
xuqinyang-doc

发布 *0.0.1*

xuqinyang

2022 年 06 月 26 日

Contents:

1	Python	1
1.1	常规	1
1.2	机器学习	1
1.3	其他	1
2	前端&后端	3
2.1	前端	3
2.2	后端	3
2.3	其他	3
2.3.1	Github Action	3
2.3.1.1	一、sphinx 的自动构建	3
2.3.1.1.1	1. 一些说明	3
2.3.1.1.2	2. 一些坑	4
2.3.1.1.3	3. 主要代码	4
2.3.1.1.4	4. 成果	8
2.3.1.2	二、基于 Github Action 和 Github Issue 的音乐生成	8
2.3.1.2.1	1. 一些说明	8
2.3.1.2.2	2. 一些坑	8
2.3.1.2.3	3. 主要代码	8
2.3.1.2.4	4. 成果	18
3	crack	19
3.1	Typora 破解 (1.3.6)	19
3.1.1	1. 抓取解密后 js:	19
3.1.2	2. 将解包出来最大的 js 重命名为 Atom.js, 找到以下代码替换 RSA Public Key 的 base64, 删除 renew:	20
3.1.3	3. 解包 app.asar:	20
3.1.4	4. 将刚刚修改好的 Atom.js 替换 “D:\Program Files\Typora\resources\app\” 下的同名文件	20
3.1.5	5. 删除 “D:\Program Files\Typora\resources\” 下的 “app.asar”	20
3.1.6	6. 编写 keygen	20
3.1.7	7. 将 keygen 生成的注册码输入到离线注册窗口并注册	21

CHAPTER 1

Python

1.1 常规

1.2 机器学习

1.3 其他

2.1 前端

2.2 后端

2.3 其他

2.3.1 Github Action

2.3.1.1 一、sphinx 的自动构建

2.3.1.1.1 1. 一些说明

想找个地方记录一下，看到许多 python 项目的文档是用 sphinx 写的，用一些平台托管自己的文档感觉不太自由，于是打算自己搭建一个。由于 sphinx 构建生成的都是静态页面，因此可以使用 Github Pages 托管。

如果每次都本地修改项目源码，然后本地构建，再将源代码上传至 Github 备份，静态页面部署至 Github Pages，不免觉得有些麻烦，偶然间发现了 Github Action 功能（~~ 这不是直接白嫖服务器吗 ~~），使用 Github Action 进行构建和部署，这样我只需要将初始源代码上传至 Github，然后想要修改或写文章的时候直接在 Github 上改。然后全自动更新 Github Pages，并且可以随时回滚版本。

2.3.1.1.2 2. 一些坑

- ① 在 Github Marketplace 上搜到的 sphinx 自动构建的 Action 大多都不能使用或版本老旧（需要修改许多地方），并且所有都不支持 markdown（需要加装个库）
- ② 使用 Github Pages 默认的 jekyll 会导致 sphinx 构建的 js 与 css 无法访问（因为 jekyll 会不会使用 _ 开头的文件/文件夹，而 sphinx 构建的 js 与 css 存放在 _static 中），因此需要禁用 jekyll
- ③ sphinx 几个库的版本要注意（之前因为版本直接不兼容导致搜索功能无法使用）

2.3.1.1.3 3. 主要代码

sphinx 项目文件放入 docs 目录下

/github/workflows/Build&Deploy.yml——启动

```
name: Build&Deploy
on:
  pull_request:
    branches: [ master ]
  workflow_dispatch:
  push:
    branches:
      - master
jobs:
  build-Github:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - # Setup Conda
        uses: conda-incubator/setup-miniconda@v2
        with:
          python-version: 3.7
      - # Runs this action
        uses: ./
        with:
          package_name: 'other_example'
      - uses: actions/download-artifact@v2
      - name: Check Artifacts
        run: |
          ls -al
          if [ ! -e documentation ]; then
            echo "documentation artifact not found"
            exit 1
          fi
          ls -al documentation
```

/action.yml——主 workflow，配置环境，构建，上传至 doc 分支，发布到 release

```
name: 'Build sphinx docs'
description: 'Builds IDS Sphinx documentation'
permissions:
  contents: write
inputs:
  docs_path: # id of input
    description: 'The path to the documentation folder from the repo root'
    required: false
```

(下页继续)

(续上页)

```

    default: 'docs'
  conda_build_env_filepath:
    description: 'Yaml Conda build environment definition file'
    required: false
    default: 'action_default'
  conda_build_env_name:
    description: 'Name of the build conda environment'
    required: false
    default: 'action_default'
  base_env_prefix: # id of input
    description: 'The prefix of the base Conda environment for self-hosted runs.'
    required: false
    default: '/usr/share/miniconda'
  artifact_name:
    description: 'Display name of the documentation artifact'
    required: false
    default: 'documentation'
  package_folder_path:
    description: 'Path to the folder containing the project's package(s) to be
↳ installed'
    required: false
    default: 'conda_package'
  package_name:
    description: 'Name of the project's Conda package'
    required: false
    default: ${ github.event.repository.name }
outputs:
  filepath:
    description: 'The file path of the generated HTML documentation'
    value: ${ steps.main.outputs.filepath }
runs:
  using: "composite"
  steps:
    - uses: actions/checkout@v2.2.0
      with:
        fetch-depth: 0 # Required due to the way Git works, without it this action won
↳ 't be able to find any or the correct tags
    - uses: actions/download-artifact@v2
      with:
        path: artifacts
    - id: main
      run: |
        echo "::set-output name=filepath::$(echo "None")"
        echo "CHECKS"
        echo "-----"
        if [ -d ${ inputs.docs_path } ]; then
          echo " Found the docs folder at ${ inputs.docs_path }"
        else
          echo " ERROR: Unable to locate the docs path, ${ inputs.docs_path }."
↳ Skipping the build of the docs."
          exit 0
        fi
        echo ""
        echo "Selecting Build Env yaml File"
        if [ ${ inputs.conda_build_env_filepath } = 'action_default' ]; then
          echo "Using the default conda configuration"

```

(下页继续)

(续上页)

```

    CONDA_BUILD_ENV_FILE="${{ github.action_path }}/envs/build-docs.yml"
elif [ -f "${{ inputs.conda_build_env_filepath }}" ]; then
    CONDA_BUILD_ENV_FILE="${{ inputs.conda_build_env_filepath }}"
else
    echo "Using the default conda configuration"
    CONDA_BUILD_ENV_FILE="${{ github.action_path }}/envs/build-docs.yml"
fi
echo "CONDA_BUILD_ENV_FILE: ${CONDA_BUILD_ENV_FILE}"
cat "${CONDA_BUILD_ENV_FILE}"

echo 'source "${{ inputs.base_env_prefix }}/etc/profile.d/conda.sh'
source "${{ inputs.base_env_prefix }}/etc/profile.d/conda.sh
echo "Checking that Conda is initialized"
if ! command -v conda &> /dev/null; then
    echo "ERROR: Conda is not setup."
    exit 1
fi
echo "  Conda is initialized"

echo "Conda build docs env name"
if [ "${{ inputs.conda_build_env_name }}" = 'action_default' ]; then
    echo "Using the default conda build env name: ${{ github.event.repository.
↪name }}-build-docs"
    CONDA_BUILD_ENV_NAME="${{ github.event.repository.name }}-build-docs"
else
    echo "Using the provided conda build env name: ${{ inputs.conda_build_env_
↪name }}"
    CONDA_BUILD_ENV_NAME="${{ inputs.conda_build_env_name }}"
fi

echo ""
echo "SETUP BUILD ENV"
echo "Set source"
echo "-----"
echo "Setting up ${{ github.event.repository.name }}-build environment"
conda env update --name ${CONDA_BUILD_ENV_NAME} \
    --file "${CONDA_BUILD_ENV_FILE}" || \
    conda env create -f "${CONDA_BUILD_ENV_FILE}"
conda activate ${CONDA_BUILD_ENV_NAME}
ls -al "${{ inputs.package_folder_path }}"
echo "-----"
if [ -a "${{ inputs.package_folder_path }}/${{ inputs.package_name }}-*.bz2" ];↪
↪then
    conda update conda-build || conda install conda-build
    echo "Installing project package"
    CHANNEL_PATH="${{runner.temp}}/channel/linux-64"
    mkdir -p "${CHANNEL_PATH}"
    cp "${{ inputs.package_folder_path }}/${{ inputs.package_name }}-*.bz2 $
↪{CHANNEL_PATH}"
    conda index "${CHANNEL_PATH}"
    conda update -c "${CHANNEL_PATH}" "${{ inputs.package_name }}" || \
        conda install -c "${CHANNEL_PATH}" "${{ inputs.package_name }}" || \
        (conda uninstall "${{ inputs.package_name }}" && \
            conda install -c "${CHANNEL_PATH}" "${{ inputs.package_name }}")

else

```

(下页继续)

(续上页)

```

        echo "Did not install project package"
    fi
    echo ""
    echo "conda info"
    conda info
    echo ""
    echo "conda list"
    conda list

    echo ""
    echo "BUILD DOCS"
    echo "-----"
    cd ${ inputs.docs_path }
    if [ -e "./setup_source.sh" ]; then
        ./setup_source.sh
    fi
    make html
    dir
    tar -cvf build.tar build
    gzip -9 build.tar
    zip -q -r build.zip build
    echo "::set-output name=filepath::$(echo '${ inputs.docs_path }/build/html')
→"

  shell: bash -l {0}
- uses: actions/upload-artifact@v2
  with:
    name: ${ inputs.artifact_name }
    path: docs/build/html
- id: mkdir123
  run: |
    mkdir -p docs/build/html
  shell: bash -l {0}
- uses: JamesIves/github-pages-deploy-action@v4.3.3
  with:
    branch: doc
    folder: docs/build/html
- id: previoustag
  uses: "WyriHaximus/github-action-get-previous-tag@v1"
  with:
    fallback: 1.0.0 # Optional fallback tag to use when no tag can be found
- id: semvers
  uses: "WyriHaximus/github-action-next-semvers@v1"
  with:
    version: ${ steps.previoustag.outputs.tag }
- uses: ncipollo/release-action@v1
  with:
    allowUpdates: true
    tag: ${ steps.semvers.outputs.patch }
    name: Release ${ steps.semvers.outputs.v_patch }
    artifacts: |
      docs/build.zip
      docs/build.tar.gz

```

/envs/build-docs.yml——需要的库

```
name: build-docs
```

(下页继续)

(续上页)

```

channels:
- defaults
- conda-forge
dependencies:
- conda=4.9.2
- pip
- pip:
- sphinx_markdown_tables==0.0.15
- sphinx==4.5.0
- recommonmark==0.7.1
- sphinx_rtd_theme==1.0.0
- sphinx-pandoc==0.6.0
- sphinx-autobuild
- sphinx-click==4.2.0
- sphinx-copybutton

```

2.3.1.1.4 4. 成果

<https://xqy2006.github.io/docs>

2.3.1.2 二、基于 Github Action 和 Github Issue 的音乐生成

2.3.1.2.1 1. 一些说明

之前写过一个前端 (Vue)+ 后端 (Flask) 版的, 无奈服务器终有一天会到期, 由于有着做第一个项目的经验, 所以想到了使用 Github Action 作为后端, 但是如果自己写前端的话势必需要对 Github 进行一些操作才能触发 workflow, 而对 Github 进行一些操作又需要登录 Github 账号, 太麻烦了, 有可能还会导致 token 泄露, 不如直接使用 Github 自带的 Issue 作为自己的前端 (~~ 其实是懒得写 ~~)

2.3.1.2.2 2. 一些坑

①numpy 绝对是大坑, 各种版本不兼容 (因此没有将 packages 上传至仓库)

②issue 是 markdown 的输入 markdown 的输出, 因此需要对 body 进行一些处理 (yaml 几乎没有对字符串处理的能力, 只能交给脚本了)

③一定要等待 Github Pages 部署完毕后再评论, 要不然结果还未部署用户就有可能发起下载请求

2.3.1.2.3 3. 主要代码

训练过程省略, 直接推理

由于 Github 最大文件限制为 100M, 故将模型压缩

music.py——推理主程序

```

import zipfile

f = zipfile.ZipFile("./Midi_Model/best_model.zip", 'r') # 压缩文件位置
for file in f.namelist():
    f.extract(file, "./Midi_Model/") # 解压位置

```

(下页继续)

(续上页)

```

f.close()
f = zipfile.ZipFile("./Midi_Model/final_model.zip", 'r') # 压缩文件位置
for file in f.namelist():
    f.extract(file, "./Midi_Model/") # 解压位置
f.close()
import sys
input = sys.argv[3]
import os
import json
from music21 import *
import base64
import paddle
import paddle.nn as nn
import numpy as np
from Reader import Reader
import Seq2Seq
from binascii import b2a_hex

batch_size = 10
train_reader = Reader(batch_size, './work/data')
import json
import time
# 初始化log写入器

# 模型参数设置
embedding_size = 256
hidden_size = 256
num_layers = 1

# 训练参数设置
epoch_num = 5000
learning_rate = 1e-5
log_iter = 200

# 定义一些所需变量
global_step = 0
log_step = 0
max_acc = 0

midi_model = Seq2Seq.Midi_Model(
    char_len=0x9FFF, # 基本汉字的Unicode码范围为4E00-9FA5,这里设置0x9FFF长, 基本够用
    embedding_size=embedding_size,
    hidden_size=hidden_size,
    num_layers=num_layers,
    batch_size=batch_size)
dur_model = Seq2Seq.Duration_Model(
    char_len=200, # midi范围一般在100左右,这里设置200长, 基本够用
    embedding_size=embedding_size,
    hidden_size=hidden_size,
    num_layers=num_layers,
    batch_size=batch_size)
midi_model.set_state_dict(paddle.load('Midi_Model/final_model'))
dur_model.set_state_dict(paddle.load('Duration_Model/final_model'))
input_lyrics = input
lyrics = []
for i, lyric in enumerate(input_lyrics.replace('\n', ' ')):

```

(下页继续)

(续上页)

```

    if i % batch_size == 0:
        lyrics.append([])
        lyrics[i // batch_size].append(ord(lyric))
    while len(lyrics[-1]) % batch_size != 0:
        lyrics[-1].append(ord('#'))
    lyrics = paddle.to_tensor(lyrics)

    params_dict = paddle.load('Midi_Model/best_model')
    midi_model.set_dict(params_dict)

    # 设置为评估模式
    midi_model.eval()

    # 模型推理
    out = midi_model(lyrics)

    # 结果转换
    results = []
    for _ in np.argmax(out.numpy(), -1).reshape(-1):
        results.append(_)

    midis = []
    dur_dic = {}
    with open('dur_dic.json', 'r') as f:
        dur_str = f.readline()
        dur_dic = json.loads(dur_str)
    for i, midi in enumerate(results):
        if i % batch_size == 0:
            midis.append([])
            midis[i // batch_size].append(midi) if midi <= 200 else midis[i // batch_size].
            ↪append(0)
    while len(midis[-1]) % batch_size != 0:
        midis[-1].append(0)
    midis = paddle.to_tensor(midis)

    params_dict = paddle.load('Duration_Model/best_model')
    dur_model.set_dict(params_dict)

    # 设置为评估模式
    dur_model.eval()

    # 模型推理
    # out = nn.Softmax(dur_model(midis))
    out = dur_model(midis)

    # 结果转换
    durations = []
    for _ in np.argmax(out.numpy(), -1).reshape(-1):
        durations.append(_)

    dur_dic = {}
    with open('dur_dic.json', 'r') as f:
        dur_str = f.readline()
        dur_dic = json.loads(dur_str)
        print(dur_dic)

```

(下页继续)

(续上页)

```

stream1 = stream.Stream()
for i, lyric in enumerate(input_lyrics.replace('\n', ' ')):
    if results[i] != 0:
        n1 = note.Note(results[i])
    else:
        n1 = note.Rest()
    n1.addLyric(lyric)
    n1.duration = duration.Duration(dur_dic[str(durations[i])])
    stream1.append(n1)
import random
name = ''
stream1.write("xml", './result/' + sys.argv[4] + ".xml")
stream1.write('midi', './result/' + sys.argv[4] + '.midi')
output = input + '.midi'
print(output)

```

verify.py——校验输入是否全部为中文字符，若不是则报错，由 verify.yml 进行后续操作

```

import sys
input = sys.argv[3]
def check_contain_chinese(check_str):
    for ch in check_str:
        if u'\u4e00' <= ch <= u'\u9f5a':
            pass
        else:
            return True
    return False
if check_contain_chinese(input):
    raise Exception('error')

```

sleep.py——等待 Github Pages 部署完成后再回复

```

import requests
import os
import urllib3
import time
urllib3.disable_warnings()
import sys
def download(url):
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:68.0) Gecko/20100101 Firefox/68.0"
    }
    r = requests.get(url=url, headers=headers, verify=False)
    return r.status_code
while download('http://xqy2006.github.io/music_generation/'+sys.argv[1]+' .xml')==404:
    print("Page undeploy")
    time.sleep(5)
print("finish")

```

./github/ISSUE_TEMPLATE/music_generation.yml——问题模板

```

name: 音乐生成
description: 音乐生成
title: "音乐生成"
labels: ["music_generation"]

```

(下页继续)

(续上页)

```
body:
- type: textarea
  id: input_text
  attributes:
    label: "歌词"
    description: 请输入要谱曲的歌词（只能是中文，无标点）：
    placeholder:
    value:
  validations:
    required: true
```

music_generation/.github/workflows/music.yml——主 workflow，将 issue 的 body 与 id 传入 python 处理，push 生成结果文件到 store 分支构建 Github Pages，回复并关闭 issue

```
name: music
on:
  issues:
    types: [opened,edited]
  workflow_dispatch:
jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - run: |
        mkdir -p ./id
        echo $ID > ./id/id
      env:
        ID: ${github.event.issue.number}
    - uses: actions/upload-artifact@v3
      with:
        name: id
        path: id/
    - uses: actions/setup-python@v4
      with:
        python-version: '3.x' # Version range or exact version of a Python version to
        ↪use, using SemVer's version range syntax
        architecture: 'x64' # optional x64 or x86. Defaults to x64 if not specified
    - run: |
        python verify.py $BODY
        git clone -b store https://github.com/xqy2006/music_generation.git "result"
        pip install paddlepaddle
        pip install music21
        pip install protobuf==3.20.0
        python music.py $BODY $ID
      env:
        BODY: ${github.event.issue.body}
        ID: ${github.event.issue.number}
    - uses: JamesIves/github-pages-deploy-action@v4.3.3
      with:
        branch: store
        folder: result
    - uses: rishabhgupta/split-by@v1
      id: split
```

(下页继续)

(续上页)

```

with:
  string: ${ github.event.issue.body }
  split-by: "歌词 \n "
- uses: mad9000/actions-find-and-replace-string@2
  id: findandreplace
  with:
    source: ${ steps.split.outputs._1 } # this translates to ref/heads/main on
    ↪ the main branch, but can be any arbitrary string
    find: "\n" # we want to remove ref/heads/ from source
    replace: '' # and replace it with a blank string (ie. removing
    ↪ it)
- run: |
  python sleep.py $ID
  echo OK!
env:
  ID: ${ github.event.issue.number }
- name: Create comment
  uses: actions-cool/issues-helper@v2
  with:
    actions: 'create-comment'
    token: ${ secrets.GITHUB_TOKEN }
    issue-number: ${ github.event.issue.number }
    body: |
      你好 ${ github.event.issue.user.login }, 生成结果如下:
      midi格式下载地址为 (导入到曲谱软件中无歌词, 可在电脑上直接播放) : https://
    ↪ xqy2006.github.io/music_generation/${ github.event.issue.number }.midi
      xml格式下载地址为 (导入到曲谱软件中有歌词, 不可在电脑上直接播放) : https://
    ↪ /xqy2006.github.io/music_generation/${ github.event.issue.number }.xml
- name: Close issue
  uses: actions-cool/issues-helper@v2
  with:
    actions: 'close-issue'
    token: ${ secrets.GITHUB_TOKEN }
    issue-number: ${ github.event.issue.number }

```

music_generation/.github/workflows/verify.yml——当 music.yml 报错后运行 (由 verify.py 抛出), 回复并关闭 issue

```

name: verify
on:
  workflow_run:
    workflows: [music]
    types: [completed]
jobs:
  on-failure:
    runs-on: ubuntu-latest
    if: ${ github.event.workflow_run.conclusion == 'failure' }
    steps:
      - name: 'Download artifact'
        uses: actions/github-script@v6
        with:
          script: |
            let allArtifacts = await github.rest.actions.listWorkflowRunArtifacts({
              owner: context.repo.owner,
              repo: context.repo.repo,
              run_id: context.payload.workflow_run.id,
            });

```

(下页继续)

(续上页)

```

    let matchArtifact = allArtifacts.data.artifacts.filter((artifact) => {
      return artifact.name == "id"
    })[0];
    let download = await github.rest.actions.downloadArtifact({
      owner: context.repo.owner,
      repo: context.repo.repo,
      artifact_id: matchArtifact.id,
      archive_format: 'zip',
    });
    let fs = require('fs');
    fs.writeFileSync(`${process.env.GITHUB_WORKSPACE}/id.zip`, Buffer.
↪from(download.data));

- name: 'Unzip artifact'
  run: unzip id.zip

- name: 'Comment on issue'
  uses: actions/github-script@v6
  with:
    github-token: ${ secrets.GITHUB_TOKEN }
    script: |
      let fs = require('fs');
      let issue_number = Number(fs.readFileSync('./id'));
      await github.rest.issues.createComment({
        owner: context.repo.owner,
        repo: context.repo.repo,
        issue_number: issue_number,
        body: "# 您的输入有误! \n请检查您的输入是否全部为中文字符, 并且没有标点\
↪n可以通过 https://xqy2006.github.io/Chinese-character/ 去除非中文字符",
      });
      await github.rest.issues.update({
        owner: context.repo.owner,
        repo: context.repo.repo,
        issue_number: issue_number,
        state: 'closed',
      });

```

Seq2Seq.py——推理所需

```

import paddle
import paddle.nn as nn

# 继承paddle.nn.Layer类
class Midi_Model(nn.Layer):
    # 重写初始化函数
    # 参数: 字符表长度、嵌入层大小、隐藏层大小、解码器层数、处理数字的最大位数
    def __init__(self, char_len, embedding_size=128, hidden_size=128, num_layers=1,
↪batch_size=20):
        super(Midi_Model, self).__init__()
        # 初始化变量
        self.MAXLEN = 1
        self.batch_size = batch_size
        self.hidden_size = hidden_size
        self.char_len = char_len
        self.num_layers=num_layers
        self.embedding_size=embedding_size

```

(下页继续)

(续上页)

```

# 嵌入层
self.emb = nn.Embedding(
    char_len,
    self.embedding_size
)

# 编码器
self.encoder = nn.LSTM(
    input_size=self.embedding_size,
    hidden_size=self.hidden_size,
    num_layers=self.num_layers
)

# 解码器
self.decoder = nn.LSTM(
    input_size=self.hidden_size,
    hidden_size=self.hidden_size,
    num_layers=self.num_layers
)

# 全连接层
self.fc = nn.Linear(
    self.hidden_size,
    char_len
)

# 重写模型前向计算函数
# 参数: 输入[None, MAXLEN]、标签[None, DIGITS]
def forward(self, inputs, labels=None):
    # 嵌入层
    out = self.emb(inputs)

    # 编码器
    out, (_, _) = self.encoder(out)

    # 按时间步切分编码器输出
    out = paddle.split(out, self.MAXLEN, axis=1)

    # 取最后一个时间步的输出并复制batch_size次
    out = paddle.expand(out[-1], [out[-1].shape[0], self.batch_size, self.hidden_
    size])

    # 解码器
    out, (_, _) = self.decoder(out)

    # 全连接
    out = self.fc(out)

    # 如果标签存在, 则计算其损失和准确率
    if labels is not None:
        # 转置解码器输出
        tmp = paddle.transpose(out, [0, 2, 1])

        # 计算交叉熵损失
        loss = nn.functional.cross_entropy(tmp, labels, axis=1)

```

(下页继续)

(续上页)

```

        # 计算准确率
        acc = paddle.metric.accuracy(paddle.reshape(out, [-1, self.char_len]),
        ↪paddle.reshape(labels, [-1, 1]))

        # 返回损失和准确率
        return loss, acc

    # 返回输出
    return out

# 继承paddle.nn.Layer类
class Duration_Model(nn.Layer):
    # 重写初始化函数
    # 参数: 字符表长度、嵌入层大小、隐藏层大小、解码器层数、处理数字的最大位数
    def __init__(self, char_len, embedding_size=128, hidden_size=64, num_layers=1,
    ↪batch_size=20):
        super(Duration_Model, self).__init__()
        # 初始化变量
        self.batch_size = batch_size
        self.MAXLEN = 1
        self.hidden_size = hidden_size
        self.char_len = char_len
        self.num_layers=num_layers
        self.embedding_size=embedding_size

        # 嵌入层
        self.emb = nn.Embedding(
            self.char_len,
            self.embedding_size
        )

        # 编码器
        self.encoder = nn.LSTM(
            input_size=embedding_size,
            hidden_size=self.hidden_size,
            num_layers=self.num_layers
        )

        # 解码器
        self.decoder = nn.LSTM(
            input_size=self.hidden_size,
            hidden_size=self.hidden_size,
            num_layers=self.num_layers
        )

        # 全连接层
        self.fc = nn.Linear(
            self.hidden_size,
            self.char_len
        )

    # 重写模型前向计算函数
    # 参数: 输入 [None, MAXLEN]、标签 [None, DIGITS]
    def forward(self, inputs, labels=None):

```

(下页继续)

(续上页)

```

# 嵌入层
out = self.emb(inputs)

# 编码器
out, (_, _) = self.encoder(out)

# 按时间步切分编码器输出
out = paddle.split(out, self.MAXLEN, axis=1)

# 取最后一个时间步的输出并复制batch_size次
out = paddle.expand(out[-1], [out[-1].shape[0], self.batch_size, self.hidden_
→size])

# 解码器
out, (_, _) = self.decoder(out)

# 全连接
out = self.fc(out)

# 如果标签存在, 则计算其损失和准确率
if labels is not None:
    # 转置解码器输出
    tmp = paddle.transpose(out, [0, 2, 1])

    # 计算交叉熵损失
    loss = nn.functional.cross_entropy(tmp, labels, axis=1)

    # 计算准确率
    acc = paddle.metric.accuracy(paddle.reshape(out, [-1, self.char_len]),
→paddle.reshape(labels, [-1, 1]))

    # 返回损失和准确率
    return loss, acc

# 返回输出
return out

```

Reader.py——推理所需

```

from music21 import note, converter
import numpy as np
import os
import json
import fractions

def Reader(DIGITS, path = './work/data'):
    dur_dic = {}
    def read_data():
        for file in os.listdir(path):
            lyrics = []
            midis = []
            durations = []
            xml = converter.parseFile(os.path.join(path, file))
            #print(dir(stream.Score()))
            for i, note in enumerate(xml.recurse().notesAndRests):
                if i%DIGITS == 0:

```

(下页继续)

```

        lyrics.append([])
        midis.append([])
        durations.append([])
        lyric = note._getLyric()
        if lyric == None:
            lyric = '#'
        lyrics[i//DIGITS].append(ord(lyric))
        try:
            midis[i//DIGITS].append(note.pitch.midi)
        except:
            midis[i//DIGITS].append(0)
            durations[i//DIGITS].append(note.duration.quarterLength)
            if type(note.duration.quarterLength) == fractions.Fraction and
↪float(note.duration.quarterLength) not in list(dur_dic.values()):
                dur_dic[len(dur_dic)] = float(note.duration.quarterLength)
            elif type(note.duration.quarterLength) != fractions.Fraction and note.
↪duration.quarterLength not in list(dur_dic.values()):
                dur_dic[len(dur_dic)] = note.duration.quarterLength
            yield [midis,durations,lyrics]
        with open('dur_dic.json','w') as f:
            f.write(json.dumps(dur_dic))
    return read_data

```

2.3.1.2.4 4. 成果

https://www.github.com/xqy2006/music_generation

3.1 Typora 破解 (1.3.6)

3.1.1 1. 抓取解密后 js:

```
pip install frida
frida "D:\Program Files\Typora\Typora.exe" -l "./unpack.js"
```

unpack.js:

```
let napi_create_string_utf8 = Module.getExportByName(null, 'napi_create_string_utf8');
var index = 0;
if (napi_create_string_utf8) {
    console.log('绑定成功');
    Interceptor.attach(napi_create_string_utf8, {
        onEnter: function (args) {
            console.log('napi_create_string_utf8', '调用', args[0], args[1].
            ↪readCString().substring(0, 100), args[2], args[3]);

            if (args[2].toInt32() > 100) { // 过滤出大文件
                index += 1;
                var f = new File('export_' + String(index) + '.js', 'wb');
                f.write(args[1].readByteArray(args[2].toInt32()));
                f.flush();
                f.close();
            }
        }
    });
} else {
    console.log('绑定失败');
}
```

3.1.2 2. 将解包出来最大的 js 重命名为 Atom.js，找到以下代码替换 RSA Public Key 的 base64，删除 renew：

```
T=JSON.parse(Buffer.from(
→ "WyItLS0tLUJFR0l0IFBVQkxJQyBLRVktLS0tLSIsIk1JSUJJakFOQmdrcWhraUc5dzBCQVFFRkFBT0NBUThtBTUlJQkNnS0NBUTU="
→ ", "base64").toString("utf8"))

//删除下面的 /api/client/renew
const a = await (await k(W + "/api/client/renew", {
  method: "POST",
  cache: "no-cache",
  body: JSON.stringify(n),
  headers: {
    "Content-Type": "application/json",
    "Cache-Control": "no-cache"
  }
})
```

3.1.3 3. 解包 app.asar：

```
npm install asar -g
cd D:\Program Files\Typora\resources
asar extract ./app.asar ./app
```

3.1.4 4. 将刚刚修改好的 Atom.js 替换 “D:\Program Files\Typora\resources\app\” 下的同名文件

3.1.5 5. 删除 “D:\Program Files\Typora\resources\” 下的 “app.asar”

3.1.6 6. 编写 keygen

RSA 公私钥生成：

```
const crypto = require('crypto');
const fs = require('fs');
const path = require('path');

const keyPair = crypto.generateKeyPairSync('rsa', {
  modulusLength: 2048,
  publicKeyEncoding: {
    type: 'spki',
    format: 'pem'
  },
  privateKeyEncoding: {
    type: 'pkcs8',
    format: 'pem',
  }
});

fs.writeFileSync("public_key.pem", keyPair.publicKey);
fs.writeFileSync("private_key.pem", keyPair.privateKey);
```

keygen:

