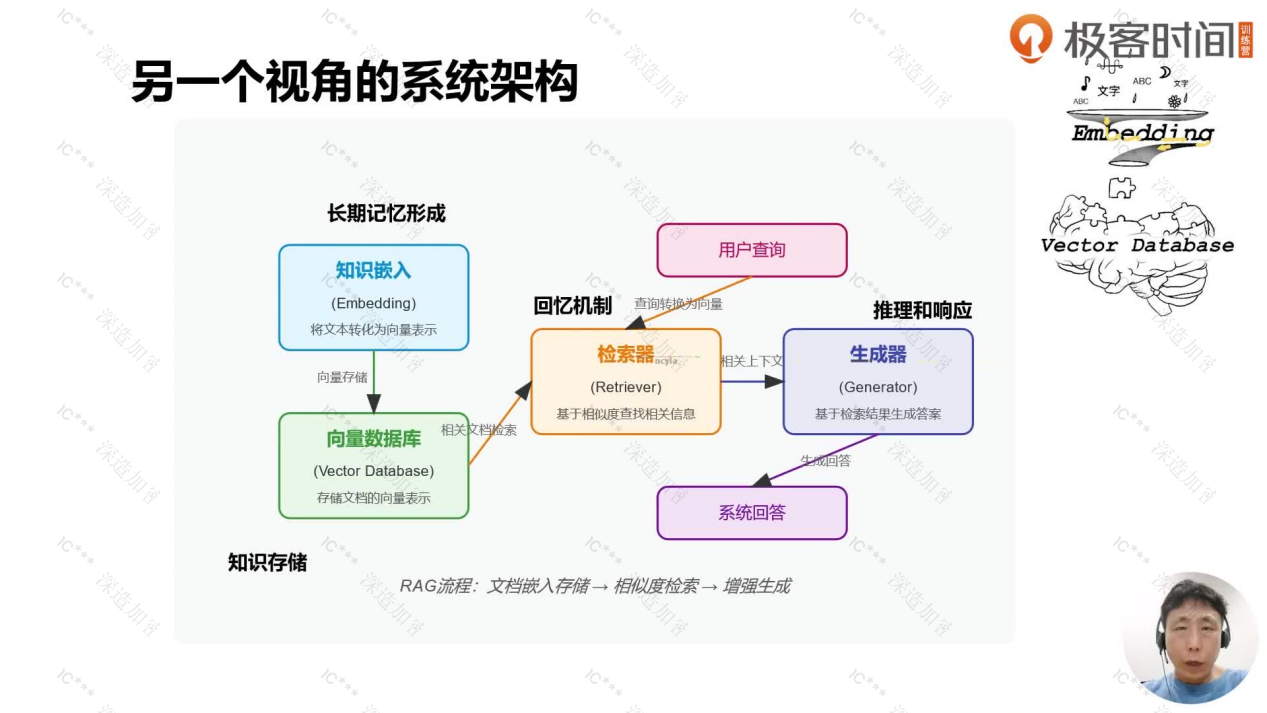
# RAG基础和项目落地

1. RAG三问
2. RAG整体框架和技术组件

1.1 AI时代的应用开发范式发展



1.2 RAG的基本框架

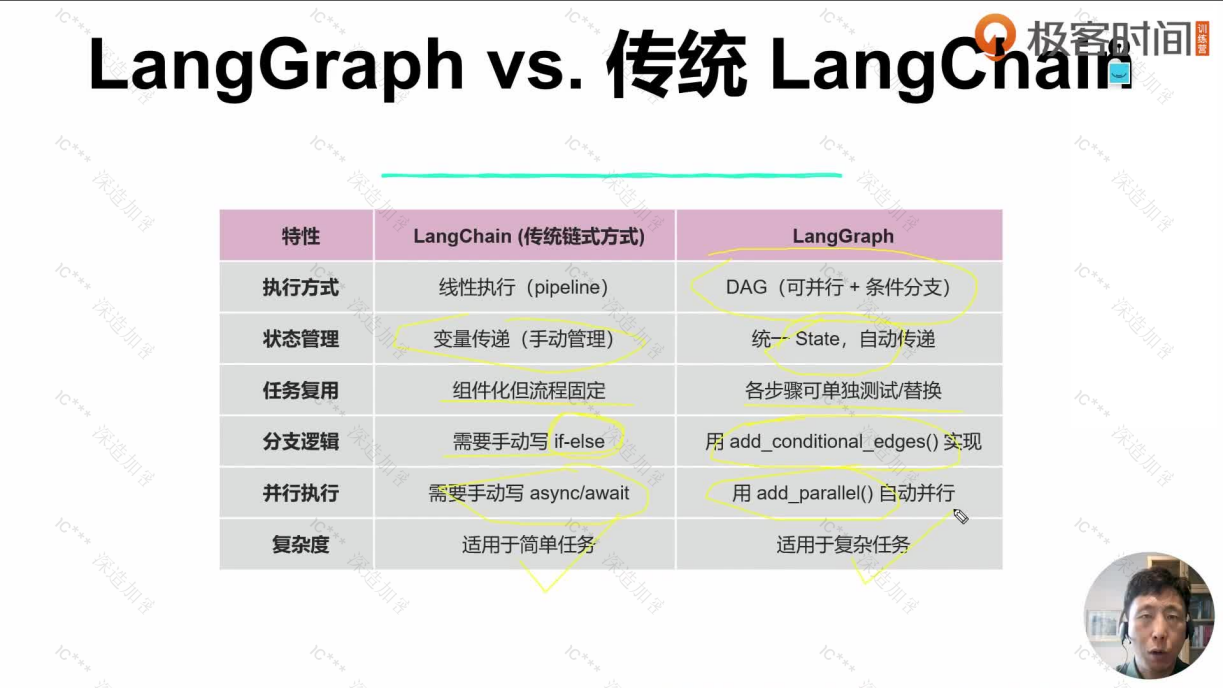


1.3 四大核心组件：知识嵌入、向量数据库、检索器、生成器(LLM为核心)，最重要是知识嵌入和检索器这两部分

2. 快速搭建RAG

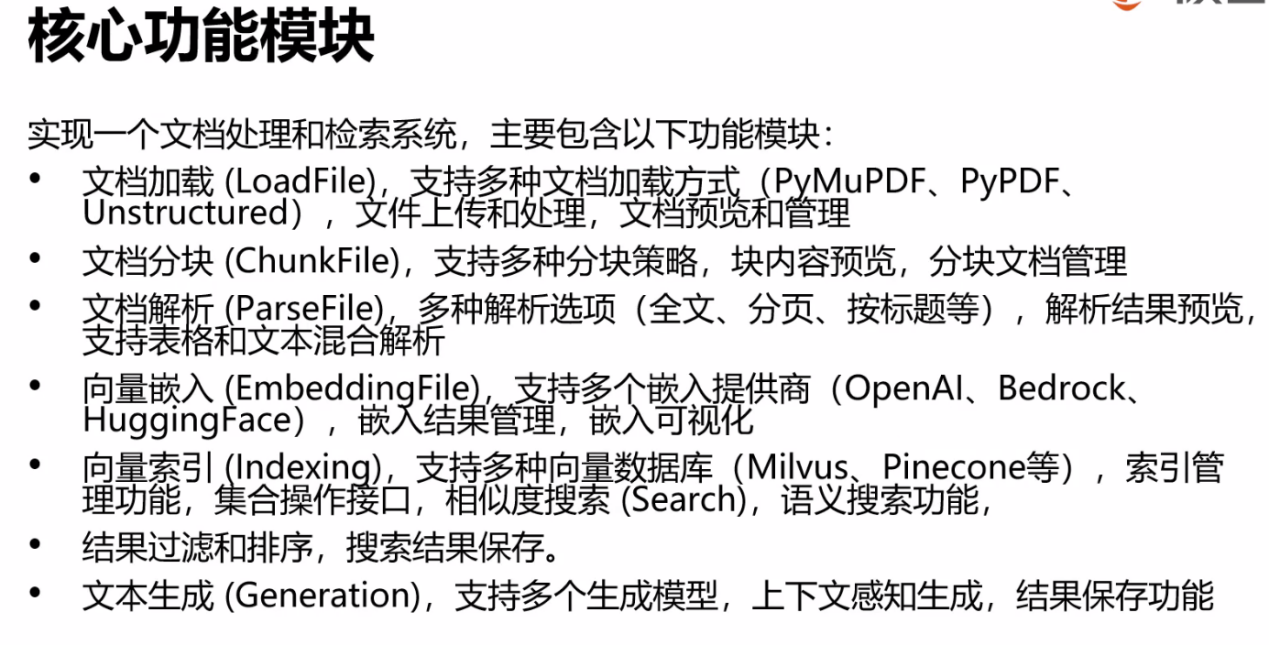
方法：Llmaindex、langchain、LCEL、LangGraph、手动搭建

对比：



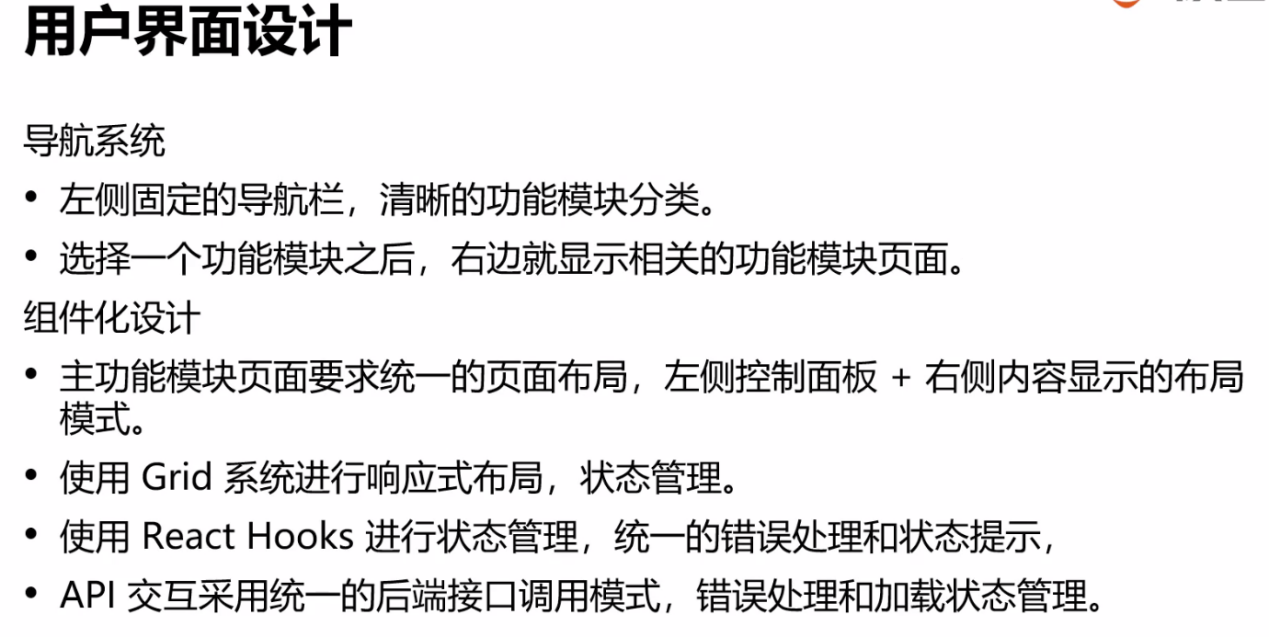
二. 构建RAG系统

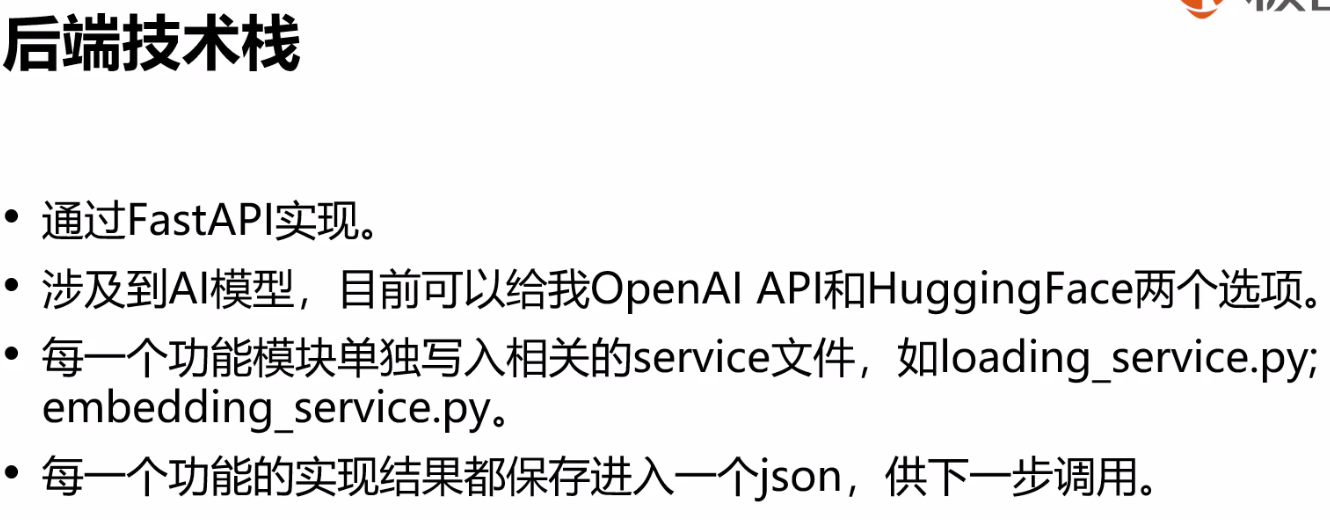
1. 核心功能



2. 前端架构



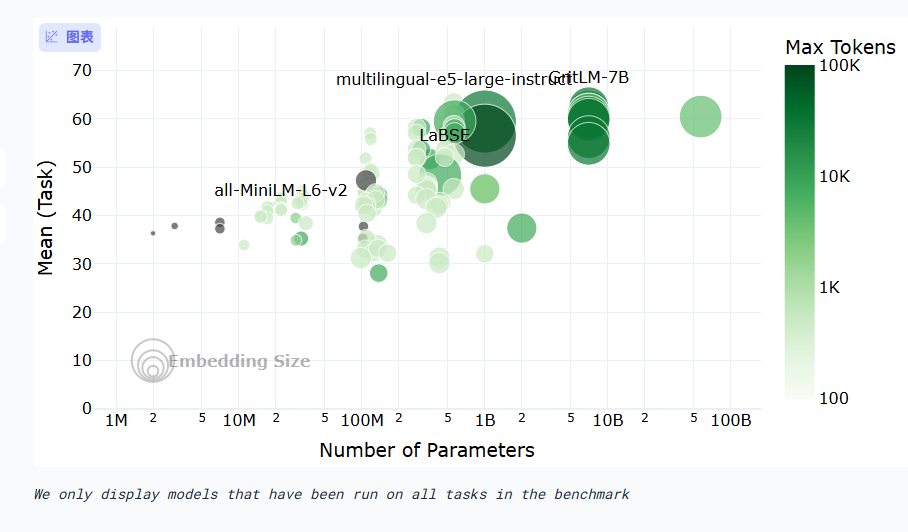


1. 后端架构

三. 项目---医疗系统

1. embedding模型选型

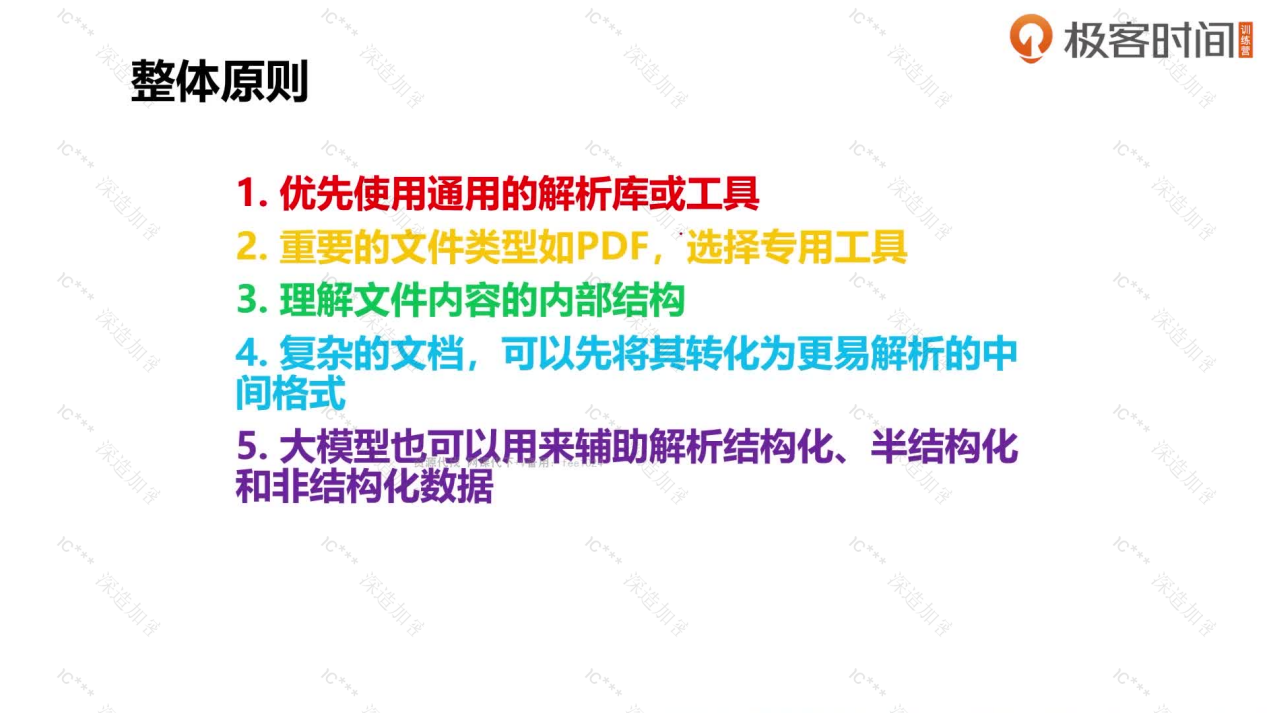
可到mteb查看效果较好的模型, 通常7B属于很大

https://huggingface.co/spaces/mteb/leaderboard 

越靠右越大, 越考上分数越高, 但目前效果最好的是openai的text-embeding-3-large收费模型

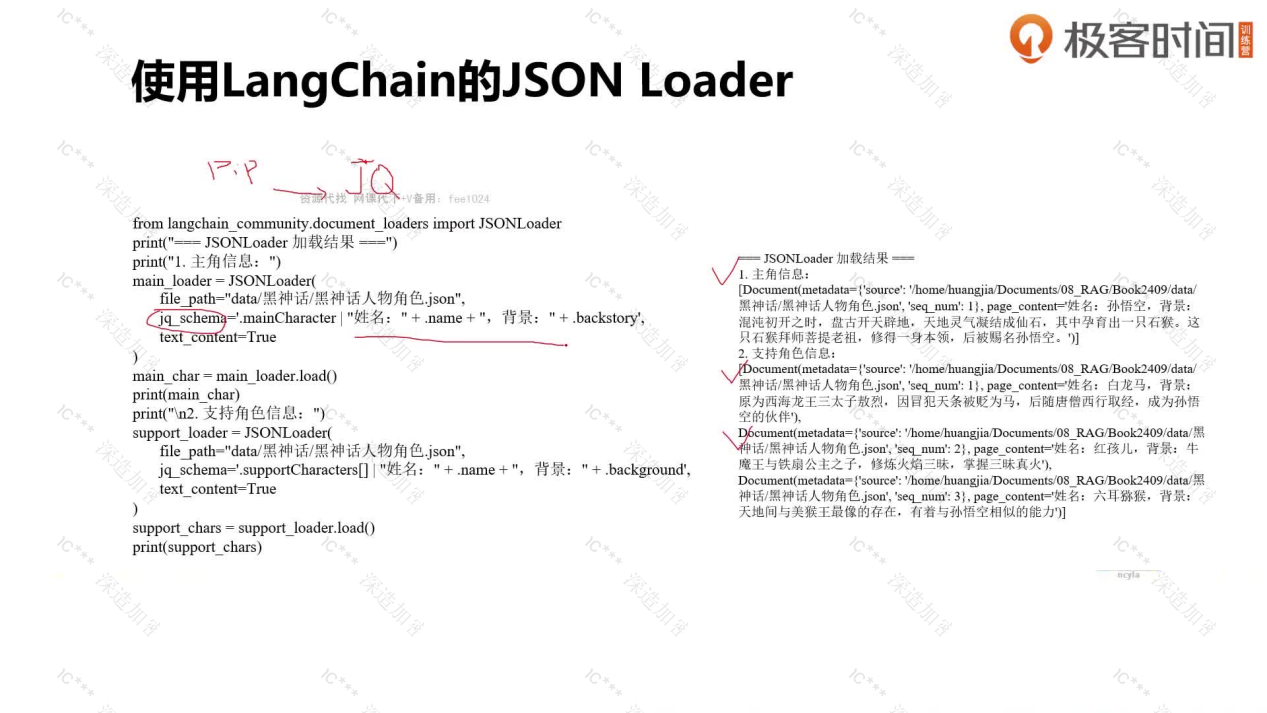
注: 获取不同嵌入模型的维度不需要手动制定, 可以直接嵌入一个字符串, 从结果获取

1. 组件——数据导入
2. 整体原则



1. 结构化提取

2.1 Json可设定jq\_shema进行一定格式的提取



2.2 UnstructuredLoader可对url或html等文件进行提取，能获取元素间的关系

UnstructuredMarkdownLoader可对markdown进行提取，能获取元素间的关系

上述注意处理父子元素等，放在一起保存

1. 图文数据的提取
   1. UnstructuredImageLoader可直接OCR识别图片中的文字，但效果一般
   2. unstructured.partition.ppt可读取PPT并识别文字
   3. from pdf2image import convert\_from\_path将PDF转成图片后，再调用多模态大模型进行图片含义的理解
2. PDF解析(重点)



主要解析方法：

基于规则(pyPDF, pyMUPDF)

基于深度学习(pytesseract+pdf2image属于OCR)

基于大模型的理解

而unstructured属于规则和深度学习的结合，更实用

4.1 pdf转markdown: Marker，MinerU和LlamaParse均是通过深度学习OCR等方法

4.2 pdf解析: unstructured可结合fitz进行页面布局的可视化，衡量解析的效果; unstructured具备多种分块策略和解析模式，如hi\_res是高精准度的解析模式，by\_title是基于标题的分块; 解析后也可自行处理和归纳父子元素

5. 表格数据解析

CSV文件、数据库:

5.1 from langchain\_community.document\_loaders import CSVLoader比from langchain\_community.document\_loaders import UnstructuredCSVLoader效果好，前者能自动识别csv的表头，后者则会全部放到一个page\_content中

5.2 llamaHUB中能找到别人发布的导入数据的包，如读数据库则可用from llama\_index.readers.database import DatabaseReader

PDF表格:

5.3 camelot, pdfplumber都是传统的解析表格的工具，但是仅能解析表格，无法获取表格上下文, 但pdfplumber解析表格更齐全，更快，安装简单

5.4 unstructured有两种方法，一是直接elements作为table, 根据窗口逻辑，获取往前n个元素作为上下文，一种是直接推断表结构, infer\_table\_structure=True则将表格区域转成markdown，则表格数据会变成html之类的元素，数据界限分明

5.5 llamaparser 解析表格并给出认为是表格的文本标题，并转成markdown格式

注：bounding box根据区域大小去除页眉页脚，才能应付连续页的情况

1. 组件——文本分块
2. 为什么要分块：模型输入的限制；减少噪音和保留语义相关性
3. 如何确定模型的最大输入长度：官方文档、开源模型则HF上的model\_cards或config.json查看
4. 分块的大小选择：



因此分块太大，容易导致较大的信息损失; 需要恰到好处，否则容易对合适的主题稀释了，导致本该更高分的部分，跟其他信息合在一起导致检索低分。分块大小需要调优，会影响问答准确性，需逐渐找到合适的大小，刚好满足但不宜太大

1. 常见方法

4.1 Charactertextsplitter: 可以指定分隔符和chunk\_size及重叠部分，允许突破分块大小，保证语义的完整性

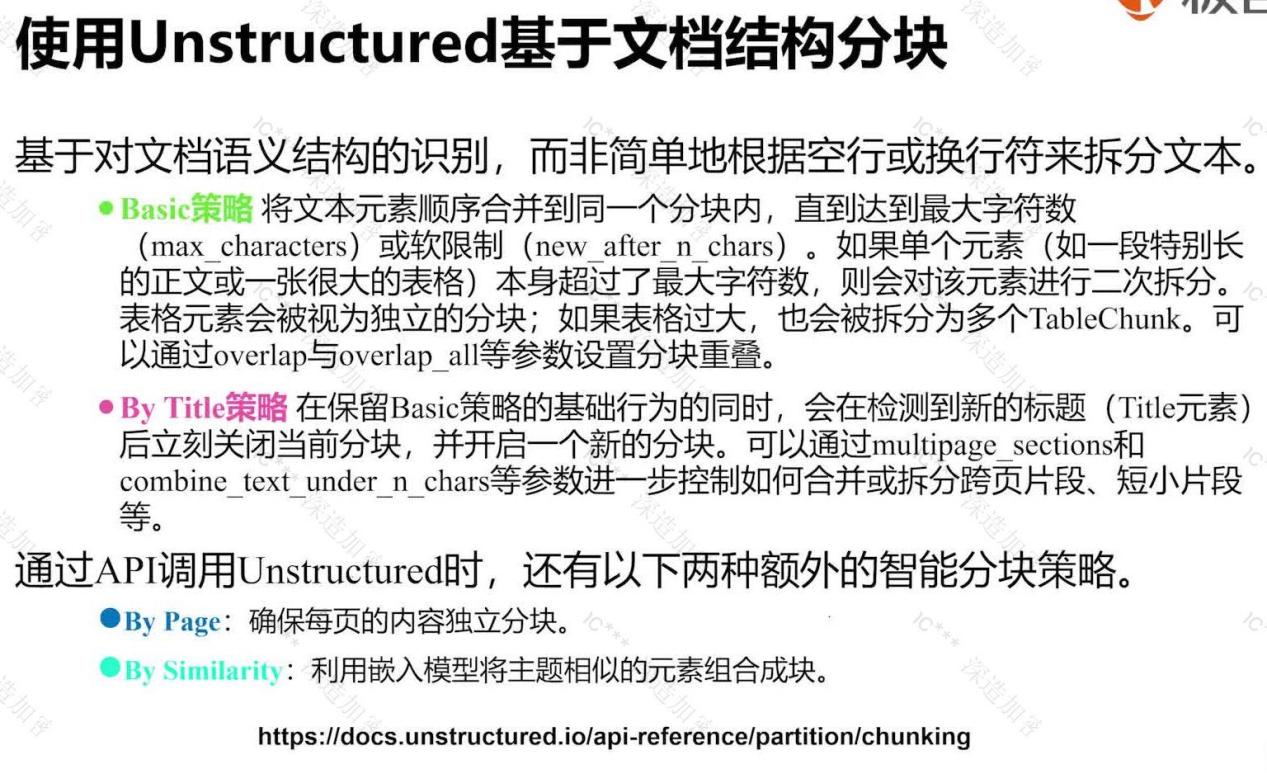
4.2 RecursiveharacterTextSplitter: 是递归分块，可指定若干个分隔符，依次按照能符合粪块大小的分隔符进行分割，比上述splitter更能满足chunk\_size，但不一定好用

4.3 代码分块: RecursiveCharacterTextSplitter.get\_separators\_for\_language(Language.PYTHON)

4.4 语义分块:

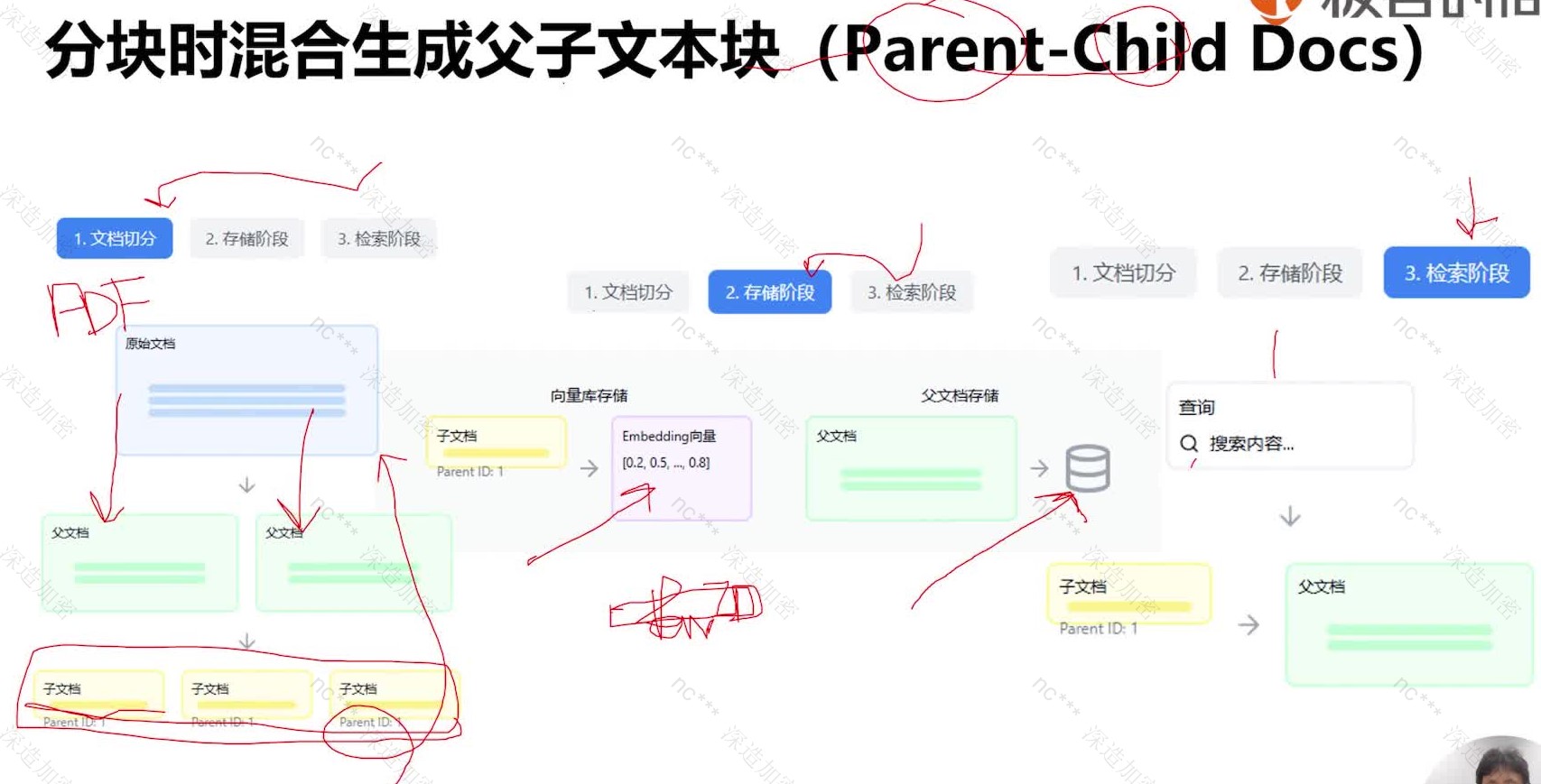
SemanticSplitterNodeParser: buffer\_size参数表明要用n个句子作为一组，各组比较向量相似度，达到某个阈值则放在一起，否则新建分块，阈值则是breakpoint\_percentile\_threshold，称为余弦不相似度，越小，越容易达到分割阈值，分块越多

4.5 unstructured对pdf分块:



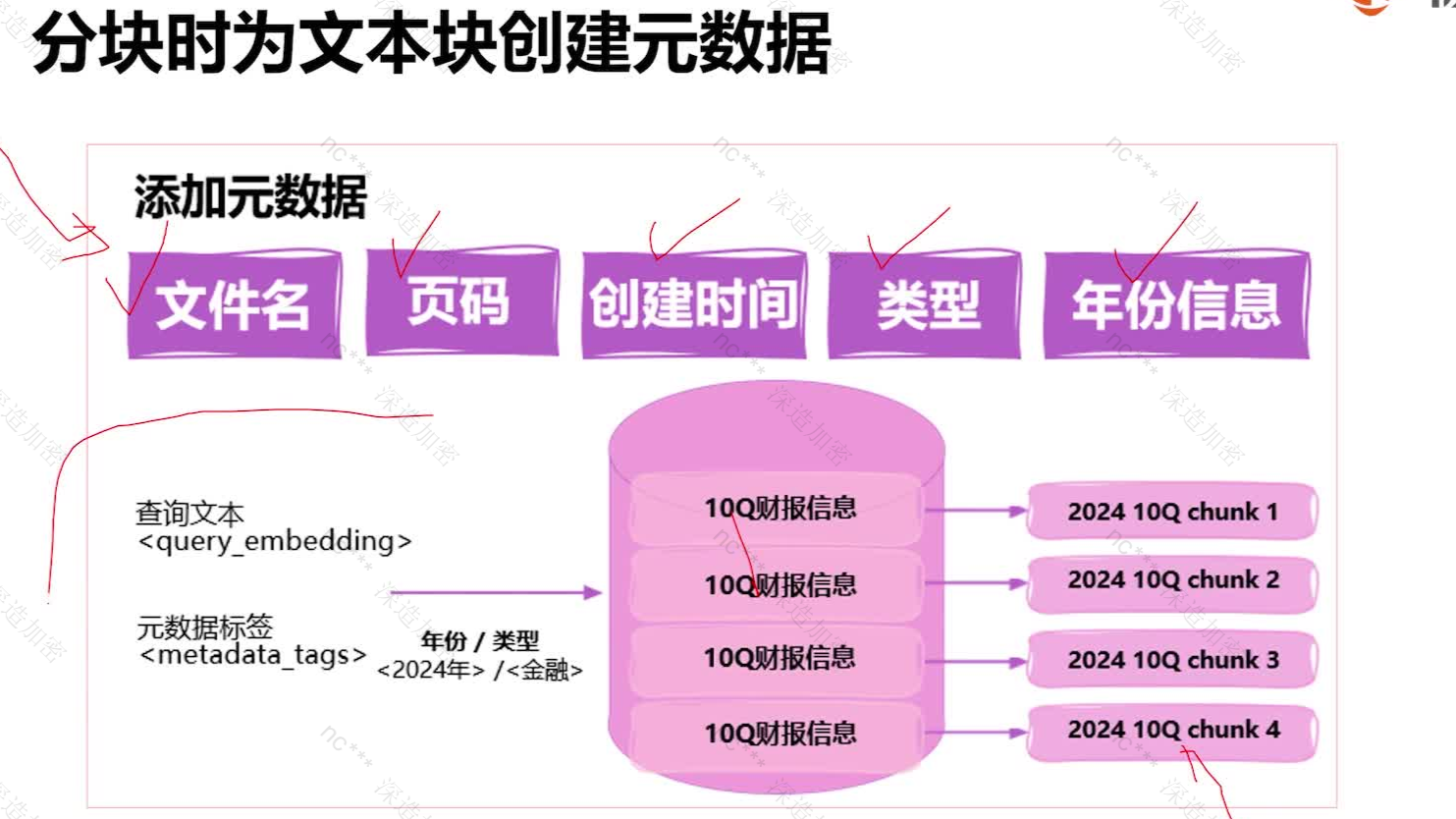
unstructured可以parition时直接chunk,也可以先partition再chunk; By similarity与上述语义分割类似

1. 与分块相关的高级索引技巧
   1. 带滑动窗口的句子切分: 切分时按句子切分, metadata使用句子的前后k个句子作为上下文, 则检索时仅按照单个句子检索, 但生成时使用了更多的上下文, 实现精细化检索&全面性生成的效果
   2. 分块时混合生成父子文本块:

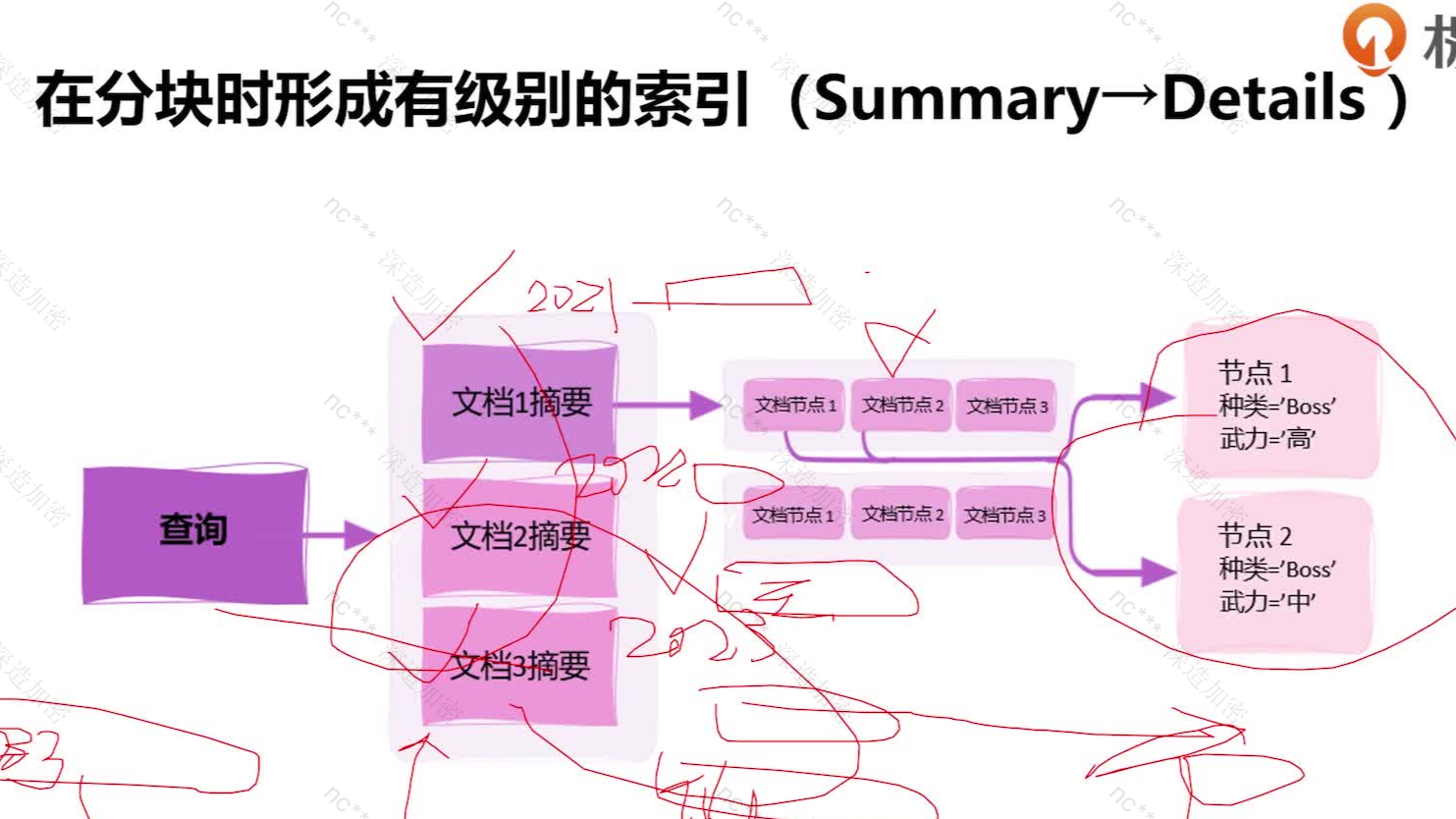


子文档记录父文档的ID, 检索出子文档后, 根据ID查找父文档, 父文档存在本地/关系数据库, 查出来作为检索结果给生成模型即可, 这种方法可以避免切块在embedding嵌入时超出token限制, 也能达到精细化检索&全面性生成的效果

* 1. 分块时为文本块创建元数据:

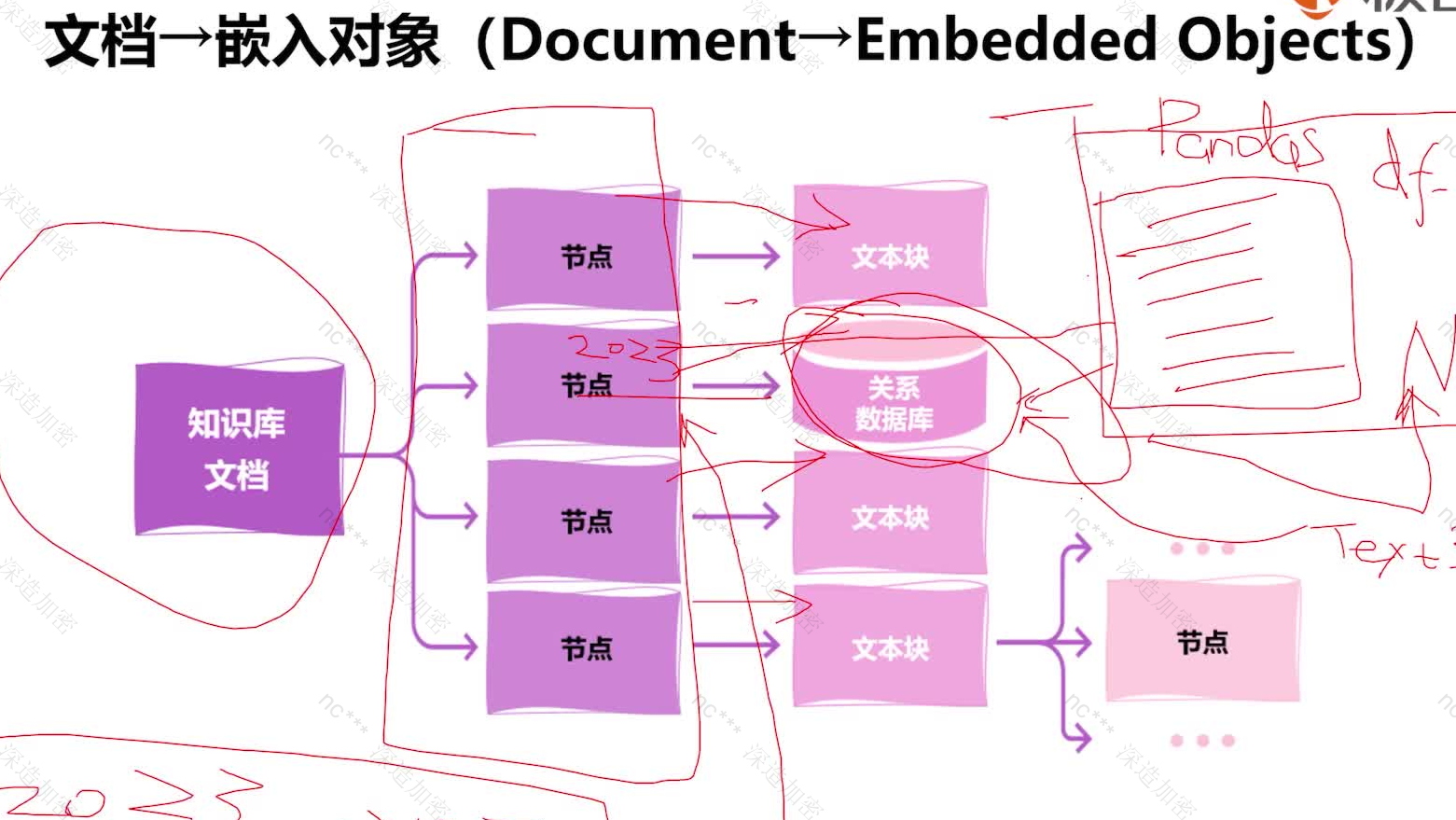


可在检索召回后使用元数据加以filter

* 1. 在分块时形成有级别的索引 

需要将文档进行概括性摘要, 根据摘要确定具体文档, 提升效果

* 1. 文档->嵌入对象



知识库中有若干节点, 如有的节点负责关系数据库查询, 则与之相关的查询会进入这里调用类似text\_to\_sql等流程进行具体数据的获取