项目申请书

项目名称: 通过 Github Actions 实现标准流程自动化
项目主导师: 欧阳文
申请人: 樊漆亮
日期: 2021.05.24
邮箱: fanqiliang@torch-fan.site

---

1. 项目背景

1. 项目基本需求:

`issue仓库地址:` https://github.com/apache/shardingsphere/issues/9697

1. 文档自动发布:

目前ShardingSphere官网的文档始终与文档仓库的最新内容保持一致，当ShardingSphere主仓库的master分支中文档内容有更新时，会立即部署到ShardingSphere-doc仓库文档的current文件夹下，随后发布到官网。

目前这一行为存在一个问题，ShardingSphere通过tag发行版本，而文档是随时更新的，这样会导致软件操作与文档中的说明不一致，这也是本项目需要解决的第一个问题。

2. issue自动化管理:

希望通过GitHub Actions自动处理部分issue，例如超时未回复的issue就自动关闭、不符合规范的issue自动关闭、issue自动指派等。这一步，通过与导师交流，导师提供了 AntDesign 项目作为参考提示，其中的GitHub Actions较为完整，可作为issue自动化管理的参考和学习案例。

3. 当版本发布时自动发布docker镜像:

当ShardingSphere发布新的版本时，通过GitHub Actions构建Docker镜像并发布到Docker仓库中。

2. 项目相关仓库:

- ShardingSphere主仓库：https://github.com/apache/shardingsphere
- ShardingSphere-doc文档仓库：https://github.com/apache/shardingsphere-doc

通过ShardingSphere-doc仓库的 工作流 内容可以看到到这两个仓库之间的联系，工作流触发条件有三个：1. 每10分钟自动执行1次；2. 当代码推送到 `asf-site` 分支时执行一次；3. 当pull request的目标文分支为 `asf-site` 时执行一次。工作流的执行主要内容封装在一个中 shell脚本 中。

通过浏览shell脚本的内容，可以看到主要功能是将ShardingSphere主仓库 `master` 分支的文档通过git clone及文件操作拷贝到ShardingSphere-doc文档仓库的相应目录下, 推送到远程分支更新ShardingSphere-doc远程仓库的 `asf-site` 分支的内容。

2. 技术方法及可行性

1. GitHub Actions相关

此前，我因为对GitHub Actions感兴趣，自己写了一些关于GitHub Actions的博客： Torch-Fan

此外GitHub Actions提供了workflow语法相关的文档: Workflow语法

GitHub还提供了应用商店可以从中找到可供使用的Actions，可以极大的加速工作流构建进程: GitHub Actions Marketplace

以项目的第二个需求为例，issue自动化管理就可以使用GitHub Actions应用商店的 `issue helper`，可以极大的方便issue自动化管理工作流构建。而项目的第一个需求 `文档自动化构建`，包含了工作流和shell脚本，其中主要工作为shell脚本编写。

创建自定义Actions可以通过Node.js进行，常用Node.js编写endpoint用来做GitHub 徽标：qiliangfan。 如果现有的Actions不能满足需求，我可以自定义Actions以满足相应功能。

2. Docker相关

曾在华为杭州研究所实习接触过容器化技术，使用minikube搭建了istio平台的bookinfo微服务用以搭建testbed，向其中注入故障并进行异常检测算法的验证，因此有一定的docker操作经验。

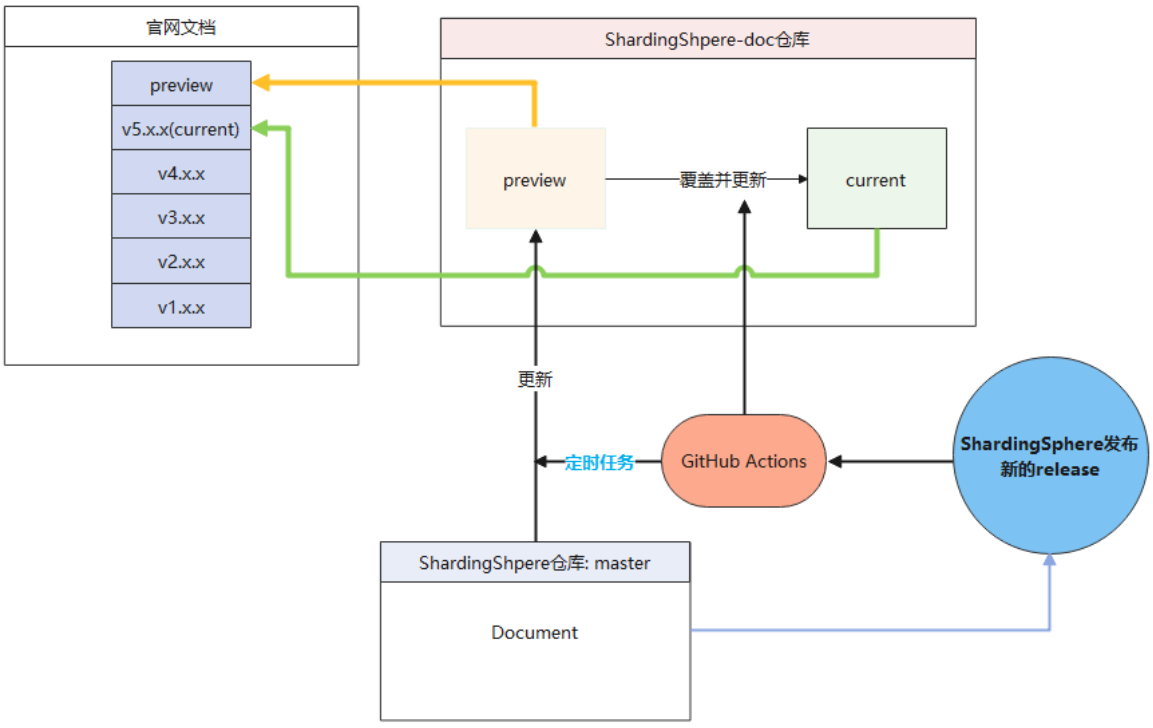
3. Linux相关

曾在商汤科技实习，有过一段Linux使用经验，且大二、大三将自己笔记本电脑装上Ubuntu18.04系统，有Linux的日常使用经验，基本可以胜任SHELL脚本相关的工作内容。

3. 项目实现细节梳理:

1. 文档自动发布:

通过前期的沟通，文档自动化发布要求官网有个预览的最新文档以及与最新ShardingSphere Release版本匹配的文档。因此文档自动化发布任务设计如下：



- 官网的文档列表新增一项`preview`，该项对应的文档随时保持更新, 当ShardingSphere `master`分支的文档内容有所更新会体现在该项内。与此同时，保留官网文档列表的`current`项，该项的文档内容与shardingSphere最新的release版本相对应。这样一来，即使`preview`中的文档不断更新，有`current`项的文档内容在，也不会存在文档的最新内容与ShardingSphere软件版本出现文档与软件不兼容的问题。
- 原先使用GitHub Actions的定时任务更新`ShardingSphere-doc`仓库的文档内容的工作流保留，但是修改更新的标文件夹将从`current`变为`preview`。以`preview`文件夹的内容作为最新、实时更新的文档内容。`current`作为与最新ShardingSphere的release版本对应的文档内容，用户在使用最新的ShardingSphere 的release时，可以从官网查看`current`文档内容。
- 当ShardingSphere 仓库创建tag发布新的release时，将触发GitHub Actions的CI/CD工作流 [1]，将ShardingSphere仓库`master`分支的文档内容同步到ShardingSphere-doc仓库的preview文件

夹，随后触发CI/CD工作流 [2]，将ShardingSphere-doc仓库的`preview`文件夹内容覆盖到`current`文件夹中，此时`preview`和`current`文件夹的文档内容和ShardingSphere的最新release版本对应。在下一个release发布前，所有的文档更新只在`preview`文件夹中同步。

2. issue标准处理流程：

> issue的管理目前有一个流行的Actions可供使用: issue-helper

issue自动化管理初期定为以下几项:

1. 超期未回复的issue自动关闭:

- 设置一个定时工作流，定期检查issue的活跃状态
- 若issue间隔N天没有新的回复，则为该issue添加label: `inactive`
- 设置一个定时工作流，定期检查有无指定label（在此任务中为`inactive`)的issue，若有，则将其关闭

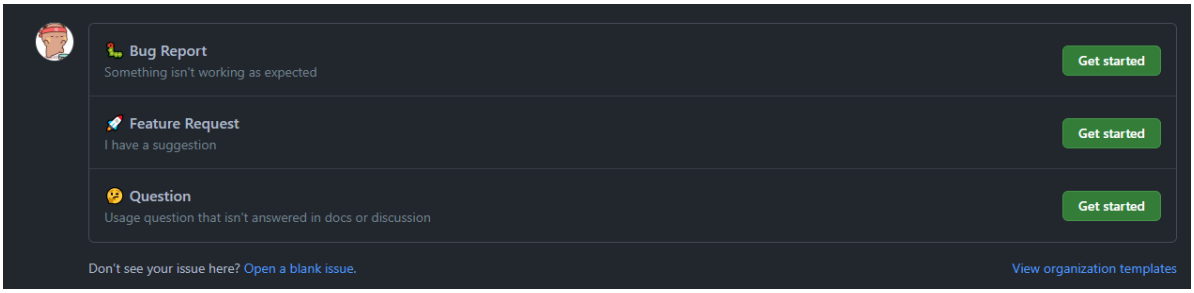
```yaml
name: Check inactive

on:
  schedule:
    - cron: "0 0 1 * *"

jobs:
  check-inactive:
    runs-on: ubuntu-latest
    steps:
      - name: check-inactive
        uses: actions-cool/issues-helper@v2.2.1
        with:
          actions: 'check-inactive'
          token: ${{ secrets.GITHUB_TOKEN }}
          inactive-day: 30
```

上面这个工作流模板, 每一天会对仓库中的issue进行检查, 超过`30`天没有新回复的issue会增加`inactive`标签。而此时只需要一个定时工作流，针对具有目标label的issue进行关闭即可。

2. 不合规范的issue自动关闭:



> ShardingSphere提供了三种issue模板，但是模板内容是文本形式的内容，如果要进行规范检查，这一步应当自定义一个GitHub Actions。

- ShardingSphere 的issue规范文档: https://shardingsphere.apache.org/community/cn/contribute/issue-conduct/

- 观察三类issue模板，可以发现一个共同特征：`三级标题给出了issue必须要具备的内容`，示例如下图。三级标题指明了: 接下来一段文字应该是具有一定描述内容的，随后才是下一个三级标题或者文本结束。

```
### Is your feature request related to a problem?

### Describe the feature you would like.
```

```
Please answer these questions before submitting your issue. Thanks!

### Which version of ShardingSphere did you use?

### Which project did you use? ShardingSphere-JDBC or ShardingSphere-Proxy?

### Expected behavior

### Actual behavior

### Reason analyze (If you can)

### Steps to reproduce the behavior, such as: SQL to execute, sharding rule configuration, when exception occur etc.

### Example codes for reproduce this issue (such as a github link).
```

因此, 在编写代码检查issue是否符合规范时，思路是根据这些`三级标题`--即issue中要求填写的内容进行检查。GitHub提供了Rest API， 可以对仓库中的issue内容进行获取，因而在使用Node.js编写自定义的GitHub Actions时，可以参照GitHub提供的Rest API文档发起请求，获取内容后，针对markdown中的三级标题对文本内容进行切分。最基本的规范检查目标是每个三级标题下都有相关内容。

接下来陈述自己对自定义GitHub Actions 来实现issue规范化的思路以及相关调查结果。前期调查中，在编写自定义的GitHub Actions一般会用以下三个库：`@actions/core`、`@actions/github`、`@octokit/rest`。 同时还找到了部分文档链接，这里是我的笔记：编写Actions常用库。 这其中`@octokit/rest`库作用主要是封装了GitHub的REST API，使得我们在调用各种繁琐API获取issue和issue的评论时，能够更加便捷。这里是GitHub官方的REST API文档 和 `@octokit/rest`库的文档。

譬如，通过如下调用API，我们可以获取一个issue的所有评论：

```
octokit.rest.issues.listComments({
  owner,
  repo,
  issue_number,
});
```

因而凭借着GitHub提供的Rest API接口，我们可以使用Node.js来自定义功能相对复杂的一些的Actions。

3. issue自动指派:

```
name: Add Assigness

on:
  issues:
    types: [opened]
```

```yaml
jobs:
  add-assigness:
    runs-on: ubuntu-latest
    steps:
      - name: Add assigness
        uses: actions-cool/issues-helper@v2.2.1
        with:
          actions: 'add-assignees'
          token: ${{ secrets.GITHUB_TOKEN }}
          issue-number: ${{ github.event.issue.number }}
          assignees: 'user1,user2'
          random-to: 1
```

该工作流文件的效果是为一个刚打开的issue随机增加一个受托者，这个受托者列表可通过进一步沟通初步确定，了解主要负责issue委派的人员列表。

issue自动委派也可以做成这种效果：定期地，根据issue的label来指派相应的受托者，而这也需要进一步沟通相关label和人员的关系。

若有额外需求，我们也可以为此自定义一个GitHub Actions来满足更多需要。

4. issue内容检查：

- 自动回复评论
- 自动为issue添加label

针对issue标题或者issue的主体中， 如果出现一些常见关键词， 譬如某一类BUG的出现常常与【IE】这个单词相关联， 那么在标题或者issue的主体内容中出现【IE】这个词时，通过编写相应的工作流，自动回复该issue相关内容。

抑或是，如果issue中存在某些关键词，则为该issue添加某些label。这一功能的实现思路主要是获取issue的title和body内容，如上文所说，由于GitHub提供了相关API这并不困难——GitHub官方的REST API文档 。

3. 发布版本时，自动发布镜像到 Registry

发布镜像，一般只需要编写好Dockerfile，随后编写一个工作流：

key	value
触发事件	release[created, edited]
动作	执行docker build指令，构建镜像并推送镜像到仓库

关于推送Docker镜像，Docker官方文档提供了如何通过GitHub Actions操作镜像的官方示例：

```yaml
 steps:

      - name: Check Out Repo
        uses: actions/checkout@v2

      - name: Login to Docker Hub
        uses: docker/login-action@v1
```

```
        with:
          username: ${{ secrets.DOCKER_HUB_USERNAME }}
          password: ${{ secrets.DOCKER_HUB_ACCESS_TOKEN }}

      - name: Set up Docker Buildx
        id: buildx
        uses: docker/setup-buildx-action@v1

      - name: Build and push
        id: docker_build
        uses: docker/build-push-action@v2
        with:
          context: ./
          file: ./Dockerfile
          push: true
          tags: ${{ secrets.DOCKER_HUB_USERNAME }}/simplewhale:latest

      - name: Image digest
        run: echo ${{ steps.docker_build.outputs.digest }}
```

可见只要有DockerHub的密钥，就能进行Docker镜像的提交，关于密钥的安全性问题，GitHub的***Secret***可以为GitHub Actions设置各种变量，因而这些重要信息将得以保证安全。

因为Dockerfile文件已经编写好了，那么这项任务重点应该只有DockerHub 密钥的安全问题，借助GitHub Actions访问GitHub的Secrets，能满足安全性的问题。

4.规划：

> 因为是保研本校，大四毕业这个暑假空闲时间非常多，可以保证较快的进度，也希望通过第一次参与开源的项目自己能学到很多新的东西。

1. 项目研发第一阶段（07 月 01 日 - 08 月 15 日）：

- [ ] 完成文档自动化部署任务，将实时更新的文档与ShardingSphere版本发布相关文档内容的分离。
- [ ] 完成issue自动化管理：
  - [ ] 自动关闭不合规范的issue
  - [ ] issue自动回复（参考：https://github.com/ant-design/ant-design/blob/master/.github/workflows/issue-reply.yml）
  - [ ] 自动指派
  - [ ] 检查不活跃issue并关闭
- [ ] 完成ShardingSphere的docker镜像向registry推送。

2. 项目研发第二阶段（08 月 16 日 - 09 月 30 日）：

- [ ] 解决在中期验收阶段中发现的问题
- [ ] 对第一阶段完成的内容进行更详细的测试
- [ ] 对第一阶段的完成内容进行总结，并输出相关文档内容
- [ ] 思考可以改进或者补充的地方

3. 期望：

希望借助这个机会，第一次参与到开源项目中，积累相关经验、学习新的知识，为日后参与更多开源项目提供一个经验借鉴。此外我对GitHub Actions非常感兴趣，希望项目结束后也能在项目CI/CD其他方面做一点点贡献。