

2025 华为软件精英挑战赛

# 题目与判题器补充说明

文档版本

v1

发布日期

2025-03-10



# 目 录

---

目 录.....	1
1 环境信息说明.....	2
1.1 整体说明.....	2
1.2 编译环境.....	2
1.3 具体语言限制.....	2
1.4 补充说明.....	3
2 判题器说明.....	5
2.1 下发的示例程序 Demo 及脚本 run.py .....	5
2.2 判题器异常判定与处理 .....	6
2.3 debug 模式说明 .....	7
2.4 replay 模式说明 .....	8
2.5 可视化程序说明 .....	9
3 数据说明.....	10
4 常见问题 QA.....	11

# 1 环境信息说明

## 1.1 整体说明

1. 参赛选手所提交的源码压缩包，由系统解压后自动编译运行，生成大赛结果；
2. 运行参赛选手源码的操作系统为 Linux 操作系统；
3. 执行机系统环境：

操作系统：基于 x86\_64 的 ubuntu 18.04

CPU：4 核

内存：8GB（选手代码、判题器、判题平台共用）

## 1.2 编译环境

1. C/C++

GNU Make 4.1

cmake version 3.22.0

gcc 7.3.0

g++ 7.3.0

2. Java

JDK 1.8

3. Python

python 3.7.3

## 1.3 具体语言限制

1. C 语言

文件名	是否可以修改	是否需要上传	说明
CMakeLists.txt	√	√	平台编译会按照上传的 CMakeLists.txt 进行编译，选手需要保证最终编译产物名称为 <b>code_craft</b> 。
main.c	√	√	main 文件，可以根据 CMakeLists.txt 修改名称

其它新增代码	√	√	参赛者可以增加新的代码文件
--------	---	---	---------------

## 2. C++

文件名	是否可以修改	是否需要上传	说明
CMakeLists.txt	√	√	平台编译会按照上传的 CMakeLists.txt 进行编译，选手需要保证最终编译产物名称为 <b>code_craft</b> 。
main.cpp	√	√	main 文件，可以根据 CMakeLists.txt 修改名称
其它新增代码	√	√	参赛者可以增加新的代码文件

## 3. Java

文件名	是否可以修改	是否需要上传	说明
Main.java	√	√	main 文件
其它新增代码	√	√	参赛者可以增加新的代码文件

## 4. Python

文件名	是否可以修改	是否需要上传	说明
main.py	√	√	main 文件
其它新增代码	√	√	参赛者可以增加新的代码文件

# 1.4 补充说明

## 1. C/C++

1.1 C/C++工程均采用 CMake 编译方式，**CMakeList.txt** 需在 **SDK** 的一级目录下（打开.zip 压缩包后即可看见 **CMakeLists.txt** 文件）。

1.2 平台编译选手源码时，会使用压缩包中的 CMakeLists.txt 文件，选手需要保证工程编译正常。

1.3 C++语言支持 C++17 规范；C 语言支持 C10 规范；

**1.4 不可引用第三方库；**

## 2. Java

2.1 系统支持 JDK1.8 版本，请在此版本下进行源码开发；

2.2 文件 Main.java 的名称不得进行修改，否则会编译失败；

**2.3 不可引用第三方库；**

## 3. Python

3.1 系统支持 Python3.7.3 版本，请在该版本下进行源码开发；

3.2 参赛选手提交的源码会在系统上直接运行，因此请保证 `main.py` 在一级目录下，不可修改该文件名称。

### 3.3 不可引用第三方库；

#### 4. 其他：

4.1 程序运行时间最长为 5 分钟，包含判题器运行时间，不包含编译时间。判题器运行时间和选手的行为相关，最长运行约为 80 秒。

4.2 压缩包中不可带有任何运行时调用的文件，包括但不限于可执行程序、动态库、数据表等。

4.3 平台在运行选手程序之前会清理所有文件，只保留最终程序，故运行时对压缩包中的文件调用均会失败。

4.4 上传源码压缩包中所包含的目录不得含有以下合法字符集以外的任何字符；目录名合法的命名字符集：英文大写字母“A-Z”、英文小写字母“a-z”、数字“0-9”、英文下划线“\_”。

4.5 上传到源码压缩包中所包含的文件名不得含有以下合法字符集以外的任何字符，且“.”符号不得连续出现。文件名合法的命名字符集：英文大写字母“A-Z”、英文小写字母“a-z”、数字“0-9”、英文下划线“\_”、英文点“.”。

4.6 选手程序的编译运行均在 linux 下，提交前保证在 linux 平台下编译运行成功，以免影响参赛成绩。

4.7 选手程序没有写权限，选手提交代码前应去掉写操作，否则会造成运行失败。

4.8 请勿利用漏洞参加大赛，发现漏洞请及时联系组委会。

# 2 判题器说明

## 2.1 下发的示例程序 Demo 及脚本 run.py

为了方便选手快速上手本题目，大赛提供了 demo 示例程序

- 提交 demo 示例程序可以获得少量的分数。
- 选手可以参考 demo 示例程序的输入输出。
- demo 示例程序不保证在任意数据下均可以运行出合法解。

同时，大赛提供了 run.py，方便选手在本地与判题器交互。run.py 支持在 windows，linux，macos 中运行。可以在终端中输入以下命令以拉起判题（需要预先安装 python）：

```
1. python run.py [interactor_path] [data_path] [player_path]
```

其中，interactor\_path 为大赛提供的判题器位置，data\_path 为数据文件位置，player\_path 为运行选手程序的命令（多个空格隔开的参数需要用引号括起来）。

具体的，你可以这样运行 demo 程序和 run.py（以 windows 系统举例）：

对于 C 语言

```
1. gcc demos\c\main.c -o demos\c\main.exe
2. python run.py interactor\windows\interactor.exe data\sample.in demos\c\main.exe
```

对于 C++语言

```
1. g++ demos\cpp\main.cpp -o demos\cpp\main.exe
2. python run.py interactor\windows\interactor.exe data\sample.in demos\cpp\main.exe
```

对于 java 语言

```
1. javac demos\java\Main.java
2. python run.py interactor\windows\interactor.exe data\sample.in "java -cp demos\java\ Main"
```

对于 python 语言

```
1. python run.py interactor\windows\interactor.exe data\sample.in "python demos\python\main.py"
```

## 2.2 判题器异常判定与处理

在交互过程中，判题器会在发现选手程序出错的第一时间报错，并终止自身进程。相关错误会打印到屏幕和 `result.txt` 中，选手可以自行查看。

判题器会将交互结果输出到 `interactor` 可执行文件目录下的 `result.txt`，格式为 json。

错误信息一定包含以下字段：

<code>error_code</code>	错误码，简单表示交互的结果。交互成功时错误码为 <code>interaction_successful</code> ，其余错误码均表示交互失败。
<code>score</code>	得分，交互失败时得分为 0

可能包含以下字段：

<code>timestamp</code>	当前时间片
<code>action</code>	操作，包括 <code>init</code> ， <code>timestamp</code> ， <code>delete</code> ， <code>write</code> ， <code>read</code> 等等
<code>role</code>	<code>data</code> 表示数据出错， <code>player</code> 表示选手输出出错
<code>request</code>	请求编号
<code>object</code>	对象编号
<code>disk</code>	硬盘编号
<code>unit</code>	硬盘存储单元编号
<code>message</code>	更具体的错误信息

例如，以下为一个判题器报错的具体例子：

如图所示，`result.txt` 文件中记录的判题结果采用 JSON 格式呈现，其结果整理后如下：

```
{
    "error_code": "unit_is_used",
    "score": "0.0000",
    "timestamp": "1",
    "action": "write",
    "role": "player",
    "object": "2",
    "disk": "3",
    "unit": "4",
    "message": "The unit has been used by object 1."
}
```

本次错误场景为存储单元已被占用（`error_code: unit_is_used`）的异常情况。具体表现为在第 1 个时间片（`timestamp: 1`），进行写入操作（`action: write`）交互时，选手程序（`role: player`）将对象 2（`object: 2`）放入硬盘 3（`disk: 3`）的存储单元 4（`unit: 4`）中，但目标存储单元已被对象 1 占用（`message`），导致操作失败。根据比赛规则，此类异常场景的得分为 0（`score: 0.0000`）。

## 2.3 debug 模式说明

debug 模式为方便选手调试代码的模式，其主要功能为打印交互过程以及打印硬盘磁头信息。

选手可通过在运行命令中添加 `-d` 或 `--debug` 参数启用 debug 模式。该模式会将交互过程记录至当前运行目录下的 `debug.txt` 文件中。

debug 模式支持 0-2 个附加参数，参数格式为：

1. `-d [<arg1> [<arg2>]]`

其中，`arg1` 表示起始时间片，`arg2` 表示终止时间片。例如：

1. `python run.py [interactor_path] [data_path] [player_path] -d`
2. `python run.py [interactor_path] [data_path] [player_path] -d 3`
3. `python run.py [interactor_path] [data_path] [player_path] -d 2 4`

分别表示打印所有交互过程，打印时间片 3 后的交互过程，打印时间片 2-4 的交互过程

注意事项：

- 请确保对运行目录具有写入权限
- 示例中路径参数 `[interactor_path]`、`[data_path]`、`[player_path]` 需替换为实际路径
- 调试信息将逐行写入 `debug.txt` 文件，在程序正常退出后将剩余信息写入 `debug.txt` 文件
- 调试模式性能较差，且会使用较多硬盘空间。

`debug.txt` 中部分输出如图所示：

1. `[interactor]`
2. `TIMESTAMP 2`
3. `[player]`
4. `TIMESTAMP 2`
5. `[interactor]`
6. `.....`
7. `[player]`
8. `#`
9. `#`
10. `r#`
11. `[debug]`
12. `Disk head position: [1]1 [2]1 [3]2`
13. `[player]`
14. `.....`

其中：

`[interactor]`：表示判题器的输出内容。

`[player]`：表示选手程序的输出内容。

`[debug]`：表示额外的调试信息：该时间片的磁头位置信息。



## 2.4 replay 模式说明

replay 模式方便选手查看某个时间片的硬盘布局信息

选手可以在 run.py 后加入 -r 或者 --replay 参数打开 replay 模式，将某一时间片的布局等信息打印到运行目录下 replay/replay\_<时间片>\_<真实时间戳>.txt，再使用可视化文件加载 replay 文件，可以查看某些图像信息（见 2.5 可视化程序说明）。

replay 模式支持 1~10 个附加参数，参数格式为：

1. -r [<arg1> [<arg2> [...]]]

其中 arg 表示要打印的时间片编号。例如：

1. python run.py [interactor\_path] [data\_path] [player\_path] -r 5 10

表示打印第 5 个和第 10 个时间分片的信息到两个 replay 文件中。

若不输入 arg1 则默认打印最后一个时间片。

注意事项：

- 请确保对运行目录具有写入权限。
- 请勿修改 replay 的文件信息，否则可视化程序可能会报错。

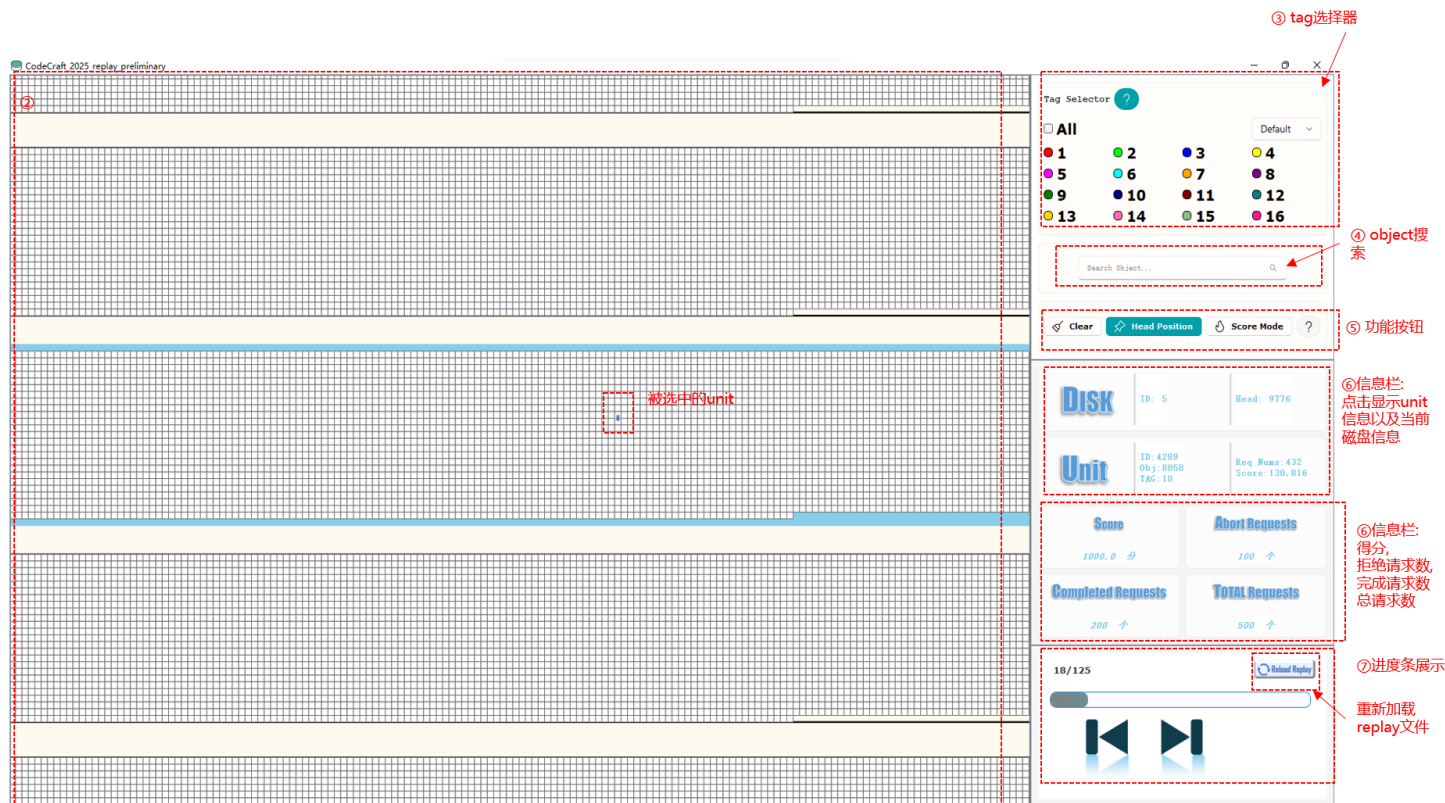
其他：

replay 模式可以和 debug 模式混合使用，例如：

1. python run.py [interactor\_path] [data\_path] [player\_path] -d 1 2 -r 3

## 2.5 可视化程序说明

为了方便选手更好的理解编程赛题，大赛提供了题目可视化程序，将赛题的运行过程和结果以直观的图形化形式展示出来。选手可以借助可视化程序快速定位问题，优化算法，更高效地完成比赛。



运行方式:

- windows: 点击运行 `code_craft_2025.exe`
- macos: 点击运行 `code_craft_2025.app`
- linux: 点击运行二进制文件 `code_craft_2025`

可视化程序由以下模块组成:

1. 标题界面，可以加载 `replay` 文件。
2. 硬盘布局，根据数据生成，代表每个硬盘的对象分布情况。
3. 对象标签选择器，支持多选，选中的标签会在 2 号区域显示出来。
4. 搜索框，按照对象编号搜索某个对象。
5. 功能按钮，切换为得分模式后，可以查看当前未成功的读取请求的得分分布情况。
6. 信息栏，展示选手程序的部分信息。
7. 进度条，可以在此处加载多个 `replay` 文件，拖动进度条查看不同时间片的分布状态。

此外，点击某个存储单元，会显示当前单元上的部分信息。

# 3 数据说明

---

大赛初赛分为两个阶段：

阶段一（练习赛阶段）：

大赛提供 `sample.in`，`sample_practice.in`。线上评测使用数据 `practice.in`，该数据不下发。

其中：

`sample.in` 为样例数据，其内容与题面中样例数据一致。

`sample_practice.in` 为线上评测数据以对象为粒度 **随机抽样** 的后的数据。其对应的 **硬盘空间**，**令牌数** 会按数据比例和数据合法性进行调整。

`practice.in` 为线上评测数据，练习赛阶段的得分和排名会使用该数据的评测结果。

阶段二（正式赛阶段）：

大赛会在正式赛开启后提供 `sample_official.in`。线上评测使用数据 `official.in`，该数据不下发。

`sample_official.in` 为线上评测数据 `official.in` 对象为粒度 **随机抽样** 的后的数据。其对应的 **硬盘空间**，**令牌数** 会按数据比例和数据合法性进行调整。

`official.in` 为线上评测数据，正式阶段的得分和排名会使用该数据的评测结果。请注意，正式赛的结果和排名是初赛晋级复赛的唯一标准。

排名：

最终排名按照正式赛数据排名，得分四舍五入保留两位小数。若两位选手得分一致，则按照程序运行时间排序，若仍然一致以提交时间排序。

选手得分为该阶段中所有提交中的最大分数。

大赛提供了一个 `sample.in` 的合法解和解释，见附件《2025 华为软件精英挑战赛 样例数据》

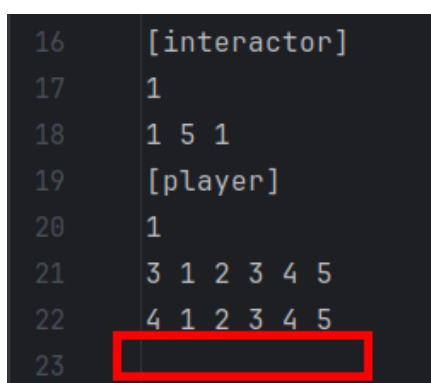
# 4 常见问题 QA

---

**Q:** 运行 `run.py`, 卡住了没有输出怎么办?

**A:** 这种情况一般是交互较慢, 或者交互器和选手程序死等输出的问题。请使用判题器 `debug` 模式, 可以在 `debug` 文件中查看死等位置以及判题器、选手最后的输出, 进而调试程序。

例如:



```
16 [interactor]
17 1
18 1 5 1
19 [player]
20 1
21 3 1 2 3 4 5
22 4 1 2 3 4 5
23
```

选手打开 `debug` 模式后, 发现 `debug.txt` 的输出在此处卡住。此处需要输出写入对象的三个副本的信息, 但是选手仅输出了两个副本, 判题器在等待选手的第三行输出, 从而发生死等。

**Q:** 本地调试程序每次都需要拉起判题器, 有更快的判题方式吗?

**A:** 无需每次都拉起判题器, 选手程序可以直接读取输入文件的数据, 而不是判题器的数据。判题器仅起到交互和及时检查错误的作用。

**Q:** 判题器会因为选手的不同输出给出后续不同的数据吗?

**A:** 不会。判题器的数据均来自于数据文件。

**Q:** 代码在本题编译成功, 在平台上编译失败, 该如何处理?

**A:** 为了防止编译错误泄露数据等, 大赛不提供编译报错具体信息。如果确实需要, 在确保本地编译无误的情况下, 选手可以联系大赛主办方查询。

**Q:** 其他题目问题应该如何反馈?

**A:** 选手可以在论坛上提问。