

# HDPHP 框架 Beta 2014.5

开发代号 : 盾友

HDPHP 框架是 100% 自由的 , 您可以尽情享受和分享

为友爱而生

后盾网 人人做后盾

2012-2014

# HDPHP 框架

# 敏捷型 PHP 框架产品

作者：后盾网向军

HDPHP 框架为友爱而生

任何人都可以加入和扩展 HDPHP 框架，我们会在后盾网论坛公示你的信息，让更多人通过 HDPHP 框架了解到你 .....

后盾网 [www.houdunwang.com](http://www.houdunwang.com) HDPHP 官网 [www.hdphp.com](http://www.hdphp.com)

社区开发团队 等待你的加入 .....

# 目录

|                          |    |
|--------------------------|----|
| 1. HDPHP 框架只为让你快速开发..... | 17 |
| 1_1 运行环境 .....           | 17 |
| 2. 许可协议 .....            | 18 |
| 2_1 遵循协议 .....           | 18 |
| 2_2 许可方式 .....           | 18 |
| 2_3 赔偿 .....             | 19 |
| 2_4 无担保声明.....           | 19 |
| 2_5 责任限制 .....           | 19 |
| 3. 使用 HDPHP 框架.....      | 20 |
| 3_1 如何获得 HDPHP 框架 .....  | 20 |
| 3_2 获得帮助 .....           | 20 |
| 3_3 安装方法.....            | 20 |
| 4. HDPHP 框架特性.....       | 21 |
| 4_1 免费 .....             | 21 |
| 4_2 轻量级 .....            | 21 |
| 4_3 快速 .....             | 21 |
| 4_4 采用 MVC 设计模式 .....    | 21 |
| 4_5 生成干净的 URL.....       | 21 |
| 4_6 功能强大 .....           | 21 |
| 4_7 可扩展的 .....           | 22 |
| 4_8 对象关系映射 ( ORM ) ..... | 22 |
| 4_9 别名 .....             | 22 |

|                               |    |
|-------------------------------|----|
| 5. 目录结构 .....                 | 23 |
| 6. GET 私有变量 .....             | 24 |
| 7. 了解 HDPHP 框架的 GACM 概念 ..... | 25 |
| 8. 单入口文件 .....                | 26 |
| 8_1 入口文件常量介绍 .....            | 26 |
| 8_2 创建单入口文件.....              | 26 |
| 8_3 核心编译文件.....               | 27 |
| 9. 配置文件 .....                 | 28 |
| 9_1 框架核心配置项.....              | 28 |
| 9_2 应用组配置.....                | 28 |
| 9_3 模块配置项.....                | 28 |
| 10. 控制器 ( 模块 )Control .....   | 29 |
| 10_1 控制器文件命名规范.....           | 29 |
| 10_2 Empty 控制器 .....          | 29 |
| 10_3 __empty 空方法 .....        | 29 |
| 11. 自动加载 .....                | 31 |
| 12. import 导入类库 .....         | 32 |
| 13. 事件 .....                  | 33 |
| 14. URL 访问方式.....             | 36 |
| 14_1 pathinfo 访问 .....        | 36 |
| 14-1-1 受影响的配置项 .....          | 36 |
| 14_2 兼容模式访问方式 .....           | 37 |
| 14_3 普通访问方式 .....             | 37 |
| 15. 获得系统数据 Q 函数 .....         | 38 |
| 16. 判断请求类型.....               | 39 |
| 16_1 通过常量判断 .....             | 39 |
| 17. URL 路由器 .....             | 40 |

|   |           |
|---|-----------|
| 17_1 普通路由.....                                | 40        |
| 17_2 正则路由.....                                | 40        |
| 17_3 支持 QUERY 方式路由 .....                      | 41        |
| <b>18. url 伪静态.....</b>                       | <b>42</b> |
| 18_1 隐藏项目入口文件 .....                           | 42        |
| <b>19. Ajax 返回数据 .....</b>                    | <b>43</b> |
| <b>20. 模型 .....</b>                           | <b>44</b> |
| 20_1 基本模型.....                                | 44        |
| 20_2 扩展模型 ( K 函数 ) .....                      | 45        |
| <b>21. CURD 操作 .....</b>                      | <b>46</b> |
| 21_1 查询单条 find.....                           | 46        |
| 21_2 查询所有记录 all ( 别名 select 及 findall ) ..... | 46        |
| 21_3 table() 设置当前操作表 .....                    | 47        |
| 21_4 cache() 查询结果缓存 .....                     | 48        |
| 21-4-1 通过配置文件指定缓存时间 .....                     | 48        |
| 21-4-2 指定结果缓存时间 .....                         | 48        |
| 21_5 max 查找最大的值 .....                         | 49        |
| 21_6 min 查找最小的值.....                          | 49        |
| 21_7 avg 求平均值.....                            | 50        |
| 21_8 sum 求和.....                              | 50        |
| 21_9 count 统计操作 .....                         | 50        |
| 21_10 field 字段集.....                          | 51        |
| 21_11 limit 取部分数据 .....                       | 51        |
| 21_12 in 匹配多个值 .....                          | 52        |
| 21_13 where 查询条件 .....                        | 53        |

|  |    |
|--|----|
| 21_14 基于 ORM 方式的 WHERE .....             | 54 |
| 21_15 排序 ORDER .....                     | 54 |
| 21_16 getField 获得指定字段值.....              | 54 |
| 21_17 GROUP 分组操作 .....                   | 55 |
| 21_18 HAVING 分组条件.....                   | 55 |
| 21_19 add 添加数据 ( 别名 : insert ) .....     | 56 |
| 21_20 addAll 批量添加数据 .....                | 57 |
| 21_21 replace 添加数据.....                  | 58 |
| 21_22 update 更新数据 ( 别名 : save) .....     | 58 |
| 21_23 inc() 增加值.....                     | 59 |
| 21_24 dec() 减少值 .....                    | 59 |
| 21_25 delete() 删除数据 ( 别名 : del ) .....   | 59 |
| 21_26 getTableFields() 获得表所有字段集.....     | 60 |
| 21_27 fieldExists() 判断表中字段是否在存在.....     | 60 |
| 21_28 tableExists() 检测表是否存在 .....        | 61 |
| 21_29 获得数据库版本信息 getVersion .....         | 61 |
| 21_30 获得最后一条 SQL 语句 getLastSql() .....   | 61 |
| 21_31 获得受影响的行数 getAffectedRows() .....   | 61 |
| 21_32 获得所有 SQL 语句 getAllSql() .....      | 61 |
| 21_33 获得最后插入的主键 ID 值 getInsertId() ..... | 61 |
| 21_34 获得数据库或者表的大小 getSize() .....        | 62 |
| 21_35 获得表信息 getTableInfo().....          | 62 |
| 21_36 创建数据库 createDatabase().....        | 62 |
| 21_37 清空表 truncate() .....               | 62 |

|                               |           |
|-------------------------------|-----------|
| 21_38 一次修复多个数据表 repair()      | 62        |
| 21_39 一次优化多个数据表 optimize()    | 63        |
| 21_40 修改表名 rename() ;         | 63        |
| 21_41 删除表 dropTable()         | 63        |
| 21_42 设置自动提交 beginTrans()     | 63        |
| 21_43 事务回滚 rollback()         | 63        |
| 21_44 提交事务 commit()           | 64        |
| 21_45 执行 SQL 语句 runSql()      | 64        |
| <b>22. 触发器 (trigger)</b>      | <b>65</b> |
| 22_1 触发器方法                    | 65        |
| 22_2 开关触发器 trigger            | 65        |
| <b>23. 多表关联</b>               | <b>67</b> |
| 23_1 关联类型                     | 67        |
| <b>24. 关联模型 RelationModel</b> | <b>68</b> |
| 24-1-1 关联操作示例所需要的表            | 68        |
| 24_1 关联模型对象创建方法               | 68        |
| 24-1-1 使用 R() 函数产生多表关联对象      | 68        |
| 24-1-2 创建扩展模型 (建议使用的方式)       | 69        |
| 24_2 关联模型定义参数说明               | 69        |
| 24_3 关联模型中使用查询语句              | 70        |
| 24_4 join 方法                  | 70        |
| 24_5 关联查询                     | 71        |
| 24-5-1 HAS_ONE (包含一条记录)       | 71        |
| 24-5-2 只匹配关联表中的部分字段           | 71        |
| 24-5-3 定义扩展模型                 | 72        |

|                                   |           |
|-----------------------------------|-----------|
| 24-5-4 HAS_MANY ( 包含多条记录 ) .....  | 73        |
| 24-5-5 BELONGS_TO( 被包含 ) .....    | 74        |
| 24-5-6 MANY_TO_MANY 多对多关联操作 ..... | 75        |
| 24_6 关联添加.....                    | 77        |
| 24-6-1 一对一添加操作 HAS_ONE .....      | 77        |
| 24-6-2 一对多关联操作 HAS_MANY.....      | 78        |
| 24-6-3 BELONGS_TO 关联插入.....       | 79        |
| 24-6-4 MANY_TO_MANY .....         | 79        |
| 24_7 关联更新.....                    | 80        |
| 24-7-1 HAS_ONE.....               | 80        |
| 24-7-2 一对多 HAS_MANY .....         | 81        |
| 24-7-3 BELONGS_TO .....           | 82        |
| 24_8 关联删除.....                    | 83        |
| 24-8-1 HAS_ONE 关联删除 .....         | 83        |
| 24-8-2 HAS_MANY 关联删除操作 .....      | 83        |
| <b>25. 多表视图.....</b>              | <b>85</b> |
| 25_1 使用视图时的参数配置.....              | 85        |
| 25_2 控制器中定义视图 .....               | 85        |
| 25_3 扩展模型中定义视图规则.....             | 86        |
| 25_4 只使用某个视图定义 .....              | 87        |
| <b>26. Backup 数据库备份 .....</b>     | <b>88</b> |
| 26_1 备份数据库.....                   | 88        |
| 26_2 数据库还原.....                   | 89        |
| <b>27. 自动处理 create() .....</b>    | <b>90</b> |
| <b>28. 自动验证 .....</b>             | <b>91</b> |

|                            |     |
|----------------------------|-----|
| 28_1 验证规则设置语法 .....        | 91  |
| 28_2 验证示例演示 .....          | 92  |
| 28_3 Validate 验证类 .....    | 93  |
| 28_4 验证函数传递参数 .....        | 93  |
| 28_5 自定义验证方法 .....         | 94  |
| 28-5-1 自定义验证函数 .....       | 94  |
| 28-5-2 在模型中创建验证方法 .....    | 94  |
| 29. 自动完成 .....             | 95  |
| 29_1 自动完成语法 .....          | 95  |
| 30. 字段自动映射 .....           | 97  |
| 31. Token 令牌 .....         | 98  |
| 31_1 Token 配置项 .....       | 98  |
| 31_2 Token 验证操作 .....      | 98  |
| 32. 验证码 .....              | 99  |
| 33. 分页处理类 .....            | 101 |
| 33_1 构造函数 .....            | 101 |
| 33_2 获得分页项数组 .....         | 101 |
| 33_3 分页样式 .....            | 102 |
| 33_4 SQL 语句所需要 LIMIT ..... | 102 |
| 33_5 分页示例演示 .....          | 102 |
| 33_6 自定义分页 Url .....       | 103 |
| 34. 图像处理类 .....            | 104 |
| 34_1 图像缩略图处理 .....         | 104 |
| 34_2 图像水印处理 .....          | 105 |
| 35. 上传类 .....              | 107 |
| 36. URI 类 .....            | 109 |

|                              |     |
|------------------------------|-----|
| 37. Email 邮件处理 .....         | 110 |
| 38. 模板视图 .....               | 112 |
| 38_1 基础知识 .....              | 112 |
| 38_2 定义模版 .....              | 112 |
| 38_3 assign 向视图层分配内容 .....   | 113 |
| 38_4 display 显示内容 .....      | 113 |
| 38_5 fetch() 获得模板解析数据 .....  | 114 |
| 38_6 isCache() 缓存是否失效 .....  | 115 |
| 38_7 读取系统变量 .....            | 115 |
| 38_8 变量调节器 .....             | 116 |
| 38-8-1 使用任意 PHP 函数 .....     | 116 |
| 38-8-2 变量调节器链式操作 .....       | 116 |
| 38-8-3 hd_substr 截取字符串 ..... | 116 |
| 38-8-4 hd_date 日期处理函数 .....  | 116 |
| 38-8-5 模板中使用函数 .....         | 117 |
| 38-8-6 模板中使用 U() 方法 .....    | 117 |
| 38-8-7 default 默认值 .....     | 118 |
| 38_9 系统模版标签使用 .....          | 118 |
| 38-9-1 less 标签 .....         | 118 |
| 38-9-2 jquery 标签 .....       | 118 |
| 38-9-3 jsconst 标签 .....      | 119 |
| 38-9-4 hdjs 标签 .....         | 119 |
| 38-9-5 validate 自动验证标签 ..... | 119 |
| 38-9-6 bootstrap 标签 .....    | 119 |
| 38-9-7 slide 轮换版标签 .....     | 120 |

|  |            |
|--|------------|
| 38-9-8 cal 日历标签 .....                                | 120        |
| 38-9-9 foreach 标签 .....                              | 120        |
| 38-9-10 list 标签 .....                                | 120        |
| 38-9-11 if 标签 .....                                  | 122        |
| 38-9-12 标签 .....                                     | 122        |
| 38-9-13 empty 标签 .....                               | 123        |
| 38-9-14 php 标签 .....                                 | 124        |
| 38-9-15 define .....                                 | 124        |
| 38-9-16 js 标签 .....                                  | 124        |
| 38-9-17 css 标签 .....                                 | 124        |
| 38-9-18 load 标签 (别名 include) .....                   | 125        |
| <b>39. 自定义模版标签 .....</b>                             | <b>126</b> |
| <b>39_1 创建自定义标签的类文件 .....</b>                        | <b>126</b> |
| <b>39_2 指定标签类文件 .....</b>                            | <b>126</b> |
| <b>39_3 配置文件中的设置 .....</b>                           | <b>126</b> |
| <b>39_4 创建自定义标签内容 .....</b>                          | <b>126</b> |
| <b>39-4-1 块标签定义 .....</b>                            | <b>126</b> |
| <b>39-4-2 独立标签定义 .....</b>                           | <b>127</b> |
| <b>39_5 HDPHP 框架整合 SMARTY 的使用 .....</b>              | <b>128</b> |
| <b>40. Data 数据处理类 .....</b>                          | <b>130</b> |
| <b>40_1 Data::tree() 获得树状数据 (建议使用) .....</b>         | <b>130</b> |
| <b>40_2 Data::channelList 获得目录列表 .....</b>           | <b>131</b> |
| <b>40_3 Data::channelLevel 获得多级目录列表 (多维数组) .....</b> | <b>132</b> |
| <b>40_4 Data::parentChannel() 获得所有父级栏目 .....</b>     | <b>133</b> |
| <b>40_5 Data::isChild() 判断是否为子栏目 .....</b>           | <b>133</b> |

|   |            |
|---|------------|
| 40_6 Data::descarte() 迪卡尔乘积 .....             | 134        |
| <b>41. 目录操作类.....</b>                         | <b>135</b> |
| 41_1 Dir::getExt() 获得文件扩展名 .....              | 135        |
| 41_2 Dir::tree() 遍历目录内容.....                  | 135        |
| 41_3 Dir::treeDir() 只显示目录树 .....              | 135        |
| 41_4 Dir::del() 删除目录与文件.....                  | 135        |
| 41_5 create() 创建目录.....                       | 136        |
| 41_6 copy() 复制目录内容 .....                      | 136        |
| 41_7 Dir::safeFile() 创建目录安全文件 .....           | 136        |
| <b>42. string 字符串处理类 .....</b>                | <b>137</b> |
| 42_1 String::toSemiangle 全角转半角 .....          | 137        |
| 42_2 string::pinyin 中文转拼音.....                | 137        |
| 42_3 string::splitWord() 中文分词 .....           | 138        |
| 42_4 String::removePunctuation() 去除标点符号 ..... | 138        |
| <b>43. 缓存控制.....</b>                          | <b>139</b> |
| 43_1 影响缓存的配置项 .....                           | 139        |
| 43_2 Memcache 缓存设置 .....                      | 139        |
| 43_3 Redis 缓存设置 .....                         | 140        |
| 43_4 CACHE 缓存类 .....                          | 140        |
| 43_5 设置缓存.....                                | 142        |
| 43_6 删除缓存.....                                | 143        |
| 43_7 删除缓存.....                                | 143        |
| 43_8 获得与设置配置 .....                            | 143        |
| 43_9 缓存队列.....                                | 143        |
| 43_10 以 Memcache 操作缓存 .....                   | 144        |

|   |            |
|---|------------|
| 43_11 以 Redis 操作缓存 .....                | 144        |
| 43_12 S() 缓存函数 .....                    | 145        |
| 43_13 F() 快速文件缓存.....                   | 145        |
| <b>44. 多语言支持.....</b>                   | <b>147</b> |
| <b>45. 生成 HTML 静态文件.....</b>            | <b>148</b> |
| 45_1 生成 HTML 文件.....                    | 148        |
| 45_2 删除 HTML 文件.....                    | 150        |
| 45-2-1 删除目录下的所有 html 文件 .....           | 150        |
| 45-2-2 删除指定的 HTML 文件 .....              | 150        |
| <b>46. cart 购物车类 .....</b>              | <b>151</b> |
| 46_1 添加购物车 cart::add() .....            | 151        |
| 46_2 更新购物车 Cart::update()               | 152        |
| 46_3 获得购物车商品数据 Cart::getGoods()         | 152        |
| 46_4 获得购物车所有数据包 Cart::getAllData()      | 152        |
| 46_5 清空购物车中的所有商品 Cart::delAll()         | 152        |
| 46_6 获得商品总价格 Cart::getTotalPrice()      | 152        |
| 46_7 统计购物车中的商品数量 Cart::getTotalNums()   | 153        |
| 46_8 获得定单号.....                         | 153        |
| <b>47. RBAC 基于角色的权限控制 .....</b>         | <b>154</b> |
| 47_1 影响 RBAC 的配置项 .....                 | 154        |
| 47_2 Rbac::IsLogin()                    | 154        |
| 47_3 Rbac::login() 用户登录 .....           | 155        |
| 47_4 Rbac::checkAccess()                | 156        |
| 47_5 Rbac::getNodeList() 获得所有节点列表 ..... | 156        |
| <b>48. session 操作.....</b>              | <b>157</b> |

|                                       |     |
|---------------------------------------|-----|
| 48_1 SESSION 配置项 .....                | 157 |
| 48_2 自定义 SESSION 参数 .....             | 157 |
| 49. session() 函数 .....                | 158 |
| 50. SESSION 引擎 .....                  | 159 |
| 50_1 session 的 mysql 处理机制 .....       | 159 |
| 50_2 session 的 Memcache 处理机制 .....    | 159 |
| 50_3 session 的 Redis 处理机制 .....       | 160 |
| 51. cookie 操作 .....                   | 161 |
| 52. Xml 操作类 .....                     | 162 |
| 52_1 Xml::create() 创建 xml 文件 .....    | 162 |
| 52_2 Xml::toArray() .....             | 162 |
| 53. IP 处理类 .....                      | 164 |
| 53_1 Ip::area() 获得 IP 来源地理位置 .....    | 164 |
| 53_2 Ip::getClientIp() 获得客户端 IP ..... | 164 |
| 54. 错误异常处理 .....                      | 165 |
| 54_1 开启调试模式 .....                     | 165 |
| 54_2 自定义错误内容显示方式 .....                | 165 |
| 55. 日志处理 .....                        | 166 |
| 56. 核心函数库 .....                       | 167 |
| 56_1 C() 载入或设置配置 .....                | 167 |
| 56_2 O() 函数 .....                     | 167 |
| 56_3 control() 实现化控制器 .....           | 168 |
| 56_4 tag() 调用标签函数 .....               | 168 |
| 56_5 U() 生成 url .....                 | 168 |
| 56_6 load() 文件加载 .....                | 171 |
| 56_7 404 错误处理 .....                   | 171 |

|   |     |
|---|-----|
| 56_8 addslashes_d() 转义字符串 .....                   | 171 |
| 56_9 array_change_key_case_d() 改变数组键名大小写 .....    | 172 |
| 56_10 array_change_value_case() 数组的值大小写转换.....    | 173 |
| 56_11 array_defined() 将数组转为常量.....                | 173 |
| 56_12 array_key_exists_d() 不区分大小写检测键名是否存在 .....   | 173 |
| 56_13 array_to_String() 将数组转为字符串表示形式.....         | 173 |
| 56_14 control() 实例化控制器对象.....                     | 173 |
| 56_15 A() 执行控制器中的方法 ( 支持分组 ) .....                | 174 |
| 56_16 encrypt() 加密处理 .....                        | 174 |
| 56_17 decrypt() 解密方法 .....                        | 174 |
| 56_18 dir_create() 函数 .....                       | 174 |
| 56_19 p() 别名 ( show,dump).....                    | 175 |
| 56_20 halt() 输出错误信息.....                          | 175 |
| 56_21 firephp() 调试插件 .....                        | 175 |
| 56_22 extension_exists() PHP 扩展模块是否存在.....        | 175 |
| 56_23 file_exists_case() 区分大小写的文件判断 .....         | 175 |
| 56_24 getDefines 获得常量 .....                       | 175 |
| 56_25 getSize() 根据大小返回标准单位 KB MB GB 等.....        | 176 |
| 56_26 go() 跳转到指定 url.....                         | 176 |
| 56_27 browser_info() 获得浏览器版本 .....                | 176 |
| 56_28 image_type_to_extension() 根据类型获得图像扩展名 ..... | 177 |
| 56_29 ip_get_client() 显示客户端 .....                 | 177 |
| 56_30 is_ssl() 是否为 SSL 协议 .....                   | 177 |
| 56_31 json_encode() 对变量进行 JSON 编码.....            | 177 |

|   |            |
|---|------------|
| 56_32 json_decode 对 JSON 格式的字符串进行编码 .....     | 177        |
| 56_33 load() 载入文件.....                        | 177        |
| 56_34 md5_d 方法.....                           | 177        |
| 56_35 mobile_area() 电话号码来源 .....              | 177        |
| 56_36 php_merge() 合并 php 文件 .....             | 177        |
| 56_37 print_const(\$view=true) 打印所有常量.....    | 178        |
| 56_38 rand_str 获得随机字符串.....                   | 178        |
| 56_39 set_http_state() 设置 HTTP 状态信息.....      | 178        |
| 56_40 compress() 压缩 PHP 代码 .....              | 178        |
| 56_41 data_format() 数据安全处理 .....              | 178        |
| 56_42 stripslashes_d() 去除转义字符.....            | 179        |
| 56_43 throw_exception() 抛出异常.....             | 179        |
| 56_44 url_param_remove() 移除 URL 中 GET 参数..... | 179        |
| 56_45 date_before() 获得几秒、几分、几年前.....          | 180        |
| 56_46 get_uuid() 获得唯一值 uuid .....             | 180        |
| <b>57. 核心常量 .....</b>                         | <b>181</b> |
| 57_1 基本常量.....                                | 181        |
| 57_2 目录与文件常量 .....                            | 181        |
| 57_3 URL 常量 .....                             | 182        |
| <b>58. 编辑器使用方法 .....</b>                      | <b>184</b> |
| 58_1 uditor 编辑器使用 .....                       | 185        |
| 58_2 kditor 编辑器使用 .....                       | 189        |
| <b>59. AJAX 多文件上传组件 .....</b>                 | <b>193</b> |
| 59_1 使用上传组件 .....                             | 193        |
| 59_2 上传成功后的回调函数 hd_upload() .....             | 196        |

|                               |            |
|-------------------------------|------------|
| 59_3 上传时传递 POST 数据 .....      | 196        |
| 59_4 自定义上传处理 PHP 脚本.....      | 196        |
| 59_5 编辑图片.....                | 197        |
| 59_6 删 除图片 .....              | 198        |
| <b>60. 局部放大镜插件 zoom .....</b> | <b>199</b> |
| <b>61. 安全建议 .....</b>         | <b>201</b> |
| 61_1 标准化的重要性和意义.....          | 201        |
| 61_2 代码标记.....                | 202        |
| 61_3 文件目录规范 .....             | 202        |
| 61_4 类命名规范.....               | 202        |
| 61_5 函数规范.....                | 203        |
| 61_6 常量规范.....                | 203        |
| 61_7 配置项规范.....               | 203        |
| 61_8 语言包规范.....               | 203        |
| 61_9 变量规范.....                | 203        |
| 61_10 缩进规范 .....              | 204        |
| 61_11 注释规范 .....              | 204        |
| 61_12 兼容性 .....               | 204        |
| 61_13 代码重用 .....              | 205        |
| 61_14 引号 .....                | 205        |
| 61_15 数据库规范 .....             | 206        |
| 61-15-1 字段结构 .....            | 206        |
| 61-15-2 SQL 语句 .....          | 206        |
| 61-15-3 定长与变长表 .....          | 206        |
| 61-15-4 运算与检索 .....           | 207        |

|                         |     |
|-------------------------|-----|
| 61-15-5 结构优化与索引优化 ..... | 207 |
| 61-15-6 查询优化 .....      | 208 |
| 61_16 版本控制 .....        | 209 |
| 61_17 总结.....           | 210 |

# 1. HDPHP 框架只让你快速开发

后盾 HDPHP 框架是一个为用 PHP 程序语言编写网络应用程序的人员提供的软件包。提供强大的、完整的类库包，满足开发中的项目需求，HDPHP 框架可以将需要完成的任务代码量最小化，大大提高项目开发效率与质量，当然使用是非常简便、快捷的。HDPHP 框架产品高效的核心编译处理机制让系统运行更快，提供丰富的的错误解决方案，让修正代码错误变得更快。

后盾 HDPHP 框架产品做为优秀的框架产品，在系统性能上做的大量的优化处理，只为让程序员使用 HDPHP 框架强悍的功能，用最短的时间完成项目的开发。为了便于快速进行项目应用开发框架提供了相应的前端功能组件使程序员从繁琐的组件调试中解脱出来。

HDPHP 框架定位：简单快速学习，简单快速开发网站。

## 1\_1 运行环境

1. PHP 版本要求 5.16 或更高版本
2. 可以运行在 UNIX、LINUX、WINDOWS、苹果 Macintosh 电脑等平台

windows 环境下推荐使用 wamp 集成化环境进行学习使用

你可以通过后盾网获得免费开发视频教程，请登录 [bbs.houdunwang.com](http://bbs.houdunwang.com) 下载

## 2. 许可协议

### 2\_1 遵循协议

HDPHP 遵循 Apache2 开源协议发布，并提供免费使用。

版权所有 Copyright © 2011-2012 by HDPHP (<http://hdphp.com>)

All rights reserved.

HDPHP 商标和著作权所有者为北京后盾计算机技术培训有限责任公司。

Apache Licence 是著名的非盈利开源组织 Apache 采用的协议。该协议和 BSD 类似，同样鼓励代码共享和尊重原作者的著作权，同样允许代码修改，再发布（作为开源或商业软件）。需要满足的条件也和 BSD 类似：

1. 需要给代码的用户一份 Apache Licence
2. 如果你修改了代码，需要在被修改的文件中说明。
3. 在延伸的代码中（修改和有源代码衍生的代码中）需要带有原来代码中的协议，商标，专利声明和其他原来作者规定需要包含的说明。

4. 如果再发布的产品中包含一个 Notice 文件，则在 Notice 文件中需要带有 Apache Licence。你可以在 Notice 中增加自己的许可，但不可以表现为对 Apache Licence 构成更改。

Apache Licence 也是对商业应用友好的许可。使用者也可以在需要的时候修改代码来满足需要并作为开源或商业产品发布 / 销售。

具体协议参考：<http://www.apache.org/licenses/LICENSE-2.0.html>

### 2\_2 许可方式

只要符合以下条件，你将被允许使用、复制、修改以及分发本软件和它相关的文档，包括你可以修改或者不修改地用于任何目的：

这个许可协议的一份拷贝必须包含在分发的软件中。

再分发源代码时必须在所有源代码文件中保留上方的版权提醒。

任何修改过的文件必须加上对原始代码修改的注释以及修改者名称。

任何由本软件衍生的产品必须在它们的文档以及 / 或者随分发提供的物品中表明它们来源于后盾网。

除非事先得到后盾网的书面许可，否则任何本软件的衍生产品都不得叫做 'HDPHP 框架'，也不得在名称中出现 'HDPHP'。

## **2\_3 赔偿**

你必须认可对任何因使用或者误用本软件或者违反任何本许可协议条款所产生的直接、间接、附带的或相应的第三者索赔、诉讼费用承担赔偿，皆与本软件的作者和任何贡献者无关。

## **2\_4 无担保声明**

该软件仅以实际效果为准，不做任何明示或暗示的保证，包括但不限于保证的质量，性能，不侵权，适销性或适用于特定用途。

## **2\_5 责任限制**

你将承担所有安装和使用本软件的风险。

# 3. 使用 HDPHP 框架

通过这部份的学习，掌握和使用 HD 敏捷框架产品的灵活使用。你会发现使用 HDPHP 框架开发是如此的简单、方便。

## 3\_1 如何获得 HDPHP 框架

获得 HDPHP 的方式很简单只要登录 <http://www.hdphp.com> 下载即可，你完全可以免费使用，在法律允许情况下可以用于任何用途，包括任何商业产品的开发。

## 3\_2 获得帮助

获得帮助有以下几种方式：

1. 通过后盾网论坛获得 <http://bbs.houdunwang.com>
2. 通过官方邮件获得帮助 [houdunwangxj@gmail.com](mailto:houdunwangxj@gmail.com)

通过论坛解决问题的途径是最快的，因为你的问题也有可能别人已经提出过，所以你可以先通过论坛获得帮助，这样效果最快。

## 3\_3 安装方法

将 HDPHP 框架下载后放到可以运行 PHP 的环境中，配置好主入口文件，你就可以开始免费使用强大的、敏捷的 HDPHP 开源框架产品。

# 4. HDPHP 框架特性

## 4\_1 免费

HDPHP 是经过 Apache/BSD-style 开源许可授权的，只要你愿意就可以免费使用它。同时后盾网 [www.houdunwang.com](http://www.houdunwang.com) 提供强大的技术支持，记住这一切都是免费的。

## 4\_2 轻量级

真正的轻量级。我们的核心系统只需要一些非常小的库，这与那些需要更多资源的框架完全相反。额外的库文件只在请求的时候加载，依需求而定，所以核心系统是非常快而且轻的。

## 4\_3 快速

速度非常快。HDPHP 框架提供多项优化策略完善的功能处理类，你要找到一个比 HDPHP 表现更优的框架应该很难吧。

## 4\_4 采用 MVC 设计模式

HDPHP 使用了模型 ( Model ) - 视图 ( View ) - 控制器 ( Controllers ) 的方法，这样可以更好地使表现层和逻辑层分离。这对项目的模板设计者来说是非常有用的，它最小化了模板中的程序代码量。

使用 MVC 的目的是将 M 和 V 的实现代码分离，从而使同一个程序可以使用不同的表现形式。比如一批统计数据你可以分别用柱状图、饼图来表示。C 存在的目的则是确保 M 和 V 的同步，一旦 M 改变，V 应该同步更新。

## 4\_5 生成干净的 URL

HDPHP 生成的 URL 非常干净而且是对搜索引擎友好化的。不同于标准的 '字符串查询' 方法，HDPHP 使用了基于段的 PATHINFO 方法，同时提供强大的 URL 路由功能。

## 4\_6 功能强大

HDPHP 框架拥有全范围的类库，可以完成大多数通常需要的网络开发任务，包括：读取数据库、发送电子邮件、数据确认、保存 session、对图片的操作，以及支持 XML-RPC 数据传输等。

## 4\_7 可扩展的

这个系统可以非常简单的通过自定义类库、辅助函数来进行扩展，或者也可以通过扩展类、系统钩子来实现。

## 4\_8 对象关系映射 ( ORM )

对象 - 关系映射 ( Object/Relation Mapping , 简称 ORM ) , 是随着面向对象的软件开发方法发展而产生的。面向对象的开发方法是当今企业级应用开发环境中的主流开发方法，关系数据库是企业级应用环境中永久存放数据的主流数据存储系统。对象和关系数据是业务实体的两种表现形式，业务实体在内存中表现为对象，在数据库中表现为关系数据。内存中的对象之间存在关联和继承关系，而在数据库中，关系数据无法直接表达多对多关联和继承关系。因此，对象 - 关系映射 (ORM) 系统一般以中间件的形式存在，主要实现程序对象到关系数据库数据的映射。

面向对象是从软件工程基本原则 ( 如耦合、聚合、封装 ) 的基础上发展起来的，而关系数据库则是从数学理论发展而来的，两套理论存在显著的区别。为了解决这个不匹配的现象，对象关系映射技术应运而生。

## 4\_9 别名

别名就是指这个函数的另外一个使用方法，如 find 用于查找单条记录，他拥有一个别名是 one , find 与 one 使用方法一模一样只是不同的名称而已。HDPHP 框架为更大的适用性引用别名概念，支持函数或类方法多个不同名称。

## 5. 目录结构

| 方法名                | 说明          |
|--------------------|-------------|
| hdphp.php          | 框架入口文件      |
| Extend             | 扩展目录        |
| Extend/Org         | 扩展应用包       |
| Extend/Tool        | 集成工具包       |
| Data               | 静态数据目录      |
| Config             | 基本配置文件      |
| Lib                | 核心核心目录      |
| Lib/Core           | 核心文件        |
| Lib/Driver/Cache   | 缓存驱动        |
| Lib/Driver/Db      | 数据驱动库       |
| Lib/Driver/Model   | 数据模型库       |
| Lib/Driver/Session | Session 驱动库 |
| Lib/View           | 视图驱动库       |
| Lib/Functions      | 函数库         |
| Lib/Language       | 语言包         |
| Lib/Tpl            | 框架模板目录      |

## 6. GET 私有变量

GET 私有变量指 HDPHP 占有的 `$_GET` 值，开发人员请不要使用这些变量，否则会产生异常。

| 方法名                         | 说明     |
|-----------------------------|--------|
| <code>\$_GET['g']</code>    | 应用组名   |
| <code>\$_GET['a']</code>    | 应用名    |
| <code>\$_GET['c']</code>    | 控制器名   |
| <code>\$_GET['m']</code>    | 方法名    |
| <code>\$_GET['page']</code> | 分页页码变量 |

# 7. 了解 HDPHP 框架的 GACM 概念

GACM 定义是 HDPHP 框架的特有定义，如果你以前使用过别的产品，可能会感觉比较陌生，不过 GACM 的操作方式，会让网站的规划更有条理，扩展性更强，下面我们来了解一下 GACM 的具体概念。

## G Group 应用组

为了进行大型项目的开发 HDPHP 框架支持应用组设置。应用组配置就是将多个应用分组管理。

## A Application 应用

我们可以把网站中的后台 admin 当成一个应用，前台会员中心 member 当成一个应用，如果你的网上还有博客、论坛、商城那么每一个当成一个应用，有了应用的概念，网站的扩展就会变的很轻松了。

## C Control 模块控制器

比如说网站的后台应用 admin 中有用户管理模块、权限分配模块、用户登录模块、友情链接管理模块等等，这些都是模块控制器，在 HDPHP 框架中即以 C 表示。

## M Method 操作方法

比如说文章管理模块包括文章的添加、修改、删除等等操作，用户登录模块包括验证码显示，用户验证等操作，在 HDPHP 框架中这些就是 M

通过以上的说明，相信你已经有了一个初步的认识，应用 Application 包含很多 Control 模块控制器，同时 Control 模块控制器中也包含很多 Method 操作方法，有了上面的根据，我们开始定义单入口文件。

注：HDPHP 框架中的控制器文件放置在应用目录下的 control 目录中

# 8. 单入口文件

HDPHP 框架采用单入口文件访问策略，无论从安全性还是方法调用及文件加载方面都带来了很高的便捷性，所以我们有必要学习 HDPHP 框架的单入口配置，这是使用 HDPHP 框架的前提！HDPHP 框架单入口访问方式非常灵活，不过先不要把每种单入口方式都掌握，这样可能会比较吃力，所以你先熟悉一种 HDPHP 框架单入口的方式，再来学习或使用其他几种单入口定义方式。

## 8\_1 入口文件常量介绍

在设置单入口文件时可以通过设置一些常量决定项目应用的部署目录结构。

|            |          |
|------------|----------|
| APP_PATH   | 应用所在的目录  |
| GROUP_PATH | 应用组所在的目录 |
| DIR_SAFE   | 创建目录安全文件 |
| DEBUG      | 调试模式     |
| TEMP_PATH  | Temp 目录  |
| TEMP_NAME  | 编译文件名    |

## 8\_2 创建单入口文件

### 最简单设置

```
1 <?php  
2 require './hdphp/hdphp.php';// 框架主文件  
3 ?>
```

一个单入口文件只影响一个应用，比如说 <http://localhost/index.php>。

php?c=User&m=add 表示执行 Admin 应用目录下的 UserControl.class.php 控制器文件的 add 方法

### 设置应用目录

```
1 <?php  
2 define('APP_PATH','Admin/');  
3 require './hdphp/hdphp.php'; // 包含 HDPHP 框架主文件  
4 ?>
```

注：定义 APP\_PATH 必须以斜杠结尾

在当前目录创建 Admin 应用目录

### 应用组模式

```
1 <?php
2 define('GROUP_PATH','Weibo');// 应用组目录
3 require './hdphp/hdphp.php'; // 包含 HDPHP 框架主文件
4 ?>
```

将所有应用归类到一个文件夹下进行操作。比如以 `http://localhost/index.php?a=admin&c=user&m=add` 访问，表示访问 Admin 应用中的 user 控制器模块中的 add 方法，这种方式与第一种方式的访问方式是一样的，不同点就是在创建目录的方式，第一种每个应用独立目录，这种方式会将所有应用都放置在一个应用组目录中。

## 8\_3 核心编译文件

如果需要对 HD PHP 框架核心代码进行修改，同时需要即时得到修改后的结果，才有必要阅读本节内容，如果没有修改 HD PHP 框架核心代码，根本不需要关注这点。

HD PHP 框架在运行时，为了提供更快的速度在细节上经过众多优化，其中包括核心文件编译机制，默認為开启核心编译，当开启编译时系统会将所有核心文件进行编译操作，在临时目录 Temp 下生成 `~boot.php` 运行文件，以后就执行这个编译后的文件，执行速度更快。

不生成核心编译文件，只需要在单入口文件中添加 `define('DEBUG',FALSE);` 就可以，这样系统就不会生成核心编译文件。

不生成编译文件的好处时，系统会执行创建应用目录等初始化操作。

### 示例

```
1 <?php
2 define('TEMP_NAME','~runtime.php');// 核心编译文件
3 require './hdphp/hdphp.php'; // 框架主文件
4 ?>
```

# 9. 配置文件

HDPHP 框架采用多级配置设置，如果有相同配置项，高级别的配置优先级更高。级别由高至低排序如下：

- 1 应用目录中 Config 目录中的 config.php 配置文件
- 2 应用组目录中的 Config 目录中的 config.php 配置文件
- 3 HDPHP 框架目录中的基础配置文件 Config/config.php

## 9\_1 框架核心配置项

框架核心配置项位于框架的 Config 目录中，程序员不用修改框架核心配置文件，只需要定义应用组或应用中的配置文件即可。

## 9\_2 应用组配置

应用组配置优先级高于框架核心配置项，应用组配置会作用于所有应用。

## 9\_3 模块配置项

应用配置文件位于应用的 Config 目录中，优先级高组应用组配置文件。

# 10. 控制器(模块)Control

控制器是视图与模型间的中间层，起到中间调度作用。控制器收到视图反馈信息，决定是否交于模型处理，之后将结果返回视图层，完成一个交互流程，所以控制器是非常重要的一个环节也是程序员精力重点投放的地方。

## 10\_1 控制器文件命名规范

控制器文件名由控制器名 + 后缀 + .class.php 构成，例如 UserControl 控制器就要创建成 UserControl.class.php

控制器后缀需要修改配置项 CONTROL\_FIX

## 10\_2 Empty 控制器

访问的控制器不存在时，如果应用中存在 EmptyControl.class.php 控制器，则调用 EmptyControl 控制器。

比如我们希望通过执行 `http://localhost/index.php/用户名` 这样的方式调用用户信息，则可以借助 EmptyControl 控制器，示例如下：

```
1 class EmptyControl extends Control{
2     function index(){
3         $user = $_GET['c'];// 此时控制器 GET 变量 C 即为用户名
4         echo '用户 :'.$user;
5     }
6 }
```

## 10\_3 \_\_empty 空方法

HDPHP 框架提供了魔术方法 \_\_empty 空方法，当访问的方法不存在时，系统会自动调用 \_\_empty 方法，示例如下：

```
1 class UserControl extends Control{
2     function __empty(){
3         echo '用户为 '.$_GET['m'];
4     }
5 }
```

如果访问的 url 为 `http://localhost/index.php/user/hdphp`，因为 UserControl 控制

器中不存在方法 `hdphp` 则系统会自动调用 `_empty()` 方法，那么我们可以利用这一特殊性，完成一些特殊的业务处理。

# 11. 自动加载

HDPHP 提供了很好的扩展特性，开发人员可以通过自动加载机制加载扩展的文件。

**注意：如果文件加载失败，请删除 Temp 目录或开启一下 DEBUG 调试模式**

自动加载实现通过修改配置项 AUTO\_LOAD\_FILE 来设置，示例如下：

```
'AUTO_LOAD_FILE' => array('./Hdcms/Lib/functions.php','functions.php')
```

**说明：**

必须添加扩展名 .php

系统首先会加载应用 Lib 目录下的文件，如果不存在尝试加载应用组 Lib 目录下的文件

如果包含 / 表示使用绝对路径

以上配置会加载 Hdcms 应用 Lib 目录下的 functions.php 文件与当前应用 Lib 目录下的 functions.php 文件

# 12. import 导入类库

HDPHP 框架提供了功能完善的类库，但不可能满足所有业务需求，为此，HDPHP 框架提供了方便的类库扩展机制 import。

语法 : `import($class=null,$base=null,$ext='.class.php')`

导入当前应用 `Lib/Html.class.php`

```
import('Lib.Html')
```

加载当前应用 `Class/Html.class.php`

```
import('@.Class.Html')
```

加载当前应用组 `Hdcms/Index/Config/Tag.class.php`

```
import('@@.Hdcms.Index.Config.Tag')
```

加载 Member 应用的 `HtmlControl.class.php`

```
import('Member.Control.HtmlControl')
```

导入框架 Extend/Tool 目录下 `Html.class.php` 类

```
import('HDPHP.Extend.Tool.Html')
```

导入网站根目录下 `/Model/RbacModel.class.php`

```
import('RbacModel','Model');
```

导入当前应用 Lib 目录下的 `Auth.inc.php` 类

```
import('@.Lib.Auth',NULL,'.inc.php')
```

# 13. 事件

HDPHP 框架提供了事件处理机制，其实意思与 JavaScript 中的事件相似，就是在某一个时机自动执行某些功能。

## 配置文件

1. 事件的配置文件为 event.php( 文件名小写 )
2. 配置文件可以定义在应用或应用组的 Config 目录中

## 事件处理程序

1. 一个事件可以指定多个事件处理类
2. 事件处理程序可以定义在应用或应用组 Event 目录下
3. 事件处理程序的后缀为 Event.class.php

## HDPHP 框架提供了一些常用的系统事件

| 事件名称          | 说明    |
|---------------|-------|
| APP_START     | 应用运行前 |
| APP_END       | 应用运行后 |
| CONTROL_START | 模块执行前 |
| CONTROL_END   | 模块执行后 |

## 示例一 ( 自动执行系统事件 )

定义配置文件 event.php 如下

```
1 <?php
2 if (!defined('HDPHP_PATH'))exit('No direct script access allowed');
3 // 在模块运行前，执行事件 header 与事件 login
4 return array(
5     'APP_START'=>array('Header','Login')
6 );
7 ?>
```

在应用目录 Event 目录下创建事件文件 LoginEvent.class.php 与 HeaderEvent.class.php 文件，下面以 HeaderEvent.class.php 为例

```
1 <?php
2 class HeaderEvent extends Event{
3     public function run(&$options){
4         header('Content-type:text/html;charset=utf-8');
5         echo '设置字符集';
6         $this->options['status']=1;
7     }
8 }
9 ?>
```

因为设置了应用 CONTROL\_START 事件，所以当前应用所有模块都会执行这个事件。

## 示例二（自定义事件）

可以通过使用 event() 函数执行事件

示例一是自动执行的系统定义的一些事件，那么开发者定义的事件可以使用 event() 函数来调用执行。

在应用 config 中定义配置文件 event.php，内容如下

```
1 <?php
2 // 定义事件 user_valid
3 return array(
4     'USER_VALID'=>array('Access','Info')
5 );
6 ?>
```

在应用目录 Event 目录下创建事件文件 AccessEvent.class.php 与 InfoEvent.class.php 文件，即完成了事件的定义，模块中调用方式如下：

```
1 <?php
2 class IndexControl extends Control
3 {
4     function index()
5     {
6         // 执行事件 user_valid ( 即执行上面定义的 2 个事件处理程序 )
7         $options=array();// 事件 run 方法的参数
8         event('USER_VALID',$options);
9     }
}
```

10 ?>

### 示例三（执行一个事件处理程序）

在应用 Event 目录下创建事件处理类 LoginEvent.class.php

```
1 <?php
2 class LoginEvent extends Event{
3     public $options=array('is_login'=>false);
4     public function run(&$options){
5         if(session('uid')){
6             $this->options['is_login']=true;
7         }else{
8             header('Content-type:text/html;charset=utf-8');
9             go(__APP__.'/login',3,'您还没有登录,3秒后返回登录页');
10        }
11    }
12 }
13 ?>
```

IndexControl.class.php 控制器代码

```
1 <?php
2 class IndexControl extends Control{
3 {
4     function index(){
5     }
6     E('Login');// 登录验证事件
7 }
8 }
9 ?>
```

# 14. URL 访问方式

HDPHP 框架支持丰富的 URL 访问方式，可以根据应用场景选择合适的 url 访问规则，下面我们来了解一下各种 url 规则的使用。

1. 如果以应用组形式访问，GET 参数中的 a 将作为传递应用名使用
2. 控制器命名必须使用 pascal 命名规范，即单词首字母大写，如 UserControl.class.php，在 URL 中使用 user 访问即可
3. 如果控制器名称中含有多个单词如 HdphpUserControl.class.php，在 URL 中使用 hdphp\_user 形式访问

## 14\_1 pathinfo 访问

pathinfo 访问规则设置非常简单，pathinfo 模式提供了更佳的 seo 优化，同时 url 地址更精巧。

可以通过 HDPHP 框架的 pathinfo 模式产生伪静态的 url 地址

### 14-1-1 受影响的配置项

|    |                 |           |                                   |
|----|-----------------|-----------|-----------------------------------|
| 1  | 'HTTPS'         | => FALSE, | // 基于 https 协议                    |
| 2  | 'URL_REWRITE'   | => 0,     | //url 重写模式                        |
| 3  | 'URL_TYPE'      | => 1,     | // 类型 1:PATHINFO 模式 2:普通模式 3:兼容模式 |
| 4  | 'PATHINFO_DLI'  | => '/',   | //PATHINFO 分隔符                    |
| 5  | 'PATHINFO_VAR'  | => 'q',   | // 兼容模式 get 变量                    |
| 6  | 'PATHINFO_HTML' | => '',    | // 伪静态扩展名                         |
| 7  | 'VAR_GROUP'     | => 'g',   | // 应用组变量                          |
| 8  | 'VAR_APP'       | => 'a',   | // 应用变量名，应用组模式有效                  |
| 9  | 'VAR_CONTROL'   | => 'c',   | // 模块变量                           |
| 10 | 'VAR_METHOD'    | => 'm',   | // 动作变量                           |

注：

如果当前为应用组模式，所有配置项必须在应用组配置文件中定义

配置项 VAR\_APP 与 DEFAULT\_APP 只能在应用组中定义

示例一（基于于默认配置）

http://localhost/index.php

上面的 URL 以采用默认应用、默认控制器、默认方法

与 `http://localhost/hdcms/index.php/index/index/index` 效果一样

### 示例二

`http://localhost/index.php/admin/user/admin`

调用 admin 应用目录下 `UserControl.class.php` 控制器文件中的 admin 方法

### 示例三

`http://localhost/index.php/admin/user/add.html`

以伪静态的形式访问，调用 admin 模块目录下 `UserControl.class.php` 控制器文件中的 add 方法

### 示例四（修改 URL 分隔符为下划线）

`http://localhost/hdcms/index.php/index_index_bb.html`

修改配置项中的 `PATHINFO_Del` 访问分隔符为 `_`，原来 `/` 的访问形式将用 `_` 来访问

## 14\_2 兼容模式访问方式

如果不支持 `pathinfo` 访问规则，可以使用兼容模式，需要修改以下几个配置：

```
1 array(  
2     URL_TYPE=>3,  
3     PATHINFO_VAR=>'q'  
4 );
```

可以使用配置项 `PATHINFO_VAR` 的值作为 `get` 参数来访问

`http://localhost/hdcms/index.php?q=admin/user/add.html`

访问 index 应用目录下的 `UserControl.class.php` 控制器文件中的 add 方法

## 14\_3 普通访问方式

以普通 GET 方式进行访问

`http://localhost/hdcms/index.php?c=user&m=add`

以上代码访问 `userControl.class.php` 控制器文件中的 add 方法

# 15. 获得系统数据 Q 函数

HDPHP 框架提供了功能增加的获得超全局数组的方法 Q()，即对超全局数组 \$\_GET、\$\_POST、\$\_SESSION、\$\_REQUEST、\$\_COOKIE、\$\_GLOBALS、\$\_SERVER 等进行更加方便的数据获得与设置手段。

语法 : Q(\$var, \$default = null,\$filter = null)

说明 :

1. 过滤函数默认使用配置项 'FILTER\_FUNCTION'
2. 变量是引用类型传参，所以会改变全局变量
3. 不指定数据类型，默认操作 \$\_REQUEST

如果 \$var 为多为数组，须将 \$fileter 设置为 NULL

| 参数        | 说明  |
|-----------|---|
| \$var     | 要处理的变量 (引用类型 )  |
| \$default | 当变量不存在时的默认值   |
| \$filter  | 过滤的函数，当为 null 时不进行顾虑函数处理，当为 " 时使用配置项 C('FILTER_FUNCTION') 中的函数进行过滤处理。 |

读取 \$\_REQUEST['cid'];

Q('cid');

获得所有 \$\_GET 数据

Q('get.');

获得所有 \$\_POST 数据

Q('post.');

获得 \$\_POST['webname'] 并执行 htmlspecialchars 与 strtoupper 函数

Q('post.webname',NULL,array('htmlspecialchars','strtoupper'));

获得 \$\_POST['webname']，当变量不存时，设置值为 ' 后盾网 '

Q('post.webname',' 后盾网 ');

获得 \$\_SESSION['uid'] 值

Q('session.uid',NULL,'intval');

获得 \$\_COOKIE['cart'] 值

1 Q('cookie.cart');

# 16. 判断请求类型

通过判断请求类型，我们可以达到以下几点好处：

1. 根据不同请求类型做出不同的业务处理
2. 同时也过滤掉不安全的请求，增强网站的安全性

## 16\_1 通过常量判断

HDPHP 将请求方式定义到了数组，我们可以通过常量快速进行判断当前请求的类型

| 方法名       | 说明            |
|-----------|---------------|
| IS_GET    | 是否为 get 请求    |
| IS_POST   | 是否为 post 请求   |
| IS_PUT    | 是否为 head 请求   |
| IS_DELETE | 是否为 delete 请求 |
| IS_AJAX   | 是否为异步 AJAX 请求 |

示例：

```
1 <?php
2 class indexControl extends Control{
3     function add_user(){
4         if(IS_POST){
5             $db = M('news');
6             $db->add();
7         }else{
8             $this->error(' 请求非法 ');
9         }
10    }
11 }
12 ?>
```

# 17. URL 路由器

路由操作类似于 Apache 的 Rewrite 操作，通过 URL 路由设置隐藏网站物理文件结构提高安全性，同时美化 URL 地址规则便于 SEO 及方便变量传递，下面来学习 HDPHP 框架的 URL 路由配置方法。

**说明：**

1. 如果以应用组形式开发，路由规则一定要定义在应用组配置文件中
2. 建议使用 U() 函数生成 url，会根据路由规则生成 url 地址

## 17\_1 普通路由

普通访问的 url 形式如下

http://localhost/index.php/index/user/uid/1

以上 url 访问 indexControl 控制器中的 user 方法传递参数 uid 值为 1

**示例一**

修改配置文件如下

```
1 'route'=>array(  
2   'user/:uid'=>'index/user'  
3 )
```

请求 URL : http://localhost/index.php/user/8

执行 IndexControl 控制器中的 user 方法并创建变量 \$\_GET['uid'] 值为 8

**示例二**

修改配置文件如下

```
1 'route'=>array(  
2   'index:_id'=>'index/index'  
3 )
```

请求 URL : http://localhost/index.php/index\_8

执行 IndexControl 控制器中的 Index 方法并创建变量 \$\_GET['id'] 值为 8

## 17\_2 正则路由

正则路由规则可以实现更加灵活的 URL 定制

修改配置文件如下

```
1 'route'=>array(  
2   '/^user(\d+)/'=>'index/user/uid/#1'  
3 )
```

访问 url: http://localhost/index.php/user16

执行 IndexControl 控制器中的 user 方法并创建变量 \$\_GET['uid'] 值为 16

注：

1. 通过正则设置一定要加上正则前后的边界符 /
2. 如果你不明白正则表达式使用 ,请收看 '后盾网正则表达式视频' 了解正则语法的使用。

## 17\_3 支持 QUERY 方式路由

HDPHP 框架路由不仅支持 PATHINFO 方式同时支持 GET 方式

```
1 'route' => array(  
2   '/^([a-z]+?)_(\d+)'=>'c=index&m=index&city=#1&row=#2'  
3 )
```

访问 url: http://localhost/hdcms/index.php?beijing\_20

解析为 http://localhost/hdcms/index.php?c=index&m=index&city=beijing&row=20

# 18. url 伪静态

url 伪静态处理易于被搜索引擎收录，同时隐藏了原始 url 所以更加安全。

url 伪静态需要隐藏项目入口文件，以及定义相应的路由规则，这样才可以生成完美的伪静态，路由的使用在上面已经进行了介绍，下面来学习隐藏项目入口文件。

## 18\_1 隐藏项目入口文件

### 服务器 rewrite 配置

- a. 将 Apache 配置文件 httpd.conf 中加载 mod\_rewrite.so 模块
- b. 修改配置 httpd.conf 中的 AllowOverride Node 为 AllowOverride All 使 Apache 支持 .htaccess 文件
- c. 在入口文件同级目录中创建 .htaccess 文件内容如下

```
<IfModule mod_rewrite.c>
# 开启 URL 重写功能
RewriteEngine On
# 请求内容不是目录
RewriteCond %{REQUEST_FILENAME} !-d
# 请求内容不是文件
RewriteCond %{REQUEST_FILENAME} !-f
# 重写 URL 规则
RewriteRule ^(.*)$ index.php\$/\$1 [L]
</IfModule>
```

### 注意

隐藏入口文件需要修改配置 'URL\_REWRITE'=>true，这样通过 U() 函数生成的 url 会去掉入口文件。

# 19. Ajax 返回数据

HDPHP 框架提供通过 ajax 返回 Ajax 请求数据，使用方法非常简单。

```
$this->ajax($data, $type = 'JSON')
```

| 参数说明   | 说明                             |
|--------|--------------------------------|
| \$data | 异步返回的数据                        |
| \$type | 返回的数据类型，包含 JSON(默认), XML, TEXT |

```
1 function index()  
2 {  
3     $data = array('webname' => '后盾网', 'url' => 'http://hdphp.com');  
4     $this->ajax($data, 'JSON');  
5 }
```

当通过 Ajax 异步请求 index 方法时，系统会返回 JSON 数据给客户端，当然可以指定不同的数据类型如 XML 返回给客户端。

# 20. 模型

模型是专门用来和数据库打交道的。例如，假设你想用 HD PHP 框架制作一个文章系统，那么你需要通过模型类来完成操作，与数据库操作的所有方法 HD PHP 框架已经封装到的模型的内部，你可以完成插入、更新、删除数据的操作，当然还有更多高级功能。

## HD PHP 框架模型的特点：

- a. 可以对基础模型类进行随意的扩充
- b. 支持加载其他模块或其他应用类某个模块的模型
- c. 丰富的加载模型方式
- d. 功能强大的数据操作方式
- e. 自动验证
- f. 自动完成
- g. 自动映射
- h. 表单令牌
- i. .....

## 20\_1 基本模型

`M($table = null, $full = null)`

| 参数说明    | 说明                      |
|---------|-------------------------|
| \$table | 表名，如果为 NULL 时使用控制器名做为表名 |
| \$full  | 值为 NULL 时不添加表前缀         |

### 使用控制器做为表名

```
$db = M()
```

### 设置表名

```
$db = M('user')
```

### 设置表名并且不添加表前缀

```
$db = M('user',true) 扩展模型
```

## 20\_2 扩展模型 ( K 函数 )

HDPHP 框架支持模型功能的扩展，扩展模型需要继续基本模型或高级模型。

**K(\$model, \$param = array())**

| 参数    | 说明     |
|-------|--------|
| model | 扩展模型名称 |
| param | 附加参数   |

### 示例

```
1 K('user',array('cid'=>1,'webname'=>'houdunwang.com')  
2 // 以下是扩展模型文件  
3 class UserModel extends Model{  
4     public $table='user';  
5     function __init($param){  
6         p($param);  
7     }  
8 }
```

### 注意：

1. 传递的参数只能在 `_init()` 方法中使用
2. 使用扩展模型中 `$table` 或 `$tableFull` 属性定义表名

# 21. CURD 操作

HDPHP 框架提供了便捷的模型方法链式操作，如果你对 CSS 熟悉或者对 JQUERY 熟悉那么这种操作方法很好理解的，如果不熟悉也没关系，因为 HDPHP 框架 CURD 的使用确实是非常简单。下面让我们来学习如何使用 HDPHP 框架的敏捷操作方式。

HDPHP 框架的 CURD 的自由度会超过你的想象，当开启 DEBUG 模式后，所有 SQL 语句均会在调试窗口显示，一目了然。

## 21\_1 查询单条 find

通过 find 操作进行简单的单条查询操作，也可以指定查询参数约束查询结果（请参考 where、limit、order by、group... 部分）

**注意：单条查找返回的是一维数据**

### 示例一

```
1 $db = M('news');  
2 $db->find();
```

返回第一条记录，生成的 SQL 如下

```
SELECT `id`,`name`,`sendtime` FROM hd_demo DESC LIMIT 1
```

### 示例二

```
1 $db = M('news');  
2 $db->find(18);
```

查询主键为 18 的记录，生成的 SQL 如下

```
SELECT `id`,`name`,`sendtime` FROM hd_demo WHERE id=18 LIMIT 1
```

### 示例三

```
1 $db=M('news');  
2 $db->where('id>2')->order('id')->find();
```

生成的 SQL 如下

```
SELECT `id`,`name`,`sendtime` FROM hd_demo WHERE id>2 LIMIT 1
```

## 21\_2 查询所有记录 all ( 别名 select 及 findall )

通过 findall 查找所有记录数，也可以指定查询参数约束查询结果（请参考 where、limit、order by、group... 部分）

## 示例一

```
1 $db = M('demo');  
2 $list = $db->findAll();
```

查询表中的所有记录，生成 SQL 如下

SELECT `id`,`name`,`sendtime` FROM hd\_demo

## 示例二

```
1 $db = M('demo');  
2 $db->where('id>2')->group('name')->having('id>2')->limit(3)->select();
```

生成 SQL 如下

SELECT `id`,`name`,`sendtime` FROM hd\_demo WHERE id>2 GROUP BY name  
HAVING id>2 LIMIT 3

## 示例三

```
1 $db = M('demo');  
2 $row = $db->field('count("uid") as usrecount')->find('age=22');
```

在表 demo 中查找 age 字段大于 22 的记录条数

## 示例四

```
1 $db = M('user');  
2 $db->where='uid=2';  
3 $db->where='username="后盾网"';  
4 $db->findAll();
```

以 ORM 方式进行组合查询，生成的 SQL 如下

SELECT `uid`,username,password` FROM info\_user WHERE uid = '2' AND  
username = '后盾网'

## 21\_3 table() 设置当前操作表

通过 table() 方法可快速切换操作表，使用方法非常简单示例如下：

```
1 $db = M('news');  
2 $row = $db->table('user')->all('age>20');// 查找表 user 中 age>20 的用户
```

## 通过 ORM 方式设置

```
1 $db=M();  
2 $db->table = 'user'; // 切换操作表
```

```
3 $row = $db->all('age>20'); // 查找表 user 中 age>20 的用户
```

## 21\_4 cache() 查询结果缓存

HDPHP 框架实现了 SQL 语句的缓存机制，可以灵活的设置全局缓存，也可以针对单条 SQL 指定缓存规则

### 影响 SQL 缓存的配置项

| 配置项                 | 说明                        |
|---------------------|---------------------------|
| CACHE_SELECT_TIME   | SQL SELECT 查询缓存时间 0 为永久缓存 |
| CACHE_SELECT_LENGTH | SELECT 结果条数超过这个值将不进行缓存    |

### 21-4-1 通过配置文件指定缓存时间

缓存时间由配置项 CACHE\_SELECT\_TIME 决定的，通过 cache() 方法设置的缓存配置级别要高于配置文件的设置，所以可以用配置文件设置全局的一个缓存时间，针对某条 SQL 可以使用 cache() 方法来单独操作。

#### 结果不缓存

```
1 $db=M('user');
2 $db->cache()->select();
```

以上 SQL 不会被缓存

### 21-4-2 指定结果缓存时间

```
1 $db=M('user');
2 $db->cache(30)->select();
```

以上查询结果会缓存 30 秒

#### 基于 ORM 方式的设置

```
1 $db = M('user');
2 $db->cache=200;
3 $db->all();
```

## 21\_5 max 查找最大的值

获得最大值，与 SQL 中的 MAX 意思相同，我们来具体看一下

### 示例一

```
1 $db = M('user');  
2 $row = $db->order('uid asc')->max('uid|maxuid');
```

求表 user 中的最大 uid 字段，生成的 SQL 如下

SELECT MAX(uid) AS maxuid FROM hd\_user

### 示例四

```
1 $db= M('user');  
2 $row = $db->where('username like "%xj%")->max();
```

统计 username 字段含有 xj 的最大的记录

## 21\_6 min 查找最小的值

获得最大值，与 SQL 中的 MIN 意思相同，我们来具体看一下

### 示例一

```
1 $db= M('user');  
2 $row = $db->min('uid');
```

求表 user 中 uid 字段最小的记录，生成的 SQL 如下

SELECT MIN(uid) AS uid FROM hd\_user ORDER BY hd\_user.`uid` ASC

### 示例二

```
1 $db= M('user');  
2 $row = $db->min('uid|uidmin');
```

通过 | 指定别名，以上代码求 user 表中的最小 uid 字段，并且另起别名 uidmin

以上代码执行后生成的 SQL 如下

SELECT MIN(uid) AS uidmin FROM info\_user

### 示例三

```
1 $db= M('user');  
2 $row = $db->where('username like "李 %")->min('uid|li');
```

在 user 表中求 username 以李开头的，uid 值为最小的记录，同时通过 | 线符号为 uid  
字段起别名为 li\_min，生成的 SQL 如下

```
SELECT MIN(uid) AS li FROM hd_user WHERE username like '李 %'
```

## 21\_7 avg 求平均值

获得平均值与 SQL 中的 AVG 意思相同，我们来具体看一下

### 示例一

```
1 $db= M('user');  
2 $row = $db->avg('uid|uidavg');
```

通过 | 指定别名，以上代码求 user 表中的 uid 字段平均值，并且另起别名 uidavg

以上代码执行后生成的 SQL 如下

```
SELECT AVG(uid) AS uidavg FROM info_user ORDER BY uid ASC
```

## 21\_8 sum 求和

获得平均值与 SQL 中的 SUM 意思相同，我们来具体看一下

### 示例一

```
1 $db= M('article');  
2 $row = $db->sum('click|total');
```

通过 | 指定别名，以上代码求 user 表中的 click 总和，并且另起别名 total

以上代码执行后生成的 SQL 如下

```
SELECT SUM(click) AS total FROM article
```

## 21\_9 count 统计操作

对查询结果进行统计，即 SQL 中的 COUNT 是一样的。HDPHP 框架中的 COUNT 不仅可以用在普通查询上，也可以用在关联操作上有关联操作的 CURD 使用，请参考多表关联操作部份章节。

### 示例一 ( 统计所有记录 )

```
1 $db = M('user');// 关联模型  
2 $row = $db->count();
```

对 user 表中的所有记录进行统计，返回值为数字

### 示例二

```
1 $db = M('user');// 关联模型
```

```
2 $row = $db->where('username like "%李四%")->count();
```

对 user 表中 username 字段包含李四的进行统计

### 示例三

```
1 $db= M('user');  
2 $row = $db->count('uid');
```

在 user 表中统计 uid 记录数量

### 示例四

```
1 $db= M('user');  
2 $row = $db->where('uid>10')->group('username')->count('uid|cc');
```

## 21\_10 field 字段集

通过 field 函数指定返回结果的字段 ,HDPHP 框架会过滤掉非法字段 , 可以设置字段的别名

### 示例一

```
1 $db = M('user');  
2 $db->field('uid,username|uu')->select();
```

查询表 demo , 返回 id、 name 字段数据集 , 生成的 SQL 如下

SELECT uid,username FROM hd\_demo

### 示例二 ( 通过数组传递参数 )

```
1 $db = M('user');  
2 $db->field(array('uid','username'))->select();
```

查询表 demo , 返回 id、 name 字段数据集 , 生成的 SQL 如下

SELECT uid,username FROM hd\_demo

## 21\_11 limit 取部分数据

limit 方法就是为了生成 SQL 的 limit 部分 ,HDPHP 框架提供了方便的操作特性

### 示例一

```
1 $db=M('demo');  
2 $db->limit(2)->order('uid desc')->all();
```

以 id 进行降序排列 , 取得前 2 个记录 , 生成的 SQL 如下

SELECT `uid` , `name` , `sendtime` FROM hd\_demo LIMIT 2

## 示例二

```
1 $db=M('demo');  
2 $db->limit(array(1,2))->all();
```

指定 2 个参数来生成 limit , 以上生成的 SQL 如下

```
SELECT `id`,`name`,`sendtime` FROM hd_demo LIMIT 1,2
```

## 示例三 ( 属性映射方式 )

```
1 $db=M('demo');  
2 $db->limit = array(10,6);  
3 $db->all();
```

以对象属性的方式生成 limit 部分 , 以上代码生成的 SQL 如下

```
SELECT `id`,`name`,`sendtime` FROM hd_demo LIMIT 10,6
```

## 21\_12 in 匹配多个值

in 关键字与原生 SQL 中的 in 意思是一样的 , 用于匹配 in 表达式中的多个值

### 示例一 ( 多参数传递 )

```
1 $db = M('user');  
2 $list = $db->in('1,2')->all();
```

查找 user 表中主键为 1 或 2 的记录集生成 SQL 如下

```
SELECT hd_user.`uid`,hd_user.`username`,hd_user.`password` FROM hd_user  
WHERE uid in(1,2)
```

### 示例二 ( 以数组形式传递参数 )

```
1 $db = M('user');  
2 $list = $db->in(array(1,2))->all();
```

查找 user 表中主键为 1 或 2 的记录集生成 SQL 如下

```
SELECT hd_user.`uid`,hd_user.`username`,hd_user.`password` FROM hd_user  
WHERE uid in(1,2)
```

### 示例三 ( 指定字段 )

```
1 $db = M('user');  
2 $list = $db->in(array('uid'=>array(1,2)))->all();
```

查找 user 表中主键为 1 或 2 的记录集生成 SQL 如下

```
SELECT hd_user.`uid`,hd_user.`username`,hd_user.`password` FROM hd_user
```

WHERE uid in(1,2)

#### 示例四

```
1 $db=M('user');  
2 $list = $db->field('username,uid')->in(array('username'=>array('向军','后盾')))->find();
```

查找 username 字段为向军或后盾的值生成的 SQL 如下

```
select `username`,`uid` from hd_user where username in('向军','后盾') limit1;
```

## 21\_13 where 查询条件

HDPHP 框架通过 where 方法操作 SQL 条件，可以指定多种参数类型，非常自由

#### 示例一 ( 参数为字符串 )

```
1 $db = M('user');  
2 $list = $db->where('uid=1 or username="后盾")->find();
```

通过字符串指定查询条件，查找 id 为 1 并且姓名为后盾的人，生成 SQL 如下

```
SELECT * FROM hd_user WHERE uid=1 or username="后盾" LIMIT 1
```

#### 示例二 ( 以数组形式查找 )

```
1 $db->where(array('uid'=>2))->find();
```

查找 id=2 的数据，生成 SQL 如下

```
SELECT * FROM hd_user WHERE uid=2 LIMIT 1
```

#### 示例三

```
1 $db->where(array('uid' => array(1, 2, 3)))->select();
```

查找 cid 属于 1、2、3 的数据，生成的 SQL 如下

```
SELECT * FROM hd_user WHERE uid in(1,2,3)
```

#### 示例四

```
1 $db->where(array('uid'=>array('elt'=>2,'gt'=>8)))->find();
```

查找 id 小于等于 2 并且大于 8 的数据，生成 SQL 如下

```
SELECT * FROM hd_user WHERE uid <= 2 AND uid > 8 LIMIT 1
```

#### 示例五

```
1 $db->where(array('uid' => array('neq' => 22), 'or', 'username' => array('eq' => '后盾')))->find();
```

查找 uid 不等于 22 或者 username 等于 '后盾' 的数据，生成 SQL 如下

```
SELECT * FROM hd_user WHERE uid <> 22 OR username = '后盾' LIMIT 1
```

## 21\_14 基于 ORM 方式的 WHERE

HDPHP 框架可以直接使用类似 ORM 方式完成查询操作，以上所有 where 的参数均可  
以通过以对象属性方式指定，非常简便，示例如下。

### 示例 1

```
1 $db->where = array('uid' => array('neq' => 2), 'rid' => array(1, 22, 3));  
2 $db->findall();
```

生成 SQL 如下：

```
SELECT * FROM hd_user WHERE uid <> 2 AND rid in(1,22,3)
```

## 21\_15 排序 ORDER

生成 SQL 语句中的排序 ORDER BY 部分

### 示例一

```
1 $db=M('demo');  
2 $db->order('id desc')->all();
```

查询所有记录以 id 降序，生成的 SQL 如下

```
SELECT `id`,`name`,`sendtime` FROM hd_demo ORDER BY id desc
```

### 示例二

```
1 $db=M('demo');  
2 $db->order(array('id'=>'asc'))->all();
```

以数组形式指定参数进行排序，生成的 SQL 如下

```
SELECT `id`,`name`,`sendtime` FROM hd_demo ORDER BY id asc
```

### 示例三

```
1 $db= M('demo');  
2 $db->order(array('id'=>'asc','name'=>'desc'))->all();
```

传递数组与字符串两种参数类型，指定排序规则，HDPHP 框架会自动处理，SQL 如下

```
SELECT `id`,`name`,`sendtime` FROM hd_demo ORDER BY id asc,name desc
```

## 21\_16 getField 获得指定字段值

为了使查询更加灵活，HDPHP 框架提供了 getField 按字段名获得结果的方法，具体使  
用示例如下：

```
1 $db->where('id=32')->getField('title');// 无论结果有多少个只返回一个值
```

```
2 $db->where('id>2')->getField('title',true);// 返回满足条件的记录，并压入一个数组中  
3 $db->getField('id,title,click');// 多个字段返回关联数组，第一个字段做为键名使用
```

---

## 21\_17 GROUP 分组操作

HDPHP 框架提供了完善的分组操作方法，自由指定分组参数使发送 SQL 更容易

### 示例一

```
1 $db=M('demo');  
2 $db->group('uname')->findall();
```

---

以 uname 字段分组查询，生成的 SQL 为

```
SELECT `id`,`uname`,`birday` FROM hd_demo GROUP BY uname
```

### 示例二

```
1 $db=M('demo');  
2 $list =$db->group('id,name')->all();
```

---

以字段 id、cc 进行分组查询

```
SELECT `id`,`name`,`sendtime` FROM hd_demo GROUP BY id,name
```

### 示例三 ( 以数组作为参数 )

```
1 $db=M('demo');  
2 $list =$db->group(array('id,name'))->all();
```

---

以字段 id、cc 进行分组查询，生成的 SQL 如下

```
SELECT `id`,`name`,`sendtime` FROM hd_demo GROUP BY id,name
```

### 示例四 ( 属性映射 )

```
1 $db=M('demo');  
2 $db->group=array('price','age');  
3 $db->field('name','age')->select();
```

---

以对象属性来指定分组参数，参数与方法 group() 传递的一样即可，生成的 SQL 如下

```
SELECT `name`,`age` FROM hd_demo GROUP BY price,age
```

## 21\_18 HAVING 分组条件

在 SQL 中增加 HAVING 子句原因是，WHERE 关键字无法与合计函数一起使用，所以使用 HAVING 对分组进行条件筛选，所以在使用 HAVING 时应该使用 group 分组。

## 示例一

```
1 $db = M('demo');  
2 $db->having('id>2 and age<20')->group('age')->select();
```

以字段 age 进行分组，查询 id 大于 2 及 age 小于 20 的数据，生成的 SQL 如下

```
SELECT `id`,`name`,`sendtime` FROM hd_demo GROUP BY age HAVING id>2  
AND age<20
```

## 示例二 ( 属性映射 )

```
1 $db=m('demo');  
2 $db->having="id>20 and name like '李 %'";  
3 $db->group('name')->findall();
```

以属性映射的方式指定 having 的参数，与使用 having() 方法的效果等价，传递参数方式也一样，生成的 SQL 如下

```
SELECT `id`,`name`,`sendtime` FROM hd_demo GROUP BY name HAVING id>20  
AND name like '李 %'
```

## 21\_19 add 添加数据 ( 别名 : insert )

语法 : add(\$data = array(),\$type='INSERT')

参数说明

| 参数     | 说明   |
|--------|--|
| \$data | 插入的数据  |
| \$type | 插入方式 : INSERT、REPLACE ( 数据中必须有主键或唯一索引字段，否则 REPLACE 等于 INSERT ) |

HDPHP 框架提供了非常便捷的插入操作，具有以下几个特点：

- a. 自动过滤非法字段
- b. 自动对插入数据进行安全处理
- c. 没有传入参数，则系统会自动将 \$\_POST 数据插入

## 示例一

```
1 $_POST=array(  
2   'username'=>'后盾网',  
3   'password'=>md5(123456),  
4 );
```

```
5 $db=M('user');
6 $lastId = $db->insert();
```

如果无参数默认以 `$_POST` 为数据源

返回成功插入记录的 ID 值，生成的 SQL 语句为：

```
INSERT INTO info_user(`username`,`password`)VALUES ('后盾网','e10adc3949ba5
9abbe56e057f20f883e')
```

## 示例二

```
1 $db=M('demo');
2 $data=array('uname'=>'后盾网','url'=>'http://www.houdunwang.com');
3 $lastId = $db->add($data)
```

插入成功后，HDPHP 框架会返回最后插入的主键 ID 值

## 示例三（基于 ORM 的添加）

```
1 $db = M('user');
2 $db->username = '李四';
3 $db->password = md5(2893289);
4 echo $db->add();
```

通过属性映射的方式 添加数据，生成的 SQL 如下

```
INSERT INTO info_user(`username`,`password`)VALUES ('李四','6c809dbcebf51e6
699618e647d8d0c6a')
```

## 21\_20 addAll 批量添加数据

当需要批量添加数据时可以使用 `addAll` 方法，可以通过以下示例学习。

### 示例

```
1 $db = M('news');
2 $data=array(
3     array(
4         'title'=>'标题 1',
5         'click'=>11
6     ),
7     array(
8         'title'=>'标题 2',
9         'click'=>121
)
```

```
10    )
11 );
12 p($db->addAll($data));
```

## 结果

```
1 Array
2 (
3   [0] => 5
4   [1] => 6
5 )
```

## 21\_21 replace 添加数据

replace 使用 sql 语句中的 REPLACEr 指令 , 如果发现表中存于此行数据 ( 根据主键或者唯一索引判断 ) 则执行替换操作 , 否则为新增操作。

```
1 $data =array('id'=>1,'name'=>' 后盾网 ');
2 $db=M('class');
3 $db->replace($data);
```

如果表中存在 id 为 1 的主键记录 , 则更新原有记录

## 21\_22 update 更新数据 ( 别名 : save)

HDPHP 框架更新操作提供了简单快速的插入操作 , HDPHP 框架的 UPDATE 更新操作要求必须输入条件 , 如果参数中有主键则系统会自动以这个主键值为条件。

### 说明

1. 自动过滤掉非法字段
2. 自动进行数据安全处理
3. 默认以 \$\_POST 数据更新
4. 如果参数中存在主键值将以这个值为条件进行更新数据

### 示例一

```
1 $_POST=array('uid'=>9,'username'=>' 刘德华 ');
2 $db = M('user');
3 echo $db->update();
```

没有指定参数将以 \$\_POST 为参数进行更新 , 所以 \$\_POST 必须有值 , 并且必须有主键值 , 否则不进行更新 , 这样可以很好的保证数据不被误操作更新

生成的 SQL 为：

```
UPDATE info_user SET `uid`='9',`username`='刘德华' WHERE `uid` = '9'
```

### 示例二

```
1 $db = M('user');
2 $data=array('uid'=>9,'username'=>'郭富城');
3 echo $db->save($data);
```

插入的数据 uid 为主键

使用 update 的别名 save 进行更新

### 示例三

```
1 $db = M('stu');
2 $db->where = 'username = "后盾网"';
3 $db->username = '后盾 PHP 框架';
4 echo $db->save();
```

通过 ORM 对象映射修改数据，生成的 SQL 如下

```
UPDATE info_user SET `username`='后盾 PHP 框架' WHERE username = '后盾网'
```

## 21\_23 inc() 增加值

```
1 $db = M('shop');
2 $db->inc('total','id=4',1);
```

将 id=4 的记录 total 加 1

```
update hd_shop set total=total+1 where id=4
```

## 21\_24 dec() 减少值

```
1 $db = M('shop');
2 $db->dec('total','id=4',1);
```

将 id=4 的记录 total 减 1

```
update hd_shop set total=total-1 where id=4
```

## 21\_25 delete() 删除数据（别名：del）

HDPHP 框架删除数据时必须指定条件，这也是为了屏蔽误删除，如果以 \$\_POST 传参

时存在主键，则删除这个主键的记录。

#### 示例一

```
1 $db=M('demo');
2 echo $db->delete(58);
```

删除主键为 58 的记录，如果删除成功返回受影响的记录数，生成 SQL 如下

DELETE FROM hd\_demo WHERE id in(58)

#### 示例二

```
1 $db=M('demo');
2 $db->delete(array(1,2,3,58));
```

删除主键为 1、2、3、58 的数据，并返回受影响的记录数，生成 SQL 如下

DELETE FROM hd\_demo WHERE id in(1,2,3,58)

#### 示例三

```
1 $db =M('demo');
2 $db->where('id>2')->del();
```

删除 id>2 的所有记录，生成的 SQL 如下

DELETE FROM hd\_demo WHERE id>2

#### 示例四

```
1 $db=M('demo');
2 $db->delete(array('id>9','id<30'));
```

在 delete 方法可以指定任何的可以用于 where 方法的参数，有关 where 参数要求请参看 where 部分，生成 SQL 如下

DELETE FROM hd\_demo WHERE id>9 AND id<30

## 21\_26 getTableFields() 获得表所有字段集

getTableFields 用于指定表的所有字段，返回值为数组

```
1 M()->getTableFields('news');
```

## 21\_27 fieldExists() 判断表中字段是否存在

```
1 $db = M();
2 $db->fieldExists('title','news');
```

注：不需要加表前缀

## 21\_28 tableExists() 检测表是否存在

tableExists 方法用于判断当前数据库中是否已经存在表

```
1 $db = M();  
2 $db->tableExists('news');
```

注：不需要加表前缀

## 21\_29 获得数据库版本信息 getVersion

用于获得数据库版本信息，示例如下：

```
1 $db = M();  
2 $db->getVersion();
```

## 21\_30 获得最后一条 SQL 语句 getLastSql()

```
1 $db = M('news');  
2 $db->find();  
3 $db->getLastSql();
```

## 21\_31 获得受影响的行数 getAffectedRows()

```
1 $db = M('news');  
2 $db->all();  
3 $db->getAffectedRows();
```

## 21\_32 获得所有 SQL 语句 getAllSql()

```
1 $db = M('news');  
2 $db->all();  
3 p($db->getAllSql());
```

## 21\_33 获得最后插入的主键 ID 值 getInsertId()

```
1 $db = M('news');  
2 $data=array('title'=>'后盾 HDPHP 框架');  
3 $db->insert($data);  
4 echo $db->getInsertId();
```

## 21\_34 获得数据库或者表的大小 getSize()

注：必须加表前缀

获得数据库大小（所有表）

```
1 $db = M();  
2 echo $db->getSize();
```

获得表 news 表与 channel 表的大小

```
1 $db = M();  
2 echo $db->getSize(array('hd_news','hd_channel'));
```

## 21\_35 获得表信息 getTableInfo()

本函数会返回数据库表信息及所有记录行数及数据库大小等详细的信息

注：该方法会忽略配置文件中的表前缀设置，所以要加表前缀

获得当前数据库的所有表信息

```
p(M()->getTableInfo());
```

获得表 hd\_user,hdphp 的信息

```
M()->getTableInfo(array('hd_user','hdphp'));
```

## 21\_36 创建数据库 createDatabase()

```
M()->createDatabase('hdphp','gbk');
```

以 gbk 编码创建数据库 hdphp

## 21\_37 清空表 truncate()

注：该方法会忽略配置文件中的表前缀设置，所以请添加表前缀

```
M()->truncate(array('hd_news','hd_user'));
```

清空表 hdphp , hd\_user

## 21\_38 一次修复多个数据表 repair()

注：该方法会忽略配置文件中的表前缀设置，所以请添加表前缀

```
M()->repair(array('hd_news','hd_user'));
```

修复数据表 hd\_news,hd\_user

## 21\_39 一次优化多个数据表 optimize()

注：该方法会忽略配置文件中的表前缀设置，所以请添加表前缀

```
M()->optimize(array('hd_news','hd_user'));
```

修复数据表 hd\_news 及 hd\_user

## 21\_40 修改表名 rename( ) ;

将表 user 改名为 hd\_user

```
M()->rename('user','hd_user');
```

## 21\_41 删除表 dropTable()

[删除数据表 hdphp](#)

```
M()->dropTable('hd_hdphp');
```

注：必须加表前缀

## 21\_42 设置自动提交 beginTrans()

如果要完成数据库的事务处理需要选择合适的表引擎如 InnoDB、NDB、BDB 这是进行事务处理的前提

```
1 $db =M('wages');  
2 $data=array('wages'=>100);  
3 $db->beginTrans();// 开启事务  
4 $db->add($data);// 添加数据  
5 $db->commit(); 提交事务
```

## 21\_43 事务回滚 rollback()

当事务完整性被破坏或者其他原因可以通过 rollback() 方法放弃本次事务操作，示例如下：

```
1 $db =M('wages');  
2 $data=array('wages'=>100);  
3 $db->beginTrans();// 开启事务  
4 $db->add($data);// 添加数据  
5 $db->rollback();// 放弃本次提交
```

## 21\_44 提交事务 commit()

如果整个事务完成正确可以通过 commit() 进行事务的提交完成最终操作，示例如下：

```
1 $db =M('wages');
2 $data=array('wages'=>100);
3 $db->beginTrans();// 开启事务
4 $db->add($data);// 添加数据
5 $db->commit();// 提交本次事务
```

## 21\_45 执行 SQL 语句 runSql()

```
1 $sql= file_get_content('sql.sql');//sql.sql 为 SQL 语句文件
2 M()->runSql($sql);
```

# 22. 触发器 (trigger)

触发器是指在执行 CURD 时动态执行的方法，类似于 Mysql 中的触发器。可以使用相应触发器完成数据初始化，以及操作结果二次处理的目的。

**注意：**

1. 触发器必须传递参数
2. \_\_before\_xx 前置触发器必须以引用值形式传参
3. 触发器必须在扩展模型中定义

## 22\_1 触发器方法

| 方法                       | 说明            |
|--------------------------|---------------|
| __init                   | 模型实例化时自动执行的方法 |
| __before_insert(&\$data) | 添加数据前执行的方法    |
| __after_insert(\$data)   | 添加数据后执行的方法    |
| __before_delete(&\$data) | 删除数据前执行的方法    |
| __after_delete(\$result) | 删除数据后执行的方法    |
| __before_update(&\$data) | 更新数据后前执行的方法   |
| __after_update(\$data)   | 更新数据后执行的方法    |
| __before_select(&\$data) | 查询数据前前执行的方法   |
| __after_select(\$data)   | 查询数据后执行的方法    |

## 22\_2 开关触发器 trigger

如果设置了触发器默认 CURD 都会执行，但有时有些动作不希望激活触发器，这时可以使用 trigger 方法开启或关闭。

语法 : public function trigger(\$stat = FALSE)

示例 1 ( 开启触发器，系统默认就是开启的 )

```
1 $db= M('new');
```

```
2 $db->trigger(true);
```

---

## 示例 2 ( 关闭触发器 )

```
1 $db=M('news');
```

---

```
2 $db->trigger(false);
```

---

# 23. 多表关联

在一个数据库中的多个表都是相互关联的。HDPHP 框架提供了多表关联支持，令开发者可以轻松应对各种数据表的关联操作。

使用 HDPHP 框架的关联操作需要定义模型属性 \$join，可以在扩展模型类中定义，也可以直接在控制器中修改属性 \$join 来定义关联。通过关联模型定义是推荐的方式，因为有更好的重用性。

注：设置关联时 `foreign_key` 与 `parent_key` 是必须设置的，否则无法关联

## 23\_1 关联类型

### 一对关联

一对一关联是最简单的关联形式，比如说网站中用户的注册信息，当用户注册时只有基本的注册资料如用户名密码等，假如说我们的网站还有其他应用，如论坛、博客等，那么当用户开通论坛与博客时也要添加相应的资料信息。

这个时候用户信息表与论坛或博客应用的用户信息表就是一个典型的一对一关联，两张表之间是一种并列的关系。

### 一对多关系

一对多关系是最常见的一种表关系，比如说一个文章可以只能属性一个栏目，但一个栏目当中可以有多个文章，那么文章表与栏目表之间就存在一种一对多关联，栏目表为一，文章表为多。

两张表之间是一种从属的关系，往往我们会把关联的键放在多的一方。

### 多对多关系

多对多关系无法区分主表与从表，比如说学生表与学生所学课程表之间就是一种多对多关系，因为一个学生可以学多个课程，一个课程也可以有多个学生来学习。

对于多对多的情况，创建一张中间表来产生一对多的关系。

# 24. 关联模型 RelationModel

多表关联使用 RelationModel 高级模型类来完成，使用 R() 函数会实例化 RelationModel 模型类，也可以创建扩展模型继承 RelationModel 来实例多表关联操作，推荐使用自定义扩展模型这种方式。

## 24-1-1 关联操作示例所需要的表

为了完成关联操作的演示，我们准备几张表，以下是各个表的表结构。

### 基本用户表 hd\_user

|          |       |
|----------|-------|
| uid      | 用户 ID |
| username | 用户名   |
| password | 密码    |

### 附加用户信息表 hd\_user\_info

|         |                   |
|---------|-------------------|
| email   | 邮箱                |
| address | 用户地址              |
| qq      | 用户 QQ 帐号          |
| user_id | 用户 ID 主键 ( 关联字段 ) |

### 文章表 hd\_news

|         |                          |
|---------|--------------------------|
| id      | 主键 ID                    |
| title   | 留言标题                     |
| con     | 留言内容                     |
| user_id | 留言用户的 ID ( 即用户表中的用户 ID ) |

### 栏目表 hd\_channel

|       |       |
|-------|-------|
| cid   | 主键 ID |
| cname | 栏目名   |

## 24\_1 关联模型对象创建方法

### 24-1-1 使用 R() 函数产生多表关联对象

```
1 $db = R('user');
2 $db->join = array(
3     'user_info' => array("// 关联表
```

```
4      'type' => HAS_ONE,// 包含一条主表记录
5      'foreign_key' => 'user_id',// 从表关联字段
6      'parent_key'=>'uid');// 主表关联字段
7  )
8 );
```

## 24-1-2 创建扩展模型（建议使用的方式）

在应用 model 创建扩展模型文件 UserModel.php 内容如下

```
1 class UserModel extends RelationModel {
2     public $join = array(
3         'user_info' => array("// 关联表
4             'type' => HAS_ONE,// 包含一条主表记录
5             'foreign_key' => 'user_id',// 从表关联字段
6             'parent_key'=>'uid');// 主表关联字段、
7     )
8 );
9 }
```

## 24\_2 关联模型定义参数说明

### type 关联类型

| 参数说明         | 说明  |
|--------------|-----|
| HAS_ONE      | 一对一 |
| HAS_MANY     | 一对多 |
| MANY_TO_MANY | 多对多 |
| BELONGS_TO   | 一对多 |

### parent\_key 主表字段

用户表与新闻表为一对多关联，那么 parent\_key 就是指用户表中的 uid 字段。

### foreign\_key 关联表字段

用户表与新闻表为一对多关联，需要新闻表 user\_id 关联用户表主键，foreign\_key 就是新闻表 user\_id。

在 HDPHP 框架中的关联非常灵活不是说仅能与主键进行关联，比如说想把留言表与新闻表进行关联，可以设置 `foreign_key` 为 `uid`（留言表与用户表关联的字段），`parent_key` 为 `uid`（新闻与用户表关联的字段）这样就可以方便的将留言表与新闻表进行关联，得到某个用户发表的留言与新闻记录集。

`field` 关联表显示的字段，支持别名定义与 HDPHP 框架模型的 `filed` 使用方法一样

关联表中有同名字段，在进行一对一关联时会出现字段的覆盖情况，通过指定 `field` 属性可以完全避免这种情况的发生，如 `title|biaoti` 就是将字段 `title` 另起别名为 `biaoti`

## 24\_3 关联模型中使用查询语句

```
1 class IndexControl extends Control {  
2     function index() {  
3         $db = R('user');// 关联模型  
4         $db->join = array(  
5             'news' => array("// 与新闻表关联  
6                 'type' => HAS_MANY,// 关联方式  
7                 'foreign_key' => 'uid',// 关联字段  
8                 'parent_key'=>'uid',// 关联字段  
9                 'field' => 'con|neirong',// 查询字段  
10            ),  
11        );  
12        $row = $db->all();  
13    }  
14 }
```

## 24\_4 join 方法

一个表常常会与多个表进行关联，但并不是每次操作都要用到这些关联表，可以通过 `join()` 方法指定关联方式如下：

`$db->join(null)->all();// 不关联任何表`

`$db->join('news')->all();// 只关联 news 表`

`$db->join(array('news','user'))->all();// 关联 news 表与 user 表`

## 24\_5 关联查询

### 24-5-1 HAS\_ONE ( 包含一条记录 )

每篇文章包含一个发布者，即为 HAS\_ONE, 查询结果是一维数组。

注：

因为结果是一维数组，有同名字段为覆盖，所以最好定义 field 属性，然后起别名

#### 示例一

```
1 class indexControl extends Control {  
2     function index() {  
3         $db = R('user');  
4         $db->join = array(  
5             'user_info' => array("// 关联表  
6                 'type' => 'HAS_ONE', // 包含一条主表记录  
7                 'parent_key' => 'uid', // 主表字段  
8                 'foreign_key' => 'user_id', // 关联字段  
9             ),  
10        );  
11        $row = $db->join('user_info')->select();  
12    }  
13 }
```

### 24-5-2 只匹配关联表中的部分字段

```
1 class IndexControl extends Control {  
2     function index() {  
3         $db = R('user');  
4         $db->join = array(  
5             'user_info' => array("// 关联表  
6                 'type' => HAS_ONE, // 包含一条主表记录  
7                 'foreign_key' => 'user_id', // 关联字段  
8                 'parent_key' => 'uid', // 关联字段  
9                 'field' => 'qq,address,email|youxiang', // 关联表检索的字段  
10            ),  
11        );
```

```
12 $row = $db->join('user_info')->select();  
13 }  
14 }
```

关联 user\_info 表，只筛选 qq,address,email 字段 ,email 字段起别名为 youxiang

### 24-5-3 定义扩展模型

在 Model 文件夹中定义扩展模型，可以被多个应用重用。

```
1 class UserModel extends RelationModel {  
2     public $join = array(  
3         'user_info' => array("// 关联表  
4             'type' => HAS_ONE, // 包含一条主表记录  
5             'foreign_key' => 'user_id', // 关联字段  
6             'parent_key' => 'id', // 主表主键  
7             'field' => 'qq,address,email|youxiang', // 关联表检索的字段  
8         ),  
9     );  
10 }
```

在控制器中执行关联查询

```
1 class IndexControl extends Control {  
2     function index() {  
3         $db = K('user');  
4         $row = $db->field('uid,username,password')->where('uid>0')->order('uid desc')->findall();  
5     }  
6 }
```

user 表与 user\_info 表关联，获得 user\_info 表的 qq、 address , email 字段及 user 表的 username 字段，并将 email 字段起别名为 youxiang。

生成的结果集如下

```
1 Array  
2 (  
3     [1] => Array  
4         (  
5             [username] => 马八
```

```
6      [password] => 999
7      [uid] => 475
8      [qq] => 北京市
9      [address] =>
10     [youxiang] => qq@sina.com
11   )
12 )
```

## 24-5-4 HAS\_MANY ( 包含多条记录 )

HAS\_MANY 与 HAS\_ONE 基本一样。 HAS\_ONE 中返回一维数组 , HAS\_MANY 返回多维数组

```
1 class IndexControl extends Control {
2   function index() {
3     $db = R('user ');
4     $db->join = array(
5       'news' => array(// 关联表
6         'type' => HAS_MANY, // 包含多条主表记录
7         'foreign_key' => 'user_id', // 关联字段
8         'parent_key' => 'uid', // 主表主键
9         'field' => 'title,con', // 关联表检索的字段
10        ),
11      );
12     $row = $db->field('uid,username,password')->findall();
13   }
14 }
```

结果集如下

```
1 Array
2 (
3   [0] => Array
4   (
5     [username] => 李四
6     [password] => 111
7     [uid] => 474
```

```
8 [news] => Array
9 (
10 [0] => Array
11 (
12 [title] => 新闻标题
13 [con] => 今天下雪了
14 )
15 )
16 )
17 )
```

## 24-5-5 BELONGS\_TO( 被包含 )

比如说栏目表与文章表进行关联，文章表中每一条记录都有栏目表中的 id，如果指定文章表为主表，那么栏目表中与文章表间的关系就是 BELONGS\_TO，意思为栏目表中有一个字段保存在文章表中，我们来通过示例来进行解释

### 示例一

```
1 class indexControl extends Control {
2     function index() {
3         $db = R('news');
4         $db->join = array(
5             'channel' => array("// 关联表
6                 'type' => BELONGS_TO, //channel 表中一个字段保存在 news 表中
7                 'foreign_key' => 'cid', // 关联字段
8                 'parent_key' => 'cid', // 主表主键
9                 'field' => 'cid,cname', // 关联表检索的字段
10            )
11        );
12        $row = $db->all();
13    }
14 }
```

BELONGS\_TO 定义与 HAS\_MANY 定义相反，具体操作与 HAS\_MANY、HAS\_ONE 是一样的，只需要理解这是一个角色互换就可以了，也就是说 HAS\_MANY 可以用

BELONGS\_TO 实现、BELONGS\_TO 也可以用 HAS\_MANY 实现。

以上代码生成的结果集为

```
1 Array
2 (
3   [0] => Array
4   (
5     [id] => 1
6     [title] => 新闻标题
7     [click] =>
8     [user_id] => 1
9     [con] => 新闻内容
10    [cid] => 1
11    [cname] => php
12  )
13
14 )
```

## 24-5-6 MANY\_TO\_MANY 多对多关联操作

一般情况下将多对多转化为多个一对多关系，这样操作起来更方便，比如说，用户表与用户组表，一个用户可以属于多个用户组，一个用户组也可以有多个用户，那么这种场景下的关系就是典型的多对多关系

**注意：**

中间表的关联字段必须与关联的两张表字段名相同

**示例一**

```
1 class IndexControl extends Control {
2   function index() {
3     $db = R('user');
4     $db->join = array(
5       'group' => array("// 关联表
6       'type' => MANY_TO_MANY, // 包含一条主表记录
7       'relation_table' => 'user_role', // 中间关联表
8       'foreign_key' => 'rid',
```

```
9      'parent_key' => 'uid',
10     )
11   );
12   $row = $db->field('username')->findall();
13 }
```

关于多对多关系需要特殊指定的属性如下

relation\_table : 多对多关联的中间表

以上示例产生的结果集如下

```
1 array
2 (
3   [0] => Array
4   (
5     [username] => 李四
6     [uid] => 474
7     [hd_group] => Array
8     (
9       [0] => Array
10      (
11        [group_id] => 2
12        [gname] => 管理员
13      )
14    )
15  )
16  [1] => Array
17  (
18    [username] => 马八
19    [uid] => 475
20    [hd_group] => Array
21    (
22      [0] => Array
23      (
24        [group_id] => 1
```

```
25           [gname] => 金牌用户
26       )
27   )
28 )
```

## 24\_6 关联添加

通过 HDPHP 框架可以快速完成关联数据的添加操作，程序员所要做的工作就是配置好关联操作模型参数即可，可以说是一劳永逸的事情

### 24-6-1 一对添加操作 HAS\_ONE

```
1 class IndexControl extends Control
2 {
3     function index()
4     {
5         $db = R('user');
6         $db->join = array(
7             'user_info' => array( // 关联表
8                 'type' => HAS_ONE, // 包含一条主表记录
9                 'foreign_key' => 'user_id', //user_info 表关联字段
10                'parent_key' => 'uid', //user 表主键
11            )
12        );
13        $_POST['username'] = '小华';
14        $_POST['password'] = 12345;
15        $_POST['user_info']['qq'] = 88888;
16        $_POST['user_info']['address'] = '北京后盾网总部';
17        $row = $db->insert();p($row);
18    }
19 }
```

通过上面的定义就可以完成关联的添加操作，生成的 SQL 如下

```
INSERT INTO hd_user(`username`,`password`)VALUES ('小华','12345')
```

```
INSERT INTO hd_user_info(`qq`,`address`)VALUES ('88888','朝阳区小营 5 号') 关联
```

## 更新

返回值是 array(2) { ['user']=> int(53) ['user\_info']=> int(58) } 即两个表最后自增加的 ID 组成的数组

## 24-6-2 一对多关联操作 HAS\_MANY

即在对一张表进行一条记录插入时，同时对其他表插入多条记录

注：从表数据必须是二维数组

### 示例一

```
1 class IndexControl extends Control
2 {
3     function index()
4     {
5         $db = R('user');
6         $db->join = array(
7             'news' => array( // 关联表
8                 'type' => HAS_MANY, // 包含多条主表记录
9                 'foreign_key' => 'user_id', // 关联字段
10                'parent_key' => 'uid', // 关联字段
11            )
12        );
13        $_POST = array(
14            'username' => '向军',
15            'password' => 'houdunwang.com',
16            'news' => array(
17                array(
18                    'title' => '标题',
19                    'con' => '内容',
20                )
21            )
22        );
23        $row = $db->insert();
24    }
```

## 24-6-3 BELONGS\_TO 关联插入

BELONG\_TO 可以理解为与 HAS\_MANY 或 HAS\_ONE 的反意，只是角色的互换

```

1 class indexControl extends Control
2 {
3     function index()
4     {
5         $db = R('news');
6         $db->join = array(
7             'channel' => array(
8                 'type' => BELONGS_TO,
9                 'parent_key' => 'cid',
10                'foreign_key' => 'cid',
11            ),
12        );
13        $data = array(
14            'title' => '标题',
15            'con' => '内容',
16            'channel' => array(
17                'cname' => '栏目名',
18            ),
19        );
20        $row = $db->insert($data);
21        p($row);
22    }
23 }

```

## 24-6-4 MANY\_TO\_MANY

支持一次 MANY\_TO\_MANY 关联方式的插入操作

```

1 class indexControl extends Control
2 {

```

```
3     function index()
4     {
5         $db = r('user');
6         $db->join = array(
7             'role' => array(
8                 'type' => MANY_TO_MANY, // 关联类型多对多关联
9                 'relation_table' => 'user_role', // 多对多的中间表
10                'parent_key' => 'uid', // 中间表与 user 关联字段
11                'foreign_key' => 'rid', // 中间表与 role 关联字段
12            )
13        );
14        $data = array(
15            'username' => '后盾向军', // 主表数据
16            'role' => array( // 关联表数据
17                'rname' => '管理员'
18            )
19        );
20        $db->add($data);
21    }
22 }
```

## 24\_7 关联更新

### 24-7-1 HAS\_ONE

#### 示例一

```
1 class indexControl extends Control{
2     function index() {
3         $db = R('user');
4         $db->join = array(
5             'news' => array(
6                 'type' => HAS_ONE,
7                 'foreign_key' => 'user_id',
8                 'parent_key'=>'uid'
```

```
9      ),
10     );
11     $data = array(
12       'uid' => 1,
13       'news' => array(
14         'title' => '后盾新闻',
15         'con' => '内容',
16       ),
17     );
18     $row = $db->save($data);
19   }
20 }
```

## 24-7-2 一对多 HAS\_MANY

一对多关联添加的使用方法与一对一是一样的

```
1 class indexControl extends Control
2 {
3   function index()
4   {
5     $db = R('user');
6     $db->join = array(
7       'news' => array(
8         'type' => HAS_MANY,
9         'foreign_key' => 'user_id',
10        'parent_key' => 'uid'
11      ),
12    );
13   $data = array(
14     'email' => 'houdunwangxj@gmail.com',
15     'news' => array(
16       array(
17         'title' => '标题',
```

```
18         'con' => '内容'
19     )
20     ),
21 );
22 $row = $db->where('username = "向军"')->limit(3)->order('uid desc')->save($data);
23 p($row);
24 }
25 }
```

## 24-7-3 BELONGS\_TO

```
1 class indexControl extends Control
2 {
3     function index()
4     {
5         $db = R('news');
6         $db->join = array(
7             'channel' => array(
8                 'type' => BELONGS_TO,
9                 'foreign_key' => 'cid',
10                'parent_key' => 'cid'
11            ),
12        );
13        $data = array(
14            'id' => '1',
15            'title' => '后盾网 HDPHP 框架更新新闻',
16            'con' => '本频道主要介绍后盾开源 HDPHP 框架进程',
17            'channel' => array(
18                'cname' => '后盾新闻频道',
19            ),
20        );
21        $row = $db->update($data);p($row);
22    }
23 }
```

通过以上关联操作完成 channel 表与 news 表的关联更新操作，因为 news 表中保存 channel 表的记录，设置 news 表为主表所以关联类型为 BELONGS\_TO

## 24\_8 关联删除

HDPHP 框架提供的关联删除操作，使多表删除操作变得异常简单

### 24-8-1 HAS\_ONE 关联删除

```
1 class indexControl extends Control
2 {
3     function index()
4     {
5         $db = R('user');
6         $db->join = array(
7             'user_info' => array(
8                 'type' => HAS_ONE,
9                 'foreign_key' => 'user_id',
10                'parent_key' => 'uid'
11            ),
12        );
13        $row = $db->del('uid>1');
14    }
15 }
```

删除 user 表中用户以及 news 表中的文章

### 24-8-2 HAS\_MANY 关联删除操作

HAS\_MANY 在关联删除中与 HAS\_ONE 意义一样，参数设置也一样

```
1 class indexControl extends Control
2 {
3     function index()
4     {
5         $db = R('channel');
6         $db->join = array(
7             'news' => array(
```

```
8      'type' => HAS_MANY,  
9      'foreign_key' => 'cid',  
10     'parent_key' => 'cid'  
11  
12     ),  
13 );  
14 $row = $db->del('cid>1');  
15 }  
16 }
```

删除栏目的同时，删除栏目下的文章

# 25. 多表视图

视图是虚表，是从一个或几个基本表（或视图）中导出的表，视图是原始数据库数据的一种变换，是查看表中数据的另外一种方式。可以将视图看成是一个移动的窗口，通过它可以看到感兴趣的数据。视图是从一个或多个实际表中获得的。

| 参数说明 | 说明                                       |
|------|--|
| type | 关联方式（常量）：INNER_JOIN LEFT_JOIN RIGHT_JOIN |
| on   | 关联条件即 SQL 中的 on 指令                       |

## 25\_1 使用视图时的参数配置

为了演示视图的使用，创建三张表如下

### 用户表 hd\_user

- 1 id 自增主键 ID
- 2 username 用户名
- 3 password 用户密码

### 用户信息表 hd\_user\_info

- 1 id 用户附加信息表自增主键 ID
- 2 email 用户邮箱
- 3 address 用户地址
- 4 qq 用户 QQ
- 5 user\_id 用户 ID

### 新闻表 hd\_news

- 1 id 新闻自增主键 ID
- 2 title 新闻标题
- 3 con 新闻内容
- 4 user\_id 发布新闻的用户 ID

## 25\_2 控制器中定义视图

- 1 class IndexControl extends Control
- 2 {

```
3     function index()
4     {
5         $db = V('user'); // 产生视图模型对象
6         $db->view = array( // 定义视图
7             'user_info' => array( // 定义 user_info 表规则
8                 'type' => INNER_JOIN, // 指定连接方式
9                 'on' => 'user.uid=user_info.user_id', // 关联条件
10            ),
11            'news' => array( // 定义 hd_news 表规则
12                'type' => LEFT_JOIN, // 指定连接方式
13                'on' => 'user.uid=news.user_id', // 关联条件
14            )
15        );
16        $row = $db->limit(3)->order('qq desc')->all();
17    }
18 }
```

按 QQ 号降序输出的前 3 条数据

## 25\_3 扩展模型中定义视图规则

Model 目录中创建扩展模型文件 userModel.php

```
1 class UserModel extends ViewModel
2 {
3     public $view = array(
4         'user_info' => array( // 定义 user_info 表规则
5             'type' => INNER_JOIN, // 指定连接方式
6             'on' => 'user.uid=user_info.user_id', // 关联条件
7         ),
8         'news' => array( // 定义 hd_news 表规则
9             'type' => LEFT_JOIN, // 指定连接方式
10            'on' => 'user.uid=news.user_id', // 关联条件
11        )
12    );
```

## 25\_4 只使用某个视图定义

定义了多个视图，但是使用时只想使用某个视图定义，可以通过执行 join 方法

```
$db->join(null)->all();// 不关联任何表
```

```
$db->join('news')->all();// 只关联 news 表
```

```
$db->join(array('news','user'))->all();// 关联 news 表与 user 表
```

# 26. Backup 数据库备份

数据是网站的生命，经常对数据表进行备份是很有必要的，HDPHP 框架提供了 Backup 数据表备份处理机制，使我们对数据的备份操作变得异常简单。

**注意：**

`$_SESSION['backup_dir']` 与 `$_SESSION['backup_fid']` 为备份类使用，开发中不要使用这两个 SESSION 变量。

`Backup::$error;` 错误信息记录（开发者可以根据此错误信息分析）

## 26\_1 备份数据库

备份数据库需要使用 Backup 类中的 backup 方法

语法：`Backup::backup($args = array())`

backup 参数必须为数组，参数说明如下

| 参数说明      | 说明                               |
|-----------|----------------------------------|
| database  | 备份的数据库，不填则使用配置项 C('DB_DATABASE') |
| table     | 需要备份的表                           |
| size      | 分卷大小，默认 200kb                    |
| dir       | 备份文件存放目录，不填则使用 C('DB_BACKUP')    |
| url       | 备份成功后跳转到的 url                    |
| step_time | 每次还原间隔时间，默认 500 毫秒               |

### 示例 1

```
Backup::backup(array('url' => U('show')) ;
```

使用配置项 C('DB\_BACKUP') 设置的目录进行备份，备份完成后跳转到当前控制器的 show 方法

### 示例 2

```
Backup::backup(array('dir'=> './backup/'.date('Ymdhis'), 'url' => __CONTROL__));
```

备份完成后跳转到当前控制器

## 26\_2 数据库还原

通过 Backup 类中的静态方法 recovery() 来还原已经备份的数据库

语法 : Backup::recovery(\$args)

| 参数说明         | 说明                |
|--------------|-------------------|
| dir ( 必须传参 ) | 备份数据存放的目录         |
| url          | 还原成功后跳转的 url 地址   |
| step_time    | 每次还原间隔时间 , 默认 1 秒 |

### 示例 1

```
Backup::recovery(array('dir'=>'backup/20130109011533','url'=>__CONTROL__));
```

还原目录 ./backup/20130109011533 中的备份数据 , 还原完毕后跳转到 index 控制器中的 show 方法

### 示例 2

```
Backup::recovery(array('dir' => './backup/20130109011533', 'step_time'=>3,'url' => 'http://www.houdunwang.com'));
```

每 3 秒请求一次服务器 , 当数据还原完毕后跳转到 http://www.houdunwang.com

## 27. 自动处理 create()

HDPHP 提供了自动验证、自动完成、自动填充等高效的数据处理机制，同时提供了 create() 方法，一次性对数据进行自动过滤、自动验证、自动完成、自动填充处理，使用 create() 控制器方法你需要对自动验证与自动完成及自动填充进行详细了解后才可以发挥 create() 方法的强大功能。

说明：

1. \$this->create() 方法就是依次执行 \$this->token(), \$this->validate(), \$this->auto(), \$this->map() 方法
2. 如果每个方法单独执行时，\$this->map() 必须放在最后执行

示例：

```
1 class ArticleControl extends Control{
2     public function add(){
3         $db=M('user');
4         if($db->create()){
5             $db->add();
6         }
7     }
8 }
```

# 28. 自动验证

验证用户信息是必不可少要做的工作，在 HDPHP 框架中程序员不用重复编写大量验证规则，只需要通过简单的配置就可以完成用户提交信息的验证处理。

验证规则是通过修改模型中的 validate 属性来完成的，validate 属性值为数组其中的每一个值是对表单中的某个字段的验证规则。

**注：必须执行 create 或 validate 模型方法，才会执行自动验证**

## 28\_1 验证规则设置语法

array('表单字段名' , '验证方法' , '错误信息' , '验证条件' , '验证时机')

**表单字段名**

**\$\_POST 中的字段**

**验证方法**

如验证方法为 check\_user，使用优先级如下

1. 扩展模型中的 check\_user 方法
2. check\_user 函数
3. 以上方法都不存在时，使用框架中的 validate.class.php 中同名方法进行验证

**错误信息**

验证失败时的错误信息，会记录到模型对象的 error 属性中

**验证条件**

| 值 | 说明            |
|---|---------------|
| 1 | 有这个表单项就验证（默认） |
| 2 | 必须验证的表单项      |
| 3 | 如果表单不为空才验证    |

**验证时机**

| 值 | 说明    |
|---|-------|
| 1 | 插入时验证 |
| 2 | 更新时验证 |

| 值 | 说明            |
|---|---------------|
| 3 | 任何动作都进行验证(默认) |

## 28\_2 验证示例演示

### 示例一 ( 通过模型的 validate() 方法验证 )

```

1 $db = M('user');
2 $db->validate = array(
3     array('username', 'nonull', '用户名不能为空',2,3)
4 );
5 if(!$db->validate()){
6     $this->error($db->error);
7 }
```

以上代码是对用户表单 username 进行验证，验证规则是不为空，如果验证不通过提示信息为 '用户名不能为空'，第 4 个参数为 2 表示必须进行验证，是不是很简单

注：验证不通过时 \$this->error 保存错误提示信息

### 示例二 ( 扩展模型中的定义 )

```

1 class userModel extends Model {
2     public $validate = array(
3         array('username', 'nonull', '用户名不能为空', 2,3),
4         array('email', 'email', '邮箱格式不正确',1,3),
5         array('age', 'maxlen:60', '年龄不能超过 60', 3,2)// 使用 validate 类方法验证
6     );
7 }
```

username 表单必须进行验证，不能为空

email 表单时验证是否为邮箱

age 表单不为空时进行验证，通过框架 Validate.class.php 类中的 maxlen 方法验证，验证最大数不能超过 60

## 28\_3 Validate 验证类

方法说明：

| 参数       | 说明                              |
|----------|---------------------------------|
| nonull   | 不能为空                            |
| email    | 验证是否为邮箱                         |
| http     | 网址验证                            |
| tel      | 验证固定电话                          |
| phone    | 验证手机                            |
| user     | 验证用户名 (必须以字母开始) 如 :user:5,20    |
| maxlen   | 最大长度 (支持中文), 如 : maxlen:10      |
| minlen   | 最大长度 (支持中文), 如 : minlen:10      |
| num      | 数字范围, 如 : num:20,60             |
| regexp   | 自定义正则, 如 : regexp:/^\d{5,20}\$/ |
| confirm  | 验证两个字段相等, 如 : confirm:password2 |
| china    | 验证中文                            |
| identity | 身份证验证                           |

## 28\_4 验证函数传递参数

有些验证函数需要传递参数进行验证，在函数名与参数间用冒号分隔，如果参数是多个用逗号分隔

示例一 ( 验证用户名长度必须大于 6 小于 20 )

```
array('username', 'user:6,20', '用户名长度必须大于 6 小于 20', 2,3)
```

示例二 ( 年龄必须大于 20 小于 60 )

```
array('age','num:20,60','年龄必须大于 20 小于 60',2,3)
```

示例三 ( 自定义正则 )

```
array('tel','regexp:/^\d{11}/','手机号格式不正确',2,3)
```

## 示例四 ( 验证两次密码一致 )

```
array('password','confirm:password2','两次密码不一致',2,3)
```

# 28\_5 自定义验证方法

## 28-5-1 自定义验证函数

### 示例

```
1 function isadmin($name, $value, $msg, $arg) {  
2     if ($value != 'admin') {  
3         return $msg;  
4     }  
5     return true;  
6 }
```

### 说明：

\$name 为字段名 \$value 为字段值 \$msg 为错误信息 \$arg 为参数

## 28-5-2 在模型中创建验证方法

在模型中创建验证方法，优先级最高

```
1 class UserModel extends Model {  
2     public $validate = array(  
3         array('username','isadmin','不是管理员',2,3)  
4     );  
5     public function isadmin($name, $value, $msg, $arg) {  
6         if ($value != 'admin') {  
7             return $msg;  
8         }  
9         return true;  
10    }  
11 }
```

# 29. 自动完成

自动完成是在模型层对数据进行自动处理的操作过程。

注：必须执行 create 或 auto 模型方法，才会执行自动完成

## 29\_1 自动完成语法

```
array('数据字段名','处理方法','方法类型','验证条件','处理时机')
```

### 参数说明

| 参数    | 说明   |
|-------|--|
| 表单字段名 | 字段名  |
| 处理方法  | 方法或函数名   |
| 方法类型  | method: 自定义的模型方法 function: 函数 string: 字符串 (默认) |
| 验证条件  | 1 有这个表单项就处理 (默认)<br>2 必须处理的表单<br>3 如果表单不为空才处理  |
| 处理时机  | 1 插入时处理<br>2 更新时处理<br>3 任何动作都进行处理 (默认)         |

### 示例 1

```
1 $db = M('news');
2 $db->auto=array(
3     array('begin_time','strtotime','function',2,1),
4 );
```

插入时使用 strtotime() 函数处理 begin\_time 字段

### 示例 2

```
1 $db = M('news');
2 $db->auto=array(
3     array('click','intval','function',1,2)
4 );
```

更新时对 click 字段执行 intval() 函数

### 示例 3

```
1 class NewsModel extends Model{  
2     public $auto=array(  
3         array('passwd','_md5','method',1,2),  
4     );  
5     public function _md5($con){  
6         return md5(serialize($con));  
7     }  
8 }
```

更新时对 passwd 字段执行 \_md5() 方法

# 30. 字段自动映射

开发中将字段暴露给前台用户会带来潜在的安全隐患，后盾 HDPHP 框架提供方便的字段映射策略用于提高系统安全性，使用方法也很简单

注：必须执行 create 或 map 模型方法，才会执行自动映射

## 前台

```
1 <form action='__CONTROL__/add' method='post'>
2   用户名 :<input type='text' name='yonghu' />
3   密码 : <input type='password' name='mima' />
4   <input type='submit' />
5 </form>
```

## 后台

测试表 user 的字段为 username 及 password

```
1 $db=M('user');
2 $db->map=array(
3   'yonghu'=>'username',
4   'mima'=>'password'
5 );
6 if ($db->create()) {
7   if ($db->join(false)->add()) {
8     $this->success('添加用户成功');
9   } else {
10    $this->error('用户添加失败');
11  }
12 }
```

以上代码将前台表单 yonhu 应用为表字段 username，将表单 mima 映射为 password，大大提高应用安全性，可以在扩展模型中定义字段映射规则

# 31. Token 令牌

为了防止灌水等恶意提交行为，HDPHP 框架提供了 Token( 令牌 ) 机制，为每一个用户的表单项记录一个唯一的 Token ,通过这个 Token 完成数据的安全校验来阻址恶意提交行为。

注：必须执行 create 或 token 模型方法才会执行令牌验证

## 31\_1 Token 配置项

| 参数         | 说明                              |
|------------|---------------------------------|
| TOKEN_ON   | 是否开启令牌功能 ( 可以在控制器中通过 C() 函数设置 ) |
| TOKEN_NAME | 令牌 Token 的表单 name 名称            |

只要开启 TOKEN\_ON 配置项，系统会为表单生成一个令牌。

## 31\_2 Token 验证操作

以下示例为开启 TOKEN\_ON 配置项

### 自动验证

再执行 Model 的 insert 方法时，Model 会自动进行验证处理，示例如下

```
1 $db= M('user');
2 if($db->create()){
3     if(!$db->add()){
4         echo $db->error;
5     }
6 }
```

执行 create 方法时 Token 将自动进行验证，错误将记录到 Model 的 error 属性中

### 用户验证

```
1 $db= M('user');
2 if(!$db->token()){
3     echo $db->error;
4 }
```

可以通过 token() 方法进行验证

# 32. 验证码

在很多场景中会用到验证码功能，比如说用户注册等，HDPHP 框架产生验证码非常简单

验证码需要很多配置，比如说宽度、长度、文字颜色、背景颜色、文字大小、字体等，使用 HDPHP 框架的验证码不用每个参数都做设置，因为所有配置项已经在基础配置项中做好了设置。如果你要定义验证码则需要设置相关参数。

## 设置方式有几种

- a. 第一种是在实例化对象时指定构造函数参数
- b. 第二种是修改对象属性方式
- c. 第三种是修改配置项，采用哪种方式都可以达到自定义验证码的功能。

## 影响验证码的配置项

- 1 'FONT' => 'font.ttf', // 文体
- 2 'CODE\_STR' => '1234567890abcdefghijklmnopqrstuvwxyz', // 验证码种子
- 3 'CODE\_WIDTH' => 80, // 验证码宽度
- 4 'CODE\_HEIGHT' => 25, // 验证码高度
- 5 'CODE\_BG\_COLOR' => '#CCE8CF', // 验证码背景颜色
- 6 'CODE\_LEN' => 4, // 长度
- 7 'CODE\_FONT\_SIZE' => 18, // 字体大小
- 8 'CODE\_FONT\_COLOR' => "", // 字体颜色

## 示例一

```
$code = new code( 宽度 , 高度 , 背景颜色 , 文字颜色 , 验证码字数 , 文字大写 );  
1 $code = new code(200,30,'#dcdcdc','#f00',8,22);  
2 $code->show();
```

## 示例二

以配置文件中的默认参数产生验证码

```
1 $code = new code();  
2 $code->show();
```

## 示例三

以修改配置文件方式产生验证码

```
1 C('CODE_WIDTH',100); // 修改宽度
```

```
2 C('CODE_HEIGHT',30);// 个性高度
3 C('CODE_len',4);// 验证码长度
4 $code = new code();// 实例化验证码对象
5 $code->show();// 生成验证码
```

#### 示例四 ( 修改对象属性 )

```
1 $code = new code();// 实例化对象
2 $code->width=100;// 修改宽度
3 $code->height=30;// 修改高度
4 $code->fontColor='#f00000';// 修改验证码颜色
5 $code->show();// 显示验证码
```

# 33. 分页处理类

HDPHP 框架提供了快速的分页操作，及方便的分页定制功能。如果没有特别指定分页处理方式，将采用配置项目的设置，影响分页的配置项包括。

## 33\_1 构造函数

```
function __construct($total, $row = "", $pageRow = "", $desc = "",  
$setSelfPage = "", $customUrl = "", $pageNumLabel = '{page}')
```

| 配置项                      | 说明  |
|--------------------------|---|
| \$total                  | 分页 GET 变量，默认为 page  |
| \$row                    | 页码显示数量  |
| \$pageRow                | 每页显示条数  |
| \$desc                   | 分页文字设置如：array('pre' => '上一页', 'next' => '下一页', 'first' => '首页', 'end' => '尾页', 'unit' => '条') |
| \$setSelfPage            | 当前页   |
| \$customUrl              | 自定义 url   |
| \$pageNumLabel           | 页码变量，默认为 {page}   |
| static \$staticTotalPage | 总页数   |
| static \$staticUrl       | 当前 url  |
| static \$fix             | 静态后缀如 .html   |
| static \$pageNumLabel    | 替换标签  |

## 33\_2 获得分页项数组

通过 HDPHP 框架的分页类提供的 get\_all() 方法可以获得所有组合分页所需要的显示内容，使用方法如下：

```
1 $db = M('demo');
```

```
2 $total = $db->count();
3 $page = new page($total);
4 $arr = $page->getAll();
5 p($arr);
6 echo $page->show();
```

## 33\_3 分页样式

\$this->show() 方法为显示分页列表，框架提供 5 种分页风格，使用 \$this->show()  
示例

```
1 $page = new Page(100,10);
2 $page->show(3);// 以第 3 种分页风格进行显示
```

## 33\_4 SQL 语句所需要 LIMIT

使用 page::limit() 方法可以返回 sql 所需要的 limit 部分，如果传参 Page::limit() 返回值为数组，Page::limit(true); 返回为字符串。

示例

```
1 $page = new Page(100,10);
2 $page->limit(true);// 返回字符串
3 $page->limit();// 返回数组
```

## 33\_5 分页示例演示

示例一 ( 根据配置项提取分页数据 )

```
1 $db = M('demo');
2 $total = $db->count();
3 $page = new page($total);// 传入总记录数，用于计算出分页数
4 $row = $db->findAll($page->limit());// 查询当前页的数据
5 echo $page->show();// 显示分页页码
```

示例二

```
1 $db = M('demo');
2 $total = $db->count();
3 $page = new page($total,10,6,2,array('pre'=>' 上一层 ','next'=>' 下一层 '));
4 $row = $db->findAll($page->limit());
```

```
5 echo $page->show();
```

设置每页显示 10 条记录 , 页码数量是 6 个 , 界面显示风格为 2 , 同时修改上一页为上一层 , 下一页为下一层

## 33\_6 自定义分页 Url

在全站静态等场景 , 需要自定义分页 url 地址 , Page 类可以轻松应付

### 示例

```
1 $page = new Page(88,10,6,"",'http://localhost/{page}.html','{page}');
```

```
2 echo $page->show(2);
```

# 34. 图像处理类

HDPHP 框架提供完善的图像处理类，包括图片缩略图处理，图片水印处理等，其它图片处理方式会不断添加。

## 34\_1 图像缩略图处理

由于前台上传的图片会很大，往往我们不希望把原图进行显示，而是只显示一个小的缩略图，这样用户浏览网站加载会更快些，减少额外的带宽占用

影响缩略图处理的配置项

- 1 'THUMB\_ON' => 1, // 缩略图开关
- 2 'THUMB\_PREFIX' => "", // 缩略图前缀
- 3 'THUMB\_ENDFIX' => '\_thumb', // 缩略图后缀
- 4 'THUMB\_TYPE' => 1, // 生成缩略图方式
- 5 'THUMB\_WIDTH' => 150, // 缩略图宽度
- 6 'THUMB\_HEIGHT' => 150, // 缩略图高度
- 7 'THUMB\_PATH' => 'smail', // 图片上传路径示例一

**生成缩略图方式 THUMB\_TYPE 项可选值如下**

- 1: 固定宽度 高度自增      2: 固定高度 宽度自增      3: 固定宽度 高度裁切  
4: 固定高度 宽度裁切      5: 缩放最大边      6: 缩略图尺寸不变，自动裁切图片

**参数与调用类方法的优先级问题**

对配置文件的修改将直接影响水印处理的效果，如果与调用水印处理方法时设置的参数发生相同设置的冲突，则调用类方法的设置项优先级最高

**图像类关于缩略图配置的对象属性**

- 1 private \$thumbOn;// 是否开启缩略图功能
- 2 public \$thumbType;// 生成缩略图的方式
- 3 public \$thumbWidth;// 缩略图的宽度
- 4 public \$thumbHeight;// 缩略图的高度
- 5 public \$thumbEndFix;// 生成缩略图文件名后缀
- 6 public \$thumbPreFix;// 缩略图文件前缀

## 缩略图类方法语法

```
public function thumb($img, $outFile = "", $thumbWidth = "", $thumbHeight = "",  
$thunbType = "", $path = "")
```

1 thumb( 原文件 , 缩略图宽度 , 缩略图高度 , 缩略图处理类型 , 存放目录路径 , 缩略图文件 )

缩略图目录路径如果不指定 , 将使用配置文件中的目录 , 如果目录不存在 HDPHP 框架会自动创建

返回值为生成的缩略图路径

### 示例一 ( 利用初始值进行缩略图处理 )

```
1 $img = new image();  
2 echo $img->thumb('r8.jpg','cc.jpg');
```

### 示例二

```
1 $img = new image();  
2 echo $img->thumb('r8.jpg',md5('r8.jpg').'.jpg',200,200,2,'upload/cc');
```

对图片 r8.jpg 进行缩略图处理 , 缩略图文件为名 md5 后的文件

上传目录为 upload/cc

如果目录不存在系统会自动创建

缩略图宽度为 200 , 高度 200

以第 2 种方式产生缩略图

## 34\_2 图像水印处理

HDPHP 框架可以方便的对图片进行水印处理 , 同时上传的文件如果是图片也会自动进行缩略图和水印处理 , 如果对上传图片不想进行自动处理可以通过修改配置文件做到 , 具体请看上传类部分

### 对水印影响的配置项

```
1 'WATER_ON' => 1, // 水印开关  
2 'WATER_FONT'=>PATH_HD.'/data/font/ggbi.ttf',// 水印字体  
3 'WATER_IMG' => PATH_HD.'/data/water/water.png', // 水印图像  
4 'WATER_POS' => 9, // 水印位置  
5 'WATER_PCT' => 60, // 水印透明度  
6 'WATER_QUALITY' => 80, // 水印压缩质量  
7 'WATER_TEXT' => '后盾网 HDPHP 框架系统', // 水印文字
```

```
8 'WATER_TEXT_COLOR' => '#f00f00', // 水印文字颜色  
9 'WATER_TEXT_SIZE' => 12, // 水印文字大小
```

对配置文件的修改将直接影响水印处理的效果，如果与调用水印方法时设置的参数发生相同设置的冲突，则调用类方法的设置项优先级最高

## 图像类关于水印的属性

```
1 private $waterOn;// 是否应用水印  
2 public $waterImg; // 水印图片  
3 public $waterPos; // 水印的位置  
4 public $waterPct;// 水印的透明度  
5 public $waterQuality;// 图像的压缩比  
6 public $waterText;// 水印文字内容  
7 public $waterTextSize;// 水印文字大小  
8 public $waterTextColor;// 水印文字的颜色  
9 public $waterTextTont;// 水印的文字的字体
```

## 图像类水印方法语法

public function water( 原图 , 另存的图像 , 水印图像 , 水印位置 , 水印文字 , 透明度 )

除第一个参数外，其他参数可以不用设置，如果不设置会以配置文件为依据处理

### 水印位置

HDPHP 框架水印位置为 9 宫格布局，即 1 为左上、2 为顶部居中、3 为顶部居右、4 为左侧垂直居中、5 为正中心、6 为右侧垂直居中、7 为左下、8 为下中、9 为下右。

可以修改配置文件或者调用水印方法时指定位置参数来改变水印添加的位置。

### 示例一

```
1 $img = new image();  
2 $img->water('houdunwang.jpg');
```

对图像 houdunwang.jpg 进行水印处理，水印设置参数以配置文件为主

### 示例二

```
1 $img = new image();  
2 $img->water('houdunwang.jpg','houdunwang_2.jpg',2);
```

对 houdunwang.jpg 图片加水印

另存为 houdunwang\_2.jpg

水印位置为 2，即顶部中间位置

# 35. 上传类

HDPHP 框架有非常方便的上传处理机制，可以指定上传类型及每种文件类型所允许的上传大小，上传类会返回所有上传成功的文件列表，包括上传成功的文件路径结构，如果是图片还会根据配置文件生成水印或缩略图，生成的缩略图也会在返回数组中包含。

HDPHP 框架上传类可以对图片数据进行安全检测，也就是说如果一个 .php 文件被改成了 .jpg 的扩展名，HDPHP 框架可以判断出，并将此文件为恶意图片信息记录到属性 error 中。

## 注意：

1. 如果上传一个文件，返回一维数组
2. 如果上传多个文件，返回二维数组

## 上传表单设置

```
1 <form action='__CONTROL__/upload' method='post' enctype='multipart/form-data'>
2 <input type='file' name='upfile[]' />
3 <input type='file' name='upfile[]' />
4 <input type='submit' />
5 </form>
```

## 说明

1. form 标签 enctype 必须设置，否则无法上传
2. 如果上传多文件，建议使用 upfile[] 数组方式上传

## 对上传文件有影响的配置项

```
1 'UPLOAD_EXT_SIZE' => array('jpg' => 200, 'jpeg' => '', 'gif' => '', 'png' => 200000, 'bm' => '',
  'zip' => '', 'txt' => '', 'rar' => '', 'doc' => ''), // 上传类型与大小
2 'UPLOAD_PATH' => ROOT_PATH.'upload/'.date('ymd'), // 上传路径
3 'UPLOAD_IMG_DIR' => 'img', // 图片上传目录名
```

### UPLOAD\_EXT\_SIZE 上传允许类型与大小，以数组形式表示

array('jpg' => 20000, 'jpeg' => 35000) 表示允许 jpg 类型上传大小为 20000 字节，  
允许 jpeg 上传大小为 35000 字节

## 文件上传类属性

```
1 public $path;// 上传路径
2 public $exts=array(); // 上传类型
```

```
3 public $thumbOn = 1; // 缩略图处理
4 public $thumb = array(); // 缩略图参数
5 public $waterMarkOn = 1; // 是否加水印
6 public $uploadedFile = array(); // 上传成功文件信息
7 public $error; // 错误信息
```

### 示例一（根据配置文件设置进行上传）

```
1 $upload = new upload();
2 $uplofiles = $upload->upload();
```

返回值为所有成功的上传文件组成的数组

### 示例二（通过构造函数配置上传规则）

```
1 $upload = new upload('hd', array('jpg', 'jpeg', 'png'), 50000, 1, 1, array(300, 80, 3));
2 $uplofiles = $upload->upload();
```

指定上传目录为 hd，允许上传类型为 jpg,jpeg,png 允许上传大小为 50000 字节，添加水印，生成缩略图，生成缩略图大小为宽 300，高 80，以固定宽度 高度裁切的方式 生成缩略图

开始上传文件，并返回成功上传文件组成的数组

**上传成功后返回数组说明**

| 属性       | 说明          |
|----------|-------------|
| path     | 上传成功的文件路径   |
| basename | 文件名         |
| filename | 主文件名（不带扩展名） |
| size     | 文件大小        |
| ext      | 文件扩展名       |
| uptime   | 上传时间        |
| image    | 是否为图像       |

## 36. URI 类

HDPHP 框架 URI 处理类为底层类 ,由系统运行时使用 ,用户不需要调用此类的方法功能 ,  
URI 类主要是为了实现完善的 PATHINFO 路径处理机制 ,伪静态机制及 URL 路由处理规则 ,  
程序员不需要调用这个类来完成工作。

# 37. Email 邮件处理

在开发过程中经常需要使用到发送邮件功能，比如说用户注册邮箱的验证，网站信息的邮件通知等等，这些都可以通过 HD PHP 框架的邮件处理功能实现，实现手段也非常简单。

注：需要开启配置项 `extension=php_sockets.dll`

影响邮件发送的配置项说明

| 配置项                         | 说明   |
|-----------------------------|--|
| <code>EMAIL_USERNAME</code> | 发送邮件邮箱用户名，如 <code>houdunwangdemo@126.com</code>                          |
| <code>EMAIL_PASSWORD</code> | 发送邮件邮箱密码   |
| <code>EMAIL_HOST</code>     | 邮箱服务器 smtp 地址，如： <code>smtp.gmail.com</code> ， <code>smtp.126.com</code> |
| <code>EMAIL_PORT</code>     | 邮箱服务器 smtp 端口，大部分为 25 如 126，gmail 端口为 465(注意)                            |
| <code>EMAIL_SSL</code>      | 服务器是否采用 SSL 126 等值为 <code>false</code> google 必须为 <code>true</code>      |
| <code>EMAIL_CHARSET</code>  | 字符集设置，中文乱码就是这个没有设置好，如 <code>utf8</code>                                  |
| <code>EMAIL_FORMMAIL</code> | 发送人发件箱显示的邮箱地址  |
| <code>EMAIL_FROMNAME</code> | 发送人发件箱显示的用户名   |

通过 126 发送邮件

配置项

```
1 <?php
2 class IndexControl extends Control{
3     function sendMail(){
4         $config = array(
5             'EMAIL_USERNAME'=>'houdunwang@126.com',
6             'EMAIL_PASSWORD'=>'*****',
7             'EMAIL_HOST'=>'smtp.126.com',
8             'EMAIL_PORT'=>25,
9             'EMAIL_SSL'=>false,
10            'EMAIL_CHARSET'=>'utf-8',
```

```
11     'EMAIL_FORMMAIL'=>'houdunwang@126.com',
12     'EMAIL_FROMNAME'=>'后盾网'
13 );
14 Mail::send('houdunwangxj@gmail.com','后盾向军','欢迎注册后盾网','感谢你注册后盾网，  
请点击链接完成注册 <a href='http://www.houdunwang.com'>http://www.houdunwang.com</a>');
15 }
16 }
```

## 使用 GMAIL 邮箱发送邮件

使用 Gmail 发送邮件只需注意以下几点，其他配置与 126 发送邮件相同

1. 修改 EMAIL\_SSL 为 1
2. 修改 EMAIL\_PORT 为 465

# 38. 模板视图

## 38\_1 基础知识

hd 模版引擎是一个 php 模版引擎。更准确的说，它分开了逻辑程序和外在的内容，提供了一种易于管理的方法。可以描述为应用程序员和美工扮演了不同的角色，因为在大多数情况下，他们不可能是同一个人。例如，你正在创建一个用于浏览新闻的网页，新闻标题，标签栏，作者和内容等都是内容要素，他们并不包含应该怎样去呈现。模板设计者们编辑模板，组合使用 html 标签和模板标签去格式化这些要素的输出 (html 表格，背景色，字体大小，样式表，等等)。有一天程序员想要改变文章检索的方式 (也就是程序逻辑的改变)。这个改变不影响模板设计者，内容仍将准确的输出到模板。同样的，哪天美工吃多了想要完全重做界面，也不会影响到程序逻辑。因此，程序员可以改变逻辑而不需要重新构建模板，模板设计者可以改变模板而不影响到逻辑。

HD 模版引擎是编译型模版引擎，模版文件只编译一次，以后程序会直接采用编译文件，效率非常高。

## 38\_2 定义模版

定义模版时需要参数配置文件，模版的目录和模版文件的扩展名都可以在配置文件中指定，但是 HD PHP 框架的配置文件是分几层的，所以你要注意配置文件的优先级，因为优先级高的配置文件会覆盖优先级低的，所以这点你要注意。

### 影响模版的配置项：

- 1 'TPL\_ENGINE' => 'hd', // 提供 HD 及 SMARTY 模版引擎，推荐使用 HD 模版引擎
- 2 'TPL\_FIX' => '.html', // 模版文件扩展名
- 3 'TPL\_TAG\_LEFT' => '<', // 模版左标签
- 4 'TPL\_TAG\_RIGHT' => '>', // 模版右标签
- 5 'TPL\_DIR' => 'tpl', // 模版文件的目录名
- 6 'TPL\_TAGS'=>array('tags','hd\_tags'),// 扩展标签库参数必须为数组
- 7 'TPL\_STYLE' => "", // 如果有多风格模版时，这里添上目录名 那么路径结果就会变成 TPL\_PATH/TPL\_STYLE 形式
- 8 'TPL\_COMPILE' => true, // 开启模板编译
- 9 'TPL\_CACHE\_ON'=>0,// 是否开启缓存
- 10 'TPL\_CACHE\_TIME' => "", // 模版缓存时间

## 38\_3 assign 向视图层分配内容

通过控制器向模版视图层分配内容非常简单，不用过多设置，只需要在控制器中使用 assign 方法就可以了。

### 示例 1

```
1 $user = array(  
2     array( 'name'=>'李四', 'age'=>33),  
3     array( 'name'=>'后盾', 'age'=>6,)  
4 );  
5 $this->assign('username',$user);  
6 $this->display('index.html');
```

通过以上代码向视图 index.html 中分配变量 \$user, 在视图层调用时需要通过 username, 所以要注意控制层的变量名与模版中的变量名是不同的。

## 38\_4 display 显示内容

HDPHP 框架使用原创的编译型模板引擎，操作非常简单同时功能强大，提供强大的模版缓存机制，简洁的自定义标签制作，总之 HDPHP 框架开发思想以简单、开放、易扩展为主。

HDPHP 框架调用模版非常方便只需要通地 \$this->diplay( 模板文件 ) 即可，但功能远不止此，比如可以非常方便的控制每个页面的缓存时间具体使用方法如下：

`$this->display($tplFile,$cacheTime=null,$cachePath=null,$contentType='text/html',$charset','=',$show=true);` 各参数说明如下

| 参数            | 说明                                    |
|---------------|---------------------------------------|
| \$tplFile     | 模版文件，不传参将使用函数名同名文件                    |
| \$cachePath   | 缓存文件路径，不传参将使用 CACHE_PATH 常量值          |
| \$cacheTime   | 缓存时间如果不传参时读取配置项 C('TPL_CACHE_TIME') 值 |
| \$contentType | 文件类型 ( 默认 text/html)                  |
| \$charset     | 字符集默认取配置文件 C('CHARSET') 设置            |

| 参数     | 说明                           |
|--------|------------------------------|
| \$show | 是否输出到浏览器，传入 false 为返回显示内容字符串 |

`$this->display('web');`

以当前应用中的 web.html 做为模板视图

`$this->display('user/add');`

以 user 控制器视图目录中的的 add.html 作为视图

`$this->display('member/user/add');`

以 member 应用中的 user 控制器视图目录的 add.html 为做视图层

以上实例都假设配置文件设置的模板扩展名为 .html

`$this->display('b',30);`

以上表示操作 b.html 模板并将结果缓存 30 秒中，使用缓存可以大大加快响应速度

`$this->display('b',30,CACHE_PATH.'tpl');`

以上表示操作 b.html 模板并将结果缓存 30 秒中，并设置缓存文件存放目录

## 38\_5 fetch() 获得模板解析数据

HDPHP 除了可以通过 display() 显示模板解析数据，还可以通过 fetch() 方法获得解析数据，比如可以做缓存使用。

### 示例 将模板解析数据缓存 10 分钟

```

1 <?php
2 class IndexControl extends Control{
3     function index(){
4         $cacheData = S('index_index');
5         if($cacheData){
6             echo $cacheData;// 输出缓存
7         }else{
8             $data = $this->fetch();// 获得显示数据
9             S('index_index',$data,600)
10    }
11   }
12 }
```

## 38\_6 isCache() 缓存是否失效

如果设置了模板缓存，可以通过 HDPHP 框架的 isCache() 方法检测缓存是否失效，如果失效则重新进行缓存处理，我们来看示例演示：

### 示例 1 HDPHP 框架模板引擎使用方式

```

1 <?php
2 class indexControl extends Control{
3     function index(){
4         if(!$this->isCache()){ // 缓存是否失效
5             $db=M('user');
6             $row = $db->limit(10)->all();
7             $this->assign('row',$row);
8         }
9         $this->display('index',60); // 指定缓存时间为 60 秒
10    }
11 }
```

如果缓存文件不存在或者缓存文件失效则重新查找数据，如果缓存没有过期则读取缓存内容不进行 SQL 操作，通过视图缓存控制可以大大加快加载速度有效解决大并发时的数据加载问题

## 38\_7 读取系统变量

|                       |                    |
|-----------------------|--------------------|
| {\$hd.const.CONTROL}  | 读取系统中的常量           |
| {\$hd.session.uid}    | 读取 SESSION 中的值     |
| {\$hd.cookie.sid}     | 读取 COOKIE 中的值      |
| {\$hd.get.page}       | 读取 GET 中的值         |
| {\$hd.post.cid}       | 读取 POST 中的值        |
| {\$hd.config.db_host} | 读取配置项的值            |
| {\$hd.language,title} | 根据语言包读取内容（详见多语言支持） |

## 38\_8 变量调节器

变量调节器使用非常简单高效，你可以使用任何 PHP 系统函数做为变量调节器，当然也可以使用任何自定义函数进行设置，只要把函数放到应用组目录或应用目录中的 libs 目录下即可文件名可以任意起，系统会自动加载。

HDPHP 模版引擎变量调节器非常灵活，任何函数都可以做为变量调节器使用，下面我们来看几个例子，你就清楚了。

### 38-8-1 使用任意 PHP 函数

在 HDPHP 框架中可以使用任意 PHP 函数做为变量调节器使用

#### 示例

```
{$data|strtoupper}
```

对变量应用系统函数 strtoupper，以上代码编译的脚本为

```
<?php echo strtoupper($data) ;?>
```

### 38-8-2 变量调节器链式操作

#### 示例

```
{$data|substr:2,8|strtolower}
```

对变量应用多个个函数，只要通过 | 分隔函数名就可以了，以上代码生成 PHP 代码为

```
<?php echo strtolower(substr($data,2,8)) ;?>
```

### 38-8-3 hd\_substr 截取字符串

本函数适应用所有字段，用于截取字符串。

#### 示例 1：

```
{$field.title|hd_substr:10}
```

截取 10 个字符标题

#### 示例 2：

```
{$field.title|hd_substr:10,'...')}
```

截取 10 个字符标题，后面以 ... 结束

### 38-8-4 hd\_date 日期处理函数

本函数主要用于格式化日期

### 示例 1 :

```
{$field.addtime|hd_date}
```

输出 “年 - 月 - 日” 格式的时间

### 示例 2 :

```
{$field.updatetime|hd_date:'y-m-d h:i:s'}
```

输出 “年 - 月 - 日 时 : 分 : 秒” 格式的时间

### 示例 3

```
{$data|date:'y-m-d h:i:s',@@@}
```

对变量应用函数 date ,@@@ 表示变量 \$data 的位置 ,如果变量不在函数参数第一个位置 ,可以通过 @@ 符号随便指定位置。

## 38-8-5 模板中使用函数

HDPHP 框架支持在模板视图中使用任意函数

注意 : 函数名前加 | , 如 { | substr:'houdunwang.com',0,2 }

### 示例 1

```
1 { |U:'index','username= 向军 &sex= 男 '}
```

以上代码书写在视图中 , 执行 U 函数传递 1 个参数 , 最后生成的 url 为 http://localhost/index.php/Index/index/username/ 向军 /sex/ 男

### 示例 2

控制器方法向视图分配变量 :

```
1 $data = array('webname'=>'后盾网','url'=>'houdunwang.com');  
2 $this->assign('data',$data);
```

在视图中执行 p() 函数输出变量内容

```
1 { |p:$data}
```

## 38-8-6 模板中使用 U() 方法

我们并不希望在程序中将 URL 写全 , 因为有时环境不支持 PATHINFO , 这种情况下 , 我们要将整个网站的 URL 重新构建 , 针对这种问题我们可以通过在视图中使用 U() 方法分配 URL , 当环境不同或者更改配置文件中有关 URL 的配置时 , 系统会动态自动识别 , 省掉更改大量 URL 的烦恼 , 同时 HDPHP 框架支持使用任意函数 , 所以不只是 U() 方法可以在

视图中使用，其实任意函数均可以视图中使用。

#### 示例 1

```
1 {|U:'index','username= 向军 &sex= 男'}
```

以上代码写在视图中，最后生成的 url 为 <http://localhost/index.php/Index/index/?username=向军&sex=男>

#### 示例 2 使用变量

```
1 {|U:'index',array('webname'=>$web)}
```

在视图中使用 U() 函数生成 URL，其中 \$web 为控制器分配 (assign) 的变量

#### 示例 3 使用括号调用函数

```
1 { |U('index',$web)}
```

在视图中使用 U() 函数生成 URL，其中 \$web 为控制器分配 (assign) 的变量

#### 示例 4 只使用一个参数

```
1 { |U('index/index/uid/$web') } 或 { |U('index/index/uid/'. $uid) }
```

在视图中使用 U() 函数生成 URL，其中 \$uid 为控制器分配 (assign) 的变量，注意要解析变量必须使用双引号

### 38-8-7 default 默认值

当变量为空或者未分配的时候，将由给定的默认值替代输出

```
{$houdunwang|default:'后盾网'}
```

当变量 \$houdunwang 不存在时显示后盾网

## 38\_9 系统模版标签使用

### 38-9-1 less 标签

引入 less.js 文件用于快速编写 css 文件

#### 示例：

```
<less/>
```

### 38-9-2 jquery 标签

加载 jquery 库

#### 语法

```
<jquery/>
```

通过调用 `<jquery/>` 标签会自动生成 `<script type='text/javascript' src='http://localhost/hdcms/HD/org/jquery-1.7.2.min.js'></script>` 这样就可以使用 jquery 库了

注 : 使用 `<hdjs/>` 标签会自动调用执行 `<jquery/>` 标签

### 38-9-3 jsconst 标签

开发中经常需要在 js 脚本中使用系统常量如 `_COTROL_`, `_METH_` 等 , 这时可以通过在使用常量前调用 `<jsconst/>` 标签 , 通过调用 `<jsconst/>` 标签后系统会自动根据常量名创建 js 变量 , 不过在使用时不要加前后的 `_`

示例 :

```
1 <html>
2 <head>
3   <jsconst/>
4 </head>
5 <body>
6   <script type='text/javascript'>
7     alert(CONTROL);
8   </script>
9 </body>
10 </html>
```

注 : 使用 `<hdjs/>` 标签会自动调用执行 `<jsconst/>` 标签

### 38-9-4 hdjs 标签

hdjs 标签用于调用后盾 HDJS 前端库 , 关于 HDJS 前端库的使用 , 请参考 hdjs 手册。

示例

```
<hdjs/>
```

注 : 调用 hdjs 标签时会自动调用 slide,jsconst,cal,jquery,hdjs( 前端库 )

### 38-9-5 validate 自动验证标签

validate 标签用于前端表单的自动验证操作

注 : 使用 `<hdjs/>` 标签会自动调用执行 `<validate/>` 标签

### 38-9-6 bootstrap 标签

bootstrap 标签用于调用 bootstrap 前端库

## 示例

```
<bootstrap/>
```

## 38-9-7 slide 轮换版标签

引用 slide 轮换版 JS 文件，使用方法请参考 hdjs 帮助手册

## 38-9-8 cal 日历标签

cal 标签调用日历插件，具体使用方法请参考 hdjs 帮助手册。

## 38-9-9 foreach 标签

与 PHP 中的 foreach 使用方法一致

### 语法

```
1 <foreach from='变量' key='键名' value='键值'>
2 内容
3 </foreach>
```

#### 示例 1

```
1 <foreach from='$user' key='$key' value='$value'>
2 {$value|strtoupper}
3 </foreach>
```

#### 示例 2 ( 多重嵌套 )

```
1 <foreach from='$user' key='$key' value='$value'>
2 <foreach from='$value' key='$n' value='$m'>
3 {$m}
4 </foreach>
5 </foreach>
```

## 38-9-10 list 标签

### 语法

```
1 <list from='变量' name='值' row='显示行数' empty='为空时显示内容'>
2 内容
3 </list>
```

#### 示例 1

```
$user = array(
```

```
array('name'=>'李四', 'age'=>33 ),  
array('name'=>'赵六','age'=>13)  
);  
1 <list from='$user' name='id' row='2' empty=' 用户数为 0'>  
2 {$id.username}  
3 </list>
```

注：属性 row='2' 表示只显示 2 行，empty 表示当数据为空时的显示结果

## 示例 2

```
1 <list from='$row' name='n' step='2'>  
2 {$n.title}  
3 </list>
```

注：属性 step='2' 表示每次间隔 2 条数据

## 示例 3

```
1 <list from='$row' name='n' start='2'>  
2 {$n.title}  
3 </list>
```

注：属性 start='2' 表示从第 2 条数据开始显示

## 示例 4

```
1 <list from='$row' name='n'>  
2     <if value='$hd.list.n.first'>  
3         {$hd.list.n.index}: 这是第一条数据  
4     <elseif value='$hd.list.n.last'>  
5         {$hd.list.n.index}: 最后一条记录  
6     <else>  
7         {$hd.list.n.index}:{$n.title}  
8     </if>  
9 </list>
```

10 共有 :{\$hd.list.n.total} 条

\$hd.list.n.first 是否为第 1 条记录

\$hd.list.n.last 是否为最后一条记录

\$hd.list.n.total 总记录数

\$hd.list.n.index 当前循环是第几条

其中 n 为 list 的 name 属性值

## 38-9-11 if 标签

if 条件表达式的语句非常简单

### 语法

```
1 <if value=' 条件 '>  
2   内容  
3 </if>
```

### 示例一

```
1 <if value='$webname eq 'houdunwang">  
2   后盾网  
3 </if>
```

### 示例二

```
1 <if value='$webname eq 'houdunwang">  
2   后盾网  
3 <elseif value='$webname eq 'baidu'"/>  
4   百度  
5 <else/>  
6   其他网站  
7 </if>
```

## 38-9-12 标签

switch 标签的作用与 php 中的 switch 相同

### 语法

```
1 <switch value=' 变量或表达式或常量 '>  
2 <case value=' 值 '>  
3   内容  
4 </case>  
5 <default>  
6   默认值  
7 </switch>
```

### 示例一

```
1 <switch value='$webname'>
```

```
2 <case value='houdunwang'> 这是后盾网 </case>
3 <case value ='sina'> 这是新浪 </case>
4 <break/> 其他网站
5 </switch>
```

## 示例二

HDPHP 框架支持在变量中使用函数

```
1 <switch value='{$webname|strtolower}'>
2 <case value='houdunwang'> 这是后盾网 </case>
3 <case value ='sina'> 这是新浪 </case>
4 <break/> 其他网站
5 </switch>
```

## 示例三 ( 使用常量做为值 )

```
1 <switch value='{$hd.const.webname}'>
2 <case value='houdunwang'> 这是后盾网 </case>
3 <case value ='sina'> 这是新浪 </case>
4 <break/> 其他网站
5 </switch>
```

## 示例四 ( 使用超全局数据如 \$\_GET,\$\_POST)

```
1 <switch value='{$hd.get.webname}'>
2 <case value='houdunwang'> 这是后盾网 </case>
3 <case value ='sina'> 这是新浪 </case>
4 <break/> 其他网站
5 </switch>
```

## 38-9-13 empty 标签

HDPHP 框架支持使用 empty 标签，与 PHP 中的 empty 标签意义一样

注：大部分情况下使用变量调节器 default 更合适，default 变量调节器使用请参考手册相关章节

## 示例

```
1 <empty value='{$config.dbname}'>
2 不存在值时显示的内容
3 <noempty/>
```

4 存在值时显示的内容

---

5 </empty>

---

## 38-9-14 php 标签

可以直接在模版中输入 PHP 代码

### 示例

1 <?php

---

2 echo '后盾网';

---

3 ?>

---

## 38-9-15 define

在视图层定义 PHP 常量

### 语法

1 <define name='常量名' value='值' />

---

### 示例

1 <define name='webname' value='后盾网' />

---

## 38-9-16 js 标签

生成 js 文件调用代码

### 语法

1 <js file='文件路径' />

---

### 示例一

1 \$js = array('path'=>'public/js');

---

2 <js file='\$js.path/c.js' />

---

3 生成的 js 标签为 : <script type='text/javascript' src='public/js/c.js'></script>

---

## 38-9-17 css 标签

生成 css 文件调用代码

### 语法

1 <css file='文件路径' />

---

### 示例一

1 \$user = array('path'=>'public/js');

---

2 <css file='\$user.path/css.css' />

---

3 生成的 html 代码为 : <link type='text/css' rel='stylesheet' href='public/js/css.css'/>

---

## 38-9-18 load 标签 ( 别名 include)

加载文件内容标签 , 比如说你可以通过这个标签 , 加载头部或底部 html 文件

### 语法

1 <load file=' 文件路径 '/>

---

### 示例一

1 \$user = array('path'=>'public/js');

---

2 <load file='\_\_PUBLIC\_\_/\_head.html'/>

---

3 上面的标签会加载模板目录下 public 目录下的 head.html 文件

---

# 39. 自定义模版标签

后盾 HDPHP 框架提供了方便快速的标签定义，可以根据网站需要自定义任意数量的标签，大大减少控制器层代码量，从而实现快速的网站开发。

后盾 HDPHP 框架标签定义基于面向对象思想开发，设置自定义标签简单、快速，下面我们来学习掌握 HDPHP 框架自定义标签的使用方法。

注：可以使用 tag() 函数方便在控制器与模板中调用标签，请查看 tag 函数使用帮助

## 39\_1 创建自定义标签的类文件

- a. 应用目录或应用组目录 Tag 目录下创建标签类文件
- b. 标签类文件名必须以 Tag.class.php 结尾
- c. 标签类必须以 Tag 结尾，如 class HtmlTag{...}

## 39\_2 指定标签类文件

因为在 libs 目录中的所有文件系统会自动加载，创建标签类文件后还需要指定哪个文件是标签类文件，这样做的目的就是从效率上来考虑，所以我们只要掌握配置项的指定即可，可以通过下面几种方式来指定哪个文件是标签类文件。

- a. 修改配置文件中的 'TPL\_TAGS' 值
- b. 通过 C() 方法指定 'TPL\_TAGS' 值

以上两种方式设置第二种方式设置的优先级更高

## 39\_3 配置文件中的设置

```
'TPL_TAGS' => array('HtmlTag') // 当前应用 Tag 目录下 HtmlTag 类
```

```
'TPL_TAGS' => array('Admin.Tag.HtmlTag') // 应用 Admin 目录 Tag 目录下的  
HtmlTag 类
```

## 39\_4 创建自定义标签内容

### 39-4-1 块标签定义

在 libs 目录中创建标签类文件 tag.class.php 文件，类文件内容如下

```
1 <?php  
2 class HtmlTag {
```

```
3  public $tag = array(  
4      'guestbook' => array('block' => 1, 'level' => 4), // 设置 block 标签为块标签，支持四层嵌  
套  
5  );  
6  function _guestbook($attr, $content, $parse) {  
7      $row = $attr['row'];  
8      $str = '';  
9      $str.= 'php $db = M('guestbook');';<br/10     $str.= '$row = $db->limit(' . $row . ')->findall();';  
11     $str.= 'foreach($row as $field):?>';  
12     $str.=$content;  
13     $str.= 'php endforeach;?&gt;';<br/14     return $str;  
15 }  
16 }  
17 ?>
```

定义标签 hd，标签名必须以 \_ 开始标签函数接收 3 个参数，第 1 个为标签属性，第 2 个为标签内容，第 3 个为没有解析过的标签体

其实标签内的代码与在控制器方法里面写的代码是一样的，而且必须是字符串，最后 return 返回就可以了，经过定义后的标签就可以在视图模板中使用了，以下是在视图中的使用方法。

```
1  <guestbook row='2'>  
2      标题 : {$field.title} 内容 : {$field.con}  
3  </guestbook>
```

以上是模版中的代码书写，最终生成的结果如下

标题：一个留言 内容：这是留言的内容

## 39-4-2 独立标签定义

独立标签定义文件如下

```
1  <?php  
2  class HtmlTag{  
3      public $tag = array(
```

```
4     'cssfile'=>array('block'=>0),// 指定标签为独立标签
5 );
6 function _cssfile($attr,$content){
7     $file = $attr['file'];
8     $str="";
9     $str.= '<link type='text/css' rel='stylesheet' href="'.$file.'"/>';
10    return $str;
11 }
12 }
13 ?>
```

以上定义的标签为载入外部 CSS 文件，视图中的使用方法如下

```
1 <html>
2   <head>
3     <title></title>
4     <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>
5     <cssfile file='css.css'/>
6   </head>
7   <body>
```

使用标签后生成的 html 如下

```
1 <html>
2   <head>
3     <title></title>
4     <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>
5     <link type='text/css' rel='stylesheet' href='css.css'/>
6   </head>
7   <body>
```

## 39\_5 HD PHP 框架整合 SMARTY 的使用

```
1 <?php
2 class IndexControl extends Control{
3     function index(){
4         if($this->isCache()){// 判断缓存是否有效
5 }
```

```
5      echo 'cache';
6  }else{
7      $this->display('index',600);// 缓存时间 10 分钟
8  }
9  }
10 }
11 ?>
```

- a. `is_cache()` 方法不需要传入任何参数，系统会根据 `$this->display()` 方法的第 2 个参数设置的时间自动进行缓存判断
- b. `$this->display()` 的第 2 个参数表示缓存时间
- c. 其他设置与 HDPHP 模板引擎一样

# 40. Data 数据处理类

Data 数据处理类主要是对数组等数据进行处理，如无限级分类操作、商品规格的迪卡尔乘积运算等，下面我们来学习使用 Data 处理类。

数据表 category 结构如下：

```
Create Table: CREATE TABLE `demo_category` (
    `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
    `pid` int(11) NOT NULL,
    `cname` varchar(30) DEFAULT NULL,
    PRIMARY KEY (`id`)
) ENGINE=MyISAM AUTO_INCREMENT=13 DEFAULT CHARSET=utf8
```

| id | pid | cname |
|----|-----|-------|
| 1  | 0   | 北京    |
| 2  | 1   | 朝阳    |
| 3  | 0   | 河北    |
| 4  | 2   | 小营    |
| 5  | 3   | 廊坊    |
| 6  | 4   | 后盾网   |

## 40\_1 Data::tree() 获得树状数据 (建议使用)

语法

```
static public function tree($data, $title, $fieldPri = 'cid', $fieldPid = 'pid')
```

| 参数         | 说明    |
|------------|-------|
| \$data     | 数组    |
| \$title    | 字段名称  |
| \$fieldPri | 主键 id |
| \$fieldPid | 父 id  |

```
1 $data = $this->order(array('list_order' => 'desc'))->all();
2 $data= Data::tree($data,'cat_name','id');
```

## 处理后结果新增加字段说明

经过 Data::tree() 方法处理后的数据，会新增一些字段，说明如下。

| 参数     | 说明                         |
|--------|----------------------------|
| _level | 当前栏目等级                     |
| _html  | html 字符前缀，就是传递的 html 参数字符串 |
| _first | 是否为同级栏目第一个（一级栏目不含此属性）      |
| _end   | 是否为同级栏目最后一个（一级栏目不含此属性）     |
| _name  | 带符号的栏目名称                   |

在模板输出后的结果（模板代码不演示，可查看 list 模板标签）

| 排序 | ID | 菜单名称  | 状态 | 类型    | 操作  |
|----|----|-------|----|-------|---|
| 0  | 1  | 设置    | 显示 | 权限+菜单 | <a href="#">添加子菜单</a>   <a href="#">修改</a>   <a href="#">删除</a> |
| 0  | 2  | —内容设置 | 显示 | 权限+菜单 | <a href="#">添加子菜单</a>   <a href="#">修改</a>   <a href="#">删除</a> |

**注意：**

初学者建议使用 Data::tree()，Data::tree() 包含了 Data::channelList() 方法的功能，

## 40\_2 Data::channelList 获得目录列表

**语法：**

```
static public function channelList($data, $pid = 0, $html = " ", $fieldPri = 'cid', $fieldPid = 'pid', $level = 1)
```

| 参数       | 说明                      |
|----------|-------------------------|
| data     | 操作的数组                   |
| pid      | 父级栏目的 id 值              |
| html     | 栏目名称前缀，用于在视图中显示层次感的栏目列表 |
| fieldPri | 唯一键名，如果是表则是表的主键         |
| fieldPid | 父 ID 键名                 |
| level    | 等级（不需要传参数，系统运行时使用）      |

## 示例

```
1 $db= M('category');
2 $data = $db->all();
3 p(Data::channelList($data,0,'-'));
```

## 处理后结果新增加字段说明

经过 Data::channelList() 方法处理后的数据，会新增一些字段，说明如下。

| 参数     | 说明                         |
|--------|----------------------------|
| _level | 当前栏目等级                     |
| _html  | html 字符前缀，就是传递的 html 参数字符串 |
| _first | 是否为同级栏目第一个（一级栏目不含此属性）      |
| _end   | 是否为同级栏目最后一个（一级栏目不含此属性）     |

## 40\_3 Data::channelLevel 获得多级目录列表（多维数组）

### 语法：

```
static public function channelLevel($data, $pid = 0, $html = " ", $fieldPri = 'cid', $fieldPid = 'pid', $level = 1)
```

| 参数       | 说明                      |
|----------|-------------------------|
| data     | 操作的数组                   |
| pid      | 父级栏目的 id 值              |
| html     | 栏目名称前缀，用于在视图中显示层次感的栏目列表 |
| fieldPri | 唯一键名，如果是表则是表的主键         |
| fieldPid | 父 ID 键名                 |
| level    | 等级（不需要传参数，系统运行时使用）      |

```
1 $db= M('category');
2 $data = $db->all();
3 p(Data::channelList($data,0,'-'));
```

## 处理后结果新增加字段说明

经过 Data::channelList() 方法处理后的数据，会新增一些字段，说明如下。

| 参数     | 说明                         |
|--------|----------------------------|
| _level | 当前栏目等级                     |
| _html  | html 字符前缀，就是传递的 html 参数字符串 |
| _data  | 子栏目数据                      |

## 40\_4 Data::parentChannel() 获得所有父级栏目

### 语法：

```
static public function parentChannel($data, $sid, $fieldPri = 'cid', $fieldPid = 'pid')
```

| 参数       | 说明              |
|----------|-----------------|
| data     | 操作的数组           |
| sid      | 子栏目             |
| fieldPri | 唯一键名，如果是表则是表的主键 |
| fieldPid | 父 ID 键名         |

```
1 $data = $db->all();
2 Data::parentChannel($data,4);
```

## 40\_5 Data::isChild() 判断是否为子栏目

### 语法

```
static function isChild($data, $sid, $pid, $fieldPri = 'cid', $fieldPid = 'pid')
```

### 参数说明

| 参数   | 说明     |
|------|--------|
| data | 操作的数组  |
| sid  | 子栏目 id |

| 参数       | 说明              |
|----------|-----------------|
| pid      | 父栏目 id          |
| fieldPri | 唯一键名，如果是表则是表的主键 |
| fieldPid | 父 ID 键名         |

### 示例 1

判断栏目 smarty 是否为 php 栏目的子栏目，数据表如下：

| cid | catname | pid |
|-----|---------|-----|
| 1   | php     | 0   |
| 2   | smarty  | 1   |

```

1 $data = M('category')->all();
2 if(Data::isChild(data,2,1)){
3 echo 'smarty 是 php 的子栏目 ';
4 }
```

## 40\_6 Data::descarte() 迪卡尔乘积

迪卡尔乘积静态方法主要方便实现商品规格的表示形式

static function descarte(\$arr, \$tmp = array())

# 41. 目录操作类

后盾 HDPHP 框架提供了完善且简洁的目录管理类 dir，HDPHP 框架的所有功能类，都不需要程序员将类文件引入系统会自动加载。下面来学习目录类的具体实用功能。

## 41\_1 Dir::getExt() 获得文件扩展名

通过 dir::get\_ext() 方法获得文件的扩展名，返回的扩展名统一为小写，get\_ext() 只有一个参数即文件名

```
static function getExt($file)  
{$file          文件名  
1  Dir::getExt('houdunwang.com.php');
```

## 41\_2 Dir::tree() 遍历目录内容

获得目录下的所有文件及目录内容，包含详细信息，支持递归遍历，参数说明如下

```
static function tree($dirName, $exts = '', $son = 0, $list = array())  
{$dirName      遍历的目录名  
$exts        显示文件的扩展名  
$son         是否递归遍历，默认只遍历当前目录  
$list        递归遍历时使用的参数，调用本函数时不用设置
```

## 41\_3 Dir::treeDir() 只显示目录树

获得指定目录下的所有目录详细信息，支持递归遍历，参数说明如下

```
static function treeDir($dirName, $pid = 0, $son=0,$dirs = array())  
{$dirName      遍历的目录名  
$son         是否递归遍历，默认只遍历当前目录  
$pid         父目录 ID  
$dirs        递归遍历时使用的参数，调用本函数时不用设置
```

## 41\_4 Dir::del() 删除目录与文件

PHP 本身的删除函数是不能删除非空目录的，使用起来比较麻烦，通过本函数只需要传

入合法的目录名系统会自动递归删除目录，语法如下

static function del(\$file)

\$dirName 要删除的目录名

1 Dir::del('hdphp');// 删除 hdphp 目录

2 dir::del('houdunwang.php');// 删除 houdunwang.php 文件

## 41\_5 create() 创建目录

本函数为创建目录使用支持递归创建，语法如下

static function create(\$dirName, \$auth = 0777)

\$dirName 目录路径

\$auth 目录权限

1 dir::create('houdunwang/bbs'); 则创建 houdunwang 目录再创建 bbs 目录

## 41\_6 copy() 复制目录内容

将目录内容包括子目录内容完整的复制到指定的目录中，语法如下

static function copy(\$olddir, \$newdir)

\$olddir 原目录

\$newdir 目录目录

## 41\_7 Dir::safeFile() 创建目录安全文件

很多时候由于服务器软件设置不当，造成目录列表被显示。大部分情况下我们是不希望出现这样的情况的，对我们的文件安全造成很大影响。

Dir::safeFile() 就是为了解决这个安全隐患而出现的，他可以在目录创建安全文件 index.html 从而让我们的目录文件得以安全的保护，如果传递第 2 个参数为 TRUE 则在子目录下也创建安全文件。

static function safeFile(\$dirName, \$recursive=false)

\$dirName 目录名

\$recursive 是否在子目录也递归创建安全文件，默认不创建

# 42. string 字符串处理类

在开发中经常对字符串进行大量处理，虽然 PHP 本身提供了很多方便的字符串处理函数，但有些特殊处理功能并没有提供，后盾网 HDPHP 框架提供了方便的字符串功能扩展如编码转换、中文转拼音、全角字符转半角字符等，下来来学习和使用 HDPHP 框架提供的强大字符串处理功能。

## 42\_1 String::toSemiangle 全角转半角

通过本静态方法可以方便的将全角字符转为半角字符，使用方法如下：

```
1 class indexControl extends Control{
2     function index(){
3         echo string::to_semiangle('【】');
4     }
5 }
```

以上操作会将全角字符 【】 转为半角字符 []

## 42\_2 string::pinyin 中文转拼音

通过本静态方法可以将中文汉字转为拼音，可以在如生成静态目录时方便获得目录名称，使用方法如下：

```
1 class indexControl extends Control{
2     function index(){
3         echo string::pinyin('后盾网');
4     }
5 }
```

以上操作会将中文字字符串 '后盾网' 转为 houdunwang 输出

注：如果转换的字符集与配置文件不同可以通过传递第 2 个参数指定字符集，如下

```
1 class indexControl extends Control{
2     function index(){
3         echo string::to_semiangle('后盾网论坛','gb2312');
4     }
5 }
```

以上操作会将 gb2312 编码的中文字字符串 '后盾网论坛' 转为 houdunwangluntan

## 42\_3 string::splitWord() 中文分词

本静态方法用于将字符串进行分词操作，结果为数组键名为中文词，键值为词数量 具体使用示例如下：

```
1 class indexControl extends Control{  
2     function index(){  
3         echo String::splitWord(' 后盾框架免费开源 ');  
4     }  
5 }
```

6 拆分后的内容为 array(' 后盾 '=>1,' 框架 '=>1,' 免费 '=>1,' 开源 '=>1); 注：如果传字符集与配置文件中的 charset 则需要传入第 2 个参数（输入字符集），如果输出字符与配置文件不同则需要传入第 3 个参数（输出字符集）如下：string::split\_word(' 后盾框架可以用于任意用途 , 完全免费 ','utf8','gbk');

## 42\_4 String::removePunctuation() 去除标点符号

本静态方法用去除字符串中的所有半角或全角标点符号，示例如下：

```
1 class indexControl extends Control{  
2     function index(){  
3         echo String::removePunctuation(' 北京后盾网 , 免费开源框架 hd。 ');  
4     }  
5 }
```

以上执行后将会把字符串 ' 北京后盾网 , 免费开源框架 hd。 ' 中的 ' , 。 ' 去除掉

# 43. 缓存控制

为了在大并发时提供更快的响应速度，HDPHP 框架提供了缓存处理机制，操作简单、高效，涵盖 file 缓存、memcache 缓存可以根据需要指派不同的缓存处理机制，只需要在配置文件中设置即可。

## 43\_1 影响缓存的配置项

对缓存影响的配置项说明

| 说明                  |  |
|---------------------|--|
| CACHE_TYPE          | 缓存类型，可选择类型有：file[ 文件缓存 ] memcache[memcache 内存缓存 ]                              |
| CACHE_TIME          | 全局默认缓存时间 如果缓存时没有指定时间将以此为准，单位秒 ,0 为不缓存 -1 为永久缓存                                 |
| CACHE_ZIP           | 缓存数据是否压缩 true 压缩 false 不压缩   |
| CACHE_SELECT_TIME   | SQL SELECT 查询缓存时间 推荐使用模板缓存 0 为关闭 -1 为永久缓存<br>SELECT 中的字段按需取不要取无用字段字段按需取不要取无用字段 |
| CACHE_SELECT_LENGTH | SELECT 结果超过这个值不进行缓存  |
| CACHE_TPL_TIME      | 模板缓存时间 0 为不缓存 -1 为永久缓存   |
| CACHE_MEMCACHE      | 当缓存驱动为 Memcache 时设置 Memcache 主机  |
| CACHE_REDIS         | 当缓存驱动为 Redis 时设置 Redis 主机  |
| CACHE_SAVE          | 记录缓存命中率  |

## 43\_2 Memcache 缓存设置

如果启用 Memcache 缓存控制，需要以下几项：

1. 设置配置项 CACHE\_TYPE 为 memcache
2. 设置配置项 CACHE\_MEMCACHE，各参数说明如下表

| 参数       | 默认值       | 说明                                   |
|----------|-----------|--------------------------------------|
| host     | 127.0.0.1 | 主机                                   |
| port     | 11211     | 端口                                   |
| timeout  | 1         | 连接超时时间 ( 单位为秒 , 不要设置过长 )             |
| weight   | 1         | 权重 ( 设置多个 memcache 服务器时有效 , 可以不用设置 ) |
| pconnect | 1         | 是否持久连接 ( 可以不用设置 )                    |

## 43\_3 Redis 缓存设置

如果启用 Redis 缓存控制 , 需要以下几项 :

1. 设置配置项 CACHE\_TYPE 为 redis
2. 设置配置项 CACHE\_REDIS , 各参数说明如下表

| 参数       | 默认值       | 说明                       |
|----------|-----------|--------------------------|
| host     | 127.0.0.1 | 主机                       |
| port     | 6379      | 端口                       |
| password |           | 主机密码 没有密码时留空             |
| timeout  | 1         | 连接超时时间 ( 单位为秒 , 不要设置过长 ) |
| db       | 1         | 使用的数据库                   |
| pconnect | 0         | 是否为长链接                   |

## 43\_4 CACHE 缓存类

语法 : cache(\$options)

不同的缓存方式所需要的参数是不同的 , 下面是对不同缓存方式所能接收的 \$options 参数进行说明

[所有缓存方式公用配置项](#)

| 配置项    | 说明    |
|--------|-------|
| driver | 缓存的类型 |

| 配置项    | 说明                                  |
|--------|-------------------------------------|
| expire | 默认的缓存时间                             |
| prefix | 缓存前缀，区分不同控制器或方法中的同名 KEY，或全局缓存区分不同应用 |
| length | 队列长度，队列即允许缓存的数量                     |
| zip    | 是否启动缓存数据压缩存储                        |

### 不同缓存类型的配置项说明

| 缓存类型          | 配置项    | 说明   |
|---------------|--------|--|
| file 缓存机制     | driver | file 缓存时必须设置为 file   |
|               | dir    | 缓存存放目录   |
| memcache 缓存机制 | driver | memcache 缓存时必须设置为 memcache   |
|               | host   | array(// 多个服务器传入设置二维数组<br>'host'=>'127.0.0.1',//127.0.0.1 主机<br>'port'=>11211,//11211 端口<br>'timeout'=>1,// 连接超时时间<br>'weight'=>1,// 权重 (多台服务器时有效设置)<br>'pconnect'=>1,// 是否持久连接 (可以不用设置)<br>)    |
| Redis 缓存机制    | driver | redis 缓存时必须设置为 redis   |
|               | host   | array(// 多个服务器传入设置二维数组<br>'host'=>'127.0.0.1',// 主机地址<br>'port'=>6379,// 端口<br>'password'=>',// 主机密码 没有密码时留空<br>'timeout'=>1,// 连接超时时间 单位为秒<br>'db'=>1,// 使用的数据库<br>'pconnect'=>0,// 是否为长链接<br>) |

### 方法说明

| 方法名称                | 参数说明   |
|---------------------|--------|
| set(\$name,\$value) | 设置缓存数据 |
| get(\$name)         | 获得缓存数据 |
| del(\$name)         | 删除缓存   |

| 方法名称                         | 参数说明                          |
|------------------------------|-------------------------------|
| delAll(\$time=null)          | 删除所有缓存，如果设置 \$time 则删除超过此值的缓存 |
| options(\$name,\$value=null) | 设置与获得缓存配置                     |

## 43\_5 设置缓存

程序员根据网站要求指定合适的缓存时间，达到最佳缓存效果，操作缓存需要实例化缓存对象。

注：以下代码中的 Cache::init() 方法不传递参数时使用配置文件设置

### 示例 1

```

1 <?php
2 class IndexControl extends Control {
3     function index() {
4         $cache = cache::init();
5         $data = array("// 要缓存的数据
6             'webname' => '后盾网',
7             'time' => time()
8         );
9         $cache->set('houdunwang',$data,800);
10    }
11 }
12 ?>

```

如果没有指定缓存时间，将采用配置文件中 CACHE\_TIME 选项的值

### 示例 2

```

1 $cache = cache::init(array('dir'=>'cache','zip'=>true,'prefix'=>'hd_'));
2 $cache->houdunwangxj='向军';// 缓存数据 KEY 为 houdunwangxj
3 echo $cache->houdunwangxj;// 得到 KEY 为 houdunwangxj 的缓存内容

```

将缓存数据放入根目录下的 cache 目录，以 hd\_ 为缓存文件前缀，启用压缩数据

## 43\_6 删除缓存

### 示例 1

```
1 $cache = cache::init();
2 $cache->houdunwangxj = '向军';
3 $cache->houdunwangxj=null;// 删除缓存数据
```

直接将缓存 KEY 赋值为 NULL 即删除缓存数据

### 示例 2

```
1 $cache->del('houdunwangxj');
```

使用以 \$cache->del() 方法删除

## 43\_7 删除缓存

### 示例 1 删除所缓存

```
1 $cache = cache::init();
2 $cache->houdunwangxj = '向军';
3 $cache->houdunwang='后盾网';
4 $cache->delAll();// 删除所有缓存
```

### 示例 2 删除过期缓存

```
1 $cache->delAll(3600);
```

删除 1 小时前的缓存文件

## 43\_8 获得与设置配置

### 示例 1 获得缓存配置

```
1 $cache = cache::init();
2 echo $cache->options('expire');
```

### 示例 2 设置缓存参数

```
1 $cache->options('expire',600);
2 echo $cache->options('expire');
```

## 43\_9 缓存队列

为了避免缓存过多，可以通过 delAll() 删除过期缓存，也可以使用 HDPHP 框架中的缓

存队列特性，即指定缓存的数量，以后进先出的原则删除最旧的缓存。

注：Memcache 等缓存控制本身具有 LRU 机制，所以缓存队列目前适用于 File 缓存

```
1 $cache = cache::init(array('length'=>2,'zip'=>false));  
2 $cache->set('name','后盾网');  
3 $cache->set('web','www.hodunwang.com');  
4 $cache->set('bbs','bbs.hodunwang.com');
```

设置了 length 即队列长度为 2 所以只能缓存 2 条记录，队列采用先进先出的原则，所以 name 将被删除

## 43\_10 以 Memcache 操作缓存

无论是使用 Memcache 或者 Redis 操作缓存，可以通过修改配置文件的方式，或者调用 cache::init() 方法时传递参数即可，示例如下：

```
1 $cache = cache::init(  
2     'driver'=>'memcache',  
3     'host'=>array('host'=>'127.0.0.1','port'=>11211))  
4 );  
5 $cache->set('webname','后盾网');  
6 echo $cache->get('webname');
```

## 43\_11 以 Redis 操作缓存

无论是使用 Memcache 或者 Redis 操作缓存，可以通过修改配置文件的方式，或者调用 cache::init() 方法时传递参数即可，示例如下：

```
1 $cache = cache::init(  
2     'driver'=>'redis',  
3     'host'=>array('host'=>'127.0.0.1','port'=>6379))  
4 );  
5 $cache->set('webname','后盾网');  
6 echo $cache->get('webname');  
7
```

## 43\_12 S() 缓存函数

上面学习了通过 cache 类操作缓存，其实 S 方法也是通过 Cache 类操作缓存，只是提供了一种快捷的函数调用方式。

如果使用 memcache 或 redis 进行缓存操作，请修改配置文件中的相应配置项

```
S($name, $value = false, $expire = null, $options = array())
```

| 参数        | 说明   |
|-----------|--|
| \$name    | 缓存名称   |
| \$value   | 缓存内容   |
| \$expire  | 缓存时间   |
| \$options | 缓存对象参数如 array('dir'=>'cache','driver'=>'memcache' , 'host'=>array('host'=>'127.0.0.1','port'=>11211,'timeout'=>1)) |

### 缓存数据

```
1 S('houdunwang','后盾网');  
2 echo S('houdunwang');
```

### 删除缓存

```
1 S('houdunwang',null)
```

### 设置缓存驱动

```
1 S('houdunwang','bbs.houdunwang.com',1440,array('dir'=>'cache','driver'=>'file','zip'=>true));
```

将数据缓存到 cache 目录，以 file 文件形式缓存，启用数据压缩

## 43\_13 F() 快速文件缓存

通过 F() 方法以文件形式快速对数据进行缓存

```
function F($name, $value = false, $path = CACHE_PATH)
```

| 参数      | 说明     |
|---------|--------|
| \$name  | 缓存数据名称 |
| \$value | 缓存数据内容 |

| 参数     | 说明     |
|--------|--------|
| \$path | 数据存放目录 |

## 缓存数据

```
1 F('houdunwang',array('name'=>'后盾网','url'=>'houdunwang.com'));
```

## 获得数据

```
1 echo F('houdunwang');
```

## 删除缓存

```
1 F('houdunwang',NULL)
```

# 44. 多语言支持

后盾 HDPHP 框架提供便捷的多语言支持，程序开发人员只需要配置好语言包即可，下面来掌握具体的使用方法。

1. 首先创建语言包即数组文件，放置在应用组中的 Common/language 目录或应用的 Extend/language 目录中都可，其他如果两处都有语言包文件则应用中的语言包文件优先级更高
2. 使用语言包可以通过修改配置项 LANGUAGE 或在控制器中通过 C('LANGUAGE','zh') 来进行设置

## 语言包 zh.php 文件

```
1 <?php  
2 if(!defined('HDPHP_PATH'))exit;  
3 return array(  
4     'title'=>'后盾语言测试'  
5 );  
6 ?>
```

## 控制器内容

```
1 <?php  
2 class indexControl extends Control{  
3     function index(){  
4         C('language','zh');// 也可以修改配置文件中 language 值  
5     }  
6 }  
7 ?>
```

在视图中可以通过 {\$hd.language.hd} 读取数据即可

# 45. 生成 HTML 静态文件

生成静态 html 好处很明显，第一点是非常有利于 SEO 的优化，搜索引擎可以对 html 文件进行更好的收录。第二点是显著减轻网站的负载，WEB 服务器在处理 html 文件时只是简单的读取操作，不会经由 PHP 模块进行处理，也不会有数据库服务器的操作，所以速度要较 PHP 文件快得多。

HTML 有很多特点，但是我们也不能盲目使用 HTML，而是应该在更新频率较低的页面采用 HTML 处理方式，比如说文章系统、博客系统，社区门户等方面，而在更新频率较高的应用如论坛、微博、SNS 等方面建议采用 HDPHP 框架的 PATHINFO 及结合 HDPHP 框架缓存机制，解决 SEO 及高负载问题，所以请根据情况适时的选择合适的处理机制。

后盾 HDPHP 框架提供高效简单的批量文件处理功能，我们来通过示例来解释 HDPHP 框架生成 HTML 文件的使用方法。

## 45\_1 生成 HTML 文件

### 生成一个静态文件

```
Html::make($control,$method,$data)
```

| 参数        | 说明   |
|-----------|--|
| \$control | 生成静态文件调用的控制器   |
| \$method  | 生成静态调用的方法  |
| \$data    | 生成静态文件的参数<br>array('aid'=>1,'_html'=>'2013/2013/09/1.html'); |

### 批量生成静态文件

```
Html::makeAll($content,$method,$data);
```

| 参数        | 说明           |
|-----------|--------------|
| \$control | 生成静态文件调用的控制器 |
| \$method  | 生成静态调用的方法    |

| 参数     | 说明  |
|--------|---|
| \$data | 生成静态文件的参数<br>array(<br>array('aid'=>1,'_html'=>'2013/2013/09/1.html'),<br>array('aid'=>2,'_html'=>'2013/2013/09/2.html')<br>) |

## 测试表

```

1 CREATE TABLE IF NOT EXISTS `hd_news` (
2   `id` int(11) NOT NULL AUTO_INCREMENT,
3   `title` char(30) DEFAULT NULL,
4   `con` text,
5   PRIMARY KEY (`id`)
6 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=65522 ;
7 INSERT INTO `hd_news` (`id`, `title`, `con`) VALUES
8 (1, '后盾网 PHP 教程', '登录后盾网论坛 bbs.houdunwang.com 下载'),
9 (65521, '后盾 HDPHP 框架', '可以免费用于任何法律允许的网站');

```

## PHP 代码

```

1 <?php
2 class indexControl extends Control {
3 // 显示内容方法
4 public function arc() {
5   $id = $_GET['id'];// 显示文章的 ID
6   $db = M('news');// 创建模型对象
7   $row = $db->find($id);// 查找单条记录
8   $this->assign('row', $row);// 向视图层分配变量
9   $this->display('arc');// 必须设置模板名称
10 }
11 // 生成静态
12 public function html() {
13   $db = m('news');// 创建新闻模型对象
14   $row = $db->findall();// 查找所有记录
15   foreach ($row as $k => $data) {// 遍历记录
16     $row[$k]['id']=$data['id'];// 文章 id
}

```

```
17 $row[$k]['_html'] = 'h/b/{$data['id']}.html';// 添加生成的 html 文件路径
18 }
19 html::makeAll('Content', 'Article', $row);// 创建 html 文件
20 }
21 }
```

参数 \_html 指生成文件的 html 路径地址

## 45\_2 删除 HTML 文件

### 45-2-1 删除目录下的所有 html 文件

```
1 class indexControl extends Control {
2     function delHtml(){
3         Dir::del('h');
4     }
5 }
```

### 45-2-2 删除指定的 HTML 文件

```
1 class indexControl extends Control {
2     function delHtml(){
3         Dir::del('/h/c/1.html');
4     }
5 }
```

# 46. cart 购物车类

HDPHP 框架为了便于商城系统的开发提供了完善的购物车处理类，使商城购物车处理更加方便快捷，程序员只需要专注业务流程而不用关注实现步骤，大大增加了开发效率

## 影响购物车类的配置项

| 配置项       | 说明                 |
|-----------|--------------------|
| CART_NAME | \$_SESSION 中购物车的名称 |

## 静态类 Cart 属性说明

| 属性               | 说明                     |
|------------------|------------------------|
| Cart::\$cartName | 储存在 \$_SESSION 中的购物车名称 |

## 46\_1 添加购物车 cart::add()

添加购物车使用 cart::add() 方法实现，购物车中的数据会写入到 \$\_SESSION 超全局数组中，具体使用方法如下：

```
1 <?php
2 class indexControl {
3     function index() {
4         $data = array(
5             'id' => 1, // 商品 ID
6             'name'=>'后盾网 PHP 视频教程光盘',//商品名称
7             'num' => 2, // 商品数量
8             'price' => 988, // 商品价格
9             'options' => array{// 其他参数如价格、颜色、可以为数组或字符串
10                'color' => 'red',
11                'size' => 'L'
12            }
13        );
14        cart::add($data); // 添加到购物车
}
```

```
15     p($_SESSION);
16 }
17 }
18 ?>
```

## 46\_2 更新购物车 Cart::update()

更新购物车需要传入商品的唯一 SID , 切记不是商品的 ID 值 , 同时传入更新的商品数量 , 示例如下

```
1 $data=array(
2   'sid'=>'4d854bc6',// 唯一 sid , 添加购物车时自动生成
3   'num'=>88
4 );
5 Cart::update($data);
```

## 46\_3 获得购物车商品数据 Cart::getGoods()

通过 cart::getGoods() 可以获得购物车中的所有商品数据 , 使用方法如下

```
1 cart::getGoods();
```

## 46\_4 获得购物车所有数据包 Cart::getAllData()

cart::getAllData() 与 cart::getGoods() 区别在于不仅获得所有商品数量 , 同时还会获得购物车中的所有商品数量及总价格 , 使用方法如下 :

```
1 cart::getAllData();
```

## 46\_5 清空购物车中的所有商品 Cart::delAll()

cart::delAll(); 方法用于清空购物车中的所有商品 , 使用方法如下

```
1 cart::delAll();
```

## 46\_6 获得商品总价格 Cart::getTotalPrice()

通过 cart::getTotalPrice() 方法可以得到购物车中的商品合计总价 , 使用方法如下 :

```
1 cart::getTotalPrice();
```

## 46\_7 统计购物车中的商品数量 Cart::getTotalNums()

通过 cart::getTotalNums() 方法得到购物车中的所有商品数量，使用方法如下：

```
cart::getTotalNums();
```

## 46\_8 获得定单号

通过 Cart::getOrderId() 方法用于获得商品唯一定单号

```
Cart::getOrderId();
```

# 47. RBAC 基于角色的权限控制

后盾 HDPHP 框架支持完善的，强大的基于角色的权限控制，所有操作都通过核心类 Rbac.class.php 完成

## 47\_1 影响 RBAC 的配置项

| 配置项                  | 说明   |
|----------------------|--|
| RBAC_TYPE            | 1 时时认证 2 登录认证  |
| RBAC_SUPER_ADMIN     | 超级管理员 认证 SESSION 键名  |
| RBAC_USERNAME_FIELD  | 用户表中的用户名字段名称   |
| RBAC_PASSWORD_FIELD  | 用户表中的密码字段名称  |
| RBAC_AUTH_KEY        | 用户登录后在 SESSION 中储存的用户 ID                                     |
| RBAC_NO_AUTH         | 不需要验证的控制器或方法如：array(index/index) 表示 index 控制器的 index 方法不需要验证 |
| RBAC_USER_TABLE      | 用户信息表  |
| RBAC_ROLE_TABLE      | 角色信息表  |
| RBAC_NODE_TABLE      | 节点信息表  |
| RBAC_ROLE_USER_TABLE | 角色与用户中间关联表   |
| RBAC_ACCESS_TABLE    | 权限分配表  |

## 47\_2 Rbac::IsLogin()

这个方法是验证用户是否登录，主要是检查 \$\_SESSION 值，因为用户登录后会在

SESSION 保存配置项中的 RBAC\_AUTH\_KEY 值

如果用户成功登录这个值是用户的表中的 ID 号

```
1 function index(){
2     if(Rbac::isLogin()){// 验证用户是否登录
3         echo '已经登录';
4     }else{
5         echo '没有登录';
6     }
7 }
```

如果用户没有登录则跳转到登录界面

## 47\_3 Rbac::login() 用户登录

登录函数，登录成功将合法用户权限信息写入 \$\_SESSION，如果用户表中存在与参数 \$superadmin 相同的用户则此用户为超级管理员 在 \$\_SESSION 中记录，超级管理员不受任何访问限制，语法如下

```
function Rbac::login($username, $password, $superadmin = null,
$fieldUserName = null, $fieldPassword = null)
```

参数说明

- a. \$username 参数为用户名，用户错误可以通过 Rbac::\$error 得到错误内容
- b. \$password 参数为密码，密码错误也可以通过 Rbac::\$error 得到错误内容
- c. \$superadmin 参数为超级管理员的帐号名，也就是说如果这个用户的帐号与这个参数相同，并且登录成功，那么这个用户为超级管理员，超级管理员用户在网站中可以随意畅行，没有限制。
- d. \$fieldUserName 用户表中的用户名字段名称
- e. \$fieldPassword 用户表中的密码字段名称

如果发生错误，均可以通过 \$this->error 得到错误信息内容

```
1 if(!Rbac::login($_POST['username'],$_POST['password'],'admin')){
2     $this->error(Rbac::$error,'index/login');
3 }
```

以上代码是用户登录验证，如果用户名正确且密码正确，用户成功登录，同时将用户权限信息写入 SESSION，如果登录失败则显示失败原因，同时跳转到登录窗口界面。

## 47\_4 Rbac::checkAccess()

这个方法主要是验证用户是否有访问应用、控制器模块、方法的权限，如果没有将返回 FALSE 值，程序员可以根据返回值来决定操作方向。

### 示例

```
1 if(!Rbac::checkAccess()){
2     $this->error('对不起你没有操作权限 ');
3 }
```

如果一个控制器中的所有方法都要验证，可以定义控制器中的 \_\_auto 方法，这个方法会自动执行，也就是说执行一个控制器方法会首先运行 \_\_auto 方法

也可以定义一个公共的 RbacControl 类来完成验证操作，将所有有关验证的代码全部放在 rbacControl 里的 \_\_auto 方法中，控制器需要验证时只需要继承 RbacControl 控制器即可

## 47\_5 Rbac::getNodeList() 获得所有节点列表

通过本函数可以获得数据表中的所有节点列表信息，用于在后台构造权限控制视图时使用，就像 HD PHP 框架中的 SETUP 应用中的 RBAC 设置

```
function getNodeList($roleId = '')
$roleId          角色 id 如果传值将获得角色的所有权限信息
```

### 示例：

```
1 class indexControl extends Control{
2     function index(){
3         $list = Rbac::getNodeList();
4         p($list);
5     }
6 }
7
```

# 48. session 操作

HDPHP 框架提供内置的 SESSION 处理引擎如 Mysql 引擎、memcached 引擎、redis 引擎等，同时提供了便捷的 session 操作方式，使操作 session 更加安全高效。

默认 session 为自动启动，如果不需要自动开启 SESSION 设置配置项 'SESSION\_AUTO\_START' => false

## 48\_1 SESSION 配置项

| 配置项                | 说明           |
|--------------------|--------------|
| SESSION_AUTO_START | 自动开启 SESSION |
| SESSION_OPTIONS    | SESSION 处理参数 |

## 48\_2 自定义 SESSION 参数

| 配置项           | 说明  |
|---------------|---|
| name          | session_name 值                            |
| path          | session_save_path 值                       |
| expire        | session.gc_maxlifetime 设置值                |
| domain        | session.cookie_domain 设置值                 |
| use_cookies   | session.use_cookies 设置值                   |
| use_trans_sid | session.use_trans_sid 设置值                 |
| cache_limiter | session_cache_limiter 设置值                 |
| cache_expire  | session_cache_expire 设置值                  |
| type          | session hander 类型。支持 mysql,memcache,redis |

## 49. session() 函数

语法 : session(\$name,\$value='', \$options)

| 功能         | 代码                        |
|------------|---------------------------|
| 设置 session | session('webname','后盾网'); |
| 获得 session | session('webname');       |
| 删除 session | session('webname',null);  |
| 检测 session | session('?webname');      |
| 注销 session | session('[destroy]');     |
| 停止 session | session('[pause]');       |
| 开启 session | session('[start]);        |

# 50. SESSION 引擎

## 50\_1 session 的 mysql 处理机制

系统提供了基于数据库的 session 处理机制，使 session 处理更加高效，同时也便于在线人数统计等操作的实现，修改配置项 SESSION\_OPTIONS 即可。

```
1 'SESSION_TYPE'=>'mysql',// 必须留空
2 'SESSION_OPTIONS'=>array(
3     'host'      =>'127.0.0.1',//Mysql 主机
4     'port'      =>3306,// 端口
5     'user'      =>'root',// 用户名
6     'password'  =>"',// 密码
7     'database'  =>'test',// 数据库
8     'table'     =>'hd_session',// 完整表名
9     'expire'    =>1440,//session 过期时间
10 )
```

注：除了表名外，其余配置项可以不设置

### 创建数据表

SESSION 表结构如下

```
1 create table hd_session(
2     sessid char(32) primary key not null,#session_id
3     card char(32) not null,#SESSION 令牌
4     data text not null,
5     atime int(10) not null,
6     ip char(15) not null,
7     )charset utf8 engine=myisam;
```

## 50\_2 session 的 Memcache 处理机制

HDPHP 框架支持 Memcache 的 session 操作机制，速度更快，在大并发，大请求发生时效果尤为明显，使用 HDPHP 框架的 Memcache 操作 session 非常简单，需要修改配置项如下：

```
1 'SESSION_TYPE'=>'memcache',// 必须留空
```

```
2 'SESSION_OPTIONS'=>array(  
3   'host'=>'127.0.0.1',// 主机  
4   'port'=>11211// 端口  
5 )
```

在 memcache 服务正常开启，配置项设置正确，那么现在的 session 机制就是在 Memcache 下运行的。

## 50\_3 session 的 Redis 处理机制

HDPHP 框架除了可以使用 file 与 Memcache 操作 session 同时也支持 Redis 处理，因为 Redis 数据操作都是在内存中所以在大并发下也会带来惊人的速度提升，需要修改配置项如下：

```
1 'SESSION_TYPE'=>'redis',// 必须留空  
2 'SESSION_OPTIONS'=>array(  
3   'host'=>'127.0.0.1',// 主机  
4   'port'=>6379,// 端口  
5   'password'=>"",// 密码  
6   'db'=>0,// 数据库  
7 )
```

在 Redis 服务正常开启的情况下，那么现在的 session 机制就是在 Redis 下运行的。

# 51. cookie 操作

通过 cookie() 函数可以任意且方便对 cookie 操作，具体使用方法如下：

cookie 的使用可以方便的通过 cookie() 函数完成

语法 : cookie(\$name,\$value='', \$options=null)

| 功能              | 代码  |
|-----------------|---|
| 设置 cookie       | cookie('webname','houdunwang',100); //100 为过期时间，为 0 表示会话 cookie |
| 获得 cookie       | cookie('webname');  |
| 删除 cookie       | cookie('webname',null);   |
| 删除指定前缀所有 cookie | cookie(NULL,'hd_');   |

参数 options 说明：

| 名称     | 说明                           |
|--------|------------------------------|
| prefix | cookie 前缀 ( 默认为配置文件设置 )      |
| expire | 默认过期时间，0 为会话时长 ( 默认为配置文件设置 ) |
| path   | 保存路径 ( 默认为配置文件设置 )           |
| domain | 有效域名 ( 默认为配置文件设置 )           |

# 52. Xml 操作类

在 web 开发中经常大量用到 XML 这种快速简便的数据传输格式 , HD PHP 框架为程序员提供了简单快速的 xml 处理机制

## 52\_1 Xml::create() 创建 xml 文件

语法 : `xmlCreate($data, $root = null, $encoding = 'UTF-8')`

\$data 数据

\$root 根节点

encoding 编码方式默认为 utf-8

示例 :

```
1 <?php
2 class indexControl {
3     function index() {
4         $data = array(
5             'news' => array(
6                 'title' => '后盾网 HD PHP 框架',
7                 'con' => 'HD PHP 框架为 WEB 开发助力'
8             )
9         );
10    header('content-type:text/xml');
11    echo xml::Create($data, 'root', 'utf-8');
12    exit;
13 }
14 ?
15 ?>
```

## 52\_2 Xml::toArray()

将 XML 文件转为标准的数组结构 , 本函数需要参数为 xml 内容的字符串数据

示例 :

```
1 <?php
2 class indexControl {
```

```
3     function index() {  
4         p(xml::toArray(file_get_contents('xml.xml')));  
5     }  
6 }  
7 ?>
```

## 53. IP 处理类

本类为静态类功能为通过 IP 获得地理位置，地址位置会根据配置文件中指定的字符集自动转换避免出现乱码

### 53\_1 Ip::area() 获得 IP 来源地理位置

```
1 echo ip::area('58.23.236.236');
```

---

注：如果 ip::area() 没有传递参数表示获得客户端 IP 地址

### 53\_2 Ip::getClientIp() 获得客户端 IP

通过静态方法 ip::getClientIp() 获得客户端 IP 地址

# 54. 错误异常处理

后盾 HDPHP 框架错误处理非常高效全面，可以针对需要自定显示及错误处理方式。

## 54\_1 开启调试模式

开启调试模式有两种方式

1. 在单入口文件定义常量 `define('DEBUG',true)`
2. 在配置项目中修改配置 '`DEBUG' => 1` 即开启了 DEBUG 调试模式

## 54\_2 自定义错误内容显示方式

后盾 HDPHP 框架可以完全根据程序员要求，自定义显示错误内容，因为全面的错误提示显示的内容会比较多，有时我们不需要得到这么多信息，比如说在 AJAX 测试时返回太多信息不利于代码纠错，那么我们可以个性配置文件中的相应配置项轻松实现自定义的显示内容。

以下是配置文件的相应选项

|                   |          |
|-------------------|----------|
| 'LOG_START' => 1  | 是否开启日志记录 |
| 'SHOW_SYSTEM'=>1  | 显示系统信息   |
| 'SHOW_INCLUDE'=>1 | 显示加载文件信息 |
| 'SHOW_CACHE' => 1 | 显示缓存监控   |

在后盾 HDPHP 框架产品中还有一项人性化的设置，当系统关闭调试模式后，为了避免出现白页的情况，可以在配置文件中指定配置项来显示友好的文字内容，这样的好处是。用户或程序员不会为出现白页而困扰，同时也会刚使用 HDPHP 框架的朋友提供了如何查看错误信息的几种方式说明。

修改以下配置项即可

`'ERROR_MESSAGE' => '系统发生错误 ...'`

## 55. 日志处理

完善的程序要有合理日志处理功能，后盾 HDPHP 框架的日志功能非常完善，根据不同错误类型提供完善的处理机制，在关闭调试模式后，也可以查看系统日志来对系统运行情况进行全面了解。根据日期分配日志文件同时文件名经过[密匙加密处理](#)所以非常安全，日志文件会完整的保存出错文件名、代码行号、函数名、出错时间等详细信息。

当然也可以对日志的处理方式进行自定的配置，自定义的思想是 HDPHP 框架的核心发展理念。通过以下配置项的设置来对后盾 HDPHP 框架进行详细的设置。

|   |           |
|---|-----------|
| 'LOG_SAVE' => 1                               | 错误写入日志文件中 |
| 'LOG_KEY' => 'houdunwang.com'                 | 日志文件保密密匙串 |
| 'LOG_SIZE' => 2000000                         | 日志文件大小    |
| 'LOG_TYPE' => array('error', 'notice', 'sql') | 保存日志类型    |

# 56. 核心函数库

## 56\_1 C() 载入或设置配置

网站运行离不开配置项，通过 C() 函数可以方便的读取与设置配置项，也是开发中最常使用的函数。

### 示例

```
1 <?php
2 class indexControl extends Control {
3     function index() {
4         echo C('db_host');// 输出配置文件中的 db_host 配置项
5         echo '<br/>';
6         echo C('db_host','localhost');// 设置配置项 db_host 为 localhost
7     }
8 }
9 输出结果为 127.0.0.1 与 localhost
```

## 56\_2 O() 函数

O() 函数用于生成对象，或者生成对象后执行类方法同时可以传入参数

### 参数：

- a. \$class      类名称
- b. \$method     方法名称
- c. \$args        参数（数组形式）

### 示例

```
1 <?php
2 class indexControl extends Control {
3     function index() {
4         O('indexControl','html',array(2,3));
5     }
6 }
```

执行 indexControl 中的 html 方法并且传入参数 2,3

## 56\_3 control() 实现化控制器

control() 方法可以方便的实例化控制器对象

### 示例 1: 实例化当前应用控制器

```
1 实例化当前应用中的 UserController 控制器
```

```
2 Control('User');
```

### 示例 2 : 实例化其他应用的控制器

```
1 // 实例化 Admin 应用的 LoginControl 控制器
```

```
2 control('Admin/Login')
```

## 56\_4 tag() 调用标签函数

tag() 函数可以方便的在控制器或模板视图中调用标签

### 示例 1 : 控制器中调用标签

```
1 class IndexControl extends Control
```

```
2 {
```

```
3     protected $upload;
```

```
4     public function index()
```

```
5     {
```

```
6         $up = tag('upload',array('name' => 'upfile','limit'=>1, 'width' => 88, 'height' => 78));
```

```
7         $this->assign('up', $up);
```

```
8         $this->display();
```

```
9     }
```

```
10 }
```

### 示例 2 : 模板视图中调用标签

```
1 {|tag('upload',array('name' => 'upfile','limit'=>1, 'width' => 88, 'height' => 78))|}
```

## 56\_5 U() 生成 url

通过 HD PHP 框架的 U() 函数可以根据配置文件中的 URL 配置项生成规范的 URL , 只需要修改配置项就可以让整个网站 URL 自动适应 , 不用手动修改全部 URL , 同时 U 方法会按路由规则进行解析处理

语法 : U(\$pathinfo, \$args = array())

### 示例 1 传一个参数为方法

```
echo U('add');
```

pathinfo 访问生成为 : http://localhost/hdcms/index.php/add.html

普通 URL 访问 : http://localhost/hdcms/index.php?m=add

### 示例 2 控制器 / 方法

```
echo U('channel/add');
```

pathinfo 访问生成为 : http://localhost/hdcms/index.php/channel/add.html

普通 URL 访问 : http://localhost/hdcms/index.php?c=channel&m=add

### 示例 3 应用 / 控制器 / 方法

入口文件定义了 `define('GROUP_NAME', 应用组)` 即以应用组形式配置时才可用

```
1 echo U('admin/channel/add');
```

pathinfo 访问生成为 : http://localhost/hdcms/index.php/admin/channel/add.

html

普通 URL 访问 : http://localhost/hdcms/index.php?a=admin&c=channel&m=add

### 示例 4 路径中传参

```
echo U('channel/add/name/houdunwang/uid/1');
```

pathinfo 访问生成为 : http://localhost/index.php/channel/add/uid/1/name/  
houdunwang.html

普通 URL 访问 : http://localhost/index.php?c=channel&m=add&uid=1&name=  
houdunwang

### 示例 5 路径中传参

```
echo U('channel/add?name=houdunwang&uid=1');
```

pathinfo 访问生成为 : http://localhost/index.php/channel/add/uid/1/name/  
houdunwang.html

普通 URL 访问 : http://localhost/index.php?c=channel&m=add&uid=1&name=  
houdunwang

### 示例 6 传字符串参数

```
echo U('channel/add','name=houdunwang&uid=1');
```

pathinfo 访问生成为 : http://localhost/index.php/channel/add/name/  
houdunwang/uid/1.html

普通 URL 访问 : http://localhost/index.php?c=channel&m=add&name=houdun  
wang&uid=1

## 示例 7 传数组参数

```
echo U('channel/add',array('name'=>'houdunwang','uid'=>1));
```

pathinfo 访问生成为 : http://localhost/index.php/channel/add/name/houdunwang/uid/1.html

普通 URL 访问 : http://localhost/index.php?c=channel&m=add&name=houdunwang&uid=1

## 示例 8 普通路由规则转换

如果配置文件中定义了如下普通路由规则

```
1 'route' => array(  
2   'info/:city/:row'=>'index/index'  
3 )
```

## U() 函数运算结果

```
1 echo U('index/index/city/beijing/row/200');  
2 echo U('index','city=beijing&row=200');
```

以上两种方式都是正确的设置结果为 : http://localhost/hdcms/index.php/info/beijing/200

## 示例 9 正则表达式路由规则转换

如果配置文件中路由定义规则如下

```
1 'route' => array(  
2   '/^([a-z]+?)_(\d+)/'=>'index/index/city/#1/row/#2'  
3 )
```

## U() 函数运算结果

```
echo U('index', 'city=beijing&row=168');
```

经过路由识别后生成的 URL 为 http://localhost/hdcms/index.php/beijing\_168

## 示例 10 普通模式正则路由规则转换

如果配置文件中路由定义规则如下

```
1 'route' => array(  
2   ':city_:row'=>'c=index&m=index'  
3 ),
```

## U() 函数运算结果

```
1 echo U('index', 'city=beijing&row=168');
```

经过路由识别后生成的 URL 为 [http://localhost/hdcms/index.php?beijing\\_168](http://localhost/hdcms/index.php?beijing_168)

## 56\_6 load() 文件加载

load 提供方便的文件加载处理，示例如下

[加载应用 Extend/Lib 目录下的 functions.php 文件](#)

```
load('functions');
```

[加载应用目录下 Common 目录下的 functions.php 文件](#)

```
load('@.Common.functions');
```

[加载网站根目录下 Common 目录下的 functions.php 文件](#)

```
load('./Common/functions');
```

## 56\_7 404 错误处理

处理 404 错误，开启 debug 时会抛出异常，关闭后会加载 404 错误模板文件。

[示例](#)

```
_404(__METHOD__.' 页面没找到 ','404.html');
```

## 56\_8 addslashes\_d() 转义字符串

系统提供了转义函数 addslashes 但只能转义字符串，通过本函数不仅可以对字符串进行转义也可以对数组及对象中的属性值的内容进行转义。

[示例](#)

```
1 function index() {  
2     $data = array(  
3         'username'=> '向军',  
4         'msg'=> "后盾网 \\"PHP\ 教育"  
5     );  
6     $data = addslashes_d($data);  
7     p($data);  
8 }
```

结果：

```
1 Array  
2 ( [username] => 向 \'军  
3 )
```

```
4 [msg] => \'后盾网 \'\\PHP\\教育  
5 )
```

## 56\_9 array\_change\_key\_case\_d() 改变数组键名大小写

PHP 提供了一个将数组的键名转为大写或小写的函数，但是只能转换一维数组，本函数可以方便的将多维数组的键名转为大写或者小写，参数 1 为大写，0 为小写。

### 示例

```
1 <?php  
2 class indexControl extends Control {  
3     function index() {  
4         $arr = array(  
5             'name'=>'后盾网 ',  
6             'url'=>array(  
7                 'bbs'=>'bbs.houdunwang.com',  
8                 'edu'=>'edu.houdunwang.com'  
9             )  
10        );  
11        $newArr = array_change_key_case_d($arr);  
12        p($newArr);  
13    }  
14 }
```

以上代码执行后的结果为：

```
1 Array  
2 (  
3     [NAME] => 后盾网  
4     [URL] => Array  
5         (  
6             [BBS] => bbs.houdunwang.com  
7             [EDU] => edu.houdunwang.com  
8         )  
9  
10 )
```

## 56\_10 array\_change\_value\_case() 数组的值大小写转换

本函数用于转换数组值的大小写，支持递归的值大小写转换

## 56\_11 array\_defined() 将数组转为常量

传入数组将数组的键名做为常量名，键值做为常量值

### 示例

```
1 <?php
2 class indexControl extends Control {
3     function index() {
4         array_defined(array('WEBNAME'=>'后盾网'));
5         echo WEBNAME;
6     }
7 }
```

以上代码生成常量 WEBNAME 值为 '后盾网'

## 56\_12 array\_key\_exists\_d() 不区分大小写检测键名是否存在

array\_key\_exists\_d() 提供与系统函数 array\_key\_exists() 同样的功能，只是不区分大小写，示例如下：

```
var_dump(array_key_exists_d('Hd',array('hd'=>'后盾网'))); // 返回真
```

## 56\_13 array\_to\_String() 将数组转为字符串表示形式

## 56\_14 control() 实例化控制器对象

语法：function control(\$control)

| 参数        | 说明   |
|-----------|------|
| \$control | 控制器名 |

```
1 control('index');// 实例化当前应用中的 index 控制器对象
```

```
2 control('member/index');// 实例化 member 应用 index 控制器对象
```

## 56\_15 A() 执行控制器中的方法 ( 支持分组 )

语法 : function A(\$arg, \$args = array())

示例 :

- 1 A('show',array('hdphp',2030));// 执行当前控制器中的 show 方法 , 并参数参数 hdphp 与 2030
- 2 A('User/add');// 执行 User 控制器中的 add 方法
- 3 control('member/index/show');// 执行 member 应用中的 indexControl 控制器的 show 方法
- 4 control('member/index/show',array('hdphp',2030));// 执行 member 应用中的 indexControl 控制器的 show 方法 , 并传递参数 hdphp,2030

## 56\_16 encrypt() 加密处理

**encrypt(\$data, \$key = null)**

\$data 加密字符串                  \$key 密钥

- 1 encrypt(' 这是加密数据 ');

## 56\_17 decrypt() 解密方法

- 1 echo decrypt(encrypt(' 后盾网人人做后盾 '));

## 56\_18 dir\_create() 函数

dir\_create() 函数用于创建目录 , 只需要指定路径本函数会递归的创建目录 , 比如传入参数 hd/houdunwang/bbs , 函数会创建 hd 目录再创建 houdunwang 目录再创建 bbs 目录

示例

- 1 <?php
- 2
- 3 class indexControl extends Control {
- 4     function index() {
- 5         if(dir\_create('h/houdunwang/bbs')){
- 6             \$this->success(' 创建目录成功 ');
- 7         }
- 8     }

以上执行会自动创建 h/houdunwang/bbs 目录

## 56\_19 p() 别名 ( show,dump)

由于 print\_r 等打印函数会将内容在一行输出不利于阅读，通过本函数可以分行格式化输出内容，更加便于阅读加快调试速度。

## 56\_20 halt() 输出错误信息

如果开启 DEBUG 时输出错误信息内容，当 DEBUG 关闭时调用 \_404() 函数输入 404 页面。无论 DEBUG 开启与关闭都会终止 PHP 脚本执行

## 56\_21 firephp() 调试插件

需要 firefox 下安装 firebug 和 firephp 插件，使用 firephp() 方法进行调试非常方便，尤其是在查看 Ajax 数据时。

使用方法：<http://bbs.houdunwang.com/thread-53710-1-1.html>

## 56\_22 extension\_exists() PHP 扩展模块是否存在

通过本函数判断 php 扩展模块是否存在，比如说在系统安装时判断 mysqli 环境是否存在，如果存在则通过 mysqli 扩展操作数据，示例如下：

```
extension_exists('mysqli'); // 判断 mysqli 扩展是否存在
```

## 56\_23 file\_exists\_case() 区分大小写的文件判断

```
1 file_exists_case('index.php');
```

## 56\_24 get\_defines 获得常量

获得常量系统，默認為获得用户定义常量

|      |  |                         |
|------|--|-------------------------|
| 语法   | get_defines(\$name = "",\$value=null, \$type = 'user') |                         |
| 参数说明 | name   | 常量名称                    |
|      | value  | 如果常量不存在时的返回值            |
|      | type   | user 为用户定义常量，true 为所有常量 |

### 示例

- 1 get\_defines(); // 获得所有用户定义常量
- 2 get\_defines('APP');// 获得常量名为 APP 的值
- 3 get\_defines('HD','后盾');//HD 常量存在返回其值，否则返回 '后盾'

```
4 get_defines('',true); // 获得系统中的所有常量定义
```

---

## 56\_25 get\_size() 根据大小返回标准单位 KB MB GB 等

传入大小单位为字节，返回以 KB 或者 MB 或者 GB 为单位的字符串，比如说我们想在视图层显示文件大小，以 PHP 默认的字节单位不便于阅读而 get\_size() 函数可以帮上你的忙。

### 示例

```
1 <?php
2 class indexControl extends Control {
3     function index() {
4         echo get_size('1000');
5     }
6 }
```

---

以上代码返回 0.9766 KB

## 56\_26 go() 跳转到指定 url

跳转到指定 url

### 示例

```
1 go('http://bbs.houdunwang.com');
```

---

跳转到 <http://bbs.houdunwang.com>

## 56\_27 browser\_info() 获得浏览器版本

使用 browser\_info() 可以方便的获得客户端浏览器，如果是 IE 浏览器将返回 msie9 等形式，开发者可以借此做出针对不同浏览器的业务处理。

56\_28 image\_type\_to\_extension() 根据类型获得图像扩展名

56\_29 ip\_get\_client() 显示客户端

56\_30 is\_ssl() 是否为 SSL 协议

56\_31 json\_encode() 对变量进行 JSON 编码

56\_32 json\_decode 对 JSON 格式的字符串进行编码

56\_33 load() 载入文件

load() 方法载入文件要比 include、include\_once、require、require\_once 更加高效，由于函数内置静态缓存机制，如果一个文件被多次加载时不会重复加载，也不会产生错误。

示例

```
1 load('config.inc.php');// 加载文件 config.inc.php  
2 load('@.Extend.Lib.funntions');// 加载 应用目录 /Extend/Lib/functions.php 文件
```

56\_34 md5\_d 方法

用于将任何内容包括数组、对象、字符串、数值类型生成 md5 序列化的字符串

示例

```
1 $array=array('webname'=>'后盾网');  
2 echo md5_d($array);  
3 # 输出结果 3deca1aa87fc07e0a649432408bdd4b2
```

56\_35 mobile\_area() 电话号码来源

通过本函数获得电话号码所在地，在分类信息等网站来说是一个很实用的功能。

```
mobile_area('13121111111');
```

注：HDPHP 框架提供的电话号数量有限，所以有些号码检索不到，大家可以从网上下载数据多的库自行配置

56\_36 php\_merge() 合并 php 文件

多个 PHP 文件进行合并，如果将参数 \$delSpace 设置为 1 则合并时会删除文件中的空

## 56\_37 print\_const(\$view=true) 打印所有常量

由于 HDPHP 框架运行中，包含大量框架定义常量及用户代码定义常量，通过本函数可以方便的知道都有哪些常量定义了及其具体的内容值，示例如下：

```
print_const(); // 打印常量  
print_const(false); // 返回所有用户定义常量
```

## 56\_38 rand\_str 获得随机字符串

## 56\_39 set\_http\_state() 设置 HTTP 状态信息

可传参数为：200,301,302,400,403,404,500,503

## 56\_40 compress() 压缩 PHP 代码

去空格，去除注释包括单行及多行注释，参数为字符串内容

```
compress('hdphp.php');
```

## 56\_41 data\_format() 数据安全处理

将函数作用于数据，比如说可以在插入数据时对数据一次性进行转义或实体化处理，如果不存递第 2 个参数系统会使用

示例：

```
1  $_POST=array(  
2      'html'=><script>alert(2);</script>,  
3      'name'=>'houdunwang.com'  
4  );  
5  p(data_format($_POST,'htmlspecialchars,strtoupper')); 或  
6  p(data_format($_POST,array('htmlspecialchars,strtoupper')));  
7  结果：  
8  Array  
9  (  
10 [html] => &LT;SCRIPT&GT;ALERT(2);&LT;/SCRIPT&GT;  
11 [name] => HOUDUNWANG.COM
```

## 56\_42 stripslashes\_d() 去除转义字符

去除数组或字符串中的转义内容，使用方法与 stripslashes\_d 一样，就不举示例了。

## 56\_43 throw\_exception() 抛出异常

## 56\_44 url\_param\_remove() 移除 URL 中 GET 参数

通过本函数可以移除 URL 中的指定 GET 变量，在进行参数项检索时很有帮助，比如分类信息应用中租房列表中的搜索会按价格、厅室、出租类型检索通过本函数可以完成此功能起到一定帮助

### 示例 1：

比如当前 URL 为 `http://localhost/index.php/index/city/beijing/price/300_500`

```
1 class indexControl extends Control{
2     function index(){
3         echo url_param_remove('price');
4     }
5 }
```

### 示例 2：

比如当前 URL 为

```
1 class indexControl extends Control{
2     function index(){
3         $url = 'http://localhost/index.php/index/city/beijing/price/300_500';
4         echo remove_url_param('price',$url);
5     }
6 }
```

通过本函数传入参数 price 即可移除 URL 中的 price 参数，最终 URL 为 `http://localhost/index/city/beijing`

## 56\_45 date\_before() 获得几秒、几分、几年前

在微博，论坛等项目中，经常要获得微博发表于几分前，几年前等字符串表示，通过 date\_before() 函数可以轻松实现，本函数需要 2 个参数，说明如下：

参数说明

参数 1：要计算的时间戳

参数 2：时间单位，必须为数组，默认为 array('年','月','日','星期','小时','分钟','秒')

## 56\_46 get\_uuid() 获得唯一值 uuid

UUID 是指在一台机器上生成的数字，它保证对在同一时空中的所有机器都是唯一的

[示例](#)

get\_uuid('-') // 获得唯一 uuid 以中线连接

---

# 57. 核心常量

可以使用 `print_const()` 方法打印所有用户定义常量。

## 57\_1 基本常量

|                  |                 |
|------------------|-----------------|
| DEBUG            | 调试模式            |
| APP              | 应用名             |
| CONTROL          | 控制器名            |
| METHOD           | 方法名             |
| HDPHP_VERSION    | HDPHP 框架版本号     |
| IS_WIN           | 是否为 Windows 系统  |
| MAGIC_QUOTES_GPC | 是否自动转义          |
| REQUEST_METHOD   | 请求类型            |
| IS_GET           | 是否为 GET 请求      |
| IS_POST          | 是否 POST 请求      |
| IS_AJAX          | 是否 IS_AJAX 请求   |
| IS_PUT           | 是否 IS_PUT 请求    |
| IS_DELETE        | 是否 IS_DELETE 请求 |
| NOW              | 当前时间            |
| NOW_MICROTIME    | 当前时间 (微妙)       |
| CHARSET          | 字符集             |

## 57\_2 目录与文件常量

|                    |        |
|--------------------|--------|
| HDPHP_PATH         | 框架核心目录 |
| ROOT_PATH          | 网站根目录  |
| APP_PATH           | 应用目录   |
| TEMP_PATH          | 编译目录   |
| TEMP_FILE          | 编译文件   |
| COMMON_PATH        | 公共目录   |
| COMMON_CONFIG_PATH | 公共配置文件 |
| COMMON_MODEL_PATH  | 公共模型目录 |

|                      |          |
|----------------------|----------|
| COMMON_LANGUAGE_PATH | 公共语言包目录  |
| COMMON_EVENT_PATH    | 公共事件目录   |
| COMMON_TAG_PATH      | 公共模板标签目录 |
| COMMON_LIB_PATH      | 公共包含目录   |
| CONTROL_PATH         | 控制器目录    |
| MODEL_PATH           | 模型目录     |
| CONFIG_PATH          | 配置目录     |
| EXTEND_PATH          | 扩展目录     |
| EVENT_PATH           | 事件目录     |
| LANGUAGE_PATH        | 语言目录     |
| TAG_PATH             | 标签目录     |
| LIB_PATH             | 自动加载目录   |
| TPL_PATH             | 模板目录     |
| PUBLIC_PATH          | 模板公共目录   |
| COMPILE_PATH         | 模板编译目录   |
| CACHE_PATH           | 缓存目录     |
| TABLE_PATH           | 表结构缓存目录  |
| LOG_PATH             | 日志目录     |

## 57\_3 URL 常量

|               |                 |
|---------------|-----------------|
| _HOST_        | 主机域名            |
| _ROOT_        | 单入口文件所在目录       |
| _WEB_         | 主页 URL 地址       |
| _GROUP_       | 应用组 URL 地址      |
| _APP_         | 当前应用 URL 地址     |
| _CONTROL_     | 模块控制器 URL 地址    |
| _HISTORY_     | 来源 URL 地址       |
| _METH_        | 当前控制器方法 URL 的地址 |
| _URL_         | 完整 URL 地址       |
| _TPL_         | 模版目录 URL 地址     |
| _CONTROL_TPL_ | 控制器模板目录         |

|                      |                    |
|----------------------|--------------------|
| _PUBLIC_             | 应用模板目录下 Public 目录  |
| _STATIC_             | 应用模板目录下的 Static 目录 |
| _HDPHP_              | HDPHP 框架目录         |
| _HDPHP_DATA_         | HDPHP 框架中的 data 目录 |
| _HDPHP_TPL_          | HDPHP 框架核心模板目录     |
| _HDPHP_EXTEND_ HDPHP | 框架扩展目录             |

## 58. 编辑器使用方法

HDPHP 框架提供了方便、快速、灵活的编辑器创建机制，提供便捷的标签调用机制来创建与调整编辑器。

HDPHP 框架集成 ueditor 与 kindeditor 两种编辑器，通过修改配置文件 editor 项即可改变当前编辑器类型，同时也可以通过编辑器标签 editor 设置，具体使用方法如下。

**注：**

1. 使用 ueditor 编辑器时页面必须定义 html 与 body 标签
2. 必须引入 jquery

| 标签属性           | 说明                                |
|----------------|-----------------------------------|
| style          | 1 为完整风格 2 精简风格                    |
| name           | 为表单的字段名即 name                     |
| content        | 为编辑器的默认值，可以通过视图的 assign 分配变量      |
| water          | 图片是否加水印 true 加水印 false 不加水印       |
| width          | 编辑器宽度                             |
| height         | 编辑器高度                             |
| imageupload    | 精简模式是否显示上传图片按钮 ,true 显示 false 不显示 |
| maximagewidth  | 上传图片最大宽度值（超过此宽度自动裁切）              |
| maximageheight | 上传图片最大高度值（超过此高度自动裁切）              |
| uploadszie     | 允许上传文件大小（单位 kb）                   |
| readonly       | 编辑区域是否是只读的                        |
| wordCount      | 是否开启字数统计                          |
| maxword        | 允许的最大字符数                          |
| autoClear      | 清除编辑器初始内容                         |

| 标签属性 | 说明       |
|------|----------|
| php  | php 处理脚本 |

## 代码高亮

如果要高亮显示代码只需要在显示页的 <head> 标签中加入 <highlight/> 就可以了。

## 58\_1 ueditor 编辑器使用

通过 HD 编辑器标签 editor 可以快速创建 ueditor 百度编辑器，不用设置复杂的配置即可创建及调整编辑器。

### 示例 1 简单的编辑器调用

```
1 <html>
2   <body>
3     <form action='__CONTROL__/post' method='post'>
4       <ueditor name='con' />
5       <input type='submit' />
6     </form>
7   </body>
8 </html>
```

这是最精简的调用方式

### 示例 2 设置编辑尺寸

```
1 <ueditor width='600' height='200' name='con' />
```

### 示例 3 设置默认值

```
1 <ueditor content='后盾网 人人做后盾' name='con' />
```

### 示例 4 上传图片加水印

```
1 <ueditor content='后盾网 人人做后盾' water='true' name='con' />
```

### 示例 5 图片自动缩放

```
1 <ueditor name='con' maximagewidth='200' maximageheight='300' />
```

2 图片如果超过 200\*300 就会自动缩放

## 示例 6 获得编辑器内容

UE.getEditor(' 标签的 name 名前加 hd\_ ').getContent() 方法用于获得编辑器内容，需要在编辑器内容加载完后才可以获得值，所以可以把方法加入按钮等元素的事件处理函数中如下

```
1 <ueditor name='con'>
2 <button id='houdunwang' type='button'> 获得内容 </button>
3 <script>
4 document.getElementById('houdunwang').onclick=function(){
5   alert(UE.getEditor('hd_con').getContent());
6 }
7 </script>
```

UE.getEditor('editor').getContent() 函数的参数为标签 name 值前加 hd\_，如读取标签 <ueditor name='con'> 的内容方式为：UE.getEditor('hd\_con').getContent()

## 示例 7 获得纯文本内容

通过 UE.getEditor(' 标签的 name 名前加 hd\_ ').getContentTxt() 方法可以获得去除 html 标签的纯文本内容

需要在编辑器内容加载完后才可以获得值，所以可以把方法加入按钮等元素的事件处理函数中如下

```
1 <ueditor name='con'>
2 <button id='houdunwang' type='button'> 获得内容 </button>
3 <script>
4 document.getElementById('houdunwang').onclick=function(){
5   alert(UE.getEditor('hd_con').getContentTxt());
6 }
7 </script>
```

注：UE.getEditor('editor').getContentTxt() 函数的参数为标签 name 值前加 hd\_，如读取标签 <ueditor name='con'> 的内容方式为：UE.getEditor('hd\_con').getContentTxt()

## 示例 8 设置编辑器的内容

UE.getEditor(' 标签的 name 名前加 hd\_ ').setContent(' 欢迎使用 hdphp 框架 ') 通过此方法可以设置编辑器内容

需要在编辑器内容加载完后才可以设置值，所以可以把方法加入按钮等元素的事件处理函数中如下

```
1 <ueditor name='con'>
2 <button id='houdunwang' type='button'> 设置内容 </button>
3 <script>
4 document.getElementById('houdunwang').onclick=function(){
5   UE.getEditor('hd_con').setContent(' 欢迎使用 hdphp 框架 ');
6 }
7 </script>
```

UE.getEditor('editor').setContent() 函数的参数为标签 name 值前加 hd\_，如设置标签 <ueditor name='con'> 的内容方式为：UE.getEditor('hd\_con').setContent(' 欢迎使用 hdphp 框架 ')

## 示例 9 判断编辑器是否有内容

通过 UE.getEditor(' 标签的 name 名前加 hd\_ ').hasContents() 方法可以判断编辑器的内容是否为空不为空返为 true

需要在编辑器内容加载完后才可以获得值，所以可以把方法加入按钮等元素的事件处理函数中如下

```
1 <ueditor name='con'>
2 <button id='houdunwang' type='button'> 判断编辑器内容是否为空 </button>
3 <script>
4 document.getElementById('houdunwang').onclick=function(){
5   alert(UE.getEditor('hd_con').hasContents());
6 }
7 </script>
```

注：UE.getEditor('editor').hasContents() 函数的参数为标签 name 值前加 hd\_，如设置标签 <ueditor name='con'> 的内容方式为：UE.getEditor('hd\_con').hasContents()

## 示例 10 自定义处理图片 PHP 脚本

图片上传分两种情况，第一种是系统自动完成上传处理，第二种是用户创建上传处理方法来完成，下面我们分别来学习使用

### 自行设置上传处理方法

在当前控制器中创建方法 `ueditor_upload`，编辑器会自动把文件上传处理交给这个方法来处理，程序员可以通过 HDPHP 框架的上传类等工具完成上传处理，最后必须完回一个 json 格式的数据，这样就可以了，下面我们来看具体的代码。

```
1 class IndexControl extends Control {  
2     function ueditor_upload() { // 编辑器图片上传处理方法  
3         C('debug', 0); // 因为要返回 json 数据，所以不能有调试结果等字符出现  
4         $title = htmlspecialchars($_POST['pictitle'], ENT_QUOTES); // 对标题实体化  
5         $upload = new Upload(ROOT_PATH); // 实例化上传类  
6         $file = $upload->upload(); // 上传图像  
7         if (!$file) {//发生上传错误  
8             echo '{"title":"' . $upload->error . "','state':'' . $upload->error . "'"};  
9         exit;  
10    }  
11    $info = $file[0];  
12    $info['url'] = __ROOT__ . '/' . $info['path'];  
13    $info['state'] = 'SUCCESS';  
14    echo '{"url":"' . $info['url'] . "','title':'' . $title . "','original':'' . $info['filename'] . "','state':'' .  
$info['state'] . "'"};  
15 }  
16 }
```

### 返回值 JSON 说明

| 标签属性  | 说明        |
|-------|-----------|
| url   | 为上传成功图片路径 |
| title | 为上传图片标题   |

| 标签属性  | 说明                               |
|-------|----------------------------------|
| state | 图片上传成功必须输入 SUCCESS 字符串，其他字符为错误内容 |

如果没有在控制器中设置 editor\_pload 方法，系统将自动采用内部上传机制进行上传，一般情况下没有必要编写上传方法，可以直接使用内容自动上传机制

### 对上传内容的配置

上传的图片，可能需要指定上传图片类型，或者将图片储存到不同的目录，这种需求的配置与使用上传类一样，根据图片上传类的配置方法配置即可，当然需要创建自定义上传方法 editor\_pload 方法。

还有一点要说明一下，如果要将上传成功的图片保存进数据库等类似的操作，程序员只能通过自定义上传处理完成图片的上传，因为自动上传机制不确定图片数据表的表结构，所以不能自动完成这些工作，比如可以通过正则将图片地址匹配出来。

## 58\_2 keditor 编辑器使用

通过标签 keditor 可以快速创建 kindeditor 编辑器，不用设置复杂的配置即可创建及调整编辑器。

语法：`<ckeditor content="" name='con'>`

| 标签属性          | 说明                             |
|---------------|--------------------------------|
| style         | 编辑器布局 1 为完整模式 2 为精简模式          |
| name          | 表单的字段名                         |
| content       | 编辑器的默认值，可以通过视图的 assign 分配变量    |
| water         | 是否加水印 true 加水印 false 不加水印      |
| width         | 编辑器宽度                          |
| height        | 编辑器高度                          |
| imageupload   | 是否显示图片上传按钮 true 为显示 false 为不显示 |
| maximagewidth | 上传图片超过这个宽度值，系统进行缩放处理 单位像素      |

| 标签属性           | 说明                           |
|----------------|------------------------------|
| maximageheight | 上传图片超过这个高度值，系统进行缩放处理 单位像素    |
| uploadsize     | 允许上传文件大小，单位是 KB              |
| filter         | 过滤 HTML 代码，true 过滤 false 不过滤 |

### 示例 1 调用编辑器

```
<keditor content='后盾网人人做后盾' width='800' height='300' name='con'/>
```

创建宽度 800px，高度 300px 的编辑器，默认值为 ' 后盾网人人做后盾 '

### 示例 2 上传图片自动缩放

```
<keditor name='hdphp' maximageheight='30' maximagewidth='30'>
```

上传图片尺寸超过 300\*200 将自动缩放处理

### 示例 3 取得 html 内容

```

1 <keditor name='hdphp'>
2 <button id='houdunwang' type='button'> 获得编辑器内容 </button>
3 <script type='text/javascript'>
4 window.onload=function(){
5   document.getElementById('houdunwang').onclick=function() {
6     alert(hd_hdphp.html());
7   };
8 }
9 </script>

```

hd\_hdphp.html() 函数的对象为标签的 name 值前加 hd\_，如编辑器标签为 <keditor name='con'> 获得编辑器内容方式为 : hd\_con.html()

### 示例 4 取得只包含 img 标签及文本

```

1 <keditor name='hdphp'>
2 <button id='houdunwang' type='button'> 获得编辑器内容 </button>
3 <script type='text/javascript'>
4 window.onload=function(){
5   document.getElementById('houdunwang').onclick=function() {
6     alert(hd_hdphp.text());
7   };
8 }
9 </script>

```

```
7      };
8  }
9 </script>
```

注：hd\_hdphp.text() 函数的对象为标签的 name 值前加 hd\_，如编辑器标签为<keditor name='con'> 取得编辑器内容的方式为：hd\_con.text()

#### 示例 5 设置编辑器 html 内容

```
1 <keditor name='hdphp'>
2 <button id='houdunwang' type='button'> 设置编辑器 HTML 内容 </button>
3 <script type='text/javascript'>
4 window.onload=function(){
5   document.getElementById('houdunwang').onclick=function() {
6     alert(hd_hdphp.html('后盾网 hdphp 框架'));
7   };
8 }
9 </script>
```

注：hd\_hdphp.html() 函数的对象为标签的 name 值前加 hd\_，如编辑器标签为<keditor name='con'> 设置编辑器内容的方式为 hd\_con.html('后盾网 hdphp 框架'))

#### 示例 6 设置编辑器内容文本内容

```
1 <keditor name='hdphp'>
2 <button id='houdunwang' type='button'> 设置编辑器文本内容 </button>
3 <script type='text/javascript'>
4 window.onload=function(){
5   document.getElementById('houdunwang').onclick=function() {
6     alert(hd_hdphp.text('后盾网 hdphp 框架'));
7   };
8 }
9 </script>
```

#### 示例 7 判断编辑器内容是否为空

```
1 <keditor name='hdphp'>
2 <button id='houdunwang' type='button'> 获得编辑器内容 </button>
3 <script type='text/javascript'>
4 window.onload=function(){
```

```
5     document.getElementById('houdunwang').onclick=function() {
6         alert(hd_hdphp.isEmpty());
7     };
8 }
9 </script>
```

注：hd\_hdphp.isEmpty() 函数的对象为标签的 name 值前加 hd\_，如编辑器标签为 <keditor name='con'> 判断编辑器内容是否为空的方式为：hd\_con.isEmpty()

### 示例 8 自定义上传图片处理 PHP 脚本

在当前控制器中创建方法 keditor\_upload（名称必须是这个），编辑器会自动把文件上传处理交给这个方法来处理，程序员可以通过 HD PHP 框架的上传类等工具完成上传处理，最后必须完回一个 json 格式的数据，这样就可以了，下面我们来看具体的代码。

```
1 <?php
2 class indexControl extends Control{
3     function keditor_upload(){
4         header('Content-Type: text/html; charset=utf-8');
5         C('debug',0);// 因为输出 json，所以关闭错误信息
6         $upload = new upload();
7         $file = $upload->upload();// 上传
8         if(!$file){// 上传失败
9             echo json_encode(array('error' => 1, 'message' =>$upload->error));exit;
10        }
11        $file_url = $file[0]['path'];
12        echo json_encode(array('error' => 0, 'url' => $file_url));exit;
13    }
14 }
15 ?>
```

# 59. AJAX 多文件上传组件

后盾 HDPHP 框架集成的方便的在线文件上传组件，通过 flash+jquery+JSON 及系统内部 PHP 文件处理机制，不需要程序员通过繁琐的配置只需要通过一行标签即可以完成上传组件的指定，所有配置系统自动完成。



## 59\_1 使用上传组件

标签的使用非常简单，程序员只需要根据场景设置相应属性即可，请看下面示例

语法：`<upload size='5' type='jpg,png,gif' limit='5' name='upload'/>`

注：此插件需要 jquery 支持，所以要先自行加载 jquery

| 标签属性  | 说明   |
|-------|--|
| name  | 隐藏表单项的 name 属性，php 端用 <code>\$_POST</code> 接收即可使用  |
| size  | 上传文件允许大小单位为 MB，如果不设置默认值为 2MB   |
| alt   | 是否显示描述，即图片下面的文本框，true 显示 false 不显示   |
| type  | 允许上传文件的类型，如果不设置默认值为 <code>*.*</code> 即允许所有文件， <code>*.jpg, *.png</code> 为允许 jpg 与 png 文件类型 |
| limit | 允许上传的文件数量，如果不设置默认值为 10 个文件   |
| data  | 编辑图片时分配的数组，具体使用请看下面示例  |
| water | 上传图片是否加水印，true 加水印 false 不加水印  |

| 标签属性                  | 说明   |
|-----------------------|--|
| waterbtn              | 是否显示加水印复选框 true 显示 false 不显示   |
| dir                   | 上传文件存放目录，如果不设置默认值为系统配置项中指定的上传目录  |
| width                 | 预览图的宽度   |
| height                | 预览图的高度   |
| thumb                 | 生成缩略图尺寸，支持生成多个缩略图，如设置值为 '200,200,300,300' 表示生成 2 张图第 1 张图为 200x200 第 2 张图为 300x300 |
| input_type            | 来源表单类型，如：是通过 input 或 img 请求的   |
| upload_img_max_width  | 最大宽度（超过这个尺寸，会进行图片裁切），默认为配置项 UPLOAD_IMG_MAX_WIDTH 值                                 |
| upload_img_max_height | 最大高度（超过这个尺寸，会进行图片裁切）默认为配置项 UPLOAD_IMG_MAX_HEIGHT 值                                 |
| message               | 是否显示上传文件大小等提示信息 true 是 false 不显示   |

## 示例 1 上传标签演示

### 视图代码

```

1 <jquery/>
2 <form action='__CONTROL__/up' method='post'>
3 <upload name='up1'>
4 <input type='submit' />
5 </form>
```

### PHP 脚本

```

1 <?php
2 class IndexControl extends Control {
3     function up(){
4         p($_POST);
5     }
6 }
```

## 示例 2 上传图片时生成不同尺寸图片

```
1 <upload name='img' thumb='200,200,500,500'/>
```

生成 2 张图片第 1 张为 200x200 文件名以 '\_200x200. 扩展名 ' 结尾 , 第 2 张图片为 500x500 , 将表单提交到 PHP 脚本 , 打印出的数据结构如下 :

```
1 Array
2 (
3   [img] => Array
4     (
5       [0] => Array
6         (
7           [alt] => 图片描述
8           [path] => upload/img/78381358007455.jpg
9           [thumb] => Array
10          (
11            [0] => 78381358007455_200x200.jpg
12            [1] => 78381358007455_500x500.jpg
13          )
14
15        )
16      [1] => Array
17        (
18          [alt] => 图片描述
19          [path] => upload/img/32291358007456.jpg
20          [thumb] => Array
21            (
22              [0] => 32291358007456_200x200.jpg
23              [1] => 32291358007456_500x500.jpg
24            )
25          )
26        )
27 )
```

有些情况下并没有 alt 与 thumb 值 , 这并没有问题 HDPHP 框架会自动处理

## 59\_2 上传成功后的回调函数 hd\_upload()

程序员可以在视图中创建 js 函数 hd\_upload() 做为文件上传成功后的进一步处理，当文件上传成功后系统会判断当前视图中有无 hd\_upload 函数，如果存在则自动执行并传递相应参数，示例代码如下。

```
1 <script type='text/javascript'>
2   function hd_upload(file,data){
3   }
4 </script>
```

| 参数   | 说明      |               |
|------|---------|---------------|
| file | index   | 文件编号          |
|      | size    | 上传文件大小        |
|      | name    | 原始文件名         |
| data | stat    | 状态码 1 成功 0 失败 |
|      | url     | url 地址        |
|      | path    | 文件在服务器端存放的路径  |
|      | isimage | 上传文件是否是图片     |

## 59\_3 上传时传递 POST 数据

如果想在上传时传递 POST 数据非法简单，使用 upload 的标签属性 post 就可以，如下：  
<upload name='pic' post='type:'pic',size:'22px' />

## 59\_4 自定义上传处理 PHP 脚本

```
1 public function hd_uploadify(){
2   $data = array();
3   $upload = new upload();
4   $file = $upload->upload();
```

```
5     if ($file) {
6         $data['stat'] = 1;
7         $data['url'] = __ROOT__ . '/' . $file[0]['path'];
8         $data['path'] = $file[0]['path'];
9         $data['thumb'] = array();
10        $data['isimage'] = 1;
11    } else {
12        $data['stat'] = 0;
13        $data['msg'] = $upload->error;
14    }
15    echo json_encode($data);
16    exit;
17 }
```

## 59\_5 编辑图片

编辑图片进行编辑只需要注意传递的数据格式即可

PHP 代码布局

```
1 class IndexControl extends Control {
2     function index() {// 显示编辑器的视图
3         $data = array(
4             0 => array(
5                 'alt' => 'abc',// 图片描述
6                 'path' => 'upload/img/4561358137721.jpg',// 原图路径
7                 'thumb' => array("// 缩略图列表
8                     'upload/img/4561358137721_200x200.jpg',
9                     'upload/img/4561358137721_300x300.jpg'
10                )
11            )
12        );
13        $this->assign('data', $data);
14        $this->display();
15    }
```

### 视图层标签调用格式

```
1 <upload name='cc' limit='3' data='$data' thumb='200,200,300,300'/>
```

---

## 59\_6 删除图片

可以在当前控制器中创建 hd\_uploadify\_del 方法用于处理图片删除动作，如果当前控制器不存在此方法，将使用框架内部删除机制。

## 60. 局部放大镜插件 zoom

开发过程中经常会用到图片局部放大的效果如产品展示页面等，通过标签就可以产生优秀的局部放大特效，为项目开发节省时间，标签展示如下：

```
<zoom data='$file' big='big_div' small='small_div' left='10' right='10'  
width='200' height='200'/>
```

注：此插件需要jquery 支持，所以要先自行加载 jquery

产生效果如下图，如果对显示效果不满意，可以通过 CSS 来控制元素的外观



| 标签属性   | 说明                          |
|--------|-----------------------------|
| data   | 图片数据，下面会详细说明数据格式（必选设置）      |
| big    | 放置大图的 HTML 元素的 ID 值（必选设置）   |
| small  | 放置多张小图的 HTML 元素的 ID 值（必选设置） |
| left   | 局部放大图片距离大图水平距离              |
| top    | 局部放大图片垂直偏移距离                |
| width  | 局部放大图片的宽度                   |
| height | 局部放大图片的高度                   |

## 示例

data 属性用于配置显示图片

每一个产品图包含 3 张图片：1 小缩略图 2 中图 3 局部放大用的大图也是最大的图片，各个图片的缩放比例要一致

```
1 class IndexControl extends Control {  
2     function index() {  
3         $data = array(  
4             array(  
5                 'upload/img/89671358010944_100x100.jpg', // 小图  
6                 'upload/img/89671358010944_300x300.jpg', // 中图  
7                 'upload/img/89671358010944_500x500.jpg', // 大图  
8             )  
9     );  
10    $this->assign('file', $data);  
11    $this->display();  
12 }  
13 }
```

index 视图文件标签书写如下

```
1 <style type='text/css'>  
2     #big_div{clear:both;}  
3     #small_div{clear:both;}  
4     #small_div li{float:left;list-style:none;}  
5 </style>  
6 <zoom data='{$file}' big='big_div' small='small_div' left='10' right='10' width='300'  
height='300' />  
7 <div id='big_div'></div>  
8 <div id='small_div'></div>
```

# 61. 安全建议

对表单项进行安全处理，建议使用模型中的 \_GET() 或 \_POST() 方法处理

对表单字段进行验证，如确定值为数字使用 intval() 函数处理

建议使用框架中的字段映射功能，将表字段名保护起来

使用表单令牌 (TOKEN)，为每一个用户提供唯一的表单令牌

前台使用合理的验证码，防止恶意灌水提交

将服务器目录权限设置为 755

将文件权限设置为 644

将目录与文件所有者与所属组更改为 Apache 服务的所有者与所属组

修改 php.ini 关闭危险函数如 disable\_functions = system,passthru,exec,shell\_exec,popen,phpinfo,escapeshellarg,escapeshellcmd,proc\_close,proc\_open,dl,show\_source,get\_cfg\_var

不要使用默认的 SESSION\_NAME，通过 HDPHP 框架配置项修改

能过相应过滤函数如 htmlspecialchars 等对表单数据进行处理，防止 XSS 攻击

对储存到客户端的 COOKIE 通过框架中的 encrypt() 与 decrypt() 函数进行加密解密处理

使用框架 SESSION 处理机制的 mysql、memcache、redis 等方式，均加入的 TOKEN 机制，防止 SESSION\_ID 被窃取，伪造用户登录

所有密码如 linux、mysql、redis 等要严格设置编码规范

## 61\_1 标准化的重要性和意义

当一个软件项目尝试着遵守公共一致的标准时，可以使参与项目的开发人员更容易了解项目中的代码、弄清程序的状况。使新的参与者可以很快的适应环境，防止部分参与者出于节省时间的需要，自创一套风格并养成终生的习惯，导致其它人在阅读时浪费过多的时间和精力。而且在一致的环境下，也可以减少编码出错的机会。缺陷是由于每个人的标准不同，所以需要一段时间来适应和改变自己的编码风格，暂时性的降低了工作效率。从而使项目长远健康的发展以及后期更高的团队工作效率来考虑暂时的工作效率降低是值得的，也是必须要经过的一个过程。标准不是项目成功的关键，但可以帮助我们在团队协作中有更高的效率并且更加顺利的完成既定的任务。

1. 方便代码的交流和维护，便于日后自己的再次阅读

2. 不影响编码的效率，不与大众习惯冲突
3. 使代码更美观、阅读更方便
4. 使代码的逻辑更清晰、更易于理解
5. 程序员可以了解任何代码，弄清程序的状况
6. 新人可以快速适应环境
7. 防止新接触 PHP 的程序员出于节省时间的考虑，自创一套风格并养成终生的习惯
8. 防止接触 PHP 的程序员一次次犯同样的错误
9. 在一致的环境下，可以减少犯错的机会

在项目开发中使用统一的命名体系，有助于多人协作及提高项目开发效率。HDPHP 命名规范基本以 pascal 规则命名与 camel 命名规则相结合。

1. pascal：第一个单词首字母大写，后面连接的每个单词首字母都大写
2. camel：第一个单词不大写，后面连接的单词首字母大写

## 61\_2 代码标记

代码标记必须使用 `<?php ?>` 形式不要使用简写 `<? ?>` 或者 `<?=$hdphp?>` 形式

1. 使用带有说明性的英文单词或中文缩写，不要缩写单词，不要随便起名，影响编码人员编程

## 61\_3 文件目录规范

1. 文件名要能反应类的内容，最好是和类同名
2. 文件夹命名全部使用小写字母
3. HDPHP 框架模板文件默认以 `.html` 为后缀，可以通过修改配置文件更改模板文件的后缀。
4. 所有包含 PHP 代码的程序文件或半程序文件，应以小写 `.php` 作为扩展名，而不要使用 `.phtml`、`.php3` 等作为扩展名。
5. 模板文件被编译后自动生成的目标程序，以小写 `.php` 作为扩展名，存放于 `./temp` 相应子目录下

## 61\_4 类命名规范

1. 类文件名使用 **pascal** 规则命名，以 `.class.php` 为后缀，例如 `UserControl.class.php`

2. 由于大部分 php 项目均部署在类 Unix 系统上 ,而 Unix 系统是对文件的大小写敏感的 ,所以实例化类时调用的类名要与类文件名大小写一致。
3. 属性命名使用 camel 命名规则 , 并且首字母小写 , 例如 \$dbHost
4. 类方法命名使用 camel 命名规则 , 例如 getTableField()
5. 类方法中以 \_ 开始的方法名如 \_\_call 为魔术方法 ( 即在某一特定时机动态调用 )
6. 参数的命名使用 camel 命名规则 , 例如 \$hostName

## 61\_5 函数规范

1. 函数库以小写 .func.php 作为函数文件扩展名
2. 函数库只能被其他程序引用 , 而不能独立运行 , 其中不能包含任何不属于任何函数或类的程序代码。
3. 函数命名使用全部小写字母并且每个单词以下划线连接 , 例如 get\_control\_file()
4. 参数的命名使用 camel 命名规则 , 例如 \$hostName
5. 具有默认值的参数应该位于参数列表的后面
6. 必须仔细检查并切实杜绝函数起始缩进位置与结束缩进位置不同的现象。

## 61\_6 常量规范

1. 常量命名使用全部大写字母且每个单词以下划线连接 , 例如 PATH HD
2. PHP 的内建值 TRUE 、 FALSE 和 NULL 必须全部采用大写字母书写

## 61\_7 配置项规范

1. 配置项以全部大写字母及下划线连接 , 例如 URL\_REWRITE

## 61\_8 语言包规范

1. 语言包中的语言项以大写字母及下划线连接 , 例如 FUNCTIONS\_CONTROL\_ERROR

## 61\_9 变量规范

1. 建议局部变量在最接近使用它时再声明
2. 变量命名建议使用 camel 命名规则如 \$userName , 项目开发中自行设置相应规范
3. 任何变量在进行累加、直接显示或存储前必需进行初使化 , 例如 \$hdphp='';
4. 判断变量是否存在要使用 isset()

5. 判断数组是否存在要使用 `is_array()`

## 61\_10 缩进规范

1. 代码的缩进要使用 tab 而不要使用 space( 空格 )
2. 每个缩进的单位约定是一个 TAB(8 个空白字符) , 需每个参与项目的开发人员在编辑器 (UltraEdit、EditPlus、Zend Studio 等) 中进行强制设定 , 以防在编写代码时遗忘而造成格式上的不规范

## 61\_11 注释规范

注释是对于那些容易忘记作用的代码添加简短的介绍性内容。使用 C 样式的注释 '`/* */`' 和标准 C++ 注释 '`//`'

程序开发中难免留下一些临时代码和调试代码 , 此类代码必须添加注释 , 以免日后遗忘。所有临时性、调试性、试验性的代码 , 必须添加统一的注释标记 , '`//debug`' 并后跟完整的注释信息 , 这样可以方便在程序发布和最终调试前批量检查程序中是否还存在有疑问的代码。例如 : `//debug` 模型字段别名替换调试代码

**注释详细说明如下 :**

1. 注释应当准确、易懂 , 防止有二义性。错误的注释不但无益反而有害。
2. 边写代码边注释 , 修改代码同时修改相应的注释 , 以保证注释与代码的一致性
3. 注释是对代码逻辑的描述 , 而不是文档或代码的翻译。程序中的注释不可喧宾夺主 , 注释太多了会让人眼花缭乱
4. 类的注释 :a. 类是谁写的 b. 类的功能有哪些 c. 类的名称 d. 类修改时间
5. 方法的注释 :a. 方法简要描述 b. 方法各个参数含义 c. 返回值类型与含义
6. 每一个代码段需要添加注释 , 说明代码段含义

## 61\_12 兼容性

代码设计应当兼顾 PHP 高低版本的特性 , 当前 , 应仍然以 `>=PHP 5.2` 作为最低通过平台 , 尽量不使用高版本 PHP 新增的函数、常数或者常量。如果使用只在高版本才具备的函数 , 必须对其进行二次封装 , 自动判断当前 PHP 版本 , 并自行编写低版本下的兼容代码 ( 如后盾框架中的 `json_encode` 处理方式 )

## 61\_13 代码重用

代码的有效重用可以减少效率的损失与资源的浪费。在开发软件项目时为了避免重复劳动和浪费时间。开发人员应尽量提高现有代码的重用率，同时将更多的精力用在新技术的应用和新功能的创新开发上面。

在需要多次使用代码，并且对于您希望实现的任务没有可用的内置 PHP 函数时，不吝啬定义函数或类，以 HDPHP 框架为例，定义文件应该放入 include 目录中

超过 3 行，实现相同功能的程序切勿在不同程序中多次出现，这是无法容忍和回避的问题；在任何时候都不要出现同一个程序中出现两段或更多的相似代码或相同代码，即便在不同程序中，也应尽力避免。

开发者应当总是有能力找到避免代码大段（超过 10 行）重复或类似的情况。开发者应切实在增强产品效率切实在增强产品效率、逻辑性和可读性上下功夫，这是一名优秀软件开发者所必须具备的基本素质。

## 61\_14 引号

PHP 中单引号和双引号具有不同的含义，最大的几项区别如下：单引号中，任何变量 (\$var)、特殊转义字符（如 '\t \r \n' 等）不会被解析，因此 PHP 的解析速度更快，转义字符仅仅支持 '' 和 '\\' 这样对单引号和反斜杠本身的转义；双引号中，变量 (\$var) 值会代入字符串中，特殊转义字符也会被解析成特定的单个字符，还有一些专门针对上述两项特性的特殊功能性转义，例如 '\\$' 和 '{\$array[ 'key' ]}'。这样虽然程序编写更加方便，但同时 PHP 的解析也很慢；数组中，如果下标不是整型，而是字符串类型，请务必用单引号将下标括起，正确的写法为 \$array[ 'key' ]，而不是 \$array[key]，因为不正确的写法会使 PHP 解析器认为 key 是一个常量，进而先判断常量是否存在，不存在时才以 'key' 作为下标带入表达式中，同时出发错误事件，产生一条 Notice 级错误。

因此，在绝大多数可以使用单引号的场合，禁止使用双引号。依据上述分析，可以或必须使用单引号的情况包括但不限于下述：

1. 字符串为固定值，不包含 '\t' 等特殊转义字符
2. 数组的固定下标，例如 \$array[ 'key' ]
3. 表达式中不需要带入变量，例如 \$string = 'test' ；，而非 \$string = 'test\$var' ；

## 61\_15 数据库规范

1. 数据表及表字段没有强制要求，建议使用全部小写字母，例如 user\_name
2. 任何类型的数据表，字段空间应当本着足够用，不浪费的原则
3. 存储多项内容的字段，或代表数量的字段，也应当以复数方式命名，例如：hits( 查看次数 )、items( 内容数量 )
4. 当几个表间的字段有关连时，要注意表与表之间关联字段命名的统一，如 hdcms\_article\_1 表中的 articleid 与 hdcms\_article\_data\_1 表中的 articleid
5. id 自增量的字段，一般情况下，使用全称的形式，例如 userid、articleid

### 61-15-1 字段结构

允许 NULL 值的字段，数据库在进行比较操作时，会先判断其是否为 NULL，非 NULL 时才进行值的必对。因此基于效率的考虑，所有字段均不能为空，即全部 NOT NULL；

预计不会存储非负数的字段，例如各项 id、发帖数等，必须设置为 UNSIGNED 类型。UNSIGNED 类型比非 UNSIGNED 类型所能存储的正整数范围大一倍，因此能获得更大的数值存储空间；

存储开关、选项数据的字段，通常使用 tinyint(1) 非 UNSIGNED 类型，少数情况也可能使用 enum() 结果集的方式。tinyint 作为开关字段时，通常 1 为打开；0 为关闭；-1 为特殊数据

MEMORY/HEAP 类型的表中，要尤其注意规划节约使用存储空间，这将节约更多内存。例如 hdcms\_sessions 表中，就将 IP 地址的存储拆分为 4 个 tinyint(3) UNSIGNED 类型的字段，而没有采用 char(15) 的方式

### 61-15-2 SQL 语句

所有 SQL 语句中，除了表名、字段名称以外，全部语句和函数均需大写，应当杜绝小写方式或大小写混杂的写法。例如 select \* from hdcms\_member; 是不符合规范的写法。

很长的 SQL 语句应当有适当的断行，依据 JOIN、FROM、ORDER BY 等关键字进行界定。

通常情况下，在对多表进行操作时，要根据不同表名称，对每个表指定一个 1~2 个字母的别名处理，以利于语句简洁和可读性

### 61-15-3 定长与变长表

包含任何 varchar、text 等变长字段的数据表，即为变长表，反之则为定长表。

对于变长表，由于记录大小不同，在其上进行许多删除和更改将会使表中的碎片更多。需要定期运行 OPTIMIZE TABLE 以保持性能。而定长表就没有这个问题；

如果表中有可变长的字段，将它们转换为定长字段能够改进性能，因为定长记录易于处理。但在试图这样做之前，应该考虑下列问题：

1. 使用定长列涉及某种折衷。它们更快，但占用的空间更多。`char(n)` 类型列的每个值总要占用  $n$  个字节（即使空串也是如此），因为在表中存储时，值的长度不够将在右边补空格；
2. 而 `varchar(n)` 类型的列所占空间较少，因为只给它们分配存储每个值所需要的空间，每个值再加一个字节用于记录其长度。因此，如果在 `char` 和 `varchar` 类型之间进行选择，需要对时间与空间作出折衷；
3. 变长表到定长表的转换，使用一个 ALTER TABLE 语句转换
4. 有时不能使用定长类型，即使想这样做也不行。例如对于比 255 字符更长的串，没有定长类型；
5. 在设计表结构时如果能够使用定长数据类型尽量用定长的，因为定长表的查询、检索、更新速度都很快。必要时可以把部分关键的、承担频繁访问的表拆分，例如定长数据一个表，非定长数据一个表。
6. 进行表结构设计时，应当做到恰到好处，反复推敲，从而实现最优的数据存储体系。

#### 61-15-4 运算与检索

数值运算一般比字符串运算更快。例如比较运算，可在单一运算中对数进行比较。而串运算涉及几个逐字节的比较，如果串更长的话，这种比较还要多。

如果串列的值数目有限，应该利用普通整型或 enum 类型来获得数值运算的优越性。

更小的字段类型永远比更大的字段类型处理要快得多。对于字符串，其处理时间与串长度直接相关。一般情况下，较小的表处理更快。对于定长表，应该选择最小的类型，只要能存储所需范围的值即可。例如，如果 `tinyint` 够用，就不要选择 `int`。对于可变长类型，也仍然能够节省空间。一个 `TEXT` 类型的值用 2 字节记录值的长度，而一个 `LONGTEXT` 则用 4 字节记录其值的长度。如果存储的值长度永远不会超过 64KB，使用 `TEXT` 将使每个值节省 2 字节。

#### 61-15-5 结构优化与索引优化

索引能加快查询速度，而索引优化和查询优化是相辅相成的，既可以依据查询对索引进

行优化，也可以依据现有索引对查询进行优化，这取决于修改查询或索引，哪个对现有产品架构和效率的影响最小。

首先，根据产品的实际运行和被访问情况，找出哪些 SQL 语句是最常被执行的。最常被执行和最常出现在程序中是完全不同的概念。最常被执行的 SQL 语句，又可被划分为对大表（数据条目多的）和对小表（数据条目少的）的操作。无论大表或小表，有可分为读（SELECT）多、写（UPDATE/INSERT）多或读写都多的操作。

对常被执行的 SQL 语句而言，对大表操作需要尤其注意：

写操作多的，通常可使用写入缓存的方法，先将需要写或需要更新的数据缓存至文件或其他表，定期对大表进行批量写操作。同时，应尽量使得常被读写的大表为定长类型，即便原本的结构中大表并非定长。大表定长化，可以通过改变数据存储结构和数据读取方式，将一个大表拆成一个读写多的定长表，和一个读多写少的变长表来实现；

读操作多的，需要依据 SQL 查询频率设置专门针对高频 SQL 语句的索引和联合索引。

而小表就相对简单，加入符合查询要求的特定索引，通常效果比较明显。同时，定长化小表也有益于效率和负载能力的提高。字段比较少的小定长表，甚至可以不需要索引。

其次，看 SQL 语句的条件和排序字段是否动态性很高（即根据不同功能开关或属性，SQL 查询条件和排序字段的变化很大的情况），动态性过高的 SQL 语句是无法通过索引进行优化的。惟一的办法只有将数据缓存起来，定期更新，适用于结果对实效性要求不高的场合。

MySQL 索引，常用的有 PRIMARY KEY、INDEX、UNIQUE 几种，详情请查阅 MySQL 文档。通常，在单表数据值不重复的情况下，PRIMARY KEY 和 UNIQUE 索引比 INDEX 更快

事实上，索引是将条件查询、排序的读操作资源消耗，分布到了写操作中，索引越多，耗费磁盘空间越大，写操作越慢。因此，索引决不能盲目添加。对字段索引与否，最根本的出发点，依次仍然是 SQL 语句执行的概率、表的大小和写操作的频繁程度。

## 61-15-6 查询优化

MySQL 中并没有提供针对查询条件的优化功能，因此需要开发者在程序中对查询条件的先后顺序人工进行优化。例如如下的 SQL 语句：

```
SELECT * FROM table WHERE a>' 0' AND b<' 1' ORDER BY c LIMIT 10;
```

事实上无论 a>' 0' 还是 b<' 1' 哪个条件在前，得到的结果都是一样的，但查询速度就大不相同，尤其在对大表进行操作时。

开发者需要牢记这个原则：最先出现的条件，一定是过滤和排除掉更多结果的条件；第二出现的次之；以此类推。因而，表中不同字段的值的分布，对查询速度有着很大影响。而 ORDER BY 中的条件，只与索引有关，与条件顺序无关。

除了条件顺序优化以外，针对固定或相对固定的 SQL 查询语句，还可以通过对索引结构进行优化，进而实现相当高的查询速度。原则是：在大多数情况下，根据 WHERE 条件的先后顺序和 ORDER BY 的排序字段的先后顺序而建立的联合索引，就是与这条 SQL 语句匹配的最优索引结构。尽管，事实的产品中不能只考虑一条 SQL 语句，也不能不考虑空间占用而建立太多的索引。

同样以上面的 SQL 语句为例，最优的当 table 表的记录达到百万甚至千万级后，可以明显的看到索引优化带来的速度提升。

依据上面条件优化和索引优化的两个原则，当 table 表的值为如下方案时，可以得出最优的条件顺序方案：

| 字段 a | 字段 b | 字段 c |
|------|------|------|
| 1    | 7    | 11   |
| 2    | 8    | 10   |
| 3    | 9    | 13   |
| -1   | 0    | 12   |

1. 最优条件： $b < '1' \text{ AND } a > '0'$
2. 最优索引：INDEX abc (b, a, c)x
3. 原因： $b < '1'$  作为第一条件可以先过滤掉 75% 的结果。如果以  $a > '0'$  作为第一条件，则只能先过滤掉 25% 的结果
4. 注意 1：字段 c 由于未出现于条件中，故条件顺序优化与其无关
5. 注意 2：最优索引由最优条件顺序得来，而非由例子中的 SQL 语句得来
6. 注意 3：索引并非修改数据存储的物理顺序，而是通过对应特定偏移量的物理数据而实现的虚拟指针

## 61\_16 版本控制

1. 不能提交有问题的代码。代码提交要及时，准确
2. 在代码提交后要告知小组开发成员，以便能及时获取最新代码。

3. 不要修改小组成员代码，更不要修改后提交小组成员代码，有修改联系书写人维护

## 61\_17 总结

1. 各设计规范应严格遵守，以保证项目顺利进行。以上规范只是总的开发规范，之后都应以此为例来规范设计，以便后续维护等。
2. HDPHP 开发基于 UTF-8 字符集编码，建议开发中程序文件使用 UTF-8 编码，且去掉 BOM 头信息（不去掉 BOM 头信息会有些意外问题如进行验证码输出时，图片不会正常显示）
3. 类的书写人可以新增，修改，删除该类的方法，其他团队成员在开发中出现的新的功能需求，需要在某个类中添加或修改，需联系该类的书写人维护。这样可以保证代码的准确性。不要随便修改他人代码
4. 以上命名规范是 HDPHP 框架产品所使用的规范，也是主流的命名规范，建议你遵循这些规范命名，但这不是强制的