

加餐 | GPT编程（下）：如何利用OpenAI的API解决语言处理类任务？

2023-05-15 郭屹 来自北京

《手把手带你写一个MiniSpring》



你好，我是郭屹。

上一节课我们一起学习了如何使用 ChatGPT 来辅助我们编程，通过问答它给出了我们想要的答案。这个过程实际上还是以手工的方式来对话，不过好在 OpenAI 还提供一些 API 供程序调用。现在微软的搜索引擎 Bing 和 Quora 网站推出的 Poe 都使用了它。所以今天我们一起看看这些 API。

API 入门

OpenAI 给开发者提供 API 来访问它的模型，帮助开发者解决语言处理类任务。

它提供多种基础功能，包括：

内容生成 Content generation

摘要 Summarization


分类及语义分析 Classification, categorization, and sentiment analysis

翻译 Translation

...

最主要的 API 访问入口是 [🔗 Completions](#)。开发者输入一些文本，API 会根据你在文本中的意图返回另一段文本。你可以以 HTTP Request 的方式与 API 进行交互，然后通过 API key 进行身份认证。

格式如下：

 复制代码

```
1 curl https://api.openai.com/v1/chat/completions \  
2   -H "Content-Type: application/json" \  
3   -H "Authorization: Bearer $OPENAI_API_KEY" \  
4   -d '{  
5     "model": "gpt-3.5-turbo",  
6     "messages": [{"role": "user", "content": "Say this is a test!"}],  
7     "temperature": 0.7  
8   }'
```

支持的模型有 7 个。

[🔗 GPT-4 Limited beta](#)：GPT-3.5 之上的改进，对自然语言的理解和处理更强。

[🔗 GPT-3.5](#)：在 GPT-3 基础上改进的，对自然语言的理解和处理更强。

[🔗 DALL·E Beta](#)：图像处理。

[🔗 Whisper Beta](#)：音频转文本。

[🔗 Embeddings](#)：将文本转成嵌入向量。

[🔗 Moderation](#)：检测敏感信息和非安全内容。

🔗 **GPT-3**: 自然语言理解和处理的模型。

有了这些 API，在我们的程序中就可以使用它们，也可以用这些 API 做出我们自己的插件来。

程序中调用 OpenAI 的 API

在 Spring 中用 HTTPClient 直接调用 OpenAI 的 API。我们从 /v1/chat/completions、/v1/completions、/v1/edits 与 /v1/images/generations 四个 API 入手，一步步进行封装。

先定义一个**基础请求实体类**，供 service 使用，参考如下：

📄 复制代码


```
1 package com.example.chatgptdemo.api.request;
2 import jakarta.validation.constraints.NotBlank;
3 import lombok.Getter;
4 import lombok.Setter;
5 @Getter
6 @Setter
7 public class BaseRequest {
8     @NotBlank(message = "openai_apiKey cannot be blank")
9     private String apiKey;
10 }
```

再按照我们示例的几种 API，定义多种请求实体类。


📄 复制代码

```
1 package com.example.chatgptdemo.api.request;
2 import lombok.Getter;
3 import lombok.Setter;
4 @Getter
5 @Setter
6 public class ChatCompletionRequest extends BaseRequest{
7     private String content;
8 }
```

```
1 package com.example.chatgptdemo.api.request;
2 import lombok.Getter;
3 import lombok.Setter;
4 @Getter
5 @Setter
6 public class CompletionRequest extends BaseRequest{
7     private String prompt;
8 }
```

 复制代码

```
1 package com.example.chatgptdemo.api.request;
2
3 import lombok.Getter;
4 import lombok.Setter;
5 @Getter
6 @Setter
7 public class EditRequest extends BaseRequest{
8     private String input;
9     private String instruction;
10 }
```

 复制代码

```
1 package com.example.chatgptdemo.api.request;
2
3 import lombok.Getter;
4 import lombok.Setter;
5 @Getter
6 @Setter
7 public class ImageGenerationRequest extends BaseRequest {
8     private String prompt;
9     private Integer n;
10 }
```

 复制代码

然后定义返回给调用方的 Response 类。

```
1 package com.example.chatgptdemo.api.response;
2 import lombok.Getter;
3 import lombok.Setter;
4 @Getter
```

 复制代码

```
5 @Setter
6 public class ChatCompletionResponse {
7     private String content;
8 }
```

 复制代码

```
1 package com.example.chatgptdemo.api.response;
2 import lombok.Getter;
3 import lombok.Setter;
4 @Getter
5 @Setter
6 public class CompletionResponse {
7     private String text;
8 }
9
```

 复制代码

```
1 package com.example.chatgptdemo.api.response;
2
3 import lombok.Getter;
4 import lombok.Setter;
5 @Getter
6 @Setter
7 public class EditResponse {
8     private String text;
9 }
```

 复制代码


```
1 package com.example.chatgptdemo.api.response;
2 import lombok.Getter;
3 import lombok.Setter;
4 import java.util.List;
5 @Getter
6 @Setter
7 public class ImageGenerationResponse {
8     private List<String> urls;
9 }
```

接下来则**定义 Service 层**，这也是我们进行核心业务定制的部分。定义 ChatgptDemoService 接口和 ChatgptDemoServiceImpl 实现类。

 复制代码

```
1 package com.example.chatgptdemo.service;
2
3 public interface ChatgptDemoService {
4     ChatCompletionResponse chatCompletions(ChatCompletionRequest request);
5     CompletionResponse completions(CompletionRequest request);
6     EditResponse edits(EditRequest request);
7     ImageGenerationResponse imageGenerations(ImageGenerationRequest request);
8 }
```

Service 的实现如下：

 复制代码

```
1 package com.example.chatgptdemo.service.impl;
2 @Service
3 public class ChatgptDemoServiceImpl implements ChatgptDemoService {
4     final private ChatgptNetworkService networkService;
5     @Autowired
6     public ChatgptDemoServiceImpl(ChatgptNetworkService networkService) {
7         this.networkService = networkService;
8     }
9     @Override
10    public ChatCompletionResponse chatCompletions(ChatCompletionRequest request)
11        MessageObject message = new MessageObject();
12        message.setContent(request.getContent());
13        message.setRole("user");
14        ChatCompletionModel model = new ChatCompletionModel();
15        model.setModel("gpt-3.5-turbo");
16        model.setMessages(Collections.singletonList(message));
17        model.setTemperature(0.7);
18        ChatCompletionResult result = networkService.ChatCompletions(model, request);
19        Optional<ChoiceResult> firstResultOptional = result.getChoices().stream()
20            .filter(choiceResult -> choiceResult.getMessage().getContent().length() > 0)
21            .findFirst();
22        ChatCompletionResponse response = new ChatCompletionResponse();
23        if (firstResultOptional.isEmpty()) {
24            return response;
25        }
26        ChoiceResult choiceResult = firstResultOptional.get();
27        response.setContent(choiceResult.getMessage().getContent());
28        return response;
29    }
```

```

28     @Override
29     public CompletionResponse completions(CompletionRequest request) {
30         CompletionModel model = new CompletionModel();
31         model.setModel("text-davinci-003");
32         model.setPrompt(request.getPrompt());
33         model.setMaxTokens(7);
34         model.setN(1);
35         model.setTopP(1);
36         model.setStream(false);
37         CompletionResult result = networkService.completions(model, request.getApiKey());
38         Optional<ChoiceResult> firstResultOptional = result.getChoices().stream()
39             .filter(choiceResult -> choiceResult.isTopChoice())
40             .findFirst();
41         CompletionResponse response = new CompletionResponse();
42         if (firstResultOptional.isEmpty()) {
43             return response;
44         }
45         ChoiceResult choiceResult = firstResultOptional.get();
46         response.setText(choiceResult.getText());
47         return response;
48     }
49     @Override
50     public EditResponse edits(EditRequest request) {
51         EditModel model = new EditModel();
52         model.setInput(request.getInput());
53         model.setInstruction(request.getInstruction());
54         EditResult result = networkService.edits(model, request.getApiKey());
55         Optional<ChoiceResult> firstResultOptional = result.getChoices().stream()
56             .filter(choiceResult -> choiceResult.isTopChoice())
57             .findFirst();
58         EditResponse response = new EditResponse();
59         if (firstResultOptional.isEmpty()) {
60             return response;
61         }
62         ChoiceResult choiceResult = firstResultOptional.get();
63         response.setText(choiceResult.getText());
64         return response;
65     }
66     @Override
67     public ImageGenerationResponse imageGenerations(ImageGenerationRequest request) {
68         ImageGenerationModel model = new ImageGenerationModel();
69         model.setPrompt(request.getPrompt());
70         model.setN(request.getN());
71         model.setSize("1024x1024");
72         ImageGenerationResult result = networkService.imageGenerations(model, request.getApiKey());
73         List<ImageGenerationResult.ImageGenerationData> resultList =
74             result.getData().stream().toList();
75         List<String> urls = new ArrayList<>(resultList.size());
76         resultList.forEach(r -> urls.add(r.getUrl()));
77         ImageGenerationResponse response = new ImageGenerationResponse();
78         response.setUrls(urls);
79         return response;
80     }

```

通过实现类的代码可以看出，基本是围绕构造 OpenAI 提供的 API 调用参数作文章，只不过为了验证，我们写死了很多参数，在实际实操过程中我们可以依据自身业务情况，进行动态调整。

我们使用了 ChatgptNetworkService 这样一个 Bean 作为通讯基础，而这个 Bean 的功能正是对 OpenAI 的 API 进行调用。我们直接使用 RestTemplate 访问的 API。

为了构造 API 调用，我们定义了 xxxModel 作为 API 调用的请求参数实体类，还定义了 xxxResult 作为 API 接口返回值。

Model 实体类：

[复制代码](#)


```
1 package com.example.chatgptdemo.network.model;
2 import com.fasterxml.jackson.annotation.JsonProperty;
3 import lombok.Getter;
4 import lombok.Setter;
5 @Getter
6 @Setter
7 public class CompletionModel {
8     private String model;
9     private String prompt;
10    @JsonProperty("max_tokens")
11    private Integer maxTokens;
12    private Integer n;
13    @JsonProperty("top_p")
14    private Integer topP;
15    private Boolean stream;
16 }
```

[复制代码](#)

```
1 package com.example.chatgptdemo.network.model;
2 import lombok.Getter;
3 import lombok.Setter;
4 @Setter
5 @Getter
```



```
6 public class MessageObject {
7     private String role;
8     private String content;
9 }
```

 复制代码

```
1 package com.example.chatgptdemo.network.model.chat;
2 import com.example.chatgptdemo.network.model.MessageObject;
3 import lombok.Getter;
4 import lombok.Setter;
5 import java.util.List;
6 @Getter
7 @Setter
8 public class ChatCompletionModel {
9     private String model;
10    private List<MessageObject> messages;
11    private Double temperature;
12 }
```

 复制代码

```
1 package com.example.chatgptdemo.network.model;
2 import lombok.Getter;
3 import lombok.Setter;
4 @Getter
5 @Setter
6 public class EditModel {
7     private String model;
8     private String input;
9     private String instruction;
10 }
```

[📄 复制代码](#)

```
1 package com.example.chatgptdemo.network.model;
2 import lombok.Getter;
3 import lombok.Setter;
4 @Getter
5 @Setter
6 public class ImageGenerationModel {
7     private String prompt;
8     private Integer n;
9     private String size;
10 }
```

Result 实体类:

[📄 复制代码](#)

```
1 package com.example.chatgptdemo.network.result;
2 import lombok.Getter;
3 import lombok.Setter;
4 @Getter
5 @Setter
6 public class CreatedResult {
7     private Long created;
8 }
```

[📄 复制代码](#)

```
1 package com.example.chatgptdemo.network.result;
2 import lombok.Getter;
3 import lombok.Setter;
4 @Getter
5 @Setter
6 public class ObjectResult extends CreatedResult{
7     private String id;
8     private String object;
9     private String model;
10 }
```

[📄 复制代码](#)

```
1 package com.example.chatgptdemo.network.result;
2 import com.example.chatgptdemo.network.model.MessageObject;
3 import com.fasterxml.jackson.annotation.JsonProperty;
```

```
4 import lombok.Getter;
5 import lombok.Setter;
6 @Getter
7 @Setter
8 public class ChoiceResult {
9     private MessageObject message;
10    private String text;
11    @JsonProperty("finish_reason")
12    private String finishReason;
13    private Integer index;
14 }
```

 复制代码

```
1 package com.example.chatgptdemo.network.result;
2 import com.fasterxml.jackson.annotation.JsonProperty;
3 import lombok.Getter;
4 import lombok.Setter;
5 @Getter
6 @Setter
7 public class UsageResult {
8     @JsonProperty("prompt_tokens")
9     private Integer promptTokens;
10    @JsonProperty("completion_tokens")
11    private Integer completionTokens;
12    @JsonProperty("total_tokens")
13    private Integer totalTokens;
14 }
```


 复制代码

```
1 package com.example.chatgptdemo.network.result;
2 @Getter
3 @Setter
4 public class CompletionResult {
5     private UsageResult usage;
6     private List<ChoiceResult> choices;
7 }
```


 复制代码

```
1 package com.example.chatgptdemo.network.result.chat;
2 @Getter
3 @Setter
```

```
4 public class ChatCompletionResult extends CompletionResult {
5 }
```

 复制代码

```
1 package com.example.chatgptdemo.network.result;
2 import lombok.Getter;
3 import lombok.Setter;
4 @Getter
5 @Setter
6 public class EditResult extends CompletionResult {
7 }
8
```

 复制代码

```
1 package com.example.chatgptdemo.network.result;
2 import com.example.chatgptdemo.api.response.ImageGenerationResponse;
3 import lombok.Getter;
4 import lombok.Setter;
5 import java.util.List;
6 @Getter
7 @Setter
8 public class ImageGenerationResult extends CreatedResult{
9     private List<ImageGenerationData> data;
10     @Getter
11     @Setter
12     public static class ImageGenerationData {
13         private String url;
14     }
15 }
```

在实体类定义完毕之后，接下来我们着手实现 ChatGptNetworkService，看看 Spring 中如何通过 RestTemplate 调用 OpenAI 的 API。

 复制代码

```
1 package com.example.chatgptdemo.network;
2 @Service
3 public class ChatgptNetworkService {
4     Gson gson = new Gson();
5     private static final Logger logger = LoggerFactory.getLogger(ChatgptNetworkSe
6     final private RestTemplate restTemplate;
7     @Autowired
```

```

8     public ChatgptNetworkService(RestTemplate restTemplate) {
9         this.restTemplate = restTemplate;
10    }
11    public ChatCompletionResult ChatCompletions(ChatCompletionModel model, String
12        HttpHeaders headers = new HttpHeaders();
13        headers.setContentType(MediaType.APPLICATION_JSON);
14        headers.setBearerAuth(apiKey);
15        HttpEntity<> entity = new HttpEntity<>(model, headers);
16        URI uri = UriComponentsBuilder.newInstance()
17            .scheme("https")
18            .host("api.openai.com")
19            .path("/v1/chat/completions")
20            .build()
21            .toUri();
22        logger.info("The parameters for request API {}: {}", uri, gson.toJson(ent
23        ResponseEntity<ChatCompletionResult> response = restTemplate.postForEntit
24        logger.info("The response result from API {}: {}", uri, gson.toJson(respo
25        if (response.getStatusCode().isError()) {
26            throw new RuntimeException(gson.toJson(response.getBody()));
27        }
28        return response.getBody();
29    }
30    public CompletionResult completions(CompletionModel model, String apiKey) {
31        HttpHeaders headers = new HttpHeaders();
32        headers.setContentType(MediaType.APPLICATION_JSON);
33        headers.setBearerAuth(apiKey);
34        HttpEntity<> entity = new HttpEntity<>(model, headers);
35        URI uri = UriComponentsBuilder.newInstance()
36            .scheme("https")
37            .host("api.openai.com")
38            .path("/v1/completions")
39            .build()
40            .toUri();
41        logger.info("The parameters for request API {}: {}", uri, gson.toJson(ent
42        ResponseEntity<CompletionResult> response = restTemplate.postForEntity(ur
43        logger.info("The response result from API {}: {}", uri, gson.toJson(respo
44        if (response.getStatusCode().isError()) {
45            throw new RuntimeException(gson.toJson(response.getBody()));
46        }
47        return response.getBody();
48    }
49    public EditResult edits(EditModel model, String apiKey) {
50        HttpHeaders headers = new HttpHeaders();
51        headers.setContentType(MediaType.APPLICATION_JSON);
52        headers.setBearerAuth(apiKey);
53        HttpEntity<> entity = new HttpEntity<>(model, headers);
54        URI uri = UriComponentsBuilder.newInstance()
55            .scheme("https")
56            .host("api.openai.com")

```

```

57         .path("/v1/edits")
58         .build()
59         .toUri();
60     logger.info("The parameters for request API {}: {}", uri, gson.toJson(ent
61     ResponseEntity<EditResult> response = restTemplate.postForEntity(uri, ent
62     logger.info("The response result from API {}: {}", uri, gson.toJson(respo
63     if (response.getStatusCode().isError()) {
64         throw new RuntimeException(gson.toJson(response.getBody()));
65     }
66     return response.getBody();
67 }
68 public ImageGenerationResult imageGenerations(ImageGenerationModel model, Str
69     HttpHeaders headers = new HttpHeaders();
70     headers.setContentType(MediaType.APPLICATION_JSON);
71     headers.setBearerAuth(apiKey);
72     HttpEntity<?> entity = new HttpEntity<>(model, headers);
73     URI uri = UriComponentsBuilder.newInstance()
74         .scheme("https")
75         .host("api.openai.com")
76         .path("/v1/images/generations")
77         .build()
78         .toUri();
79     logger.info("The parameters for request API {}: {}", uri, gson.toJson(ent
80     ResponseEntity<ImageGenerationResult> response = restTemplate.postForEnti
81     logger.info("The response result from API {}: {}", uri, gson.toJson(respo
82     if (response.getStatusCode().isError()) {
83         throw new RuntimeException(gson.toJson(response.getBody()));
84     }
85     return response.getBody();
86 }
87 }

```

代码一点都不复杂，就是按照 API 的规定进行常规的 HTTP REST 请求。

有了这个 service，再定义 Controller 层。这层是整个系统与外部调用 API 的门户。我们先定义 ChatgptDemoController 类。

 复制代码

```

1 package com.example.chatgptdemo.controller;
2
3 @RestController
4 public class ChatgptDemoController {
5     final private ChatgptDemoService chatgptDemoService;
6     @Autowired

```

```

7     public ChatgptDemoController(ChatgptDemoService chatgptDemoService) {
8         this.chatgptDemoService = chatgptDemoService;
9     }
10    @PostMapping("/v1/chat/completions")
11    public ChatCompletionResponse chatCompletions(@Validated @RequestBody ChatCor
12        return chatgptDemoService.chatCompletions(request);
13    }
14    @PostMapping("/v1/completions")
15    public CompletionResponse completions(@Validated @RequestBody CompletionReque
16        return chatgptDemoService.completions(request);
17    }
18    @PostMapping("/v1/edits")
19    public EditResponse edits(@Validated @RequestBody EditRequest request) {
20        return chatgptDemoService.edits(request);
21    }
22    @PostMapping("/v1/images/generations")
23    public ImageGenerationResponse imageGenerations(@Validated @RequestBody Image
24        return chatgptDemoService.imageGenerations(request);
25    }
26 }
27

```

在这里，我们用的接口路径，依然和 OpenAI 的 API 保持一致，便于我们的理解。

接下来，基于封装好的框架，我们用几个常见的例子，来实际调用一下。

1. 把自然语言转换成程序代码：我们利用 API 来编写 “hello world” 程序。

请求参数如下：

 复制代码


```

1 POST http://localhost:8080/v1/chat/completions
2 Content-Type: application/json
3
4 {
5     "apiKey": "sk-xxxxxx",
6     "content": "Please output helloworld in Java"
7 }

```

在这里我们本地启动 Springboot 项目，传入 apiKey（填入在 OpenAI 官网上申请的 Open_API_KEY，均为 sk 前缀开头）。content 字段则敲入自然语言表达我们的意图。

返回结果如下：

 复制代码


```
1 HTTP/1.1 200
2 Content-Type: application/json
3 Transfer-Encoding: chunked
4 Date: Tue, 02 May 2023 00:42:59 GMT
5 Connection: close
6 {
7   "content": "Here's the code to output \"Hello World\" in Java:\n\n```\npublic c
8 }
```

通过返回结果可以看出，返回值在一个字段中以字符串形式完整返回，代码是可以正常运行的。

2. 生成 SQL 语句


SQL 也是开发过程中不可绕过的一个话题，同样地，我们可以借助 API 帮我们编写 SQL 语句。在这里我们测试一个简单的场景：在学生表中查询所有学生的姓名。

请求参数如下：

 复制代码

```
1 POST http://localhost:8080/v1/chat/completions
2 Content-Type: application/json
3
4 {
5   "apiKey": "sk-xxxxxxx",
6   "content": "Please select all students' name from Student table in SQL"
7 }
```

返回结果如下：


 复制代码

```
1 HTTP/1.1 200
2 Content-Type: application/json
3 Transfer-Encoding: chunked
4 Date: Tue, 02 May 2023 00:50:11 GMT
5 Connection: close
6 {
7   "content": "SELECT name FROM Student;"
8 }
```

3. 查找图片

我们还可以通过描述查询对应的图片，一般描述越精确，图片准确度越高。你可以看一下这个示例。

请求参数：

 复制代码

```
1 POST http://localhost:8080/v1/images/generations
2 Content-Type: application/json
3
4 {
5   "apiKey": "sk-xxxxxxx",
6   "prompt": "find a tiger image",
7   "n": 1
8 }
```

返回参数如下：

 复制代码

```
1 HTTP/1.1 200
2 Content-Type: application/json
3 Transfer-Encoding: chunked
4 Date: Tue, 02 May 2023 04:12:48 GMT
5 Connection: close
6 {
7   "urls": [
8     "https://oaidalleapiprodscus.blob.core.windows.net/private/org-mnyBsdtqxGUtwx
9   ]
```


```
10 }
```

可以看到，返回值中返回了 URL 的数组，这个 URL 对应的图片与请求参数中的语义尽可能保持一致，而返回的 URL 个数则取决于请求参数中 `n` 的值。

4. 拼写检查


拼写错误也是我们经常会遇到的场景，而且总是难以发现。正好有个接口可以帮我们校验拼写错误。

请求参数如下：

 复制代码

```
1 POST http://8.218.233.184:8080/v1/edits
2 Content-Type: application/json
3
4 {
5     "apiKey": "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",
6     "input": "It is an rainn day",
7     "instruction": "Fix the spell mistakes"
8 }
```

返回参数如下：

 复制代码


```
1 HTTP/1.1 200
2 Content-Type: application/json
3 Transfer-Encoding: chunked
4 Date: Tue, 02 May 2023 05:25:38 GMT
5 Connection: close
6 {
7     "text": "It is a rainy day\n"
8 }
```

可以看到，这个接口纠正了我们两处拼写错误。

5. 为代码添加注释


我们写的代码，一样可以被分析，下面是一个简单的冒泡排序算法的实现。不过为了解析方便，我们用了许多换行符来把多行代码压缩到一行了。

请求参数：

 复制代码

```
1 POST http://localhost:8080/v1/edits
2 Content-Type: application/json
3
4 {
5     "apiKey": "sk-xxxxxx",
6     "input": "public static void sort(int[] arrays) { \nif (arrays.length == 0) {
7     "instruction": "Explain this Java code"
8 }
```

返回参数如下：

 复制代码

```
1 HTTP/1.1 200
2 Content-Type: application/json
3 Transfer-Encoding: chunked
4 Date: Tue, 02 May 2023 23:45:37 GMT
5 Connection: close
6 {
7     "text": "/* Java\nI discovered this code from a tutorial from edx from Havard
8 }
9
```

可以看到返回内容非常多，不仅给出了测试用例，还详细解释了每一行的功能。给人的感觉就是，它真的读懂了我们的代码。

小结

好了，这就是这节课的主要内容，最后我们来总结一下。

这节课我们了解了 OpenAI 的 API 的基础能力，包括内容生成、摘要、分类及语义分析、翻译等等。我也带你一步步实现了在程序中调用 OpenAI 的 API，还用它完成了很多任务，比如写代码、生成 SQL 语句、查找图片、检查拼写等等。利用好这些能力能够帮助我们程序员省去很多繁琐的工作，所以我们要更加积极地去拥抱这个变化。

我们可以将 GPT 作为我们编程的助手来提高工作效率，如编写标准程序单元、测试用例、生成注释说明，之后还可以进一步利用它来构建低代码平台和工具集，基于它训练特定业务领域的大模型。我们一起参与这个过程，会让新时代来得更早、更快。

最后我想用软件大师 Alan Kay 的话来结束这节课的内容，“The best way to predict the future is to invent it.”。

如果你觉得这节课的内容对你有帮助的话，可以分享给你的朋友，同时也欢迎你把你的感悟和思考分享到评论区和我讨论。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言 (1)



peter

2023-05-18 来自北京

感谢老师的加餐！

Q1:spring我知道用于网站后端开发，除此之外，是否还有其他用途？

Q2：能否加两餐讲一下spring中的设计模式？

Q3：能否加两餐讲几个spring的典型面试题？

作者回复：不光是网站后端，几乎所有后端都是以Spring为基础的。

你说的话模式和面试什么的，好几个人提到过，这个需要跟极客商量。这是很有用处的，不过跟这门课本身关系又不太大，况且极客平台上，已经有专门讲面试和设计模式的课程。



