

目录

01

基础阶段

02

项目阶段

03

进阶阶段

The screenshot shows an IDE window with the following components:

- Editor:** Displays Rust code in `operators.rs`. The code includes functions `swiglu` and `matmul_transb`. Comments in Chinese explain the operations and provide hints. The `matmul_transb` function performs matrix multiplication with various assertions for dimensions.
- Run Console:** Shows the output of running tests. It indicates that 5 out of 5 tests passed in 16 milliseconds. It also displays warnings generated by the compiler and the completion of the build process.
- Bottom Bar:** Shows the current file path `learning-lm-rs > src > operators.rs`, the status `正在提交: 正在分析代码...`, and various settings like `(D) 1323 B/s / 1803 B/s`, `Cargo Check`, `3:1`, `LF`, `UTF-8`, `4个空格`, `x86_64-unknown-linux-gnu`, and a `非商业使用` (Non-commercial use) label.

完成基础`5`个测试

The screenshot shows an IDE window titled "learning-lm-rs" with several tabs: "model.rs", "main.rs", "operators.rs", "params.rs", and "kvcache.rs". The "model.rs" tab is active, displaying Rust code for a Llama model implementation. The code includes a `generate` function that uses a `Tokenizer` to process input and generate output tokens. The output is displayed in the bottom panel, showing a story about a girl named Lily.

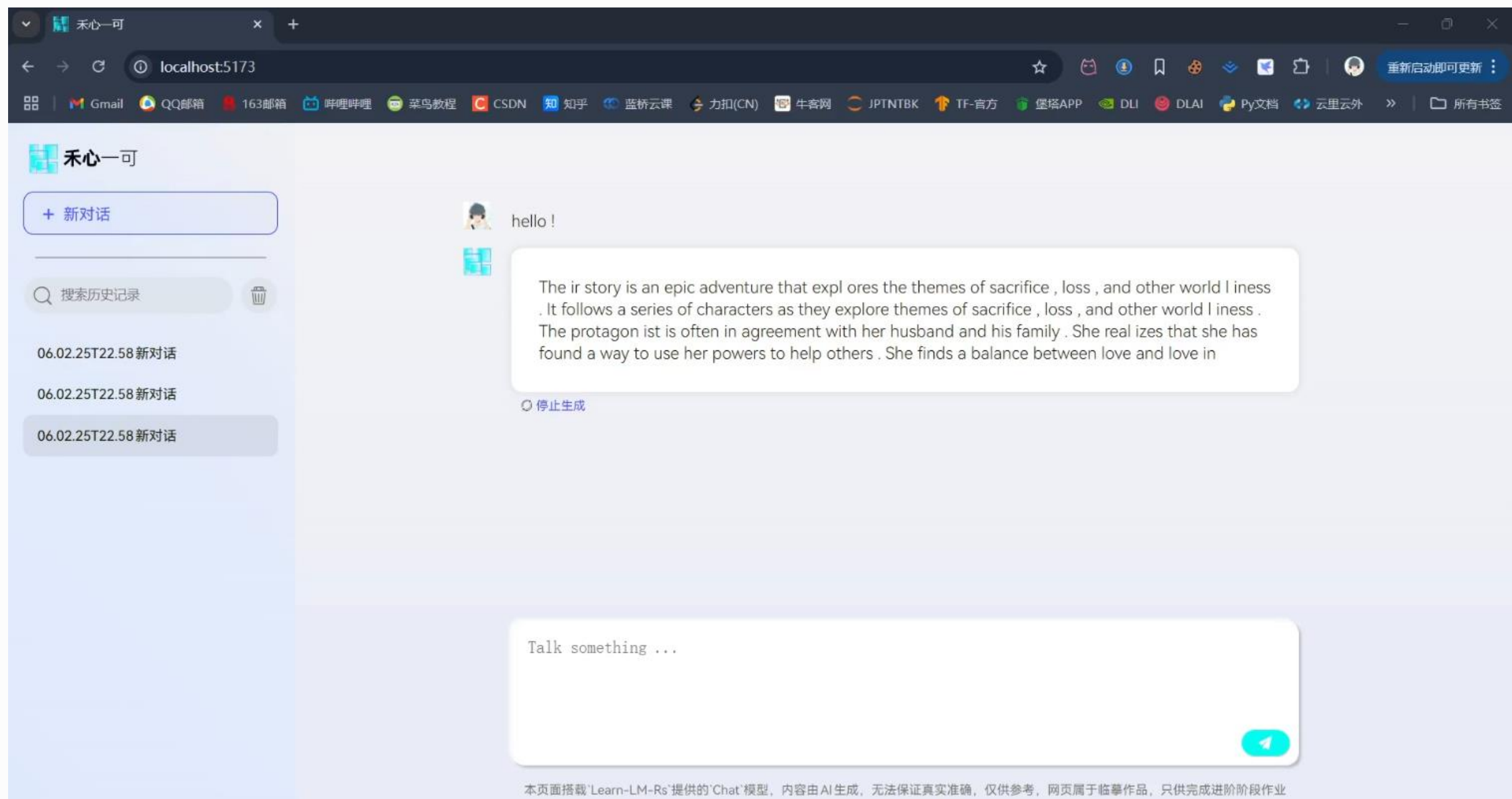
```
55 impl Llama<f32> {
171     pub fn generate(
189         while result.len() < max_len {
204             input = vec![new_word_id];
205
206             // // 调试-B
207             use std::path::PathBuf;
208             use tokenizers::Tokenizer;
209             let project_dir :&str = env!("CARGO_MANIFEST_DIR");
210             let model_dir :PathBuf = PathBuf::from(project_dir).join(path: "models").join(path: "story");
211             let tokenizer :Tokenizer = Tokenizer::from_file(model_dir.join(path: "tokenizer.json")).unwrap();
212             // println!(
213             //     "new word id: [{}], curr length: [{}], new word is [{}]",
214             //     new_word_id, result.len(), tokenizer.decode(&[new_word_id], true).unwrap()
215             // );
216             // 逐步输出-B
217             flush_print!("{}", tokenizer.decode(&[new_word_id], true).unwrap());
218             // 逐步输出-E

```

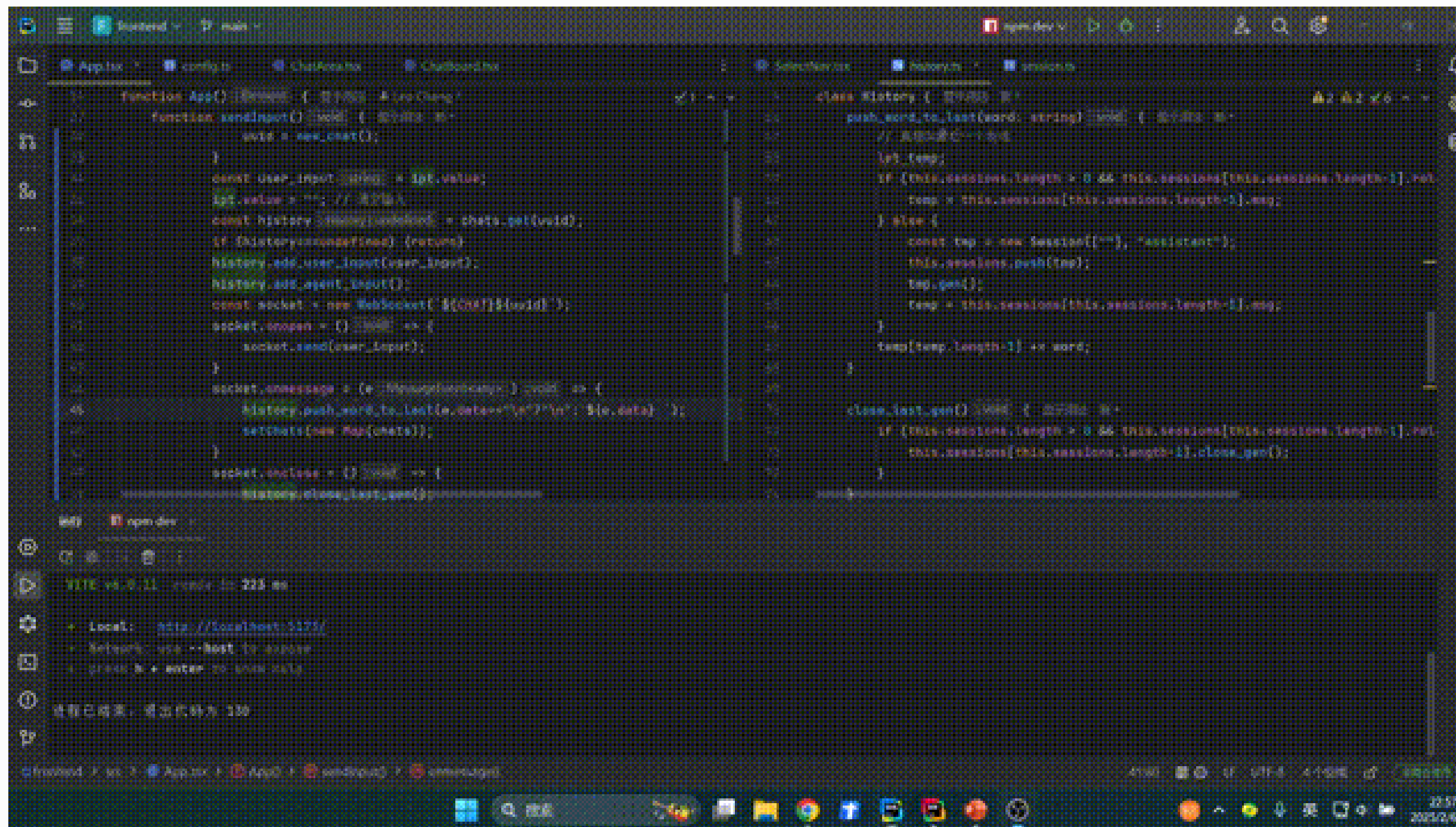
The output panel shows the following text:

```
Once upon a time
cone of the otter, there was a generous little girl named Lily. Lily loved to play and learn every day.
One day, Lily saw a little girl walking neara big rock. Lily's mom had a friend named Tim. Tim said, "Wow, Lily. But I want to read, just so going to poke!"
Lily and Tim went to her friend, Sara. They looked at the box. Lily said to the cars, but they didn't know what happened. Sara looked scared, but they couldn't say sorry. Mom said,
"Okay, Lily. Let's keep climbing the tree." Mom said, "Let's count our big ball."
They played with the toy around and roared until they were diffiered. She went inside and looked at her mind. Tim started to come and started to draw. They laughed and laughed. Lil
y was very happy and said, "Oh sorry, Tom!"
From that day on, Lily went to see. And she also thoughtabout the west. The moral of the story is that when we still find it.<|end_story|>
进程已结束，退出代码为 0
```

完成基础故事任务



完成UI – 成品展示



完成UI+前后端对接 - 成品展示（全损画质GIF）

支持多会话以及历史回滚等思路说明

前置说明:

Fontend: 采用React+Less开发单页面应用

Wasmend: 暂时弃用的方案, 详见后记 (最后一页)

Backend: 使用Rocket开发的Rust后端程序

Backend相比于原learn-lm-rs新增 (主要):

options.rs : GLOBAL_LLAMA、GLOBAL_TOKEN、struct History、HISTORY
message.rs : struct Role、Session、Message、Prompt、SESSION
chat_iter.rs : struct ChatIter
restful_api.rs : WebSocket接口、以及其他接口 (计划功能未全部接入)

全局变量说明:

GLOBAL_LLAMA、GLOBAL_TOKEN是类似于`以下`声明的全局复用的变量

OnceCell<Arc<RwLock<Tokenizer>>>、OnceCell<Arc<RwLock<Llama<f32>>>>

HISTORY、SESSION一个是uuid: kvcache的映射表, 一个是uuid: session的映射表:

HISTORY: Lazy<HashMap<String, Arc<History>>> = Lazy::new(|| HashMap::new());
struct History: cache: Arc<RwLock<KVCache<f32>>>, pass_len: Arc<RwLock<usize>>>,

message.rs其余主要变量均为格式化数据利用as_str以及fmt生成格式化输入的

支持多会话以及历史回滚等思路说明

思路说明： 在前置说明的基础上

多会话： 本质是利用 `HashMap<String, ...>` 来利用 `uuid: String` 获取历史对话数据 `Session` 和 `KVC-History` 另外，`WebSocket` 的接口也是基于 `uuid` 继续会话的： `#[get("/c/<uuid>")]`。多记录实际上只有最后一条记录有效（即：AI 在同一个历史进度生成多次），因为多分支成本大，所以只根据最后一条继续推即可。

对话推理： 刚开始想使用生成器： `coroutines/yield` 但是总会有奇奇怪怪的错误，就留下了一个弃用方案的函数： `options.rs/ async_inference`。后续成功的方案是 `chat_iter.rs/ ChatIter` 迭代器代替弃用的生成器。

迭代器： 传入 `uuid`，用户输入，`top_p`，`top_k` 等信息创建 `ChatIter` 迭代器，迭代器会根据 `uuid` 在全局两个历史池里寻找（或者找不到就地新建）历史记录，之后通过滑动窗口剪切对话数据到合适大小后推理

滑动窗口： 使用指定大小之内的对话信息生成模板字符串，起码保证此次推理生成时不会触发 `KVC` 的溢出。而当滑动窗口剪裁对话信息的时候也意味着历史 `KVC` 不再有效，所以长度置0，在 `kvcache` 函数中为：
`kvcache.refresh()`

历史回滚： `Session`（文本对话信息）回滚：最后一条是AI，那删掉就好。`KVC-History`回滚：如果回滚前一次对话 `Session` 被裁剪过，则上次的 `KVC` 不再有效，直接 `refresh` 清空，重新推理，其余情况即还有效，则使用上一次推理前记录的 `KVC` 的 `length`（在 `History.pass_len` 记录），位置的脏位数据计算到时自会覆盖

需要改进

前端的对话名字使用`Date`对象+新对话`生成，之后需要加一个改名字的接口

还有较多的细节API未接/未写，比如：发送按钮、历史重置的按钮在生成状态下应该改变icon等等

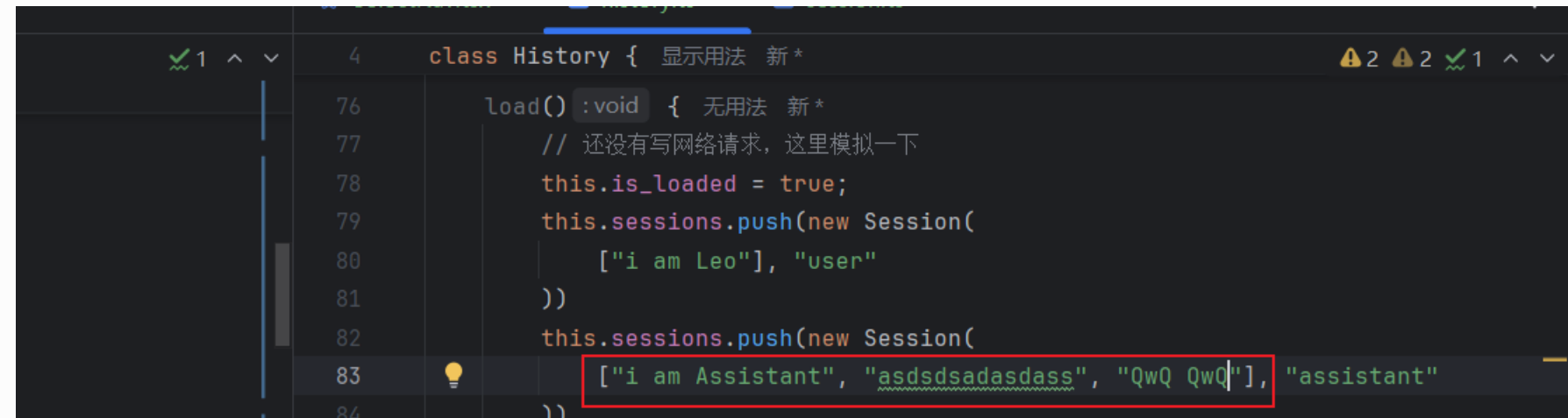
之后有时间把加载模型参数那里改为根据torch.type来混合精度加载，泛型推理，Tensor.rs可以加一个dtype等等

没有使用时间戳和浏览器提供的Storage/DB存储对话数据

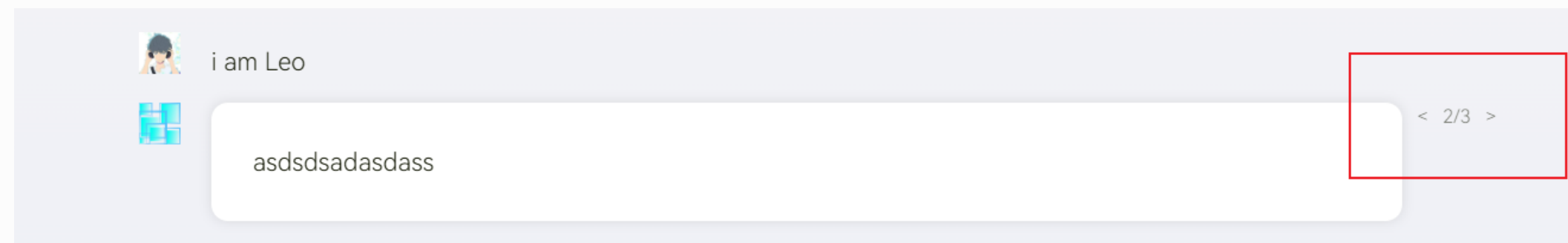
只实现了主要功能，细节一概全部.....QwQ

前端UI在后端生成\n后没有及时替换为：

本地推理的逻辑仅仅是web方案的debug: `fn main() { loop { io::stdin().read_line(&mut ipt) inference(uuid, ipt...) } }`



```
class History {  
    load() :void {  
        // 还没有写网络请求，这里模拟一下  
        this.is_loaded = true;  
        this.sessions.push(new Session(  
            ["i am Leo"], "user"  
        ));  
        this.sessions.push(new Session(  
            ["i am Assistant", "asdsdsadasdass", "QwQ QwQ"], "assistant"  
        ));  
    }  
}
```



后记

原计划使用Rust: WasmPack部署至浏览器本地部署，之后Js负责利用浏览器提供的Storage/IndexedDB接口保存对话文本数据，加载远程模型，或者打开本地模型，Rust端负责维护KVC，KVC与前端使用Map管理的完整对话文本数据的对应是Uuid。对话溢出的处理方式和现在的前后端项目一致，都是滑动窗口空出足够的Space + RefreshKVC。但是本着“先完成，再完美”，先完成了前后端的简易框架，仍然有Wasm的残留代码，这将作为一个Release。对比wasm残留的代码可以发现，无非就是#[get("/xxx")]换为#[wasm_bindgen]功能逻辑依然一致。

后续的代码将会完全移除前后端，将变为：每次提交到rls分支，CI使用wp编译wasm -> 移动到前端src/wasm, vite编译静态界面，最后部署静态界面到gh-page。这样子每次打开将会远程加载git lfs的模型数据，或者也可以本地选择模型打开进行推理，相关的代码现在wasm文件夹依然有残留。

项目“**禾心一可**”名字起源是自己名字对半拆分重新组合：“**禾可**”；“**心一**”是因为国产大模型“X心一X”的命名格式：文心，蓝心.....仅仅玩梗，切勿上纲上线.....
