常见问题

本页回答了一些关于 Jinja 的常见问题。

为什么叫做 Jinja？

选择 Jinja 作为名字是因为 Jinja 是日本寺庙的名称，并且 temple 和 template 的发音类似。它并不是以乌干达的金贾市（Jinja）命名的。

它有多快?

我们相当厌烦基准测试，尤其是因为它们并不能影响什么。一个模板的性能取决于许多因素，而你可能需要在不同环境中对不同的引擎做基准测试。测试套件中的基准测试表明， Jinja2 与 Mako 的性能相近，比 Django 的模板引擎或 Genshi 快 10 到 20 倍。这些数字应该相当有刺激性，因为基准测试只测试一些性能相关的场景，比如循环，来获取这些数字。大体上，一个模板引擎的性能几乎不会成为一个 web 应用的瓶颈，而应该是数据库或应用的代码。

Jinja2 与 Django 兼容性如何?

Jinja2 的语法与 Django 的语法很多都匹配。但这并不意味着你可以直接在 Jinja2 中原封不动地使用 Django 模板。例如过滤器参数使用函数调用语法而不是用冒号分隔过滤器名和参数。此外， Jinja 中的扩展接口与 Django 的有根本区别，这意味着你的自定义标签不能再正常工作。

总体而言，因为 Jinja 模板系统允许你使用一个 Python 表达式的特定子集，你会使用相当少的自定义扩展，且可以替代大多数的 Django 扩展。例如，不是用下面的这样：

```
{% load comments %}
{% get_latest_comments 10 as latest_comments %}
{% for comment in latest_comments %}
    ...
{% endfor %}
```

你会更可能提供一个对象，用属性来检索数据库中的评论：

```
{% for comment in models.comments.latest(10) %}
    ...
{% endfor %}
```

或直接为快速测试提供模型：

```
{% for comment in Comment.objects.order_by('-pub_date')[:10] %}
    ...
{% endfor %}
```

请记住即使你能在模板中放置这样的东西，这也不是一个好主意。查询应该在视图代码中执行，而不是模板中！

把逻辑放在模板里是不是个可怕的主意?

毫无疑问，你应该尽可能把逻辑从模板中移除。但没有逻辑的模板意味着你不得不在代码中做所有的处理，而这是无趣和愚蠢的。一个如此做的支持 Python 的模板引擎名为 *string.Template* ，它没有循环和 if 条件，且是迄今

为止你可以在 Python 中使用的最快的模板引擎。

所以模板中某种数量的逻辑会让你愉悦。而且 Jinja 把要在模板中放置多少逻辑的问题几乎都留给你了。这里有一些你可以做和不可以做的限制。

Jinaja2 不允许你在模板中放置任意的 Python 代码，也不允许全部的 Python 表达式。操作符限定为最常用的那些，且不支持诸如列表推导式和生成器表达式等高级表达式。这使得模板引擎易于维护，并且模板有更好的可读性。

为什么自动转义不默认开启?

之所以自动转义不是默认的模式，且不是推荐的模式有许多原因。自动转义变量意味着你不再会面临 XSS 问题，它也导致在模板引擎中导致巨大规模的额外处理和严重的性能问题。因为 Python 不提供标记字符串为不安全的方式，Jinja 提供了一个自定义的字符串类（ **Markup** 字符串）hack 这个限制来与安全和不安全的字符串交互。

在显式的转义中，无论如何模板引擎都不需要运行对变量的任何安全检查。同样，一个自然人知道不去转义永远不会包含需要转义的或先前的 HTML 标记的整数和字符串。例如，当在一个整数和浮点数的表中迭代时，根据表的状况，模板设计者可以略过转义，因为他知道整数和浮点数不包含任何不安全的参数。

此外， Jinja2 是一个通用模板引擎，且不只用于 HTML/XML 生成。例如你会生成 LaTeX 、 邮件 、 CSS 、 JavaScript 或是配置文件。

为什么上下文是不可修改的?

当写 **contextfunction()** 或类似的东西时，你可能会注意到上下文试图阻止你修改它。如果你已经设法用内部的上

下文 API 修改了上下文，你会注意到上下文上发生的修改在模板中是不可见的。这个的原因是 Jinja 为性能因素，只把上下文作为模板变量的主要数据源。

如果你想要修改上下文，写一个函数返回一个变量，并用 set 赋值：

```
{% set comments = get_latest_comments() %}
```

加速模块是什么和为什么缺失它?

要用在自动转义开启的情况下实现高性能，在老版本的 Jinja2 中转义函数也需要用纯 C 实现，如果 Jinja2 与加速模块一同安装也会使用它。

因为这个特性本身对非模板引擎也很有用，它被移动到 PyPI 上的一个独立的项目，名为 MarkupSafe 。

Jinja2 不再伴随一个 C 实现而是纯粹的 Python 实现。无论如何，它会检查是否存在安装好的 MarkupSafe ，如果有，使用 MarkupSafe 的 Markup 类。

所以如果你想加速，只需要导入 MarkupSafe 。

我的回溯看起来很怪异。发生了什么?

如果没有编译调试支持模块并且你在使用一个没有 ctypes 的 Python 安装（ Python 2.4 没有 ctypes ， Jython 或 Google 的 AppEngine ） Jinja2 不能提供正确的调试信息，且回溯可能是不完整的。这也是为什么现在不能与 Jython 或是 AppEngine 良好工作的原因，没有 ctypes 就不可能使用调试支持扩展。

如果你在 Google Appengine 开发服务器上工作，你可以把 ctypes 模块添加到白名单来恢复回溯。这无论如何都不

能用于生产环境:

```
import os
if os.environ.get('SERVER_SOFTWARE', '').startswith('Dev'):
    from google.appengine.tools.dev_appserver import HardenedModulesHook
    HardenedModulesHook._WHITE_LIST_C_MODULES += ['_ctypes', 'gestalt']
```

这个片段的声誉见 Thomas Johansson

为什么没有 Python 2.3 支持?

Python 2.3 中缺少许多在 Jinja2 中大量使用的特性。做出这个艰难的决定是因为使用即将到来的 Python 2.6 和 3.0 版本，再为老版本 Python 维护代码更加困难。如果你确实需要 Python 2.3 支持，你可以使用 Jinja 1 或其它仍然支持 2.3 的模板引擎。

我的宏被什么东西给覆盖了

在某些情况下， Jinja 的作用域会是任意的:

layout.tmpl:

```
{% macro foo() %}LAYOUT{% endmacro %}
{% block body %}{% endblock %}
```

child.tmpl:

```
{% extends 'layout.tmpl' %}
{% macro foo() %}CHILD{% endmacro %}
{% block body %}{{ foo() }}{% endblock %}
```

这在 Jinja2 中会打印 `LAYOUT` 。这是父模板在子模板之后求值的副作用。这允许子模板传递信息到父模板。要避免这个问题，重命名父模板中的变量或宏让它们有不同的前缀。