

Webanwendungen testen mit twill und Selenium

C. Titus Brown, Grig Gheorghiu und Jason R. Huggings
Übersetzung von Lars Schulten

Dieses TecFeed ist eine Einführung in die Erstellung automatisierter Webtests mithilfe der beiden Werkzeuge twill und Selenium. twill ist eine einfache Web-Scripting-Sprache, die eingesetzt werden kann, um Webtests zu automatisieren. Selenium ist ein Web-Testing-Framework, das in jedem Browser läuft und dazu verwendet werden kann, komplexe Websites zu testen, die ausgiebig Gebrauch von JavaScript machen.

Am besten nutzen Sie dieses TecFeed, indem Sie die Beispiele direkt anwenden. Wir gehen davon aus, dass Sie innerhalb einer Stunde damit beginnen können, Ihre eigenen Funktionstests in twill oder Selenium zu schreiben, und innerhalb eines Tages die meisten, wenn nicht sogar alle Möglichkeiten und Einschränkungen dieser Werkzeuge verstanden haben.

O'REILLY®

INHALT

Einführung | 2

Web-Funktionstests mit twill | 4

Web-Funktionstests
mit Selenium | 27

Eine einfache Webanwendung
mit twill und Selenium testen | 42

Abschlussbemerkung | 56



TecFeeds

www.tecfeeds.de

Einführung

Willkommen!

Dieses TecFeed ist eine Einführung in die Erstellung automatisierter Webtests mithilfe der beiden Werkzeuge twill und Selenium. twill ist eine einfache Web-Scripting-Sprache, die eingesetzt werden kann, um Webtests zu automatisieren. Selenium ist ein Web-Testing-Framework, das in jedem Browser läuft und dazu verwendet werden kann, komplexe Websites zu testen, die ausgiebig Gebrauch von JavaScript machen.

Am besten nutzen Sie dieses TecFeed, indem Sie die Beispiele direkt anwenden. Wir gehen davon aus, dass Sie innerhalb einer Stunde damit beginnen können, Ihre eigenen Funktionstests in twill oder Selenium zu schreiben, und innerhalb eines Tages die meisten, wenn nicht sogar alle Möglichkeiten und Einschränkungen dieser Werkzeuge verstanden haben.

Dieses TecFeed ist in drei Teile aufgliedert. Erst werden wir twill vorstellen und Ihnen zeigen, wie Sie mit twill im Web surfen und Webanwendungen testen können. Dann stellen wir Selenium vor und betrachten, wie man mit Selenium Sites testet, die JavaScript- und Ajax-Technologien verwenden. Schließlich bauen wir mit twill und Selenium ein paar einfache Tests für eine simple Django-Anwendung und zeigen Ihnen, wie Sie diese Tests in Ihren Entwicklungsprozess integrieren.

Was wir nicht behandeln

Dies ist kein How-to zum Testen von Anwendungen.

Das heißt, dass dieses TecFeed als eine Einführung in *technische Lösungen* für die Automatisierung von Webtests gedacht ist. Wir, die Autoren, halten die Webtestautomatisierung für einen entscheidenden Bestandteil beim Entwurf, Schreiben und Verteilen zuverlässiger Webanwendungen, und wir bevorzugen Strategien der agilen Softwareentwicklung und des agilen Testens. Es ist aber nicht unsere Absicht, eine ausführliche Diskussion dieser Strategien zu liefern.



Wir empfehlen die folgenden Einführungsbücher zum Testen: *Lessons Learned in Software Testing* von Cem Kaner, et al. (Wiley), *Testing Computer Software* von Cem Kaner, et al. (Wiley), *A Practitioner's Guide to Software Test Design* von Lee Copeland (Artech House Publishers) sowie *How to Break Software* von James A. Whittaker (Addison-Wesley).

Was sind twill und Selenium?

twill ist ein HTTP-Protokolltreiber, der einen Kommandozeilen-Webbrowser imitiert. Es »spricht« HTTP und ermöglicht Ihnen, Anwendungen interaktiv über die Kommandozeile aufzurufen. twill ermöglicht Anwendern außerdem, den Webzugriff zu *skripten*, versteht allerdings kein JavaScript und kann deswegen nur die clientseitigen Funktionalitäten testen.

Selenium ist ein browserbasiertes Programmiersystem. Es ermöglicht Ihnen, Webbrowser zu steuern, indem Sie entweder eine einfache, tabellenbasierte Sprache einsetzen oder ein Fernsteuerungsprotokoll. Weil Selenium *innerhalb* des Browsers läuft, kann es clientseitiges JavaScript ziemlich einfach testen. Und da es außerdem in allen beliebteren Webbrowsern läuft, kann es darüber hinaus eingesetzt werden, um Ihre clientseitigen JavaScript-Funktionalitäten in all diesen Browsern zu testen. Damit stellt es sicher, dass Ihre Website in allen Browsern funktioniert.

Wir hoffen, dass Ihnen, nachdem Sie dieses TecFeed gelesen haben, die Möglichkeiten und Nachteile des Einsatzes von twill und Selenium gleichermaßen klar sind.

Warum behandeln wir nur twill und Selenium? Es gibt viele, viele andere Werkzeuge für das Web-Testing. Dass wir diese beiden und nur diese beiden behandeln, hat verschiedene Gründe:

1. Wir haben sie geschrieben, sind deswegen mit ihnen vertraut und fühlen uns wohl mit ihrem Design, ihrer Funktionalität und ihren Einschränkungen.
2. twill und Selenium umfassen unsere Testbedürfnisse sehr gut. twill ist eine leichtgewichtige Testlösung, die sich leicht verwenden, automatisieren und ausführen lässt, in Python leicht erweitert werden kann, plattformunabhängig ist und von Haus aus dazu in der Lage ist, fast alle HTTP-Funktionalitäten zu testen. Selenium ist eine schwerge-

wichtigere Testlösung, die fast alle Websites in den meisten Browsern und auf jeder Plattform testen kann.

3. twill und Selenium sind beide Open Source. Wenn Sie sie verwenden, »kaufen« Sie sich in eine *Gemeinschaft* ein, nicht in ein Unternehmen. Sie werden nie aus falschen Gründen auf diese Werkzeuge festgelegt und müssen kein Geld ausgeben, um sie zu testen. Wir erwarten, dass Sie beide Werkzeuge anhand dieses TecFeeds in weniger als einem Tag beurteilen und eine fundierte Entscheidung darüber treffen können, ob sie Ihren Bedürfnissen genügen oder nicht.
4. Die meisten anderen Werkzeuge für Webtests sind entweder plattform- oder browserspezifisch. Weder twill noch Selenium schränken Sie auf eine Plattform oder (im Fall von Selenium) auf einen einzigen Browser ein. Außerdem sind beide Werkzeuge sprachneutral: twill wurde in Python geschrieben, kann aber mit Websites verwendet werden, die in einer beliebigen Sprache geschrieben sind. Selenium Core implementiert eine eigene Sprache, die auf JavaScript basiert, während Selenium Remote Control mit fast jeder Sprache funktioniert. Selenium kann also eingesetzt werden, um dynamische Websites zu testen, die in einer beliebigen Sprache geschrieben sind.

Zwei weitere Open Source-Werkzeuge für die Automatisierung von Webtests, die die meisten dieser Kriterien ebenfalls erfüllen (vom ersten abgesehen, da wir nicht ihre Autoren

sind), sind Watir (<http://wtr.rubyforge.org/>) und Sahi (<http://sahi.co.in/>).

Watir ist ein Ruby-basiertes System für die Steuerung des Internet Explorer, das auch mit Firefox funktioniert. Es ist ein sehr ausgereiftes System für Webtests, und wir empfehlen Ihnen, es auszuprobieren. Für browserübergreifendes Testen lässt es sich allerdings nicht so problemlos einsetzen wie Selenium.

Sahi nutzt eine Proxy-basierte Technik, um Browser über JavaScript zu steuern. Es »konkurriert« direkt mit Selenium. Es ist erheblich neuer und könnte genauso mächtig sein. Aktuell gibt es jedoch unserer Kenntnis nach keinen guten Grund dafür, Sahi Selenium vorzuziehen. Wir sind mit ihm allerdings auch nicht sehr vertraut, da es erst spät aufkam, während wir an diesem TecFeed schrieben.

Mit den Autoren in Verbindung treten

Unter webtesting@idyll.org können Sie direkt mit den Autoren in Verbindung treten, und unter tip@lists.idyll.org können Sie auf der `testing-in-python`-Mailingliste allgemeinere Fragen zum Testen stellen. Mehr Informationen finden Sie unter <http://webtesting.idyll.org/resources>.

Danksagungen

Unser besonderer Dank gilt Jacob Kaplan-Moss, der den Entwurf dieses TecFeeds begutachtet hat!

Web-Funktionstests mit twill

Eine kurze Geschichte von twill

(von C. Titus Brown)

2003 begann ich das erste Mal, nach einer Möglichkeit für die Automatisierung von Webtests zu suchen. Ich hatte zwei große, nicht mehr zu bewältigende Websites, die von Hunderten von Leuten genutzt wurden, und meine Standard-Teststrategie war es, den Sites einfach neue Features hinzuzufügen, diese zu veröffentlichen und auf die Fehlermeldungen zu warten. Dass das nicht besonders befriedigend war, weder für die Anwender noch für mich, muss man gar nicht erwähnen!

Zu dieser Zeit stieß ich auf Cory Dodts PBP, eine Web-Scripting-Sprache, die auf John J. Lees mechanize-Paket aufsetzt, das wiederum Python ermöglicht, HTTP »zu sprechen« und Webseiten zu verstehen. (Johns mechanize-Paket war ein Python-Port von Perls WWW::Mechanize, und PBP hatte Ähnlichkeiten mit Perls WWW::Mechanize::Shell.) Unglücklicherweise befand sich PBP zu dieser Zeit in einer frühen Phase und funktionierte für meine Websites nicht sehr gut.

Spät im Jahr 2004 entschloss ich mich, mir PBP noch einmal anzusehen, weil ich unbedingt einige automatisierte Webtests schreiben musste. Inzwischen war PBP erheblich gereift, und ich konnte damit 90 Prozent meines Weges zur Webtestlösung für meine Websites zurücklegen. Ich begann, Cory Patches zuzusenden, um es zu einer 100-prozentigen Lösung

zu machen. Irgendwann erkannte ich allerdings, dass Cory PBP eigentlich in eine andere Richtung führen wollte. Außerdem begann mich die Architektur von PBP zu enttäuschen, die sich für bestimmte Funktionen auf die ausufernde Twisted-Bibliothek stützte. (PBP entstand aus der Suche nach einem Weg, funktionelle Tests für *twisted.web* zu schreiben: Diese Verknüpfung war also nicht überraschend.)

Ich begann, PBP von Grund auf neu zu schreiben. Meine ursprüngliche Neufassung nutzte eine IPython-basierte Shell, aber ich entdeckte schnell, dass IPython zwar ausgezeichnet für die interaktive Python-Ausführung geeignet ist, aber weniger gut ist, wenn es um die Arbeit mit Skripten geht. Vor allem konnte das Exception-Handling der Shell nicht überschrieben werden. Deswegen begann ich, *cmd* zu verwenden, ein Framework zum Schreiben von kommandozeilenartigen Anwendungen, das mit Python selbst ausgeliefert wird. Und das ist es auch, was twill heute nutzt.

Es überraschte mich etwas, dass twill sofort ein kleiner Erfolg war. Diverse Anwender nahmen es auf und teilten mir das mit. Schnell gingen Fehlermeldungen und Feature-Anfragen ein.

Seit der Portierung auf *cmd* hat sich die Architektur von twill nur wenig geändert. Im Wesentlichen ist twill Verbindungscode, der *mechanize* mit *cmd* verbindet.

Und ich hoffe sogar, dass ich eines nahen Tages dazu kommen werde, die beiden Websites zu testen, die mich überhaupt dazu motiviert haben, twill zu schreiben ...

Was ist twill?

twill ist eine einfache Sprache für Webtests und das Surfen im Web. Die twill-Shell imitiert die Funktionen eines Browsers, so gut es eben geht, bietet statt einer grafischen Schnittstelle aber eine Kommandozeilenschnittstelle. twill soll eine einfache Automatisierung der meisten clientseitigen Webinteraktionen ermöglichen und Erweiterungseinrichtungen bieten, um komplexeren Ansprüchen entgegenzukommen.

All unsere nachfolgenden Beispiele für das Testen von Websites sind in Python geschrieben. twill funktioniert auch wunderbar bei Websites, die in Java, PHP, Perl und Ruby geschrieben sind – allerdings gibt es einige besondere Tricks, die Sie einsetzen können, um Python-Sites mit twill zu testen.

Dieser Abschnitt beginnt mit einer Einführung in die twill-Sprache und führt dann einige einfache Techniken für Webtests vor. Auch wenn das alles ist, was Sie benötigen, um mit Ihren eigenen Webtests zu beginnen, werden wir auch darüber sprechen, wie Sie Ihre Webtests so organisieren, dass ihre Ausführung und Wartung einfacher wird. Anschließend werden wir den *scotch*-Rekorder vorstellen, eine Software, die zwischen Ihrem Browser und einer Website sitzt und den Webverkehr aufzeichnet. *scotch* ermöglicht Ihnen, Ihren Pfad durch eine Website aufzuzeichnen und diesen dann wieder abzuspielen oder als twill-Skript zu speichern.

Die zweite Hälfte dieses Abschnitts erörtert einige fortgeschrittenere Themen einschließlich der Erweiterungsfunk-

tionen, die in twill eingeschlossen sind, und der Frage, wie Sie selbst twill mit Python erweitern können.

Einführung in twill

Am einfachsten stellt man twill vor, indem man es in Aktion zeigt. Zunächst müssen Sie twill installieren (Anleitungen finden Sie unter <http://twill.idyll.org>). Führen Sie dann twill-sh aus und geben Sie Folgendes ein:

```
go http://webtesting.idyll.org/
```

Geben Sie anschließend dies ein:

```
show
```

Sie sehen nun folgende Ausgabe:

```
=> blah
Hello, world!
<p>
<a href="link1">top link</a>
<p>
<a href="link2">another link</a>
<p>
...
```

(Mit `save_html <Dateiname>` können Sie die Ausgabe einer show-Anweisung in einer Datei speichern.)

Probieren Sie den Befehl `showlinks`. Sie werden dann eine Liste aller Links auf der Seite sehen:

Links:

```
0. top link ==> link1
1. another link ==> link2
2. JavaScript alert box ==> JavaScript:%20alert_it()
```

Gleichermaßen zeigt Ihnen der Befehl `showforms` eine Liste aller Formulare auf der Seite:

```
Form #1
## ## __Name_____Type___ID_____Value_____
1   num1                text   (None)
2   num2                text   (None)
```

Das sind einfache Beispiele für die Sprache twill, die Ihnen zeigen, wie Sie eine bestimmte Webseite laden und sich das nackte HTML, die Links auf der Seite sowie die Formulare auf der Seite ansehen können.



Hinweis: Von jetzt an markieren wir in den Beispielen den Text, der in twill eingegeben wird, mit `>>` und lassen darauf die Ausgabe des Beispiels folgen.)

Es ist ziemlich einfach Links nachzugehen:

```
>> follow "top link"
==> at http://webtesting.idyll.org/link1
```

Mit `back` können Sie auch zurückgehen. Dieser Befehl wechselt zur vorangegangenen Seite, ohne sie erneut vom Webserver anzufordern. Nutzen Sie `reload`, um die Seite neu anzufordern: