复赛提交说明

每支队伍需提交一个`.tar.gz`格式的docker镜像文件，该docker镜像用于创建容器，容器将提供一个代码自动生成API服务。该服务需要部署在`localhost:8000`，并向外部暴露`8000`端口。

API接口规范

接口地址与方法

- **URL**: `localhost:8000`
- **方法**: POST

请求参数

请求参数为JSON格式，具体字段如下：

- `title` (string): 题目标题
- `description` (string): 题目的文字描述
- `type` (string): 需要生成的块类型，可为`FUNCTION`或`FUNCTION_BLOCK`
- `name` (string): 块的名称
- `input` (array): 块接口的输入参数列表
  - `name` (string): 参数名称
  - `type` (string): 参数类型
  - `description` (string): 参数的文字描述
  - `fields` (array, optional): 结构体的成员，仅在参数类型为结构体`Struct`时出现。每一个成员的内容如下：
    - `name` (string): 结构体成员名称
    - `type` (string): 结构体成员类型
    - `description` (string): 结构体成员的文字描述
- `output` (array): 块接口的输出参数列表
  - `name` (string): 参数名称
  - `type` (string): 参数类型
  - `description` (string): 参数的文字描述
  - `fields` (array, optional): 结构体的成员，仅在参数类型为结构体`Struct`时出现。每一个成员的内容如下：
    - `name` (string): 结构体成员名称
    - `type` (string): 结构体成员类型
    - `description` (string): 结构体成员的文字描述
- `in/out` (array): 块接口的输入输出参数列表
  - `name` (string): 参数名称

    - `type` (string): 参数类型
    - `description` (string): 参数的文字描述
    - `fields` (array, optional): 结构体的成员，仅在参数类型为结构体`Struct`时出现。每一个成员的内容如下：
        - `name` (string): 结构体成员名称
        - `type` (string): 结构体成员类型
        - `description` (string): 结构体成员的文字描述
- `return_value` (object, optional): 函数的返回值，仅在块类型为`FUNCTION`时出现
    - `type` (string): 返回值类型
    - `description` (string): 返回值的文字描述
    - `fields` (array, optional): 结构体的成员，仅在返回值类型为结构体`Struct`时出现。每一个成员的内容如下：
        - `name` (string): 结构体成员名称
        - `type` (string): 结构体成员类型
        - `description` (string): 结构体成员的文字描述

响应参数

响应参数为JSON格式，具体字段如下：

- `name` (string): 块的名称，与请求参数中的`name`字段一致
- `code` (string): 生成的代码，写入scl文件后可以直接用TIA Portal编译运行

性能要求

该接口从接受请求到返回响应的时间间隔不得超过90秒。若某道题在生成代码时超时则该题目不得分。

网络访问规范

- 选手提交的Docker容器只允许访问以下指定的网络接口：https://open.bigmodel.cn/api/。此访问权限仅覆盖https协议下的指定域名及其合法子路径，其他任何形式的网络请求（包括但不限于HTTP、FTP、SSH等）均不被允许。
- 所有其它外部网络访问将被严格限制，以确保比赛的公平性和安全性。

测试机配置

- CPU：Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz * 2
- 内存：400G
- 硬盘：1.8T
- docker版本：19.03.13

请注意，测试机不含GPU，故请勿在代码里包含GPU操作，在执行本地大模型推理等通常需要GPU的操作时需要考虑对性能的影响。BERT量级的模型推理可以用CPU完成。

流程示例

1. 编写应用程序，提供符合API规范的服务。

   以Flask为例，以下是一个简单的Flask应用示例，供参考：

```python
from flask import Flask, request, jsonify
import time

app = Flask(__name__)

@app.route('/', methods=['POST'])
def generate_code():
    data = request.get_json()

    # 模拟代码生成逻辑
    code = "FUNCTION \"...\"\n..." # 这里替换为实际的代码生成逻辑

    response = {
        "name": data.get("name"),
        "code": code
    }
    return jsonify(response)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000)
```

2. 编写Dockerfile，配置基础环境和依赖。

   以下是一个Dockerfile的编写示例：

```dockerfile
# 基础镜像
FROM python:3.11-slim

# 设置工作目录
WORKDIR /app

# 复制依赖文件到工作目录
COPY requirements.txt /app/requirements.txt

# 安装依赖
RUN pip install --no-cache-dir -r requirements.txt

# 复制项目文件到工作目录
COPY . /app

# 暴露端口
EXPOSE 8000

# 启动服务
CMD ["python", "app.py"]
```

3. 使用以下命令构建docker镜像：

```
docker build -t code-gen-api .
```

4. 使用以下命令将docker镜像导出为`.tar`文件：

```
docker save -o code-gen-api.tar code-gen-api
```

   - `-o`选项指定输出文件名，这里为`code-gen-api.tar`。
   - `code-gen-api`是镜像名称。

注意事项

- 请使用指定的端口号，以避免评测时无法调用服务。
- 容器启动后，应验证API服务是否按预期运行。