УДК 681.3.06

АВТОМАТИЗАЦИЯ ФУНКЦИОНАЛЬНОГО ТЕСТИРОВАНИЯ WEB-ПРИЛОЖЕНИЙ

[1]Медведев Ю.С., [2]Терехов В.В.

[1]Северо-Кавказский филиал ФГБОУ ВПО «Российский государственный университет правосудия», Краснодар, e-mail: ysm-73@ya.ru;

[2]Краснодарское высшее военное авиационное училище лётчиков имени Героя Советского Союза А.К. Серова, Краснодар, e-mail: partner2002@front.ru

Для взаимодействия с конечным пользователем корпоративные информационные системы в последнее время широко используют web-интерфейс. По мере увеличения сложности web-приложений неизменно возрастает вероятность нарушения их функциональности. Нередко обнаружить ошибку в работе web-приложения бывает весьма сложно. Это связано как с большим количеством страниц, требующих тестирования, так и с трудоёмкостью перебора сочетаний значений всех входящих параметров, влияющих на сгенерированный результат. К тому же web-приложения постоянно модифицируются: добавляются новые страницы, дополняется функциональность приложения. Поэтому разработчики были вынуждены отказаться от тестирования web-приложений вручную по причине его высокой трудоёмкости. В статье анализируются различные подходы к автоматизации функционального тестирования web-приложений. Такая диагностика позволяет выявить причины нарушения функциональности, обеспечив надёжную работу web-приложения.

Ключевые слова: web-интерфейс, web-приложение, ajax-приложение

AUTOMATION OF FUNCTIONAL TESTING OF WEB APPLICATIONS

[1]Medvedev Yu.S., [2]Terekhov V.V.

[1]North-Caucasian branch of «Russian State University of Justice», Krasnodar, e-mail: ysm-73@ya.ru;

[2]Krasnodar higher military aviation school named after Hero of the Soviet Union A.K. Serov, Krasnodar, e-mail: partner2002@front.ru

For interaction with end user corporate information systems in recent times are widely using the web interface. With increasing complexity of web applications invariably increases the likelihood of violations of their functionality. Often to find a bug in the web application is very difficult. This is due to a large number of pages, requiring the testing and the complexity of iterating through the combinations of the values of all input parameters that affect the generated result. Besides, web applications are constantly modified, add new pages, complemented by the functionality of the application. So the developers were forced to abandon testing web applications manually due to its high complexity. The article analyzes various approaches to automating functional testing of web applications. Such diagnostics allows to identify the causes of violation of functionality, ensuring reliable operation of a web application.

Keywords: web interface, web application, ajax application

Корпоративные информационные системы, предназначенные для интеллектуальной поддержки и организации технологических процессов, широко используют возможности глобальной сети Internet для реализации удаленного доступа к необходимым приложениям. Поэтому наибольшее распространение для взаимодействия с конечным пользователем в последние годы получил web-интерфейс.

До недавнего времени web-ресурсы, как правило, представляли собой лишь копии традиционных источников информации в формате html. И лишь спустя годы web-приложения перестали быть системами распространения статического контента. На сегодняшний день это распределённые персонализированные приложения уровня предприятия [1]. Разработка и поддержание таких программных комплексов является весьма сложной задачей.

Развитие технологий программирования на стороне web-браузера достаточно долго сдерживалось, так как web-браузеры обеспечивали весьма слабую совместимость, позволяя кроссбраузерно отображать только самые простые HTML-документы. Код более сложных web-приложений разрабатывался, как правило, для конкретной версии web-браузера [2].

Появление Rich Internet application (RIA, «богатое Интернет-приложение») позволило расширить круг решаемых задач и расширить сферу применения web-приложений. До появления технологии Ajax web-приложения выполнялись преимущественно на стороне web-сервера. Web-браузер в таком случае играл роль пассивного монитора, который отображал полученный с web-сервера HTML-документ. Управление поведением элементов пользовательского интерфейса осуществлялось путём полной перерисовки HTML-документа, полученного новым запросом с web-сервера, а это слишком тяжеловесное решение [2].

В противоположность классическому web-приложению, Ajax-приложение выполняется как на web-сервере, так и на web-клиенте, т.е. в web-браузере. Использование фонового (без перезагрузки основного HTML-документа) обмена данными между web-браузером и web-сервером позволяет создавать по-настоящему динамические web-приложения.

Настоящий прорыв совершила технология Ajax, приблизив web-приложения к традиционным интерактивным настольным приложениям. На сегодняшний день невозможно представить web-приложения без использования графических изображений, CSS, JavaScript.

Для повышения производительности применяют кеширование объектов, прежде всего графических изображений. Для загрузки изменённых файлов, как правило, изменяют их имена, программно обновляя ссылки на них в web-приложениях, применяя сценарий сборки. Для оптимизации кеширования настраиваются соответствующим образом заголовки в запросах, передавая инструкции серверу о сроке хранения ресурсов [1, 2].

Широкое распространение получили следующие методы повышения скорости загрузки web-страницы [3, 4]: использование многоуровневой архитектуры FrontEnd-BackEnd; кэширование данных на стороне сервера; кэширование web-страниц (на стороне сервера либо на стороне клиента); использование web-сервера, построенного по FSM (Finite State Machine), сжатие передаваемых данных средствами протокола HTTP. Каждый из этих методов в определённой мере способствует повышению производительности web-приложения, они могут применяться как по отдельности, так и комплексно разработчиками web-приложений.

Значительного внимания заслуживает механизм взаимодействия с DOM. Производительность web-приложения в значительной мере можно повысить, грамотно используя стили вместо многократных перерисовок изображения страницы изменениями программного кода [1].

Очевидно, что удалённые ресурсы медленнее загружаются по сети. Резервом увеличения производительности является использование сети доставки содержания (CDN), которая, кроме того, применяет gzip-сжатие и контроль за кешированием файлов [1, 2]. Распределение файлов по сети основывается на географическом расположении web-клиентов.

При помощи расширений большинства современных браузеров, Firebug для Mozilla® Firefox®, Web Inspector для Safari®, Developer Tools для Chrome®, Developer Tools для Internet Explorer® 8 [5], можно осуществить анализ сетевого трафика, вызванного загрузкой страницы web-приложения (количество HTTP-запросов, размер и количество загружаемых ресурсов, HTTP-заголовки, сведения о кэше для исследуемого файла и т.д.). Такая диагностика позволяет выявить причины низкой производительности и повысить скорость работы web-приложения.

Применительно к языку PHP высокую эффективность показали программы-акселераторы, позволяющие кэшировать скомпилированный байт-код [2]. Для кэширования неоткомпилированного кода предлагается использовать программу Memcached. Это может ускорить процесс обращения к базе данных [2].

Популярные в настоящее время в среде web-программистов библиотеки JavaScript, такие как Prototype, JQuery, Dojo, MooTools, YUI, ExtJS, дают возможность ускоренной разработки web-компонентов, позволяют решить проблемы кросс-браузерной совместимости. Однако следует иметь в виду, что их применение целесообразно в том случае, если используется весь арсенал имеющихся средств. В противном случае разумнее использовать более легковесное решение – написать код JavaScript. Следует также ограничить число обработчиков событий в web-приложении, т.к. их чрезмерное использование также неминуемо скажется на производительности web-приложения.

В целях уменьшения размеров файлов используют специальные утилиты (YUI Compressor, PngCrush, PngOptimizer). В файлах JavaScript и CSS удаляются лишние биты: удаляются пробелы, комментарии, переводы строк, заменяются на более короткие имена переменных. Изображения также оптимизируются. Дополнительно на web-сервере файлы могут сжиматься утилитой GZIP [4].

Отличительной особенностью динамичных интерактивных web-приложений является высокая нагрузка на сервер. Для повышения производительности применяют кеширование объектов, прежде всего графических изображений. Ведь зачастую объекты не изменяются, поэтому при последующих запросах они могут использоваться браузером повторно (к примеру, верхний (header) и нижний (footer) блоки web-страниц). Для загрузки изменённых файлов, как правило, изменяют их имена, программно обновляя ссылки на них в web-приложениях, применяя сценарий сборки. Для оптимизации кеширования настраиваются соответствующим образом заголовки

в запросах, передавая инструкции серверу о сроке хранения ресурсов [2].

Для количественной оценки влияния значительной части рассмотренных методов на производительность web-приложений в ходе исследования авторами проводились имитационные эксперименты. Используемые программные средства: сервер СУБД MySQL 5.0, web-сервер Apache 2.2.6, язык сценариев PHP 5, web-сервер с архитектурой FSM Nginx 0.6.25, программа имитации клиентских запросов siege. Результаты приведены в табл. 1. Прирост производительности в значительной степени варьируется в зависимости от количества клиентских запросов.

Рассмотренные методы в отдельности могут не привести к значительному повышению производительности web-приложения, однако проведение комплекса предлагаемых мероприятий может значительно увеличить скорость загрузки сайта, что особенно актуально для web-приложений с большим трафиком.

Таблица 1

Результаты имитационных экспериментов

№ п/п	Исследуемый метод	Прирост производительности, %
1	Кэширование данных на стороне сервера	до 30
2	Кэширование web-страниц на стороне сервера	до 90
3	Кэширование web-страниц на стороне клиента	до 85
4	Использование многоуровневой архитектуры FrontEnd-BackEnd	до 125
5	Использование web-сервера, построенного по FSM	до 140
6	Сжатие передаваемых данных средствами Apache	до 135

По мере увеличения сложности web-приложения неизменно возрастает вероятность нарушения его функциональности. К примеру, в электронных версиях многих газет и журналов зачастую встречаются ссылки, при переходе по которым мы получаем сообщение типа «Страница не найдена».

Нередко обнаружить ошибку в работе web-приложения бывает весьма сложно. Это связано как с большим количеством страниц, требующих тестирования, так и с трудоёмкостью перебора сочетаний значений всех входящих параметров, влияющих на сгенерированный результат. К тому же web-приложения постоянно модифицируются: добавляются новые страницы, дополняется функциональность приложения.

Поэтому достаточно давно разработчики были вынуждены отказаться от тестирования web-приложений вручную по причине его высокой трудоёмкости.

Существует ряд подходов к процессу автоматизации функционального тестирования web-приложений, которые нашли свою реализацию в целом ряде программных продуктов.

Наиболее распространённым является подход **Record & Playback** [1]. Сценарии тестирования создаются на основе записи действий пользователей при реальной работе с web-приложением. При тестировании записанные действия выполняются, а результат сравнивается с эталонными страницами.

Как правило, для сравнения с эталоном принимаются во внимание такие параметры, как URL, название, размер и контрольная сумма для текста страницы, дата её последнего изменения и др. Значения всех этих параметров не позволяют сделать правильные выводы о функциональности современных динамичных интерактивных web-приложений, работающих во взаимодействии с базой данных, записи которой постоянно модифицируются.

Определённую сложность представляет процесс записи сценария тестирования, так как если при записи возникнет ошибка, то её необходимо исправить (для тестирования нужен эталонный результат) и повторить запись. Очевидно, что после модификации web-приложения сценарии тестирования теряют свою актуальность, требуется их перезапись.

Подход **Data Driven** развивает предыдущий подход [2]. Для расширения функциональности программы тестирования позволяют задавать точность сравнения, а также дополнительно задавать диапазоны значений входных параметров. При этом для каждого возможного варианта поведения web-приложения потребуется свой сценарий тестирования.

К достоинствам рассмотренных подходов следует отнести простоту освоения, так как разработчику тестов не требуются навыки программирования.

К тому же современные инструменты позволяют оперировать командами типа «ввести значение», «нажать кнопку», «проверить результат», которые затем автоматически воспроизводятся программой применительно к тестируемому web-приложению (подход **Keyword Driven**) [1, 3]. Тем самым предпринимается попытка абстрагироваться от конкретного приложения. Кроме того, создание тестов может осуществляться параллельно с разработкой самого web-приложения.

Большинство предлагаемых программ тестирования способны анализировать код страницы на наличие в нём ссылок на другие разделы web-приложения. Автоматически осуществляя переход по извлекаемым ссылкам, программа проверяет работоспособность приложения.

Однако без тонкой настройки программы тестирования, ввиду отсутствия эталонных страниц для сравнения, такая функция зачастую оказывается бесполезной либо находит применение только при нагрузочном тестировании web-приложений [2].

(например, LSB Навигатор, разработанный ИСП РАН) [1, 3]. Помимо анализа результатов выполнения сценариев тестирования необходимо также уделить внимание содержанию лог-файлов web-севера, куда заносятся события, так или иначе связанные с нарушением функционирования web-приложения.

Для количественной оценки эффективности рассмотренных подходов проводились имитационные эксперименты, в ходе которых оценивалась возможность поиска ошибок в программном коде приложений. Результаты приведены в табл. 2.

Таблица 2

Результаты имитационных экспериментов

№ п/п	Исследуемый подход	Процент нахождения ошибок в программном коде приложений
1	Подход Record & Playback	до 43
2	Подход Data Driven	до 67
3	Подход Keyword Driven	до 74
4	Подход, основанный на анализе исходного кода web-приложений	до 92

Наибольший интерес представляют методы функционального тестирования, основанные на анализе исходного кода web-приложений.

Как известно, в скриптовых языках доступ к передаваемым параметрам осуществляется через ассоциативные массивы. Так, в популярном языке web-программирования PHP для этого используется суперглобальный массив $_REQUEST, либо массивы $_GET и $_POST (для соответствующих методов передачи данных по протоколу HTTP). Имена переменных, значения которых передаются скрипту, являются ключами рассматриваемых массивов. Несложно извлечь из исходного кода имена переменных, проанализировать передаваемые значения, а затем использовать полученные данные при создании тестов.

Конечно, существуют определённые сложности, связанные с обращением скрипта при его работе к базе данных, возможном подключении дополнительных файлов, необходимости определения в исходном коде условных операторов, содержащих константы, с которыми сравниваются значения передаваемых скрипту параметров, и др.

Однако современные программные продукты умеют эти трудности преодолевать

Современные многофункциональные web-приложения – это приложения, которые удобны для пользователя и обеспечивают функциональность, ставшую уже привычной для настольного (desktop) приложения. Для исключения возможных сбоев необходимо уделять должное внимание мероприятиям по функциональному тестированию. В помощь web-разработчику предлагается значительный арсенал рассмотренных средств. Такая диагностика позволяет выявить причины нарушения функциональности, обеспечив надёжную работу web-приложения.

Список литературы

1. Веллинг Л., Томсон Л. Разработка Web-приложений с помощью PHP и MySQL, 3-е изд.: пер. с англ. – М.: Вильямс, 2013. – 880 с.

2. Овчаренко А.В. Ajax на примерах. – СПб.: БХВ-Петербург, 2010. – 432 с.

3. Петин В.А. Сайт на AJAX под ключ. Готовое решение для интернет-магазина. – СПб.: БХВ-Петербург 2011. – 432 с.

4. Фаулер М. Архитектура корпоративных программных приложений. – М.: Изд. дом «Вильямс», 2006. – 540 с.

5. Хольцнер С. jQuery. Практическое применение. – М.: Эскмо, 2010. – 224 с.