

系统设计文档

1. 项目概述

1.1 项目背景

在大数据时代，分布式系统已经成为信息存储和处理的主流系统。由于其庞大和复杂的特性，分布式系统的故障发生的平均几率较高，运维的难度和复杂度也大大提高。如何对分布式系统进行高效、准确的运维，成为保障信息系统高效、可靠运行的关键问题。

SystemHealer(系统治愈师) 项目是一个基于机器学习的分布式系统故障诊断系统，它的目标是通过分析分布式系统的故障数据，设计故障诊断模型，高效地分析并识别故障类别，实现分布式系统故障运维的智能化，快速恢复故障的同时大大降低分布式系统运维工作的难度，减少运维对人力资源的消耗。

1.2 项目目标

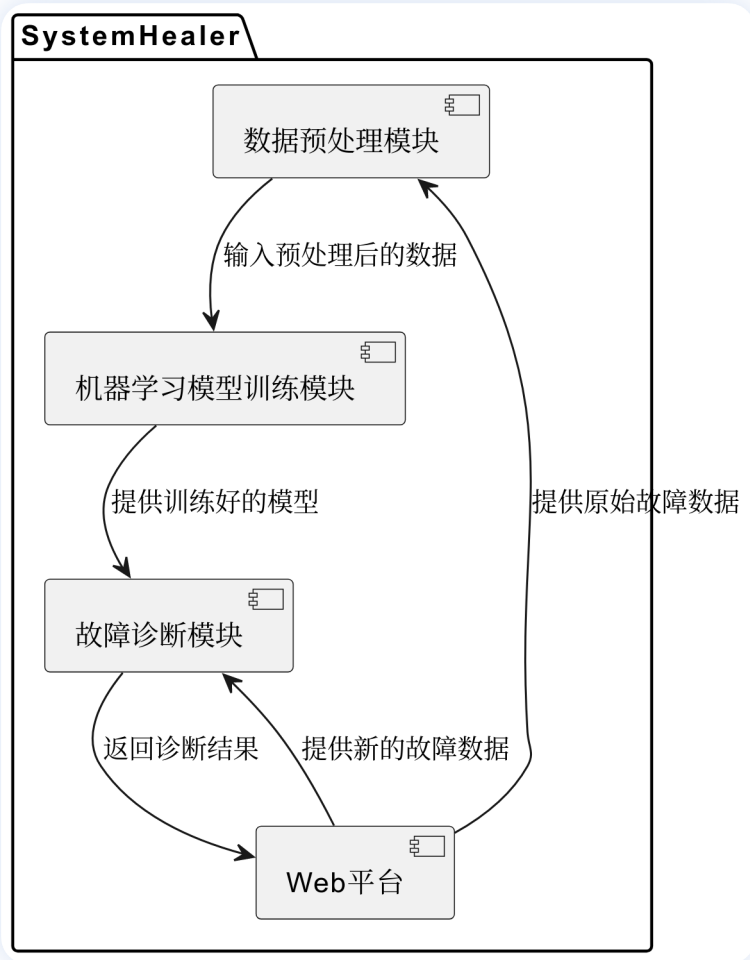
SystemHealer 的主要目标是：

- 利用机器学习技术，对分布式系统的故障数据进行深度分析，设计并实现故障诊断模型。
- 通过故障诊断模型，能够高效地分析并识别分布式系统的故障类别，实现分布式系统故障运维的智能化。
- 构建一个用户友好的Web平台，该平台支持用户上传训练数据并在线训练，训练完成后可下载训练模型。同时，该平台支持单条或者批量测试样本上传，并可视化分类结果，同时支持下载分类结果。
- 通过提高故障诊断的准确性和效率，快速恢复故障，大大降低分布式系统运维工作的难度，减少运维对人力资源的消耗。

2. 系统架构

2.1 系统组件

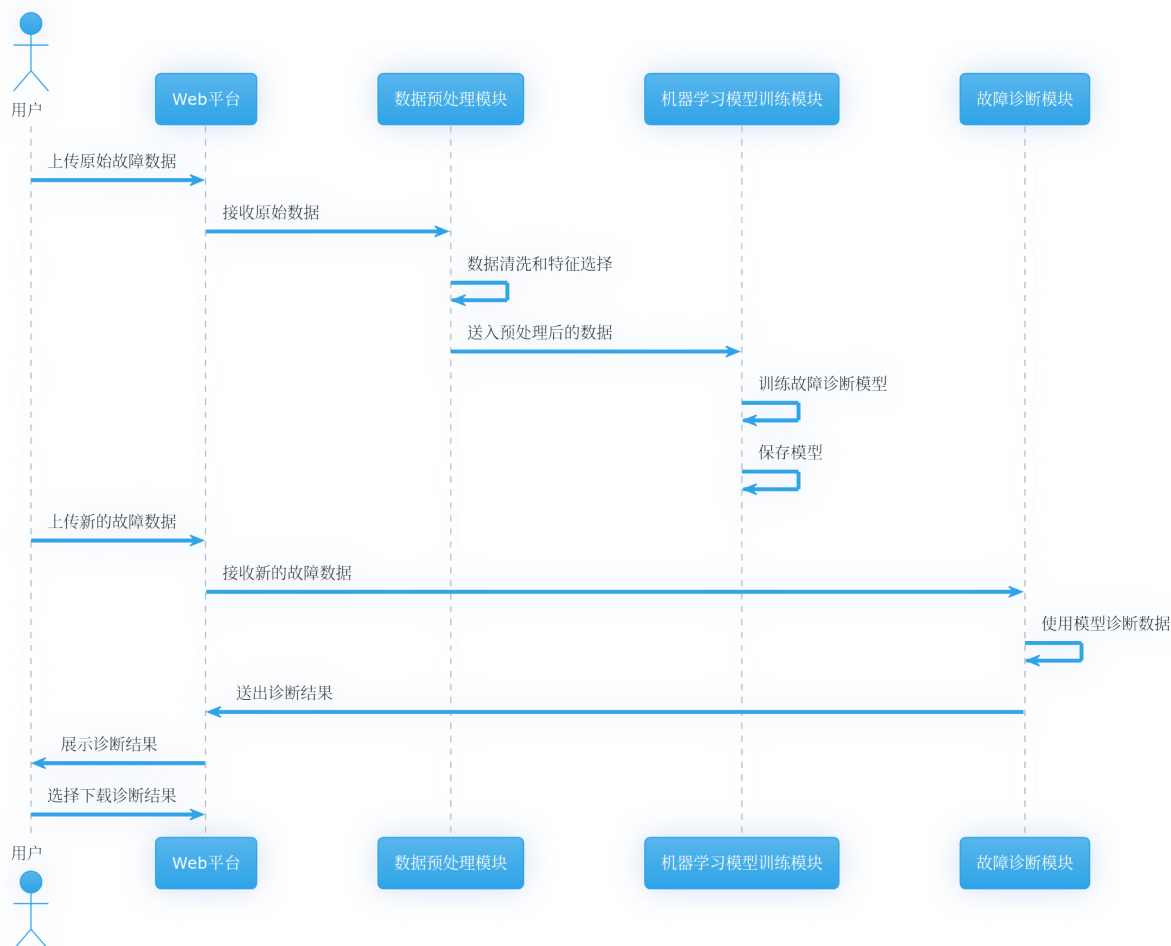
SystemHealer 项目主要由以下几个关键组件构成：



- 1. 数据预处理模块：**这个模块负责对原始的故障数据进行预处理，包括数据清洗、特征选择等，以便于后续的机器学习模型训练。
- 2. 机器学习模型训练模块：**这个模块负责使用预处理后的数据训练机器学习模型。模型训练完成后，可以用于后续的故障诊断。
- 3. 故障诊断模块：**这个模块负责使用训练好的机器学习模型对新的故障数据进行诊断，识别出故障的类别。
- 4. Web平台：**这是一个用户交互界面，支持用户上传训练数据进行模型训练，也支持用户上传测试数据进行故障诊断。同时，该平台还提供了数据可视化功能，可以直观地展示诊断结果。

2.2 系统流程

以下是 `SystemHealer` 项目的主要工作流程：



- 数据预处理：**首先，用户通过Web平台上传原始的故障数据。数据预处理模块接收到数据后，进行数据清洗和特征选择等预处理操作。
- 模型训练：**预处理后的数据被送入机器学习模型训练模块。该模块使用这些数据训练出一个故障诊断模型，并将模型保存下来，供后续使用。
- 故障诊断：**当有新的故障数据需要诊断时，用户可以通过Web平台上传这些数据。故障诊断模块接收到数据后，使用之前训练好的模型对这些数据进行诊断，识别出故障的类别。
- 结果展示：**诊断结果会被送到Web平台上，通过数据可视化的方式展示给用户。用户也可以选择下载诊断结果。

3. 系统实现

3.1 数据预处理模块：

这个模块负责对原始的故障数据进行预处理，包括数据清洗、特征选择等，以便于后续的机器学习模型训练。在我们的Python代码中，这个模块由两个函数 `preprocess` 和 `test_preprocess` 实现。`preprocess` 函数接收一个 `DataFrame` 类型的数据作为输入，进行去重、缺失值填充和标准化等预处理操作，然后返回处理后的数据和对应的标签。`test_preprocess` 函数则用于处理测试数据，它只进行缺失值填充和标准化操作。

C preprocess
<ul style="list-style-type: none"> • preprocess(data: DataFrame): Tuple[DataFrame, Series]

C test_preprocess
<ul style="list-style-type: none"> • test_preprocess(data: DataFrame): DataFrame

3.2 机器学习模型训练模块：

这个模块负责使用预处理后的数据训练机器学习模型。模型训练完成后，可以用于后续的故障诊断。在我们的Python代码中，这个模块由 `MLP` 类和 `train` 函数实现。`MLP` 类定义了一个多层感知机模型，包括模型的初始化和前向传播过程。`train` 函数则负责读取数据，进行数据预处理和模型训练，并保存训练好的模型。

C MLP	C EarlyStopping
<ul style="list-style-type: none"> • __init__(input_size: int, hidden_size: int, num_classes: int) • forward(x: Tensor, return_features: bool = False): Tensor 	<ul style="list-style-type: none"> • __init__(patience: int = 7, verbose: bool = False, delta: float = 0) • __call__(val_loss: float, model: nn.Module) • save_checkpoint(val_loss: float, model: nn.Module)
C train	
<ul style="list-style-type: none"> • train(input_path: str, file_basic_path: str, output_path: str): Dict 	

3.3 故障诊断模块：

这个模块负责使用训练好的机器学习模型对新的故障数据进行诊断，识别出故障的类别。在我们的Python代码中，这个模块由 `test` 函数实现。`test` 函数读取测试数据，进行数据预处理，然后使用训练好的模型进行故障诊断，并保存诊断结果。

C test
<ul style="list-style-type: none"> • test(input_path: str, file_basic_path: str, output_path: str): Dict

3.4 Web平台：

这是一个用户交互界面，支持用户上传训练数据进行模型训练，也支持用户上传测试数据进行故障诊断。同时，该平台还提供了数据可视化功能，可以直观地展示诊断结果。

4. 系统部署

4.1 使用脚本进行部署

在 `SystemHealer/` 目录运行 `setup.sh` 脚本，可以一键构建环境，脚本内容如下：

```
1  #!/bin/bash
2
3  # 安装 virtualenv
4  pip install virtualenv
5
6  # 创建虚拟环境
7  virtualenv venv
8
9  # 激活虚拟环境
10 source ./venv/bin/activate
11
12 # 安装依赖
13 pip install -r requirements.txt
14
15 # 迁移数据库
16 python3 manage.py migrate
17
18 # 运行服务器
19 python3 manage.py runserver
20
21
22 # 睡眠1s
23 sleep 1
24
25 # 打开浏览器并访问localhost:8000
26 xdg-open http://localhost:8000
```

4.2 使用 Docker 进行部署

在 `SystemHealer/` 目录写有 `Dockerfile` 文件：

```
1  # 使用官方 Python 运行时作为父镜像
2  FROM python:3.8-slim-buster
3
4  # 设置工作目录
5  WORKDIR /app
6
```

```
7 # 将当前目录内容复制到容器的 /app 目录中
8 COPY . /app
9
10 # 安装项目需要的包
11 RUN pip install --no-cache-dir virtualenv && \
12     virtualenv venv && \
13     . ./venv/bin/activate && \
14     pip install --no-cache-dir -r requirements.txt
15
16 # 运行迁移
17 RUN python3 manage.py migrate
18
19 # 使端口 8000 可供此容器外的环境使用
20 EXPOSE 8000
21
22 # 定义环境变量
23 ENV NAME SystemHealer
24
25 # 运行服务器
26 CMD ["python3", "manage.py", "runserver", "0.0.0.0:8000"]
```

然后，运行以下命令来构建 Docker 镜像：

```
1 docker build -t systemhealer .
```

最后，运行以下命令来启动 Docker 容器：

```
1 docker run -p 8000:8000 systemhealer
```

现在，您可以在浏览器中输入 `localhost:8000` 来访我们的Web系统。