Advances in Intelligent Systems Research, volume 147
International Conference on Network, Communication, Computer Engineering (NCCE 2018)

Design and Implementation of HTTP Interface Automation Test Framework PHPUnit

Tao Zhang [a)] and Lei He [b)]

Department of Informatics, Beijing University of Technology, Beijing, 100124, China.

[a)] Corresponding author: zhangtony8888@qq.com
[b)] 3300732309@qq.com

Abstract. Because the system under test contains multiple subsystem interactions, the correctness and consistency of the interface return between the subsystems plays an important role in the entire process. It is very important to ensure the quality of the interface testing. However, there are many types of interfaces and the structure is complex. As a result, there are problems such as long test execution time, low efficiency, and missing test in the interface test process. To solve this problem, the paper proposes and implements an interface automation test framework based on PHPUnit, so as to improve the test efficiency and ensure the test quality through automated testing. The framework is based on the tested interface services using the HTTP protocol and the development language using PHP. The PHPUnit lightweight unit testing framework was used to transform the design and development. Based on the functional design of the PHPUnit framework, the layered design process realizes functions such as scene-driven data isolation, data verification logic encapsulation, etc. Making the framework clear and readable in structure, and easy to use. Such advantages can effectively improve the test efficiency and reduce the test cost.

Key words: PHPUnit; automated test; test framework; HTTP protocol; PHP.

INTRODUCTION

With the continuous development of software development technology, a layered software architecture is generally used in the development process of the development product. In the layered software system, the lower layer provides an application program interface for the upper layer, which is commonly referred to as API (Application Program Interface) as a bridge between software development and different subsystems and modules [2]. The correctness and consistency of the interfaces between systems play a crucial role in the entire software system. At the same time, the Internet business has become increasingly complex and the system has become increasingly large, and the release cycle has become shorter, which has made the testing of products more difficult and the workload has increased significantly. Non-technical and repetitive test work in product testing occupies a large proportion of the total test workload, which requires a lot of resources to guarantee the quality of the product [3].

Although the workload of interface testing is very large, but there are many repetitive tasks, such as interface file parsing, definition and setting of interface parameter values, and the test is expected to be easily expressed. These characteristics indicate that the interface test is suitable for use. Automated testing techniques to achieve [3]. This paper studies the related interface testing and automated testing framework. Based on the transformation of the PHPUnit unit testing framework, it proposes and implements an automated interface testing framework. By automating the test regression function through the framework, it can reduce a large number of repetitive testing tasks and save Test time improves test efficiency.

INTRODUCTION TO RELATED PROTOCOL LANGUAGES AND AUTOMATED TESTING FRAMEWORK

HTTP protocol and PHP

Hypertext Transfer Protocol (HTTP) is the most widely used network protocol on the Internet. HTTP is a stateless application layer protocol based on request and response mode. The client/server mode information exchange process is shown in Figure 1 [5]. The information exchange process is divided into four parts: establishing a connection, sending request information, sending response information, and closing the connection [5].

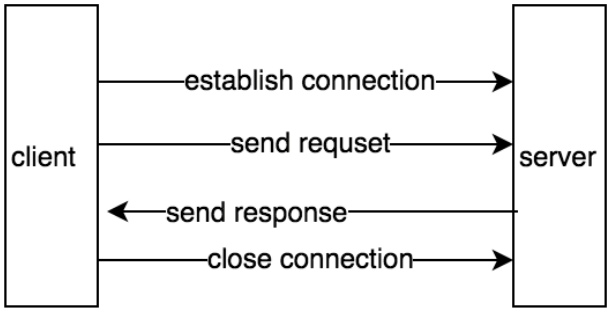

FIGURE 1. client/server mode information exchange process.

PHP language is a general open source scripting language, widely used in WEB application development, simple and flexible, its syntax absorbs the characteristics of C language, Java and Perl, has the advantages of open source, free, fast, and PHP performance Strong, simple, stable, easy to deploy, PHP can help you complete the task at low cost [6].

Interface Testing

Interface testing is the process of testing the API of a system or module using software testing techniques and methods, including the correctness of the verification function and logic, the input and output of the interface, the processing of data, and the compatibility of the system. Interface testing also needs to ensure that the system development process is correct and efficient [7]. Interface testing is different from unit testing. It requires comprehensive and continuous attention to interface quality from many aspects such as system flow, business logic, and user experience. In the absence of automated testing, there are often various reasons leading to missed measurement problems. Automated regression through the interface can both relieve the repeated return of manpower and ensure the quality of the test. Interface testing is the best solution under the drive of ensuring the inherent quality of high-complexity system quality and low-cost economic benefits. As a complete system, the overall quality is fully guaranteed [8].

Interface Automation Test Common Framework

The use of test frameworks varies from development language to development language. From a linguistic point of view, test-driven development originated in the object-oriented Smaltalk community and is shaped in the Java language. It is marked by the introduction of JUnit, the automated unit testing framework for Java widely used in practice [9]. The XUnit family derived from JUnit has become an indispensable tool to make multilingual unit testing possible [9]. Through the appropriate design and expansion of existing frameworks, it constantly enriches its own functions and ease of use to make it more effective. The following describes the commonly used automated testing framework:

(1) JUNIT. JUnit is the de facto standard test framework of JAVA language, and it is the most basic tool in interface testing technology [9]. It fully combines the superiority of Java programming language and continues the design principle of high aggregation and low coupling [10]. Contains the following main features: batch operation,

assertion mechanism, test report, easy to extend [9]. Analyzing the design architecture of JUnit and analyzing its adopted design patterns not only helps to grasp the JUnit unit testing method flexibly, but also provides theoretical preparation for secondary development [10].

(2) PHPUNIT. PHPUnit is a lightweight PHP testing framework. It is a member of the XUnit family of test frameworks and usually represents a subclass inherited from PHPUnit_Framework_TestCase. It has its own language features such as support classes, precisely because of this PHPUnit may be able to imitate the structure of JUnit [11]. It provides automated testing methods for PHP unit testing, making automated testing of agile development possible. PHPUNIT supports open source framework for test-driven development with PHP [12].

PHPUnit conducts unit tests in four phases: setting up the fixture, executing the system under test (marking @), verifying the result (asserting assert), and removing the fixture. Based on this, PHPUnit supports the declaration of explicit dependencies between test methods. This dependency is not defined in the execution order of the test method, but allows the producer to return an instance of the test fixture and pass the instance to consumers who depend on it.A producer is a test method that can generate a unit under test and use it as a return value; a consumer is a test method that depends on one or more producers and their return values.

(3) DbUnit. Dbunit is a database testing framework based on the JUnit extension. The purpose is to keep the database in a known state before and after the test runs. It provides a large number of classes abstracted and encapsulated for database-related operations [13]. DbUnit ported to PHP/PHPUnit can be used to provide support for database interaction testing.

PROGRAM DESIGN BASED ON PHPUNIT INTERFACE AUTOMATION TESTING FRAMEWORK

Automated test framework module design

The software testing framework is a set of automated testing specifications, the basic code of test scripts, and a collection of testing ideas and conventions that can be used to reduce redundant code, increase code productivity, improve code reusability, and maintainability. Its benefits lie in code reconstruction and regression testing [14]. Based on the PHP interface implemented by the tested service, PHPUnit can better support the PHP language. Therefore, this paper designs and implements it based on the PHPUnit unit automation framework.

The automated test framework is divided into 7 functional modules according to the module's function: tools, conf, data, case, lib (scene and verification), data acquisition, and public test API. See Figure 2. The function of each module is as follows:

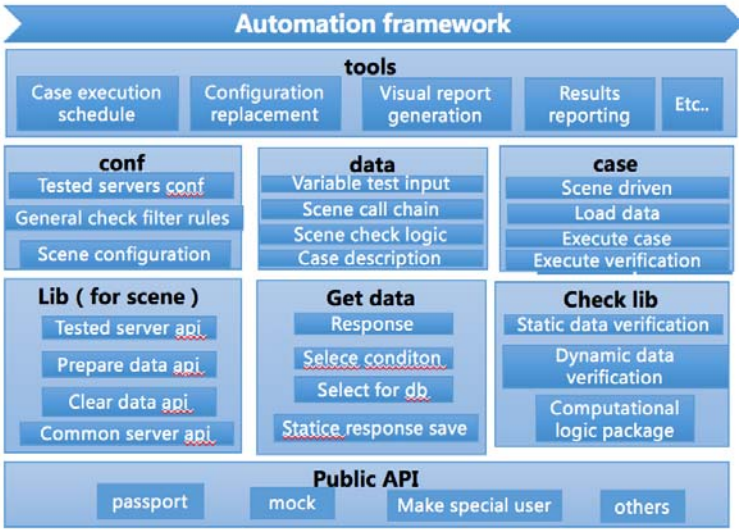

FIGURE 2. Automated testing framework.

Tools: mainly includes some script files, which are mainly responsible for serial execution of the case, specifying the configuration replacement, generating visual test result reports, and uploading results.

conf: conf is divided into two major functional modules. Some are responsible for generic filter rules validation, such as culling of data that does not require validation for interface returned items or database check items. Another function is mainly the definition of the related constants such as the address and configuration of the service under test and the configuration of the scene.

Data: data is mainly responsible for test case input parameters and test scenario call chains. The different input conditions corresponding to different test cases make the framework have the function of supporting variable test input. Different input parameters also make up different scene calls.

Case: mainly divided into four parts: scene-driven, run test cases before the initialization of test data; load the input parameters in data; execute test cases in accordance with the scene; the results of the implementation of the return, database, etc. through the api call for verification.

Lib: lib is mainly divided into function APIs provided by scene dimensions and apis for verifying automation results. The function api covers api methods that provide the tested service api, test case execution data preparation api, test case execution data restore api and public service invocation. The result check api is mainly divided into check-up methods for dynamic and static result check api and check result involving some calculation logic.

Get data: provides the handling of the response result returned by the interface, the contents of the database query involved in the execution of the case, and the access functions for some static results.

Public api: mainly contains packages of some common methods, such as login methods, public query interfaces, and mock interfaces.

Automation Framework Implementation Process

Case execution executes the case file name through the phpunit command. The case execution process is responsible for the data construction of the entire process, case execution, and result verification. The datacaseProvider in the case file provides the core manager method. This method mainly obtains the data use case set, data validation items, and the switch control of the result generation or comparison. If the verification result is generated, the generated result is written into the specified data file. If the result verification, the return of the actual test interface is compared with the expected result of the interface in the specified file. The test case execution process involves the invocation of conf, lib, public api and other modules, the request for data preparation and construction before requesting the tested interface, the sending of interface requests, and the verification of response results with expected results. Call the process shown in Figure 3.

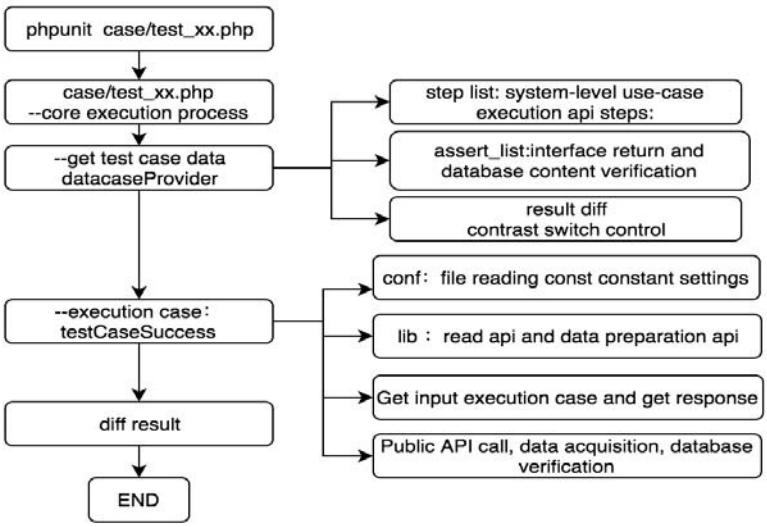

FIGURE 3. Automation framework case execution procedure call.

Through the above transformation of the automation framework, an automated case can be implemented that includes the execution of multiple test cases. Through the case's entry file execution, the case gets the set of use cases in the data and executes the use case. Before executing the use case, the data construction api will be called before execution of the use case to satisfy the premise of the scenario. After the case is executed, the restore information API is called to satisfy the condition that the case can be repeatedly executed. Case execution also involves the acquisition of data results and the invocation of some public test APIs. Finally, the execution of an automated test case invokes the various modules covered by the automation framework.

IMPLEMENTATION OF THE TEST FRAMEWORK

The automation framework code structure is shown in Figure 4:

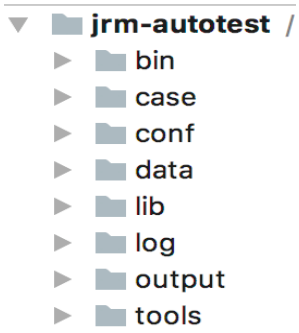

FIGURE 4. Automation framework code structure.

The case module is the entry to the case execution and is responsible for calling the core of each module. The case**Provider uses the functionality provided by the PHPUnit framework to obtain the contents of a specified data file. Here is the test case in the data file, returned in array format. An array represents a test case scenario. The function that starts with testcase** is the entry function for execution, as shown in Figure 5:

```
class TestZLPreCreditAuthen extends FbuBase {
    protected function setUp() {
        $this->logid = $this->openCaseLog();
    }
    public function caseSuccessProvider() {
        return $this->addPath(DataZLPreCreditAuthen::case_list_success(), className: 'DataZLPreCreditAuthen');
    }
    public function testCaseSuccess($stepList, $assertList, $staticFile) {
        $objSceneLoanManager = new SceneManager();
        $requestAndResponseAndDbList = $objSceneLoanManager->manager($stepList, $assertList, $staticFile);

    }
}
```

FIGURE 5. the main flow code.

The function manager is responsible for the core processes throughout the execution of the case. Including the preparation of the input parameters of the test execution flow, auto_set_params_by_conf implements the integration of the relation Map relation specified in data. The retList is responsible for receiving the returned result of the tested interface. Auto_check_by_step is responsible for comparing the specified expected result with the actual interface returned result. Is_write controls whether to compare the results or write the desired comparison file through the switch in the data file, shown in Figure 6.

```php
public function manager($stepList,$assertList,$staticFile){
    $fileList=explode( delimiter: '/',$staticFile['name']);
    $retList=array();
    foreach($stepList as $index =>$stepIndex){
        foreach($stepIndex as $stepName=>$value){
            if(isset($value['jumpMap'])) {
                $jumpMap = $value['jumpMap'];
                $value['params'] = self::auto_set_params_by_data_case($jumpMap,$value['params'],$retList);
                if($value['params']['jump']) {
                    continue;
                }
            }
            //scene_conf
            if(isset(conf_scene::$functionMap[$stepName]['relationMap'])){
                $relationMap=conf_scene::$functionMap[$stepName]['relationMap'];
                $params_keys=self::get_params_key($value['params']);
                if(is_null(conf_scene::$functionMap[$stepName]['relationMap'])){
                    break;
                }
                $relationMap=self::unset_relation_map($params_keys,$relationMap);
                $value['params']=self::auto_set_params_by_conf($relationMap,$value['params'],$retList);
            }
            $api=explode( delimiter: ".",conf_scene::$functionMap[$stepName]['api']);
            $className=$api[0];//conf_scene::$functionMap[$stepName]['className'];
            $functionName=$api[1];//conf_scene::$functionMap[$stepName]['functionName'];
            $classObj=new $className();
            $retList[$index]=array($stepName=>$classObj->$functionName($value['params']));

            if($assertList[$index]!=null){
                $objAutoGet=new AutoGet();
                $retList[$index][$stepName]['db']=$objAutoGet->auto($retList[$index][$stepName],$assertLis
                $objChecker=new Checker();
                $objChecker->auto_check_by_step($retList[$index],$assertList[$index],$staticFile,$index);
            }
        }
    }
    self::is_write($retList,$assertList,$staticFile);
    return $retList;
}
```

FIGURE 6. Logic for manager.

The data content is shown in the example, which is returned to the input parameter of the case file interface by case_list_success. Step_list mainly stores interface related parameters, assert_list handles validation of response and db specified results. StaticFile specifies the file name and description of the desired result, as shown in Figure 7.

```php
class DataZLPreCreditAuthen {
    public static function step_list($para_arr) {
        $precreditAuthenParams = $para_arr;
        $stepList = array(
            '0' => array(
                'realNameSubmit' => array(
                    'params' => $precreditAuthenParams,
                ),
            ),
        );
        return $stepList;
    }
    public static function assert_list() {
        $assertList = array(
            0 => array(
                'realNameSubmit' => array(
                    'response' => 'static',
                    'db' => array(
                        'user_user' => 'static',
                    ),
                ),
            ),


        );
        return $assertList;
    }
    public static function case_list_success() {
        $caseList = array(
            0 => array(
                self::step_list(
                    array(
                        'login' => array(
                            'login_name' => 'jrmall01',
                            'u_passid' => '2500430250',
                        ),
                        'name' => 'xixijun',
                        'idcard' => '110101198406043048',
                    )
                ),

                self::assert_list(),
                'staticFile' => array(
                    'name' => 'case_s_000',
                    'write' => 'off',
                    'desc' => 'test demo success case',
                ),
            ),
        );
        return $caseList;
    }
}
```

FIGURE 7. Logic for data.

EFFECT APPLICATION

At present, the framework has been applied to Baidu's financial mall product project. Prior to the projects testing, it has given precedence to automated test cases. Some grammatical problems have been discovered before the testers have been involved in testing, and new functions have introduced problems that affect old functions, and they are extremely save testers' test time. The test case execution effect is shown in Figure 8:

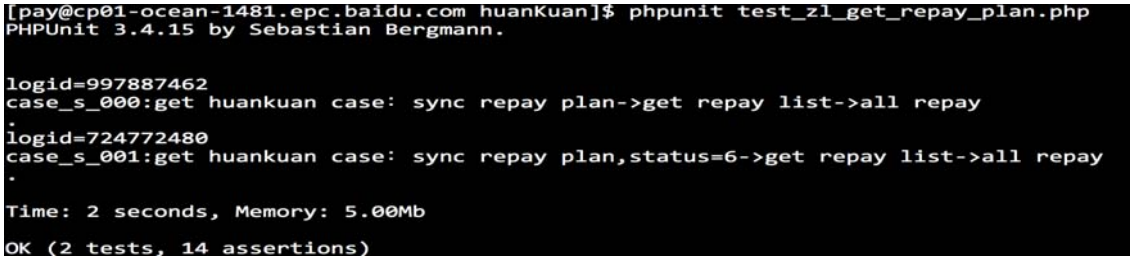

FIGURE 8. Automated execution results.

SUMMARY

This article mainly studies the design and implementation of the HTTP interface using the PHPUnit framework for automatic testing. The concept of the HTTP protocol and interface is introduced. The development language based on the business product implemented by the test case is determined to be developed using the PHPUnit framework. The existing unit testing frameworks Junit, PHPUnit and Dbunit are introduced. Finally, the design and implementation of the interface automation testing framework based on PHPUnit is implemented. The framework tools are applied to commercial products, which saves testing time and improves testing efficiency.

REFERENCES

1. Design and Implementation of a Data Driven Software Interface Automation Test Framework. Liu Zhi Nanjing Institute of Electronic Technology.
2. Research and Implementation of Software Interface Automated Test Technology Ye Bo, Sun Junruo, Lin Jie Nanjing Institute of Electronic Technology, Nanjing, Jiangsu Province.
3. Min Jing, Zhong Yiping, Zhang Shiyong. Automated Web Performance Testing Based on Protocol Analysis [J]. Computer Engineering, 2005, 31(7):66-69.
4. WASSERMANN G, Yu Dachuan, CHANDER A. Dynamic test input ceneration for Web applications[J]. ISST ’08, July, 2008: 20-24.
5. DOUGLAS E, COMER, DAVID L, et al. Internetworking with TCP/IP (Volume 1) 4th Edition [M]. Lin Yao, Jiang Hui, Du Weixuan, and Trans. Beijing: Publishing House of Electronics Industry, 1998.
6. Baidu Encyclopedia.
7. Taobao (China) Software Co., Ltd., Interface Testing White Paper [R], Zhejiang, Taobao Platform Testing Group, V0.1, 2009, 3-5.
8. Zheng Renjie, Software Testing [M], Beijing, People's Posts and Telecommunications Press, 2011, 70-72, 153-155.
9. Kent Beck, Erich Gamma, JUnit Cookbook [J/OL]http://junit.sourceforge.net/doc/cookbook/cookbook.htm.
10. Design and Implementation of Automation Testing Framework for Communication Equipment Based on JUnit Gu Mengqi Lanzhou University.
11. Test Driven Development in PHP Application Renmin University of China School of Information.
12. PHPUnit Pocket Guide.
13. DBUNIT ORG, About DbUnit[P/OL],2010, http://www.dbunit.org/.
14. Liu Sheng, Software Automation Testing Framework Design and Practice [M], Beijing, People's Posts and Telecommunications Press, 2009, 112-127.