bool SelectAll(Json::Value* images) {
    char sql[1024 * 4] = {0};
    // 可以根据 tag_id 来筛选结果
    sprintf(sql, "select * from image_table");
    int ret = mysql_query(mysql_, sql);
    if (ret != 0) {
      printf("执行 sql 失败! %s\n", mysql_error(mysql_));
```

```
    return false;
  }
  MYSQL_RES* result = mysql_store_result(mysql_);
  if (result == NULL) {
    printf("获取结果失败! %s\n", mysql_error(mysql_));
    return false;
  }
  int rows = mysql_num_rows(result);
  for (int i = 0; i < rows; ++i) {
    MYSQL_ROW row = mysql_fetch_row(result);
    Json::Value image;
    image["image_id"] = atoi(row[0]);
    image["image_name"] = row[1];
    image["size"] = atoi(row[2]);
    image["upload_time"] = row[3];
    image["md5"] = row[4];
    image["content_type"] = row[5];
    image["path"] = row[6];
    images->append(image);
  }
  return true;
}

bool SelectOne(int32_t image_id, Json::Value* image) {
  char sql[1024 * 4] = {0};
  sprintf(sql, "select * from image_table where image_id = %d", image_id);
  int ret = mysql_query(mysql_, sql);
  if (ret != 0) {
    printf("执行 sql 失败! %s\n", mysql_error(mysql_));
    return false;
  }
  MYSQL_RES* result = mysql_store_result(mysql_);
  if (result == NULL) {
    printf("获取结果失败! %s\n", mysql_error(mysql_));
    return false;
  }
  int rows = mysql_num_rows(result);
  if (rows != 1) {
    printf("查找结果不为 1 条. rows = %d!\n", rows);
    return false;
  }
  MYSQL_ROW row = mysql_fetch_row(result);
  (*image)["image_id"] = atoi(row[0]);
  (*image)["name"] = row[1];
  (*image)["size"] = atoi(row[2]);
  (*image)["upload_time"] = row[3];
  (*image)["md5"] = row[4];
  (*image)["content_type"] = row[5];
  (*image)["path"] = row[6];
  return true;
}

bool Delete(int image_id) {
```

```
    char sql[1024 * 4] = {0};
    sprintf(sql, "delete from image_table where image_id=%d", image_id);
    int ret = mysql_query(mysql_, sql);
    if (ret != 0) {
      printf("执行 sql 失败! sql=%s, %s\n", sql, mysql_error(mysql_));
      return false;
    }
    return true;
  }
private:
  MYSQL* mysql_;
};
}  // end blog_system
```

测试数据库操作

```
// 实现一个数据库接口测试程序，用来验证前面的数据库接口是否正确
#include <iostream>
#include "db.hpp"
using namespace image_system;

void TestImageTable() {
  bool ret = false;
  // 更友好的格式化显示 Json
  Json::StyledWriter writer;
  MYSQL* mysql = MySQLInit();

  Json::Value image;
  image["name"] = "滑稽.jpg";
  image["size"] = 16 * 1024;
  image["upload_time"] = "2019/01/01";
  image["md5"] = "987654321";
  image["content_type"] = "image/jpg";
  image["path"] = "./滑稽.jpg";

  std::cout << "==============测试插入====================" << std::endl;
  ImageTable image_table(mysql);
  ret = image_table.Insert(image);
  std::cout << "Insert: " << ret << std::endl;

  std::cout << "==============测试查找====================" << std::endl;
  Json::Value images;
  ret = image_table.SelectAll(&images);
  std::cout << "SelectAll: " << ret << std::endl
            << writer.write(images) << std::endl;

  Json::Value image_out;
  ret = image_table.SelectOne(1, &image_out);
  std::cout << "SelectOne: " << ret << std::endl
            << writer.write(image_out) << std::endl;

  std::cout << "==============测试删除====================" << std::endl;
```

```
  int image_id = 2;
  ret = image_table.Delete(image_id);
  std::cout << "Delete: " << ret << std::endl;

  MySQLRelease(mysql);
}

int main() {
  TestImageTable();
  return 0;
}
```

Makefile

```
FLAGS=-std=c++11 -L/usr/lib64/mysql -lmysqlclient -ljsoncpp -lpthread -g

.PHONY:all
all:db_test

db_test:db_test.cc db.hpp
    g++ db_test.cc -o db_test $(FLAGS)

.PHONY:clean
clean:
    rm db_test
```

服务器基本框架

使用 cpp-httplib

```
#include "httplib.h"

int main() {
  using namespace httplib;
  Server server;
  server.Get("/", [](const Request& req, Response& resp) {
    (void)req;
    resp.set_content("<html>hello</html>", "text/html");
  });
  server.set_base_dir("./wwwroot");
  server.listen("0.0.0.0", 9094);
  return 0;
}
```

编译选项

```
g++ main.cc -lpthread -std=c++11
```

服务器基本框架2

```
//////////////////////////////////////////////////////
// 构建 HTTP 服务器提供约定的 API 接口
//////////////////////////////////////////////////////

#include <signal.h>
#include <jsoncpp/json/json.h>
#include "util.hpp"
#include "db.hpp"
#include "httplib.h"
#include <openssl/md5.h>

std::string StringMD5(const std::string& str);

const std::string base_path = "./image_data/";

MYSQL* mysql = NULL;

int main() {
  using namespace httplib;
  using namespace image_system;
  Server server;

  // 1. 数据库客户端初始化和释放
  mysql = MySQLInit();
  signal(SIGINT, [](int) { MySQLRelease(mysql); exit(0);});
  ImageTable image_table(mysql);

  // 2. 先按照 cpp-httplib 的文档演示基本的图片上传处理过程
  server.Post("/image_test", [](const Request& req, Response& resp) {

  });

  // 3. 新增图片.
  server.Post("/image", [&image_table](const Request& req, Response& resp) {

  });

  // 4. 查看所有图片的元信息
  server.Get("/image", [&image_table](const Request& req, Response& resp) {

  });


  // 5. 查看图片元信息
  // raw string(c++ 11), 转义字符不生效. 用来表示正则表达式正好合适
  // 关于正则表达式, 只介绍最基础概念即可. \d+ 表示匹配一个数字
  // http://help.locoy.com/Document/Learn_Regex_For_30_Minutes.htm
  server.Get(R"(/image/(\d+))", [&image_table](const Request& req, Response& resp) {

  });

  // 6. 查看图片内容
  server.Get(R"(/image/show/(\d+))", [&image_table](const Request& req, Response& resp) {
```

```
  });

  // 设置静态文件目录
  server.set_base_dir("./wwwroot");

  server.listen("0.0.0.0", 9094);
  return 0;
}

// 需要包含头文件
// #include <openssl/md5.h>
// Makefile 需要 -lcrypto
std::string StringMD5(const std::string& str) {
  const int MD5LENTH = 16;
  unsigned char MD5result[MD5LENTH];
  // 调用 openssl 的函数计算 md5
  MD5((const unsigned char*)str.c_str(),str.size(),MD5result);
  // 转换成字符串的形式方便存储和观察
  char output[1024] = {0};
  int offset = 0;
  for (int i = 0; i < MD5LENTH; ++i) {
    offset += sprintf(output + offset, "%x", MD5result[i]);
  }
  return std::string(output);
}
```

测试上传图片

代码参考 Github 上的文档示例

```
  server.Post("/image_test", [](const Request& req, Response& resp) {
    auto size = req.files.size();
    bool ret = req.has_file("filename");
    const auto& file = req.get_file_value("filename");
    // file.filename
    // file.content_type
    auto body = req.body.substr(file.offset, file.length);
    LOG(INFO) << "size: " << size << ", ret: " << ret << ", " << file.filename << ", "
              << file.content_type << ", " << file.offset << ", "  << file.length <<
std::endl;
    FileUtil::WriteFile(file.filename, req.body.substr(file.offset, file.length));
    resp.set_content("ok", "text/html");
  });
```

实现一个测试页面 upload.html, 放到 wwwroot 目录中

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
</head>
<body>
```

```
    <form method="POST" enctype="multipart/form-data"
action="http://47.98.116.42:9094/image">
        <table>
            <tr>
                <td>文件上传:</td>
                <td><input type="file" name="filename"/></td>
            </tr>
            <tr>
                <td></td>
                <td><input type="submit" value="上传"/></td>
            </tr>
        </table>
    </form>
</body>
</html>
```

验证上传的图片是否能够成功.

实现完整上传图片接口

```
  // 3. 新增图片.
  server.Post("/image", [&image_table](const Request& req, Response& resp) {
    Json::FastWriter writer;
    Json::Value resp_json;

    LOG(INFO) << "新增图片" << std::endl;
    // 1. 进行参数校验
    bool ret = req.has_file("filename");
    if (!ret) {
      resp_json["ok"] = false;
      resp_json["reason"] = "req has no filename field!";
      resp.status = 400;
      resp.set_content(writer.write(resp_json), "application/json");
      return;
    }
    // 2. 构造 Json 格式数据，并调用数据层接口插入数据
    const auto& file = req.get_file_value("filename");
    const std::string& image_body = req.body.substr(file.offset, file.length);

    Json::Value image;
    image["name"] = file.filename;
    image["size"] = (int)file.length;
    image["upload_time"] = TimeUtil::FormatTime();
    image["md5"] = StringMD5(image_body);
    image["content_type"] = file.content_type;
    // 为了防止重复，用 md5 作为文件名更稳妥
    image["path"] = base_path + file.filename;

    ret = image_table.Insert(image);
    if (!ret) {
      resp_json["ok"] = false;
      resp_json["reason"] = "insert db failed!";
      resp.status = 500;
```

```
    resp.set_content(writer.write(resp_json), "application/json");
    return;
  }

  // 3. 保存文件实体
  FileUtil::WriteFile(image["path"].asString(), image_body);

  // 4. 构造响应
  resp_json["ok"] = true;
  resp.set_content(writer.write(resp_json), "text/html");
});
```

实现查看所有图片元信息接口

```
// 4. 查看所有图片的元信息
server.Get("/image", [&image_table](const Request& req, Response& resp) {
  (void) req;
  Json::Reader reader;
  Json::FastWriter writer;
  Json::Value resp_json;
  LOG(INFO) << "查看所有图片信息: " << std::endl;

  // 1. 调用数据库接口查询数据
  Json::Value images;
  bool ret = image_table.SelectAll(&images);
  if (!ret) {
    resp_json["ok"] = false;
    resp_json["reason"] = "SelectAll failed!\n";
    resp.status = 500;
    resp.set_content(writer.write(resp_json), "application/json");
    return;
  }
  // 2. 构造响应结果
  resp.set_content(writer.write(images), "application/json");
  return;
});
```

实现查看单个图片元信息接口

```
// 5. 查看图片元信息
// raw string(c++ 11), 转义字符不生效. 用来表示正则表达式正好合适
// 关于正则表达式, 只介绍最基础概念即可. \d+ 表示匹配一个数字
// http://help.locoy.com/Document/Learn_Regex_For_30_Minutes.htm
server.Get(R"(/image/(\d+))", [&image_table](const Request& req, Response& resp) {
  Json::Reader reader;
  Json::FastWriter writer;
  Json::Value resp_json;

  // 1. 获取到图片 id
  int image_id = std::stoi(req.matches[1]);
  LOG(INFO) << "查看图片信息: " << image_id << std::endl;
```

```
  // 2. 调用数据库接口查询数据
  Json::Value image;
  bool ret = image_table.SelectOne(image_id, &image);
  if (!ret) {
    resp_json["ok"] = false;
    resp_json["reason"] = "SelectOne failed!\n";
    resp.status = 500;
    resp.set_content(writer.write(resp_json), "application/json");
    return;
  }
  // 3. 构造响应结果
  resp.set_content(writer.write(image), "application/json");
  return;
});
```

实现查看图片内容接口

```
// 6. 查看图片内容
server.Get(R"(/image/show/(\d+))", [&image_table](const Request& req, Response& resp) {
  Json::Reader reader;
  Json::FastWriter writer;
  Json::Value resp_json;

  // 1. 获取到图片 id
  int image_id = std::stoi(req.matches[1]);
  LOG(INFO) << "查看图片信息: " << image_id << std::endl;

  // 2. 调用数据库接口查询数据
  Json::Value image;
  bool ret = image_table.SelectOne(image_id, &image);
  if (!ret) {
    resp_json["ok"] = false;
    resp_json["reason"] = "SelectOne failed!\n";
    resp.status = 500;
    resp.set_content(writer.write(resp_json), "application/json");
    return;
  }
  std::string image_body;
  ret = FileUtil::ReadFile(image["path"].asString(), &image_body);
  if (!ret) {
    resp_json["ok"] = false;
    resp_json["reason"] = "path open failed\n";
    resp.status = 500;
    resp.set_content(writer.write(resp_json), "application/json");
    return;
  }
  // 3. 构造响应结果
  resp.set_content(image_body, image["content_type"].asCString());
  return;
});
```

实现删除图片接口

```
// 7. 删除图片
server.Delete(R"(/image/(\d+))", [&image_table](const Request& req, Response& resp) {
  Json::Value resp_json;
  Json::FastWriter writer;
  // 1. 解析获取 blog_id
  // 使用 matches[1] 就能获取到 blog_id
  // LOG(INFO) << req.matches[0] << "," << req.matches[1] << "\n";
  int image_id = std::stoi(req.matches[1]);
  LOG(INFO) << "删除指定图片: " << image_id << std::endl;
  // 2. 调用数据库接口删除博客
  bool ret = image_table.Delete(image_id);
  if (!ret) {
    resp_json["ok"] = false;
    resp_json["reason"] = "Delete failed!\n";
    resp.status = 500;
    resp.set_content(writer.write(resp_json), "application/json");
    return;
  }
  // 3. 包装正确的响应
  resp_json["ok"] = true;
  resp.set_content(writer.write(resp_json), "application/json");
  return;
});
```

使用 Postman 进行测试

在 Postman 中构造请求, 并验证即可.

后续扩展点

1. 多个小图片拼接成一个大文件, 提高存储效率
2. 支持图片处理功能(例如返回指定大小的图片)
3. 防盗链 https://www.cnblogs.com/wangyongsong/p/8204698.html
4. 引用计数方式保存多个相同的图片

附录

使用 devtool 升级 g++ 到 7.3 版本

以下命令在 root 下使用

```
yum install centos-release-scl -y
yum install devtoolset-7 -y
```

以下命令添加在 bashrc 中

```
source /opt/rh/devtoolset-7/enable
```

使用 g++ -v 验证 g++ 版本

安装 httplib

直接从 github 上下载, 拷贝 httplib.h 到需要的目录即可.

https://github.com/yhirose/cpp-httplib

安装 Postman

官方下载地址

https://dl.pstmn.io/download/latest/win64

注意!!! 网上大多数关于 POSTMAN 的安装教程都过时了. 当前不用翻墙就能直接在官网上下载安装包.