第 1 章

1 拥抱大语言模型

本章主要介绍大语言模型的定义、发展和广泛的应用场景，通过介绍主流的大语言模型，了解大语言模型的现状和优势，最后介绍LangChain和大语言模型的关系。

1.1 大语言模型简介

AI时代席卷而来，几乎每个人的工作和生活都因此发生了改变。试想一下，突然间你拥有一个超级智能的助手，它能读懂所有主流书籍、文章和开源代码库，并像人类一样高效地进行交流。显而易见，这个助手正是我们今天的主角——大语言模型（Large Language Model，LLM）。

它就像一个超级大脑，拥有数十亿甚至万亿个“参数”，能够学习语言的各种模式和规则。它能够完成很多令人惊叹的任务：

- 像领域专家一样：从理工科到人文领域，它能化身专家，引经据典，提供专业咨询和决策。
- 像百变写手一样：通过简单沟通，快速确定主题和内容，创作出各种爆款文章和段子，助力自媒体品牌的打造。
- 像专属老师一样：任何问题都难不倒它，24小时不间断地提供服务，加快你的学习和工作进度。
- 像结对编程的工程师一样：分析你的代码和实现，提供细致入微的专业指导和协作。成为强大的编程副驾驶员，为你的开发之旅保驾护航。

当然，大语言模型也有一些不足之处，比如：

- 可能存在偏差：由于训练数据的限制，AI可能在某些问题上存在偏向性，无法保证主

观上的中立性。

- 容易出现幻觉：大模型可能因接收到虚假或错误的信息，导致生成的文本与现实不符，即所谓的大模型“幻觉”现象。

总之，大语言模型是一项非常强大的计算机技术，具有较高的训练成本和开发成本。它已经开始改变我们的世界，全世界各大顶尖科技公司正不断完善和开发大语言模型，未来既属于我们，也属于大语言模型。

1.1.1　大语言模型的定义

虽然大语言模型目前还没有统一的定义，但笔者较为认可AWS的定义：“大语言模型是基于大量数据进行预训练的超大型机器学习的深度训练模型。通常由上亿个参数的神经网络组成，参数通过自监督学习或半监督学习进行训练，能够理解人类的自然语言。”

它是一种通用型的模型，并非为解决某个特定领域问题而设计，而是通过海量文本数据进行预训练，能够解决各种类型的任务。

武侠世界中的“百晓生”精通各种武器和招式，而大语言模型也可视为一个通过大量文本（如书籍、代码、论文等）训练出来的“超级大脑”。

目前，主流的大语言模型都选择Transformer模型，OpenAI公司和Google公司都基于此模型开发了它们各自的大语言模型。Transformer模型的架构图如图1-1所示。该架构通过输入模块将数据传递至多个神经网络进行处理，整个过程非常复杂，后续章节将重点讲解相关内容。

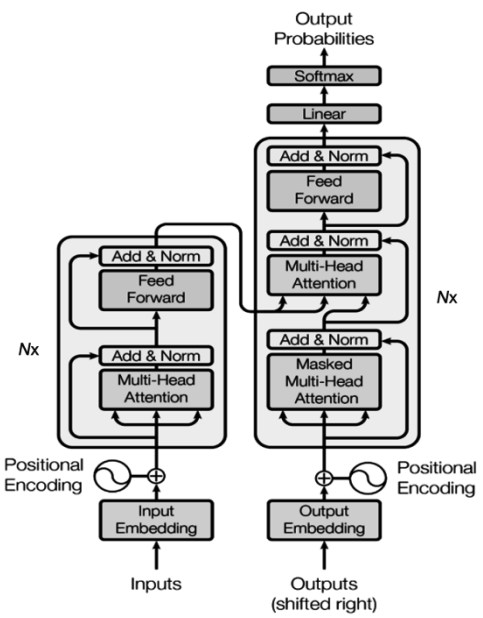

图 1-1　Transformer 模型架构图

训练大语言模型的过程包含以下几个步骤：

步骤 01 数据收集（Data Collecting）：收集多样化的文本数据集。

步骤 02 预处理（Preprocessing）：对收集的文本数据进行清理和标准化处理。

步骤 03 分词（Tokenization）：将预处理后的文本分割成更小单元，即 token（标记，也被称为词元）。

步骤 04 预训练（Pre-training）：基于数据开始训练模型。

步骤 05 微调（Fine-tuning）：通过调整模型微调，持续优化模型性能。

步骤 06 评估（Evaluation）：评估模型的效果和准确度。

步骤 07 部署（Deployment）：将训练好的模型部署到实际系统中使用。

1.1.2　大语言模型的发展和应用场景

大语言模型的发展与应用场景在最近三年呈现出爆发式增长，推动了自然语言处理（NLP）领域的创新与应用。从最早期的无代表性模型到如今基于深度学习的Transformer架构，大语言模型经历了数十年的发展历程，其应用场景也变得极为广泛。

1. 波澜壮阔的发展历程

大语言模型的发展历程如下：

（1）无代表模型（1970年左右）：基于手写的规则，处理少量数据。

（2）自然语言统计模型（1970—2000年）：通过统计方法预测后续词语出现的概率，如N-gram模型等。推理结果受数据集影响很大，因此预测偏差较大。

（3）神经网络模型（2000年—至今）：在2017年之前是小模型，2017年后开始使用大量数据进行模型训练。随着深度学习技术的飞速发展，尤其是Transformer架构的提出，大语言模型迎来了爆发式增长。其中，以OpenAI公司开发的GPT系列模型为代表，包括GPT、GPT-2、GPT-3、GPT-4等，目前最新的模型是GPT-4o，它可以无缝处理文本、图像和音频，成为“王炸级”的模型。

Google和Meta（前身为Facebook）作为全球科技巨头，在大语言模型领域也作出了重要贡献。它们分别推出了自己的大语言模型，Google的代表作是BERT（Bidirectional Encoder Representations from Transformers），而Meta推出了M2M-100（Many-to-Many Multilingual Model），两者在各自领域都取得了显著成就。

特别是Meta公司近年来开源了大语言模型Llama，使得本地计算机运行大模型成为可能。

目前，Llama 3是最新版本，它的上下文长度已经达到8k。历代Llama的上下文长度对比图如图1-2所示。

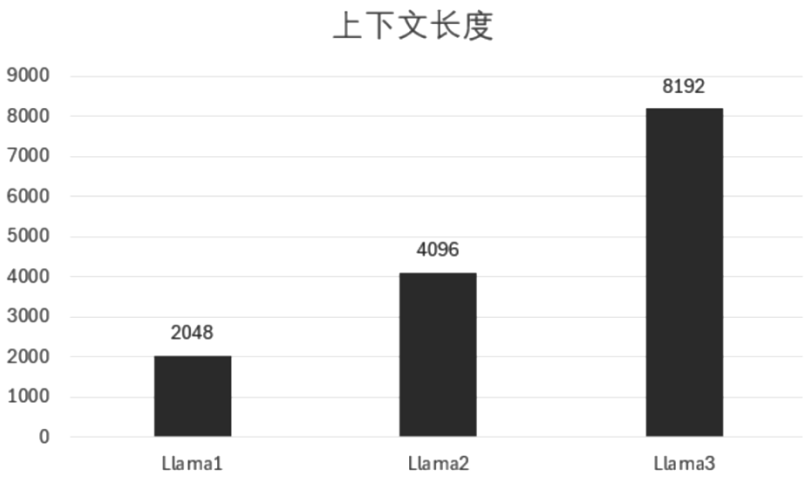

图 1-2　不同版本的 Llama 上下文长度对比图

随着AI技术的进步，大语言模型变得越来越强大。最初，大语言模型很小，只能处理简单任务，参数数量只有几百万个，训练数据仅有几吉字节（GB）。因此，它们在理解和生成自然语言方面表现一般。

如今，大语言模型的规模已经大幅度扩展，参数数量可达数十亿甚至上百亿级别，训练数据也高达数十太字节（TB）。这使得它们能够快速地识别出更复杂的语言结构和语义信息，从而生成更高质量的文本输出。

具体来说，大语言模型在以下几个方面有了显著改进：

- 更好地理解自然语言的真正含义，厘清词语之间的关系，全面理解自然语言的含义。
- 生成更高质量的文本，生成效果接近人类水平。
- 完成更复杂的任务。包括多语言翻译、百科类问答、各类写作以及科学实验知识讲解等任务。

2. 应用场景

大语言模型的应用场景非常广泛，包括但不限于：

- 文本创作：大语言模型可以生成极具价值的文本，如人文小说、诗歌、文章概括等。它可以模拟不同的语言风格，用于自动生成命题文章、创意文案、带有关键字的指定风格的歌曲歌词等。
- 文本分析和理解：处理海量文字，提取有用信息，如概括文章大意和关键知识点。对科学论文进行分析整理，帮助人们快速理解论文内容和重点。
- 智能对话：聊天机器人利用大语言模型，可以更智能地与用户进行交流，解答用户疑

问，满足多样化需求。它既可用于个人领域的专家服务，也可应用于商务客服等场景。

- 多语言处理与跨语言应用：不仅能够顺畅地处理英语，还能处理其他语言，如德语、法语以及其他小语种，实现多语言文本的快速生成、翻译和理解。
- 编程辅助：大语言模型可用于生成代码段、编写完整功能的程序，辅助开发者进行各类软件和硬件开发。它还能分析用户提供的代码段，查找编程漏洞，优化现有代码，提高程序性能，甚至辅助生成单元测试代码，提高编程效率和代码质量。
- 医疗和科学研究辅助：越来越多的传统医疗公司转型为AI公司，利用真实临床数据训练大模型，促进药物研发、临床应用、药物制造和供应等方面的工作。
- 教育赋能：大语言模型可以应用于教育领域，帮助学生学习各种学科知识并拓展知识面。它就像一个24小时在线的贴身老师，让教育永远“在线”。
- 娱乐产业：大语言模型能够生成流行音乐、创意绘画、跌宕起伏的电影剧本等，为创作者和艺术家提供无限的灵感和无微不至的创作支持。它还可以用于游戏行业的原画设计、数值设计等，提高娱乐产业的生产效率。

综上所述，大语言模型的发展和应用为人工智能技术在自然语言处理领域的应用带来了新的突破和可能性，成为人工智能发展的一块基石，也是最近几年最大的技术和创业风口。每天都有无数AI应用在各大应用商城上架，基于大语言模型的开发正如火如荼地展开。

1.2　主流的大语言模型

如今，大语言模型呈现百家争鸣之势，国外一些大公司的通用大模型已经改变了人们生活的方方面面。国内的科研院校和头部科技公司也积极投入中文大模型的研发，甚至华为还推出了政务大模型和城市大模型，服务政务并助力智慧城市建设。本节将分别介绍国内外各大主流大语言模型，以帮助读者了解大模型的现状和应用场景。

1.2.1　OpenAI 的大语言模型

OpenAI是一家位于美国旧金山的人工智能研究公司。该公司于2015年年底由埃隆·马斯克（Elon Musk）、山姆·奥尔特曼（Sam Altman）、格雷格·布罗克曼（Greg Brockman）等人创立。最初，OpenAI是一家非营利性公司，旨在开发通用人工智能（AGI）以造福全人类。微软在2019年7月对OpenAI进行了首次投资，金额达10亿美元，并达成了多年合作伙伴关系，这为OpenAI从纯研究型人工智能向商业化转型提供了重要支持。

随后，微软于2021年追加投资10亿美元，如今，微软的Copilot已基于最新的GPT-4模型，生成式AI与PC（个人电脑）已完全融合。

OpenAI开发了多种大语言模型，其中最负盛名的就是GPT系列。GPT系列大语言模型采

用了无监督学习方法，并基于Transformer架构进行开发。

GPT系列语言模型的发展历史如图1-3所示。

图 1-3　GPT 发展历史

- GPT-1：2018年首次发布，包含12层解码器，约1.15亿个参数，能够生成类似人类语言的文本，初见其锋芒。
- GPT-2：2019年发布，拥有15亿个参数，相比GPT-1增长了10倍。解码器增至48层，能生成更自然、连续的文本。该模型已开源，全世界的开发者可以进行学习研究。
- GPT-3：2020年发布，参数增加到1 750亿个，较以前的任何非稀疏语言模型多出10倍。GPT-3在多个NLP数据集上表现出强大的能力，能够完成不同风格文本的创作、问答、多语言翻译及动态推理等任务。
- GPT-4：目前最新的GPT系列模型，拥有10 000亿个参数，是GPT-3的5倍多。GPT-4拥有更强的学习能力，能够快速理解和学习新知识，适应性更强。GPT-4能理解更长的上下文，在对话过程中能够动态学习和推理。它的逻辑推理能力也大幅增强，能够完成更复杂的推理分析。

下面整理了各版本GPT的相关指标，如表1-1所示。

表 1-1　GPT 各版本对比

模型名称	GPT-1	GPT-2	GPT-3	GPT-4
版本	GPT-1	GPT-2	GPT-3	GPT-4
发布时间	2018	2019	2021	2024
参数数量	1.17 亿	1.5 亿	1 750 亿	17 600 亿
训练数据量	7000 本未发布图书	40GB Reddit 文章	6800 亿标记，来自多个来源，如 Wikipedia（维基百科）等	未公开，但远大于 GPT-3
性能		更强的泛化能力	更广泛的理解	多模态能力更强
语言理解能力	有限	更好	更复杂	等同于大学生水平
生成文本质量	有限	更自然	更精细	非常高

总的来说，GPT-4的应用场景广泛：

（1）图片生成和处理：能够高效生成和解析图片，成为设计师的重要助手。

（2）编程任务：根据用户需求编写代码，轻松完成编程任务。

（3）小说创作：根据给定的主题和关键字，自动编写高质量小说，模拟文字风格和人类的情绪表达。

（4）客服服务：在垂直领域为用户提供产品演示和咨询服务，及时响应用户需求。

（5）语音交互：可以处理语音、图片和文本，并以音频或文字方式进行回复，提供更佳的用户体验。

（6）个人助理：通过模型微调或其他AI技术对业务系统进行整合，学习业务知识。GPT-4能更好地理解自然语言，做好个人助理的工作，完成个性化任务。

目前最新的模型GPT-4o更是大放异彩，能够实时处理并推理音频、视觉和文本。人机交互体验也上了一个台阶，在非英文语言文本的处理上效率惊人，可以出色地完成更复杂的多模态任务。

1.2.2　Meta 的 Llama 模型

Meta公司拥有全世界最大的真实用户数据，并在2023年2月发布了第一代大语言模型Llama（又称为Llama 1），并将其开源。Llama迅速成为当时最优秀的大语言模型之一，提供了4种不同参数版本：7B、13B、30B、65B，其中B代表Billion（十亿），7B表示拥有70亿个参数。

Llama也是基于Transformer架构开发的，但进行了如下改进：

（1）优化训练过程：使用优化的训练技术，使其能够高效地从海量文本数据中学习。例

如，它采用了梯度检查点、混合精度训练等技术来降低训练成本和内存消耗。此外，Llama使用了因果多头注意力（Causal Multi-head Attention）机制，以减少内存使用和运行时间，显著提升了训练速度。在大规模预训练下，Llama能够学习并掌握广泛的语言模式、实现和推理能力。

（2）更先进的Transformer变体：Llama对Transformer层进行了改进，优化了注意力机制和前馈网络（Feedforward Networks），并通过归一化技术对输入进行处理，从而提高了训练效率和输出的准确性。特别重要的是，Llama使用了改进的注意力机制，如稀疏注意力（Sparse Attention）和长距离注意力（Long-range Attention），这些机制能够更好地处理具有长依赖关系的复杂文本。

由于Meta公司将Llama大模型完全开源且性能强大，Llama已经成为开源社区中最受欢迎的大模型之一。基于Llama的模型生态快速发展，衍生出了许多变体模型，其强大的变体模型生态如图1-4所示。

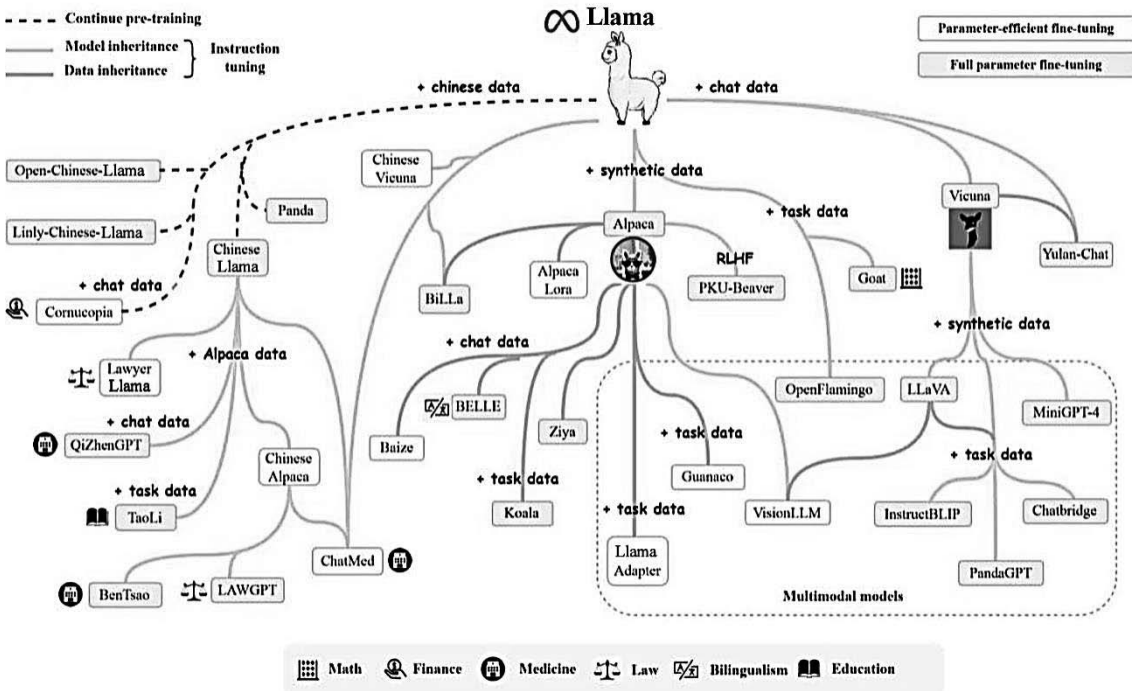

图 1-4　Llama 模型变体生态

从图1-4中可以看出，Llama也有针对中文的训练变体，例如专门的中文大模型Chinese-Llama。在此基础上，深圳大学的大数据系统计算机国家工程实验室发布了Linly中文大模型，将Llama的强大语言能力迁移到中文任务中。

很多中文大模型都基于Llama大模型进行开发，例如Chinese-Llama-Alpaca，目前已经基于Llama 2完成了中文大模型的训练和开发，并正在基于Llama 3进行升级。

实际上，Llama经历了多个版本的迭代，它的发展历史如图1-5所示。

图 1-5　Llama 的发展历史

目前，最新的Llama 3一经发布，便引起了业界和开发者的关注。Llama 3提供了8B和70B两个参数版本，同时提供了预训练基础版和指令调优版本。与Llama 2相比，Llama 3使用了更大的数据集进行训练，在推理、数学问题、代码生成和文本自由创作方面有显著提升。

Llama 3可以在所有主流云平台上部署，并且提供了模型API。据Meta官方宣布，不久后将可以在AWS、Databricks、谷歌云、Hugging Face、Kaggle、IBM WatsonX、微软Azure、NVIDIA NIM和Snowflake等平台上使用，并得到AMD、AWS、戴尔、英特尔、NVIDIA等公司提供的硬件平台的支持。

Meta公司继续引领大模型开源浪潮，Llama 3模型的应用将遍布全球。

Llama 3在标准基准测试中，比同级别的GPT大模型和Google Gemini大模型表现更为出色。根据官方公布的数据，Llama 3 与 Claude、GPT-3.5 的对比结果如图 1-6 所示。

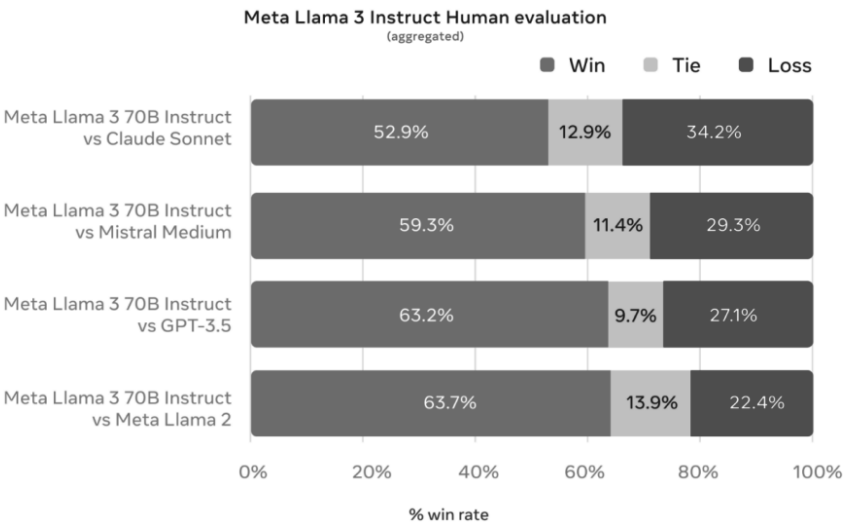

图 1-6　不同模型的人类评估结果

除了性能提升外，Llama还非常注重安全性。许多非开源大模型的内部实现细节无法被用户了解，可能存在用户数据泄露的风险。相比之下，Llama可以直接部署到本地，从而确保用户业务数据的绝对安全性和高度私密性，特别适合作为企业内部人工智能助手（AI Assistant）的核心模型。

在后续章节中，我们将使用LangChain框架来调用Llama模型，完成大语言模型应用的开发工作。

1.2.3　Claude 大语言模型

Claude是由美国初创Anthropic人工智能科技公司推出的大语言模型，该公司的创始团队来自OpenAI的早期研究员。该公司致力于研发通用型大语言模型，目前最具知名的产品是Claude系列。

Anthropic的核心理念是开发安全、无害的人工智能系统，强调生成内容的准确性和安全性，这与其他大模型相对宽容人工智能幻觉的态度形成了鲜明对比。

Claude可协助用户完成各种任务：

- 内容生成：能够生成文案、小说、剧本等多类型文本内容。
- 图像解释：最多可一次性上传5张图片，由AI分析和解释图片内容。
- 摘要：根据提供的文本或上传的文档进行分析，自动生成摘要。
- 分类：对文本、图像完成分类任务。
- 翻译：支持包括西班牙语、日语等在内的多种语言翻译，表达准确、流畅。
- 情感分析：能够理解人类的情感和感受，给出更富有建设性的建议和理性分析。

- 代码解释和生成：能够精确分析用户提供的代码，给出易于理解的解读，并根据需求生成高质量的代码。
- 问答：经过海量数据集的训练，Claude能够解答不同领域的问题，并提供实用的解决方案。
- 创意写作：可根据用户需求生成优美流畅，创意十足的文本，甚至完全避免典型的“AI风格”。

由于Claude的开发团队来自OpenAI的早期研究人员，其底层算法与GPT系列存在一定的相似性。在谷歌与亚马逊等科技巨头的支持下，Claude实现了快速的迭代发展。截至2024年4月，Claude已迭代至Claude 3版本，如图1-7所示。截至2025年5月22日，最新版本则已更新至Claude 4。

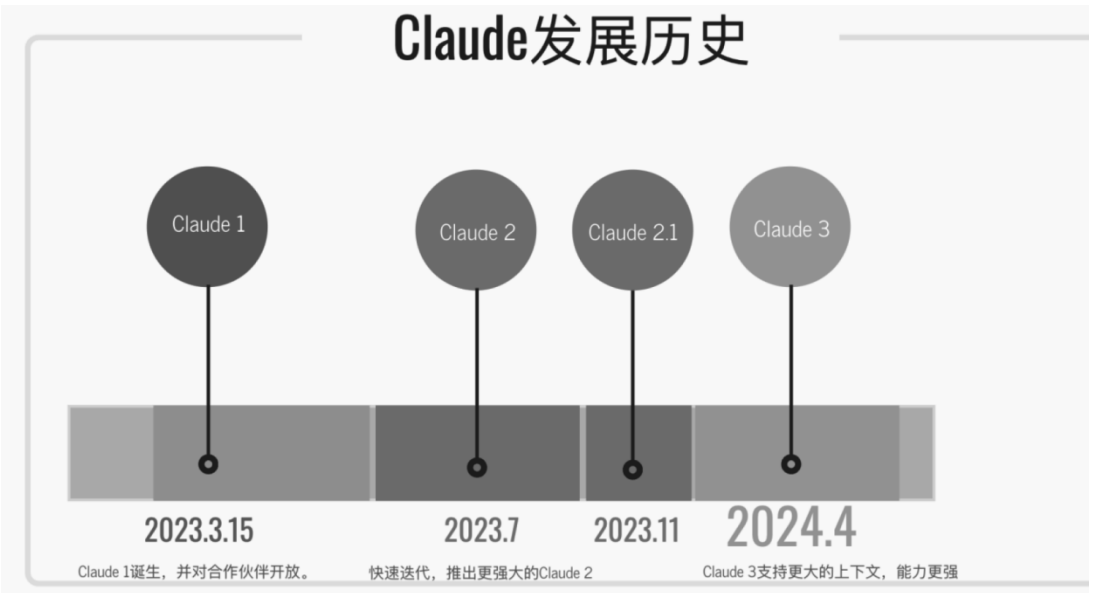

图 1-7　Claude 的发展历史

Claude 3实际上提供了3种不同定位的模型：

- Claude 3 Opus：这是Claude系列中智能水平最高的大模型，在常见评估基准上优于同行，拥有相当于本科水平的专业知识（MMLU）和研究生水平的专家推理能力（GPQA），适用于科研、金融分析等复杂场景的任务。
- Claude 3 Sonnet：该模型在智能水平和响应速度之间取得了理想的平衡，成本较低且性能强大，非常适合企业级应用开发。
- Claude 3 Haiku：这是系列中响应速度最快、最紧凑的模型，可实现即时交互，适合构建无缝互动的AI应用。

Claude 3的出现给OpenAI的GPT带来了很大的竞争压力，也进一步加速了大模型的迭代与发展。

1.2.4　国内自研大语言模型：ChatGLM、MOSS 和文心一言

尽管国外的大模型占据了主流位置，但国内也涌现了一批厚积薄发的优秀自研大模型，值得开发者学习和使用。其中，ChatGLM、MOSS和文心一言是国内自研型大模型中的佼佼者。接下来，我们首先介绍ChatGLM大模型。

1. ChatGLM大模型

ChatGLM系列是由智谱AI和清华大学联合开发的开源、支持中英双语的对话语言模型。该系列的首个版本是ChatGLM-6B，拥有6亿个参数，采用了模型量化技术，可以在普通消费级显卡上实现本地部署，最低显存需求仅为6GB。ChatGLM采用了与ChatGPT类似的技术架构，并特别针对中文问答与对话场景进行了优化。因此，如果是针对中文的对应应用场景，ChatGLM模型是一个理想的选择。

ChatGLM可以轻松完成各种任务，例如自我认知、提纲撰写、文案写作、邮件辅助写作、信息抽取、角色扮演、评论比较、旅游向导等。

随后，清华大学推出了ChatGLM2-6B。在保持初代模型对话流畅和部署门槛较低等优点的同时，引入了GLM的混合目标函数，使其在MMLU、CEval、GSM8K、BBH等数据集上的表现大幅提升，在同规模的开源模型中具有较强竞争力。

在代码生成方面，研发团队还发布了基于ChatGLM2-6B的CodeGeeX2代码生成器模型。该模型的编程能力全面提升，支持超过100种编程语言，并支持中英文对话，可用于代码解释、代码翻译、代码纠错、文档生成等任务，帮助程序员提高开发效率。

目前，ChatGLM最新的开源版本是ChatGLM3-6B。该版本采用了更多的训练数据，在语义理解、数学、推理、代码生成、知识问答等不同维度的数据集上均表现出色。ChatGLM3-6B不仅支持多轮对话，还支持工具调用（Function Call）、代码执行（Code Interpreter）和Agent任务等更复杂的应用场景。此外，ChatGLM的基础模型ChatGLM3-6B-Base、长文本对话模型ChatGLM3-6B-32K以及ChatGLM3-68-128K均已完全开源。在登记之后，这些模型可免费商用，因此ChatGLM3-6B也成为很多中文大模型创业者的首选模型之一。

ChatGLM官方还提供了在线版的GLM-4和GLM-3，具有联网功能，可以给出实时答复。除此之外，对于编程问题它也十分在行。例如，当被问及如何用Python脚本获取李白的一首诗时，ChatGLM给出的回复如图1-8所示。

图 1-8　ChatGLM 的回复

简单来说，ChatGLM是一种基于Transformer架构的人工智能模型，利用自注意力机制（Self-Attention）和位置编码（Positional Encoding）技术来理解和处理输入的对话内容。在生成对话回复时，ChatGLM会分析对话历史，然后生成既符合语境又贴切的回答。该模型经过了大量的对话数据训练，能够很好地把握人类语言的复杂性和丰富性，从而提供流畅自然的对话体验。

ChatGLM的工作机制主要包括编码器和解码器两个部分。编码器负责读取输入的对话内容，将其转换成一系列功能捕捉语义信息的固定长度向量。解码器根据这些向量以及自身的当前状态逐步生成回复序列。在生成回应时，ChatGLM采用贪婪搜索策略：不断地预测下一个最可能的单词，并将这个单词添加到回应序列中，直到达到预定的长度或碰到生成终止标记。

为了更好地对比ChatGLM和GPT之间的差异，笔者整理了表1-2，从架构、训练目标、应用领域等方面进行对比。

表 1-2　ChatGLM 和 GPT 的对比

模　　型	ChatGLM	GPT
架　　构	Transformer，包含编码器和解码器	Transformer，主要使用解码器，自回归语言模型
训练目标	根据上下文生成符合条件的内容	预测文本序列中的下一个词语
应用领域	更适用于聊天机器人、客服服务等场景	文本生成能力强，适用于更广泛的领域
语言理解	更专注于特定上下文的对话理解	长期会话中上下文能力有所不足

2. MOSS大模型

MOSS大模型是国内首个开放测试的类ChatGPT大语言模型，由复旦大学计算机科学技术学院团队研发并开源。

MOSS拥有160亿（16B）个参数，支持中英文双语，具有多轮对话能力，并提供多种插件。通过插件，MOSS可以实现网络查询、计算等任务。此外，该模型能以INT4/8精度在单张RTX 3090显卡上运行，具备较低的本地部署成本。然而，由于它的训练数据集只包含约7000亿个中英文单词，并采用自回归生成范式，因此在部分情况下可能出现误导性回复或不当内容。

目前，MOSS的最新版本是MOSS-003，增强了中文对话能力和逻辑推理能力，并提升了生成内容的安全性和可靠性。MOSS团队正在开发其多模态能力，逐步将语音、图像等模态融入MOSS大模型中，值得广大开发者持续关注。

3. 文心一言大模型

最后要介绍的是百度AI团队推出的大语言模型——文心一言。该大模型是国内知名的大模型之一，通过百度平台的大力推广而广为人知。

在ChatGPT掀起AI热潮后，百度研发团队快速响应挑战，推出了专注于中文对话的文心一言大模型，在内测数月之后，于2023年8月31日正式向全球用户开放使用。

文心一言和OpenAI一样提供了API调用大模型，LangChain也可以通过文心一言的Python版本的SDK来接入文心一言API。

1.3　大语言模型的开发工具 LangChain

不同的大语言模型各自所长，但普遍存在对话轮次和上下文长度的限制。为有效应对这些限制，开发者通常使用词嵌入（Word Embedding）和向量数据库（Vector Database）等技术，来优化和扩展大语言模型的能力。在此背景下，大语言模型应用框架的重要性显得尤为突出。

LangChain正是大语言模型应用开发领域内非常流行的一个框架。它简化了大语言模型应用的开发流程，提供了许多强大的核心组件和功能接口。第2章将更详细地介绍LangChain框架及其使用方法。

第 2 章 LangChain初体验

2

本章主要介绍LangChain框架及其应用场景，包括如何搭建环境和安装框架，最后通过编写一个AI写作工具来初步体验LangChain的魅力。

2.1 LangChain 介绍和安装

LangChain是LLM应用开发的重要工具，值得每一位LLM应用开发者学习和使用。本节将介绍LangChain框架及其关键概念，通过实际操作帮助开发者搭建本地开发环境，展示LangChain的广泛应用场景。

2.1.1 什么是 LangChain

LangChain是一个由Python编写的LLM应用开发框架，主要用于简化使用大语言模型（如GPT-3、Llama等）构建应用程序的过程。它提供了一系列可模块化和可重用的组件，可以组合起来构建复杂的AI应用程序。

LangChain的重要特性包括：

- 模型抽象：LangChain提供了标准化接口，以便在单个应用程序中快速切换不同模型或使用多个模型。
- 数据引入和检索：LangChain支持从各种来源（如文件、数据库和API）引入和检索数据，使LLM能够处理结构化和非结构化数据。

- 链和代理：链（Chain）是指可在LLM上执行的结构化操作序列，这些操作序列（LLM或应用程序）可以形成复杂的多步骤工作流。而代理（Agent）则是更高级的抽象服务模块，能够与LangChain提供的工具和多种数据源交互，以完成特定任务。
- 内存管理：LangChain提供了管理LLM短期和长期内存的机制（Memory），使LLM能够维护较长的上下文，并从先前和用户的交互中获得有效信息。
- 评估和监控：LangChain提供了用于评估和监控LLM性能的工具，可以帮助开发人员找到对应的性能瓶颈并改进其LLM应用服务。例如，大名鼎鼎的LangSmith，我们将在后续章节具体介绍它。

总之，LangChain的出现大幅降低了LLM应用的门槛，使开发人员能够专注于业务逻辑而非底层实现，从而显著提高了LLM应用的开发效率和产品质量。

2.1.2　环境搭建

建议读者在macOS或Linux系统环境下来学习本书内容和运行相关代码。因为许多Python科学计算库和机器学习相关库在Windows环境下存在一些不兼容的情况，为了让开发过程更加顺畅，环境更加稳定，不推荐在Windows环境中进行开发。

对于因为客观原因不得不使用Windows系统的读者，推荐使用WSL（Windows Subsystem for Linux）或Docker来学习和开发LangChain项目。笔者使用的是macOS系统，后续内容也将以macOS为基础进行讲解。

由于LangChain是使用Python编写的框架，因此我们首先需要在本地计算机上安装Python环境，并选择适合的集成环境（IDE）或开发工具，最后根据需要安装LangChain库和相关库。

1. Python 环境

安装Python有多种方式，可直接从Python官方网站下载适用于当前操作系统的安装包。然而，笔者推荐使用Anaconda来安装Python及其相关库。

Anaconda是面向数据科学和机器学习的Python发行版，集成了众多科学计算库和工具，能够简化软件包的管理和部署，并为Windows、Linux和macOS等平台提供一致的使用体验。使用Anaconda不仅能确保无论使用何种操作系统，都能拥有稳定的Python环境和统一的包管理，而且它还支持Python虚拟环境管理，便于在不同版本的Python之间灵活切换，极大地方便了学习和工作。

我们可以通过Anaconda官方网站下载对应的安装包来完成Anaconda的安装。

如图2-1所示，可以根据自己计算机的操作系统选择对应的安装包。因为笔者使用的是macOS系统，所以选择macOS版本的安装包。推荐使用Graphic Installer（图形化安装器），它的安装过程更加简单直观。

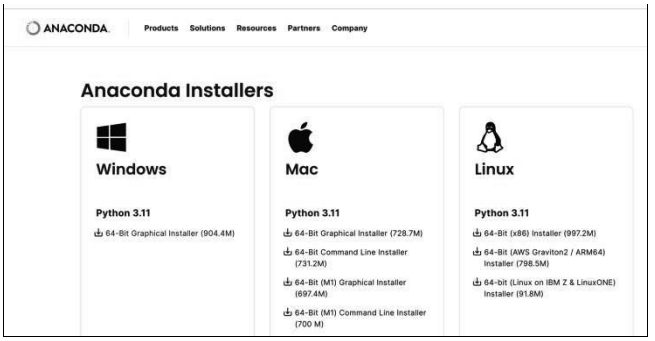

图 2-1　Anaconda 安装包下载界面

安装完成后，可以打开终端（Terminal），执行以下命令来验证Anaconda是否安装：

```
conda --version
```

得到的输出结果如下：

```
conda 23.3.1
```

这表明终端环境能够正确找到conda命令，且当前安装的conda版本为23.3.1。

2. 开发工具

“工欲善其事，必先利其器”。为更高效地开展后续开发工作，建议读者选择一个功能强大的IDE或代码编辑器。有以下两个推荐的选择：

- Visual Studio Code：由微软开发和开源的免费代码编辑器，支持多种编程语言，内置语法高亮和代码自动补全功能，并提供了扩展商城来安装扩展以增强编辑器的能力。其最大的优势是轻量级，开源免费，适用于几乎所有主流编程语言的开发需求。
- PyCharm：它是JetBrains公司推出的专为Python开发者设计的强大IDE（集成开发环境），提供专业的代码分析、图形调试器和集成单元测试工具，还图形化了Git操作界面等，使调试和开发过程更便捷、高效。PyCharm提供了社区版和专业版：社区版免费，适合纯Python项目的开发；专业版支持数据科学和Web开发，内置了相应开发环境的框架。专业版长期使用需要付费，官方提供了30天的免费试用期。

作为LangChain项目的开发者，推荐使用PyCharm作为主要开发工具。PyCharm提供了丰富的功能和高效的工具，使开发者能够更加专注于业务代码的编写，而无须在开发环境的配置和调试上耗费过多的时间。通过PyCharm的强大代码分析和自动完成功能，可以提高编码效率并减少错误。

此外，PyCharm内置的图形调试器和集成单元测试工具，使调试和测试过程更加直观、顺畅，有助于保障LangChain项目的质量和稳定性。PyCharm的插件生态系统也非常丰富，可以

根据后续LangChain项目的特定需求进行定制与扩展，满足个性化开发需求。

PyCharm同时支持多种主流Python框架（如Django和Flask）以及科学计算库（如Pandas和Scikit-learn），为LangChain项目的开发和集成提供了更加广泛的支持和便利。

因为工作和学习的需要，笔者使用的是PyCharm专业版，读者可根据自身情况选择免费的社区版或者付费购买PyCharm专业版。

读者可到JetBrains官方网站下载该软件，下载页面如图2-2所示。官网提供了PyCharm的Windows、macOS以及Linux版本，读者可根据自己的操作系统选择对应的安装包进行下载。

图 2-2　PyCharm 下载页面

3. LangChain的安装和依赖

LangChain框架支持多种安装方式，推荐直接使用pip或conda命令进行快速安装。以下为使用conda的推荐命令：

```
conda install langchain -c conda-forge
```

其中，conda-forge是一个社区驱动的conda软件源，拥有丰富的开源库资源。除了主框架langchain外，还有其他常用的组件库，如langchain-core、langchain-community与langchain-experimental。可以执行以下命令分别安装：

```
pip install langchain-core
pip install langchain-community
pip install langchain-experimental
```

- langchain-core是LangChain框架的核心库，通常在安装主框架时会自动安装，也可单独安装。
- langchain-community是一个集成第三方集成服务的社区库，支持与OpenAI模型的对接。
- langchain-experimental为实验性的LangChain功能库，用于探索尚未发布的LangChain框

架新特性。

如果读者先单独安装过langchain-community，然后安装LangChain主框架库时可能出现依赖版本不兼容的问题，报错提示信息如下：

```
    ERROR: pip's dependency resolver does not currently take into account all the
packages that are installed. This behaviour is the source of the following dependency
conflicts.
    langchain-community 0.0.38 requires langchain-core<0.2.0,>=0.1.52, but you have
langchain-core 0.2.3 which is incompatible.
```

最简单的解决办法是升级langchain-community的版本，执行以下升级命令：

```
    pip install --upgrade langchain-community
```

除此之外，LangChain还提供了相关的工具和库，比如命令行工具LangChain CLI，使用以下命令安装即可：

```
    pip install langchain-cli
```

这条命令将安装LangChain的官方CLI工具，使开发者能够更高效地使用LangChain模板。LangChain CLI提供了一种简洁的方式来操作LangChain模板和其他LangServe项目。通过安装LangChain CLI，开发者可以更有效地管理和使用LangChain框架的各种功能和工具。

安装完成后，即可正常使用langchain命令了。

在实际安装LangChain的过程中，部分开发者可能会遇到如下报错：

```
    Fatal Python error: config_get_locale_encoding: failed to get the locale encoding:
nl_langinfo(CODESET) failed
    Python runtime state: preinitialized
```

该错误通常是由于系统语言环境设置不当所致。可通过以下命令将终端的LANG环境变量设置为英文：

```
    export LANG="en_US.UTF-8"
```

需要注意的是，这种设置是临时性的。当关闭当前终端窗口，或在一个新的终端中运行langchain命令时，该问题再次出现。因此，建议将编码设置写入用户配置文件以永久生效，编辑~/.profile文件，在文件末尾添加如下配置：

```
    export LC_CTYPE="en_US.UTF-8"
```

保存后，重启终端或打开一个新的终端窗口即可彻底解决编码问题。

对于想深入学习LangChain源码的开发者，可以选择源码安装方式。首先，通过Git命令克隆LangChain的代码仓库到本地，在终端执行如下命令：

```
    git clone https://github.com/langchain-ai/langchain.git
```

然后在当前目录下进行本地源码的编译安装：

```
pip install -e .
```

除了Python版本的LangChain框架外，LangChain官方还推出了JavaScript版本。由于越来越多的科技公司选择使用Node.js开发新的商业项目，因此在开发LangChain应用时，也会选择保持统一的技术栈。

与Python环境中的多版本管理工具pyenv类似，Node.js也提供了nvm来管理多个Node.js版本。nvm的安装非常便捷，只需要在终端中执行以下命令：

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
```

Node.js安装LangChain非常简单，使用npm命令即可：

```
npm i langchain
```

2.1.3　LangChain 的应用场景

LangChain是一个基于大语言模型（LLM）的开发框架，支持与多种主流大语言模型集成交互，帮助开发者快速开发强大的AI服务。其应用场景非常广泛，涵盖多个行业与领域：

- GPT模型：可以构建出多模态能力的GPT模型。比较知名的项目是AudioGPT。它可以同时处理语音和文本，完成语音识别、文本转语音、音频剪辑和生成等复杂任务。基于AudioGPT，可以开发语音助手、播客创作等应用。
- 智能客服：越来越多的公司基于LangChain开发智能客服，往往使用业务数据进行预训练，让机器人更懂产品和服务。例如，AWS推出的云服务咨询机器人可以解答用户关于AWS相关服务的各种问题。智能客服将逐步替代初级人工客服，甚至最终替代所有人工客服，帮助企业以更低的成本提供更优质的产品咨询、售后服务。
- 聊天机器人：无论是企业还是个人，都可以利用LangChain打造专属的聊天机器人。例如，GitHub上有一个非常受欢迎的基于Notion文档的问答机器人项目。该项目基于Langchain开发并集成了OpenAI的大语言模型，可以针对用户的Notion文档进行提问。
- 内容生成（AGCI）：根据用户需求自动生成高质量的推文、产品软广等内容，帮助企业快速创作爆款文案和文章，从而节省营销成本。例如，国内一些自媒体公司根据关键词和主题自动生成热门的小红书软文，迅速积累粉丝和点赞，为产品推广助力。
- 数据分析：LangChain可以通过大语言模型进行数据分析，生成具有商业价值的分析报告。一些国外顶级投行公司已在某些较为成熟的投资领域尝试使用LangChain生成一个投资报告自动生成系统，帮助投资分析师从海量的研报中获取有用的信息，从而找到有效的投资标的。
- 低代码服务：LangChain本身是一个简化的LLM开发框架，也可以开发可视化的LLM

开发平台。目前最著名的例子是LangFlow，它提供了可视化的工具来构建多代理和RAG应用。

2.2　小试牛刀：开发一个 AI 文章生成工具

“纸上得来终觉浅，绝知此事要躬行”。在了解到LangChain的强大功能之后，我们决定哦尝试开发一个AI文章生成工具，以此来体验LangChain开发LLM应用的标准流程。

假设你是一名在娱乐传媒公司工作的自媒体小编，每天需要根据热门话题编写公众号文章和小红书推文。今天是娱乐圈盛典的开幕式，出现了多个热门话题：星光之夜红地毯、某女明星的新电影、流行歌手A出了新专辑等。由于工作繁忙，你已经忙不过来了，无法逐一亲自写稿。于是，你想到了求助于高中同学极客君，他建议使用LangChain开发一个自动生成文章的LLM应用。

该LLM应用主要包含以下功能：

（1）根据话题（Topic）生成文章标题。

（2）根据文章标题生成文章内容。

技术架构设计如图2-3所示。用户输入热门话题后，LangChain通过PromptTemplate生成对应的提示词（Prompt），再调用OpenAI的大语言模型来生成文章标题。为了简化开发过程，本次开发没有引入预训练和相关的文章数据集。OpenAI的大语言模型有多个版本，目前比较新的版本是GPT-4o和GPT 4.5。然而，对于本项目而言，使用GPT-3.5-turbo已经足够了。在用户交互方面，笔者选择使用Python的streamlit库，该库不用专门编写HTML和CSS代码，提供了丰富的UI组件，可以便捷地生成美观且功能复杂的网页和视图。

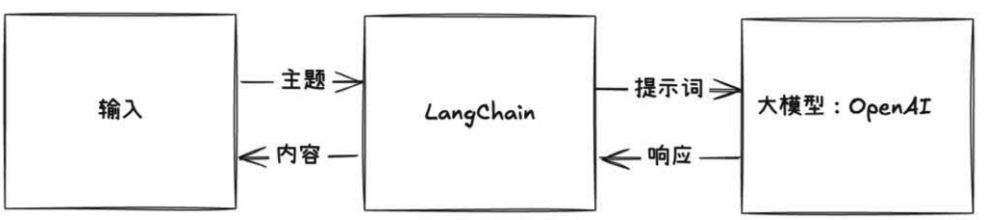

图 2-3　AI 生成文章技术框架图

2.2.1　初始化项目和配置

首先，在PyCharm中创建一个新的项目，命名为article-generator，如图2-4所示。设置使用conda安装Python 3.8版本作为运行环境。对于每个项目，建议使用conda创建一个单独的虚拟运行环境，以确保运行环境的一致性和稳定性。

图 2-4　初始化项目界面

在IDE中打开一个终端，执行**`python --version`**命令，检查Python版本，如图2-5所示，确认Python环境已按照要求生效。

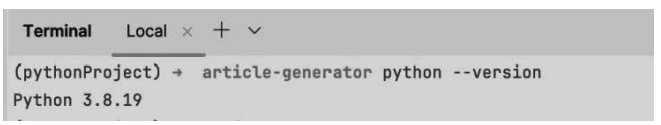

图 2-5　Python 版本检查

为了能够在后续工作中使用OpenAI的GPT-3.5-turbo模型，事先准备好OpenAI API密钥至关重要。如果能直接支付OpenAI账单，可登录OpenAI平台后台生成API key。对于无法直接完成支付的开发者，可以通过电商平台购买稳定的API key。在项目根目录下创建.env文件，并填写准备好的OpenAI的key（密钥）：

```
OPENAI_API_KEY=sk-xewrewrwerwerwerewe
```

安装所需的库可以通过**`pip install lib1, lib2, lib3...`**命令的方式安装，但更专业的做法是创建requirements.txt文件，将项目所需的所有库列入该文件，内容如下：

```
langchain
```

```
python-dotenv
streamlit
```

然后在终端执行命令来安装这些库：

```
pip install -r requirements.txt
```

2.2.2　编写标题生成服务

初始化工作完成之后，正式开始编写代码。首先，我们需要一个UI界面，让用户输入想要写的热门话题。可以使用streamlit来实现这个简单页面，代码如下：

```
import streamlit as st
# 设置页面显示的标题
st.title('热门文章AI生成器')
# 生成一个文本框，里面带有提示信息
topic = st.text_input('请输入你想写的话题：')
```

在IDE的终端执行streamlit命令运行Web服务：

```
streamlit run app.py
```

程序会自动唤起浏览器，并跳转到http://localhost:8501，可以看到如图2-6所示的页面。

图 2-6　热门文章生成器页面

在输入框中输入话题之后，变量topic就可以读取输入内容。

如果想用PyCharm内置的运行按钮运行streamlit编写的项目，则需要单独配置。单击界面右上角的▮按钮，选择编辑配置，然后在弹窗中添加一个Run/Debug Configuration，设置以module（模块）方式运行，然后在Options中设置run app.py。整个设置如图2-7所示。

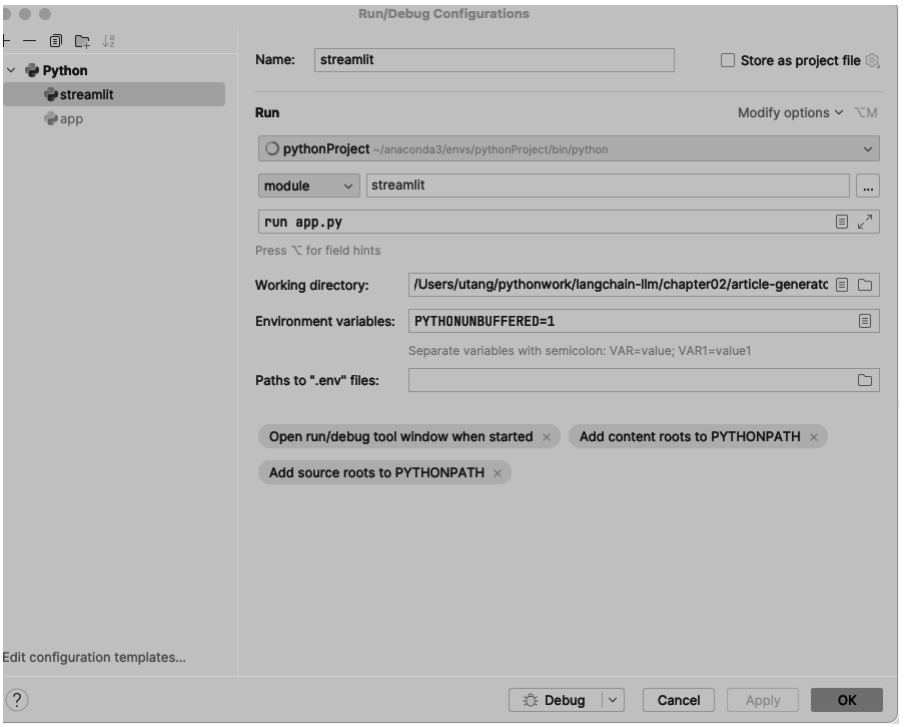

图 2-7　PyCharm 设置 streamlit 命令运行配置

以这种方式运行的最大好处是可以利用PyCharm对代码进行断点调试。

随后，需要用到LangChain调用OpenAI模型，这需要单独安装langchain-openai库，该库是Langchain框架对OpenAI的集成库，封装了OpenAI模型的常用功能。运行如下命令进行安装：

```
pip install -qU langchain-openai
```

命令参数解析：

- -q参数代表quiet，使安装过程处于安静模式，不会输出安装过程中的详细信息，只会显示关键信息。
- -U参数代表upgrade，确保安装的是最新版本的库，如果已经安装了旧版本，则会自动升级到最新版本。

langchain-openai提供了针对聊天模型的ChatOpenAI类和LLM模型的OpenAI类。我们先体验一下OpenAI类。

安装完毕后，我们可以生成一个OpenAI的LLM对象，并尝试使用它来回答问题，将下面的代码保存到一个名为test_openai.py的文件中：

```
import os
from dotenv import load_dotenv
```

```
from langchain_openai import OpenAI

# 默认从当前目录下的.env文件加载环境变量
load_dotenv()
# 从环境变量中获取OpenAI的API key
api_key = os.getenv("OPENAI_API_KEY")
# 初始化OpenAI大模型对象，选择使用gpt-3.5-turbo-instruct模型
llm = OpenAI(api_key=api_key, model_name="gpt-3.5-turbo-instruct")
# 向AI询问是否知道大熊猫花花的故乡
question = "What is the hometown of the hua hua panda?"
response = llm.invoke(question)
# 打印出大模型的回复
print(response)
```

关于OpenAI可用的模型，可以参考官方网站的列表：

```
https://platform.openai.com/docs/models/codex
```

在此我们选择使用gpt-3.5-turbo系列中的gpt-3.5-turbo-instruct，官方网站文档的介绍如图2-8所示。

gpt-3.5-turbo-instruct	Similar capabilities as GPT-3 era models. Compatible with legacy Completions endpoint and not Chat Completions.	4,096 tokens	Up to Sep 2021

图 2-8　gpt-3.5-turbo-instruct

简单来说，这个模型的主要目标是有效地遵循指令。它不是一个用于对话的模型，而是一个面向任务的模型。与聊天模型有所不同，它在提供精确响应方面异常高效，更适合完成一些指令式的任务。

接下来，使用Python命令执行该脚本，输出结果如图2-9所示。

```
(ai-coding-helper) ➜  article-generator git:(master) ✗ python test_openai.py


Hua hua pandas do not have a specific hometown as they are a rare and endangered species found in the Sichuan, Shaanxi, and Gansu provinces of China.
```

图 2-9　运行结果

很容易发现，它默认回复的语言和问题所用的语言保持一致。如果我们选择用中文提问，它也会自动用中文回复：

```
大熊猫花花的故乡是中国四川省成都市周边的秦岭山脉和四川盆地一带。
```

我们把OpenAI大模型正式引入标题生成的服务中，更新后的代码如下：

```
import streamlit as st
import os
from dotenv import load_dotenv
```

```
    from langchain_openai import OpenAI
    # 默认从当前目录下的.env文件加载环境变量
    load_dotenv()
    # 从环境变量中获取OpenAI的API key
    api_key = os.getenv("OPENAI_API_KEY")
    # 初始化OpenAI大模型对象，选择使用gpt-3.5-turbo-instruct模型
    llm = OpenAI(api_key=api_key, model_name="gpt-3.5-turbo-instruct",
temperature=0.9)

    # 设置页面显示的标题
    st.title('热门文章AI生成器')
    # 生成一个文本框，里面带有提示信息
    topic = st.text_input('请输入你想写的话题：')

    if topic:
        response = llm.invoke(topic)
        print(response)
```

上述代码中，在生成OpenAI类的实例时设置了temperature参数为0.9。temperature也被称为模型温度，用于控制大语言模型生成文本时的创造性和多样性，其取值范围为0~2的正数。较低的温度倾向于选择高概率词语，从而生成更具确定性的输出；较高的温度则会引入更多的随机性，从而可能产生更具创造性和多样化的结果。一般而言，0.7~0.9适用于创造性任务，1.0会生成更多样化的输出。鉴于本任务属于创造性文本生成，设置为0.9是较为合适的选择。

显然，直接把输入的话题（topic）传递给大语言模型（LLM）是不够的。LLM拿到一个话题，如“流行歌手A”，无法判断用户是想查询该歌手的资料，还是基于该话题生成文章标题。为此，我们需要通过提示词来引导LLM完成具体任务。

为了避免每次修改代码后都需要手动重新启动streamlit应用，可以安装watchdog库来监听文件修改并自动重新加载服务。可使用如下命令进行安装：

```
pip install watchdog
```

在此基础上，我们可以在调用LLM之前增加引导词（Prompt），修改的代码如下：

```
    if topic:
        prompt = '请根据' + topic + '编写一个有创意的标题'
        response = llm.invoke(prompt)
        # 将AI生成的标题输出到网页上
        st.write(response)
```

引入提示词后，OpenAI大模型可以明确理解我们希望基于输入的话题生成一个富有创意的标题。

得益于watchdog库的使用，我们无须重启脚本，只需刷新网页，并在页面上的话题中输入“苹果醋”并按Enter键，等待一两秒，即可看到如图2-10所示的标题生成结果。

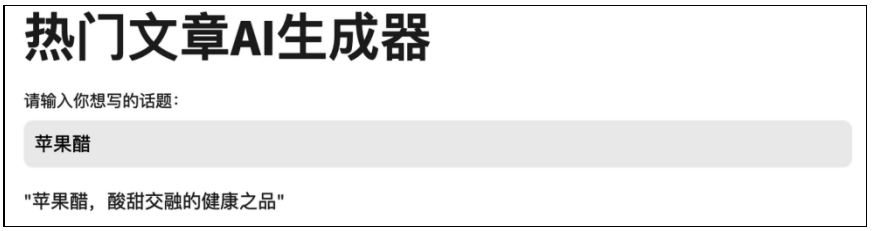

图 2-10　标题生成结果

在实际的LLM开发中，我们不推荐将提示词硬编码在代码中。主要原因有两个：一是缺乏灵活性，提示词一旦修改需要把服务重新部署到服务器（无论是部署到云服务器还是进行私有化部署）；二是LangChain提供了强大的提示词模板功能PromptTemplate，支持通过配置化、参数化的方式构建提示词。

借助PromptTemplate，我们可以灵活组装提示词，比如控制生成标题数的量（如一次生成3个），也可以指定生成的语言（如英文标题），从而提升效率与多样性。

修改后的完整代码如下：

```
    import streamlit as st
    import os
    from dotenv import load_dotenv
    from langchain_openai import OpenAI
    from langchain_core.prompts import PromptTemplate
    # 默认从当前目录下的.env文件加载环境变量
    load_dotenv()
    # 从环境变量中获取OpenAI的API key
    api_key = os.getenv("OPENAI_API_KEY")
    # 初始化OpenAI大模型对象，选择使用gpt-3.5-turbo-instruct模型
    llm = OpenAI(api_key=api_key, model_name="gpt-3.5-turbo-instruct",
temperature=0.9)

    # 设置页面显示的标题
    st.title('热门文章AI生成器')
    # 生成一个文本框，其中带有提示信息
    topic = st.text_input('请输入你想写的话题：')

    if topic:
        # 创建一个提示词模板
        prompt_template = PromptTemplate.from_template("请用{language}根据话题：
{topic}编写{count}个有创意的标题")

        # 格式化提示词并添加变量
        prompt = prompt_template.format(topic=topic, count=3, language='英文')
        # 将提示词传给OpenAI类来执行
        response = llm.invoke(prompt)
```

```
        # 将AI生成的标题输出到网页上
        st.write('生成的标题如下:\r' + response)
```

在页面中输入话题“LangChain技术”后运行程序，将看到如图2-11所示的一次生成3个英文标题的结果。

图 2-11　“LangChain 技术”的标题生成结果

简单总结一下，使用PromptTemplate有以下明显优势：

- 可重用性：PromptTemplate允许我们定义一次模板，然后在多个地方重复使用，避免重复编写相同的提示逻辑，提高代码的可维护性和可重用性。
- 关注点分离：PromptTemplate将提示格式化与模型调用分离，使代码更加模块化。这样开发者可以独立更改模板或模型，而不影响其他部分的代码。
- 动态提示：模板允许开发者根据需要动态生成提示，填充模板变量，适用于根据用户输入或运行时条件自定义构建提示。
- 可读性：模板可以将复杂的提示逻辑封装在简单的接口中，提高代码的可读性。命名变量通常更易于理解和维护。

2.2.3　编写文章生成服务

生成标题之后，我们可以基于标题进一步生成文章。一般来说，一篇热门文章的篇幅控制在800字以内，所以生成文章时也将字数限制在800字左右。

我们只需继续使用PromptTemplate来生成提示词模板和最终的提示词。为了更为直观和简洁地演示，之前生成3个标题并返回英文的过程将改为只生成1个标题，并返回中文。关键代码如下：

```
    if topic:
        # 创建一个提示词模板
        prompt_template = PromptTemplate.from_template("请用{language}根据话题:
{topic}编写{count}个有创意的标题")

        # 格式化提示词并添加变量
```

```
        prompt = prompt_template.format(topic=topic, count=1, language='中文')
        # 将提示词传给OpenAI类来执行
        title = llm.invoke(prompt)
        # 将AI生成的标题输出到网页上
        st.write('生成的标题如下:\r' + title)
        if title:
            # 创建生成文章的提示词模板
            prompt_template = PromptTemplate.from_template(
                "请根据标题：{title}，生成一篇论述清晰的{language}文章，字数{chars}个字以内，
如果超出请进行精简。"
            )
            prompt = prompt_template.format(title=title, language='中文', chars=800)
            # 必须设置大一点的最大token数，以避免生成的文章内容被截断
            article = llm.invoke(prompt, max_tokens=4000)
            print(article)
            st.write('生成的文章内容：\r' + article)
```

刷新页面，重新输入话题“诸葛亮”，按Enter键后，可以看到如图2-12所示的结果。

图 2-12　文章生成结果图

大模型生成的文章语言流畅，创意丰富紧扣主题，质量相当不错。

需要注意的是，在初始化OpenAI类的实例时设置了max_tokens参数。这个参数用于指定生成文本的最大token数量。由于中文的一个字可能对应多个token，因此我们需要把max_tokens设置得远大于800。如果不设置max_tokens参数，可能会导致生成的文本不足800个字或被截断。同样，生成英文版本的文章时，也需要适当设置更大的max_tokens值。

2.2.4　多链合并

在LangChain中，每个步骤的任务可以视为一个链（Chain）。在我们的AI文章生成项目中，有两个主要步骤：

步骤01 根据话题生成标题。

步骤02 根据上一步生成的标题进一步生成文章内容。

因此，对应的是两个链，并且这两个链是一个顺序链（Sequential Chain）。顺序链按顺序执行每个链，并将其封装成可复用的功能模块，它的流程图如图2-13所示。

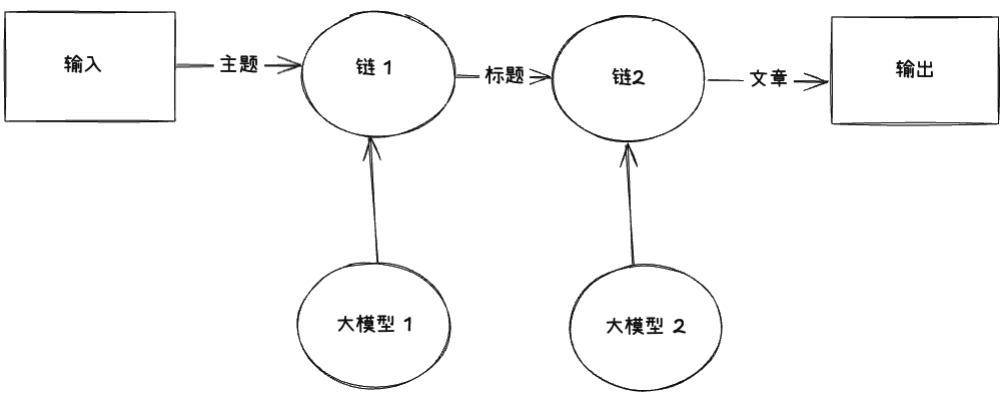

图 2-13　文章生成的顺序链流程图

由图2-13可知，在不同的链中，可以根据业务需求使用不同的LLM来完成相应的任务。例如，对于生成标题，我们选择使用指令式的模型gpt-3.5-turbo-instruct，而对于生成文章内容，可以选择文本能力更强的聊天模型gpt-3.5-turbo。

LangChain框架提供了顺序链这样的功能，下面将2.2.3节的代码改为顺序链来实现：

```
    from langchain.chains import LLMChain, SequentialChain
    from langchain_openai import ChatOpenAI
    # 省略其他逻辑代码
    # 初始化OpenAI大模型对象，选择使用gpt-3.5-turbo-instruct模型
    llm = OpenAI(api_key=api_key, model_name="gpt-3.5-turbo-instruct",
temperature=0.9)
    chat_llm = ChatOpenAI(api_key=api_key, model_name="gpt-3.5-turbo",
temperature=0.9)
```

```
    # 设置页面显示的标题
    st.title('热门文章AI生成器')
    # 生成一个文本框，里面带有提示信息
    topic = st.text_input('请输入你想写的话题：')

    if topic:
        # 创建一个提示词模板
        prompt_template = PromptTemplate.from_template("请用{language}根据话题：
{topic}编写一个有创意的标题")
        prompt = """
                 你是一个擅长撰写各种文体的自媒体编辑，
                 请用{language}根据话题：{topic}编写一个有创意的标题。
                 """
        prompt_to_create_title = PromptTemplate(
            input_variables=["topic", "language"],
            template=prompt)
        # 格式化提示词并添加变量
        # 创建一个链来生成title
        chain_to_create_title = LLMChain(llm=llm,
                                         prompt=prompt_to_create_title,
                                         output_key="title", verbose=True)
        # 创建生成文章的提示词模板
        prompt_template_for_article = """
              你是一个擅长编写各种文体的自媒体编辑，
              请根据标题：{title}，生成一篇论述清晰的{language}文章，字数{chars}个字以内，
如果超出请进行精简。
              """

        prompt_to_create_article = PromptTemplate(
            input_variables=["title", "language", "chars"],
            template=prompt_template_for_article)

        chain_to_create_article = LLMChain(llm=chat_llm,
prompt=prompt_to_create_article, output_key="article")

        overall_chain = SequentialChain(
            chains=[chain_to_create_title, chain_to_create_article],
            input_variables=["topic", "language", "chars"],
            # 在这里返回多个变量
            output_variables=["title", "article"],
            verbose=True
        )
        response = overall_chain({"topic": topic, "language": "中文", "chars": "100"})
        # 将AI生成的标题输出到网页上
```

```
    st.write('生成的标题如下:\r' + response.title)
    st.write('生成的文章内容：\r' + response.article)
```

这段代码使用了另一种模板组装形式，通过原样格式语法，并利用input_variables将变量传递给原样字符串。最后，通过overall_all函数传入后续两个链中所用到的参数，按照顺序执行链中的大模型任务，上一个链的输出值title被传递给下一个链作为入参，最后返回大模型生成的title和article并输出到页面上。

由此可见，使用LangChain的顺序链具有以下好处：

- 逻辑封装：顺序链可以将多个任务按顺序连接在一起，形成一个完整的逻辑流程，便于管理和维护。
- 代码复用：顺序链可以将多个任务组合成可重复使用的模块，提高了代码的复用性和可维护性。
- 简化调用：顺序链可以将多个任务组织在一起，简化了调用过程，使代码结构更加清晰。
- 错误处理：顺序链可以更好地处理任务之间的依赖关系和错误，确保任务按顺序执行并处理异常情况。
- 性能优化：顺序链通过优化任务的执行顺序，提高整体性能和效率。

通过使用LangChain的顺序链，开发者可以更好地组织和管理任务流程，提高代码的可读性和可维护性。

SequentialChain提供的功能远不止本案例所展示的，它还支持复杂的业务流程、条件分支、循环等控制结构。下面是一个包含条件分支的例子：

```
from langchain.chains import LLMChain, SequentialChain
# 创建第一个LLMChain实例
chain1 = LLMChain(llm=model1, prompt=prompt1, output_key="output1")

# 创建第二个LLMChain实例
chain2 = LLMChain(llm=model2, prompt=prompt2, output_key="output2")

# 创建第三个LLMChain实例
chain3 = LLMChain(llm=model3, prompt=prompt3, output_key="output3")

# 创建一个条件判断函数
def condition_check(output1):
    if output1 == "desired_output":
        return True
    else:
        return False
```

```
# 创建一个包含条件判断的SequentialChain
overall_chain = SequentialChain(
    chains=[chain1, chain2, chain3],
    input_variables=["input_data"],
    output_variables=["output1", "output2", "output3"],
    verbose=True
)

# 定义输入数据
input_data = "your_input_data"

# 运行SequentialChain并获取输出
outputs = overall_chain.run(input_data)

# 提取第一个链的输出
output1 = outputs["output1"]

# 根据条件判断输出，执行相应的操作
if condition_check(output1):
    # 执行特定操作
    print("Condition met, performing specific task.")
else:
    # 执行其他操作
    print("Condition not met, performing alternative task.")
```

在这个例子中，我们创建了3个链，并将它们组合成一个SequentialChain。通过定义条件判断方法condition_check，根据chain1返回的output1来判断是否执行条件分支。

对于有上下游依赖关系且需要顺序执行的LLM应用，优先选择使用SequentialChain来实现。LangChain还提供了更简化的SimpleSequentialChain类，其语法和SequentialChain类似，最大的不同在于：SimpleSequentialChain只支持一个入参和一个出参。

下面是一个执行多个链的代码：

```
overall_chain = SimpleSequentialChain(chains=[chain1, chain2], verbose=True)
overall_chain.run(topic, language, chars)
```

2.3　LLM 开发的工作原理和标准流程

开发完AI文章生成工具之后，我们应该对常规的LLM应用开发有了一定的了解。LLM应用和其他软件开发有共通之处，但也有其独特性，下面内容为LLM开发的工作原理和标准流程。

1. 模块划分和功能点梳理

任何项目都应具有范围清晰的功能模块和功能列表，理清功能模块和功能点有助于开发者更好地进行开发。例如，本次开发的AI文章生成工具，主要有两个功能点：① 根据话题生成具有创意的标题；② 根据创意标题生成创意文章。因为功能较为简单，所以只能抽象出一个功能模块——文章模块。

对于更复杂的项目，比如电商网站，可以先从功能模块拆分为购物车模块、订单模块、支付中心模块、库存管理模块等。每个模块再进一步拆分功能点，以订单模块为例，包含的功能点有：订单创建、订单更新、订单详情查看、订单列表。

2. 选择合适的技术栈和技术架构

根据不同的项目和技术积累，选择适合的技术栈和技术架构来完成开发。大多数情况下，LangChain项目优先推荐使用Python进行开发。如果你的技术团队更多使用其他编程语言（如Node.js），那么也不用舍近求远来选择Python，LangChain同样支持Node.js。技术架构不仅涉及软件本的选择，还包括硬件和云平台的选择。例如，若你开发的LLM应用有较大的访问流量且流量大小分布较为随机，那么AWS云服务中的Lambda会是一个不错的选择。

3. 画出业务流程图或时序图

“好记性不如烂笔头”，把自己大脑中的流程画出来，能有效梳理出业务细节，并帮助开发者更直观地理解业务流程。

4. 迭代式编写代码

在2.2.2节中，我们一步一步推导并优化代码，降低了编程的心智压力。优秀的代码通常是通过不断迭代出来的，并非一步到位编写而成。每次迭代都会有少量修改，这样做可以让代码审核人员更轻松地检查每轮修改。

5. 自测和单元测试

充分的自我测试和单元测试能确保业务逻辑的正确和结果符合预期。不少公司为了追求开发效率，甚至忽视代码质量直接上线，一旦出现重大问题，将给个人和团队带来巨大损失。与其亡羊补牢，不如从一开始就做好功能测试，编写覆盖业务逻辑和边界场景的单元测试。

6. 持续优化和改进

发布只是软件生命周期的开始，而非结束。每个项目都可以通过反思找到改进的空间。例如，本节中提到的AI文章生成工具项目可进行诸多改进：

（1）增加文章生成后写入数据库的操作，这样可以持久化保存这些文章，以备后续使用。甚至可以建立AI文库进行管理，方便个人和公司长期使用。

（2）增加聊天功能，方便用户通过聊天方式不断改进文章内容。

（3）在UI上增加设置一次性生成文章数量和生成语言选项的功能，提高用户生成文章的效率。

（4）大模型的template（模板）选择可以做成UI上的选项方式，类似于Bing界面上的Copilot，提供不同的模式供用户选择。