

Grok or How Zope3 reinvented itself

Róman Joost

12th September 2009

1 Introduction

Zope is an open-source application server and web development framework written in Python and has been around for almost a decade. In this time, the application server was rewritten from scratch and a component architecture was introduced. This led to two major generations of Zope: Zope 2 codebase is referred to as the old code, and Zope 3 refers to the new codebase including the component model.

Because the emphasis changed to more lightweight web development frameworks, such as Ruby on Rails and Django, the Zope community introduced Grok, which is based on Zope 3, but tends to be more agile.

This presentation provides an introduction to Grok. It illustrates how fast web applications can be written, without losing the benefits of Zope 3.

2 Barriers for Zope 3?

By the introduction of the component architecture in Zope 3, a lot of things changed for Zope developers. The component architecture made the old code incompatible with Zope 3¹. Another paradigm was to be more explicit, than implicit. The implicit behaviour of Zope 2 applications sometimes resulted in unpredictable and difficult to debug behaviour. For the explicit configuration of the component architecture, a new XML based markup language called Zope Configuration Markup Language (ZCML) was introduced.

All of this made Zope more extensible and pluggable. The drawback was though, that it raised the learning curve to another level.

3 Why Grok

Writing ZCML can be a tedious task for building a small solution. Furthermore, it makes it much harder for newcomers to work with Zope 3. This resulted in a new project called *Grok*, which aimed to:

¹Although it is now possible to migrate Zope 2 projects to Zope 3, by using Five

- make most of the ZCML configuration either available in Python code,
- present helpful error messages to the developer,
- make writing web applications in Zope more simple and Zope more effective to use.

3.1 How to get started

To create a new grok project, the community created installers, which generate most of the boilerplate. Once grokproject is installed, a simple call creates a developers sandbox using `zc.buildout`:

```
$ grokproject rhang
Enter user (Name of an initial administrator user): admin
Enter passwd (Password for the initial administrator user):
Downloading info about versions...
Creating directory ./rhang
Downloading zc.buildout...
Invoking zc.buildout...
Develop: '/tmp/rhang/'
Installing eggbasket.
[...]
```

After buildout created the sandbox, the developer can either fire up the instance to test it, or start developing.

The most important part of the sandbox is the *src* directory. The grokproject script created a basic application, a basic doctest and view templates.

3.2 Writing tests

A sample test is created by *grokproject*. You can find it in your *src* directory with the name *app.txt*. Most people use doctests for the tests, which have the advantage of being more descriptive about what you're testing.

Another good component is the testbrowser, which is used in the example test. You can use it for testing most of your views, except dynamic AJAX implementations.

3.3 Common Pitfalls

3.3.1 Version Conflicts

It is possible to end up in a version conflict if you use additional components. One reason for this is the version pinning (*versions.cfg*) to achieve a stable set of working components.

One good solution is to try an older version of the component. Most components are very well tested, falling back to an older version doesn't make a big difference for most of the beginners.

3.3.2 Grok won't find my view

Make sure you defined a view class which inherits from `grok.View`. Also ensure, that the context of the view you defined in your view class is correct. You can explicitly associate a view to a model by using `grok.context(my.model.Model)`. Another common pitfall is the naming: check if you are navigating to the view name which corresponds to the view class name, e.g.:

```
>>> class MyView(grok.View):
...     pass
>>> # wrong view name, which doesn't correspond to view class
>>> browser.open('http://localhost:8080/myapp/my_view.html')
>>> # correct view name
>>> browser.open('http://localhost:8080/myapp/myview.html')
```

3.4 Documentation and Further Reading

Grok provides an excellent Tutorial on how to get started: <http://grok.zope.org/doc/current/tutorial.html>. Other references include the Zope wiki at <http://wiki.zope.org/zope3/Zope3Wiki> and the Grok IRC channel `#grok` on irc.freenode.net.

References

- [1] Grok Project Homepage, <http://grok.zope.org>
- [2] Grokproject PyPi Page, <http://pypi.python.org/pypi/grokproject>
- [3] zc.buildout Homepage, <http://pypi.python.org/pypi/zc.buildout/1.4.1>
- [4] Five Homepage, <http://codespeak.net/z3/five/>
- [5] Python Package Index, <http://pypi.python.org/pypi>