**一个简便的分析平台，用于可视化不断发展的基于Flask的Python Web服务**

**Patrick Vogel, Thijs Klooster, Vasilios Andrikopoulos, Mircea Lungu**

Johann Bernoulli Institute for Mathematics and Computer Science

University of Groningen, Netherland

**摘要：**

Flask是一种基于Python的web框架，成千上万的web应用程序是用Flask编写的。尽管flask有丰富的扩展生态系统，但是没有一个扩展是支持开发人员洞察他们开发的web应用程序不断演变的性能。在本文中，我们介绍了Flask Dashboard，一个解决这个问题的flask扩展库。我们展示了该flask扩展库可以轻松集成到现有的web应用程序中，讨论了该扩展库提供的一些可视化视角并指出类似的扩展库的在未来面临的一些挑战。

1. **导言**

就像最近的一篇文章[1]所说，构建一个分布式的系统是一个不可避免的事情。事实上，即使是大学二年级学生最简单的项目也是一个由两层架构实现的web应用程序，前端是Javascript/HTML5架构，后端服务通常是REST API架构。

Python是实现web应用程序后端的最流行的编程语言之一。GitHub包含超过50万个开源Python项目并且Tiobe排名[[1]](#footnote-1)截至2016年6月，Python在最受欢迎的编程语言中排名第四。

在Python社区中，Flask[[2]](#footnote-2)是一个非常流行的web框架[[3]](#footnote-3)。它通过实现一个最低限度的web服务器提供了简单性和灵活性，因此经常被宣传为一个微服务框架。Flask教程展示了如何通过不超过5行的Python代码[2]建立一个简单的Flask“Hello World”的web服务。

尽管flask很受欢迎，但据我们所知，目前没有简单的解决方案来监控Flask web应用程序不断演变的性能。因此，这些项目的每一个开发人员在需要收集他们开发的web服务的运行状态的信息时都将面临以下选择：

1. 使用一个将目标API当作黑盒状态的商业监控程序 (例如Pingdom、Runscope)。
2. 实施他们自己特定的分析解决方案，不得不重新实现基本的可视化和交互策略。
3. 放弃对他们服务进行深入分析

对于有预算的项目(如研究、创业)，由于时间和资金限制，第一个和第二个选项通常不会被采用。即使在使用第三方的分析解决方案时，也缺少对web应用程序提供的服务的演变过程的批判性洞察力，因为这种解决方案没有版本控制的概念，也没有与开发生命周期进行集成。[3]

在本文中，为了避免项目选择第三种方案，即放弃分析活动，我们介绍了Flask Dashboard，这是一个基于Flask框架的Python web服务的低工作量服务的监控扩展库，它易于集成并支持对服务演变进行敏捷地评估。[4]

作为一个案例研究，我们将说明我们的解决方案，我们将使用一个开源的API，它在上述第三种情况下使用了一年多。

1. **案例研究**

Zeeguu[[4]](#footnote-4)是一个加速外语词汇学习的平台和应用生态系统[5]。生态系统架构的核心是一个用Python的Flask框架实现的API和一系列附属应用程序，它们共同为学习者提供三个主要的相互关联的功能:

1. 为那些对读者来说太难的文本提供便利翻译功能的阅读程序。
2. 基于学习者的偏好和喜好进行个性化地互动练习
3. 为读者推荐适当难度的文章。文章的难度是根据读者过去的练习和阅读活动进行评估的。

核心API相应地提供了三种类型的功能：上下文翻译、文章推荐和个性化练习建议。系统的核心API是一个研究项目，在撰写本文的时候，它支撑着大约200个活跃的Beta测试帐户进行阅读和实践。

在本文的剩余部分，我们将使用Zeeguu API作为案例研究。本文的所有数据都来自Zeeguu API[[5]](#footnote-5)环境中实际部署的Flask DashBoard获取的数据。

1. **Flask Dashboard 介绍**

在本文中，我们将介绍Flask Dashboard，这是一个嵌入式Python扩展库，允许开发人员通过最小的工作量监控他们基于Flask框架的Python web应用程序。

Flask Dashboard和案例研究中被监控的web应用程序使用的Flask框架是用Python编写的。这使得绑定到应用程序的web服务变得相对容易，也为服务添加了额外的路由，以便与Flask Dashboard进行交互。

假设我们已经安装了Flask，为了开始使用我们的Python扩展库，那么安装需要的Python包[[6]](#footnote-6)只需在Flask web服务中添加两行代码:

|  |
| --- |
| **import** dashboard  ...  # flask\_app is the Flask app object  dashboard.bind(flask\_app) ... |

在绑定到服务后，Flask Dashboard扩展库在Flask应用程序的/dashboard路径上便可以使用了。程序员也可以在配置文件中自定义路由。

在绑定期间，Flask Dashboard将搜索目标应用程序中的所有端点。这些端点将在工具web界面中呈现给用户，用户可以在该界面中选择应该监控哪些端点，参见图1。

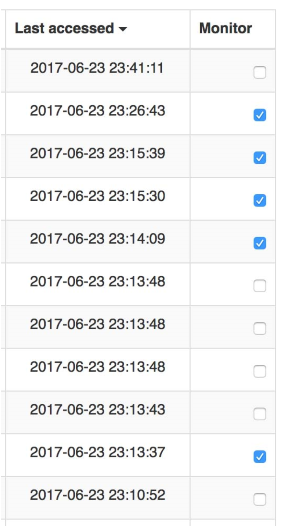
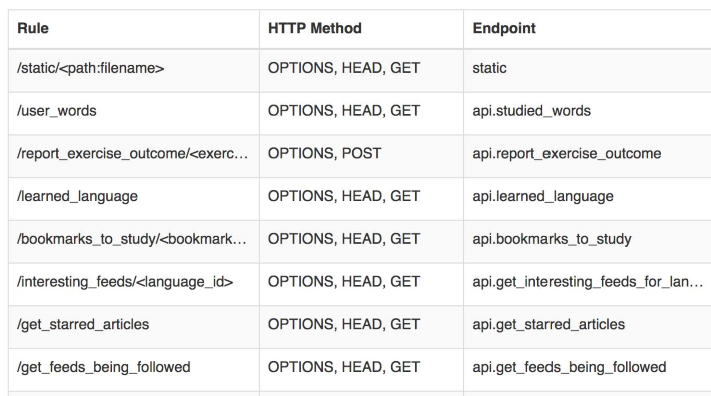


图1.Zeeguu应用程序的所有端点都会显示出来，这样就可以选择对它们进行监控

为了监控一个端点，Flask Dashboard为对应于该端点的API函数创建一个装饰器函数。这样，无论何时API被调用，装饰器函数都将被执行。装饰器函数包含负责监控端点的代码。装饰器函数收集的数据保存在本地数据库中。

使用Flask Dashboard可以获得三种主要类别的透视图:

1. 服务利用率提供了所有相关端点的使用信息，
2. 端点性能表示各种服务端点的响应时间，
3. 用户体验提供了关于服务端点的用户感知性能的信息。

在本文的剩余部分，我们将展示其中的几个视角。[[7]](#footnote-7)

1. **服务利用率**

服务维护者需要的最基本的洞察力是服务利用率。

图2展示了Flask Dashboard提供的关于端点利用率的第一个透视图:按天分组的各个端点的点击次数的堆积条形图[[8]](#footnote-8)。图2特别展示了API在峰值时每天大约有12000次点击。用户与平台交互的方式也可以推断出来，因为端点是不同活动类型的指示器，例如:

1. api.get\_possible\_translations是用户正在进行的外语阅读量的指标，并且
2. api.report\_exercise\_outcome是用户正在进行的外语词汇练习量的指标。

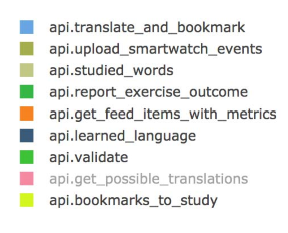
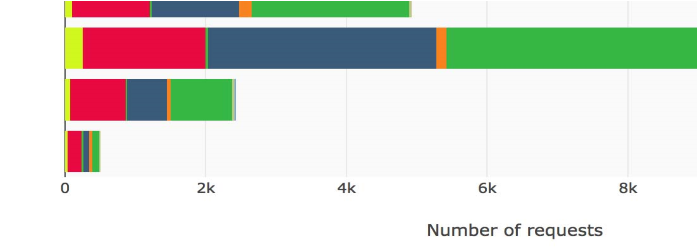
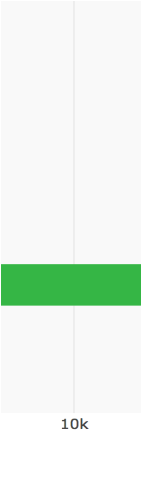
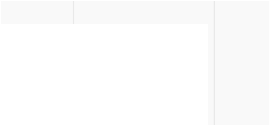
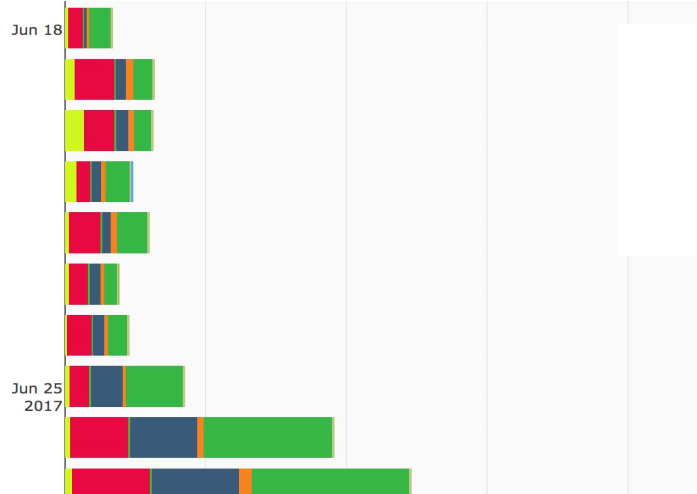


图2.每天每个端的请求数视图展示了受监控应用程序的总体利用率

除了显示整体利用率，这个端点还为维护人员提供了与端点弃用决策相关的信息——这是理解下游需求的最基本的方式之一[7]。在我们的案例研究中，维护人员发现他们认为没有使用的一个端点(即words\_to\_study)实际上正在被使用[[9]](#footnote-9)，这和之前他们的预期恰恰相反。

Flask Dashboard可以自动回答的第二种类型的利用问题是借助于热力图显示一天中每小时的循环使用模式，如图3所示。

图3使用模式在每小时请求热力图中变得很容易发现

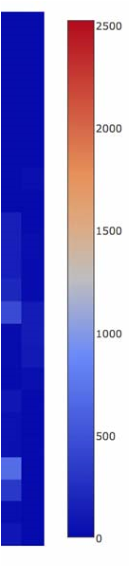
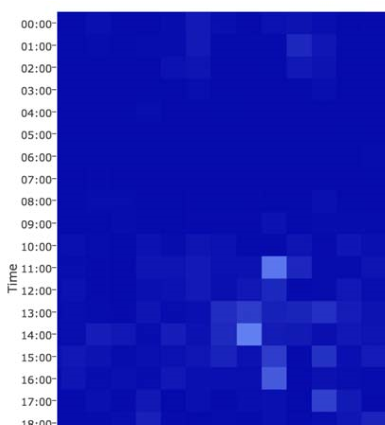


图3显示了在清晨时段API没有被使用，大部分使用活动集中在工作时间同时晚上有少量使用活动。这与当前用户都位于欧洲中部时区的事实是一致的。此外，该图显示，在之前图片中也可以看到的利用率峰值出现在下午/晚上。

1. **端点性能**

Flask Dashboard还收集有关端点性能的信息。图4中的视图通过使用盒须图总结了各种端点的响应时间。

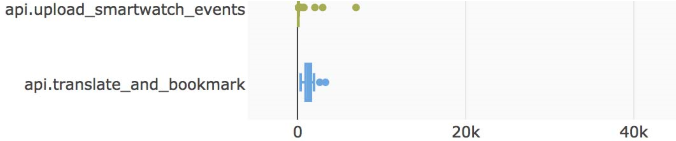
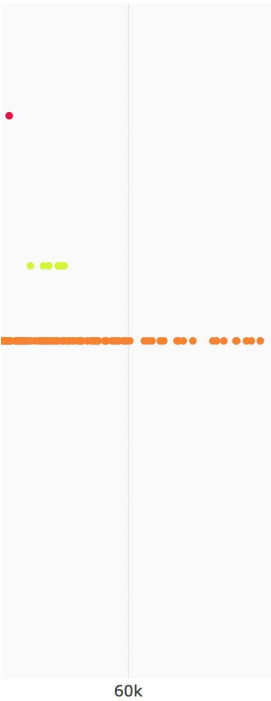


图4每个受监控端点的响应时间(毫秒)视图允许识别性能变化和平衡问题

在这个图片中维护人员可以清楚地发现四个端点在性能上有非常大的差别。对应用程序来说，最关键同时也是最先优化的是api.get\_possible\_translations端点，它是依赖于Zeeguu API的reader应用程序中交互循环的一部分。此外，如图2展示的，该端点是系统中被使用最多的几个端点之一。

然而，根据工具的当前配置，维护者不可能看到优化带来的改进。一种方法是添加一行额外的配置文件来允许Flask Dashboard找到已部署服务的git[[10]](#footnote-10)文件夹:

dashboard.config.git = 'path/to/.git’

有了这个额外的配置，Flask Dashboard现在可以自动检测项目的版本，并按版本对测量结果进行分组[[11]](#footnote-11)。图5是api.get\_possible\_translations端点的视图的放大版本，纵坐标按照版本顺序从上到下递增



图5可视化api.get\_possible\_translations端点的性能演变

该视图证实了api.get\_possible\_translations端点的性能在最近的版本中得到改善:在最近三个版本的端点的性能中值不断地向左移动，并且从1.4秒(图5中最上面的方框中)缩短到最新版本中的0.8秒(图5中最下面的方框)。

Flask仪表板收集这些异常值的额外信息:Python堆栈跟踪、CPU负载、请求参数等。以便让维护人员调查导致这些异常缓慢的响应的原因。

为了解决这个问题同时不降低整体性能，Flask Dashboard跟踪每个端点的平均响应时间。当它检测到给定的请求的响应时间和过去平均响应时间相比出现异常的时候，它触发异常值数据收集进程，该进程会储存所有之前列出的关于当前执行环境的额外信息。

1. **用户体验**

对于进行实时计算的服务端点，系统维护人员希望了解端点在每一个用户使用下的响应性能，尤其是当系统响应时间是某个用户的负载函数[[12]](#footnote-12)的情况下。

要实现这一点，Flask Dashboard必须配置成将API调用与给定用户相关联的形式。最简单的方法是利用Flask应用程序的架构，在该架构中，可以使用全局flask.request对象来检索会话，从而查找到每次请求时的用户标识。

|  |
| --- |
| # app特定的提取用户的方式  # from a flask请求对象定义get\_user\_id():  sid = int(flask . request . args[' session '])session = user . find \_ for \_ session(sid)返回user\_id  #附加获取用户标识函数dashboard.config.get\_group\_by =获取用户标识 |

在Zeeguu中，api\_get\_feed\_items\_with\_metrics端点检索特定用户的推荐文章列表。参见图4，它是具有最慢响应时间和最高可变性的端点。这样做的原因是用户可以订阅一到三十多个文章源，对于每个源，系统必须在每次请求时计算每个文章的个性化推荐的难度。

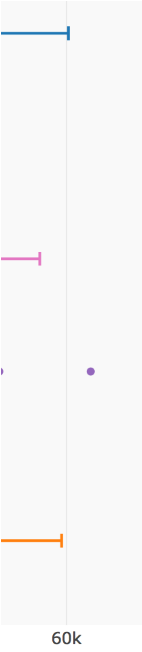
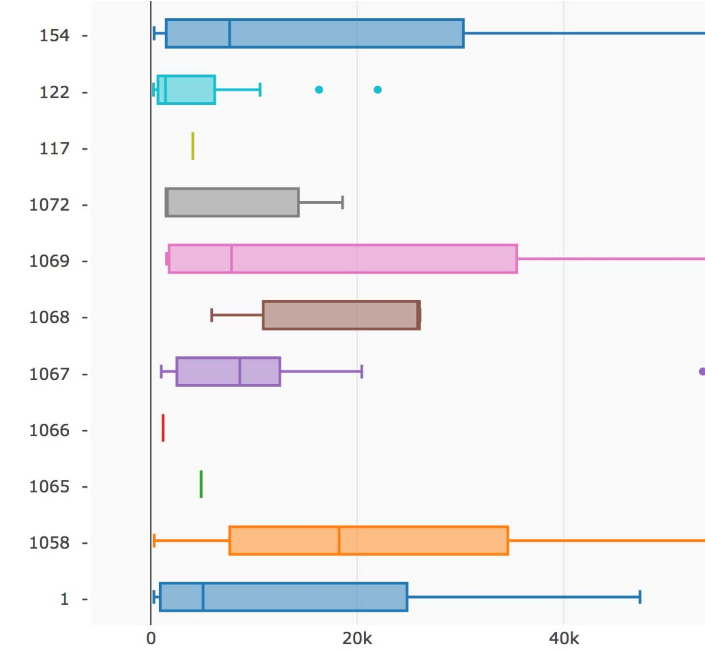


图6。API . get \_ feed \_ items \_ with \_ metrics显示了非常高的用户之间的差异

每个用户的性能展示应该显示不同用户的不同响应时间。图6显示了Flask Dashboard中对应的视图。该图显示，对于不同的用户，该端点的响应时间会有很大的不同，在某些极端情况下，用户必须等待一分钟，才能看到推荐给他们的文章[[13]](#footnote-13)。

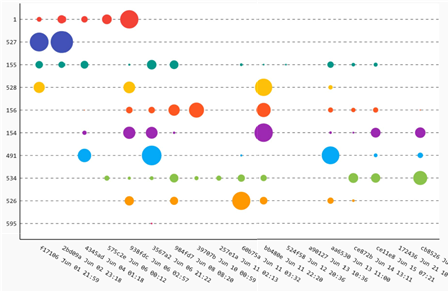
前一个视图的局限性在于它不能按照版本顺序显示信息。为了解决这个问题，可以定义一个多版本用户分析的图表。图7通过将给定用户的平均执行时间(行)和给定版本的平均执行时间(列)映射到相应的圆形区域上展示了这样的图表。图表上的不同颜色代表不同的用户。该图显示了不同用户和版本之间的平均性能差异，没有明确的趋势:这可能是因为不同的用户工作负载(即用户使用的资源数量)不同，因而响应时间有所差异。

图7。这个透视图显示了不同版本(x轴)的单个用户响应时间的演变(水平线)

1. **讨论**

A.自动化监控系统发展

Flask Dashboard的主要设计目标是可以通过对正在运行的API进行尽可能小的更改来收集分析数据和获得深入了解的能力。该技术假设作为监控目标的web应用程序是使用git以下列方式部署的:

1. 部署工程师从服务器上获取最新版本的代码；这将导致头指针指向新的提交
2. 部署工程师重新启动新版本的web应用服务。此时，Flask Dashboard检测到本地代码库中出现了一个新的头部，因此开始将所有新的数据点与这个新的提交相关联 。[[14]](#footnote-14)

这种方法的优点是只需要很少量的配置工作，正如上文所展示的那样。缺点是它在同等基础上出现改动便会提交，甚至仅仅是修改注释也会提交，以及最短生命周期的提交，例如在更新具有错误修复的新版本之前每次提交只会活跃半个小时，同时软件的主要和次要版本都是如此。因此，需要添加一种机制来控制哪些版本对于监控目的来说是重要的。这里另一个可能的扩展点是支持其他版本控制系统(例如Mercurial)。然而，这只是一个简单的扩展。

B.用户意识

对于用户信息不可用的情况，Flask Dashboard默认跟踪关于不同IP的信息，并且在某些情况下，这和用户的实际身份和呈现出的多样性十分相似。

本文第六节中介绍的用户体验视角的可视化已经在数百名用户中进行了测试(在这其中约有200名用户在研究过程中处于活跃状态)，但是可视化的可扩展性需要对拥有数万名用户的web服务进行进一步深入调查。

C.其他可能的分组

对于维护者来说，还有其他对服务利用率和性能很重要的分组在本文中没有探讨。例如，如果web服务使用OAuth，那么在每一个请求的报头中都有关于发送请求的应用程序的信息。在这种情况下，按发送请求的应用程序的不同对信息进行分组可能很重要。

一般来说，需要提供一种机制允许非常容易地指定分组(作为代码注释、作为普通代码或作为配置选项)，这是Flask Dashboard和任何其他类似扩展库必须面对的一个公开问题。

1. **相关著作**

使用可视化来深入了解软件性能有着悠久的传统。像Jinsight [8]和Web Services Navigator [9]这样的工具为Java和与SOAP消息通信的Web应用服务开创了这样的方法。两者都对服务/对象及其交互有一个“全面”的视角。与他们相反，在我们的工作中，我们提出了一个分析平台，它专注于从单个Python web服务自身的角度来监控它。

从服务监控的角度来看，我们的工作属于服务的服务器端运行时监控[10]。虽然我们没有实现相关监控解决方案的更高级的功能，如驱动监控的QoS策略，但它提供了一种易于使用的方法来帮助提高web应用程序的性能。

1. **结论和未来的工作**

在本文中，我们已经表明开发人员只需付出很少的努力，就可以创建一个监控解决方案，提供对web服务利用率和性能的基本了解。我们这项工作的目标用户群是使用Flask和Python构建web应用程序的应用程序开发人员，他们实现自己的监控解决方案的预算有限或没有预算。工作重点是让这些用户了解服务的性能是如何随着应用程序本身一起发展的。我们相信同样的架构和经验可以应用于其他的开发框架和其他计算机语言。

在不远的将来，我们计划对其他系统进行案例研究，目的是发现其他需求同时淘汰Flask DashBoard中不太有用的可视化功能。我们还计划扩展该工具，以支持跨多个节点的相同应用程序的多种部署(例如，应用程序与负载均衡器一起部署的情况)。最后，我们计划将Flask Dashboard与单元测试集成起来，对应用程序的性能发展提供相关信息。

1. **工具和源代码的可用性**

本文中的图片是Zeeguu API环境中Flask Dashboard部署的交互式可视化图表的截图。该系统的部署可由本文的读者和审稿人在线查阅[[15]](#footnote-15)。

Flask Dashboard是由Python 3.6实现的，可从Python包索引存储库中获得[[16]](#footnote-16)。零和一个拥有Python的计算机通过可以在命令行中运行“pip install flask\_monitoring\_dashboard ”的命令来安装扩展库

Flask Dashboard的源代码是在许可的MIT许可下发布的，可以在GitHub[[17]](#footnote-17)上获得。

**参考文献**

[1] M. Cavage, “There is no getting around it: you are building a distributed system,” Communications of the ACM, vol. 56, no. 6, pp. 63–70, 2013.

[2] A. Ronacher, “Quickstart,” http://flask.pocoo.org/docs/0.12/quickstart/ #quickstart, 2010, [Online; accessed 25-June-2017].

[3] M. Papazoglou, V. Andrikopoulos, and S. Benbernou, “Managing evolving services,” Software, vol. 28, no. 3, pp. 49–55, 2011.

[4] O. Nierstrasz and M. Lungu, “Agile software assessment,” in Proceedings of International Conference on Program Comprehension (ICPC 2012), 2012, pp. 3–10. [Online]. Available: http://scg.unibe.ch/ archive/papers/Nier12bASA.pdf

[5] M. F. Lungu, “Bootstrapping an ubiquitous monitoring ecosystem for accelerating vocabulary acquisition,” in Proccedings of the 10th European Conference on Software Architecture Workshops, ser. ECSAW 2016. New York, NY, USA: ACM, 2016, pp. 28:1–28:4. [Online].

Available: http://doi.acm.org/10.1145/2993412.3003389

[6] B. Shneiderman, “The eyes have it: A task by data type taxonomy for information visualizations,” in IEEE Visual Languages, College Park, Maryland 20742, U.S.A., 1996, pp. 336–343.

[7] N. Haenni, M. Lungu, N. Schwarz, and O. Nierstrasz, “A quantitative analysis of developer information needs in software ecosystems,” in Proceedings of the 2nd Workshop on Ecosystem Architectures (WEA’14), 2014, pp. 1–6. [Online]. Available:http://scg.unibe.ch/archive/papers/Haen14aQuantitativeEcosystemInformationNeeds.pdf

[8] W. De Pauw, E. Jensen, N. Mitchell, G. Sevitsky, J. M. Vlissides, and J. Yang, “Visualizing the execution of java programs,” in Revised Lectures on Software Visualization, International Seminar. London, UK: Springer-Verlag, 2002, pp. 151–162.

[9] W. D. Pauw, M. Lei, E. Pring, L. Villard, M. Arnold, and J. F. Morar, “Web services navigator: Visualizing the execution of web services,” IBM Systems Journal, vol. 44, no. 4, pp. 821–845, 2005.

[10] C. Ghezzi and S. Guinea, “Run-time monitoring in service-oriented architectures,” in Test and analysis of web services. Springer, 2007, pp. 237–264.

1. TIOBE 编程社区排名是一个衡量编程语言流行程度的重要衡量标准，这个由TIBOE公司创造和维护，公司地址在Eindhoven，Netherland。 [↑](#footnote-ref-1)
2. http://flask.pocoo.org/ [↑](#footnote-ref-2)
3. 在Github网站上有超过两万五千项目（占所有Python项目的5%）是基于Flask完成的（比如在GitHub上搜索“language:Python Flask”） [↑](#footnote-ref-3)
4. https://www.zeeguu.unibe.ch/ [↑](#footnote-ref-4)
5. 在Flask Dashboard 中每一个图片都有着基本的交互方法比如筛选，展示详细信息放大等，可以根据需要进行使用[6] [↑](#footnote-ref-5)
6. 本文第十节展示了如何安装扩展库 [↑](#footnote-ref-6)
7. 我们建议使用本文中所采用的配色方案以获得更好的视觉效果 [↑](#footnote-ref-7)
8. 每个端点的颜色在不同的图片中是一样的 [↑](#footnote-ref-8)
9. 在图1所示的视图中还可以发现一种补充的使用信息，在其中看到一个端点从未被访问，这可以增加维护者的信心，即一个给定的端点没有被使用，尽管这永远不能被用作证据。 [↑](#footnote-ref-9)
10. https://git-scm.com/ [↑](#footnote-ref-10)
11. 或者，如果web应用系统不使用git，维护人员可以通过配置文件为web应用程序手动添加版本标识符。 [↑](#footnote-ref-11)
12. 例如，在GMail中，一些用户有两封电子邮件，而另一些用户有两万封，这导致不同用户的响应时间不同 [↑](#footnote-ref-12)
13. 看到这种情况后，维护人员应该重构系统架构，将系统中计算复杂的部分移出了交互循环 [↑](#footnote-ref-13)
14. Flask Dashboard在被分析系统第一次附加到程序上时检测其当前版本，因此，假设Flask应用程序在部署新版本时自动重新启动。这与Flask的当前版本是一致的，但是如果web服务器将来支持动态更新，就必须考虑这一点。 [↑](#footnote-ref-14)
15. https://zeeguu.unibe.ch/api/dashboard; username: guest, password: vissoft [↑](#footnote-ref-15)
16. https://pypi.python.org/pypi/flask-monitoring-dashboard/1.8 [↑](#footnote-ref-16)
17. https://github.com/mircealungu/automatic-monitoring-dasboard [↑](#footnote-ref-17)