Building web application

NJR 的简单介绍

Gaffey Eggzilla

May 10, 2011

混乱

Chaos / 混乱的代码/ 混乱的风格

更希望

- Calling

更希望

- Calling
  - 命名清晰简洁

更希望

- Calling
  - 命名清晰简洁
  - 接口参数标准

更希望

- Calling
  - 命名清晰简洁
  - 接口参数标准
- Coupling

更希望

- Calling
  - 命名清晰简洁
  - 接口参数标准
- Coupling
  - 逻辑封装

更希望

- Calling
  - 命名清晰简洁
  - 接口参数标准
- Coupling
  - 逻辑封装
  - 调用有效

更希望

- Calling
  - 命名清晰简洁
  - 接口参数标准
- Coupling
  - 逻辑封装
  - 调用有效
- Debuging

更希望

- Calling
  - 命名清晰简洁
  - 接口参数标准
- Coupling
  - 逻辑封装
  - 调用有效
- Debuging
  - I/O(数据库、缓存...)

更希望

- Calling
  - 命名清晰简洁
  - 接口参数标准
- Coupling
  - 逻辑封装
  - 调用有效
- Debuging
  - I/O(数据库、缓存...)
  - 问题定位 (嗯, 大家懂得:)

更希望

- Calling
  - 命名清晰简洁
  - 接口参数标准
- Coupling
  - 逻辑封装
  - 调用有效
- Debuging
  - I/O(数据库、缓存...)
  - 问题定位 (嗯, 大家懂得:)
- Log? Log!

更希望

- Calling
  - 命名清晰简洁
  - 接口参数标准
- Coupling
  - 逻辑封装
  - 调用有效
- Debuging
  - I/O(数据库、缓存...)
  - 问题定位 (嗯, 大家懂得:)
- Log? Log!
  - BI 和 OPS 的各种要求

更希望

- Calling
  - 命名清晰简洁
  - 接口参数标准
- Coupling
  - 逻辑封装
  - 调用有效
- Debuging
  - I/O(数据库、缓存...)
  - 问题定位 (嗯, 大家懂得:)
- Log? Log!
  - BI 和 OPS 的各种要求
  - 客服 MM 的各种问题

更希望

- Calling
  - 命名清晰简洁
  - 接口参数标准
- Coupling
  - 逻辑封装
  - 调用有效
- Debuging
  - I/O(数据库、缓存...)
  - 问题定位 (嗯, 大家懂得:)
- Log? Log!
  - BI 和 OPS 的各种要求
  - 客服 MM 的各种问题
  - echo,var_dump 控

更希望

- Calling
  - 命名清晰简洁
  - 接口参数标准
- Coupling
  - 逻辑封装
  - 调用有效
- Debuging
  - I/O(数据库、缓存...)
  - 问题定位 (嗯, 大家懂得:)
- Log? Log!
  - BI 和 OPS 的各种要求
  - 客服 MM 的各种问题
  - echo,var_dump 控
  - warn,info,error 们

NJR 来了!

- **NJR(NanJi Repository)** 是构建 web 应用程序的基础类库。
- 它封装了操作系统和外部独立组件 (如数据库, 缓存等) 的 API,支撑业务应用层完成数据交换和传输。

NJR 来了!

- **NJR(NanJi Repository)** 是构建 web 应用程序的基础类库。
- 它封装了操作系统和外部独立组件 (如数据库, 缓存等) 的 API,支撑业务应用层完成数据交换和传输。

NJ 是南方基地的简称, 被选择做当前版本的项目代号.

Import class

类库自动载入

```
defined('NJR') or
define('NJR','/usr/share/NJR/');
require(NJR.'NJ.php');
//直接实例化相应的类
$class = new NJ_Class();
```

Configure with Array

数组式的初始化配置

```
$class = new NJ_Class(array $conf);
```

MySQL API

- exec()

MySQL API

- exec()
- query()

MySQL API

- exec()
- query()
- setAlive()

MySQL API

- exec()
- query()
- setAlive()

MySQL API

- exec()
- query()
- setAlive()

example

```
$config = array('host' => '127.0.0.1','db' =>
'mysql',
'port' => '3306',
'username' => 'user',
'password' => 'pass');

$db = new NJ_DBC($config,$debug = true, $profile
= true);
$db->exec();
$db->setAlive(false);
```

Cache API

- Origin API

Cache API

- Origin API
  - get()

Cache API

- Origin API
  - get()
  - set()

Cache API

- Origin API
  - get()
  - set()
  - del()

Cache API

- Origin API
  - get()
  - set()
  - del()
  - add()

Cache API

- Origin API
  - get()
  - set()
  - del()
  - add()
  - flush()

Cache API

- Origin API
  - get()
  - set()
  - del()
  - add()
  - flush()
- Array-like Retrieve

Cache API

- Origin API
  - get()
  - set()
  - del()
  - add()
  - flush()
- Array-like Retrieve
- Serialized Store

Memcache

- 采用 libmemcached 作为连接驱动

Memcache

- 采用 libmemcached 作为连接驱动
  - 同步/异步传输

Memcache

- 采用 libmemcached 作为连接驱动
  - 同步/异步传输
  - 一致性 HASH 算法

Memcache

- 采用 libmemcached 作为连接驱动
  - 同步/异步传输
  - 一致性 HASH 算法
  - memcached 服务管理

Memcache

- 采用 libmemcached 作为连接驱动
  - 同步/异步传输
  - 一致性 HASH 算法
  - memcached 服务管理
  - key 匹配

Memcache

- 采用 libmemcached 作为连接驱动
  - 同步/异步传输
  - 一致性 HASH 算法
  - memcached 服务管理
  - key 匹配
- inc/des (++/–)

Memcache

- 采用 libmemcached 作为连接驱动
  - 同步/异步传输
  - 一致性 HASH 算法
  - memcached 服务管理
  - key 匹配
- inc/des (++/–)
- append/prepend

Memcache

- 采用 libmemcached 作为连接驱动
  - 同步/异步传输
  - 一致性 HASH 算法
  - memcached 服务管理
  - key 匹配
- inc/des (++/–)
- append/prepend

Memcache

- 采用 libmemcached 作为连接驱动
  - 同步/异步传输
  - 一致性 HASH 算法
  - memcached 服务管理
  - key 匹配
- inc/des (++/–)
- append/prepend

example

```
$config = array('servers' => array(0 => array('host' => '127.0.0.1',
'port' => '11211',
'weight'= 1),),
'compress' => false,'serilize' => false);
$mc = NJ_Memcache::Instance($config);
$mc['test1'] = 'foo';
$mc['test1']++;
$mc->append('test1','push string into end');
```

APC Cache

- PHP VirtueMachine Cache(lifecycle wihin httpd process)

APC Cache

- PHP VirtueMachine Cache(lifecycle wihin httpd process)
- Apc_Compiler_Cache(**Opcode** lifecycle with apc.ttl)

APC Cache

- PHP VirtueMachine Cache(lifecycle wihin httpd process)
- Apc_Compiler_Cache(**Opcode** lifecycle with apc.ttl)
- Apc_User_Cache

APC Cache

- PHP VirtueMachine Cache(lifecycle wihin httpd process)
- Apc_Compiler_Cache(**Opcode** lifecycle with apc.ttl)
- Apc_User_Cache

APC Cache

- PHP VirtueMachine Cache(lifecycle wihin httpd process)
- Apc_Compiler_Cache(**Opcode** lifecycle with apc.ttl)
- Apc_User_Cache

hint

./configure –enable-apc –enable-mmap **–enable-apc-spinlocks** –disable-apc-pthreadmutex

APC Cache

- PHP VirtueMachine Cache(lifecycle wihin httpd process)
- Apc_Compiler_Cache(**Opcode** lifecycle with apc.ttl)
- Apc_User_Cache

hint

./configure –enable-apc –enable-mmap **–enable-apc-spinlocks** –disable-apc-pthreadmutex
apc.stat = 0 better performance in **require()**,but restart httpd when code updated

apc.slam_defense=0
apc.write_lock=1

File Cache

example

```
$config = array('fs_path' => '/var/tmp',
'fs_level' => 1,
'gc_chance' => 10000, 'serilize' => false);

$f = new NJ_Fscache($config);
$f->add('test1','asdasdsa');
$f->append('test1','sssssssssss');
```

Share Cache

- cache in local memory ('memcache' :p)

Share Cache

- cache in local memory ('memcache' :p)
- shared between **processes**

Share Cache

- cache in local memory ('memcache' :p)
- shared between **processes**
  - 比文件系统更好

Share Cache

- cache in local memory ('memcache' :p)
- shared between **processes**
  - 比文件系统更好
  - 默认比 APC 可以存储更大的数据

Share Cache

- cache in local memory ('memcache' :p)
- shared between **processes**
  - 比文件系统更好
  - 默认比 APC 可以存储更大的数据
  - 没有网络开销

Share Cache

- cache in local memory ('memcache' :p)
- shared between **processes**
  - 比文件系统更好
  - 默认比 APC 可以存储更大的数据
  - 没有网络开销

Share Cache

- cache in local memory ('memcache' :p)
- shared between **processes**
  - 比文件系统更好
  - 默认比 APC 可以存储更大的数据
  - 没有网络开销

example

```
$sc = new NJ_Shcache();
$sc['test1'] = 'foo';
$sc->close();
```

Remote Request

- 单接口访问 send()

Remote Request

- 单接口访问 send()
- 非阻塞请求 asend()

Remote Request

- 单接口访问 send()
- 非阻塞请求 asend()

Remote Request

- 单接口访问 send()
- 非阻塞请求 asend()

example

```
$stack = array(array('url' => 'http://www.baidu.com','params' => array('q'=>'test','lang'=>'zh-cn'),
'timeout'=>10),
array('url' => 'http://www.google.com','params' => '?q=test','timeout'=>10), );

NJ_Request::asend($stack);
```

Log

还没写

更多功能

- 终端程序支持 (守护进程..)

更多功能

- 终端程序支持 (守护进程..)
- 常用数据结构

更多功能

- 终端程序支持 (守护进程..)
- 常用数据结构
  - Queue

更多功能

- 终端程序支持 (守护进程..)
- 常用数据结构
  - Queue
  - Stack

更多功能

- 终端程序支持 (守护进程..)
- 常用数据结构
  - Queue
  - Stack
  - Collection

更多功能

- 终端程序支持 (守护进程..)
- 常用数据结构
  - Queue
  - Stack
  - Collection
- 安全相关

谢谢参与

- NJR 需要配套的业务层代码

谢谢参与

- NJR 需要配套的业务层代码
- NJR 期待各位的贡献

谢谢参与

- NJR 需要配套的业务层代码
- NJR 期待各位的贡献
- LaTeX 是一个功能强大的排版工具