项目申请书

项目名称: 存储和AI框架的深度集成

项目主导师: Tidas Chen

申请人: 樊漆亮 日期: 2021.05.24

邮箱: [fanqiliang@torch-fan.site](mailto:fanqiliang@torch-fan.site)

1. **项目背景**
   1. 项目基本需求：
      1. 文档自动发布:
      2. issue自动化管理：
      3. 当版本发布时自动发布docker镜像：
   2. 项目相关仓库：
2. **技术方法及可行性**
   1. GitHub Actions相关
   2. Docker相关
   3. Linux相关
3. **项目实现细节梳理**:
   1. 文档自动发布:
   2. issue标准处理流程：
      1. 超期未回复的issue自动关闭:
      2. 不合规范的issue自动关闭:
      3. issue自动指派:
      4. issue内容检查：
   3. 发布版本时，自动发布镜像到 Registry
4. **规划：**
   1. 项目研发第一阶段（07 月 01 日 - 08 月 15 日）：
   2. 项目研发第二阶段（08 月 16 日 - 09 月 30 日）：
   3. 期望：

# 项目背景

## 项目基本需求：

<https://github.com/apache/shardingsphere/issues/9697>

issue仓库地址:

* + 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。

* + 1. 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



[AntDesign](https://github.com/ant-design/ant-design)

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

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。

项目作为参考

* + 1. **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。

## 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](https://github.com/apache/shardingsphere-doc/blob/asf-site/.github/workflows/build-website.yml) 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 中。



[shell脚本](https://github.com/apache/shardingsphere-doc/blob/asf-site/.github/scripts/build-website.sh)

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。

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



[Torch-Fan](https://torch-fan.site/)



[Workflow语法](https://docs.github.com/en/actions/reference/workflow-syntax-for-github-actions)

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](https://github.com/marketplace?type=actions)

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：

。 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。



[qiliangfan](https://github.com/QiliangFan)

* 1. 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。

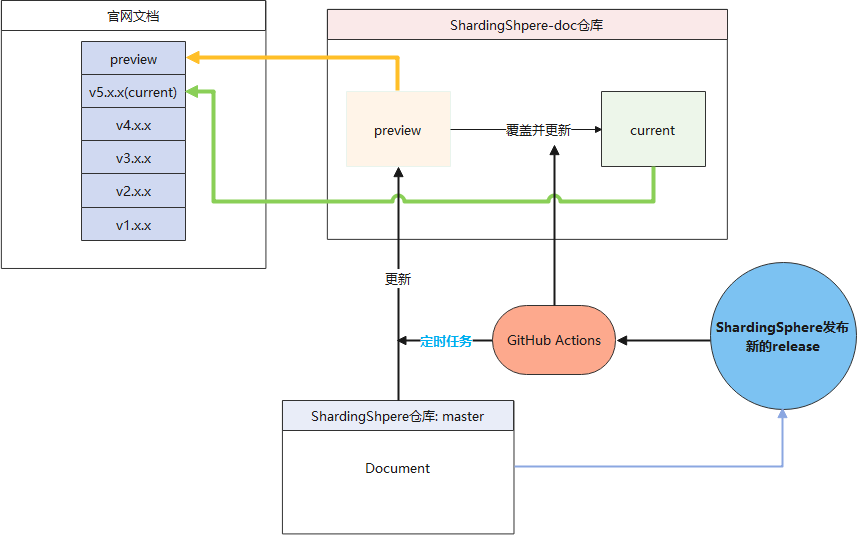
* 1. Linux**相关**

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。

# 项目实现细节梳理:

## 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 曰 ，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 因 , 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。

* 1. 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](https://github.com/marketplace/actions/issues-helper#check-inactive)

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

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](mailto:actions-cool/issues-helper@v2.2.1) with:

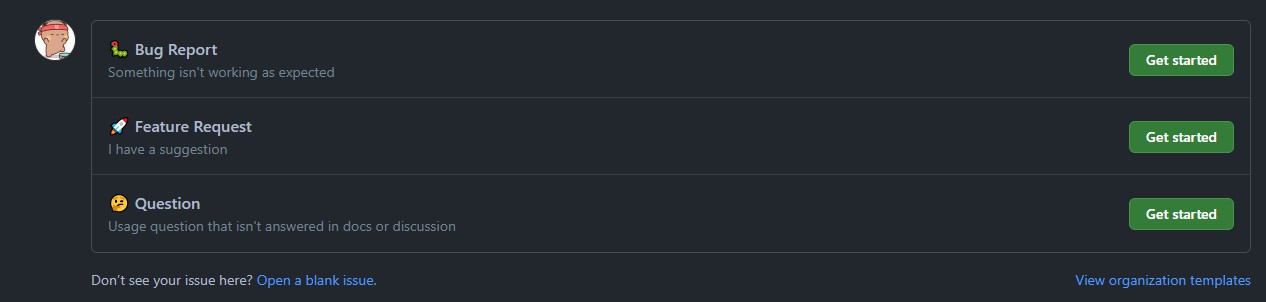
actions: 'check-inactive'

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。

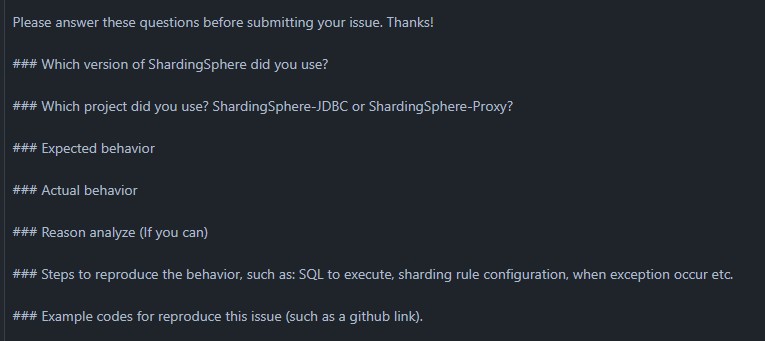
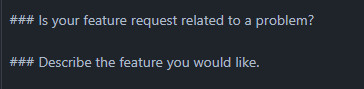
* + 1. **不合规范的**issue**自动关闭**:



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/contribu te/issue-conduct/](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。



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](https://docs.github.com/en/rest/reference/issues#list-issue-comments-for-a-repository)， 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](https://wiki.torch-fan.site/github-actions-chang-yong-dai-ma-pian-duan/)。 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](https://docs.github.com/en/rest/reference/issues) 和 @octokit/rest 库的[文档](https://octokit.github.io/rest.js/)。

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。

* + 1. issue**自动指派**:

name: Add Assigness

on:

issues:

types: [opened]

jobs:

add-assigness:

runs-on: ubuntu-latest steps:

- name: Add assigness

uses: [actions-cool/issues-helper@v2.2.1](mailto: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。

* + 1. 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](https://docs.github.com/en/rest/reference/issues) 。

* 1. **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：

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。

# 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/w orkflows/issue-reply.yml）](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。

* 1. **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

## 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。

**1. 项目背景**

在AI训练领域，存储解决方案的多样性为研发人员带来了选择的自由，同时也带来了兼容性和效率的挑战。为了提升AI训练的研发效率，降低框架接入存储的难度，本项目旨在为AI框架提供统一的存储接口。

**2. 项目技术方案**

**2.1 项目理解**

Alluxio作为一个虚拟分布式存储系统，是连接计算框架和存储系统的桥梁。它允许计算应用通过一个通用接口连接到多种存储系统。Alluxio的设计允许它作为一个数据编排层，为AI和机器学习在云中提供支持。

**2.2 需求分析**

基于Alluxio的设计理念，本项目将专注于以下几个核心需求：

1. **统一接口设计**：为AI框架提供一个统一的存储接口，以简化接入过程。
2. **存储API集成**：学习并集成主流存储API，如S3、POSIX等。
3. **测试与验证**：对集成的API进行单元测试、功能测试和性能测试。
4. **代码合并**：将开发完成的代码合并到Alluxio的主分支中。
5. **项目报告**：编写并提交项目报告，详细记录开发过程和结果。

**2.3 方案描述**

* **接口设计**：将设计一个中间层，该层将抽象不同存储API的共性，为AI框架提供一个简洁的接口。
* **API集成**：通过深入学习各种存储API的文档和源码，实现对它们的集成。
* **模块添加**：开发新的模块以支持不同的存储API，确保它们与Alluxio现有架构的兼容性和高效通信。

**2.4 依据引用**

* **其他项目经验**：参考类似项目如Apache Hadoop的统一存储接口设计。
* **理论依据**：引用相关论文，如"Alluxio: A Virtual Distributed File System"。

**3. 时间规划**

**3.1 第一阶段（1-4周）**：需求分析与技术准备

* 完成对Alluxio项目的深入学习和理解。
* 设计统一接口的初步方案。
* 学习并熟悉至少两种主流存储API。

**3.2 第二阶段（5-8周）**：API集成与模块开发

* 开始集成选定的存储API。
* 开发必要的模块以支持API集成。
* 进行初步的单元测试。

**3.3 第三阶段（9-12周）**：测试与优化

* 完成所有集成API的单元测试。
* 进行功能测试和性能测试。
* 根据测试结果优化代码。

**3.4 第四阶段（13-16周）**：代码合并与项目报告

* 将完成的代码合并到Alluxio主分支。
* 编写并提交项目报告。

**4. 参与动机与期望**

选择参与本项目是因为我对AI和云原生技术充满热情，并且相信通过为AI框架提供统一的存储接口，可以显著提升AI训练的效率。我期望通过这个项目，不仅能够提升个人技术水平，也能够为开源社区做出贡献。

**5. 个人技术背景**

* 熟练掌握Java语言，有面向对象编程经验。
* 了解存储系统和云原生技术。
* 有使用Python进行数据处理的经验。

**6. 项目成果仓库**

* [Alluxio GitHub仓库](https://github.com/Alluxio/alluxio)