三满百步网设框架（Bible Web Design Frame）

cai_yankun

目　录

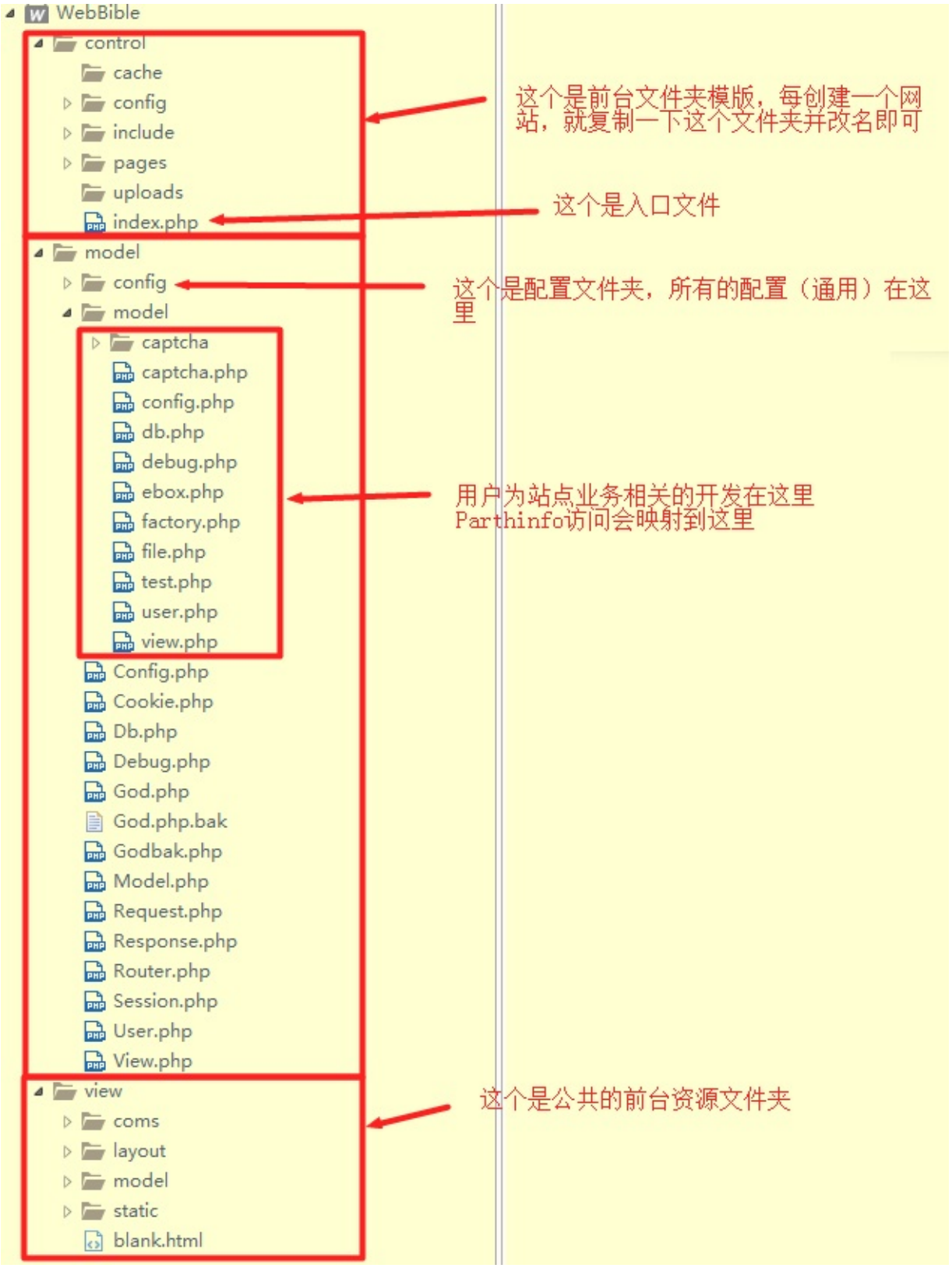
框架简介

框架的作用和特点是：

1. 轻量级，惰性加载；
2. 不仅包括后端（LAMP），也包括了前端的设计（JS，Ajax，工厂机制），以及前后端的交互机制，是一套前后端拉通的快速开发网站的思路和工具；
3. 彻底实现了前后端分离，降低了服务器的运行负荷，简化了后台PHP的复杂度，并且实现了数据访问的效率以及后期维护的便捷
4. 支持一套框架同时驱动多个独立的站点，不同的站点之间实现灵活的共享、独立；
5. 支持框架项目与普通项目的无缝结合，能将框架快速的集成到一个非框架类项目中进行快速开发；也能将一个非框架类的项目快速适配到框架项目中继续进行开发；
6. 重定义了MVC架构，将控制功能的实现前移，移至js，目的是简单，高效；

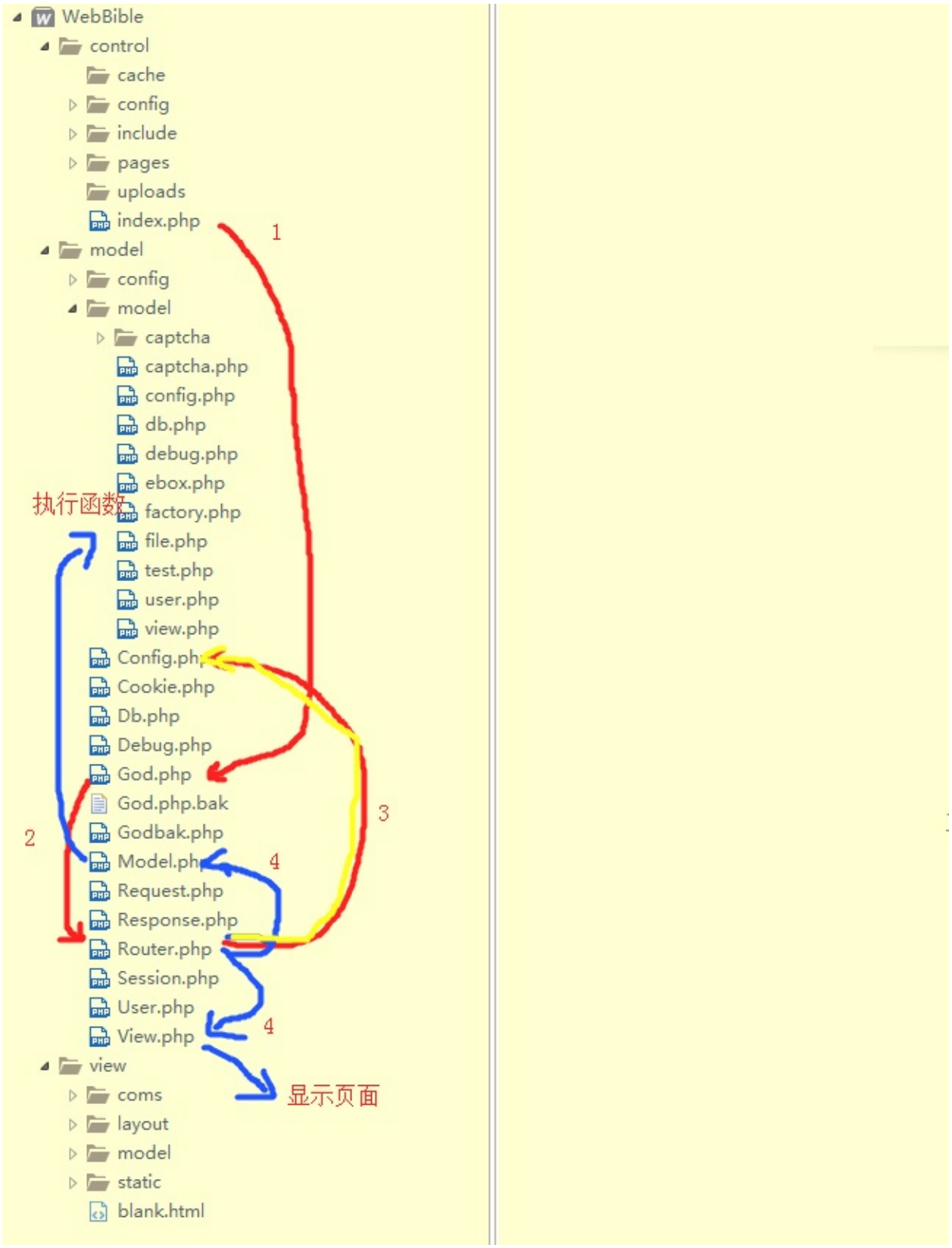
- Model类封装了框架固有的一些类（配置，SESSON，COOKIE，DB），实现了Pathinfo到PHP函数映射，权限处理等问题；
- Controller为逻辑业务的实现，不在PHP中实现，前移到前端JS中实现；简化后台PHP为数据流通管道功能；
- View类封装了Pathinfo到前台展示页面的映射，以及前台代码相关的的功能；

框架原理



- Model类封装了框架固有的一些类（配置，SESSON，COOKIE，DB），实现了Pathinfo到PHP函数映射，权限处理等问题；
- Controller为逻辑业务的实现，不在PHP中实现，前移到前端JS中实现；简化后台PHP为数据流通管道功能；

- View类封装了Pathinfo到前台展示页面的映射，以及前台代码相关的的功能；



运行流程：地址栏输入成功进入到入口函数以后，首先加载

`God.php /*启动惰性加载功能，然后直接调用router进行分发*/`

Router.php

框架的安装

安装要求

PHP版本>5

安装步骤

解压到服务1器文件夹

https://caiyankun@github.com/caiyankun/webbible.git

配置单站点apache:

`/etc/httpd/conf/httpd.conf`

`DocumentRoot "/var/webbible/control"`

配置多站点apache：

`/etc/httpd/conf/httpd.conf`

```
<VirtualHost *:80>
    ServerAdmin cai_yankun@qq.com
    DocumentRoot /var/webbible/site1.com
    ServerName site1.com
    ErrorLog logs/more.sanmantech.com-error_log
    CustomLog logs/more.sanmantech.com-access_log common
</VirtualHost>
<VirtualHost *:80>
    ServerAdmin cai_yankun@qq.com
    DocumentRoot /var/webbible/site2.com
    ServerName site.com
    ErrorLog logs/more.sanmantech.com-error_log
    CustomLog logs/more.sanmantech.com-access_log common
</VirtualHost>
```

开通地址栏重写：

`/etc/httpd/conf/httpd.conf`

```
LoadModule rewrite_module modules/mod_rewrite.so
```

配置Apache用户/PHP脚本的可写权限：

在webbible目录下

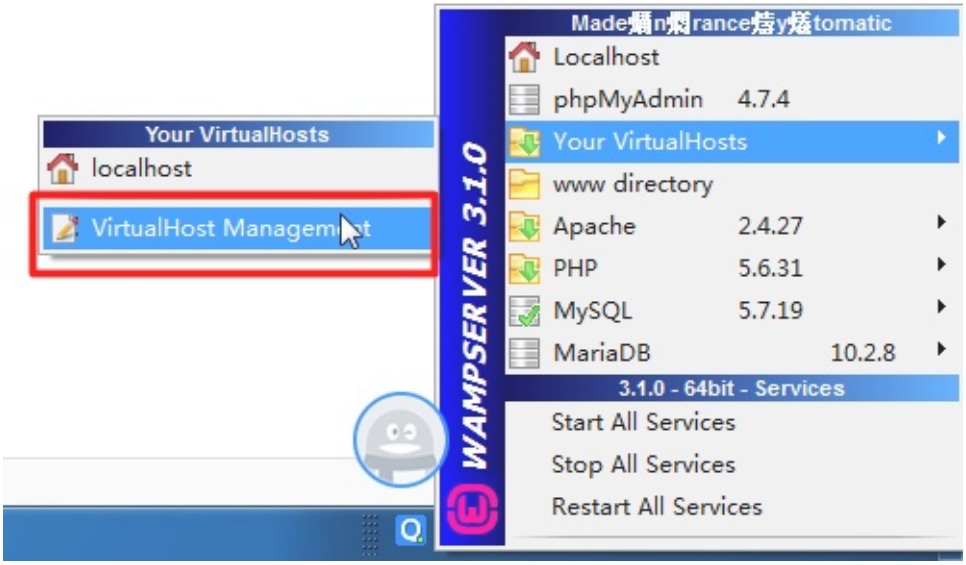
```
chown apache:apache -R
ls-l
```

安装测试：

浏览器输入`www.site1.com`

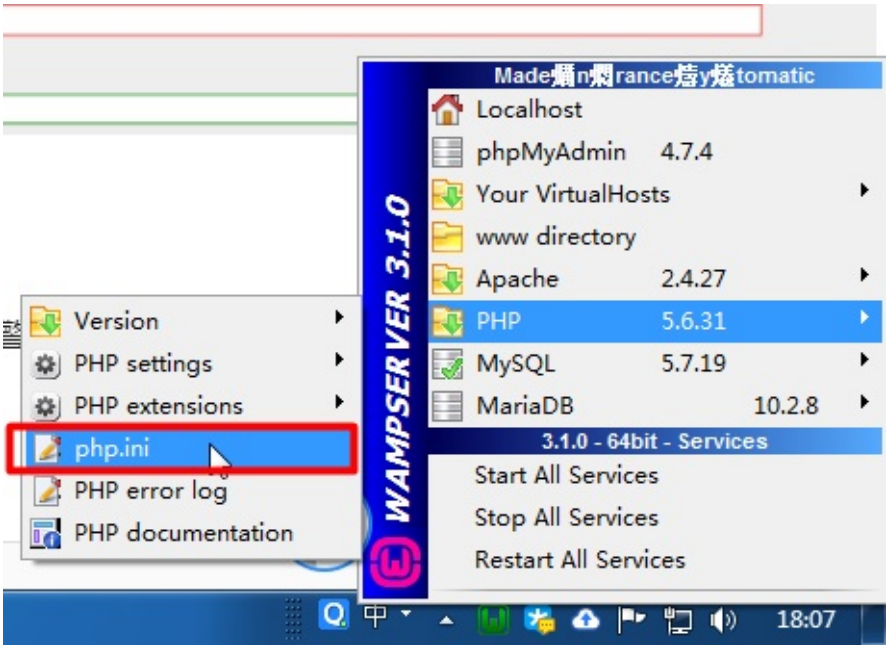
本地PHP服务器环境快速搭建

- 下载WampServer64，安装并运行；
  如果出现运行时错误（缺少.dll）就先卸载，并安装相应的VC++ redistribute
  120 对应的是2013版本
  140对应的是2015版本
  100对应的是2012版本
  每个版本都要装
- 创建虚拟主机：



填写需要使用的域名，以及项目所在的路径，路径中好像

- 修改PHP报错级别
  新版本的PHP报错级别很低，可以设置忽略告警信息：



搜索 error_reporting

设置error_reporting=E_ERROR | E_PARSE

```
如果参数 level 未指定，当前报错级别将被返回。下面几项是 level 可能的值：
    值      常量                    描述
    1       E_ERROR                 致命的运行错误。错误无法恢复，暂停执行脚本。
    2       E_WARNING               运行时警告(非致命性错误)。非致命的运行错误，脚本执行不会停止
。
    4       E_PARSE                 编译时解析错误。解析错误只由分析器产生。
    8       E_NOTICE                运行时提醒(这些经常是你代码中的bug引起的，也可能是有意的行为
造成的。)
    16      E_CORE_ERROR            PHP启动时初始化过程中的致命错误。
    32      E_CORE_WARNING          PHP启动时初始化过程中的警告(非致命性错)。
    64      E_COMPILE_ERROR         编译时致命性错。这就像由Zend脚本引擎生成了一个E_ERROR。
    128     E_COMPILE_WARNING       编译时警告(非致命性错)。这就像由Zend脚本引擎生成了一个E_WAR
NING警告。
    256     E_USER_ERROR            用户自定义的错误消息。这就像由使用PHP函数trigger_error（程
序员设置E_ERROR）
    512     E_USER_WARNING          用户自定义的警告消息。这就像由使用PHP函数trigger_error（程
序员设定的一个E_WARNING警告）
    1024    E_USER_NOTICE           用户自定义的提醒消息。这就像一个由使用PHP函数trigger_error
（程序员一个E_NOTICE集）
    2048    E_STRICT                编码标准化警告。允许PHP建议如何修改代码以确保最佳的互操作性向
前兼容性。
    4096    E_RECOVERABLE_ERROR     开捕致命错误。这就像一个E_ERROR，但可以通过用户定义的处理捕
获（又见set_error_handler（））
    8191    E_ALL
```

- 初始化Mysql密码：

  刚安装时PHPmyadmin可用root+空密码即可登录

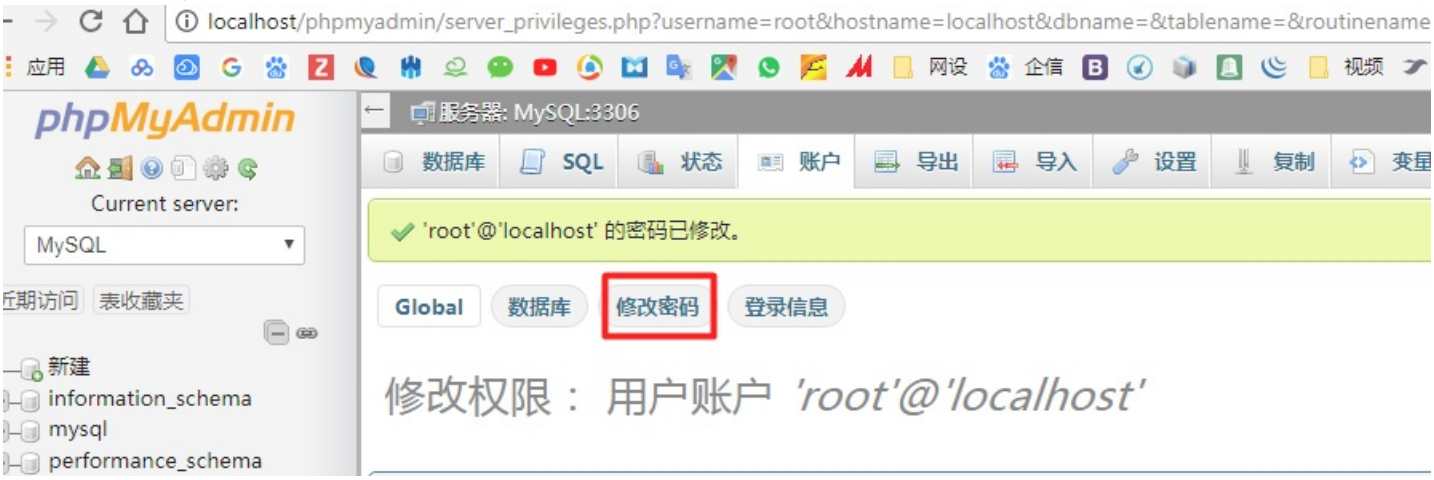
  如果要创建密码，请先登录console

```
USE mysql;
```

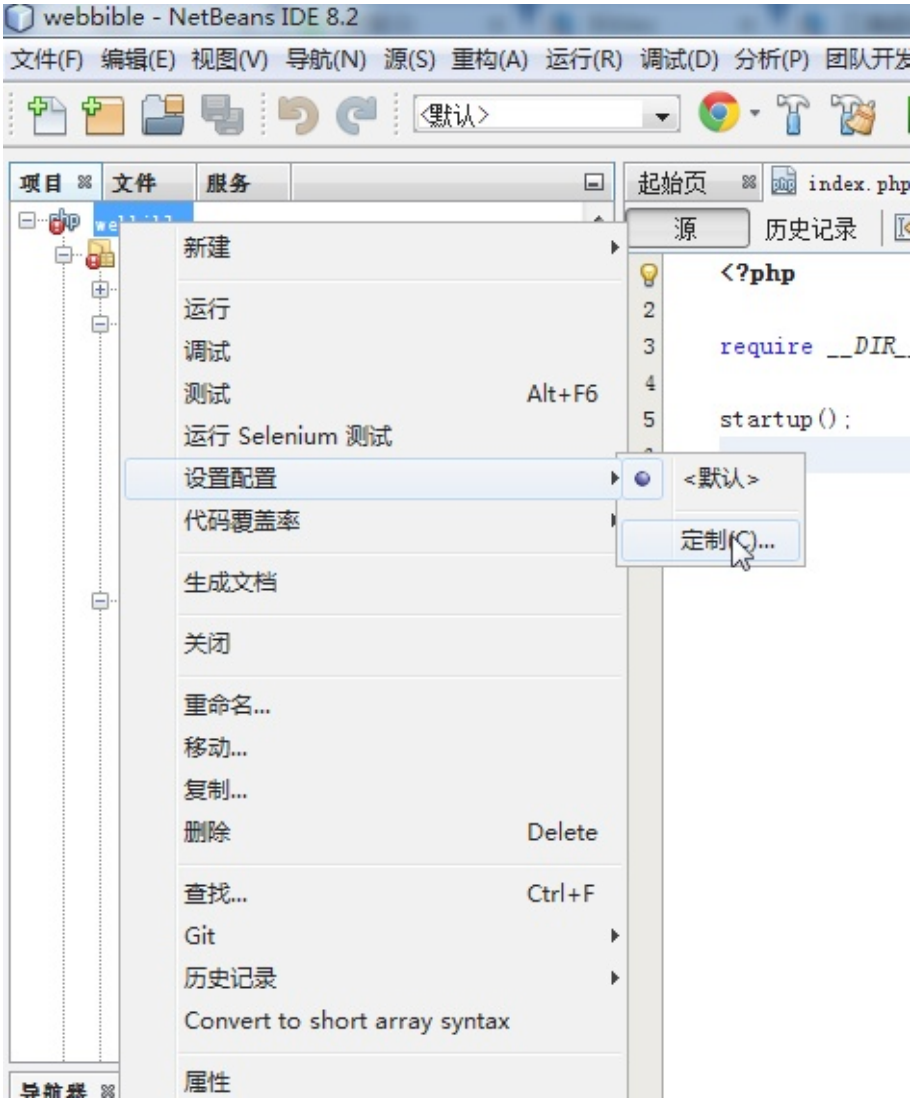
```
update user set password=“1234Abcd!” where user=’root’;
flush privileges;
quit;
```

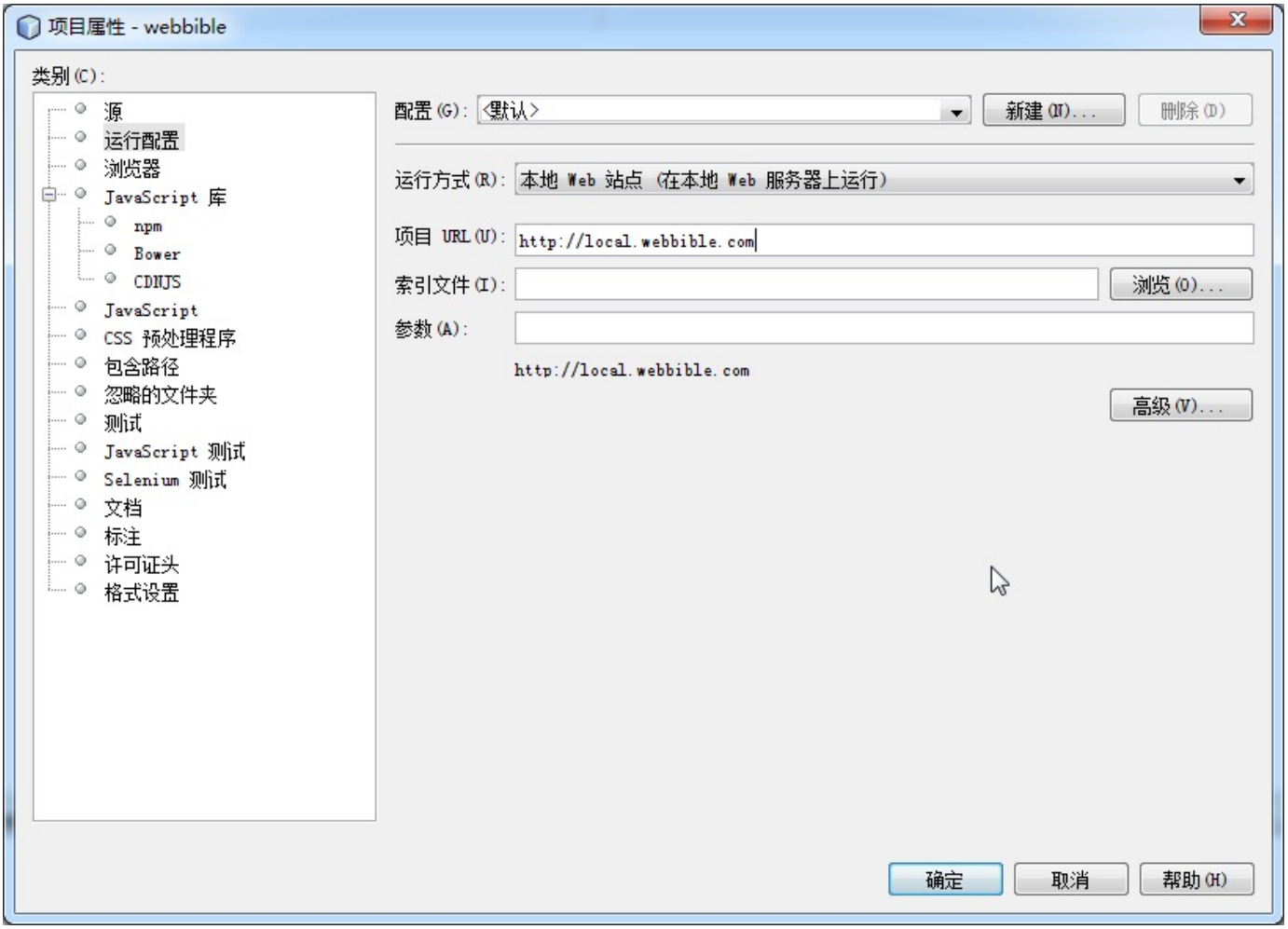
重启mysql服务

或者在phpmyadmin中直接更改



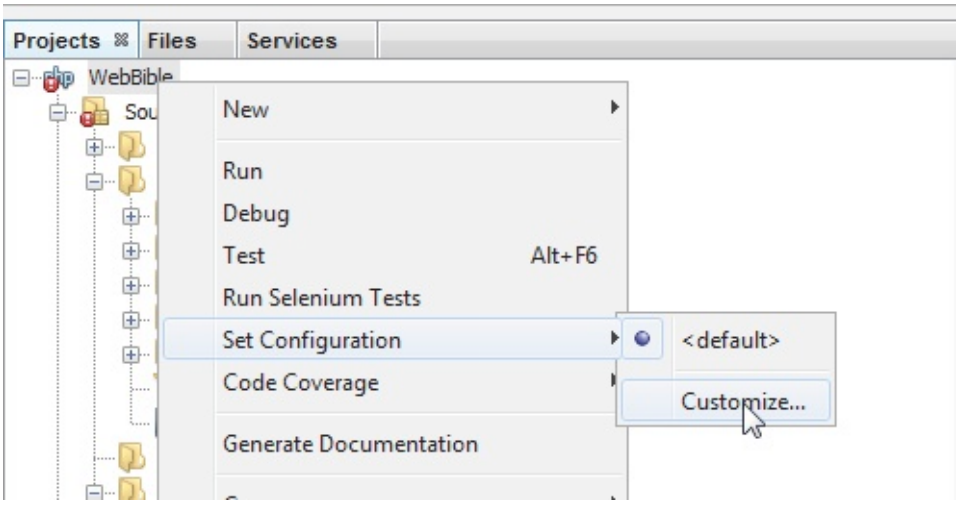
- 重启所有服务；
- 设置Netbeans 调试服务器网址；

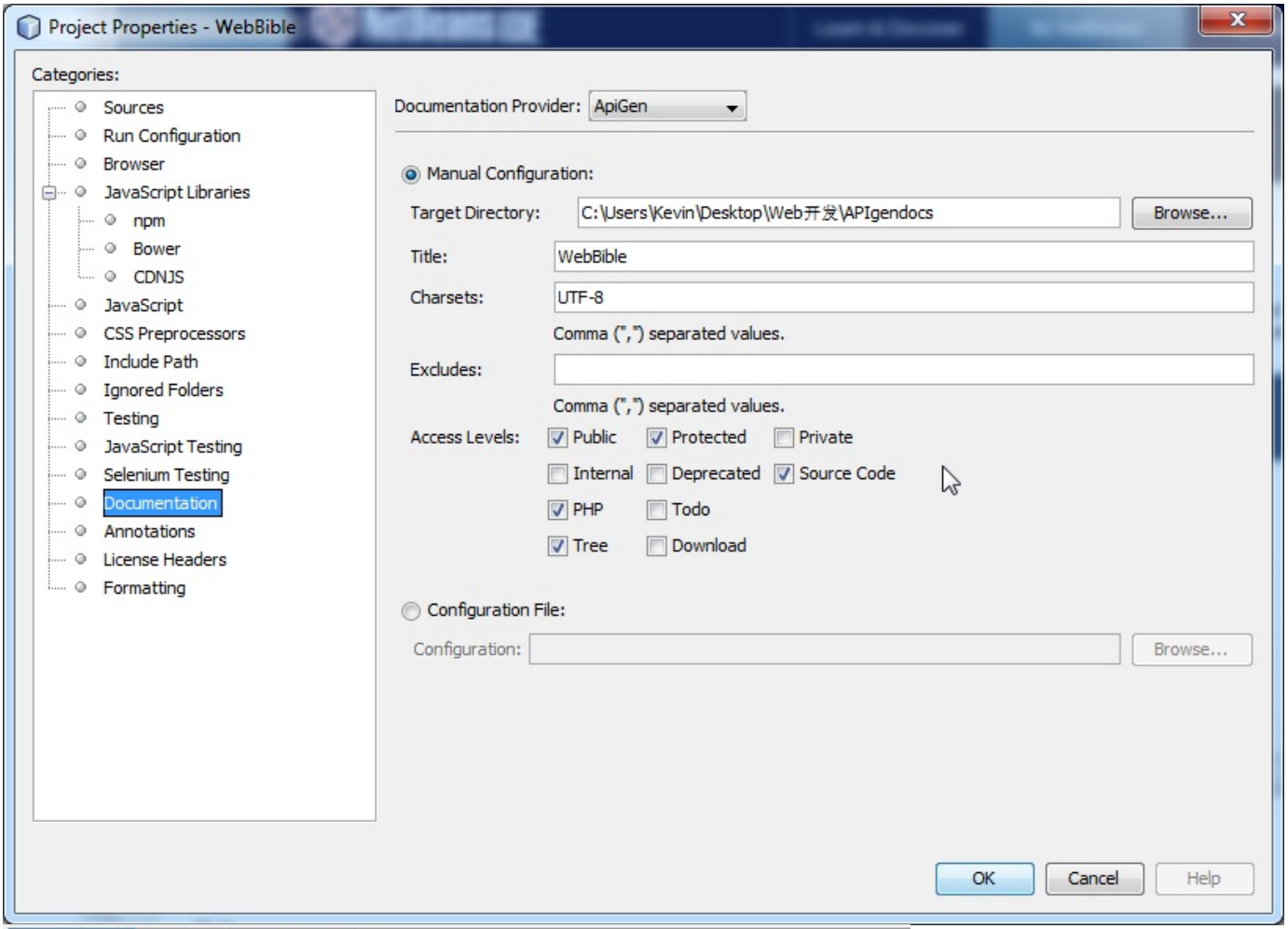


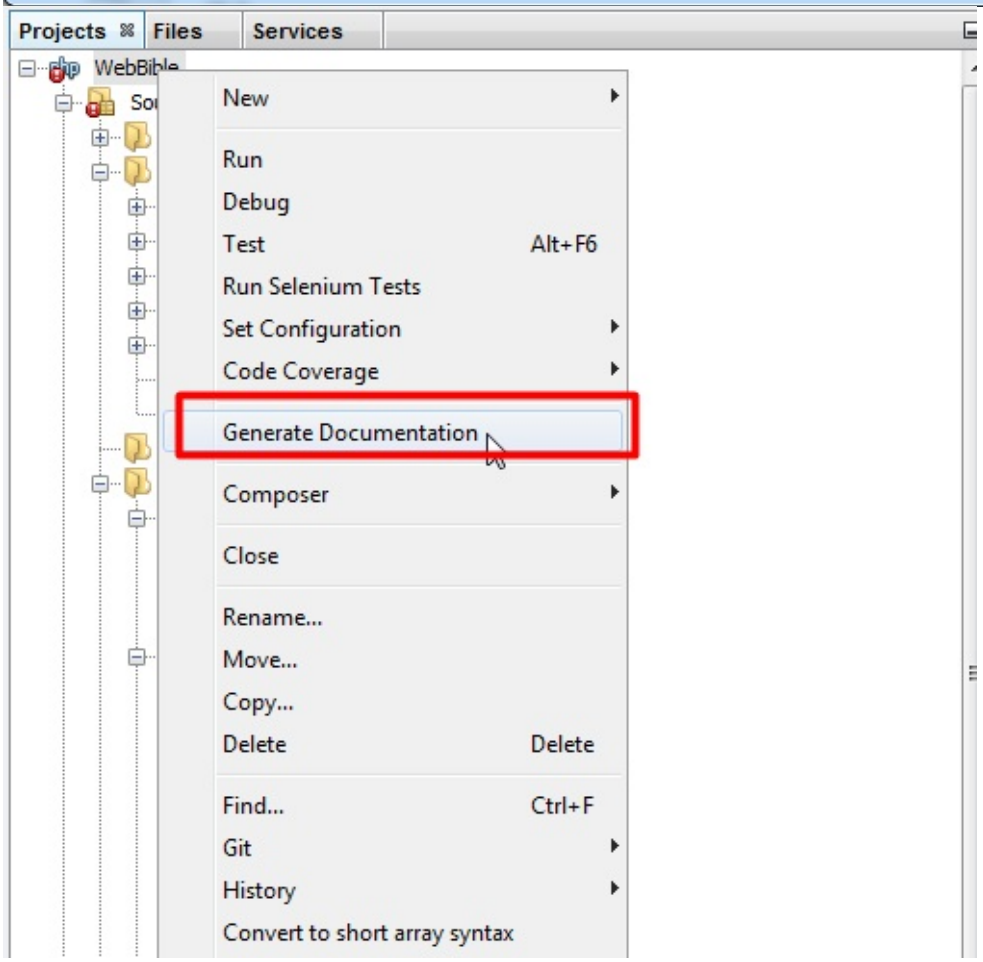

项目属性 - webbible
类别(C):
源
运行配置
浏览器
JavaScript 库
npm
Bower
CDNJS
JavaScript
CSS 预处理程序
包含路径
忽略的文件夹
测试
JavaScript 测试
Selenium 测试
文档
标注
许可证头
格式设置
配置(G): <默认>
新建(N)...
删除(D)
运行方式(R): 本地 Web 站点 (在本地 Web 服务器上运行)
项目 URL(U): http://local.webbible.com
索引文件(I):
浏览(O)...
参数(A):
http://local.webbible.com
高级(V)...
确定
取消
帮助(H)

Netbeans开发工具配置

- 安装Netbeans
- 安装有用的插件
  php
  composer
  git
- 安装文档生成工具APIGen
  下载
  http://apigen.org/apigen.phar
  把这个文件拷贝到 C:\Windows\System32
  然后在相同的目录下创建apigen.bat 内容为：
  @C:\wamp64\bin\php\php5.6.31\php.exe "%~dp0apigen.phar" %*
  在cmd中运行apigen，如果能成功显示版本信息，则说明安装成功
  命令行使用方法如下：
  apigen generate --source "D:\web\ruionline" --destination "D:\web\ruionline\doc"
  Netbeans使用如下：

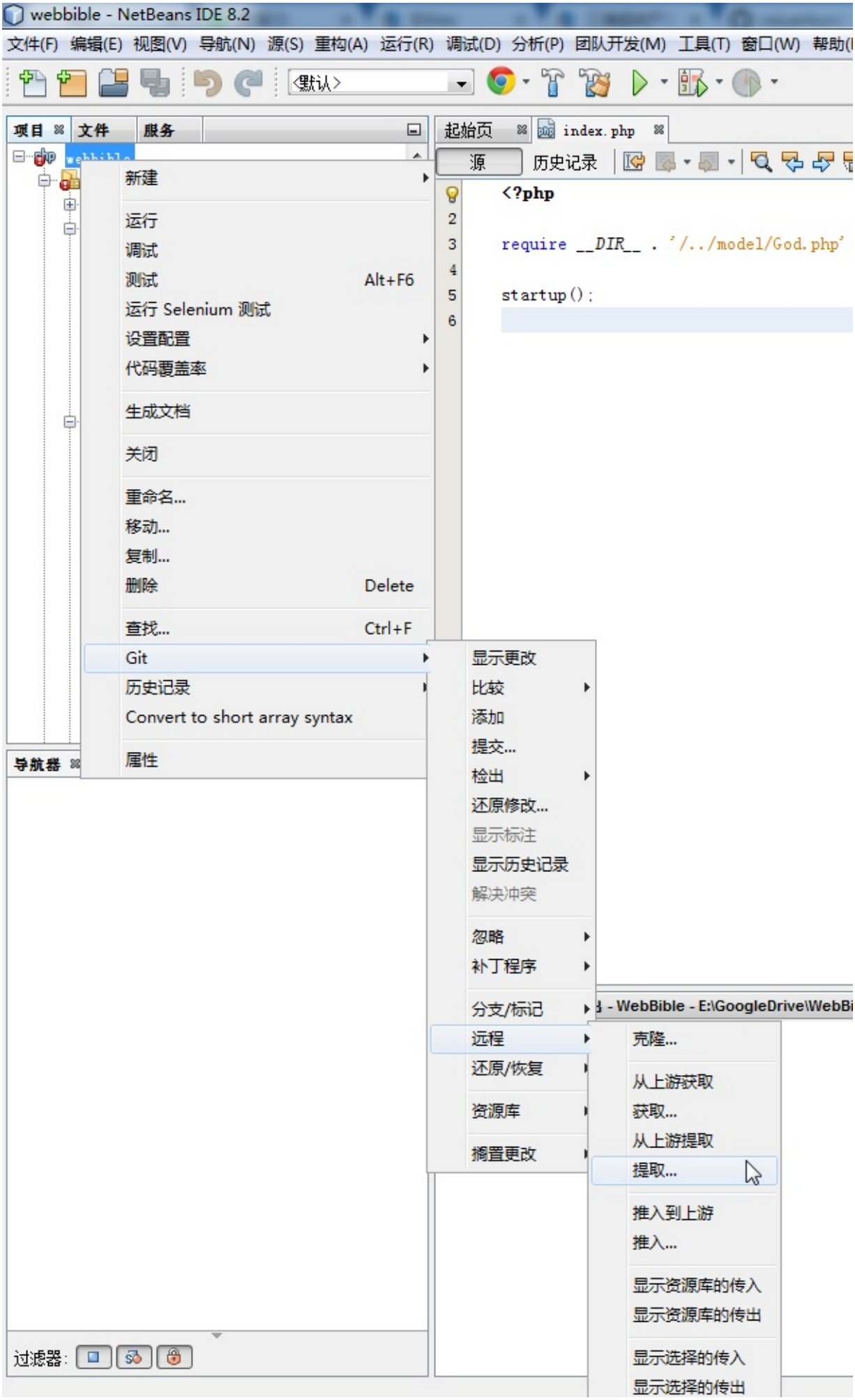






- 在github上创建一个库，记下链接；

- 从Git库检出：Git Pull，填写相应的链接


webbible - NetBeans IDE 8.2
文件(F) 编辑(E) 视图(V) 导航(N) 源(S) 重构(A) 运行(R) 调试(D) 分析(P) 团队开发(M) 工具(T) 窗口(W) 帮助(
<默认>
项目
文件
服务
起始页
index.php
源
历史记录
<?php
require __DIR__ . '/../model/God.php';
startup();
新建
运行
调试
测试 Alt+F6
运行 Selenium 测试
设置配置
代码覆盖率
生成文档
关闭
重命名...
移动...
复制...
删除 Delete
查找... Ctrl+F
Git
历史记录
Convert to short array syntax
属性
显示更改
比较
添加
提交...
检出
还原修改...
显示标注
显示历史记录
解决冲突
忽略
补丁程序
分支/标记
远程
还原/恢复
资源库
搁置更改
克隆...
从上游获取
获取...
从上游提取
提取...
推入到上游
推入...
显示资源库的传入
显示资源库的传出
显示选择的传入
显示选择的传出
导航器
过滤器:

pull完成后如果不是一个工程文件会提示你是否要新建一个工程

- 如果你是本地创建的文件想push到github，那么先要pull（合并）
  然后在push

框架的使用

惰性加载机制

- 惰性加载机制是在god.php实现的
- 惰性加载能成功的前提是：

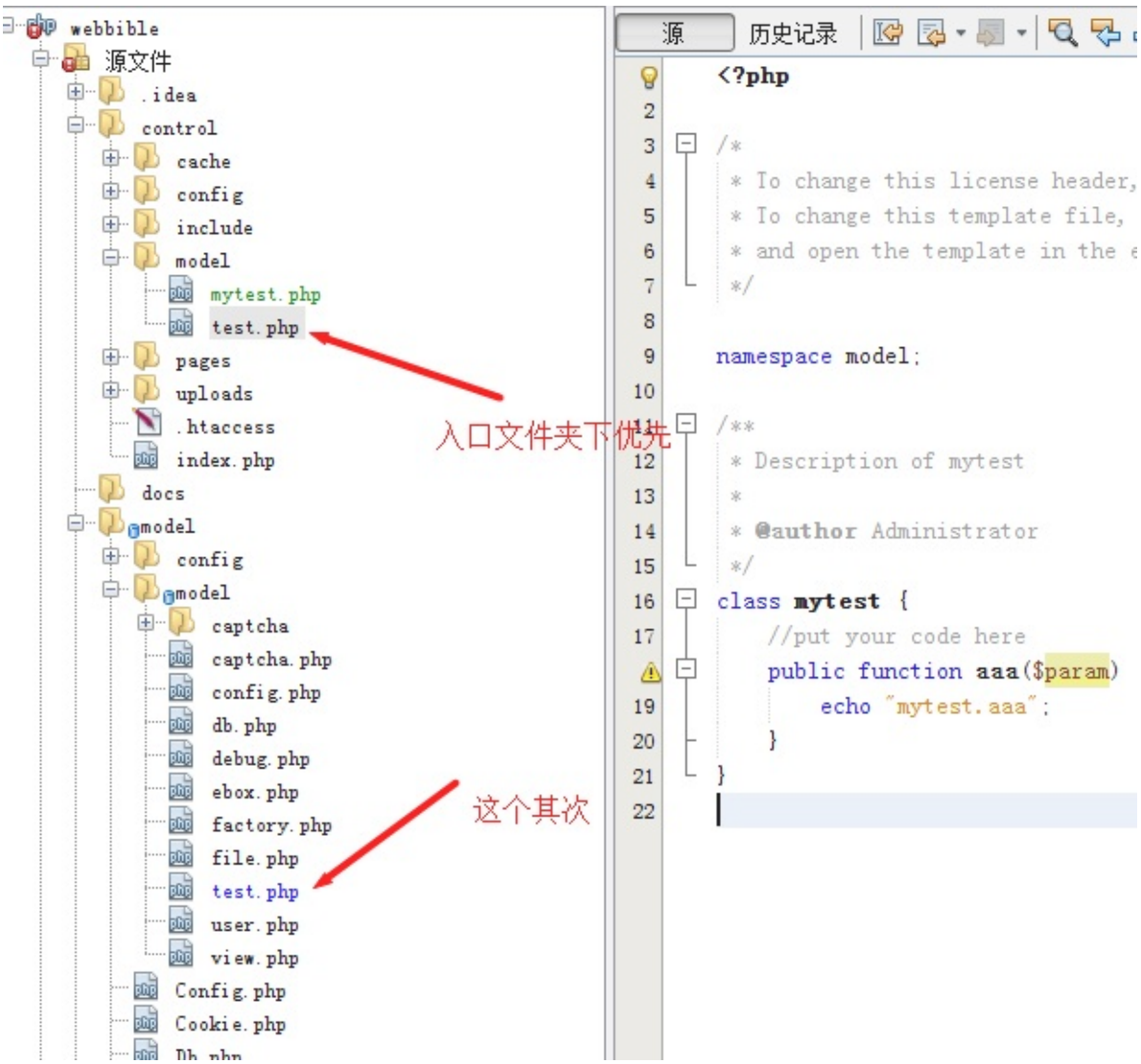
1. 编写的php类要与其文件名严格相同，
2. 而这个文件名相对于框架的MODEL文件夹的相对路径与这个类所在的命名空间的路径是严格一致的

- 当程序调用一个未知的函数时，会自动跳转到autoload函数尝试加载，之后再调用一次
  `spl_autoload_register(__NAMESPACE__ .'\Loader::autoload'); // 注册自动加载函`

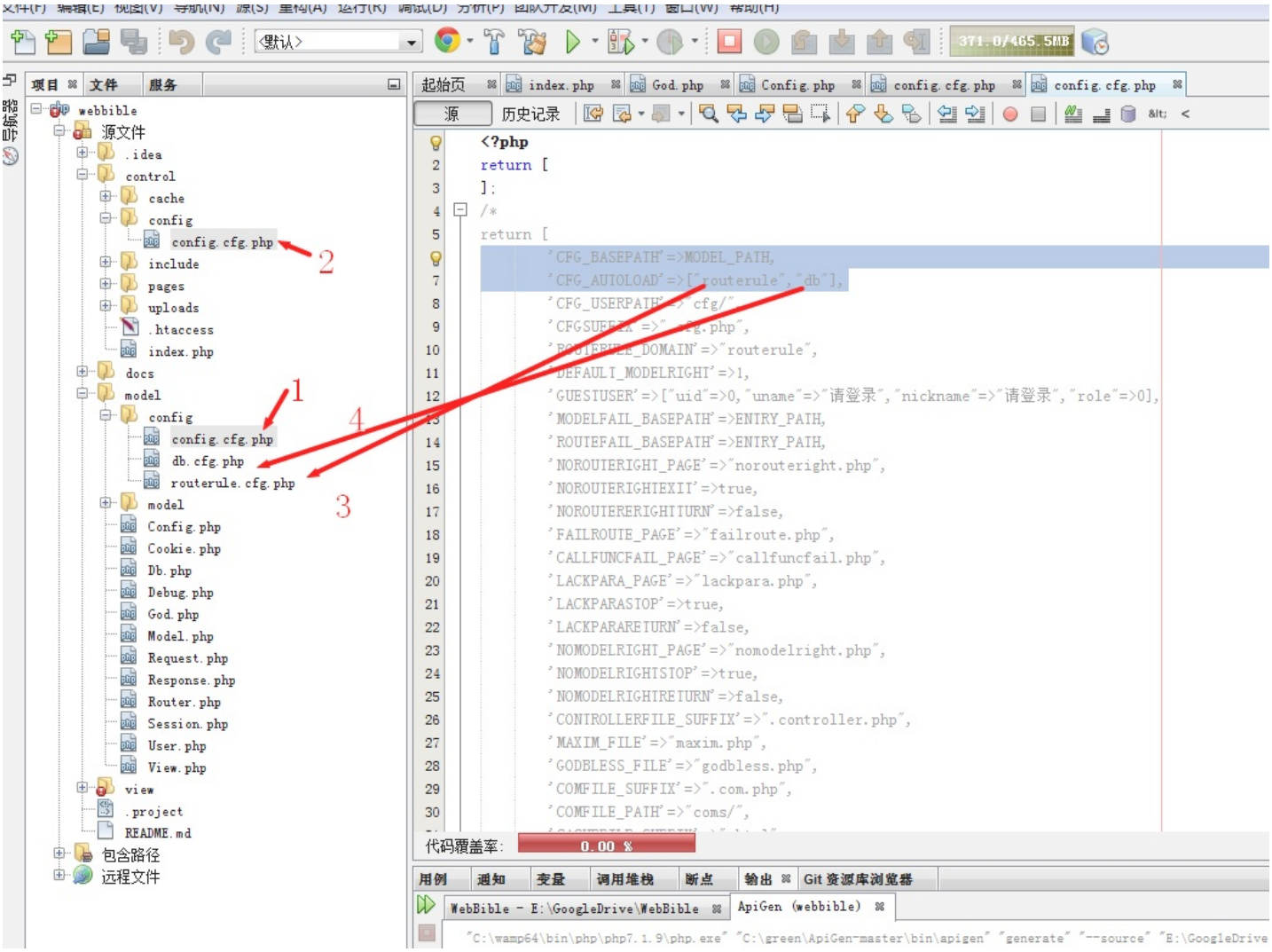
```
//实现类的自动加载功能，通过spl_autoload_register
class Loader
{
        /* 路径映射 */
        public static $namespaceroot = MODEL_PATH;
        public static $loadlog =[];
        /**
        * 自动加载器
        */
        public static function autoload($class)
        {
            //echo "<br>Request class:".$class;
            $file = self::findFile($class);
            //echo "<br>class file should be:".$file ;
            if (file_exists($file)) {
                self::includeFile($file);
                //echo "<br>loaded:".$file ;
            }
        }
        /**
        * 解析文件路径
        */
        private static function findFile($class)
        {
            $rsv=self::$namespaceroot.$class.".php";
            $rsv=strtr($rsv,"\\",DIRECTORY_SEPARATOR);
            return $rsv; // 文件标准路径
        }
        /**
        * 引入文件
        */
        private static function includeFile($file)
        {
            if (is_file($file)) {
                include $file;
                //echo "<br>loaded file:".$file;
            }
        }
```

```
}
```

- model命名空间是特殊的命名空间，当框架检测到你要加载一个model命名空间的类时，会优先判断入口文件的model文件夹下是否有相应的文件，如果有就加载之，如果没有，才加载框架的model文件夹下的相应文件。
  这种实现机制保障了不同的网站之间可以有公共的自定义类，也保障了每个网站可以有自己独特的自定义类



配置文件加载机制



这个类是为了方便的按照一定规则读取配置文件，
如果用户没有进行相应的设置，就返回客户调用时指定的值；
这个文件在惰性加载的时候会自动执行一次Autoload：

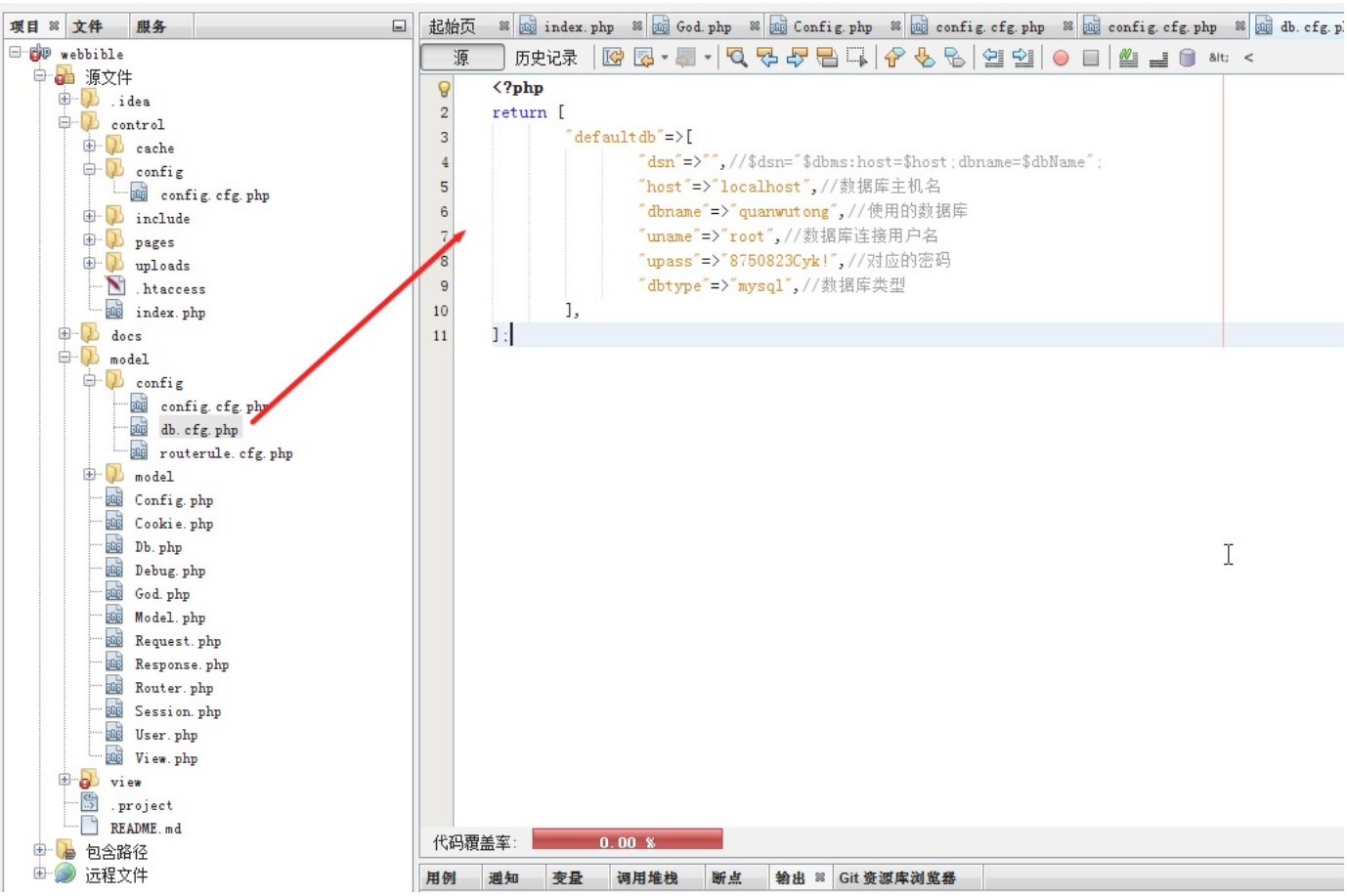
1,首先 Model目录下的主配置文件是必须要加载的；
2,其次 入口文件下的config目录下的主配置文件也是必须要加载的；
3,加载完成后读取到内存中的cfg_userpath,CFG_AUTOLOAD,CFGSUFFIX 这三个变量，然后依次加载 CFG_AUTOLOAD 中定义的配置文件；
4,配置文件是有domain限制的，Model主配置文件和用户主配置文件中的配置所属的domain为空，每个额外的单独的配置文件，其domain为该文件名，如db.cfg.php中的变量，将会存放在domain为db的配置中
5,配置存取时都会强制转换为大写，因此在配置文件中，或者在使用配置时，是不用区分大小写的。

```
    public static function autoload(){
        self::load(MODEL_PATH.MODELCFG_FILE);//加载配置变量
```

```
        $cfg_userpath=MODEL_PATH.self::get("cfg_userpath","","");
        $cfgfiles=self::get("CFG_AUTOLOAD","",[]);
        $cfgsuffix=self::get("CFGSUFFIX","",DEFAULT_CFGSUFFIX);
        foreach ($cfgfiles as $cfgfile){
            self::load($cfg_userpath.$cfgfile.$cfgsuffix,$cfgfile);
        }
    }
```

数据库配置机制

- 数据库的配置：



这里可以配置多种数据库配置，在执行的时候可以进行选择指定，如果不指定，会使用默认的数据库；如：

```
Db::query("select * from user;");//会使用默认数据库
Db::bycfg("anotherdb")::query("select * from user;");//会使用anotherdb的连接
Db::setup([
        "dsn"=>"",//$dsn="$dbms:host=$host;dbname=$dbName";
        "host"=>"localhost",//数据库主机名
        "dbname"=>"test",//使用的数据库
        "uname"=>"root",//数据库连接用户名
        "upass"=>"",//对应的密码
        "dbtype"=>"mysql",//数据库类型
])::query("select * from user;");//会使用anotherdb的连接
```

- 数据库的连接：
  数据库会自动连接，直接执行配置或者查询语句就可以，连接机制如下：

Query -》getInstance-》Init

init的时候会自动加载默认配置，如果调用setup或者bycfg，则会覆盖默认配置

```
  public static function query($str,$assoc=0){
       $conn=self::getInstance();
       if(is_null($conn)){
           return false;
       } else {
           self::clear();
           try{
               $pdostate=$conn->query($str,$assoc?PDO::FETCH_ASSOC:PDO::FETCH_NUM);
               self::$data=[];
               self::$data[0]=$pdostate->fetchAll();
           } catch (PDOException $e) {
               self::$error=$e->getCode();
               self::$info=$e->getMessage();
               return false;
           }
           return true;
       }
   }

   public static function getInstance(  ) {
    self::init();
       if(!self::$objInstance){
           if(empty(self::$config["dsn"])){
               self::$config["dsn"]=self::$config["dbtype"].":host=".self::$config["
host"].";dbname=".self::$config["dbname"];
           }
           $dsn=self::$config["dsn"];
           $uname=self::$config["uname"];
           $upass=self::$config["upass"];
           //echo json_encode(self::$config);
           //exit(0);
           try {
             self::$objInstance = new PDO($dsn, $uname, $upass);
             self::$objInstance->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPT
ION);
             self::$objInstance->query("SET NAMES utf8");
           } catch (PDOException $e) {
             self::$error=$e->getCode();
             self::$info=$e->getMessage();
             self::$objInstance=null;
           }
       }
       return self::$objInstance;
   }


   private static function init(){
    if(!self::$inited){
           self::$config=array_merge(self::$config,Config::get("defaultdb","db",[]))
;
           self::$inited=true;
    }
```

```
     return true;
    }
```

- 数据库的查询：

```
    public static function query($str,$assoc=0)；//所有的查询都通过这个函数最终执行
    public static function callproc ($procname,$params)；//调用数据库的存储过程.举例：cal
lproc("user.logincheck",[uname=>"cyk",_upass=>"123",_result=>0]), 函数会将执行拆分成三个
语句来执行：select @uname:="cyk",@_upass=123;
    callproc..
    获取返回值，装在Db::data中；
    select @_pass;
    获取返回值，装在Db::outvars[]中；
    public static function simplecall($procname,$params)；//调用简易存储过程时，参数是普通
数组即可，一样可以带有输出参数，输出参数必须以_开头.举例：simplecall("user.logincheck",["cyk",
123,_result]).该函数不能有输入输出参数
```

- 数据库的查询结果：
  数据库执行成功与否直接由函数的返回值为true或者false决定；
  当判断为执行成功的时候，结果集会存放在

使用query返回的至多是二维数组；存放在Db:;data[0]中；

使用callproc返回的也是三维数组（不带关联）；数据的标题存放在Db::$datatitle中；

如果存储过程带有输出参数，输出参数会放在self::$outvars 中；
直接使用Db::$outvars[参数名]即可访问

- 错误处理机制：

在编写存储过程的时候，如果判断业务执行不成功，存储过程可通过抛出异常的方式告知PHP

```
BEGIN
   DECLARE matchnum INT default 0;
   call username_check(p_uname,matchnum);
   if matchnum <1 then
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No such user!';
   else
       select count(*) into matchnum from user_basic where uname=p_uname and upass=p
_upass;
       if matchnum>0 then
         select uid,uname,ulevel,uoption from user_basic where uname=p_uname and upa
ss=p_upass;
       else
         SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = 'Password incorrect!';
       end if;
   end if;
END
```

PHP在收到异常的时候，首先query函数返回false，然后把错误码和错误信息记录下来：Db::$error和Db::$info中

```
    public static function query($str,$assoc=0){
        $conn=self::getInstance();
        if(is_null($conn)){
            return false;
        } else {
            self::clear();
            try{
                $pdostate=$conn->query($str,$assoc?PDO::FETCH_ASSOC:PDO::FETCH_NUM);
                self::$data=[];
                self::$data[0]=$pdostate->fetchAll();
            } catch (PDOException $e) {
                self::$error=$e->getCode();
                self::$info=$e->getMessage();
                return false;
            }
            return true;
        }
    }
```

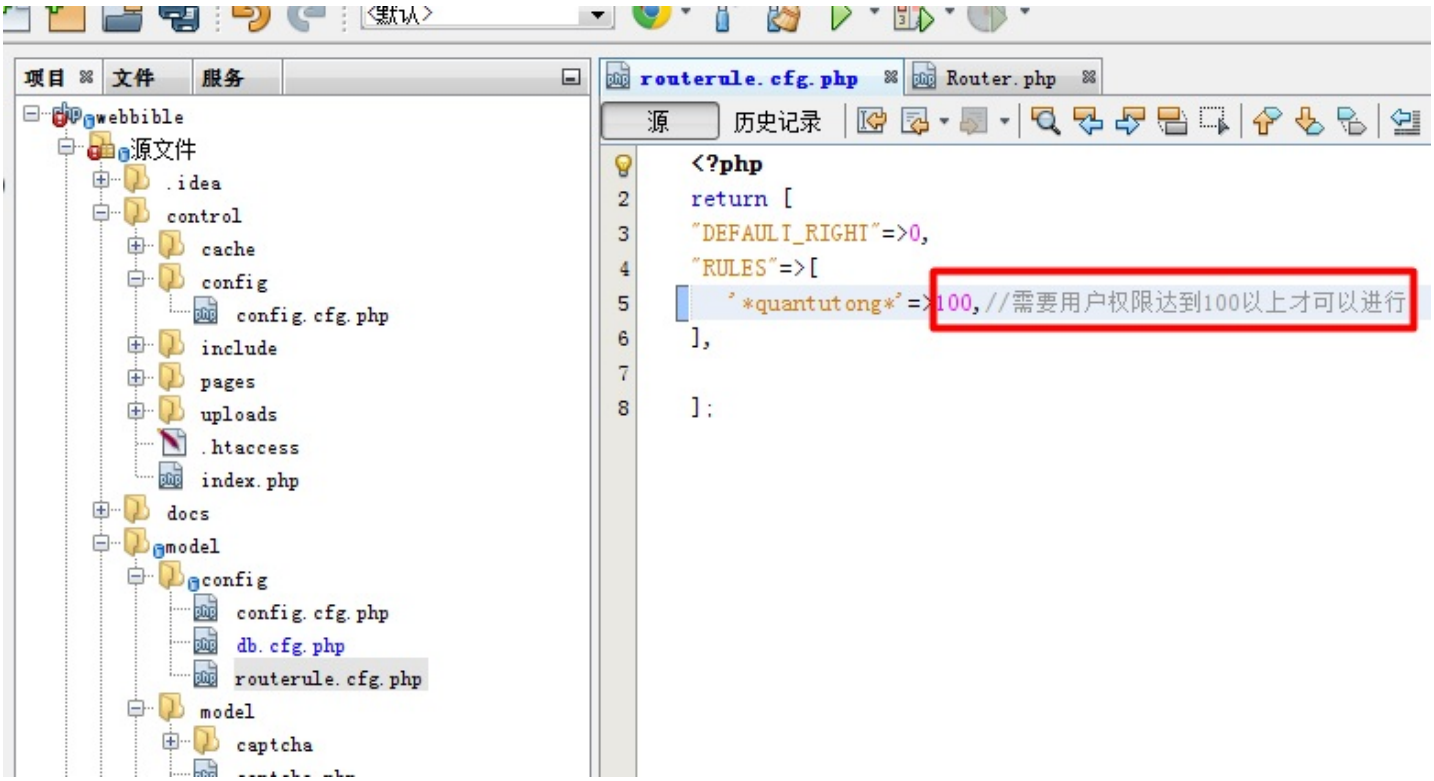
路由机制

- 路由的执行流程：
  接收用户对框架的一切访问，按照相应的规则进行转发；
  （1）当pathinfo为空时，直接转发到入口文件夹下的myindex.php，不做其他任何操作以节省消耗；参数机制怎么实现？
  （2）当pathinfo不为空时，首先匹配路由权限表（由配置文件定义），对路由进行权限鉴定；
  （3）当后缀为.php时，查找入口文件夹下相应的xxx.yyy.controller.php进行执行；
  （4）当后缀为.func时，查找用户model类中相应的函数进行执行；

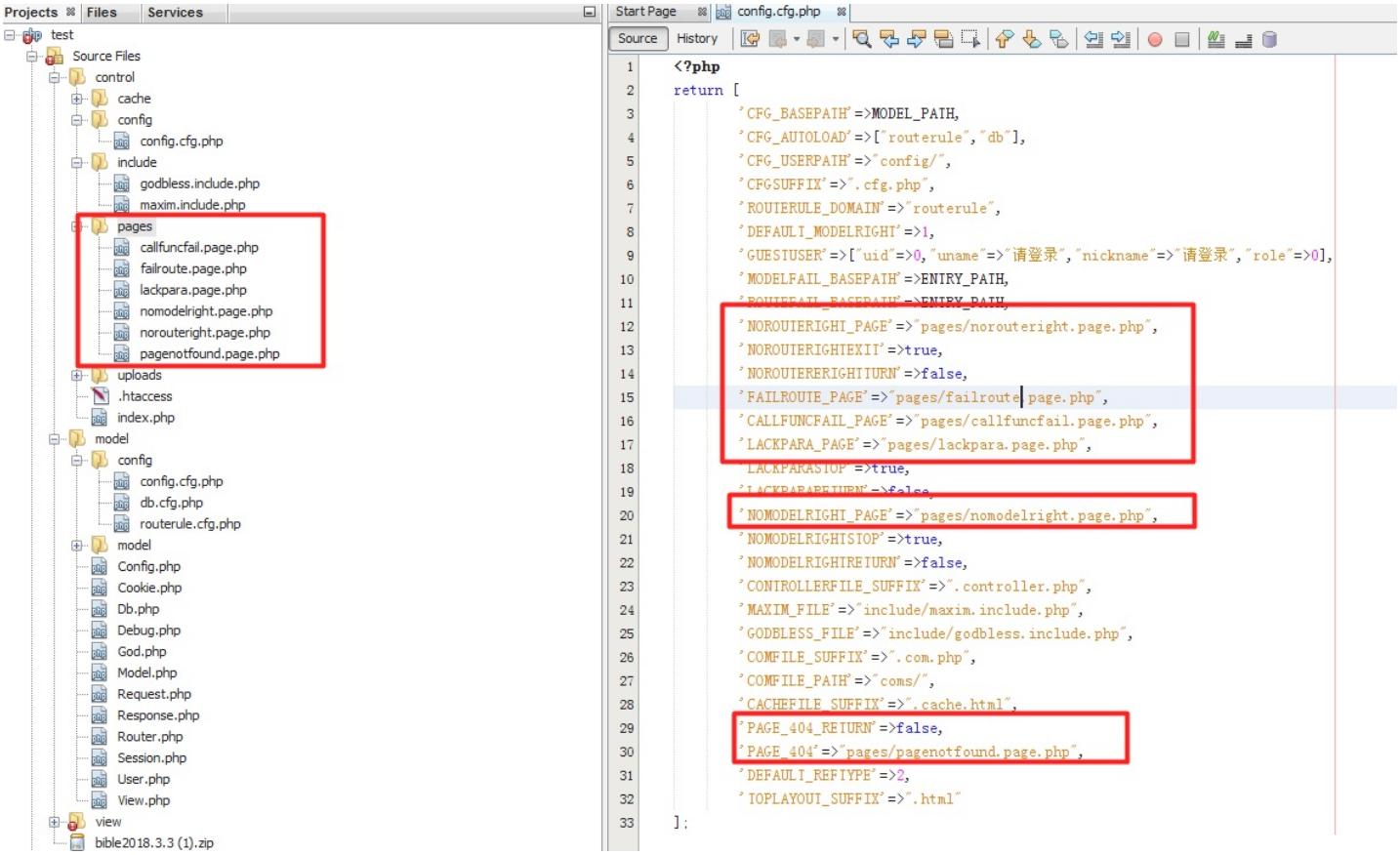
```
    public static function distribute($pathinfo) {
        if(empty($pathinfo)){
            empty(include ENTRY_PATH ."myindex.php")&&empty(include ENTRY_PATH ."ind
ex.html");
            return "";
        }
            $pathinfo=preg_replace("/^\//","",$pathinfo);
            self::$pathinfo=$pathinfo;

            (self::checkrouteright($pathinfo)||self::norouteright($pathinfo))&&
            self::beforedistribute()&&
            self::distriutetoview()||
            self::distriutetomodel()||
            self::faildistribute();
            return self::$result;
    }
```

- 路由的权限校验机制：
  框架没有实现鸡肋的路由转发功能
  每个pathinfo默认需要的用户权限为0，即任何人都可以访问
  当在rules列表中用正则表达式定义了匹配当前pathinfo的规则时（只以第一次匹配为准），则是否有权限取决于其后面的键值

```
public static $routeright=0;
public static function checkrouteright($pathinfo){
    $rules=Config::get("rules",ROUTERULE_DOMAIN,[]);
    self::$routeright=Config::get("DEFAULT_RIGHT",ROUTERULE_DOMAIN,0);
    foreach ($rules as $rule=>$rightlevel){
        if(preg_match($rule,$pathinfo)){
            self::$routeright=$rightlevel;
            return User::checkright(self::$routeright);
        }
    }
    return User::checkright(self::$routeright);
}
```



- 路由失败机制： 路由失败后，跳转到配置文件中定义的页面中
  没有路由权限：
  路由失败：不符合controller文件规则，又不符合用户model函数规则；
  文件不存在；能匹配出controller文件规则，但找不到该controller文件；
  函数不存在：能匹配出函数调用规则，但找不到该函数；



用户及权限机制

- 框架在运行的时候，一定会绑定一个用户身份，而这个用户身份一定会有它的权限值。
- 这个权限值在路由，Model执行时会被自动拿来和路由，model执行需要的权限进行对比，失败后进行报错。
- 开发者可以在任何代码处检查用户的权限值，在不满足的情况下进行跳转。
  `User::checkright(100)||Response::returnnoright("You don't have the right!!`
- 在入口文件开始执行的时候，框架默认会以一个匿名的用户身份运行，在进行登录，退出动作的时候，用户身份会自动更改。

路由时自动检测权限：

pathinfo映射到Model方法时也检查权限，在用户定义的类中，只有继承了Model类的才会有自动权限校对。没有继承的，一律不校对。

```
define("MODELFAIL_BASEPAIH",Config::get('MODELFAIL_BASEPAIH',"",ENIRY_PAIH));
define("DEFAULT_MODELRIGHT",Config::get('DEFAULT_MODELRIGHT',"",1));
define("LACKPARASTOP",Config::get('LACKPARASTOP',"",true));
define("LACKPARARETURN",Config::get('LACKPARARETURN',"",false));
define("NOMODELRIGHTSTOP",Config::get('NOMODELRIGHTSTOP',"",true));
define("NOMODELRIGHTRETURN",Config::get('NOMODELRIGHTRETURN',"",false));

!defined('NOMODELRIGHT_PAGE')&&define('NOMODELRIGHT_PAGE',Config::get('NOMODELRIGHT_PAGE',"","pages/nomodelright.page.php"));
!defined('LACKPARA_PAGE')&&define('LACKPARA_PAGE',Config::get('LACKPARA_PAGE',"","pages/lackpara.page.php"));
!defined('CALLFUNCFAIL_PAGE')&&define('CALLFUNCFAIL_PAGE',Config::get('CALLFUNCFAIL_PAGE',"","pages/callfuncfail.page.php"));
```

```
class Model
{
    public $accesslevel=null;
    public static $failcode=null;
    public static $result=null;
    function __construct() {
        is_null($this->accesslevel)&&($this->accesslevel=DEFAULT_MODELRIGHT);
        User::checkright($this->accesslevel)||$this->noright();
    }
```

在配置文件中可以修改默认Model映射的权限
'DEFAULT_MODELRIGHT'=>1,

调试及日志机制

/*实现Debug功能，拦截地址栏输入，实现访问任务的分发：
*（0）记录程序的执行过程;
*（1）错误，调试帮助；
*（2）有开关可以控制；
*/

```php
defined("DEBUG_ENABLE")||define("DEBUG_ENABLE",false); //是否使能Debug功能
defined("DEBUG_ON")||define("DEBUG_ON",false);//记录日志的时候是否直接打印出来
defined("LOG_GLUE")||define("LOG_GLUE","<br>\r\n");//打印的时候，每一条日志之间的
class Debug
{
    public static $on=DEBUG_ON;//on: 记录+输出; off: 只记录;
    public static $log=[];
    public static $logcount=0;
    public static $logglue=LOG_GLUE;
    public static function turnon(){...4 行 }
    public static function turnoff(){...4 行 }
    public static function logglue($glue=null){...8 行 }
    public static function log($obj){...15 行 }
    public static function dump($spliter="<br>\r\n"){...4 行 }
}
```

配置文件中配置 DEBUG_ENABLE

在任何需要记录日志信息的地方执行：
`DEBUG_ENABLE && Debug::log("内容");`

Session&Cookie机制

- Session和Cookie被封装了起来，不需要人为启动，在调用的时候会自动启动;
- Session在自动启动的时候会自动判断Cookie中是否有autosession=>sessionid 这样的字段，如果有的话会自动使用该Sessionid； 这样就相当于实现了session生命周期的自动延续,一直延续到cookie失效；

```
savesession($timer=-3600)
hassession()
getsession()
```

- 用户登录保存机制：

用户登录，其实就是在Session中写入一个_user变量，其值为一个User对象；所有时候，要获取用户信息，只要从Session中取_user变量的值即可

session在浏览器生命周期内一直有效，然后当用户关闭浏览器后会自动失效。通过登录的时候设置的savesession($timer=-3600)可延长session的有效期直到cookie失效，实现了用户登录状态的保存。这种方式不仅能保存用户的登录状态，而且保存了session中所有的变量，（例如，如果购物车是通过session实现的，那么也会保存下来）

User::checkright()->User::getrole()->User::info()->Session::has("_user")->Session::boot()->Session->init()->Cookie::getsession()->session_id(Cookie::get("autosession"))

cookie机制：

```
/**
     * Cookie清空
     * @param string|null $prefix cookie前缀
     * @return mixed
     */
    public static function clear($prefix = null)

        /**
     * Cookie 设置、获取、删除
     *
     * @param string $name  cookie名称
     * @param mixed  $value cookie值
     * @param mixed  $option 可选参数 可能会是 null|integer|string
     *
     * @return mixed
     * @internal param mixed $options cookie参数
     */
    public static function set($name, $value = '', $option = null)

        /**
```

```
     * 判断Cookie数据
     * @param string        $name cookie名称
     * @param string|null   $prefix cookie前缀
     * @return bool
     */
    public static function has($name, $prefix = null)

        /**
     * Cookie获取
     * @param string        $name cookie名称
     * @param string|null   $prefix cookie前缀
     * @return mixed
     */
    public static function get($name, $prefix = null)

        /**
     * Cookie删除
     * @param string        $name cookie名称
     * @param string|null   $prefix cookie前缀
     * @return mixed
     */
    public static function delete($name, $prefix = null)
```

```
* 原始cookie set函数
```

```
setcookie(name,value,expire,path,domain,secure)
参数 描述
name    必需。规定 cookie 的名称。
value   必需。规定 cookie 的值。
expire  可选。规定 cookie 的有效期。
path    可选。规定 cookie 的服务器路径。
domain  可选。规定 cookie 的域名。
secure  可选。规定是否通过安全的 HTTPS 连接来传输 cookie。
```

前后台交互规范

原则：

1. 完全的前后端分离；
2. 后端与数据库操作分离；
   规范：

- 所有的数据库操作都用存储过程实现，PHP不实现任何数据库操作语句；
- 业务执行成功与否由存储过程自行判断，只要业务执行不成功，就抛出自定义异常告知php。PHP在调用存储过程中，只要没有检测到异常，就判定为业务执行成功。然后再读取存储过程返回的数据。
- 前台向后台提交数据时要么用地址栏的地址，要么用Ajax，前台向后台传递数据时有多种方式地址栏？形式的GET，POST，pathinfo方式
- 提交的数据与映射到的后台函数的参数是一一对应的，这个对应动作和参数赋值是在Model::callfunccontrol中实现的。
  比如以pathinfo方式提交时，pathinfo的各个数据会依顺序依次传递给映射函数的参数；
  以POST或者get方式提交时，与参数名称相同的变量会被赋值给映射函数的参数；
  当存在映射函数的某些没有默认值的参数得不到赋值时，会报错“lack para”

调用方式举例：

- 当使用了不止一种方式给后台提交数据时，其给映射函数的赋值优先级是：pathinfo>POST/GET>默认参数值

其实现机制是：
1，首先按函数的定义获取一个参数名称列表；
2，对参数名称进行逐一遍历：
2.1 如果当前参数名有默认值，就先把默认值填入参数列表；
2.2 如果当前参数名有通过？的形式将参数通过get方法传递，就将其值覆盖掉默认参数值；
2.3 如果客户有通过pathinfo模式传递参数值，该值就会最终按顺序覆盖掉相应的参数值
2.4 如果某个函数名对上面三个都没成功，进行报错

- 后台向前台返回的所有业务性质的数据，全部通过Response::returntaskok() 或者 Response::returntaskfail() 方式返回
  其目的是固定封装返回的JSON为如下格式：
  `{"_taskresult":"我是数据","_taskstat":{"code":0,"info":"我是信息"}}`
  只有返回的数据是这种结构，而且code为0时，前端才能判定该业务是执行成功的（如登录成功，注册成功等），
  当成功时，前台提取_taskresult为需要的数据；
  当失败时，前台提取_taskstat.info作为错误的提示；

允许ajax跨域请求的设置

什么是跨域：

URL	说明	是否允许通信
http://www.a.com/a.js http://www.a.com/b.js	同一域名下	允许
http://www.a.com/lab/a.js http://www.a.com/script/b.js	同一域名下不同文件夹	允许
http://www.a.com:8000/a.js http://www.a.com/b.js	同一域名，**不同端口**	不允许
http://www.a.com/a.js **https**://www.a.com/b.js	同一域名，**不同协议**	不允许
http://www.a.com/a.js http://170.32.82.74/b.js	域名和域名对应ip	不允许
http://www.a.com/a.js http://script.a.com/b.js	同一域名，不同二级域名	不允许
http://**www.a.com**/a.js http://**a.com**/b.js	**二级域名和一级域名**	不允许（cookie这种情况下也不允许访问）
http://www.b.com/a.js http://www.a.com/b.js	不同域名	

解决跨域问题的方法：

- Jsonp：只能用get方式，虽然目前使用较多，但已过时。本框架不推荐；
- HTML5 后台添加文件头（IE9及以下不支持）：

```
header('Access-Control-Allow-Origin:*');//允许所有域名过来的访问
header('Access-Control-Allow-Origin:http://www.client.com');//允许指定的单个域名的访问

//通过判断实现某几个域的访问（框架可以包装成配置文件）
$origin = isset($_SERVER['HTTP_ORIGIN'])? $_SERVER['HTTP_ORIGIN'] : '';
$allow_origin = array(
    'http://www.client.com',
```

```
    'http://www.client2.com'
);
if(in_array($origin, $allow_origin)){
    header('Access-Control-Allow-Origin:'.$origin);
    header('Access-Control-Allow-Methods:POST');
    header('Access-Control-Allow-Headers:x-requested-with,content-type');
}
```

```
后台：header('Access-Control-Allow-Credentials:true');
前台：$.ajax({
url:"B.abc.com",
xhrFields:{
withCredentials:true
},
crossDomain:true
});
//这样设置可以使得后台能够访问前台的cookie
```

`header('Access-Control-Allow-Methods:POST'); //允许post方式过来的ajax`

框架的设置：

由于所有后台功能和前台页面都是通过入口文件实现的，因此只需要在入口文件处添加以上设置即可

多人协作和服务器同步

- 关于Git的简易入门图文教程：http://www.bootcss.com/p/git-guide/
- Centos下安装最新的Git

- 先删除之前的版本：yum remove git
- 官网下载最新版本：https://www.kernel.org/pub/software/scm/git/
  wget https://www.kernel.org/pub/software/scm/git/git-2.16.2.tar.gz
- 解压缩 tar zxvf git-2.16.2.tar.gz
- github不支持tlsv1.1后, 出现SSL connect error
  `git config --global http.sslversion tlsv1`

实战：开发网站后台应用

例子：开发more.sanmantech.com 网站的后台（假定该域名已经指向本机地址），实现用户注册，登录验证的后台功能

实现方法：

1，安装框架，设置好开发环境；

2，复制一下control文件夹，命名为more.sanmantech.com;

3，配置apache的documentroot或者虚拟主机documentroot 为more.sanmantech.com文件夹

4，在more.sanmantech.com 的 model目录下创建一个测试文件test.php

```
<?php
namespace model;

class test {
    public function para($arg1,$arg2=2,$arg3=3){
        echo $arg1;
        echo $arg2;
        echo $arg3;
    }
}
```

注意点：

- 命名空间必须为model；
- 类名必须与文件名一致，为test；
- 类名里面的每个成员函数即可为一个后台应用映射函数，对应一个后台业务处理的网址；
- 这个测试的para函数实现的后台功能为将前台提交的三个参数打印到浏览器

5，在浏览器中输入网址查看是否正常：

```
http://local.webbible.com/test/para.func  //输出lack para
http://local.webbible.com/test/para.func/a  //输出 a23
http://local.webbible.com/test/para.func?arg1=4 //输出423
```

6, 在more.sanmantech.com目录的config目录下新建/修改config.cfg.php 修改配置文件的基地址和自动加载db配置文件

```
<?php
return [
    'CFG_BASEPATH'=>ENTRY_PATH,  //配置框架加载本网站独有的配置文件
```

```
    'CFG_AUTOLOAD'=>["db"], //配置自动加载额外的db.cfg.php两个配置文件
    'CFG_USERPATH'=>"config/",//额外的配置文件所在的路径（和本文件在同一目录）
    'CFGSUFFIX'=>".cfg.php",//额外的配置文件的后缀
];
```

7,在more.sanmantech.com目录的config目录下新建/修改db.cfg.php，配置数据库的连接

```
<?php
return [
    "defaultdb"=>[
        "dsn"=>"",//$dsn="$dbms:host=$host;dbname=$dbName";
        "host"=>"localhost",//数据库主机名
        "dbname"=>"user",//使用的数据库,在使用存储过程方式的工程中，这个字段无关紧要
        "uname"=>"test",//数据库连接用户名
        "upass"=>"1234Abcd！",//对应的密码
        "dbtype"=>"mysql",//数据库类型
    ],
];
```

假定user数据库中已经包含两个存储过程，一个是用于登录检查，一个是用于注册新用户：

login(p_uname,p_upass)

```
BEGIN
   DECLARE matchnum INT default 0;
   call username_check(p_uname,matchnum);
   if matchnum <1 then
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No such user!';
   else
       select count(*) into matchnum from user_basic where uname=p_uname and upass=p
_upass;
       if matchnum>0 then
         select uid,uname,ulevel,uoption from user_basic where uname=p_uname and upa
ss=p_upass;
       else
         SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = 'Password incorrect!';
       end if;
   end if;
END
```

register(p_uname,p_upass)

```
BEGIN
  declare userexist int default 0;
  call username_check(p_uname,userexist);
  if userexist>0 then
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'User already exist!';
  else
       insert into user_basic set uname=p_uname,upass=p_upass;
```

```
        if row_count()<1 THEN
             SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = 'Insert failed! Affected row
is 0';
        end if;
  end if;
END
```

8,在more.sanmantech.com下的model文件夹下创建user.php

```
<?php
namespace model;

class user
{
    public function login($uname,$upass) {
        if(!\Db::simplecall("user.login",array($uname,md5($upass)))){
            \Response::returntaskfail(null,\Db::$error, \Db::$info);
        } else {
            \Response::returntaskok(\Db::tabledata());
        }
    }
    public function register($uname,$upass) {
        if(!\Db::simplecall("user.register",array($uname,md5($upass)))){
            \Response::returntaskfail(null,\Db::$error, \Db::$info);
        } else {
            \Response::returntaskok(\Db::tabledata());
        }
    }
}
```

9,后台基本功能已经实现，在浏览器中进行测试

输入一个不存在的用户：
`http://local.webbible.com/user/login.func/aaa/bbb`
返回：
`{"_taskresult":null,"_taskstat":{"code":"45000","info":"SQLSTATE[45000]: <>:`
输入：
`http://local.webbible.com/user/register.func/aaa/bbb`
返回：
`{"_taskresult":[[]],"_taskstat":{"code":0,"info":""}}`

再次输入：
`http://local.webbible.com/user/login.func/aaa/bbb`
返回：
`{"_taskresult":[["42","aaa","1",null]],"_taskstat":{"code":0,"info":""}}`

附录1：常量

- 使用define定义的常量是没有命名空间限制的，所有的命名空间可以直接访问；
- 使用cost定义的常量是有命名空间显示的，只能带着命名空间访问；
- 如果多处可能出现对同一个常量的声明，一般都先判断是否声明，因此其值为第一次定义时的值

God.php

```
define('WEBROOT_PATH',$_SERVER['DOCUMENT_ROOT']);//整个站点的根目录,假定是后面不带\的
define('ENTRY_PATH',dirname($_SERVER['SCRIPT_FILENAME']).'/');//入口文件 Path,有的时候
没入口文件呢?
define('PATH_INFO',$_SERVER['PATH_INFO']);//截取Pathinfo信息
define('MODEL_PATH',__DIR__.'/');//模型库目录,命名空间的顶部空间
define('FRAMEROOT_PATH',MODEL_PATH.'../');//框架根目录
define('VIEW_PATH',FRAMEROOT_PATH.'view/');//VIEW目录(旧工厂目录)
define('CONTROL_PATH',FRAMEROOT_PATH.'control/');//VIEW目录(旧工厂目录)
define('STATIC_PATH',VIEW_PATH.'static/');
define('COMMON_PATH',FRAMEROOT_PATH.'common/');
define('3COM_PATH',COMMON_PATH.'3com/');
define('CACHE_PATH',ENTRY_PATH.'cache/');
define('HTMLROOT_PATH',str_replace(WEBROOT_PATH,'',ENTRY_PATH)); //入口文件目录相对于站
点目录的相对路径，因为站点目录不一定是部署在根目录
define('HTMLSTATIC_PATH',HTMLROOT_PATH."static/"); //入口文件目录相对于站点目录的相对路径，
因为站点目录不一定是部署在根目录
define('HTML3COM_PATH',HTMLROOT_PATH."static/"); //入口文件目录相对于站点目录的相对路径，因
为站点目录不一定是部署在根目录
define('HTMLCACHE_PATH',HTMLROOT_PATH."cache/"); //入口文件目录相对于站点目录的相对路径，因
为站点目录不一定是部署在根目录
```

Rooter.php：

```
defined("CONTROLLERFILE_SUFFIX")||define("CONTROLLERFILE_SUFFIX",Config::get('CONTRO
LLERFILE_SUFFIX',"",".controller.php"));
define("NOROUTERIGHTEXIT",Config::get('NOROUTERIGHTEXIT',"",true)); //当发现没有路由权
限时是否要停止执行脚本
define("NOROUTERERIGHTTURN",Config::get('NOROUTERERIGHTTURN',"",false));//当发现没有路
由权限时返回什么值
define("ROUTERULE_DOMAIN",Config::get('ROUTERULE_DOMAIN',"","routerule"));
!defined('FAILROUTE_PAGE')&&define('FAILROUTE_PAGE',Config::get('FAILROUTE_PAGE',"",
"pages/failroute.page.php"));
!defined('NOROUTERIGHT_PAGE')&&define('NOROUTERIGHT_PAGE',Config::get('NOROUTERIGHT_
PAGE',"","pages/failroute.page.php"));
```

Model.php

```
define("MODELFAIL_BASEPATH",Config::get('MODELFAIL_BASEPATH',"",ENTRY_PATH));
define("DEFAULT_MODELRIGHT",Config::get('DEFAULT_MODELRIGHT',"",1));
```

```
define("LACKPARASTOP",Config::get('LACKPARASTOP',"",true));
define("LACKPARARETURN",Config::get('LACKPARARETURN',"",false));
define("NOMODELRIGHTSTOP",Config::get('NOMODELRIGHTSTOP',"",true));
define("NOMODELRIGHTRETURN",Config::get('NOMODELRIGHTRETURN',"",false));

!defined('NOMODELRIGHT_PAGE')&&define('NOMODELRIGHT_PAGE',Config::get('NOMODELRIGHT_
PAGE',"","pages/nomodelright.page.php"));
!defined('LACKPARA_PAGE')&&define('LACKPARA_PAGE',Config::get('LACKPARA_PAGE',"","pa
ges/lackpara.page.php"));
!defined('CALLFUNCFAIL_PAGE')&&define('CALLFUNCFAIL_PAGE',Config::get('CALLFUNCFAIL_
PAGE',"","pages/callfuncfail.page.php"));
```

View.php

```
defined("CONTROLLERFILE_SUFFIX")||define("CONTROLLERFILE_SUFFIX",Config::get('CONTRO
LLERFILE_SUFFIX',"",".controller.php"));
define("CACHEFILE_SUFFIX",Config::get('CACHEFILE_SUFFIX',"",".cache.html"));
define("MAXIM_FILE",Config::get('MAXIM_FILE',"","include/maxim.include.php"));
define("GODBLESS_FILE",Config::get('GODBLESS_FILE',"","include/godbless.include.php"
));
define("COMFILE_SUFFIX",Config::get('COMFILE_SUFFIX',"",".com.php"));
define("COMFILE_PATH",Config::get('COMFILE_PATH',"","coms/"));
define("DEFAULT_REFTYPE",Config::get('DEFAULT_REFTYPE',"",2));//0：分开引用；1:直接内嵌
；2：组合成一个大的JS，CSS进行引用。前提是模板要同时重建
!defined("PAGE_404")&&define("PAGE_404",Config::get('PAGE_404',"","pages/pagenotfoun
d.page.php"));

!defined("FACTORY_PATH")&&define("FACTORY_PATH",Config::get('FACTORY_PATH',"",ENTRY_
PATH."factory/"));
```

Debug.php

```
defined("DEBUG_ENABLE")||define("DEBUG_ENABLE",false); //是否使能Debug功能
defined("DEBUG_ON")||define("DEBUG_ON",false);//记录日志的时候是否直接打印出来
defined("LOG_GLUE")||define("LOG_GLUE","<br>\r\n");//打印的时候，每一条日志之间的
```

附录2：配置项

- 配置文件后缀默认为xxx.cfg.php

```
<?php
return [
    'CFG_BASEPATH'=>MODEL_PATH,//额外的配置文件的基地址，框架默认的两个基本的配置文件不受以下四个变量控制
    'CFG_AUTOLOAD'=>["routerule","db"],//需要自动加载的额外的配置文件，每个文件名是一个domain,如果要连接数据库，必须放db在里面且名称不可更改
    'CFG_USERPATH'=>"config/",//额外的配置文件相对于基地址的路径
    'CFGSUFFIX'=>".cfg.php",//额外的配置文件的后缀名字
    'ROUTERULE_DOMAIN'=>"routerule",//路由权限规则表在配置中的哪个domain
    'DEFAULT_MODELRIGHT'=>1,//默认的继承自Model类的类执行所需要的用户权限
        'REQUEST_LIMIT'=>false,//配置全局的是否限制只响应POST请求
    'GUESTUSER'=>["uid"=>0,"uname"=>"请登录","nickname"=>"请登录","role"=>0],//当用户未登录时查询用户信息所显示的内容
    //'MODELFAIL_BASEPATH'=>ENTRY_PATH,丢弃
    //'ROUTEFAIL_BASEPATH'=>ENTRY_PATH,丢弃
    'NOROUTERIGHT_PAGE'=>"pages/norouteright.page.php",//没有路由权限时显示的页面
    'NOROUTERIGHTEXIT'=>true,//没有路由权限时，是否停止执行后面的脚本
    //'NOROUTERERIGHTTURN'=>false,丢弃
    'FAILROUTE_PAGE'=>"pages/failroute.page.php",//路由失败时显示的页面
    'CALLFUNCFAIL_PAGE'=>"pages/callfuncfail.page.php",//调用映射函数失败时显示的页面
    'LACKPARA_PAGE'=>"pages/lackpara.page.php",//缺乏必须的参数时显示的页面
    'LACKPARASTOP'=>true,//缺乏参数时，是否停止执行后面的脚本
    'LACKPARARETURN'=>false,//缺乏参数时，当选择继续执行后面脚本时lackpara检测函数的返回值
    'NOMODELRIGHT_PAGE'=>"pages/nomodelright.page.php",//没有权限执行映射函数时显示的页面
    'NOMODELRIGHTSTOP'=>true,//没有权限执行映射函数时是否停止执行后面的代码
    'NOMODELRIGHTRETURN'=>false,//没有权限执行映射函数时当选择继续执行后面的代码时检测函数的返回值
    'CONTROLLERFILE_SUFFIX'=>".controller.php",//控制器函数的后缀
    'MAXIM_FILE'=>"include/maxim.include.php",//这个文件中存放PHP数组，当通过框架显示前台页面时，框架自动把这些变量转换为js变量
    'GODBLESS_FILE'=>"include/godbless.include.php",//这个文件中存放对前台框架运行必须加载的js，css，通过框架显示前台页面时，框架会自动加载这个
    'COMFILE_SUFFIX'=>".com.php",//xxx
    'COMFILE_PATH'=>"coms/",//xxx
    'CACHEFILE_SUFFIX'=>".cache.html",  //xxx
    'PAGE_404_RETURN'=>false,//xxx
    'PAGE_404'=>"pages/pagenotfound.page.php",//找不到前台页面时显示的页面
    'DEFAULT_REFTYPE'=>2,//xxx
    'TOPLAYOUT_SUFFIX'=>".html"//顶级模版文件的后缀
];
```

db.cfg.php

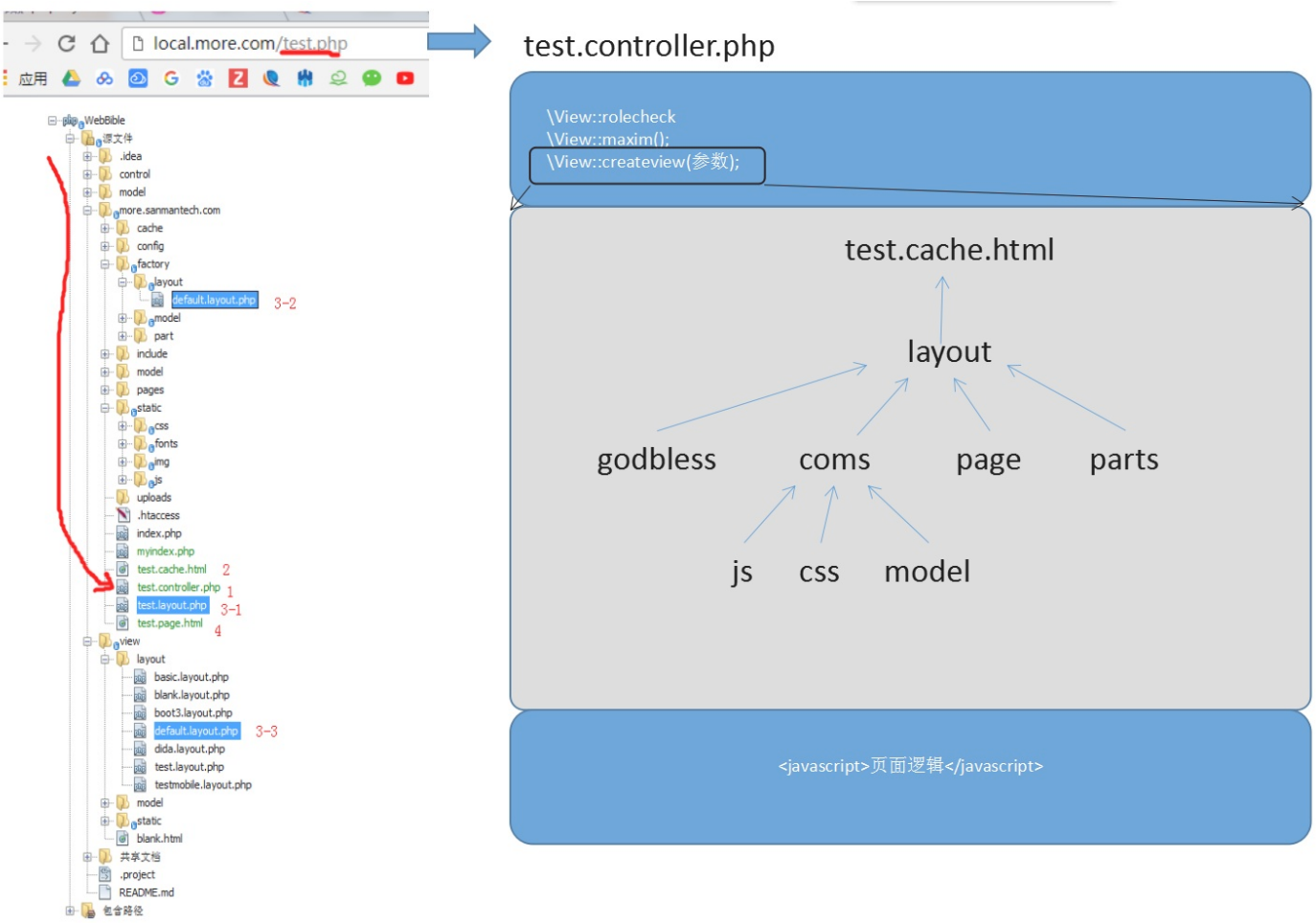
```
<?php
return [
```

```
    "defaultdb"=>[
            "dsn"=>"",//$dsn="$dbms:host=$host;dbname=$dbName";
            "host"=>"localhost",//数据库主机名
            "dbname"=>"quanwutong",//使用的数据库,在使用存储过程方式的工程中，这个字段无关紧要
            "uname"=>"root",//数据库连接用户名
            "upass"=>"1234Abcd!",//对应的密码
            "dbtype"=>"mysql",//数据库类型
        ],
];
```

routerule.cfg.php

```
<?php
return [
"DEFAULT_RIGHT"=>0,
"RULES"=>[
   '*quantutong*'=>100,//需要用户权限达到100以上才可以进行
],

];
```

前台页面的映射机制



- 当访问 http://主机/xxx/yyy.php 时，实际访问的时入口文件夹下的xxx.yyy.controller.php
- 一切逻辑源自此controller，一般controller中做如下动作：
  （1）检查用户访问该页面的权限（php）
  （2）返回php要传递给该页面的初始数据（一般是用户信息，Staticpath，rootpath，用户权限值等）（PHP）
  （3）显示静态页面内容：（PHP实现，内容为纯静态的HTML页面）
  （a），如果允许调用缓存，并且缓存文件存在，并且缓存文件判断未过时，则直接显示缓存文件；
  （b），否则则通过createview组装一个缓存文件，该缓存文件必定是纯HTML的，并且和PHP无关；
  （c），组装的过程就是将构造参数传递给恰当的模版文件（php），模版文件会自动按照传递进来的参数填充js引用，css引用，coms，page，parts等
  （d），组装完成后会生成缓存文件保存，并显示到前台
  （4）真正主角：javascript登场，页面所有的UI显示，业务逻辑的实现和后台交互的动作全部由javascript完成！
- 当地址栏仅输入主机名时是特殊情况，此时入口文件加载完框架后会直接加载myindex.php文件，如果该文件不存在，便会输出index.html文件，如果还是没有，就什么都不输出...
- 当地址栏中输入的文件或者路径名能正常访问到文件时，则直接打开该文件，根本不会进入到入口文件，前面的所有机制描述都不适用

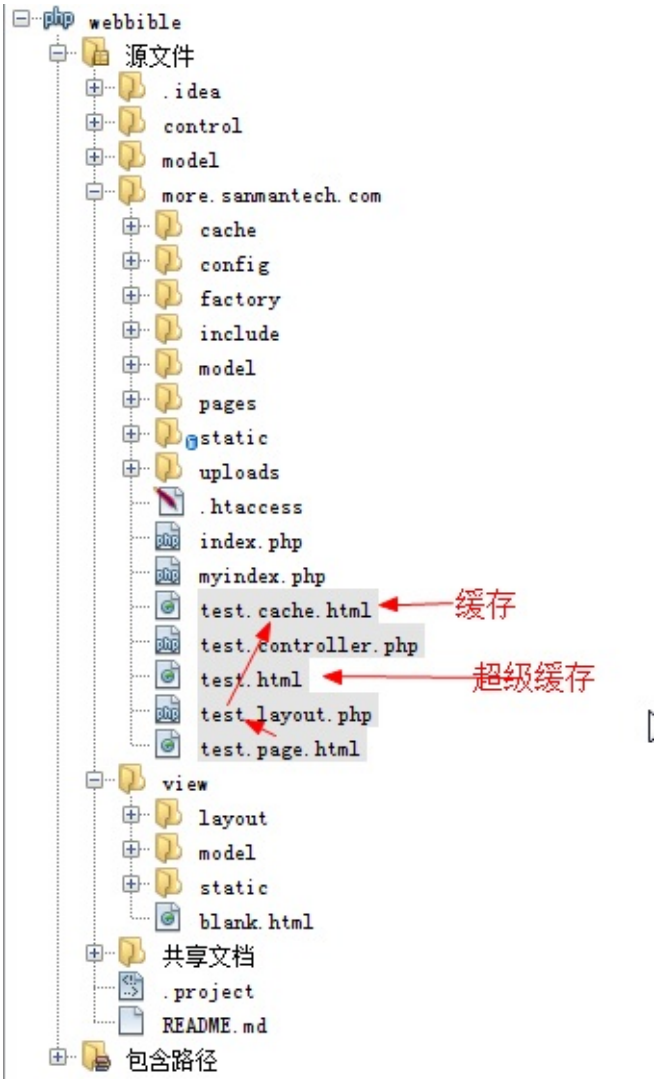
前台缓存及布局组装机制

缓存分类：

- 页面缓存
- js，css调用缓存

缓存机制：

- 只有在controller中使用了\View::createview($items=["default.layout"],$usecache=1,$reftype=null)时，才会启动缓存，布局组装的机制
- 当存在超级缓存文件时（xxx.yyy.html），无论你有没有开启缓存功能，都将直接显示它而不执行下面的内容
- 在地址栏访问时提供forcenewcache参数，则会强制不使用普通页面缓存（超级缓存不受此开关影响）
- 对于一个确定的controller文件，其缓存文件名称和路径是确定的xxx.yyy.cache.html（普通）xxx.yyy.html(超级)
- 缓存文件一定是一个与后台数据无关，用户登录状态无关即适合所有场景的纯静态页面；
- 普通缓存页面的初始化数据由\View::maxim()决定
- 普通缓存页面的业务逻辑实现是由controller中定义的javascript决定



缓存功能的开关

（1）地址栏中的forcenewcache

（2）createview中的第二个参数

（3）超级缓存文件不受开关和createview 布局参数影响

页面缓存文件过期的判定标准

缓存文件比controller文件新（这种判断不正确，当文档引用的js，css更新时缓存文件本应更新却不会更新）

布局组装机制

- 当未开启缓存功能或者缓存文件过期时才会启动布局组装
- 布局组装的关键是找到布局文件：
  （1）布局文件由\View::createview第一个数组参数的第一个元素指定，格式必须为xx.layout，如果系统判定为不是这种格式，则直接使用degault.layout
  （2）当入口文件夹下存在xxx.yyy.layout.php时，将会直接选用它，而不会执行后面的2、3；
  （3）判断入口文件夹下的factory/layout/目录下是否有要求的布局文件，有则使用
  （4）判断VIEW文件夹下的layout/目录下是否有要求的布局文件，有则使用
- 找到布局文件后，其实现原理是直接include这个php，布局文件会自动填充自己需要的内容；
- 布局组装填充什么内容（选什么模版，填充什么组件，加载什么js，css），由\View::createview中的

组装参数决定；而按照什么格式组装（页面片段：parts、page 以及所有填充内容的位置），由布局文件决定；

jscss的加载缓存机制

- 仅在开启了合并文件引用的方式（默认方式）时起作用
- 首先判断Createview中的布局数组顺序和内容是否有更改
- 其次判断缓存文件的日期是否比布局数组中所有要引用的文件新 （好像还忘了一个点，想不起来了）

前台组件加载机制

框架定义的系统组件

```
//数据交换组件
if(God.coms("ajax")){...190 行 }
God.coms("datasource").extendproto({...20 行 });
God.coms("datavalidation").extendproto({...43 行 });
God.coms("dataexchange").extendproto({...31 行 });
God.coms("page").extendproto({...18 行 });
God.coms("event").extendproto({...35 行 });
God.coms("dialog").extendproto({...27 行 });
God.coms("autocomplete").extendproto({...37 行 });

God.coms("ui").extendproto({...124 行 });
```

组件（coms）、片段（part）、页面（page）的区别：

- 组件是页面中可以动态添加，删除，反复使用的可见或不可见功能部件，组件加载完后不显示任何东西
- 片段是网页中除掉HTML特征标签后的一部分网页HTML代码，它是为了解决多个页面中有相同代码片段的简单共享机制，片段加载后会直接显示，不具有组件反复复制的功能
- 页面（page）是特殊的一个代码片段，每个网页中顶多有一个页面，它是这个网页中最主要的页面主体的内容

组件的组成和引入

- 一个组件比如xxx，是由sm.xxx.js、sm.xxx.css、xxx.model.html 三个部分组成的
- 一个页面需要哪些组件，是在controller文件执行createview时，由布局参数指定的

```
<?php
    \View::rolecheck(101,">=","","/user/login.php");
    \View::maxim();
    \View::createview(["godbless","bootstrap3.3.5"]);
?>
```

- 框架会自动按照既定的规则搜索并加载组件中的js和css文件，以及model文件；
- js和css的内容也可以简单的融合到model.html中
- 对第三方js，css的调用和引用，也可以将其按照组件方式命名并转化为组件式加载方式
  布局参数的组成
  模版（xxx.layout）[可选],godbless[可选，是否加载godbless],组件1，组件2....
  当引入js和css时，如果选择合并文件并引入，其会强制使用缓存机制，这是为了解决开发过程中浏览器使用缓存而不能刷新页面以及正常部署时的效率提升

前台模版渲染机制

模版渲染的相关接口

```
God.coms("ui").extendproto({//对话框
make:function(model,data,curlevel,extradata,datafunc){...48 行 },//真实执行模版替换，由多个模版替换动作完成
makeout:function(modelname,data,def,extradata,datafunc){...6 行 },//执行模版替换返回格式为
jqmakeout:function(modelname,data,def,datafunc){...3 行 },//执行模版替换，返回格式为jq格式
warehouse:function(modelname){...20 行 },//从仓库中获取模版原型
parsepara:function(paras,def){...32 行 },//把数组改为关联数组
modelreplace:function(str,obj){...12 行 },//模版的简单替换动作

});
```

参数解析机制

- parsepara的作用就是把一个普通数组的最后一层按照def的格式调整成关联数组模式，比如：paras=[[11,22,33],[44,55,66]],def={uid:1,upass:2,ww:3} =>[{uid:11,upass:22,ww:33},{uid:44,upass:55,ww:66}]
- 当def的长度比数组参数长时，多余的参数值按def来进行；paras=[[11,22,33],[44,55]],def={uid:1,upass:2,ww:3} =>[{uid:11,upass:22,ww:33},{uid:44,upass:55,ww:3}]
- 当def的长度比数组参数短时，相差的参数以paran来命名；paras=[[11,22,33],[44,55，66]],def={uid:1} =>[{uid:11,para1:22,para2:33},{uid:44,para1:55,para2:3}]
- 特殊情况：

-- paras为字符串时，直接返回，后端makeout是

-- def为字符串时，视为只有一个元素的对象

-- 当paras的最后一层不是对象时，makeout会把每一个底层元素视为一个只有一个元素且名字为para的对象
比如，如果传递[11,22,33]给makeout，makeout会认为传递是：[{para:11},{para:22},{para:33}]

额外参数及参数替换前的预处理

- extradata中可以列一些额外的数据，每次替换前会和paradata合并然后再进行替换；
- il这个变量会被自动添加到extradata中，因此在模版中可以直接使用{il}
- 如果定义了datafunc函数，那么在执行替换前，会先执行datafunc函数，this指向paradata本身，判定后进行更改，然后再替换。

模版的书写方法

```
<div class="list-group biblemodel domainlist">
  <a href="#" class="list-group-item " il=1 style="border-radius:0px;"><span class="badge">{para1}</span>{para0}</a>
</div>
```

Model的最外层，每次makeout就复制一个

第一层循环

数组最后一层的参数

模版渲染方法

```
sm.ui.jqmakeout('domainlist',[[1,2],[3,4]],{}).appendTo("body");
```

前台变量自动交互机制

在HTML代码中无论什么元素都可以绑定一个js变量，绑定方法如下：

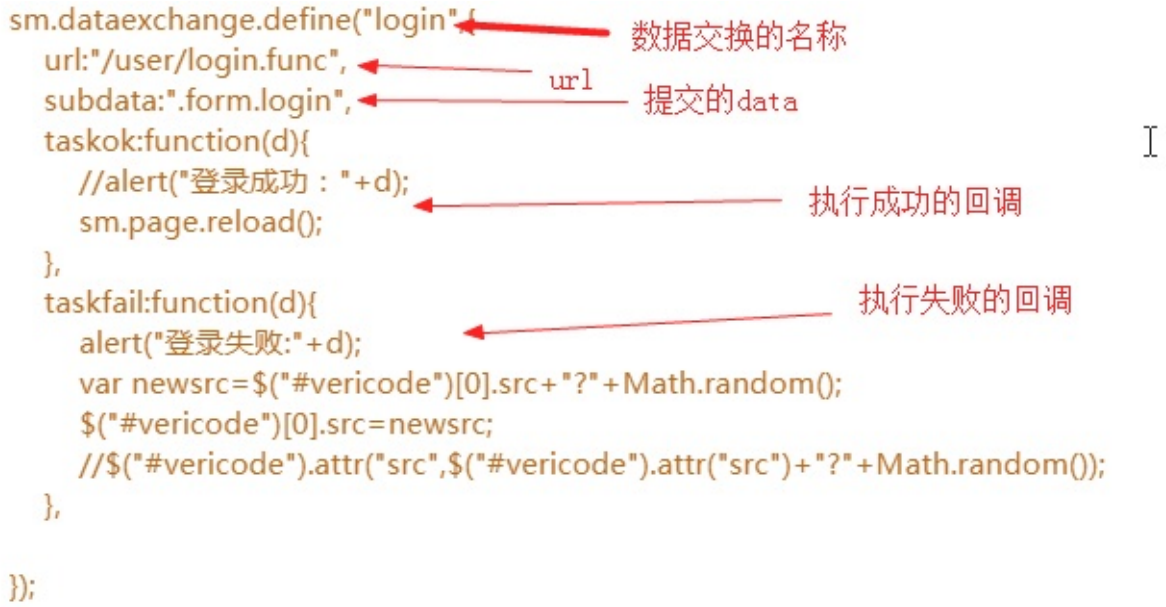
- 引用型的值可以直接读取任何一个另外元素的值，或者一个数组：

则uname的值为选择器.filelist 所有值组成的数组

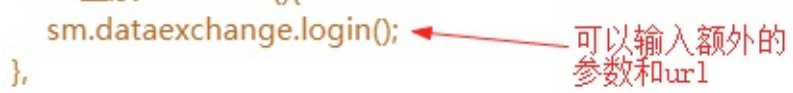
JS对HTML变量的访问如下：
God.func.htmlvar=function(varname,valueifnoexist)//获取单个HTML变量的值，如果不存在则取默认值
God.func.htmlvars=function(filter) //获取某一组HTML变量的值（以类名filter为选择器）返回包含他们的对象

前台表单提交及验证机制

页面交互的定义：



页面交互的执行：



页面交互的过程：

当调用页面交互的时候：

1，会按照给定的选择器选择所有的HTML vars变量；

2，会逐一检查每个var是否定义了validation校验规则（正则或者函数）；

3，如果校验失败，则执行校验失败的函数，并终止交互的动作

4，如果全部校验成功，则post到定义好的URL中，并且弹出屏幕显示：正在ajax中....等交互完成以后进行隐藏

```
<script language="javascript" type="text/javascript">

//0-6 程序执行的开始，声明页面必须的变量，变量的要求，以及数据交换业务的定义,以及事件
!(typeof God==='undefined')&&!(typeof jQuery==='undefined')&&$(document).ready(funct
ion(){
    //1-6 定义页面中要使用的全部变量声明
    sm.datasource.extend({
        title:"欢迎登录！",//文档的标题
        uname:"",
        upass:"",
        vericode:"",
        keeplogin:-3600,
```

```
    }).mergemaxim();
    //2-6 定义页面中数据的校验
    sm.datavalidation.extend({
        uname:/@/,
        upass:/.{8,}/,
        vericode:/^.{4}$/,
        keeplogin:function(){
            //alert(JSON.stringify(this));
            if($$.htmlvar("keeplogin")){
                this.keeplogin=60*60*24*7;
                //alert(this.keeplogin);
            } else {
                this.keeplogin=-3600;
                //alert(this.keeplogin);
            }

            return true;
        },
    });
    sm.datavalidation.fail=function(varname){
        $$.jqhtmlvar(varname).addClass("alert alert-danger");
        $(".validfailinfo").html("数据校验不通过：'+varname).show();
    }
    //3-6 定义页面数据用到的交换,
    sm.dataexchange.define("login",{
        url:"/user/login.func",
        subdata:".form.login",
        taskok:function(d){
            //alert("登录成功："+d);
            sm.page.reload();
        },
        taskfail:function(d){
            alert("登录失败:"+d);
            var newsrc=$("#vericode")[0].src+"?"+Math.random();
            $("#vericode")[0].src=newsrc;
            //$("#vericode").attr("src",$("#vericode").attr("src")+"?"+Math.random()
);
        },

    });
    //4-6 定义页面中的事件地图
    sm.event.map("body *","click.button",{
        'text:登录':function(){
            sm.dataexchange.login();
        },

        'text:没有账户?点击注册':function(){
            sm.page.reload("/user/register.php");
        },
    }).map("body *","change.input",{
        'input.form':function(){
            $(this).removeClass("alert alert-danger");
```

```
            $(".validfailinfo").html("").hide();
        },

    });
    //5-6 执行初始化
    $$.autominheighttowindow();
    sm.autoinit();

});


//6-6 定义自定义变量，函数

</script>
```

\n\n~~~"},"options":{"plugins_host":null,"context":"ebook","base":"./","features":["internal_link"]},"style":""}

前台事件代理机制

```
//4-6 定义页面中的事件地图
sm.event.map("body *","click.button",{
    'text:登录':function(){
        sm.dataexchange.login();
    },

    'text:没有账户?点击注册':function(){
        sm.page.reload("/user/register.php");
    },
}).map("body *","change.input",{
    'input.form':function(){
        $(this).removeClass("alert alert-danger");
        $(".validfailinfo").html("").hide();
    },

});
```

以文本对应执行的函数

直接以选择器定义对应的函数

前台其他常用内置组件

页面相关操作：

```
God.coms("page").extendproto({//目的是
reload:function(newurl){...8 行 },
ram:{},
open:function(newurl){...4 行 },
```

对话框：

```
God.coms("dialog").extendproto({//对话框

show:function(title,content,funcmap){...8 行 },
countshow:function(s,title,content){...10 行 },
close:function(filter){...5 行 },

});
```

直接的数据交互：

```
sm.ajax.url(url).post(paras).success(tcb.success).error(tcb.error).taskok(tcb
```

Jquery的扩充：

```
jQuery.fn.extend({//扩充Jquery开始
sm:sm,//完善与sm的链式操作
outerhtml:function(){...6 行 },//获取当前节点的outerhtml
innertext:function(txt){...14 行 },//这个函数的目的是在设置节点下面的文本时而不删除子节点
showme:function(){...9 行 },//打印出当前集合中的outerhtml
safedo:function(f){...10 行 },//当集合不为空的时候才执行
naming:function(id){...8 行 },//这个函数的作用是给当前集合按顺序打上一个id标签
additems:function(type,items){...21 行 },//这个函数的目的是想把关键字中container和filler匹配的上的才append进来，不匹配的自动忽略
addto:function(selector,type){...4 行 },//和additems相同，调用者不同
```

全局函数：

```
God.func.isempty=function(varname){return !varname;}
God.func.isnull=function(varname){return  (!varname&& typeof(varname)!="undefined" && varname!=0);}
God.func.isobj=function(varname){return Object.prototype.toString.call(varname) === '[object Object]'; }
God.func.isarray=function(varname){return Object.prototype.toString.call(varname) === '[object Array]'; }
God.func.isna=function(varname){return isNaN(varname);}
God.func.isundefined=function(varname){return typeof(varname) == "undefined";}
God.func.isdefined=function(varname){return !(typeof(varname) == "undefined");}
God.func.isfunction=function(varname){return (typeof(varname) == "function");}

God.func.hasmaxim=function(){return (typeof (maxim)!='undefined') && $$.isobj(maxim);}//是否存在maxim
God.func.merge=function(o){for (x in o) {this[x]=o[x];} return this;},//把给定的对象的全部属性融合到自身中
God.func.safemerge=function(o){for (x in o) {this.hasOwnProperty(x)&&(this[x]=o[x]);} return this;}//把给定的对象中与自身相交的属性值更新到自身中
```

实战：6步法开发网站前台登录页面

```
<script language="javascript" type="text/javascript">
//0-6 程序执行的开始，声明页面必须的变量，变量的要求，以及数据交换业务的定义,以及事件
!(typeof God==='undefined')&&!(typeof jQuery==='undefined')&&$(document).ready(funct
ion(){
    //1-6 定义页面中要使用的全部变量声明
    sm.datasource.extend({
        title:"欢迎登录！",//文档的标题
        uname:"",
        upass:"",
        vericode:"",
        keeplogin:-3600,
    }).mergemaxim();
    //2-6 定义页面中数据的校验
    sm.datavalidation.extend({
        uname:/@/,
        upass:/.{8,}/,
        vericode:/^.{4}$/,
        keeplogin:function(){
            if($$.htmlvar("keeplogin")){
                this.keeplogin=60*60*24*7;
            } else {
                this.keeplogin=-3600;
            }
            return true;
        },
    });
    sm.datavalidation.fail=function(varname){
        $$.jqhtmlvar(varname).addClass("alert alert-danger");
        $(".validfailinfo").html("数据校验不通过："+varname).show();
    }
    //3-6 定义页面数据用到的交换，
    sm.dataexchange.define("login",{
        url:"/user/login.func",
        subdata:".form.login",
        taskok:function(d){
            sm.page.reload();
        },
        taskfail:function(d){
            alert("登录失败:"+d);
            var newsrc=$("#vericode")[0].src+"?"+Math.random();
            $("#vericode")[0].src=newsrc;
        },

    });
    //4-6 定义页面中的事件地图
    sm.event.map("body *","click.button",{
        'text:登录':function(){
            sm.dataexchange.login();
```

```
        },
        'text:没有账户?点击注册':function(){
            sm.page.reload("/user/register.php");
        },
    }).map("body *","change.input",{
        'input.form':function(){
            $(this).removeClass("alert alert-danger");
            $(".validfailinfo").html("").hide();
        },
    });
    //5-6 执行初始化
    $$.autominheighttowindow();
    sm.autoinit();
});
//6-6 定义自定义变量，函数

</script>
```

\n\n~~~"},"options":{"plugins_host":null,"context":"ebook","base":"./","features":["internal_link"]},"style":""}

其他必备知识总结

Mysql存储过程语法

Mysql存储过程语法

存储过程如同一门程序设计语言，同样包含了数据类型、流程控制、输入和输出和它自己的函数库。

----------基本语法----------

一.创建存储过程
create procedure sp_name()
begin
.....
end

二.调用存储过程
1.基本语法：call sp_name()
注意：存储过程名称后面必须加括号，哪怕该存储过程没有参数传递

三.删除存储过程
1.基本语法：
drop procedure sp_name//

2.注意事项
(1)不能在一个存储过程中删除另一个存储过程，只能调用另一个存储过程

四.其他常用命令

1.show procedure status
显示数据库中所有存储的存储过程基本信息，包括所属数据库，存储过程名称，创建时间等

2.show create procedure sp_name
显示某一个MySQL存储过程的详细信息

----------数据类型及运算符----------
一、基本数据类型：
略

二、变量：

自定义变量：DECLARE a INT ; SET a=100; 可用以下语句代替：DECLARE a INT DEFAULT 100;

变量分为用户变量和系统变量，系统变量又分为会话和全局级变量

用户变量：用户变量名一般以@开头，滥用用户变量会导致程序难以理解及管理

1、 在mysql客户端使用用户变量
mysql> SELECT 'Hello World' into @x;

mysql> SELECT @x;

mysql> SET @y='Goodbye Cruel World';
mysql> select @y;

mysql> SET @z=1+2+3;
mysql> select @z;

2、 在存储过程中使用用户变量

mysql> CREATE PROCEDURE GreetWorld( ) SELECT CONCAT(@greeting,' World');
mysql> SET @greeting='Hello';
mysql> CALL GreetWorld( );

3、 在存储过程间传递全局范围的用户变量
mysql> CREATE PROCEDURE p1( ) SET @last_procedure='p1';
mysql> CREATE PROCEDURE p2( ) SELECT CONCAT('Last procedure was ',@last_procedure);
mysql> CALL p1( );
mysql> CALL p2( );

三、运算符：
1.算术运算符

```
加   SET var1=2+2;       4
```

```
减   SET var2=3-2;       1
```

```
 乘   SET var3=3*2;       6
```

/ 除 SET var4=10/3; 3.3333
DIV 整除 SET var5=10 DIV 3; 3
% 取模 SET var6=10%3 ; 1

2.比较运算符
> 大于 1>2 False
< 小于 2<1 False
<= 小于等于 2<=2 True
>= 大于等于 3>=2 True
BETWEEN 在两值之间 5 BETWEEN 1 AND 10 True
NOT BETWEEN 不在两值之间 5 NOT BETWEEN 1 AND 10 False

IN 在集合中 5 IN (1,2,3,4) False
NOT IN 不在集合中 5 NOT IN (1,2,3,4) True
= 等于 2=3 False
<>, != 不等于 2<>3 False
<=> 严格比较两个NULL值是否相等 NULL<=>NULL True
LIKE 简单模式匹配 "Guy Harrison" LIKE "Guy%" True
REGEXP 正则式匹配 "Guy Harrison" REGEXP "[Gg]reg" False
IS NULL 为空 0 IS NULL False
IS NOT NULL 不为空 0 IS NOT NULL True

3.逻辑运算符

4.位运算符
| 或
& 与
<< 左移位
>> 右移位
~ 非(单目运算，按位取反)

注释：

mysql存储过程可使用两种风格的注释
双横杠：--

该风格一般用于单行注释
c风格：/* 注释内容 */ 一般用于多行注释

----------流程控制----------
一、顺序结构
二、分支结构
if
case

三、循环结构
for循环
while循环
loop循环
repeat until循环

注：
区块定义，常用
begin
.....

end;
也可以给区块起别名，如：
lable:begin
.....
end lable;
可以用leave lable;跳出区块，执行区块以后的代码

begin和end如同C语言中的{ 和 }。

----------输入和输出----------

mysql存储过程的参数用在存储过程的定义，共有三种参数类型,IN,OUT,INOUT
Create procedure|function([[IN |OUT |INOUT] 参数名 数据类形...])

IN 输入参数
表示该参数的值必须在调用存储过程时指定，在存储过程中修改该参数的值不能被返回，为默认值

OUT 输出参数
该值可在存储过程内部被改变，并可返回

INOUT 输入输出参数
调用时指定，并且可被改变和返回

IN参数例子：
CREATE PROCEDURE sp_demo_in_parameter(IN p_in INT)
BEGIN
SELECT p_in; --查询输入参数
SET p_in=2; --修改
select p_in;--查看修改后的值
END;

执行结果:
mysql> set @p_in=1
mysql> call sp_demo_in_parameter(@p_in)
略
mysql> select @p_in;
略
以上可以看出，p_in虽然在存储过程中被修改，但并不影响@p_id的值

OUT参数例子
创建:
mysql> CREATE PROCEDURE sp_demo_out_parameter(OUT p_out INT)
BEGIN

```
SELECT p_out;/查看输出参数/
SET p_out=2;/修改参数值/
SELECT p_out;/看看有否变化/
END;
```

执行结果:

```
mysql> SET @p_out=1
mysql> CALL sp_demo_out_parameter(@p_out)
```

略

```
mysql> SELECT @p_out;
```

略

INOUT参数例子：

```
mysql> CREATE PROCEDURE sp_demo_inout_parameter(INOUT p_inout INT)
BEGIN
SELECT p_inout;
SET p_inout=2;
SELECT p_inout;
END;
```

执行结果：

```
set @p_inout=1
call sp_demo_inout_parameter(@p_inout) //
```

略

```
select @p_inout;
```

略

附：函数库

mysql存储过程基本函数包括：字符串类型，数值类型，日期类型

一、字符串类

```
CHARSET(str) //返回字串字符集
CONCAT (string2 [,... ]) //连接字串
INSTR (string ,substring ) //返回substring首次在string中出现的位置,不存在返回0
LCASE (string2 ) //转换成小写
LEFT (string2 ,length ) //从string2中的左边起取length个字符
LENGTH (string ) //string长度
LOAD_FILE (file_name ) //从文件读取内容
LOCATE (substring , string [,start_position ] ) 同INSTR,但可指定开始位置
LPAD (string2 ,length ,pad ) //重复用pad加在string开头,直到字串长度为length
LTRIM (string2 ) //去除前端空格
```

REPEAT (string2 ,count) //重复count次
REPLACE (str ,search_str ,replace_str) //在str中用replace_str替换search_str
RPAD (string2 ,length ,pad) //在str后用pad补充,直到长度为length
RTRIM (string2) //去除后端空格
STRCMP (string1 ,string2) //逐字符比较两字串大小,
SUBSTRING (str , position [,length]) //从str的position开始,取length个字符,
注：mysql中处理字符串时，默认第一个字符下标为1，即参数position必须大于等于1
mysql> select substring(‘abcd’,0,2);
+————————————+
| substring(‘abcd’,0,2) |
+————————————+
| |
+————————————+
1 row in set (0.00 sec)

mysql> select substring(‘abcd’,1,2);
+————————————+
| substring(‘abcd’,1,2) |
+————————————+
| ab |
+————————————+
1 row in set (0.02 sec)

TRIM([[BOTH|LEADING|TRAILING] [padding] FROM]string2) //去除指定位置的指定字符
UCASE (string2) //转换成大写
RIGHT(string2,length) //取string2最后length个字符
SPACE(count) //生成count个空格

二、数值类型

ABS (number2) //绝对值
BIN (decimal_number) //十进制转二进制
CEILING (number2) //向上取整
CONV(number2,from_base,to_base) //进制转换
FLOOR (number2) //向下取整
FORMAT (number,decimal_places) //保留小数位数
HEX (DecimalNumber) //转十六进制
注：HEX()中可传入字符串，则返回其ASC-11码，如HEX(‘DEF’)返回4142143
也可以传入十进制整数，返回其十六进制编码，如HEX(25)返回19
LEAST (number , number2 [,..]) //求最小值
MOD (numerator ,denominator) //求余

POWER (number ,power) //求指数
RAND([seed]) //随机数
ROUND (number [,decimals]) //四舍五入,decimals为小数位数]

注：返回类型并非均为整数，如：

(1)默认变为整形值
mysql> select round(1.23);
+————————-+
| round(1.23) |
+————————-+
| 1 |
+————————-+
1 row in set (0.00 sec)

mysql> select round(1.56);
+————————-+
| round(1.56) |
+————————-+
| 2 |
+————————-+
1 row in set (0.00 sec)

(2)可以设定小数位数，返回浮点型数据

mysql> select round(1.567,2);
+———————————-+
| round(1.567,2) |
+———————————-+
| 1.57 |
+———————————-+
1 row in set (0.00 sec)

SIGN (number2) //返回符号,正负或0
SQRT(number2) //开平方

三、日期类型

ADDTIME (date2 ,time_interval) //将time_interval加到date2
CONVERT_TZ (datetime2 ,fromTZ ,toTZ) //转换时区
CURRENT_DATE ( ) //当前日期
CURRENT_TIME ( ) //当前时间
CURRENT_TIMESTAMP ( ) //当前时间戳

DATE (datetime) //返回datetime的日期部分
DATE_ADD (date2 , INTERVAL d_value d_type) //在date2中加上日期或时间
DATE_FORMAT (datetime ,FormatCodes) //使用formatcodes格式显示datetime
DATE_SUB (date2 , INTERVAL d_value d_type) //在date2上减去一个时间
DATEDIFF (date1 ,date2) //两个日期差
DAY (date) //返回日期的天
DAYNAME (date) //英文星期
DAYOFWEEK (date) //星期(1-7) ,1为星期天
DAYOFYEAR (date) //一年中的第几天
EXTRACT (interval_name FROM date) //从date中提取日期的指定部分
MAKEDATE (year ,day) //给出年及年中的第几天,生成日期串
MAKETIME (hour ,minute ,second) //生成时间串
MONTHNAME (date) //英文月份名
NOW ( ) //当前时间
SEC_TO_TIME (seconds) //秒数转成时间
STR_TO_DATE (string ,format) //字串转成时间,以format格式显示
TIMEDIFF (datetime1 ,datetime2) //两个时间差
TIME_TO_SEC (time) //时间转秒数]
WEEK (date_time [,start_of_week]) //第几周
YEAR (datetime) //年份
DAYOFMONTH(datetime) //月的第几天
HOUR(datetime) //小时
LAST_DAY(date) //date的月的最后日期
MICROSECOND(datetime) //微秒
MONTH(datetime) //月
MINUTE(datetime) //分

注：可用在INTERVAL中的类型：DAY ,DAY_HOUR ,DAY_MINUTE ,DAY_SECOND ,HOUR ,HOUR_MINUTE ,HOUR_SECOND ,MINUTE ,MINUTE_SECOND,MONTH ,SECOND ,YEAR
DECLARE variable_name [,variable_name...] datatype [DEFAULT value];
其中，datatype为mysql的数据类型，如:INT, FLOAT, DATE, VARCHAR(length)

例：

```
DECLARE l_int INT unsigned default 4000000;
DECLARE l_numeric NUMERIC(8,2) DEFAULT 9.95;
DECLARE l_date DATE DEFAULT '1999-12-31';
DECLARE l_datetime DATETIME DEFAULT '1999-12-31 23:59:59';
DECLARE l_varchar VARCHAR(255) DEFAULT 'This will not be padded';
```