Ling （灵）

AI 流式结构化输出解决方案

bearbobo.com

01 为什么设计 Ling 框架

02 Ling 框架的核心能力

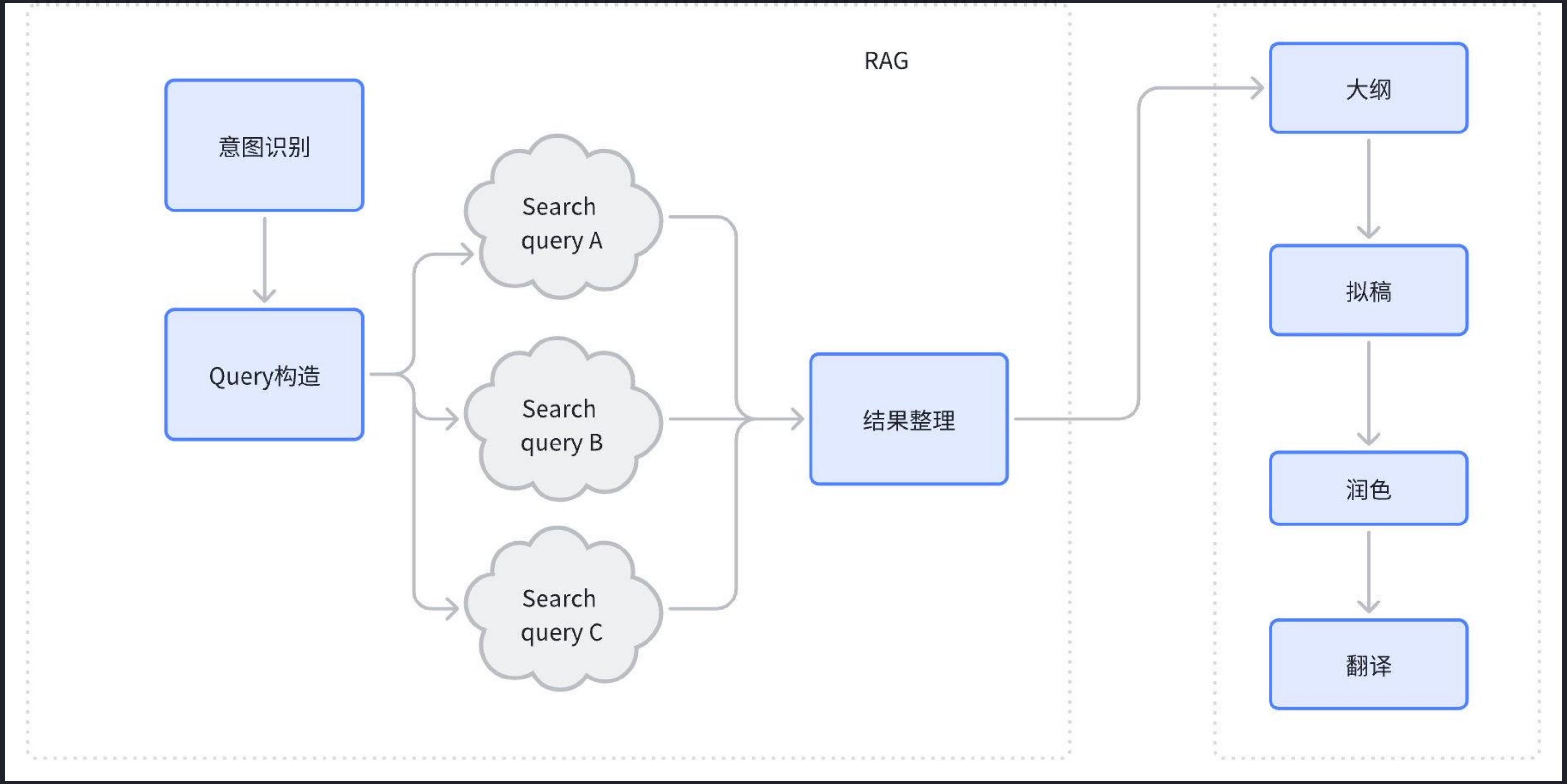
03 如何使用和扩展 Ling

01 为什么设计 Ling 框架

JSON 对于 AIGC 的作用非同一般

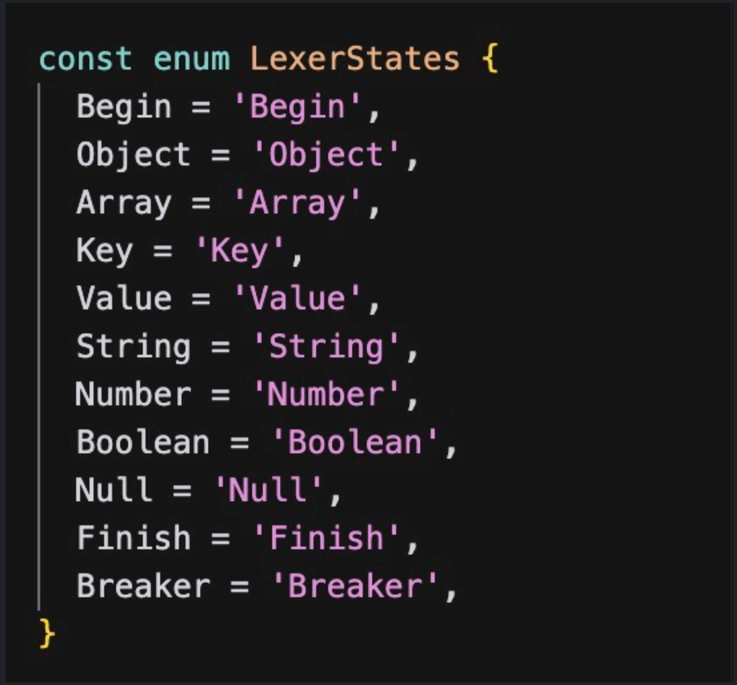
看例子 ->

并行实时工作流数据处理的要求

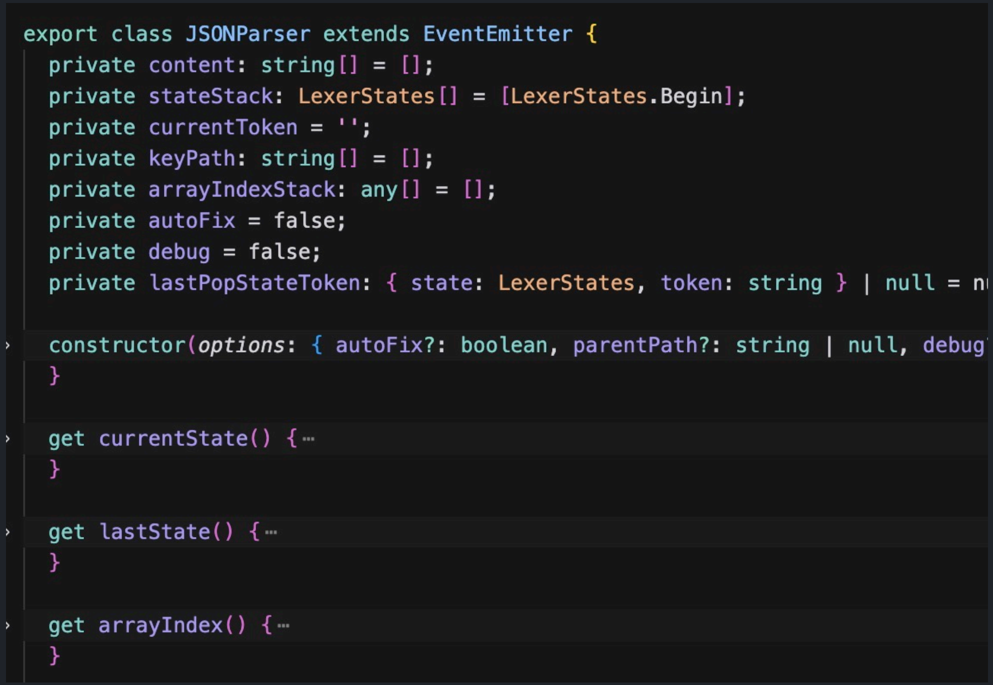


JSON 对于数据实时处理的劣势？

流式实时解析 JSON



```
const enum LexerStates {
  Begin = 'Begin',
  Object = 'Object',
  Array = 'Array',
  Key = 'Key',
  Value = 'Value',
  String = 'String',
  Number = 'Number',
  Boolean = 'Boolean',
  Null = 'Null',
  Finish = 'Finish',
  Breaker = 'Breaker',
}
```



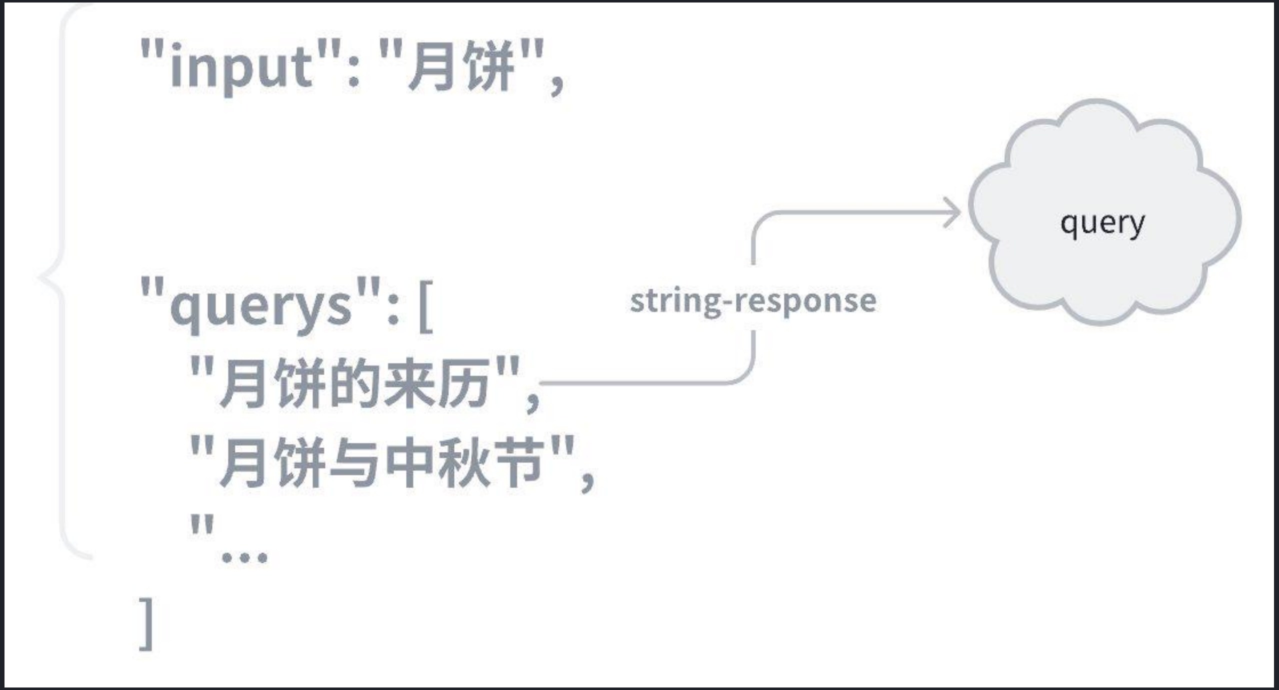
```
export class JSONParser extends EventEmitter {
  private content: string[] = [];
  private stateStack: LexerStates[] = [LexerStates.Begin];
  private currentToken = '';
  private keyPath: string[] = [];
  private arrayIndexStack: any[] = [];
  private autoFix = false;
  private debug = false;
  private lastPopStateToken: { state: LexerStates, token: string } | null = nu

  constructor(options: { autoFix?: boolean, parentPath?: string | null, debug
  }

  get currentState() {…
  }

  get lastState() {…
  }

  get arrayIndex() {…
  }
```

核心特性：

1. 输入 token 实时解析
2. 流式输出 JSONURI
3. 自动补全和错误修复

例子 ->

支持 SSE 实时流式输出



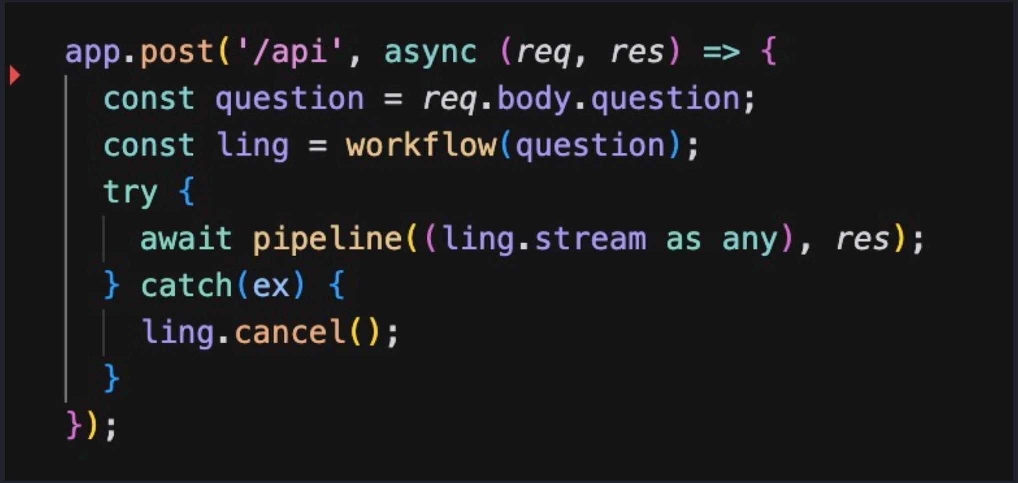
事件与生命周期：

1. message
2. string-response
3. inference-done
4. response
5. error

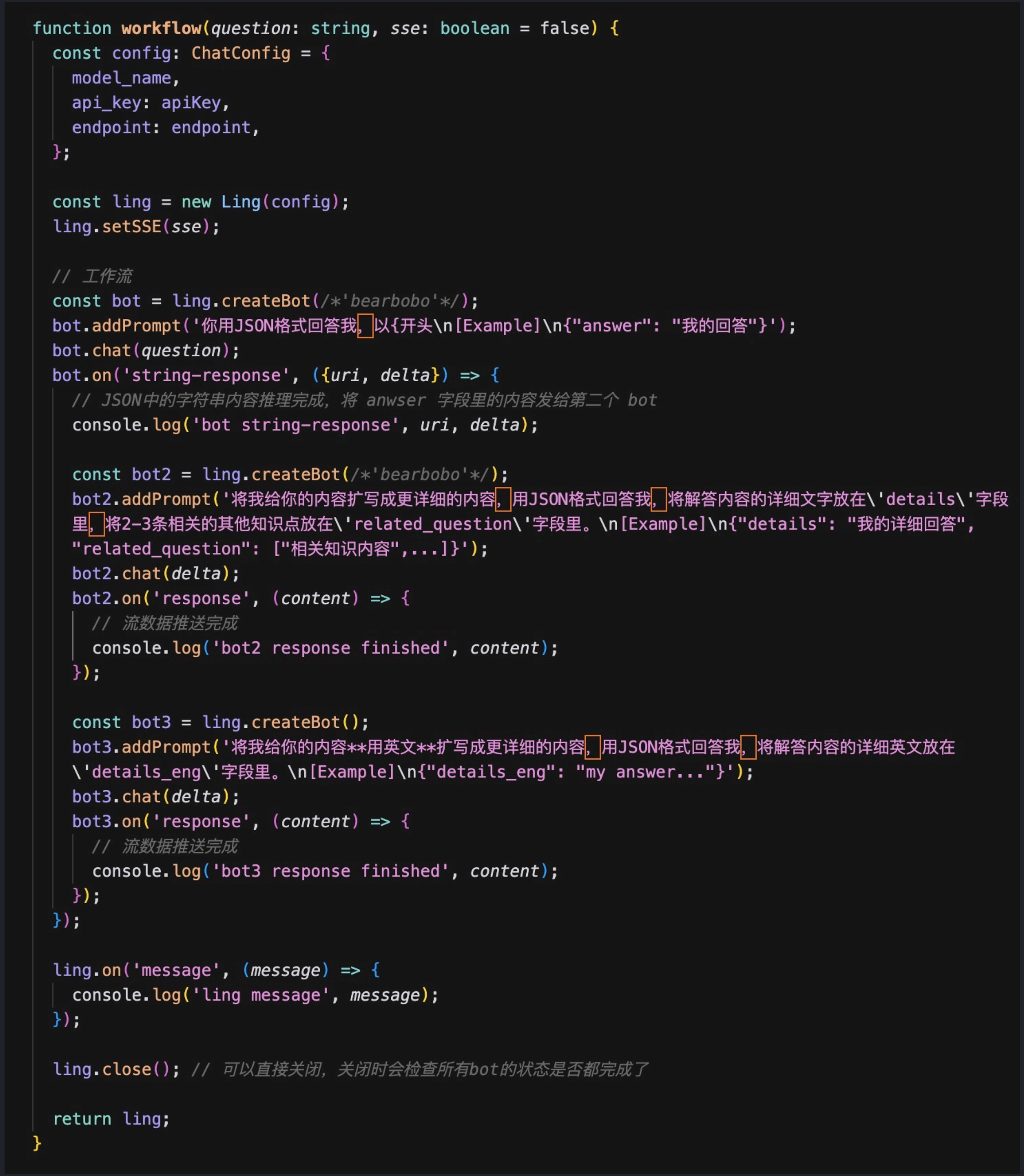
03 如何使用和扩展 Ling

基础用法：

1. 异步工作流
2. HTTP 返回 stream 对象



```
app.post('/api', async (req, res) => {
  const question = req.body.question;
  const ling = workflow(question);
  try {
    await pipeline((ling.stream as any), res);
  } catch(ex) {
    ling.cancel();
  }
});
```

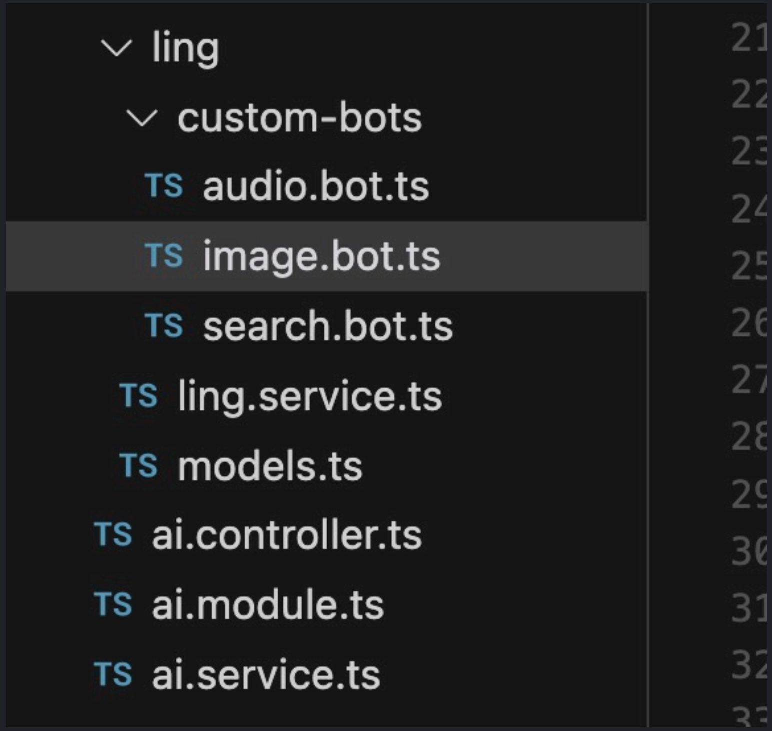


```
function workflow(question: string, sse: boolean = false) {
  const config: ChatConfig = {
    model_name,
    api_key: apiKey,
    endpoint: endpoint,
  };

  const ling = new Ling(config);
  ling.setSSE(sse);

  // 工作流
  const bot = ling.createBot(/*'bearbobo'*/);
  bot.addPrompt('你用JSON格式回答我，以{开头\n[Example]\n{"answer": "我的回答"}');
  bot.chat(question);
  bot.on('string-response', ({uri, delta}) => {
    // JSON中的字符串内容推理完成，将 anwser 字段里的内容发给第二个 bot
    console.log('bot string-response', uri, delta);

    const bot2 = ling.createBot(/*'bearbobo'*/);
    bot2.addPrompt('将我给你的内容扩写成更详细的内容，用JSON格式回答我，将解答内容的详细文字放在\'details\'字段
    里，将2-3条相关的其他知识点放在\'related_question\'字段里。\n[Example]\n{"details": "我的详细回答",
    "related_question": ["相关知识内容",...]}');
    bot2.chat(delta);
    bot2.on('response', (content) => {
      // 流数据推送完成
      console.log('bot2 response finished', content);
    });

    const bot3 = ling.createBot();
    bot3.addPrompt('将我给你的内容**用英文**扩写成更详细的内容，用JSON格式回答我，将解答内容的详细英文放在
    \'details_eng\'字段里。\n[Example]\n{"details_eng": "my answer..."}');
    bot3.chat(delta);
    bot3.on('response', (content) => {
      // 流数据推送完成
      console.log('bot3 response finished', content);
    });
  });

  ling.on('message', (message) => {
    console.log('ling message', message);
  });

  ling.close(); // 可以直接关闭，关闭时会检查所有bot的状态是否都完成了

  return ling;
}
```

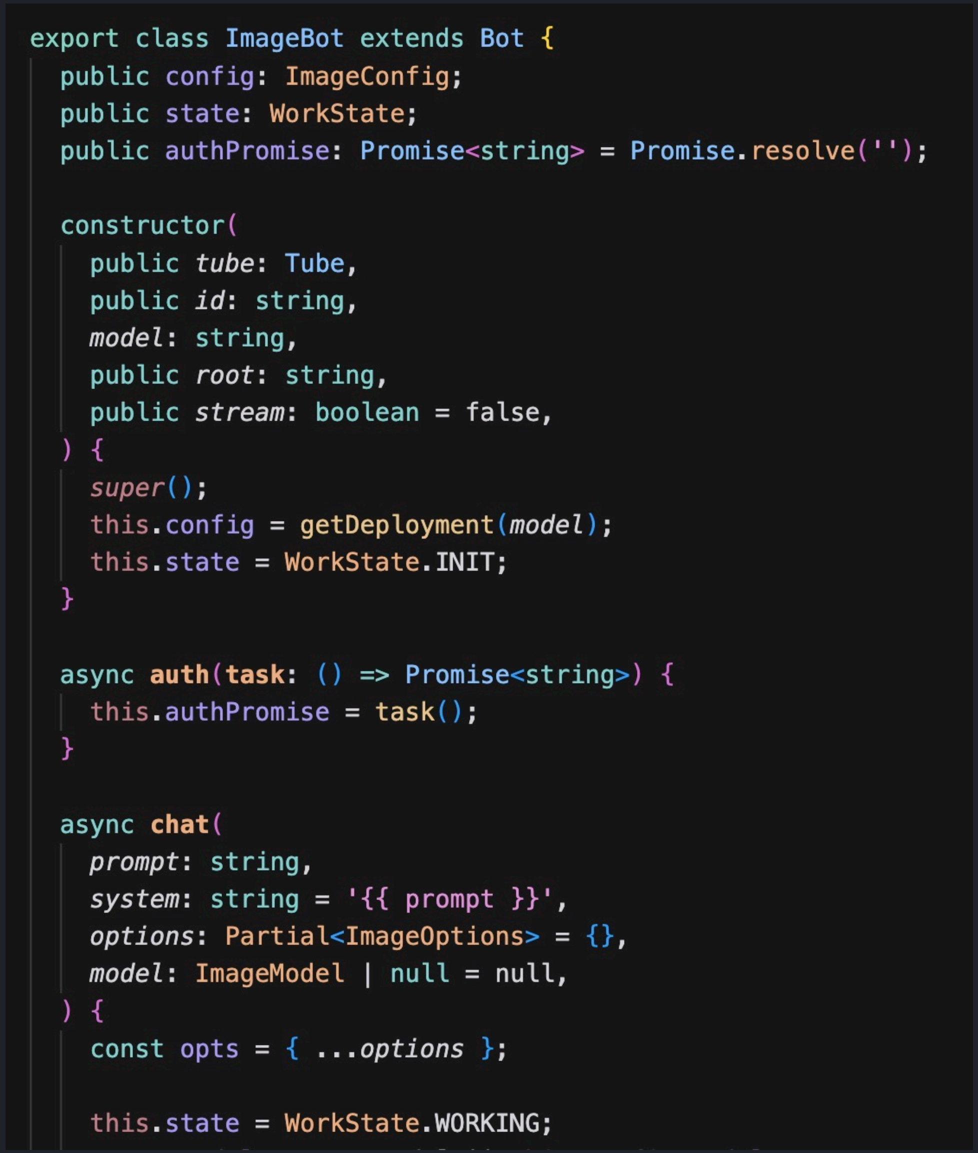
03 如何使用和扩展 Ling

高级用法：

1. 扩展 Bot



```
ling
  custom-bots
    audio.bot.ts
    image.bot.ts
    search.bot.ts
  ling.service.ts
  models.ts
ai.controller.ts
ai.module.ts
ai.service.ts
```



```ts
export class ImageBot extends Bot {
  public config: ImageConfig;
  public state: WorkState;
  public authPromise: Promise<string> = Promise.resolve('');

  constructor(
    public tube: Tube,
    public id: string,
    model: string,
    public root: string,
    public stream: boolean = false,
  ) {
    super();
    this.config = getDeployment(model);
    this.state = WorkState.INIT;
  }

  async auth(task: () => Promise<string>) {
    this.authPromise = task();
  }

  async chat(
    prompt: string,
    system: string = '{{ prompt }}',
    options: Partial<ImageOptions> = {},
    model: ImageModel | null = null,
  ) {
    const opts = { ...options };

    this.state = WorkState.WORKING;
```

https://ling.bearbobo.com/

感谢聆听