Pros and Cons of different approaches to web programming in Python [closed]

Asked 16 years, 3 months ago Modified 3 years, 5 months ago

Viewed 6k times

24

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 8 days ago.

Improve this question

I'd like to do some server-side scripting using Python. But I'm kind of lost with the number of ways to do that.

It starts with the do-it-yourself CGI approach and it seems to end with some pretty robust frameworks that would basically do all the job themselves. And a huge lot of stuff in between, like web.py, Pyroxide and Django.

- What are the **pros** and **cons** of the frameworks or approaches that *you've worked on*?

- What **trade-offs** are there?
- For **what kind of projects** they do well and for what they don't?

Edit: I haven't got much experience with web programing yet.

I would like to avoid the basic and tedious things like parsing the URL for parameters, etc.
On the other hand, while the video of blog created in 15 minutes with Ruby on Rails left me impressed, I realized that there were hundreds of things hidden from me - which is cool if you need to write a working `webapp` in no time, but not that great for really understanding the magic - and that's what I seek now.

python **frameworks** **cgi** **wsgi**

Share

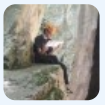
Improve this question

Follow

edited Jul 1, 2021 at 12:21

user15801675

asked Sep 4, 2008 at 13:00



Tomas Sedovic

44.3k ● 9 ● 41 ● 30

9 Answers

Sorted by: Highest score (default)

17

CGI is great for low-traffic websites, but it has some performance problems for anything else. This is because every time a request comes in, the server starts the CGI application in its own process. This is bad for two reasons: 1) Starting and stopping a process can take time and 2) you can't cache anything in memory. You can go with FastCGI, but I would argue that you'd be better off just writing a straight WSGI app if you're going to go that route (the way WSGI works really isn't a whole heck of a lot different from CGI).

Other than that, your choices are for the most part how much you want the framework to do. You can go with an all singing, all dancing framework like Django or Pylons. Or you can go with a mix-and-match approach (use something like CherryPy for the HTTP stuff, SQLAlchemy for the database stuff, paste for deployment, etc). I should also point out that most frameworks will also let you switch different components out for others, so these two approaches aren't necessarily mutually exclusive.

Personally, I dislike frameworks that do too much magic for me and prefer the mix-and-match technique, but I've been told that I'm also completely insane. :)

How much web programming experience do you have? If you're a beginner, I say go with Django. If you're more experienced, I say to play around with the different approaches and techniques until you find the right one.

Follow

answered Sep 4, 2008 at 13:35

Jason Baker
198k ● 138 ● 382 ● 520

12

The simplest web program is a CGI script, which is basically just a program whose standard output is redirected to the web browser making the request. In this approach, every page has its own executable file, which must be loaded and parsed on every request. This makes it really simple to get something up and running, but scales badly both in terms of performance and organization. So when I need a very dynamic page very quickly that won't grow into a larger system, I use a CGI script.

One step up from this is embedding your Python code in your HTML code, such as with PSP. I don't think many people use this nowadays, since modern template systems have made this pretty obsolete. I worked with PSP for awhile and found that it had basically the same organizational limits as CGI scripts (every page has its own file) plus some whitespace-related annoyances from trying to mix whitespace-ignorant HTML with whitespace-sensitive Python.

The next step up is very simple web frameworks such as web.py, which I've also used. Like CGI scripts, it's very simple to get something up and running, and you don't need any complex configuration or automatically generated code. Your own code will be pretty simple to understand, so you can see what's happening. However,

it's not as feature-rich as other web frameworks; last time I used it, there was no session tracking, so I had to roll my own. It also has "too much magic behavior" to quote Guido ("upvars(), bah").

Finally, you have feature-rich web frameworks such as Django. These will require a bit of work to get simple Hello World programs working, but every major one has a great, well-written tutorial (especially Django) to walk you through it. I highly recommend using one of these web frameworks for any real project because of the convenience and features and documentation, etc.

Ultimately you'll have to decide what you prefer. For example, frameworks all use template languages (special code/tags) to generate HTML files. Some of them such as Cheetah templates let you write arbitrary Python code so that you can do anything in a template. Others such as Django templates are more restrictive and force you to separate your presentation code from your program logic. It's all about what you personally prefer.

Another example is URL handling; some frameworks such as Django have you define the URLs in your application through regular expressions. Others such as CherryPy automatically map your functions to urls by your function names. Again, this is a personal preference.

I personally use a mix of web frameworks by using CherryPy for my web server stuff (form parameters, session handling, url mapping, etc) and Django for my object-relational mapping and templates. My

recommendation is to start with a high level web framework, work your way through its tutorial, then start on a small personal project. I've done this with all of the technologies I've mentioned and it's been really beneficial. Eventually you'll get a feel for what you prefer and become a better web programmer (and a better programmer in general) in the process.

Share Improve this answer Follow

edited Mar 25, 2009 at 13:21

answered Sep 4, 2008 at 14:11

Eli Courtwright

193k ● 68 ● 218 ● 257

7

If you decide to go with a framework that is WSGI-based (for instance TurboGears), I would recommend you go through the excellent article Another Do-It-Yourself Framework by Ian Bicking.

In the article, he builds a simple web application framework from scratch.

Also, check out the video Creating a web framework with WSGI by Kevin Dangoor. Dangoor is the founder of the TurboGears project.

Share Improve this answer Follow

edited Mar 24, 2009 at 22:41

answered Sep 4, 2008 at 13:24

codeape

101k ● 26 ● 174 ● 200

4

If you want to go big, choose Django and you are set. But if you want just to learn, roll your own framework using already mentioned WebOb - this can be really fun and I am sure you'll learn much more (plus you can use components you like: template system, url dispatcher, database layer, sessions, et caetera).

In last 2 years I built few large sites using Django and all I can say, Django will fill 80% of your needs in 20% of time. Remaining 20% of work will take 80% of the time, no matter which framework you'd use.

Share Improve this answer

Follow

answered Sep 17, 2008 at 20:04

zgoda

12.9k ● 4 ● 39 ● 46

3

It's always worth doing something the hard way - once - as a learning exercise. Once you understand how it works, pick a framework that suits your application, and use that. You don't need to reinvent the wheel once you understand angular velocity. :-)

It's also worth making sure that you have a fairly robust understanding of the programming language behind the framework *before* you jump in -- trying to learn both Django and Python at the same time (or Ruby and Rails,

or X and Y), can lead to even more confusion. Write some code in the language first, then add the framework.

We learn to develop, not by using tools, but by solving problems. Run into a few walls, climb over, and find some higher walls!

Share Improve this answer

Follow

answered Oct 21, 2008 at 14:49

hmason

603 ● 3 ● 8 ● 12

2

If you've never done any CGI programming before I think it would be worth doing one project - perhaps just a sample play site just for yourself - using the DIY approach. You'll learn a lot more about how all the various parts work than you would by using a framework. This will help in you design and debug and so on all your future web applications however you write them.

Personally I now use Django. The real benefit is very fast application deployment. The object relational mapping gets things moving fast and the template library is a joy to use. Also the admin interface gives you basic CRUD screens for all your objects so you don't need to write any of the "boring" stuff.

The downside of using an ORM based solution is that if you do want to handcraft some SQL, say for performance reasons, it much harder than it would have been otherwise, although still very possible.

Share Improve this answer Follow

edited Sep 4, 2008 at 13:32

answered Sep 4, 2008 at 13:17

David Webb

193k ● 57 ● 316 ● 302

2

If you are using Python you should *not* start with CGI, instead start with WSGI (and you can use wsgiref.handlers.CGIHandler to run your WSGI script as a CGI script. The result is something that is basically as low-level as CGI (which might be useful in an educational sense, but will also be somewhat annoying), but without having to write to an entirely outdated interface (and binding your application to a single process model).

If you want a less annoying, but similarly low-level interface, using WebOb would provide that. You would be implementing *all* the logic, and there will be few dark corners that you won't understand, but you won't have to spend time figuring out how to parse HTTP dates (they are weird!) or parse POST bodies. I write applications this way (without any other framework) and it is entirely workable. As a beginner, I'd advise this if you were interested in understanding what frameworks do, because it is inevitable you will be writing your own mini framework. OTOH, a real framework will probably teach you good practices of application design and structure. To be a really good web programmer, I believe you need to try both seriously; you should understand everything a

framework does and not be afraid of its internals, but you should also spend time in a thoughtful environment someone else designed (i.e., an existing framework) and understand how that structure helps you.

Share Improve this answer

Follow

answered Jul 30, 2009 at 21:44

ianb

899 ● 7 ● 7

1

OK, rails is actually pretty good, but there is just a little bit too much magic going on in there (from the Ruby world I would much prefer merb to rails). I personally use Pylons, and am pretty darn happy. I'd say (compared to django), that pylons allows you to interchange ints internal parts easier than django does. The downside is that you will have to write more stuff all by youself (like the basic CRUD).

Pros of using a framework:

1. get stuff done quickly (and I mean lighning fast once you know the framework)
2. everything is compying to standards (which is probably not that easy to achieve when rolling your own)
3. easier to get something working (lots of tutorials) without reading gazillion articles and docs

Cons:

1. you learn less
2. harder to replace parts (not that much of an issue in pylons, more so with django)
3. harder to tweak some low-level stuff (like the above mentioned SQLs)

From that you can probably devise what they are good for :-) Since you get all the code it is possible to tweak it to fit even the most bizzare situations (pylons supposedly work on the Google app engine now...).

Share Improve this answer

Follow

answered Sep 4, 2008 at 19:55

Bartosz Radaczyński

18.6k ● 14 ● 56 ● 61

1

For smaller projects, rolling your own is fairly easy. Especially as you can simply import a templating engine like Genshi and get alot happening quite quickly and easily. Sometimes it's just quicker to use a screwdriver than to go looking for the power drill.

Full blown frameworks provide alot more power, but do have to be installed and setup first before you can leverage that power. For larger projects, this is a negligible concern, but for smaller projects this might wind up taking most of your time - especially if the framework is unfamiliar.

Share Improve this answer

Follow

answered Dec 9, 2009 at 13:02

Adam Luchjenbroers
5,019 2 32 37