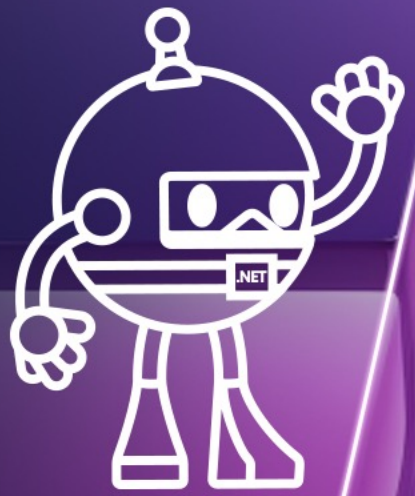
.NET Conf China 2023

2023/12/16
09:30 - 18:00

中国・北京



.NET中文社区

中国 · 北京

.NET Conf China 2023

Handlebars 模板
在 .NET 和 Semantic Kernel 中的应用

衣明志

基普智能 创始人
Semantic Kernel 中文技术社区发起人

Handlebars 是什么？

- Handlebars 是一个轻量级的模板语言。
- 允许开发者使用模板和输入数据来动态生成 HTML 或其他文本格式的内容。
- Handlebars 模板看起来像常规的文本，但是它带有嵌入式的 Handlebars 表达式 。


```
<p>{{firstname}} {{lastname}}</p>
```

Handlebars的 一个简单的例子

```
<p>{{firstname}} {{lastname}}</p>
```

```
{
  firstname: "Yehuda",
  lastname: "Katz",
}
```

```
<p>Yehuda Katz</p>
```

Handlebars 在 javascript 中的简单使用

Handlebars

```
<!-- Include Handlebars from a CDN -->
<script src="https://cdn.jsdelivr.net/npm/handlebars@latest/dist/handlebars.js">
</script>
<script>
  // compile the template
  var template = Handlebars.compile("<p>{{firstname}} {{lastname}}</p>");
  // execute the compiled template and print the output to the console
  console.log(template({ firstname: "Yehuda",lastname:"Kat" }));
</script>
```

Handlebars 在 .NET 中的简单使用

Handlebars

```
// dotnet add package Handlebars.Net

using HandlebarsDotNet;

var template = Handlebars.Compile("<p>{{firstname}} {{lastname}}</p>");

Console.WriteLine(template(new {firstname = "Yehuda",lastname = "Katz"}));
```

GitHub https://github.com/Handlebars-Net/Handlebars.Net

有没有一种似曾相识的感觉？

Razor Template

Razor Template

```
<p>@Model.FirstName @Model.LastName</p>
```

如果只是为了给 .NET 增加一种视图模板渲染引擎，是没多大意义的。

我们要讨论的是对.NET 和 Semantic Kernel 的价值

Semantic Kernel

- Semantic Kernel 是一个开源 SDK，可让您轻松构建可以调用现有代码的 AI Agent。
- 作为高度可扩展的 SDK，您可以将 Semantic Kernel 与 OpenAI、Azure OpenAI、Hugging Face 等模型一起使用！
- 通过将现有的 C#、Python 和 Java 代码与这些模型相结合，您可以构建能够回答问题和自动化流程的 AI Agent 。

Semantic Kernel Prompt Template

- Semantic Kernel 提供一种内置的 Prompt Template

Semantic Kernel Prompt Template

```
Hello {{$name}}, welcome to Semantic Kernel!
The weather today is {{weather.getForecast}}.
The weather today in {{$city}} is {{weather.getForecast $city}}.
```

- 这套模板支持包含变量，调用外部函数，以及将参数传递给函数

SK Prompt Template 应对稍微复杂的场景

```
var chat = kernel.CreateFunctionFromPrompt(
    @"{{$history}}
    User: {{$request}}
    Assistant: "
);
```

```
var chatResult = kernel.InvokeStreamingAsync<StreamingChatMessageContent>(
    chat,
    new() {
        { "request", request },
        { "history", string.Join("\n", history.Select(x => x.Role + ": " + x.Content))}
    }
);
```

Handlebars Prompt Template 处理更加复杂的场景

```
var getIntent = kernel.CreateFunctionFromPrompt(
    new()
    {
        Template = @"
        <message role=""system"">Instructions: What is the intent of this request?
        Do not explain the reasoning, just reply back with the intent. If you are unsure, reply with {{choices[0]}}.
        Choices: {{choices}}.</message>

        {{#each fewShotExamples}}
            {{#each this}}
                <message role=""{{role}}"">{{content}}</message>
            {{/each}}
        {{/each}}

        {{#each chatHistory}}
            <message role=""{{role}}"">{{content}}</message>
        {{/each}}

        <message role=""user"">{{request}}</message>
        <message role=""system"">Intent:</message>",
        TemplateFormat = "handlebars"
    },
    new HandlebarsPromptTemplateFactory()
);
```

Handlebars Prompt Template 处理更加复杂的场景

```
// Create choices
List<string> choices = ["ContinueConversation", "EndConversation"];

// Create few-shot examples
List<ChatHistory> fewShotExamples = [
    [
        new ChatMessageContent(AuthorRole.User, "Can you send a very quick approval to the marketing team?"),
        new ChatMessageContent(AuthorRole.System, "Intent:"),
        new ChatMessageContent(AuthorRole.Assistant, "ContinueConversation")
    ],
    [
        new ChatMessageContent(AuthorRole.User, "Thanks, I'm done for now"),
        new ChatMessageContent(AuthorRole.System, "Intent:"),
        new ChatMessageContent(AuthorRole.Assistant, "EndConversation")
    ]
];
```

Handlebars Prompt Template 处理更加复杂的场景

- 使用 Kernel 运行 Prompt
- 在主聊天循环中添加右侧代码，一旦意图达到，循环就可以终止。

```
// Invoke prompt
var intent = await kernel.InvokeAsync(
    getIntent,
    new() {
        { "request", request },
        { "choices", choices },
        { "history", history },
        { "fewShotExamples", fewShotExamples }
    }
);

// End the chat if the intent is "Stop"
if (intent.ToString() == "EndConversation")
{
    break;
}
```

Handlebars Prompt Template 嵌套函数

```
var getIntent = kernel.CreateFunctionFromPrompt(
    new()
    {
        Template = @"
        <message role=""system"">Instructions: What is the intent of this request?
        Do not explain the reasoning, just reply back with the intent. If you are unsure, reply with {{choices[0]}}.
        Choices: {{choices}}.</message>

        {{#each fewShotExamples}}
            {{#each this}}
                <message role=""{{role}}"">{{content}}</message>
            {{/each}}
        {{/each}}

        {{ConversationSummaryPlugin.SummarizeConversation history}}

        <message role=""user"">{{request}}</message>
        <message role=""system"">Intent:</message>",
        TemplateFormat = "handlebars"
    },
    new HandlebarsPromptTemplateFactory()
);
```

Handlebars 还支持其他各种复杂语法

计算上下文

```
{{#with person}}
{{firstname}} {{lastname}}
{{/with}}
```

```
{
  person: {
    firstname: "Yehuda",
    lastname: "Katz",
  },
}
```

条件助手

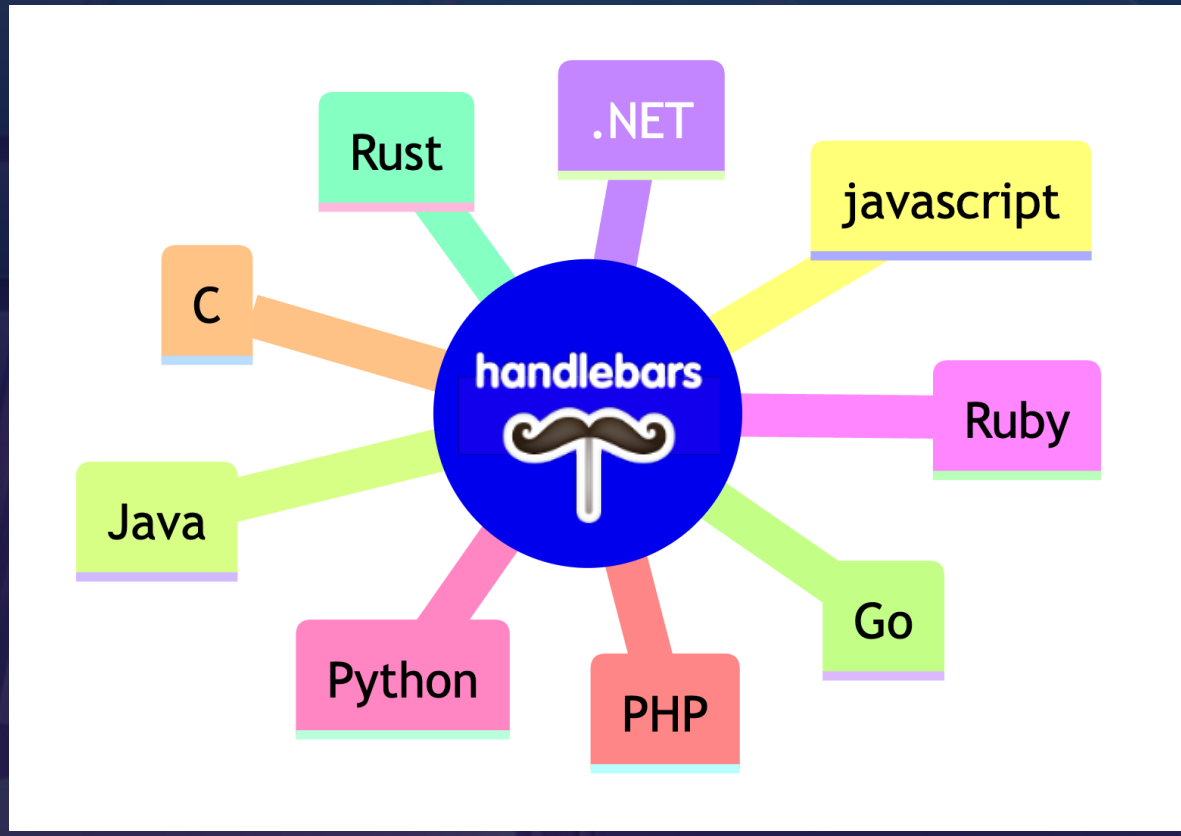
```
<div class="entry">
{{#if author}}
<h1>{{firstName}} {{lastName}}</h1>
{{else}}
<h1>Unknown Author</h1>
{{/if}}
</div>
```

模板注释

```
{{! This comment will not show up in the output}}
<!-- This comment will show up as HTML-comment -->
{{!-- This comment may contain mustaches like }} --}}
```

Handlebars 支持的语言

- Handlebars 最初是设计为在 JavaScript 环境中运行的，
- 不过，Handlebars 的概念和语法已经被多种编程语言和平台所采纳。



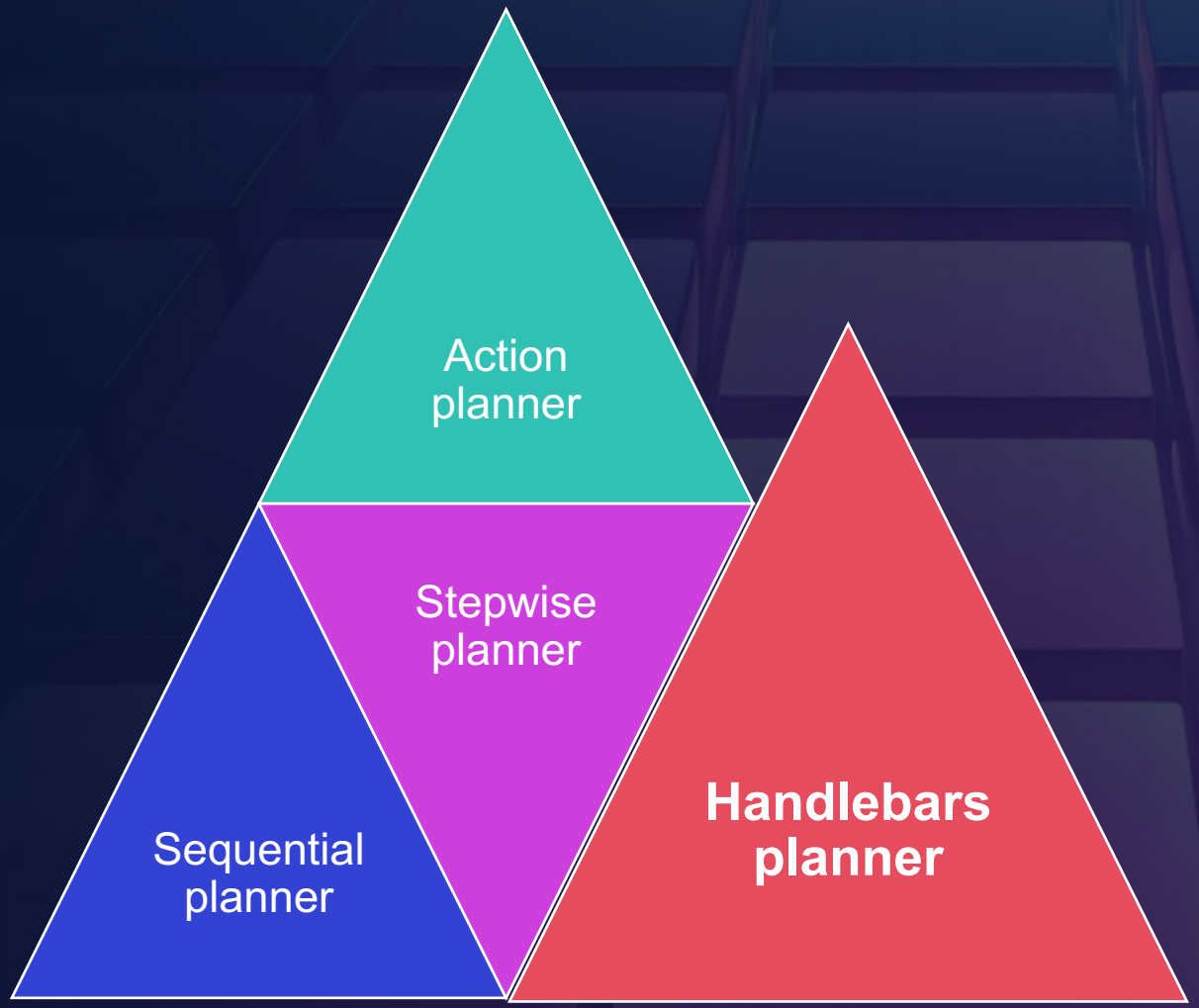
我们再来聊聊 Semantic Kernel 的 Planer

它可能是 Semantic Kernel 中最神奇的东西

Semantic Kernel Planner

- Planner 是一个函数，它接受用户的请求并返回有关如何完成请求的计划。
- 它通过使用人工智能来混合和匹配在 Kernel 中注册的插件来实现这一点，以便它可以将它们重新组合成完成目标的一系列步骤。
- 这是一个强大的概念，它允许您创建原子函数，这些函数可以以您作为开发人员可能没有想到的方式工作。

Semantic Kernel 中现有的 Planner



Action Planner 的原理

```
A planner takes a list of functions, a goal, and chooses which function to use.
For each function the list includes details about the input parameters.
[START OF EXAMPLES]
{{this.GoodExamples}}
{{this.EdgeCaseExamples}}
[END OF EXAMPLES]
[REAL SCENARIO STARTS HERE]
- List of functions:
{{this.ListOfFunctions}}
- End list of functions.
Goal: {{ $input }}
```

```
[EXAMPLE]
- List of functions:
// Read a file.
FileIOPlugin.ReadAsync
Parameter ""path"": Source file.
// Write a file.
FileIOPlugin.WriteAsync
Parameter ""path"": Destination file. (default value: sample.txt)
Parameter ""content"": File content.
// Get the current time.
TimePlugin.Time
No parameters.
// Makes a POST request to a uri.
HttpPlugin.PostAsync
Parameter ""body"": The body of the request.
- End list of functions.
Goal: create a file called ""something.txt"".
{""plan"":{
""rationale"": ""the list contains a function that allows to create files"",
""function"": ""FileIOPlugin.WriteAsync"",
""parameters"": {
""path"": ""something.txt"",
""content"": null
}}}
#END-OF-PLAN
```

Action、Sequential、Stepwise 三个Planner

- 都是基于类似人类常规文本的 Prompt 生成具有一定格式要求的文本返回，再用解释器解析执行。
- 由于基本上都是非常规逻辑代码类文本（json、xml等）生成，加上 AI 的创造性（幻觉），具有相对高一些的不可靠性；需要通过 OneShort 之类的提示词工程来加强输出结果的格式可靠性。
- Planner 的 Prompt 巨为复杂，对 token 的输入消耗较高，对应的使用成本也较高。

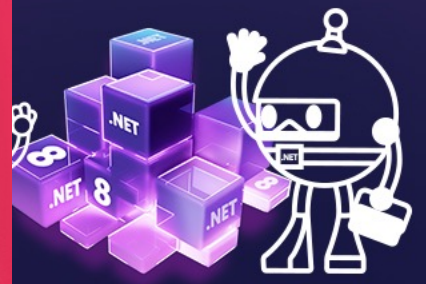
Handlebars Planner 的优势

- Handlebars 是一种具有逻辑的模板语言，介于常规文本和代码文本之间，具有更高的逻辑严谨性，规划的结果相对更加可靠。
- Handlebars 自己的内置助手来编写循环和条件，可以简单处理复杂的规划需求；
- Handlebars 支持注释，可以在大模型的输入输出中可以更好的提高可靠性和可读性；
- 几乎所有编码语言都有 Handlebars SDK，使得以后可以轻松地将这个规划器移植到 Semantic Kernel 的 Python、Java 甚至未来可能的 Js/Ts、PHP等语言的实现版本中，具有更好的通用性。
- 大语言模型很多都训练过 Handlebars 的相关代码，所以 Handlebars Planner 输出的结果具有较高的代码级可靠性，使得整体的效能更加优秀。

Handlebars Planner 输出结果示例

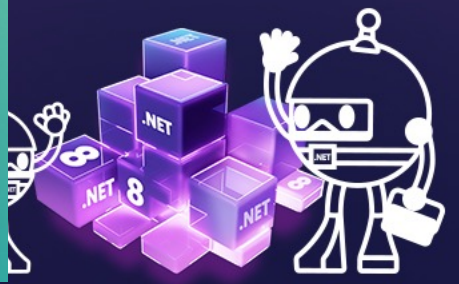
创建一本包含 3 章的书，内容涉及名为“The Thinking Caps”的俱乐部中的一群孩子

```
{{!-- 步骤 1: 初始化书籍标题和章节数目 --}}
{{set "bookTitle" "The Thinking Caps"}}
{{set "chapterCount" 3}}

{{!-- 步骤 2: 根据给出的章节数生成小说大纲 --}}
{{set "novelOutline" (WriterPlugin-NovelOutline input=(get "bookTitle") chapterCount=(get "chapterCount"))}}

{{!-- 步骤 3: 遍历章节并为每个章节生成内容 --}}
{{#each (range 1 (get "chapterCount"))}}
    {{set "chapterIndex" this}}
    {{set "chapterSynopsis" (MiscPlugin-ElementAtIndex input=(get "novelOutline") index=(get "chapterIndex"))}}
    {{set "previousChapterSynopsis" (MiscPlugin-ElementAtIndex input=(get "novelOutline") index=(subtract (get
    "chapterIndex") 1))}}

    {{!-- 步骤 4: 使用 WriterPlugin-NovelChapter helper 编写章节内容 --}}
    {{set "chapterContent" (WriterPlugin-NovelChapter input=(get "chapterSynopsis") theme=(get "bookTitle")
    previousChapter=(get "previousChapterSynopsis") chapterIndex=(get "chapterIndex"))}}

    {{!-- 步骤 5: 输出章节内容 --}}
    {{json (get "chapterContent")}}
{{/each}}
```



Handlebars Planner 的调用示例

```
using Microsoft.SemanticKernel.Planning.Handlebars;

// 创建内核 (kernel)
var builder = Kernel.CreateBuilder();
builder.Services.AddAzureOpenAIChatCompletion(/* 在此添加您的配置信息 */);
builder.Plugins.AddFromType<Plugins.Math>();

var kernel = builder.Build();
var planner = new HandlebarsPlanner(new HandlebarsPlannerConfig() { AllowLoops = true });
var plan = await planner.CreatePlanAsync(kernelWithMath, ask);

// 将计划打印到控制台
Console.WriteLine($"计划: {plan}");

// 执行计划
var result = plan.Invoke(kernelWithMath, []).Trim();

Console.WriteLine($"结果: {result}");
```



Handlebars Planner 的计划重用的方式

```
// 将计划对象序列化为字符串
var serializedPlan = plan.ToString();
HandlebarsPlan reloadedPlan = new HandlebarsPlan(serializedPlan, "");
// 执行计划
var result = plan.Invoke(kernelWithMath, []).Trim();

// 将结果打印到控制台
Console.WriteLine($"结果: {result}");
```

总结

- Handlebars 模板无论是在提示词模板场景还是 Planner 的场景下，都可以更加节省 token，并提高准确性，是一个**多快好省**的优秀模板选择。

欢迎关注 Semantic Kernel 中文技术社区公众号

Q&A

谢谢