Ice Framework

Ice框架介绍

----应用开发角度

目录

- 为什么用ICE?
- Ice整体架构
- Ice功能Feature
- Ice应用开发

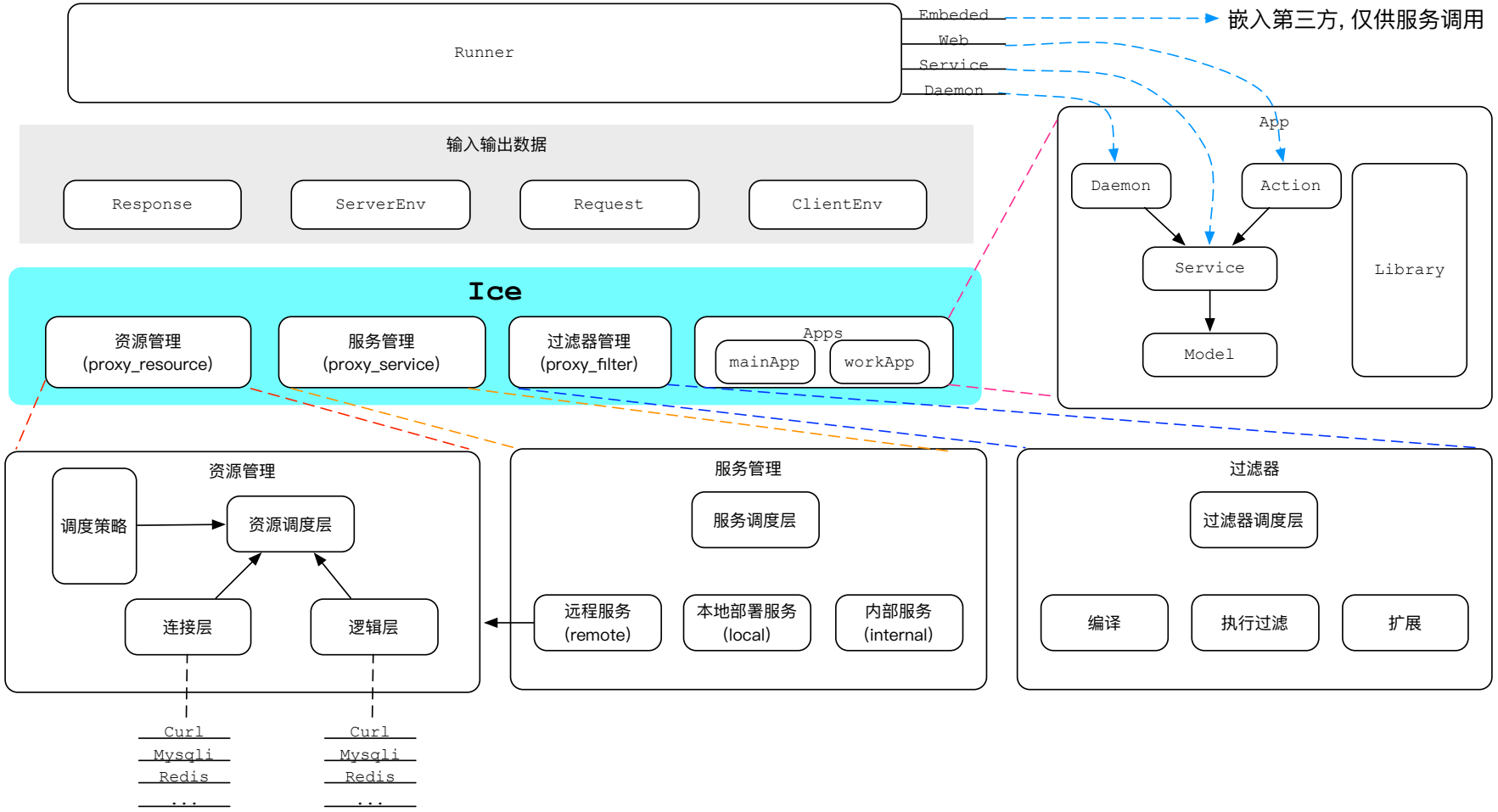
*为什么用ICE?

为什么用ICE?

- 规范
  - 基于composer构建
  - 符合PSR4
  - 明确的命名和开发规范，并有示例引导
- 可用性
  - 服务间逻辑隔离
  - 所有资源统一调度，可用性控制易扩展
  - 所有资源面向多集群设计
- 便利性
  - 快速开发：ice-skel
  - 丰富的工具集：数据库操作，数组等基础数据结构通用操作
  - 针对客户端API开发中差异性管理的Feature机制
  - 多种运行方式，满足不同场景需求

*设计层面

Ice整体架构


Runner
Embeded
Web
Service
Daemon
嵌入第三方, 仅供服务调用
输入输出数据
Response
ServerEnv
Request
ClientEnv
App
Daemon
Action
Service
Model
Library
Ice
资源管理
(proxy_resource)
服务管理
(proxy_service)
过滤器管理
(proxy_filter)
Apps
mainApp
workApp
资源管理
调度策略
资源调度层
连接层
逻辑层
Curl
Mysqli
Redis
...
Curl
Mysqli
Redis
...
服务管理
服务调度层
远程服务
(remote)
本地部署服务
(local)
内部服务
(internal)
过滤器
过滤器调度层
编译
执行过滤
扩展

Ice整体架构： 框架目录结构

```
框架文件结构
composer.json
class_alias.php              # 类名映射. 在composer.json中已定义自动加载
tpl/                         # 用户生成演示应用的模板文件结构
bin/
    ice-skel                 # 用于生成ice的演示应用的脚本工具. (执行前根据需求修改tpl/build.conf)
src/
    DB/                      # DB访问基础封装. 结构化的方法调用来做SQL查询. Model层可继承DB_Query
    Filter/                  # 过滤器. 按照过滤器指定语法, 描述数据规则, 进行校验/数据修正
    Message/                 # 针对跨地域IDC的分布式通信方案(未生产环境验证)
    Frame/
        Abs/                 # 框架的一些基础抽象类
        Error/               # 框架错误处理
        Runner/              # 启动器. 共三类: web(建议做面向用户的应用入口), service(建议做内部服务), daemon
        Daemon/              # daemon启动器内部处理涉及的相关代码
        Service/             # service启动器内部处理涉及的相关代码. 服务管理proxy.
        Web/                 # web启动器内部处理涉及相关代码.
        Embeded/             # embeded启动器内部处理涉及相关代码
        Feature.php          # Feature组件
        Config.php           # 配置加载工具
        Ice.php              # 框架主对象
        App.php              # 应用对象.
        Logger.php           # 日志工具
    Resource/                # 资源管理的封装
        Proxy.php            # 资源管理的入口proxy
        Connector/           # 资源的连接封装
        Handler/             # 资源的操作封装
        Strategy/            # 资源调度策略
    Util/                    # 基础数据结构, 通用工具的封装
```

Ice整体架构： 应用目录结构

```
composer.json
class_alias.php              # 如有需要可自己在composer.json注册，默认不使用
var/                         # 默认的运行时输出目录
    logs/                    # 默认日志路径
    run/                     # 运行时目录
        filter/              # 默认的FILTER编译产出路径
src/
    action/                  # web入口层
    daemon/                  # daemon入口层
    service/                 # 服务层
    model/                   # Model层
    lib/                     # 应用的本地类库
    embeded_demo.php         # 嵌入式Runner的示例
    conf/
        app.php              # 应用主配置
        service.inc          # service runner的个性化配置
        daemon.inc           # daemon runner的个性化配置
        web.inc              # web runner的个性化配置
        service.php          # 依赖的服务配置文件
        resource.php         # 依赖的资源配置文件
        message.php          # 跨地域IDC的分布式通信消息配置(未生产环境验证)
```

Ice功能Feature： 3+1种Runner

- Web： 面向终端用户的应用层构建．有模板引擎层，有自由的路由控制
  - 客户端API
  - OPENAPI
  - H5
  - PC-Web
- Service： 提供内部服务，固定的交互协议(输入输出)
  - 纵向业务角度划分的服务模块
- Daemon： 后台运行的服务
  - 单进程模式
  - master-worker模式(暂未支持)
- Embeded： 将基于Ice开发的Service嵌入任何其他PHP代码
  - 重构旧系统服务

Ice功能Feature： 资源管理

核心思想

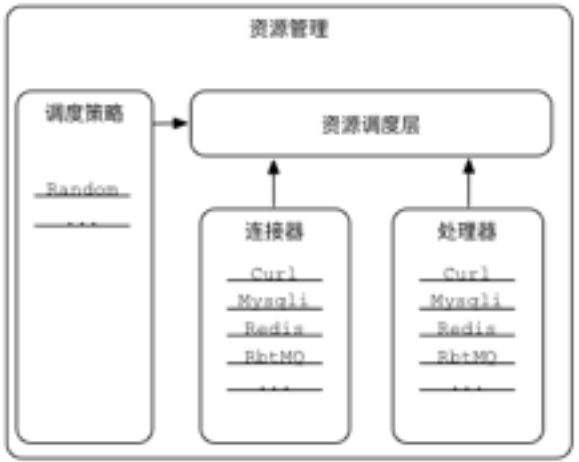
- 资源LIB = 连接 + 操作
- 资源可用性评估多在连接层面完成

资源的抽象：

```
ALGO        := "random"    # 目前仅实现了随机分配算法
SCHEME      := 资源类型
UNITNAME    := 业务层面集群划分，比如：用户集群，账号集群
CLUSTER     := 技术层面集群划分，比如：MySQL或Redis的主从集群

PARAMETER   := "force_new=" [1 | 0] | "algo=" ALGO
PARAMETERS  := PARAMETER | PARAMETER "&" PARAMETERS

RESOURCE    := SCHEME "://" UNITNAME [ "/" CLUSTER ] [ "?" PARAMETERS ]
```



应用

```
$proxyMysql = $manager->get('mysql://user/
master');
$proxyMysql->query('SHOW DATABASES');

$proxyRedis = $manager->get('redis://user/
master');
$proxyRedis->get('uinfo:5012470');

$proxyMQ = $manager->get('rabbitmq://user');
$proxyMQ->produce('action_exchange', 'new:
5012470');
```

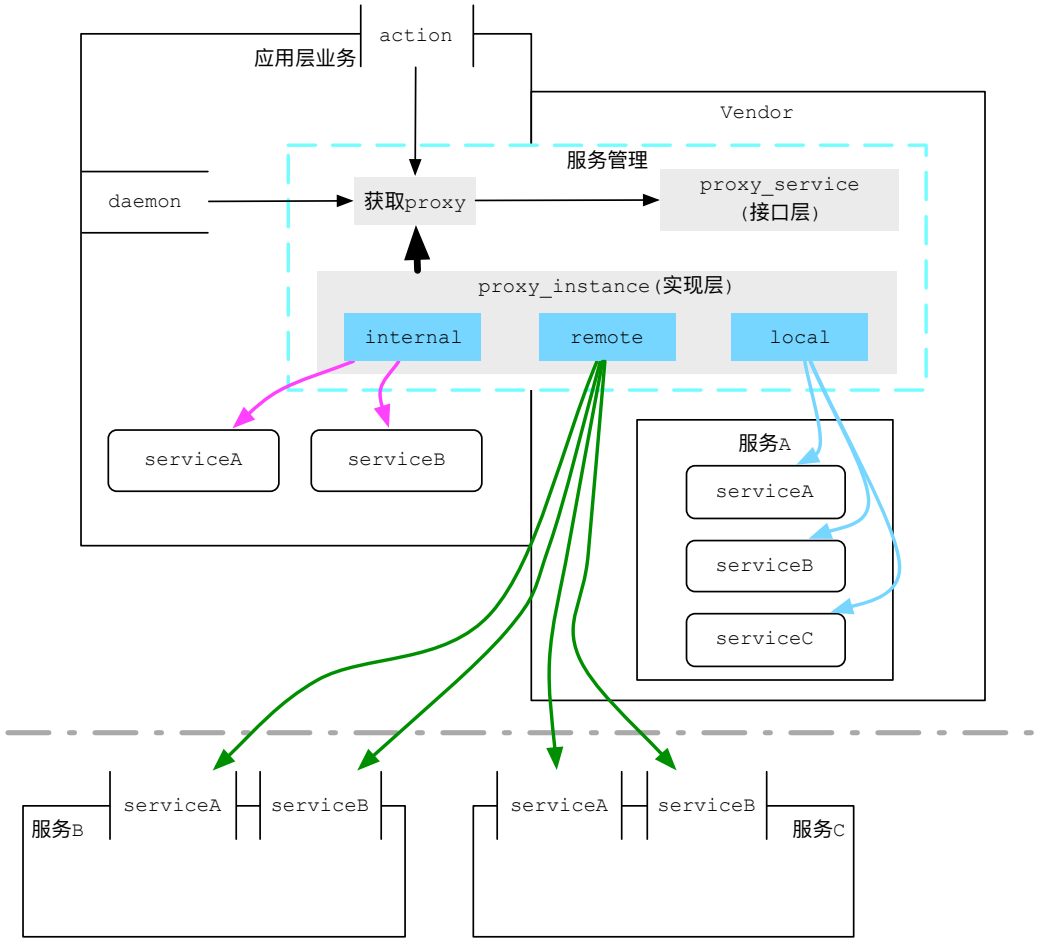
优点：

1. 应用简单且规范统一
2. 资源获取统一管理，容灾调度策略易扩展

Ice功能Feature： 服务管理层设计

核心思想

- 关注逻辑解耦
- 开发不受部署/调用行为影响
- 完善的日志跟踪消息流



Ice功能Feature： Filter要解决的问题

输入数据校验

```
/user/register
$uname = htmlspecialchars($_POST['uname']);
$password = htmlspecialchars($_POST['password']);
if (!preg_match(';[\w-]{6,12};', $uname)) {
    // ...
}

/user/modifyName
$uid = intval($_POST['uid']);
$uname = htmlspecialchars($_POST['uname']);
$password = htmlspecialchars($_POST['password']);
if (!preg_match(';[\w-]{6,12};', $uname) || $uid < 0) {
    // ...
}

/user/findPassword
$uname = …
```

输出数据修正

```
/user/info
$return array(
    'uid'          => get_encrypt_id((int)$uid),
    'uname'        => (string)$uname,
    'avatar'       => get_full_url((string)$avatar),
    'photo_count'  => (int)$photo_count,
    'settings'     => array(...),
)

/social/followList
$return array(
    'uid'     => get_encrypt_id((int)$uid),
    'uname'   => (string)$uname,
    'avatar'  => get_full_url((string)$avatar),
)

/social/fansList
…
```

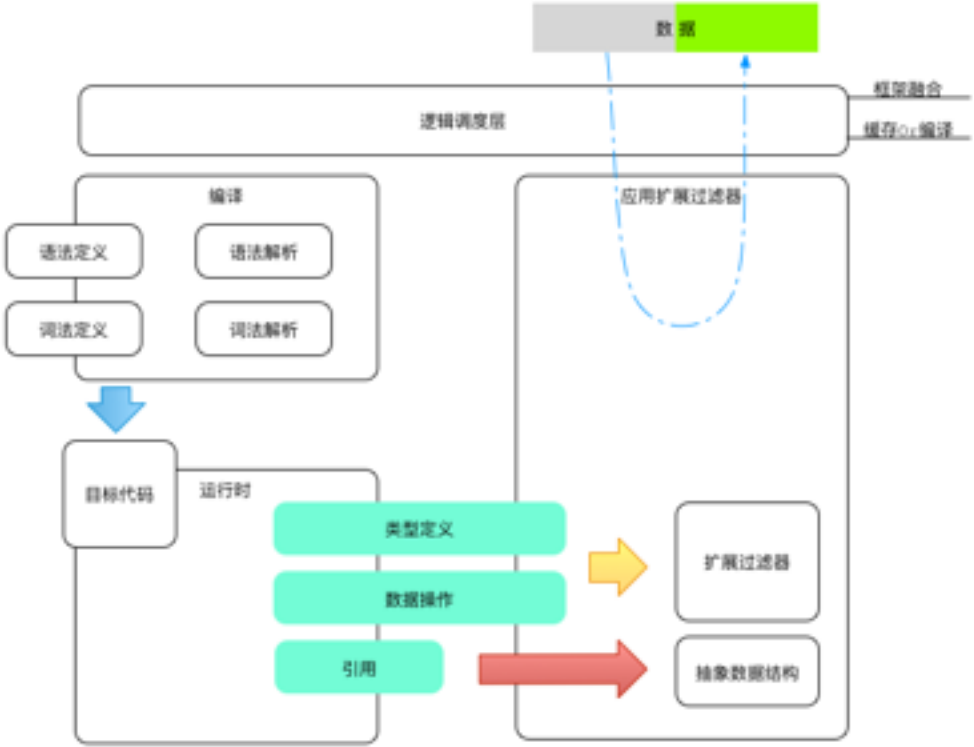
Ice功能Feature: Filter设计

核心思想

- 数据的基础校验，是通用的.
- 业务数据结构，应该被明确定义.
- 系统的边界数据的通用规则，应该被专门一层处理.

设计思路

- 用一套描述性语法规则，描述对数据的处理规则



Ice功能Feature: Filter的语法定义

```
LITERAL_ID      := REGEXP( [-_a-zA-Z0-9]+ )
LITERAL_STRING  := "\"" REGEXP( .* ) "\"" | "'" REGEXP( .* ) "'"
LITERAL_NUMERIC := [ "-" ] REGEXP( [0-9]+ ) [ "." REGEXP( [0-9]+ ) ]

OP_NAME         := LITERAL_ID
FIELD_NAME      := LITERAL_ID
TYPE_NAME       := LITERAL_ID
OP_ARG          := LITERAL_STRING | LITERAL_ID | LITERAL_NUMERIC
REQ_OR_DEFAULT := "__opt" | "__req" | LITERAL_STRING | LITERAL_ID | LITERAL_NUMERIC

TYPE         := "(" TYPE_NAME [ ":" REQ_OR_DEFAULT ] ")"
OP_ARG_LIST := OP_ARG | OP_ARG "," OP_ARG_LIST

FIELD_RULE_NONAME := [ TYPE ] FILTER_LIST
FIELD_RULE        := FIELD_NAME FIELD_RULE_NONAME
FIELD_RULE_LIST   := FIELD_RULE [ ";" ] | FIELD_RULE ";" FIELD_RULE_LIST

BLOCK_FILTER  := "{" FIELD_RULE_LIST "}"
EXTEND_FILTER := "@" LITERAL_STRING
OP_FILTER     := OP_NAME [ ":" OP_ARG_LIST ]

FILTER_LIST := BLOCK_FILTER | EXTEND_FILTER | OP_FILTER | BLOCK_FILTER FILTER_LIST | ( EXTEND_FIL
TER | OP_FILTER ) [ "|" ] FILTER_LIST

ROOT := FIELD_RULE_NONAME
```

Ice功能Feature: Filter应用

```php
$data = array(
    'code' => 0,
    'data' => array(
        'uid'     => 5012470,
        'uname'   => 'goosman-lei',
        'service' => array('code' => 1, 'data' => 'Hello Jack'),
        'user'    => array('id' => '5012470', 'name' => 'goosman-lei', 'location' => '北京'),
    ),
);
$filteredData = $this->ice->mainApp->proxy_filter->get('(map){
    code(int);
    data(map){
        uid(int);
        uname(str);
        service(map){
            code(int);
            data(str);
        };
        user(map){
            id(int);
            name(str);
            location(str);
        }
    }
}')->filter($data);
```

Ice功能Feature： Filter对目标问题的解决

输入数据校验

```
/user/register
$filter = '(map){
  uname(str) xss|match:";[\w-]{6,12};";
  password(str) xss
}'

/user/modifyName
$filter = '(map){
  uid(int) range:1_;
  uname(str) xss|match:";[\w-]{6,12};";
  password(str) xss;
}'

/user/findPassword
$uname = …
```

输出数据修正

```
$user_sample = '(map){
  uid(int) encrytp_id;
  uname(str);
  avatar(str) full_url
}'
$user_detail = '@"user_sample" | {
  photo_count(int)
  settings(map)
}"

/user/info
$proxy->get('@"user_detail"')->filter($data);

/social/followList
$proxy->get('@"user_sample")->filter($data);

/social/fansList
…
```

Ice功能Feature： Feature要解决的问题

复杂的客户端环境

iOS 9; nice 3.2; 2G
iOS 7; nice 2.8; 4G
iOS 8.2; nice 3.9; wifi
Android 4.2; nice 2.6; 3G; 360
Android 4.5; nice 3.6; 4G; Baidu
Android 4.5; nice 3.6; 4G; Offical
.....

变态的产品需求

iOS 7以上, 4G或wifi环境出高清图
nice 3.5以上支持私聊表情
nice 3.9.1灰度渠道用户插入广告
广东地区用户使用金山云CDN
北京地区用户插入一条广告
Android官方渠道实验新splash
.....

Feed API

夹缝中生存的Coder们

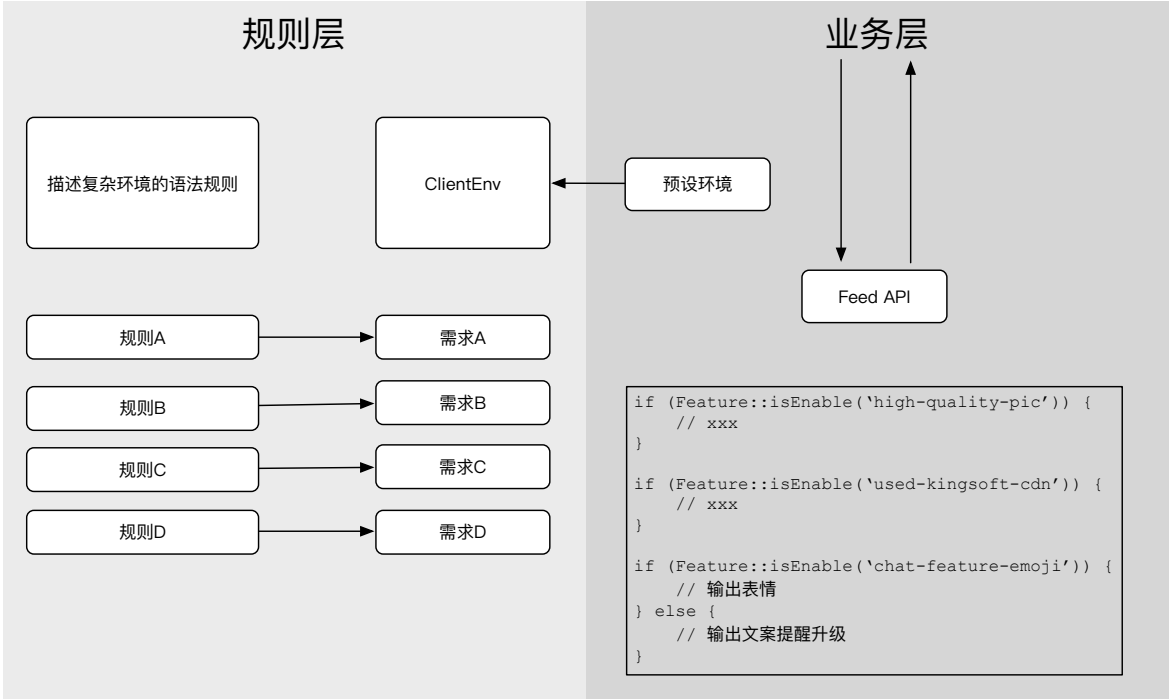
```
$osName     = $_GET['osn'];
$osVersion  = $_GET['osv'];
$channel    = $_GET['ch'];
$appVersion = $_GET['appv'];

if ($osName == 'ios' && $osVersion == '3.9.2') {
    // xxx
}
if ($osVersion == 'android' && $osVersion == '4.2' && $appVersion <= '3.2.1') {
    // xxx
}
if (....
```

Ice功能Feature： Feature的设计思路

核心思想

- 将复杂的环境，描述为简单的概念.
- 应用层关心需求的概念
- 规则层关注需求的前提条件.
- 参考：C语言环境相关的宏.



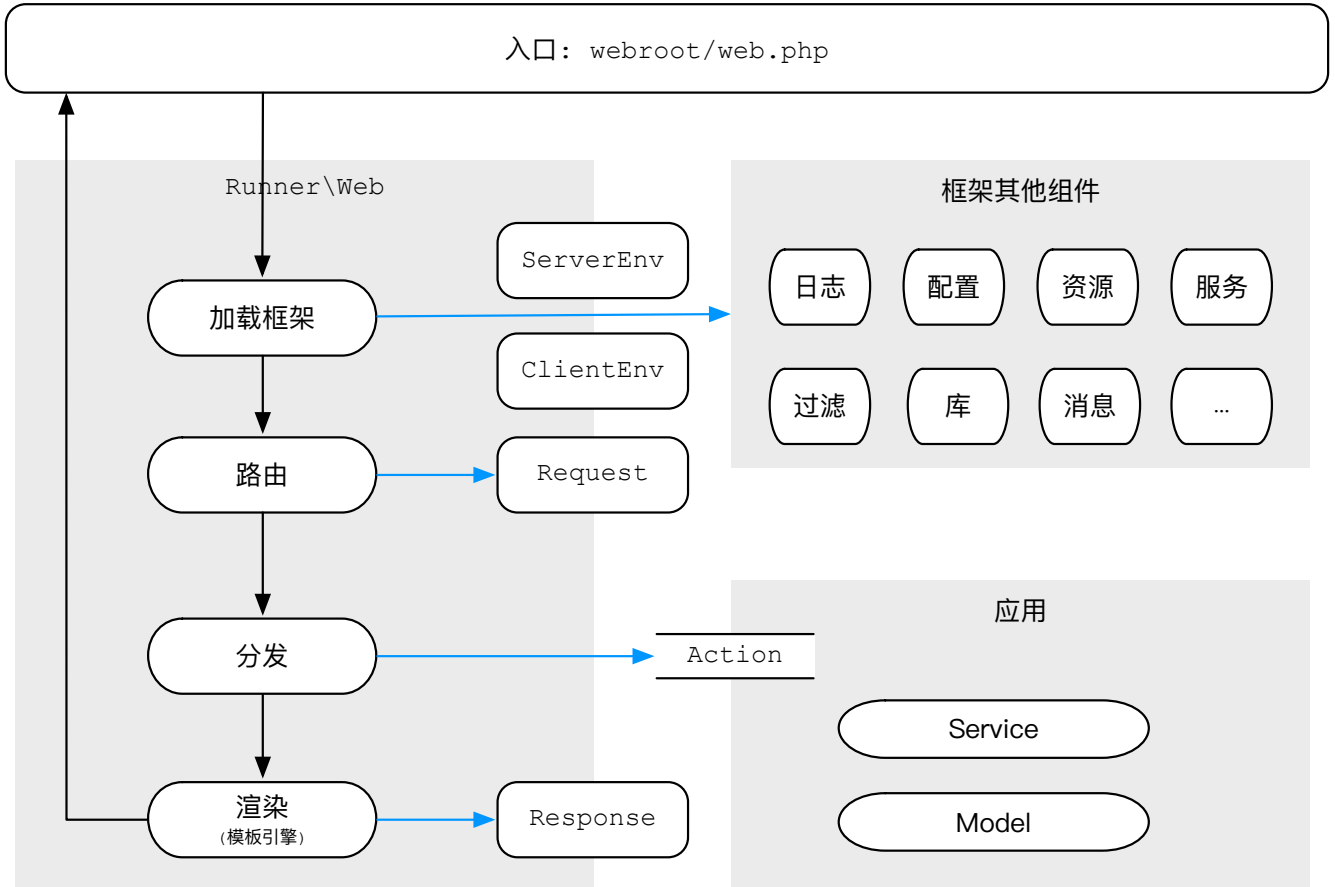
*应用层面

Ice应用开发： 构建应用项目

- 修改tpl/build.conf

- 执行bin/ice-skel
- output/demo下即为生成的应用骨架代码
- 将output/demo拷贝出来， 建立新的git

Ice应用开发： Web应用结构



Ice应用开发：Web应用开发

- 入口文件：src/webroot/web.php
- 编写逻辑：
  - src/action/Say/Helloworld.php：入口函数execute()
  - src/service/Say.php
  - src/model/User.php
- action中持有的句柄：
  - $this->request
  - $this->response
  - $this->clientEnv
  - $this->serverEnv
  - $this->feature
  - $this->ice
- action的HOOK：
  - $this->prevExecute()
  - $this->postExecute()

Ice应用开发： Web应用开发之路由

- 配置文件： src/webroot/web.inc

```
$web_routes = array(
   'default' => '\\Ice\\Frame\\Web\\Router\\RStatic', # 默认使用静态路由器
   'i= /say/helloworld' => array('class' => 'Say', 'action' => 'Helloworld'),
   '~= ;^/say/hi/(.+);i' => array(
      'class' => 'Say',
      'action' => 'Hi',
      'params' => array(
        1 => 'name'
      ),
   )
);
```

- 路由规则优先级

1	"==": 精确匹配	2	"i=": 不区分大小写精确匹配
3	"^=": 精确前缀匹配	4	"i^": 不区分大小写前缀匹配
5	"$=": 精确后缀匹配	6	"i$": 不区分大小写后缀匹配
7	"~=": 正则匹配	8	自定义路由：逗号分隔，直到一个路由器返回TRUE

Ice应用开发： Web应用开发之模板引擎

- 配置文件：src/webroot/web.inc

```
$web_temp_engine = array(
   'engines' => array(
      'json' => array(
         'adapter' => '\\Ice\\Frame\\Web\\TempEngine\\Json',
         'adapter_config' => array(...),
      ),
      ...
   ),
   'routes' => array(
      '*' => 'json',
      'say' => array(
         '*' => 'smarty',
         'hi' => 'json',
      ),
   ),
);
```

Ice应用开发： Web应用开发之输出数据交互

- 输出HTTP Header

```
$this->response->addHeader('User-Defined-Header: Value');
```

- 设置Cookie

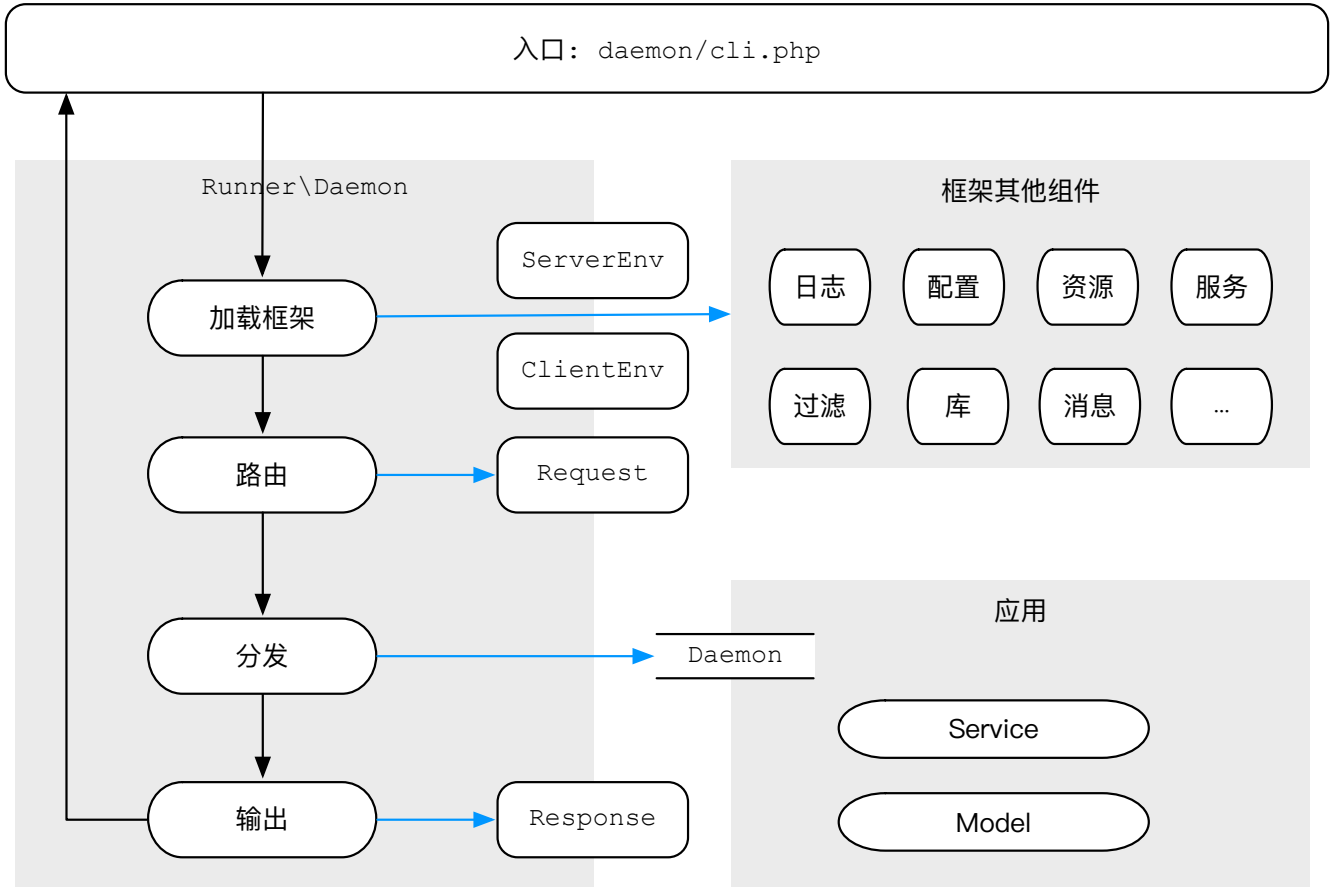
```
$this->response->addCookie('user-defined-cookie', 'value'); # 参数同setcookie()
```

- 直接添加body内容

```
$this->response->appendBody('Hello');
```

- 模板数据
  - Action的execute()返回数据， 会被设置到模板引擎中.
  - 另外可以通过$this->response->addTplData($datas)添加模板数据

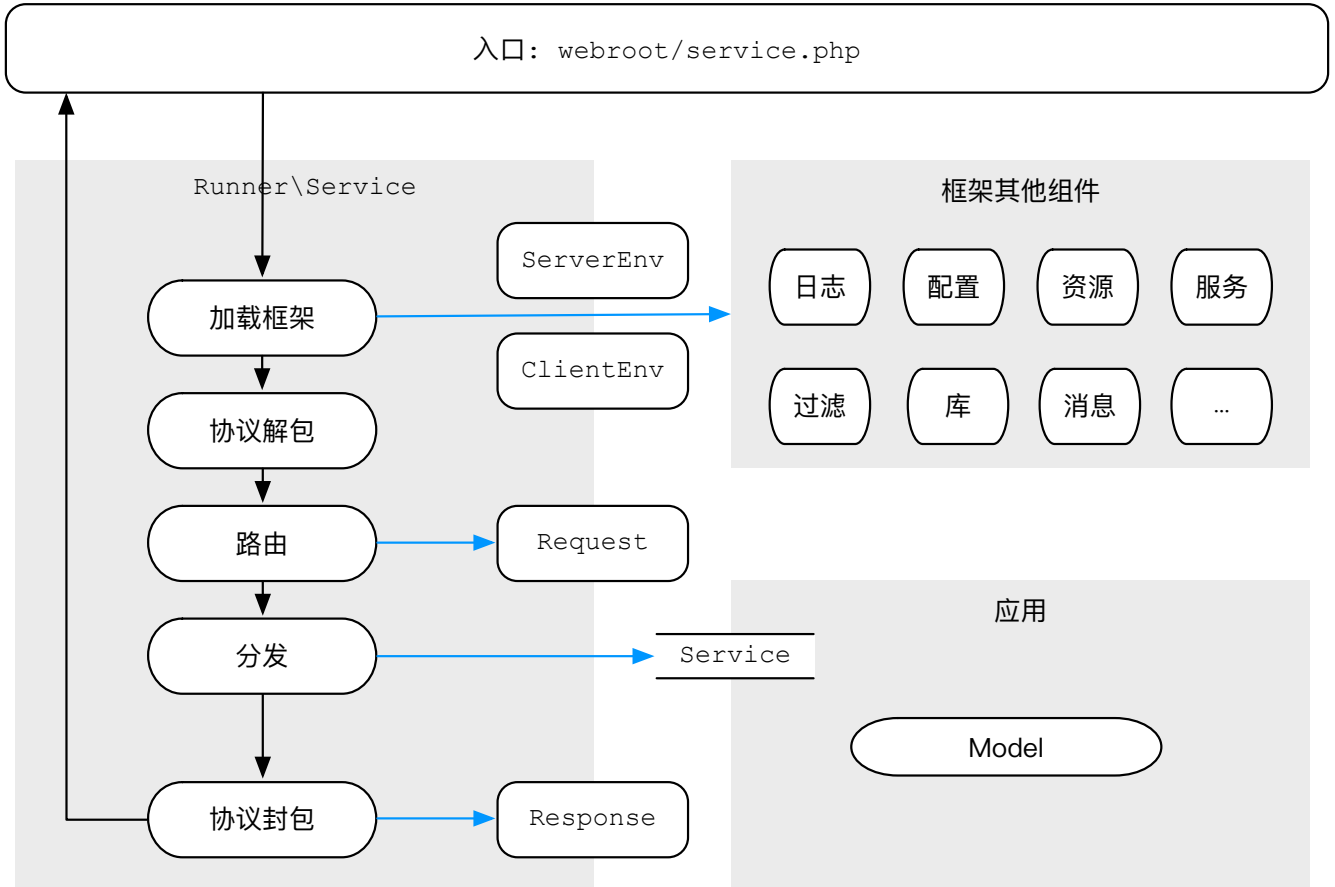
Ice应用开发： Daemon应用结构



Ice应用开发：Daemon应用开发

- 入口文件：src/daemon/cli.php
- 编写逻辑：
  - src/daemon/Demo/Command.php：入口函数execute()
  - src/service/Say.php
  - src/model/User.php
- daemon中持有的句柄：
  - $this->request
  - $this->response
  - $this->clientEnv
  - $this->serverEnv
  - $this->ice

Ice应用开发： Service应用结构



Ice应用开发：Service应用开发

- 入口文件：src/webroot/service.php
- 编写逻辑：
  - src/service/Say.php
  - src/model/User.php
- service中持有的句柄：
  - $this->ice
- service中的通用方法：
  - $this->succ($data = null)
  - $this->error($code, $data = null)
- service统一返回结构：

```
array(
  'code' => $code,
  'data' => $data,
)
```

Ice应用开发： 核心对象ICE

- 全局唯一静态Ice对象
  - \F_Ice::$ins
- Ice对象持有的句柄
  - \F_Ice::$ins->runner
  - \F_Ice::$ins->mainApp: 主应用
  - \F_Ice::$ins->workApp: 当前应用.(service local依赖时存在)
- Runner持有句柄
  - $runner->clientEnv
  - $runner->serverEnv
  - $runner->request
  - $runner->response
  - $runner->feature (仅web runner)

Ice应用开发： 核心对象App

- App持有句柄(mainApp/workApp)
  - $app->config

```
$confArr = $app->config->get('resource.pool.mysqli.user');
```

  - $app->proxy_resource

```
$proxy = $app->proxy_resource->get('mysqli://user/master');
```

  - $app->proxy_service

```
$proxy = $app->proxy_service->get('internal', 'Say')
```

  - $app->proxy_filter

```
$proxy = $app->proxy_filter->get('(map){
   code(int);
   data(map);
}');
```

  - $app->logger_xxx

```
$app->logger_comm->warn(array('uid' => 5012470), 30203);
```

- App便利方法(mainApp/workApp)
  - $app->getModel('user')

Ice应用开发： 服务调用

- 内部调用： 直接调用本项目内的服务

```
$proxy = $app->proxy_service->get('internal', 'User');
$proxy->getInfo('5012470');
```

- 外部服务本地调用： 逻辑解耦的服务本地部署调用

```
$proxy = $app->proxy_service->get('local-user', 'User');
$proxy->getInfo('5012470');
```

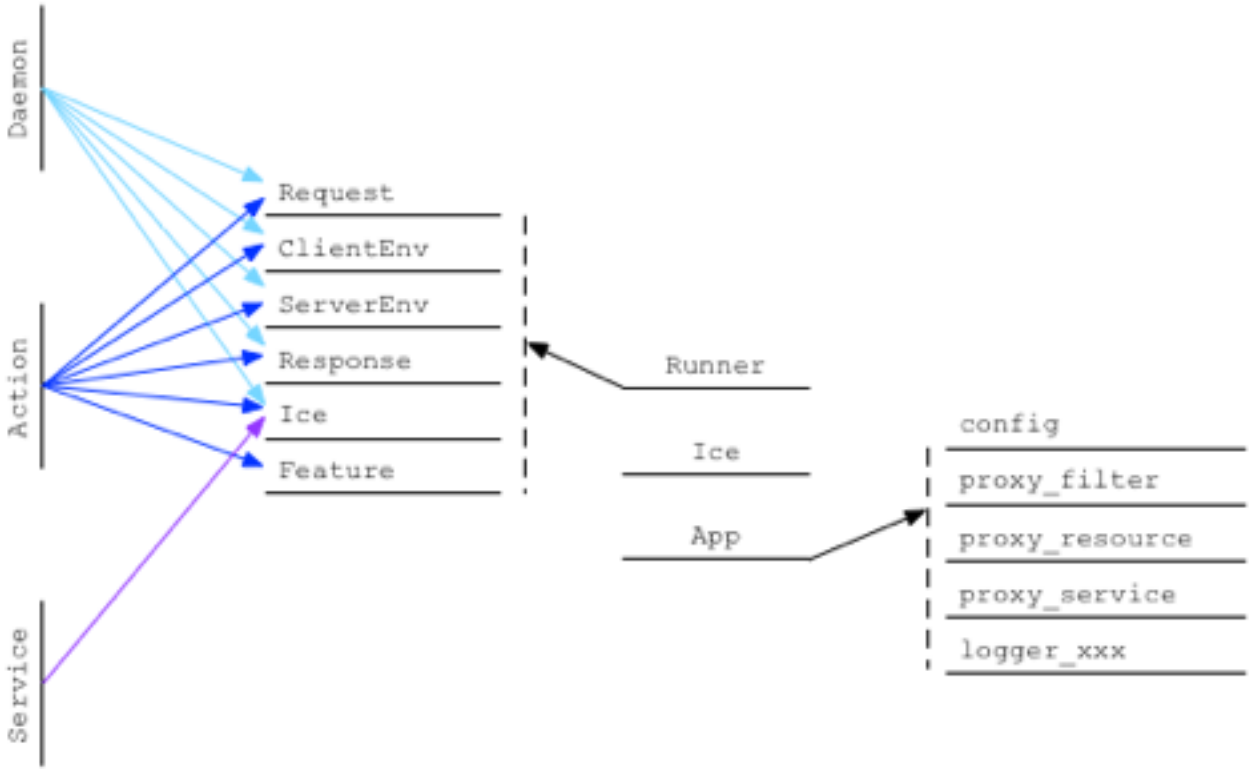
- 外部服务远程调用： 逻辑解耦的服务， 通过HTTP+JSON方式调用

```
$proxy = $app->proxy_service->get('remote-user', 'User');
$proxy->getInfo('5012470');
```

- 上述调用对应配置(src/conf/service.php)

```
$pool = array(
   'local-user' => array(
      'proxy' => 'local',
      'config' => array('project_group' => 'ice', 'project_name' => 'demo_user'),
   ),
   'remote-user' => array(
      'proxy' => 'remote',
      'config' => array('resource' => 'curl://service'),
   )
)
```

Ice应用开发： 核心类关联关系



Ice应用开发： 数据库访问模块

- Model类实现

```
namespace <project-group>\<project-name>\Model;
class User extends \DB_Query {
   protected $tableName = 'user';
   protected $dbResource = 'demo';
   protected $mapping = array(
      'id' => 'i',
      'name' => 's',
   );
}
```

- 资源关系映射

Model::$dbResource被解析为mysqli://<$dbResource>/[master | slave]

master or slave由SQL语句是否更新类确定

- 使用示例

```
$userModel = new \<project-group>\<project-name>\Model\User;
$user
proxy = $app->proxy_service->get('remote-user', 'User');
$proxy->getInfo('5012470');
```

Ice应用开发： 日志记录及配置读取

- 日志记录

```
$app->logger_comm->info(array(
   'action'      => 'get user info',
   'session_uid' => $sessionUid,
   'target_uid'  => $targetUid,
));
$app->logger_comm->fatal(array(
   'error'      => 'get info from user service failed',
   'target_uid' => $targetUid,
), 2028302);
```

- 配置读取

```
$config = $app->config->get('resource.mysqli.user.master');
```

Ice应用开发： 错误处理

- 不建议应用层使用trigger_error自定义错误
- 如果应用层需要使用异常， 请在应用层内部捕获并处理
- 框架层最外部对异常和error有拦截处理， 仅记录日志， 对会中断脚本执行的异常/错误， 会执行异常流

```
\F_Ice::$ins->runner->response->error(\F_Ecode::PHP_ERROR);
```

异常流的具体错误处理， 依赖各类runner的response实现有差异：

1.web runner下， 可为每种模板引擎配置error_tpl来统一处理错误.
2.service runner下， 则直接按照service标准返回结构返回错误信息.
3.daemon runner下， 则将错误信息， 输出到标准错误文件句柄上.

Ice应用开发： 类加载及规范

- 所有项目命名为： <project-group>/<project-name>
- 所有类命名遵循PSR4规范.(名字空间分隔的目录分布)
- 业务层代码命名规范
  - action: \<project-group>\<project-name>\Action\Class\Action
  - service: \<project-group>\<project-name>\Service\Class::method
  - daemon: \<project-group>\<project-name>\Daemon\Class\Action
  - model: \<project-group>\<project-name>\Model\Class
  - lib: \<project-group>\<project-name>\Lib\Namespace\Class
- mainApp指代入口应用， workApp指代当前应用
  - 记录日志： 除非你明确为什么要使用workApp的配置记录日志， 否则使用mainApp->logger_xxx
  - 读取配置： 除非你明确为什么要使用mainApp的配置信息， 否则使用workApp->config->get('a.b')
  - 使用3类proxy组件： 全部使用workApp->proxy_xxx(filter, resource, service)
- 不允许跨层调用， 允许的调用关系
  - action => service
  - daemon => service
  - service => model
  - [action | daemon | service | model] => lib
  - [action | daemon | service | model | lib] => 框架

BB了这么多，看几个DEMO顺带Q&A吧

Thanks