语义增强可编程知识图谱 SPG

（Semantic-enhanced Programmable Graph）

白皮书（v1.0）

——新一代工业级知识语义框架及引擎

离散实体要素深度语义网络化，稀疏关系自动补全显性稠密化

加速企业海量数据知识化集成，无缝衔接 AI 技术框架应用落地

蚂蚁集团 × OpenKG

蚂蚁集团 × OpenKG 联合出品

2023 年 8 月

版权声明

本白皮书版权属于蚂蚁集团 × OpenKG，并受法律保护。转载、摘编或利用其他方式使用本白皮书文字或观点的，应注明“来源：蚂蚁集团 × OpenKG”。违反上述声明者，蚂蚁集团和OpenKG将追究其相关法律责任。

编写说明

牵头编写单位：蚂蚁科技集团股份有限公司

参与编写单位：同济大学、天津大学、恒生电子股份有限公司、浙江创邻科技有限公司、达观数据有限公司、海乂知信息科技（南京）有限公司、浙江大学、之江实验室、中国科学院计算技术研究所

编写组成员

蚂蚁科技集团股份有限公司：梁磊、张志强、彭晋、赵培龙、郭智慧、何雨潇、袁琳

同济大学：王昊奋

天津大学：王鑫、王翔

恒生电子股份有限公司：白硕、陈佼

浙江创邻科技有限公司：周研、张晨

达观数据有限公司：王文广、贺梦洁

海乂知信息科技（南京）有限公司：胡芳槐、丁军

浙江大学：陈华钧、张文

之江实验室：章衡

中国科学院计算技术研究所：白龙

蚂蚁集团 × OpenKG

推 荐 语

知识图谱是早期专家系统和语义网技术的延续，自 2012 年 Google 将其应用于搜索推荐领域以来，知识图谱技术在各领域得到了广泛应用。然而，长期以来知识图谱语义表示和技术框架并未有显著进步，这大大提高了各领域图谱的构建成本和业务落地的复杂度。我很高兴地了解到，蚂蚁集团和 OpenKG 合作，结合蚂蚁集团多年的知识图谱工业实践提出了兼容大数据体系和 AI 技术体系的知识语义框架 SPG。SPG 具有可编程性和框架化特性，具备较强的跨场景迁移能力，可以加速知识图谱的产业化落地，是知识图谱技术框架的突破性技术。自 2022 年底以来，ChatGPT、GPT4 等大模型掀起了人工智能的新浪潮，但当前大模型仍然存在知识幻觉性、复杂推理谬误和计算成本高等问题。符号化知识图谱的技术体系作为大模型的补充，可以实现可控的内容理解和内容生成，为大模型产业落地提供正确的领域知识和复杂推理能力的支持。期待 SPG 成为知识图谱领域的重要技术，结合蚂蚁集团多元化场景的持续打磨以及与 OpenKG 社区力量的共建，推动产业在知识图谱领域的发展，促进不同领域之间的知识互通互联，促进大模型和知识图谱技术可控低成本产业落地。

—— 清华大学人工智能研究院知识智能研究中心主任、教授 李涓子

知识图谱作为符号化的知识表示体系，具备高阶语义、结构严谨、复杂推理等能力。在大语言模型（LLM）飞速发展的时代，知识图谱与 LLM 之间有丰富的互动关系，一方面 LLM 为低成本构建大规模知识图谱提供了有力工具，能否借助 LLM 构建超出现有知识图谱规模 1-2 个数量级的世界知识图谱成为一个有趣的研究问题；另一方面知识图谱的高质量、可解释的知识表示和推理能力，也为解决 LLM 的空想问题提供了一种可能的探索方向。

传统知识语义框架，如 RDF/OWL 及 LPG 等在知识管理方面有显著不足，很难支撑 LLM 时代的知识图谱构建与应用。SPG 是蚂蚁知识图谱团队多年业务实践的总结，它有效克服了 RDF/OWL 及 LPG 的在知识管理上的不足，是一种新一代知识语义框架，借助 SPG 语义规范及可编程范式构建引擎架构，可以支持各领域图谱的高效构建和跨领域的知识语义对齐。

知识图谱的未来发展，离不开活跃的社区，未来蚂蚁将在 SPG 以及世界知识图谱构建与演化等方面持续与 OpenKG 社区合作，加速其技术成熟和产业落地。我们也欢迎产学各界同仁积极参与共建，共同促进知识图谱技术的成熟进步，促进不同领域之间的知识互通和流通，构建知识图谱+LLM 双驱动可控落地的新一代 AI 技术体系。

—— 蚂蚁集团技术研究院院长、副总裁 陈文光

蚂蚁集团拥有多元化的业务场景和海量的领域数据。SPG 框架是基于蚂蚁多年的知识图谱实践经验而打磨而成的。由于蚂蚁业务数据具有多源异构、时序动态和关联复杂等特点，这为大规模知识图谱构建提供了良好的孵化环境。SPG 框架通过对多业务、多场景问题的抽象总结，定义了新一代企业级知识管理范式，具备较强的企业级应用适应性。它通过数据的知识化，将海量数据转化为知识，并通过复杂模式计算和图学习推理等方法解决高维业务问题。SPG 框架为高效的领域图谱构建和跨领域图谱语义对齐提供了更多创新的可能性。此外，在大型模型时代，通过基于 SPG 构建的图谱框架和领域图谱，可以实现大型模型在安全风控、小微信贷、数字金融等业务领域的可控落地。通过与 OpenKG 的合作，我们希望通过社区和产业的力量加速推进 SPG 框架的完善，促进知识图谱技术的成熟，并推动产业的发展。在这个过程中，我们欢迎各位同仁积极参与共建，共同推动知识图谱技术的发展和创新，真正实现大模型与知识图谱双向驱动的可控 AIGC，从而加速产业的落地。

—— 蚂蚁集团机器智能部负责人、研究员 周俊

蚂蚁集团 x OpenKG

序 言

知识图谱作为一种建模和管理数据的方法，已经在企业数字化过程中发挥了重要作用。然而，随着企业对知识图谱的需求不断增加，传统的知识图谱技术面临着一些挑战。基于对当前知识图谱技术的深入研究和实践经验的总结，蚂蚁集团发现，传统的知识图谱技术在应对复杂的业务场景和大规模数据时存在一些局限性。例如，知识图谱的构建需要统一的工业级知识建模框架，以便适应不同领域的需求；知识图谱的推理能力需要更加高效和可解释；知识图谱的构建和推理过程需要更好的可编程性和跨场景迁移性。

作为蚂蚁集团知识引擎的负责人，梁磊带领团队研制了一个工业级知识图谱语义框架——SPG (Semantic-enhanced Programmable Graph)。当他第一次向我介绍蚂蚁的思考和 SPG 时，我惊喜地发现大家不约而同地在解决类似的问题，原来约定的 1 小时会议也慢慢演变成了一个上午的深度交流。之后我愈发感觉我们整合力量去扩展 SPG 来应对大模型时代新的机遇和需求，并向整个社区开源这个一站式全新的知识图谱平台工具。当我将这个想法告诉了梁磊，他和蚂蚁集团非常支持，我们也积极推进 OpenKG 的各个研发力量和蚂蚁知识图谱团队的合作，最终形成了一个虚拟团队开展了后续的双周交流，设计规划和研发工作。

SPG 框架以属性图为基础，融合了 RDF/OWL 的语义性和 LPG 的结构性，兼具语义简洁和大数据兼容的优势。通过 SPG 框架，我们可以实现知识的动态到静态自动分层、领域内知识的唯一性和知识之间的依赖关系定义。同时，SPG 框架还提供了可编程的范式，支持快速构建新的领域图谱和跨场景迁移。其在解决典型问题和场景方面具有广泛的应用价值。在黑产图谱和产业链事理图谱中，SPG 框架可以帮助企业更好地识别和应对黑灰产对抗，提高风险防控能力；在知识推理和智能问答中，SPG 框架可以提供更加准确和可解释的推理结果，提升用户体验和决策效果。

在本白皮书中，我们将详细介绍 SPG 框架的设计原理、技术模块和应用案例。我们希望通过这份白皮书，能够为读者提供一个全面了解 SPG 框架的机会，并激发更多的讨论和合作。我们相信，SPG 框架将为企业数字化提供更加强大和灵活的知识图谱技术支持，推动知识图谱技术的发展和应用。最后，我们要感谢您对本白皮书的关注和支持。如果您对 SPG 框架或知识图谱技术有任何问题或建议，欢迎随时与我们联系。让我们一起开创新一代工业级知识图谱的未来！

谢谢！

——王昊奋、梁磊和 SPG 团队

蚂蚁集团 x OpenKG

目 录

蚂蚁集团 x OpenKG

蚂蚁集团 x OpenKG

第 1 章 从数据化到知识化：企业深化竞争优势，图谱技术与时俱进

在企业的数字化过程中，积累了海量的数据，既包括文本、图像、视频、音频等非/半结构化数据，又包括用户行为、商品订单、产品服务、商户画像等结构化数据，还包括为支撑业务发展采买的专业知识库、外部渠道获取的行业数据等。面对海量数据，企业需要不断地为用户创造价值，同时实现高效的经营管理和风险控制。这对企业的数字化基建提出了很高的要求，也为知识图谱 (Knowledge Graph, KG)、大语言模型 (Large Language Model, LLM)等 AI 技术提供了多样化的落地场景，也带来了新的机遇和挑战，AI 技术可以帮助企业从海量数据中快速发现规律、分析趋势、预测未来，从而更加精准地了解客户需求、优化产品设计、提升生产效率，还可以帮助企业进行智能风险管理、反欺诈识别等。而企业内因业务发展、部门差异等又广泛存在数据孤岛、数据一致性冲突、数据重复等问题，为提升数据利用效率，需要加强数据管理和应用，提高数据的利用率和价值。面对海量数据，企业需建立应用友好的管理范式，按业务模型定义数据结构，明确语义、消除歧义、发现错误等；面对数据孤岛，企业也期望建立数据孤岛的连接机制，实现跨系统、跨部门的数据共享和协同利用；面对口径差异，企业需建立标准化的数据和服务协议，以实现高效的数据协同、专家经验协同、人机协同等。通过更高效的数据管理机制，标准化数据建模、消除歧义提升一致性、连接数据孤岛，是企业数字化升级面临的关键问题，更高效的组织管理企业数据，利用 AI 技术充分挖掘数据价值，已成为企业未来增长的核心内驱力。

1.1 知识图谱作为新一代企业级知识管理范式的期待

作为 AI 技术重要分支的知识图谱因可以帮助企业更好地组织和管理知识数据，通过对数据进行语义化建模，构建知识图谱，企业可以更加直观地了解数据之间的关系，从而更好地发现隐藏在数据中的价值，也受到了越来越多的青睐。Gartner 2021 年预测以知识图谱技术为基础的 Data Fabric 是下一代数据架构，Neo4j, Cambridge Semantic 也分别发布白皮书介绍基于知识图谱的新一代知识管理范式，Neo4j 认为知识图谱是语义增强的图，通过一定范式对图进行语义增强以帮助企业从多维度深度关联中发现更多隐式线索。Cambridge Semantic 认为知识图谱是 Data Fabric 的杀手级应用，知识图谱对真实世界的实体、事实、概念以及它们之间的关系建模，提供面向不同角色一致的建模能力，能更精确的表示组织数据，它通过强 Schema 驱动可有效连接数据源和图存储及下游 AI/BI 任务，连接数据孤岛，按需集成、按需加载、无缝衔接。自 2018 年以来，企业数字化垂直领域的图谱应用越来越广泛，如金融、医疗、公安和能源等领域[1,2,3]。一份报告[4]显示，到 2026 年，中国图谱市场空间将达到 290 亿元，其中金融和公安是主要的拉动力量。企业数字化中的知识图谱应用，以商家图谱商户风险防控为例，因对中小商户、新用户、沉睡户等薄数据客群的

画像覆盖和风险洞察[1]的需要而对知识的深度上下文（即 Deep Context）有更多感知要求，企业级知识管理正在实现从二元静态向多元动态的模式跃迁。

1.2 从二元静态到多元动态：知识管理模式的跃迁

知识图谱是一种建模和管理数据的方法，它利用图结构、知识语义和逻辑依赖，提供存储、推理和查询事实知识的能力。早期的应用主要是从公开语料中提取百科类<s, p, o>三元组来构建静态知识图谱，以提高搜索推荐的效率和体验。随着知识图谱应用从搜索推荐的 C 应用转向风险防控/经营管理的企业级 B/C 联动的领域应用，因前文所述长尾稀薄客群画像覆盖和风险洞察的需求，领域图谱需要具备全面性、正确性和可解释性等特点，图谱数据的来源也从文本语料转向了企业多源异构数据。这些数据包括非/半结构化的 UGC/PGC 内容、业务经营沉淀的结构化基础画像、交易事务、日志记录等，以及各领域特有的业务专家经验。围绕增长经营和风险防控，构建完整的客户、物料、渠道等的立体画像，以商家为例，图 1 展示了构建过程的示意图。

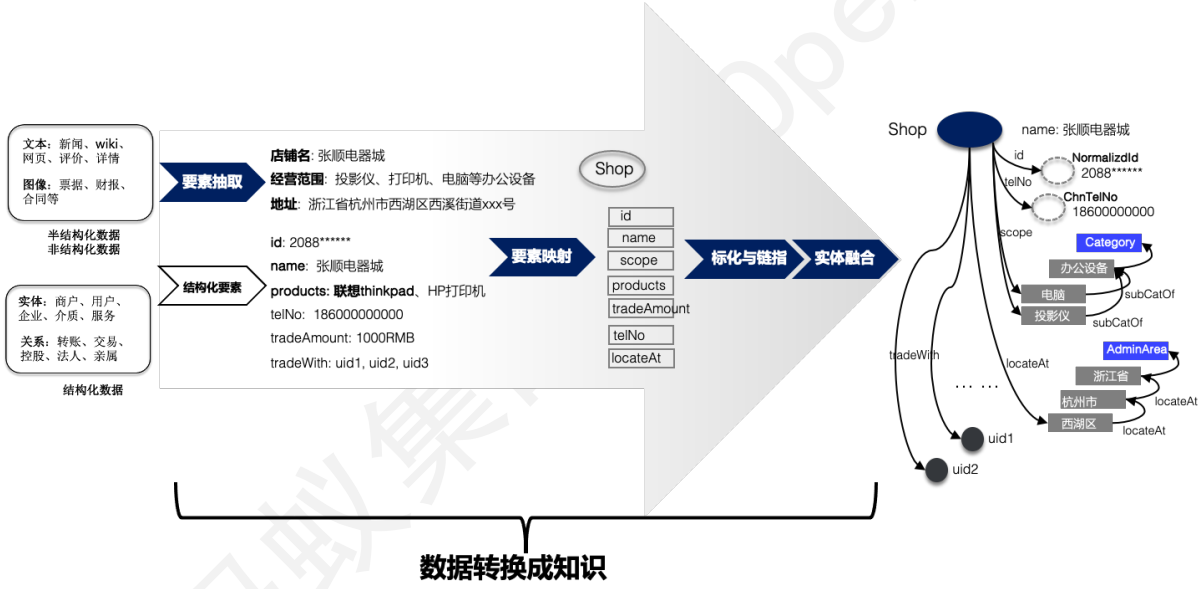

图 1 商家实体构建的过程

当前，商家已经突破了静态门店的限制，收款码使得任何人都可以成为商家，但是这也增加了风险防控的难度。仅仅通过文本概念标签来进行风险防控是没有意义的，添加交易、社交等实际事实关系也远远不够。如图 2 所示，需要实体多要素的深度信息协同才能发现更多有效的关联。图谱构建的要求也从静态常识转向 Deep Context 动态时空。这既需要基于介质（如 WIFI、电话、Email 等）来实现关系传导，又需要对地理连续空间 (Spatial）实现边界化的聚集关联[5,6]，还需要跟踪中/宏/微观事件的多元传导脉络，实现实体间稀疏关系语义可解释的稠密化。

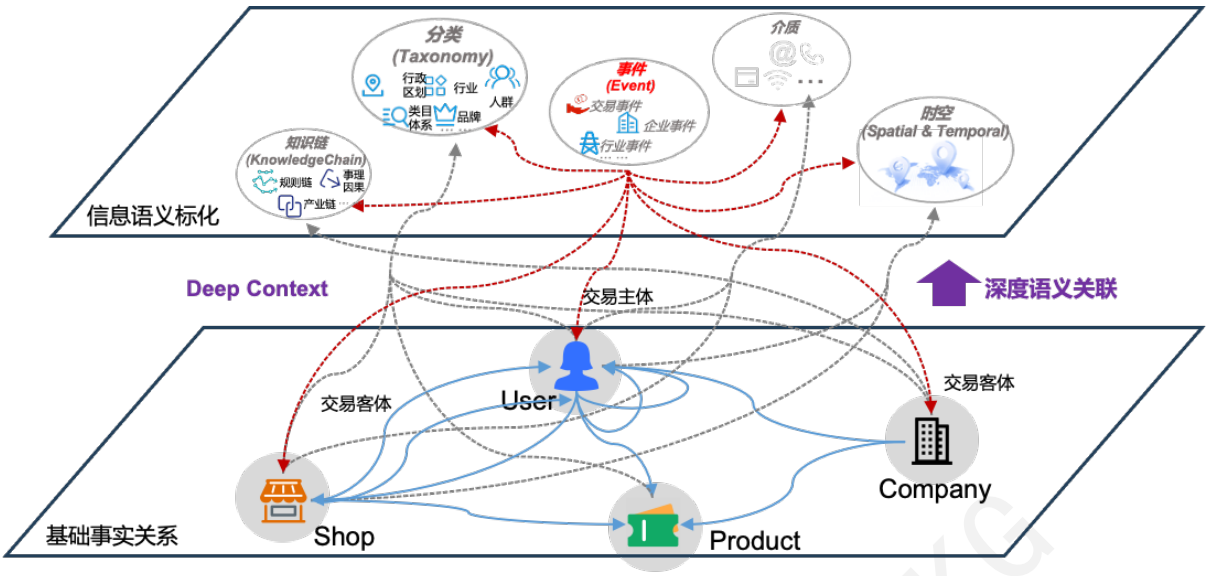

图 2 Deep Context 语义扩展的基础事实图谱

在业务应用方面，知识图谱可以用于构建知识推理任务，例如：1) 商品推荐：通过类目、意图、时空等语义连接人-商品、人-商户、商品-渠道等，实现语义联想的商品召回和表征迁移；2) eKYB (Electronic Know Your Business)：通过介质关联、行为事件和时空聚集，识别商户同人、同店等，实现有效的画像补全和风险洞察。此外，基于知识图谱还可以实现结构感知的可控文本生成[7]，例如：1) 反洗钱智能审理识别定性和报文生成：结合 Deep Context 预测风险行为、挖掘团伙，通过资金链、时空聚集、设备关联等还原团伙/异常结构，并通过知识图谱到文本的转换输出可解释报文；2) AI 电话唤醒受害者：将识别到的可疑设备、钓鱼域名/AppID、团伙等事实关联传导到交易用户，生成沟通话术提醒用户并拦截风险。这些应用旨在实现更加智能化和精准化的风险控制和业务推理，提高商业运营的效率和价值。

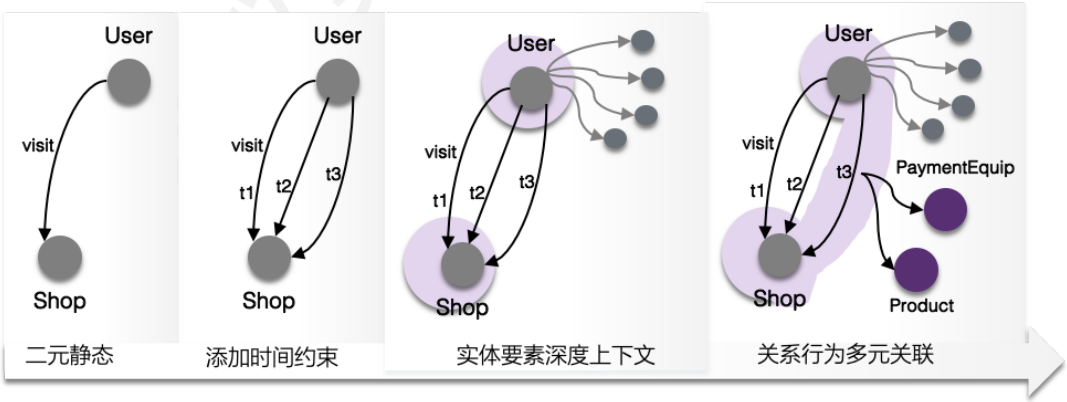

图 3 知识表示从二元到多元的演进

在商户经营与风险防控的案例中，知识管理需要具备较强的上下文感知能力。常见的常识知识图谱由于卸掉了可感知上下文的信息和时空关联，在实际应用中，若论元要素出现了多元化或相互交织，由于无法感知个体差异，仅使用概念层归纳，推理应用的效果会大打折扣[8]。类似的问题也出现在公安反诈、保险理赔、医疗问诊、企业授信等领域中。因此，企业垂直领域对知识

图谱的期望发生了较大变化。知识表示也从图 3 所示的二元静态结构发展到时空多元动态关联，以更好地适应实际应用的要求。

1.3 与领域知识结合为 AI 可控、可靠落地提供了新思路

中国工程院院士潘云鹤认为，在 AI 走向 2.0 的过程中，数据和知识是两个最重要的关键元素。处理大数据和多重知识，形成了 AI 发展的两类核心技术，知识可以有效助力人工智能认知、决策和学习。

在数字化转型的过程中，通过对海量数据的抽提或业务经验的积累，沉淀大量领域知识，比如事实知识、专家经验、操作流程等，这些知识存在于各个行业，也难以公开获取，蕴含着巨大的价值，将行业专家知识与 AI 有效结合可解决 AI 应用过程中可控、安全、可解释等问题。2022 年底，ChatGPT 火爆全球，随后国内也掀起百“模”大战。然而，由于 LLM 是一种黑箱概率模型[9]，难以捕获和获取事实知识，因此存在较多幻觉和逻辑错误[10]。与此同时，知识图谱的事实性、时效性和逻辑严谨性成为了 LLM 的绝佳能力补充。通过将知识图谱作为约束和复杂推理能力的来源，LLM+KG 的应用范式引起了广泛关注，并催生了许多应用探索和研究[9,10,11]。

表 1 LLM 和 KG 在企业数字化不同场景下的应用

场景\能力		仅用大模型	知识图谱增强大模型	大模型增强知识图谱	仅用知识图谱
经营增长	交互应用	闲聊	知识问答、服务检索、报告分析等	-	营销推荐、事件脉络、营销决策等
	经营管理	-	报表查询、人群圈选、智能文案等	-	事件分析、物料分析、人群分析等
风险防控	风险防控	-	可解释报文生成、唤醒机器人等	-	线索追踪、事件传导、规则核赔、企业授信、最终受益人、股权穿透等
知识生产	图谱构建	-	-	文档要素抽取、事件抽取、实体链指等	基于结构化经营数据的图谱构建
	知识挖掘/补全	-	-	获取实体LLM embedding表征、从LLM中萃取知识补充缺失知识	关系挖掘、属性预测、团伙挖掘、规则挖掘等

在各种应用场景中，以商户经营与风控为例，算法任务可以分为以下五个方面：1) 交互应用：包括消费端 (C) 产品上的商品/服务透出和供应端 (B) 产品上的服务/商家入驻等；2) 经营管理：企业经营、商户经营所必须的经营分析、物料管理等；3) 风险防控：黑灰产对抗是企业经营永恒的话题，企业必须增强对薄数据客群的认知覆盖和对新风险模式的快速识别；4) 知识构建：将外部非/半结构化、结构化数据转换成领域知识；5) 知识挖掘：企业促增长和控风险，不断提升主体要素、跨主体关系的长尾覆盖。表格 1 中列举了不同分类下 LLM、KG 及 LLM 与 KG 相互增强可能的落地应用。这些应用可以帮助企业在商户经营和风控领域中获得更好的效果和成果。

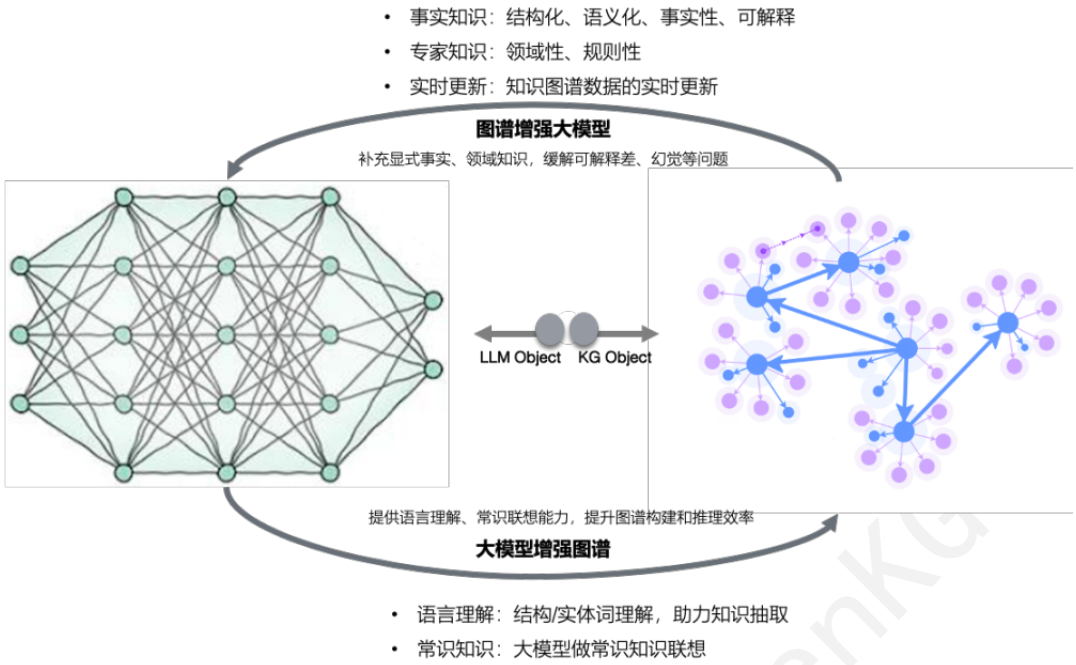

图 4 大模型与知识图谱的相互驱动

总体而言，以商户经营与风控应用场景为例，LLM 和 KG 应用的算法任务主要可以分为三类：1) LLM only：由于领域专业性和事实性的要求，LLM 在商户经营与风控领域尚未有明确可落地的场景；2) LLM + KG 双驱动：主要体现在知识问答、报告生成等用户交互类场景中，比如前文提到的 AI 电话唤醒受害者和反洗钱智能审理报文生成等。此外，还有知识要素抽取、实体链指等知识构建类场景。文献[10]中详细描述了 LLM 与 KG 的双驱动，包括 KG 增强的 LLM、LLM 增强的 KG 以及 LLM+KG 框架协同三个方面，如图 4 所示；3) KG only：在推理决策、分析查询、知识挖掘类等不需要复杂语言交互和意图理解的决策/挖掘场景中，基于图谱结构化知识直接做图表征学习、规则推理、知识查询等。通过框架的协同实现 LLM 与 KG 双驱动，支持跨模态知识对齐、逻辑引导知识推理、自然语言知识查询等。这对 KG 知识语义的统一表示和引擎框架的跨场景迁移提出了更高的要求。

1.4 知识图谱技术体系的发展需与时俱进

知识图谱自身技术框架的发展和对其在新知识数据管理范式、大模型的双轮驱动的期待并不完全匹配，图谱技术的发展也需要与时俱进。具体而言，存在以下问题：首先，缺乏工业级统一的知识建模框架。尽管资源描述框架 (Resource Description Framework, RDF)/Web 本体语言 (Web Ontology Language, OWL）这种强语义、弱结构的技术框架已经发展多年，但并未出现成功的企业级/商业化应用。相反，强结构、弱语义的属性图 (Labeled Property Graph, LPG）成为了企业级应用的首选。其次，缺乏统一的技术框架[2]，导致跨领域迁移性较差，由于工具繁多、链路复杂，每个领域的图谱构建都需要从零开始。除了以上两点，其他方面也存在较大的技术挑战，例如表 2 所列。

表 2 新范式下图谱面临的技术挑战

分类	子项	描述
1、总体框架	1.1 工业易用的知识语义框架	衔接大数据与AI技术体系，支持事实与逻辑融合表示的知识语义框架
	1.2 可跨场景迁移的图谱引擎	图谱引擎具备跨场景迁移性，支持快速孵化新的领域图谱
2、图谱构建	2.1 统一知识抽取框架	基于非/半结构化数据的统一知识抽取， 保障大规模数据下的吞吐和性能
	2.2 统一实体链指框架	知识要素的统一链指/标化框架， 保障大规模数据下的吞吐、性能和一致性
3、图谱推理	3.1 专家规则知识表示	将业务系统中端到端的决策规则，构建成知识之间逻辑依赖的分层表示
	3.2 规则引导可解释推理	实现有效的规则注入、规则约束，输出可解释的推理结果[7]

知识图谱的目标是构建一个机器可理解、可推理的数字世界，实现知识语义的统一表示和框架化能力分层，以支持不同领域图谱的快速构建和跨场景迁移。这是图谱产业化加速过程中必须解决的基本核心问题。

1.5 基于 SPG 的工业级知识图谱引擎

蚂蚁知识图谱平台经过多年金融领域业务的支撑，沉淀了基于属性图的语义框架——语义增强可编程框架 (Semantic-enhanced Programmable Graph, SPG)。它创造性地融合了 LPG 结构性与 RDF 语义性，既克服了 RDF/OWL 语义复杂无法工业落地的问题，又充分继承了 LPG 结构简单与大数据体系兼容的优势。

首先，SPG 明确定义了数字世界知识的概念。知识是人类对物质世界和精神世界探索结果的总和，数字世界的机器对知识的认知该如何定义？SPG 通过形式化描述和客观事实两个视角，明确了数字世界知识的定义，结合图 5 的说明，从三个维度对形式化表示进行了定义，分别是：

1) **领域类型结构约束。**在客观世界中，任何事物（Thing）都属于至少有一个类型（Class），数字世界也是如此。基于 SPG 的领域结构类型约束（SPG Domain Model Constrained， SPG DC），可帮业务实现知识的主体分类和由动态时空到静态常识的自动分层。

2) 领域内实例唯一性。在客观世界中，不存在完全相同的两个事物，数字世界也当如此。然而，由于数字世界存在多源异构和数据拷贝等问题，导致大量数据存在冗余和重复。SPG Evolving 利用 SPG Programming 知识生产 SDK 框架提供的实体链指、概念标化和实体归一等算子能力，结合自然语言处理(Natural Language Processing, NLP)和深度学习算法，提高单个类型（Class）中不同实例（Instance）的唯一性水平，支持领域图谱的持续迭代演化。

3) **知识间逻辑依赖性。**在客观世界中，任何事物都存在着和其他事物由此及彼的关联，不存在不与其他事物关联的事物，数字世界也不例外。SPG Reasoning 利用谓词语义和逻辑规则来定义知识之间的依赖和传递，并提供可编程的符号化表示，以方便机器理解。

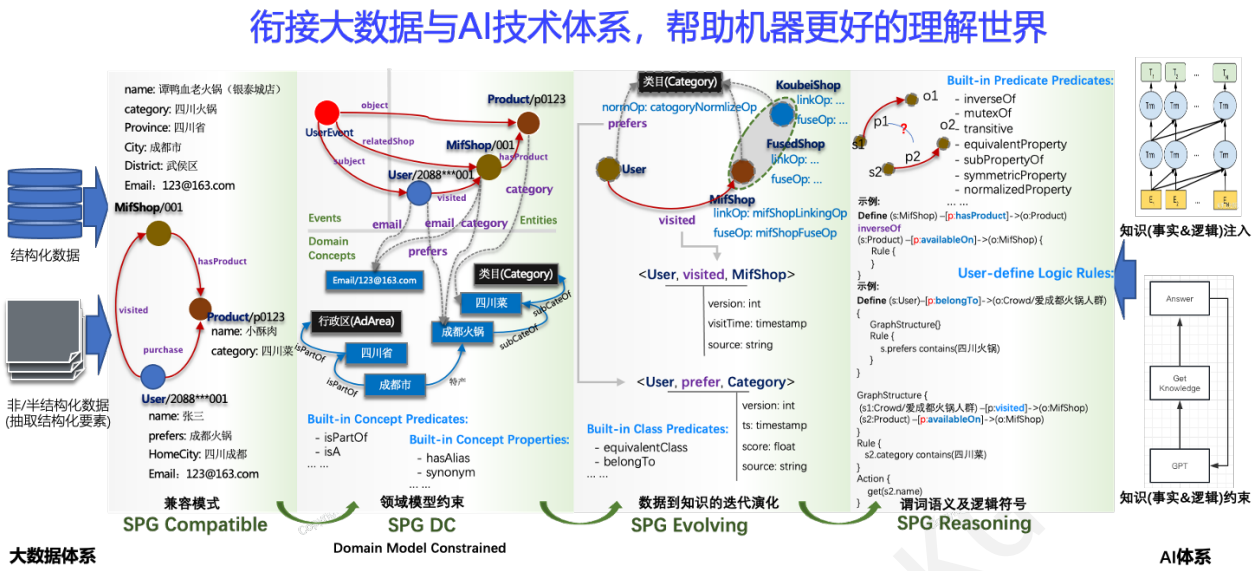

图 5 SPG 知识语义框架

然后，SPG 充分融合了 LPG 的优势，并通过 SPG Compatible 兼容大数据体系。基于 SPG 构建的知识引擎在图谱构建阶段与大数据架构兼容衔接，提供了知识构建算子框架，实现从数据到知识的转换。在存储阶段，可适配属性图以充分利用其存储和计算能力。在推理应用阶段，形式化成了 KGDSL（Knowledge Graph Domain Specific Language），机器可理解的符号表示支持下游规则推理、神经/符号融合学习、KG2Prompt 联动 LLM 知识抽取/知识推理等。同时，通过架构的分层，新的领域图谱构建只需定义 Schema、准备数据、开发生产/推理 Operator 即可。

知识图谱技术依然处于快速发展时期，也处于关键的技术拐点期，统一的技术框架能大幅降应用门槛促进生态的繁荣。为此，本白皮书也重点从企业级知识管理这个最根本的问题出发，推导知识管理、图谱构建与推理应用的全生命周期，以期实现工业级可迁移的知识表示与引擎框架。如前文所述，LPG 属性图因其兼容大数据架构的独特优势成为绝大多数企业知识建模的首选，本白皮书也是从属性图知识管理的实际业务问题出发，推导企业级知识管理所必须的语义能力。

第 2 章 基于属性图的知识管理存在的问题

在企业级知识图谱应用中，如第 1 章所述，属性图因其高效和对大数据体系的兼容性，使其成为领域图谱建模的首选，以实现快速落地业务价值。虽然基于属性图的图谱构建前期成本较低，但随着业务快速发展和知识体量的大幅增加，因其知识语义及管理能力的缺失，属性图的种种弊端会逐步显露。首先，知识模型的变更演化变得越来越困难，Schema 变得越来越复杂。其次，由于点/边模型的灵活性，带来了大量冗余的类型创建和重复的数据准备，导致不同关系/属性之间逻辑的一致性和合理性也越来越难以维持。第三，朴素的属性/关系模型难以刻画事物(Thing)的内在语义和事物之间的语义依赖。这给图谱业务项目的持续迭代升级带来了较大的障碍。当规模膨胀到难以为继时，不得不新创建项目重新构建 Schema 和图谱数据；业务应用阶段也不得不添加大量硬编码，实现业务语义的解析和对齐。本章节将结合黑产风控和企业事理图谱两个业务案例，介绍业务应用的背景和主要痛点问题，并对相关问题进行归类总结。接下来，我们将在第 3/4/5/6/7 章尝试提出解决方案，最终在第 9 章中提供两个案例基于 SPG 的完整方案，以期在应用属性图的优势的同时，规避其弊端，为企业级图谱应用提供高效的语义建模和知识管理工具。

2.1 典型案例 1：黑产知识图谱

为了实现黑产图谱的主要业务目标，通过构建用户相关的风险画像及设备、介质、交易等相关的关联网络，并根据显式或隐式关联挖掘出黑产涉案人员并进行风险管控措施。以 App 网络风险防控为例，某 App 被发现涉嫌风险应用（赌博、色情、欺诈等），期望可以通过该风险 App 的关联网络实现以下两个目标：1）挖掘背后的风险人员，根据挖掘线索进行对应的风控策略。2）挖掘其他未被发现风险的 App，阻断风险的蔓延扩大。

然而，在实践中，黑产涉案人员通常会伪装或隐匿其行为，例如使用大量虚拟设备、虚拟 IP、虚拟身份等，这些行为会被掩藏在正常用户中。因此，本章节将以表 3、表 4、表 5、表 6 列举的部分数据举例说明，提炼当前属性图知识管理所遇到的问题。其中**娱乐为被举报的赌博应用，王武应当被判定为赌博应用开发者；李四应当为赌博公司 B 的老板；张三为李四的同人用户。

表 3 黑产图谱用户实体基本信息

用户id	用户名	用户电话号码	持有设备mac列表	持有证书	用户类型
1	王武	154xxxx3456	06:8A:5F:2E:AB:85 06:8A:5F:2E:AB:86	证书1	自然人
2	李四	135xxxx5532	06:8A:5F:2E:A1:85		自然人
3	张三	135xxxx5532	06:8A:5F:2E:A1:85		自然人
4	公司B	131xxxx3456		证书2	公司
...	...	...	...	...	...

表 4 黑产图谱持股关系基本信息

持股用户名	被持股公司名	持股份额比例
张三	公司A	100%
公司A	公司B	100%
...	...	...

表 5 黑产图谱应用实体基本信息

应用id	应用名	应用被安装的设备mac列表	持有证书	是否为赌博应用(基于用户投诉标注)
1	**娱乐	06:8A:5F:2F:AB:85, 06:8A:5F:2E:AB:86	证书1	是
2	捕*达人	06:8A:5F:2E:AB:85	证书2	未知
...	...	...	...	

表 6 黑产图谱转账关系

转出用户	转入用户	转账金额
李四	王武	10000
...	...	....

数据层面的直接表达和业务期望之间存在较大的差距，具体表现为：

- 不同主体之间的深层次关联难以体现：无法从数据的构造中直接得到应用和用户、应用的关联关系。
- 同一主体不同刻画维度的对齐：自然人和用户不能直接等价，例如本例中张三和被标记为赌博老板两个用户属于一人。

在业务实践中，虽然应用和用户、应用和应用之间不存在直接的关联关系，但往往可以通过一些设备、证书之类中间介质间接关联；同样用户和用户之间也可通过同手机、同设备等方法发掘同人。业务人员为了应对如上复杂的网络关系，图谱一般会如下演进：

第一步：将表数据转换成为属性图表示。

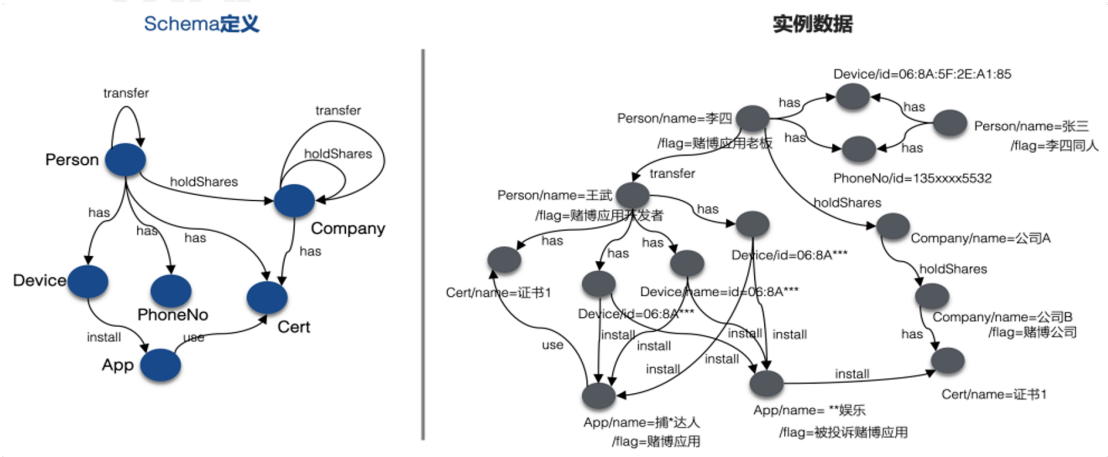

图 6 基于数据表直接转换成属性图构建图谱

图 6 将表数据映射到图谱数据结构，此时已经能够根据多跳关系得到风险应用和风险人员的关系，但仍需要业务专家进行分析研判，无法直接得到业务目标，业务目标应当如图 7 所示。图 6/图 7 中实体实例的文字结构为：类型/实例属性名=属性值。

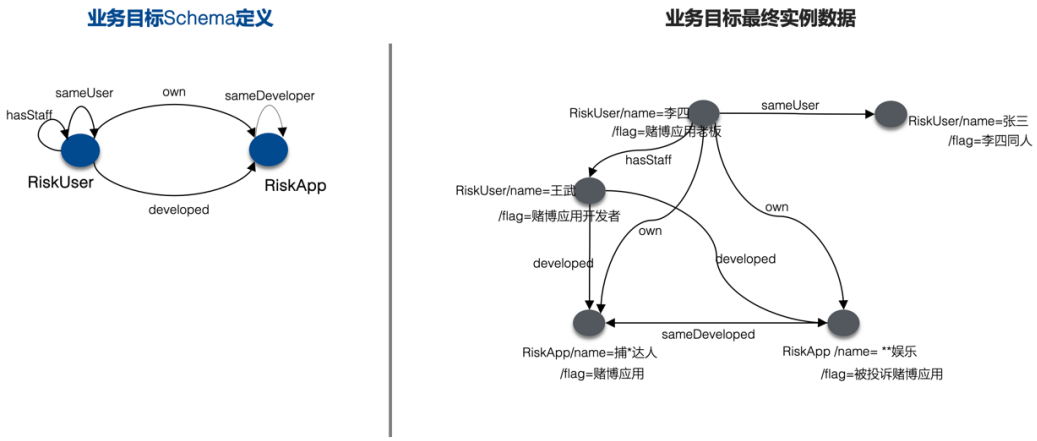

图 7 业务期望通过隐式推导得到的图谱结构

业务所需的图谱数据结构通常与原始图数据不同。原始图数据是客观的基础数据，而业务所需的数据是基于客观数据挖掘出的关联关系，也需要重新融合到原始数据中。为了挖掘这些隐式关联，业务专家制定了一系列规则，例如同用户的判定规则、用户对应用的拥有规则和应用开发者关系规则等，如果两个用户使用了相同的手机号或设备，则认为他们之间存在同手机或同设备关系；如果一个用户对一个法人存在控股关系，则该法人发布的 App 实际拥有者为该用户；如果一个用户持有多个设备均安装了同一个 App，那么该用户为该 App 的开发者。通过这些规则，基于外挂大数据系统完成规则计算，新增类型、新增关系得到业务所需的图谱数据结构，同时保留原始基础信息的定义，以支持业务更好的决策和风险控制。

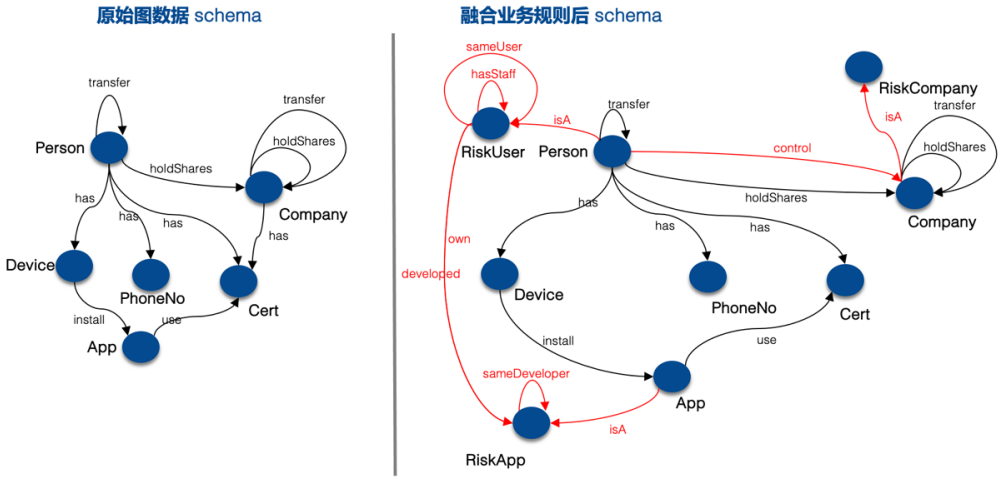

图 8 黑产图谱中融合业务规则后 Schema 差异

如上示例展示了业务决策过程中冗余创建的一部分。在知识图谱管理中，从基础事实中挖掘复杂的隐式关联是基本需求。然而，我们需要解决如何避免由于业务目标细化而导致 Schema 持续膨胀的问题，并确保规则计算与基础事实的逻辑一致性。这些都是知识管理必须解决的基本问题。

2.2 属性图应用于黑产图谱所存在的问题

- **点边独立数据准备造成图谱构造成本的大幅增加。**为构造黑产图谱所需实体、关系，因点、边数据独立准备的要求，需提供远远大于原始 4 张表的数据。
- **不同图谱间难以直接复用造成的重复数据准备。**在本业务中，需要构造资金转账和股权结构的图谱数据。通常情况下，这些数据已经作为基础数据存在于其他图谱中。
- **实体及要素之间存在逻辑依赖带来的不一致问题。**在业务建模的图中，图 7 和图 8 的新增类型、关系均是从图 6 的已有数据衍生产出。当基础数据发生变更时，此类衍生的数据必须同步变更，否则必然会出现图谱数据不一致问题。
- **业务目标的迁移变化导致图谱结构的持续膨胀。**在本案例中，通过介质隐式关联的方式挖掘应用背后的涉黑用户。但是黑产对抗更新快，必然会频繁更新、创建不同的实体、关系类型。图谱 Schema 及实例的规模会持续的膨胀扩展，最终变得难以管理。

因此，在构建图谱时，我们需要考虑这些问题并采取相应的措施加以解决，优化数据转换过程、提高图谱数据的复用性，在设计 Schema 时支持知识之间的逻辑关联表达提升业务语义迁移的表达效率。帮助我们构建更加高效、可靠和易维护的图谱系统。

2.3 典型案例 2：金融事理图谱

事理图谱的知识管理过程更注重对事件之间顺承关系、因果关系、条件关系和上下位等事理逻辑的刻画，因此事理图谱的基础是事件，实践应用中一般是从事件与图谱的应用逐步发展到事理图谱：捕获企业相关的生产、经营事件，提取事件的关键要素，实现事件要素与内部企业/产业链图谱之间的联动，构建风险事件与企业/产业链图谱之间联动的事理逻辑链，捕获到外部风险事件后能快速联动内部预警或风险处置。

当一个金融领域的事件发生后，我们需要基于基础事实对事件进行推理，来尝试得到以下问题的答案：

- 事件自身性质及影响程度
- 涉事主体有哪些？对其周边关联实体产生何种影响
- 关联主体是否会进一步衍生其他影响，如何影响

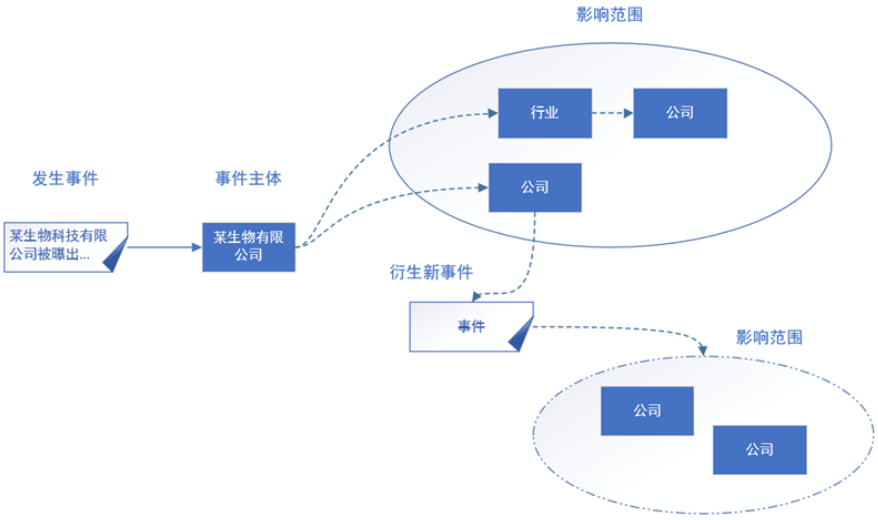

图 9 事件影响传导示意图

例如，某生物科技有限公司被曝出生产的化肥重金属严重超标的事件，其本身的利害程度需要进行进一步的影响分析。针对具体事件进行分析时，分析师需要基于对事件的理解，在基础事实知识上反复查询洞察，并结合常识性知识得出事件影响结论。然而，各种推理逻辑及数据往往呈碎片化分散在各处，需要有效整合连接。因此，事理图谱的应用中存在诸多待解决的问题。对于该事件，需要分析其对企业关系网中哪些周边实体产生影响，影响路径及程度如何，以及对其他实体的影响是否会衍生新的事件，从而进一步扩大影响范围。

问题 1：事件分类纷繁复杂，仅靠先验知识进行预先定义事件分类体系无法充分覆盖实际应用场景

传统的做法是通过业务专家定义多层次的事件类型，构建事件类型树，基于业务专家对权益市场、固收市场、宏观经济变化的理解进行事件定义、说明、分类，划分不同事件的边界。同时，事件可以定义为金融市场的“变化”，因此不同的事件类型背后往往也关联着一系列的金融指标。通过业务专家预定义一系列标签的形式组成一系列的“事件树”，再基于“事件树”和历史数据构建不同的金融事件传导网络。但这样的做法往往难以满足实际金融市场的需求，主要是由于下列几个原因：

1、事件类型的理解不同。不同业务专家背景不同，沉淀出来的“事件树”往往存在无法统一的情况，甚至对于同一个事件的理解可能会有差异，不同类型的事件和事件之间的边界不清。

2、静态的事件树难以满足金融市场的动态发展，无法应对新的金融事件类型的出现。特别是在 08 年金融危机后，全球经济进入新常态，国内经济也在近些年来不断体现出新的特征。例如，新冠疫情对全球经济和各个行业造成了重大冲击，但是以往的“事件树”中一般归类于“重大卫生安全”等类型，大量的业务分析视角也是将 2019 年的新冠疫情与 2003 年的 SARS 对比说明，借此分析未来的影响。然而，虽然两者都是“重大卫生安全”事件，但是无论从影响时间、影响范围等不同角度，它们的差异都非常大。

综上所述，由于金融事件的复杂性，仅仅依靠一组业务专家进行事件的预先定义，无法覆盖实际应用场景。我们需要一套体系动态生成衍生的金融事件体系。

问题 2：事件相互之间存在因果、顺承等多中关联关系，这种关联关系往往还需要通过实体网络动态连接，需要更强大的描述能力

由于金融事件网络的复杂性，在相同的事件发生后，对于其它事件的影响方向可能也会不同，这往往取决于不同事件背后的实体与关系的差异，决定该事件的影响方向不同。

举例来说 A 公司股价上涨，由于其扩大产能，资本市场对其未来发展看好。那作为其竞争对手的B公司，其股价到底是上涨还是下跌，往往会取决于多种因素，包括市场需求、产能扩大的规模、以及公司和竞争对手的相对市场份额等。假设公司 A 是一家半导体制造商，它决定扩大其生产能力。对于其竞争对手B来说，这可能是一个利好的消息。如果全球半导体市场需求强劲且供应紧张，那么 A 的产能扩大可能会有助于缓解这种供需失衡，从而稳定整个市场。在这种情况下，由于市场环境得到了改善，竞争对手B也可能因此获益。这种情况下的逻辑是：如果整个行业的需求超过了供应，那么任何增加供应的行动都可能对整个行业产生积极的影响，因为这有助于维持市场稳定并防止价格暴涨或其他可能导致市场不稳定的因素。另一方面，如果公司 A 是一家汽车制造商，并决定扩大其生产能力，这可能对其竞争对手B产生负面影响。在这种情况下，如果市场需求没有增长，A 的产能扩大可能会导致市场供过于求，进而引发价格竞争。因此，对于竞争对手 B 来说，这可能会降低其销售量和利润，因此可以视为是一个利空的消息。这种情况下的逻辑是：如果一个行业的供应增长超过需求，那么这将导致供应过剩，可能引发价格竞争，进而影响所有厂商的利润水平。

综上所述，由于金融事件网络的复杂性，在描述不同事件和事件之间传导关系的同时，需要借助其相关的实体网络进行动态链接，并基于此构建强有力的描述能力。

问题 3：如何更好地对事件的影响传播进行描述和分析

由于金融事件推理的复杂性，因此需要从事件在实体网络传播和事件网络传播两个角度出发。以“公司 A 宣布破产/债券违约”为例，我们可以从这个事件的实体关系网络传播效应和事理网络的传播两个角度进行分析：

1、实体关系网络传播效应：公司 A 的破产会直接影响其股东，尤其是大股东，他们的财务状况可能因此受损，从而进一步影响他们在其他公司的投资。此外，公司 A 的竞争对手可能会因其破产而受益，市场份额可能会有所提升。同样，公司 A 的供应商和债权人可能会因为公司 A 的破产而遭受经济损失。这些影响将在实体网络中传播，影响相关的其他实体。

2、事理网络的传播：公司 A 的破产可能会被其他公司作为一个警示的例子，以防止类似的事情发生。比如，可能会引发相关行业或市场的风险防范意识增强，那些在财务管理、风险控制等

方面存在问题的公司可能会从中吸取教训，进行必要的改进。这个事件的影响会在事理网络中传播，形成新的事件并影响到其他的实体。

这两个传播过程并不是孤立的，而是相互交织的。例如，公司 A 的破产可能会引起其竞争对手的注意，并影响它们的决策，从而在实体网络中引发新的事件。同时，这个新的事件也可能成为事理网络中的新节点，进一步影响其他公司的行为。

问题 4：金融事理推理的过程仅靠关系网络数据并不足以支撑，往往需要用到大量的外部数据辅助分析

在 2019 年发生了一件巴西淡水河谷的溃坝事件，这个事件造成了铁矿石价格上涨，从而又导致炼钢成本上涨。在整个事件影响链中，部分属于行业竞争关系的企业受益，利润有所上升。但对产业链的下游造成了负面影响，成本上升导致利润下降。

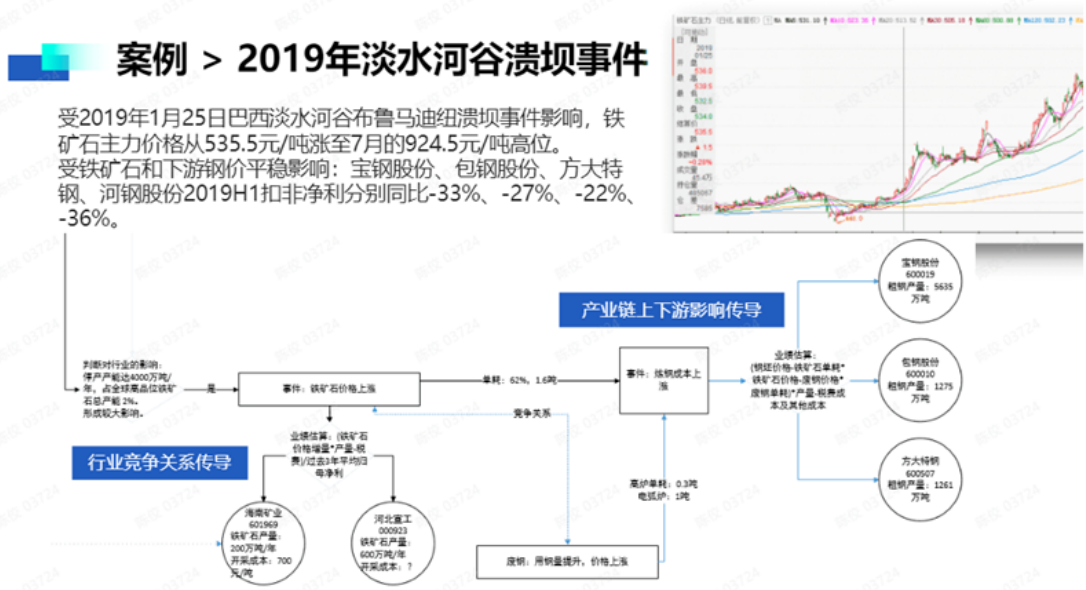

图 10 淡水河谷溃坝事件影响链传导图

整个铁矿石产业链最上游是铁矿石开采，淡水河谷矿业公司 (Vale S.A.）是全球矿业业界的重要参与者，其经营活动对全球的铁矿石供应和价格有重大影响。正是因为该公司在全球铁矿石产业链的重要性，导致了其溃坝事件导致了全球铁矿石价格上涨。

中国作为基建大国同样也是全球最大的铁矿石消费国，对全球铁矿石市场的需求严重依赖,因此一家巴西的公司发生重大事件后，才会导致铁矿石产业链发生【原材料上涨】，并将该事件成功传导到了国内资本市场。另外一方面，铁矿石进口后，会在高炉中经过冶炼生成生铁，然后生铁在转炉中经过再炼生成粗钢，在这个过程中需要经过熔炼、转炉炼钢、脱气、连铸等过程。粗钢生产出来后，粗钢是钢铁产品的基础，可以根据需求进一步加工成各种不同的钢材，比如长产品包括钢筋、线材等，又如扁平产品如热轧卷、冷轧卷等，这些产品往往会用于汽车、家电、造船等行业，另外还包括管材（无缝钢管、焊接钢管等），除此之外还包括特钢、高强度钢等不同

产品。而在这些产业链上下游存在各种各样的中国上市公司，如宝钢、包钢、方大特钢等。具体的传导逻辑和影响，需要再结合论元的细节进行分析，分析内容包括：

1、该企业本身是否有在衍生品市场进行对冲，如果有的话，对冲的货值有多少。

2、该企业产品的市占率和细分产业的竞争格局如何，一般认为特钢的竞争格局相对普通钢铁会更好。

3、该企业是否有能力将上游生产压力，转移到下游等等。国内是否有上游替代品可以替代。

只有将上述论元进行细颗粒度拆分后，并引入相应的外部数据，才能构建完整的传导网络。

2.4 属性图应用于事理图谱所存在的问题

通常情况下，对事件的影响分析，都是分析师基于对事件的理解，在基础事实知识上反复查询洞察，再结合常识性知识得出事件影响结论。由此可见，整个事件的推理过程都在图谱之外，因为基础事实知识里并不具备常识和推理逻辑，纯粹的事件图谱也无法表达事件的脉络。在实际应用中为了完成事件推理，各种逻辑不得不分散在图谱之外的各种地方，再通过各种图谱外挂的方式运行推理。这样的应用方式，必然给事理图谱带来了很多应用上的问题：

- **预定义 Schema 静态性和事件的动态性矛盾。**事件纷繁复杂,若是采用强 Schema 结构约束的属性图,一般无法实现完成事件的预定义,只能针对限定场景;若采用 Schema free 属性图,则过度宽松模式会造成数据管理和使用成本越来越高。
- **无法表达整个事件传递的事件脉络。**由于图谱中只有基础事实，没有事件的定义，更没有事件的传递关系，事件分析的专家规则更是无从谈起，自然也就没有办法表示出事件传导影响的整个脉络。要表达出事件脉络，不仅需要能将事件随时间的演化过程表达出来，更需要结合抽象实体将事件在事件域内通过抽象层级之间的关联性表达出来。
- **图谱和推理逻辑分离难以评估推理逻辑正确性，更不利于推理逻辑的复用。**由于 Schema 和推理逻辑分离，在维护基础事实数据时，就无法评估对外部推理逻辑执行正确性的影响。比如数据变更了属性名称，删除了关系等，有可能会造成存在图谱之外的推理逻辑运行失败。这样的问题在传统的事件图谱上是无法避免的。另外，推理逻辑可能是查询语句+脚本的组合，这些内容可能都管理在分析师各自的本地存储上，不利于将一些通用性强的推理逻辑形成复用。
- **事件传导推理结论的可解释性差。**由于外挂推理逻辑可能是多条查询语句+脚本的组合方式，最终事件影响的实体计算出来的时候，是无法通过结果直观看到事件起因到结果的演绎过程，这时可解释性就成了黑盒，只有对查询语句和脚本进行理解才能明白推理逻辑。

2.5 知识建模中结构定义与语义表示的耦合导致的复杂性及异构性

RDF/OWL 是语法层面的表示框架定义，导致较高的建模学习成本；在传统知识工程的本体建模中，要通过描述逻辑语法定义分类体系；属性图 LPG 语法元素简单，但仅是数据结构的声明；上述方法都没解决'设计模式'本身问题。在实际业务落地过程中，由于建模过程中数据结构定义和知识语义本体设计的耦合，导致建模时的决策困难。领域图谱的 Schema 设计是比较主观的，同一类主体因命名、粒度不同而定义为不同类型，普遍存在因 Schema 定义不同而带来异构性问题，阻碍了知识的传播和复用，也会进一步加剧知识的不一致性。

2.5.1 因业务目标不同带来实体类型颗粒度差异导致的重复构建问题

在黑产图谱的应用中，业务需要对 Person 实体进行分类，分别判定其是否是黑产人员，黑产人员又可划分成为赌博人员、庄家、洗钱等等，如图 11 所示。

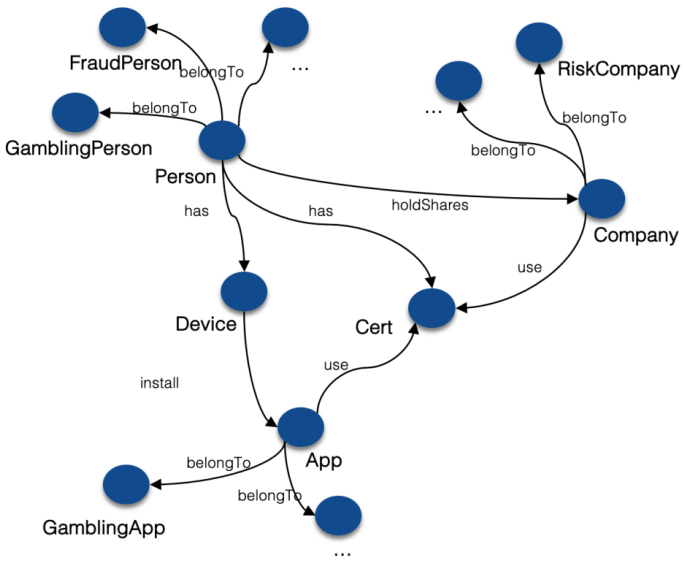

图 11 黑产图谱建模过程中实体颗粒度扩散

同一图谱项目也会根据内部不同诉求再次新建实体类型，同实体多类型的诉求也通常会通过冗余新类型来解决，导致图谱项目及 Schema 越来越复杂，业务演进到某一阶段被迫删库重新设计，以黑产图谱为例：

- **对不同 App 分析的诉求：**黑产团伙像应用工厂一样，批量换壳生产大量 App，需要对 App 进行分类细化类型，以便应对不同风控策略
- **对实体类型按需细化的诉求：**团伙挖掘任务会把部分人是否是庄家、诈骗关联起来，单独衍生了 GamblingPerson、FraudPerson 等类型，随着业务的发展实体的分类会继续细化。

这部分问题通常和业务场景强相关，随业务变化而变化，随场景不同而不同。而从数据管理的角度看，这些 App 或 Person 使用了相同/相似的数据结构；但从业务逻辑视角又需要在语义层面

的类型细分。不同视角的 Schema/本体建模混在一起导致用户理解和维护成本持续升高。实体类型的冗余构建也增大了数据表的准备和维护成本。

2.5.2 不同图谱对同一类实体定义不同

以资金流转为例（跨图谱），如图 12 所示。

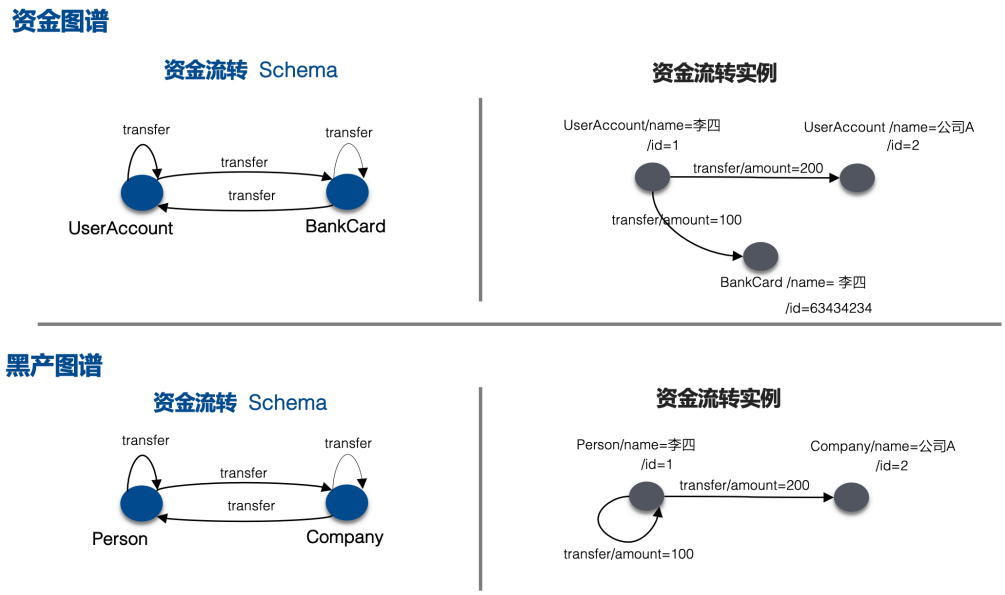

图 12 跨图谱融合示意图

在黑产图谱中，我们重点关注用户与用户、公司之间的交易关系，以便确认幕后黑手。而在资金图谱中，我们的重点是对资金流向进行分析，因此会对涉及到的资金产品部署追踪及管控策略，并将更细粒度的资金产品作为实体类型。在这两个场景中，我们都涉及交易关系的处理以满足各自的业务需求。但这里存在如下两个问题：

- 不同业务对相同数据做类似处理，对于共性的需求无法沉淀，也无法将业务累积的经验共享，每个新接入的业务均需要重头开始准备数据，增加了业务使用门槛。
- 跨图谱的知识共享，例如资金图谱中存在 BankCard 实体，因反洗钱、反诈等业务需求需要 BankCard 实体时，无法隐私安全的使用资金图谱中的 BankCard。

2.5.3 因构建成本带来的定义为属性、关系的抉择困难

在属性图模型下，每类实体、关系都需要独立的数据准备，M 种实体、N 种关系，因属性量的差异，需要准备 M + N 种消息结构或数据表以完成图谱构建，而原始数据往往分布在<M，带来较大的数据准备成本，随着图谱关联分析的需求越来越高，这种成本持续膨胀。属性图中定义的所有实体、关系，需要独立的数据准备，业务需为当下的简单应用和未来扩展性之间做平衡取舍。当属性直接构建为关系后，增加了简单应用使用的复杂度，如属性过滤的缺失等。

考虑一个简单的问题：将使用相同 Wifi 的设备连接起来。如图 13 所示，我们通常的做法是：1）构建 Device, Wifi 实体类型, Device -[useWifi]->Wifi 关系类型。2）为如上实体、关系单独数据准备，并需要为 Wifi 生成独立的实体 ID。

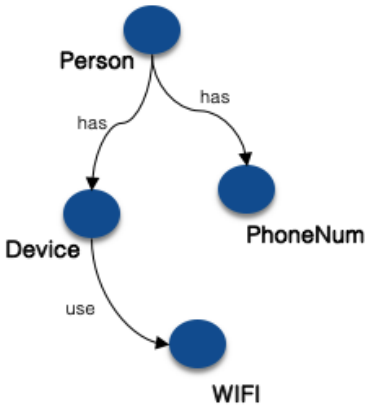

图 13 黑产图谱属性转关系需求

大量增加数据准备的复杂度, 需要为每一类实体、关系单独准备数据，极端情况下若每个类型一张数据表，从 2 张表变为 6 张表，洗数据的工作量变大。假定有 m 张实体表，平均每张实体表中有 n 个属性列需要变为关系，那么我们需要从 m 张表中，洗出总共 m*(2*n+1) 张表，用户使用门槛变得很高。

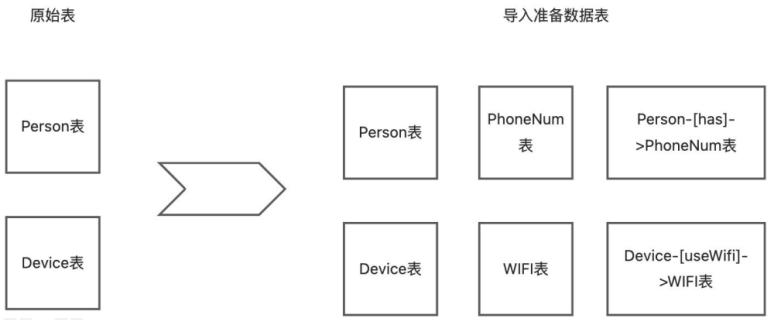

图 14 因点/边数据准备带来的成本膨胀

2.6 对领域知识多元异构性表达能力不足

在金融领域图谱落地过程中，存在用户行为、行业事件、宏观事件等时空多元的异构表达需求，如事理图谱既要表达单个事件的时空多元关联，又要建模事理层的因果、顺承、共现、结构等简单或复杂逻辑关联，基于 RDF/OWL 很难实现无损表达，超图(HyperGraph)[12]的引入能进一步缓解此类问题，但也没很好的和 RDF/OWL 体系融合，加剧了用户应用和理解成本。

2.6.1 事件时空多元结构的表示问题

事件多要素结构表示也是一类超图(HyperGrpah)无损表示的问题，它表达的是时空多元要素的时空关联性，事件是各要素因某种行为而产生的临时关联，一旦行为结束，这种关联也随即消失，如图 15 所示。

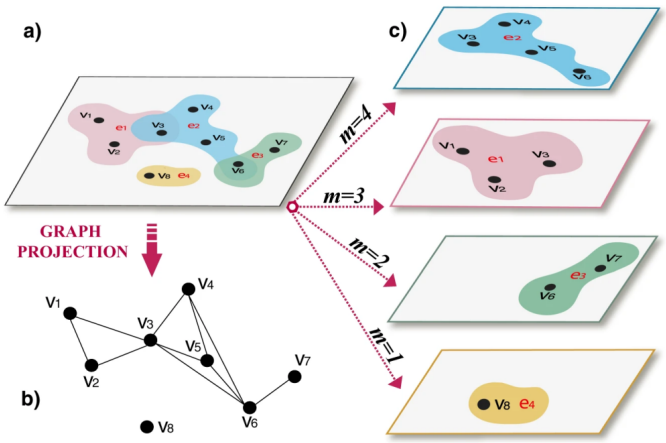

图 15 超图结构表示[13]

RDF-Star[14]的表示方法扩展了 RDF 对此类建模方式的表达，2022 年 W3C 也成立了 RDF-Star 工作组，为 RDF 进一步打补丁。以事理图谱的应用为例，企业的一次安全生产事件，它的简单结构表示如图 16 所示：

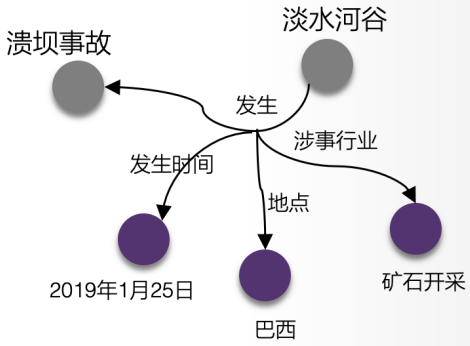

图 16 基于 RDF-Star 三元组的多元关联扩展

在<s, p, o>三元组的表示形式下，第一步需结合时间要素扩展为<s, p, t, o>才能进一步体现时间维度的约束，表示为<企业、发生、发生时间、安全风险事件>。但事件关联往往是复杂的多要素组合，将描述事件的各个方面拆解成独立的元素, 如图 16 所示进一步对关系要素的拓展。企业中基于属性图构建领域图谱已发展多年，RDF-Star 如何在属性图中落地并没有解决方案，我们需要基于属性图构建时空事件超图的表示能力，才能构建事理图谱所需要的事件表达和推理能力。

2.6.2 事理顺承、组合、结构、逻辑依赖问题

事理图谱有本体层，这意味着事件之间、事件和实体之间不仅有横向的由此及彼的关联，还有纵向的由特殊到一般/由一般到特殊的关联。横向关联是漫游、是联想、是类比，纵向关联是归纳、是演绎、是演化。所以，相应的架构，要在综合考虑到这些情况的基础上，慎重做出决定。具体到定义与实例化层面，又分为抽象实体、具体实体、抽象事件、具体事件这四块内容，物理上是联通的，是一张图，逻辑上则可以左右划分为实体域和事件域，上下可以划分为本体域和实例域，形成所谓“四象限”架构[15]。如图 17 所示：

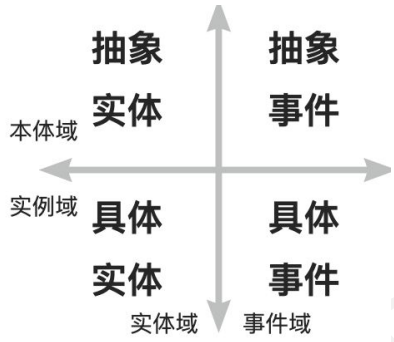

图 17 事理图谱四象限

这里面临的主要问题是，事件模型与事理模型共存表示的问题，常见的事件图谱仅仅表示了卸掉了论元的裸事件的关联关系，但在企业级的应用中，事件实例中蕴含更加丰富的信息，如企业风险事件中包括了涉事主体、涉事行业、是否停产等信息，他们可以作为裸事件更详细的补充，两者互为增益。我们需要事件模型与事理模型的共存，而事件模型是一个时空多元的超图结构表达，事理层又存在因果、顺承、逻辑等可推理的组合关系，如 土地价格上涨导致财政收入增加，“A 省土地价格上涨”又是行政单位与抽象事件之间的二元组合，也应当可推导出“A 省财政收入增加”，财政收入增加又会级联到影响链的传播。类似的，还有论元之间的上下位之间的表达，如“加息”，“日元加息”等。最后可形成“事件→抽象实体（上位）→抽象实体（下位）/具体实体→事件”的具体化路径。

2.7 知识间逻辑依赖带来的一致性及传导推理问题

领域图谱不同属性、关系之间也会存在隐式逻辑依赖关系，金融风控类的应用场景也需要通过建立属性要素之间的逻辑依赖来构建风险的自动传导能力，属性图模型下要求所有的关系、属性都必须提前清洗准备好。但往往会存在因计算时效性、逻辑正确性等带来的不一致问题，对多要素之间的逻辑关联依赖，这种问题也会更加明显，同时也增加了前置计算/构建的复杂度。

2.7.1 因数据逻辑依赖带来的不一致和冗余构建问题

图 18 简单示例了黑产图谱中因跨实体隐式逻辑联动导致属性错误问题。

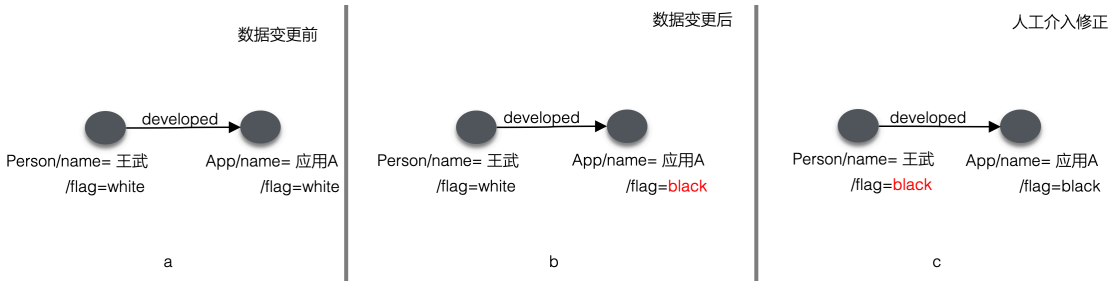

图 18 隐式逻辑关联

从黑产发现的视角，业务定义了这样一条规则“当 A 公司发布的 App 被标记为黑产后，A 也需要被标记为黑产"，如图 18b，公司和 App 都带有 mark 属性，当应用 A 被业务举报被判定为 black 时，此时王武依然标记为 white，此时数据已经出现不一致，需要等待外部系统计算完成后一步更新，如图 18c，或人工干预手动介入处理，错误的数据或更新延迟会导致业务误判得出错误的结论，在更正期间图谱属于不可用状态。

2.7.2 因逻辑依赖传递导致风险传导/透传受阻问题

在 2.3 节的 2019 淡水河谷溃坝事件中溃坝事故导致原材料价格上涨，继而引发下游企业生产成本上升，最终导致下游企业利润下降的传导链条。从事理层的角度看，这个事件是从企业生产事故，传导到了产业链，再引发下游企业金融风险。在传导过程中，并不是简单的关系扩散，而是含有逻辑依赖的因果传递，并且在传导链条的每个事件节点上都还保有起因事件的关键要素，这些都是在基于基础事实的事件图谱中难以实现的，一旦出现逻辑依赖就会导致事件传导受阻。要构建事件风险的传导，需要有事件的触发机制、事件影响传递和传递规则。

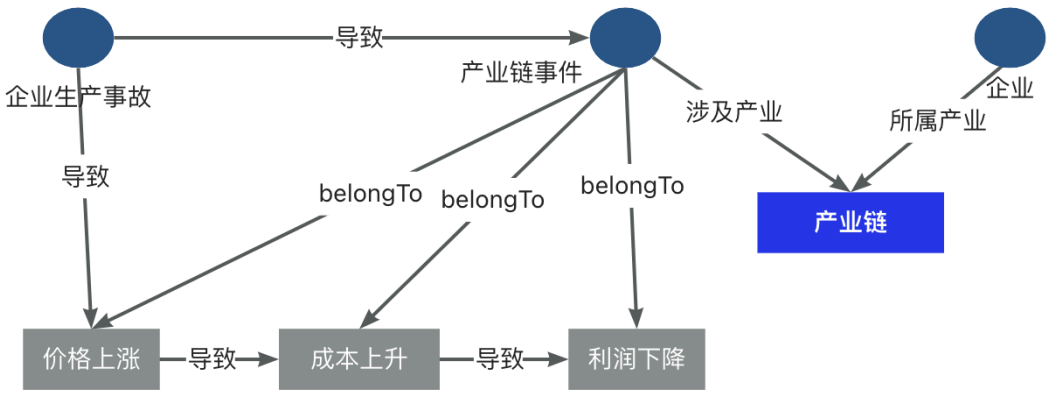

图 19 事件在实例间的影响传递

- **事件触发机制。**基于外部相关资讯抽取或对关键数据变动的监听得到结构化事件要素，得到事件实例。基于具体的事件实例，触发相应的事件传导规则。
- **事件影响传递。**事件影响传递沿关系直接传播。如事理图谱中的企业安全生产事故的影响，通过事件实例的发生主体的产业特征，传递到其所属产业。事件的关系传递可以表达符合

什么条件的事件可以传递到另一个目标事件，条件中可以使用传递路径中关联子图查询得到的实体、关系的各种属性，如判断发生主体是否上市公司、公司的产业和下游产业等 。

传递过程中，判断逻辑中可以对当前判断条件所处位置的所有前序节点/关系的相关属性进行引用。如图 20 所示，在“价格上涨”中需要引用“淡水河谷溃坝事件”的主体的所属产业属性；在“成本上升”中需要引用“价格上涨”的产业的下游产业属性；在“利润下降”中需要引用“成本上升”的产业属性。

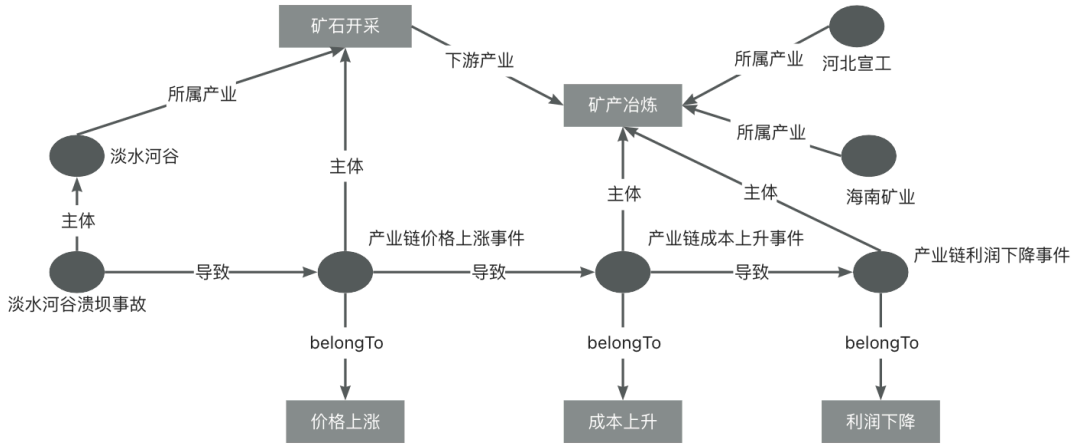

图 20 基于规则的事件概念归纳及影响传递

2.8 面向非完备数据集的图谱构建与演化问题

企业级知识图谱的构建往往是基于非完备数据集的，来源和构建策略都会不断变化，需要通过不断的迭代来持续提升覆盖率、准确率，减少冲突和错误。这种不完备性往往包括两个方面，数据来源的多源异构性及跨图谱的多元异构性，数据来源的多元异构性表现为同一个实体类型的不同实例、不同属性存在>=1 个数据源，既要解决不同数据源的消歧、对齐、融合，又要根据不同的置信度策略做评估和选择实现数据源的可回溯、可量化。跨图谱的异构性表现为不同领域图谱中存在对同一类实体的重复定义，但因业务领域需求和数据的差异性又有不同，需要实现跨图谱的融合与链接。

2.8.1 多源异构数据构建图谱的可靠融合、可信回溯问题

在企业图谱应用中，同一个实体类型的不同属性、关系可能来源于不同的数据源，基于多源异构数据构建实体通常的做法为实体链指 和 实体归一，实体链指为每一次数据更新找到一个准确的唯一实体 ID，实体归一则是将档次更新的属性和归一后实体实现属性、关系的合并。结合事理图谱的案例如图 21 所示，事理图谱涉及的企业实体构建过程中，涉及到多种数据源的合并，如企业公告抽取、工商基本信息、法院公告等。

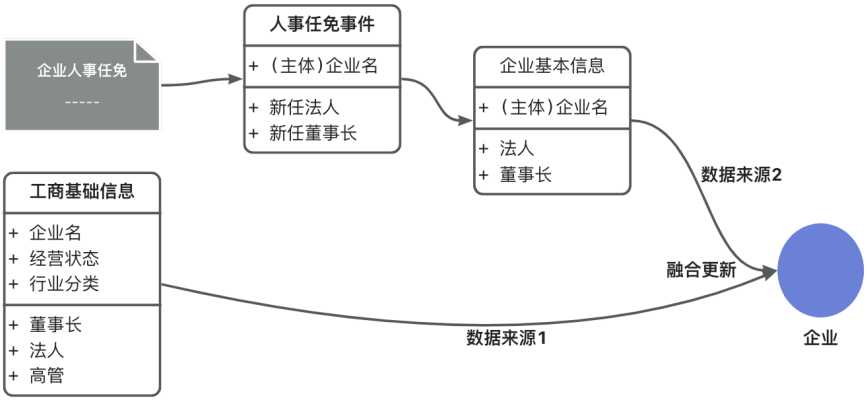

图 21 基于多元异构数据源的企业实体更新

企业实体定义一般为<Company, legalPerson, String>的状态，实际应用中出现属性值冲突时，需要根据来源 (sourceType)、算法预测得分 (score) 等维度来决定如何保留或更新。比如 工商基础信息的置信度是最高的，需要实现无条件覆盖，但工商信息和企业公告的更新时效不一定一致，可能出现企业公告已抓取但工商信息还没同步的情况，需要在属性要素上保留二级的描述信息，可形式化表示为: <Company, legalPerson, String> as p，为 p 添加 p.sourceType,p.score 以及在 Schema 上记录 p 的覆盖规则，比如: p.fuseRule = "sourceType ==‘工商信息’; score > p.score "。

2.8.2 跨图谱融合的实体对齐、实时更新及融合/溯源问题

跨图谱融合的问题和 2.5.2 类似，当实现黑产图谱和资金图谱中的用户实体合并时，需要确定在新的 FuseEntityType 中如何保留属性和关系。

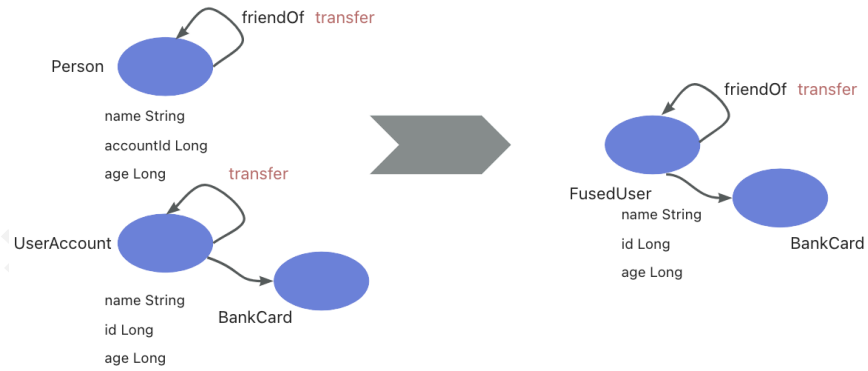

图 22 跨图谱稳定融合及可溯更新问题

为了保证 Person、UserAccount 体属性/关系更新时，能及时触发对 FusedUser 的更新，且正向保证结果的稳定性，反向支持结果的可解释、可跟踪，我们需要对属性和关系进行扩展，记录融合和更新策略的附属属性，并在实体更新阶段触发执行。面向非完备数据集的图谱迭代演化，知识建模框架需要解决的问题有：

- 属性/关系可携带附属属性。 这些附属属性用于描述属性/关系的来源、置信度、相关度、作者等相关资产信息。

- 属性/关系可定义更新策略。需要支持可执行的规则表达式，用以定义属性/关系的选择和优先级策略。这样可以在不同数据源随机到来的情况下保证结果的稳定性。
- 实体类型可绑定链指算子。在工业级应用中，许多数据无法获取标准化 ID。因此，我们需要使用文本匹配、时空聚类等链指策略来找到目标实体 ID。需要支持为目标实体类型绑定链指算子，以确保不同来源数据更新时可执行相同的链指算子从而保证结果的稳定性。

2.9 无语义不可编程的属性图所存在的问题总结

首先，知识管理是伴生业务全生命周期的，这要求其具备迭代演化的能力，支持业务持续迭代又有效避免组合爆炸和重复构建。然后，知识管理面对的是非完备数据集、多源异构数据源、多业务专家视角下的复杂知识建模问题，这也要求其具备可编程范式实现差异化视角、多源异构数据的轻量级对齐，降低系统复杂度。最后，知识管理需建立必要的知识分层、分类体系，实现各层级之间有效的联动、归纳、演绎等，实现静态常识的自动剥离支持跨业务的高效复用，实现核心资产的有效沉淀。 在后面的第 3/4 章将详细介绍属性图上的语义增强可编程范式。

蚂蚁集团 x OpenKG

第 3 章 语义增强可编程框架 SPG

为解决第 2 章提出的问题，我们结合企业级业务场景的应用特点，抽象出了基于属性图的语义表示框架 SPG。该框架从三个方面来定义和表示知识语义。首先，SPG 明确定义了"知识"的形式化表示和可编程框架，使其可定义、可编程，机器可理解和处理。其次，SPG 实现了知识层级间的兼容递进，支持工业级场景下非完备数据状态的图谱构建和持续迭代演化。最后，SPG 有效衔接大数据与 AI 技术体系，支持对海量数据进行高效的知识化转换，帮助提高数据价值和应用价值。通过 SPG 框架，我们可以更加高效地构建和管理图谱数据，同时可以更好地支持业务需求和应用场景。由于 SPG 框架具有良好的可扩展性和灵活性，新的业务场景可以通过扩展领域知识模型及开发新算子，快速构建其领域模型和解决方案。

3.1 SPG 语义框架模型

SPG 的总体语义框架模型如第 1 章图 5 所示，并在第 1 章中也做了简要概述。首先，SPG 从三个维度对知识做形式化定义：

1) 领域类型结构约束，客观世界中不存在无领域类型的事物，但数字世界存在大量文本/数字无领域类型表示。SPG DC 要求每个事物 (Thing)都必须有一个明确的领域类型 (Class)，并且该领域类型必须有自己内在的结构表示，包括属性、关系等，通过关系与其他事物发生关联。同时，SPG DC 还按照领域知识的由动态到静态、由特殊到一般、由实例到概念的原则，将领域类型分为事件超图 (Event HyperGraph)、实体 (Entity)、概念 (Concept)三类。这样可以方便业务高效的知识分类和复用，并且实现知识的动态到静态自动分层，详见 4.1.1 和 4.2 章节的描述。

2) 领域内实例唯一性，客观世界不存在完全相同的两个事物，但数字世界因数据拷贝、不同描述视角、多源异构等存在大量同一事物的不同实例，为在数字世界构建和客观世界一致的表示，SPG 要求每个领域类型下的实例 (instance)必须是唯一的，以保证知识的准确性和一致性。为此，SPG Evolving 通过 SPG-Programming 提供可编程的实体链指、属性标化、实体归一等算子能力，可以使用内置或自己开发算法 Operator 实现链指、归一算子，不断提升实例的唯一性，在第 7.2 章 SPG-Programming 中有相关描述，更详细的介绍预计在 SPG 白皮书 2.0 中发布

3) 知识间逻辑依赖性，客观世界不存在不和其他事物关联的事物，我们往往通过事物之间的联系来认识事物，这种联系即表达为事物的内在特性，也表达为和其他事物之间的逻辑/物理关联，既有归纳意义的公理共通性，又有实例层面的私有特殊性。SPG 通过 SPG Reasoning 谓词/逻辑体系定义知识之间依赖，包括属性、关系、类型等之间的逻辑依赖与传导。同时，SPG 还通过谓词体系定义基础的谓词原语，以支持知识的推理和推断。这样可以更好地处理知识之间的关联和依赖关系，并且支持对复杂的业务场景进行建模和分析。在 4.3 及 4.4 章节有详细描述。

然后，SPG 框架实现了知识层级间的兼容递进，以适应工业级的知识图谱应用。在实际应用中，业务往往面对非完备数据集、非完备专家经验、非完备图谱理解的客观现实。因此，一方面业务希望借助图谱快速实现业务价值，另一方面业务数据覆盖和图谱经验也是非完备的，需要通过不断的业务迭代来逐步加深图谱的理解和应用。与此同时，RDF/OWL 要求完备知识体系下做知识交换，这与实际应用场景的客观现实是不一致的。为了解决这一问题，SPG 在定义知识表示时，要求从左到右必须是兼容递进的。用户可以选择最简单的 SPG Compatible 模式直接从大数据体系构建属性图的表示，也可以增加 SPG DC 领域模型约束提升主体模型的语义明确性。此外，通过 SPG Evolving 添加链指、归一算子来不断提升主体的唯一性和主体间的语义关联。最后，通过复杂的谓词与逻辑体系构建知识的符号化表达。通过 SPG 的分层兼容递进，可以极大降低图谱业务的落地成本。在图谱应用的过程中，用户可以根据自身需求和数据情况选择不同的模式和算子来逐步完善和优化领域知识图谱。

最后，SPG 框架通过分层递进，可以有效衔接大数据架构，实现数据体系到知识体系的自动构建。具体而言，基于大数据表构建 ER 模型转换到 SPG (ER2SPG)的方式，我们可以将大数据体系中的数据转化为 SPG 知识图谱表示，从而实现数据和知识的无缝衔接。同时，通过 SPG-Reasoning 构建机器可理解的符号体系，可以方便地通过知识约束、逻辑符号等实现与深度学习模型的联动，为知识图谱应用提供更多的可能性。除此之外，SPG 框架还期望通过 SPG-Reasoning 实现与大模型的符号化联动。具体而言，通过将大模型的输出结果映射为符号化表示，将知识图谱符号化表示输入给大模型，可以更好地将其与知识图谱进行融合和协同，从而实现知识和模型的高效交互和共同进化。这一点对于实现更加智能化的应用场景具有重要意义。

综上，SPG 框架通过衔接大数据架构和构建机器可理解的符号体系，实现了从数据到知识的自动转换和应用。在未来，SPG 框架还将继续发挥其优势，探索更多的应用场景，并且与大模型进行更加紧密的联动，为知识图谱应用带来更多的可能性。

蚂蚁集团 OpenKG

3.2 SPG 分层架构

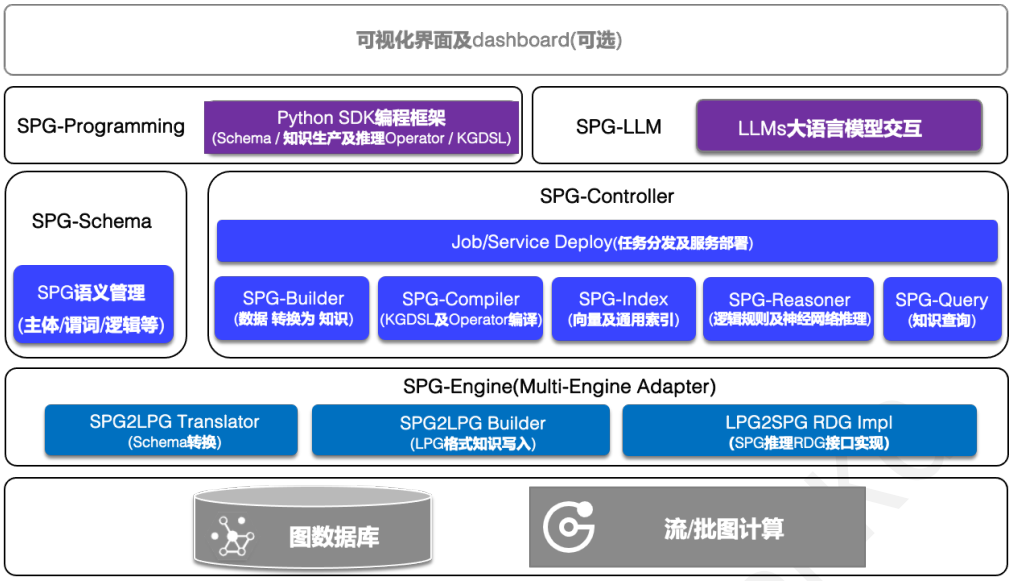

图 23 基于 SPG 的知识引擎总体架构

SPG 的核心目标是构建基于 SPG 的标准化知识引擎架构，给领域图谱构建提供明确的语义表示、逻辑规则定义、算子框架（构建、推理）等，支持各厂商可插拔的适配基础引擎、算法服务，构建解决方案等。 本章节简要概述总体框架。

- SPG-LLM: 负责 LLM 大模型交互子系统，如自然语言 NL、用户指令、查询等。详见第 8 章描述。
- SPG-Schema: 负责属性图语义增强的 Schema 框架设计，如主体模型、演化模型、谓词模型等。详见第 4 章描述。
- SPG-Controller: 负责控制中心子系统设计，如控制框架、命令分发、plugin 集成等。详见第 6 章描述。
- SPG-Programming: 可编程框架子系统，负责 SDK 框架及编译子模块的设计，如知识生产、知识演化、专家经验投影沉淀、图谱推理等。详见第 7 章描述。
- SPG-Engine：知识图谱引擎子系统设计，负责多引擎的 integration/adaptation layer 的设计，如推理引擎、查询引擎等。详见第 5 章描述。

3.3 SPG 的目标能力

我们期望构建的是基于 SPG 的新一代认知引擎基础架构，总体能力范围图 24 所示，图中图例也表示了本白皮书的覆盖范围。

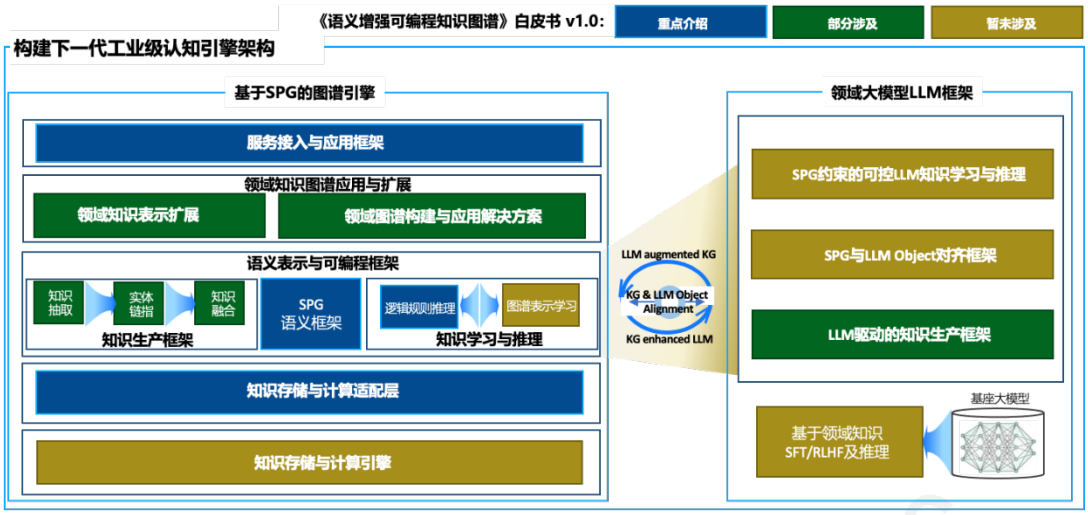

图 24 SPG 和 LLM 双向驱动的目标架构(初稿)

本白皮书为《语义增强可编程知识图谱 SPG》1.0 首次发布，结合第 2 章介绍的两个案例，介绍知识图谱发展当下存在的痛点问题和可能的解决思路，同时也介绍 SPG 的解决思路、核心能力和总体框架。未来，SPG 将持续完善白皮书内容，包括领域模型扩展、可编程框架、知识生产引擎、知识推理引擎、LLM 与 KG 的双轮驱动等。同时，SPG 也将加速语义及基础引擎框架的开源，促进知识图谱的产业化落地。本次发布的 1.0 版白皮书将重点介绍如下内容：

- SPG 语义基础框架，介绍 SPG 产生的背景及解决的核心问题，并通过两个业务案例介绍 SPG 的语义框架和 Schema 模型。
- SPG 逻辑规则框架，介绍 SPG 的逻辑规则体系，如何基于 SPG 实现逻辑规则与事实知识的有机融合。
- SPG 多引擎适配层，结合 SPG2LPG 和 LPG2SPG 的适配抽象，详细介绍适配层的能力模型，以方便各厂商图存储、图计算引擎的高效接入。

我们将持续更新白皮书内容，包括 2.0 和 3.0 版本。本次发布中，部分内容如可编程框架、知识推理等只做了概要式介绍，未来我们将会单独重点论述，还有 SPG 与 LLM 的双驱动我们也将保持持续的探索与突破，本白皮书第 11 章也给出了 SPG 未来的发布计划。

第 4 章 SPG-Schema 层

4.1 SPG-Schema 总体架构

SPG 的核心目标是充分利用属性图兼容大数据架构的优势，并以此为基础，从工业实践的实际问题出发，实现语义增强，构建完备的语法体系。本章节主要结合 SPG DC 主体分类模型扩展与 SPG Reasoning 逻辑谓词语义扩展两个方面详细介绍。首先，在大数据表 Schema 定义或大数据表字段定义基础上扩展主体模型是最直接、最灵活的，将大数据表模型的列字段定义或字段定义映射到 SPG 主体模型的类型、属性、关系表达，通过映射将多源异构的数据表映射到非完备状态的主体结构。然后，再基于非完备状态的主体结构持续迭代演化并实现逻辑谓词语义的扩展。在这个过程中，SPG 充分借鉴了 ρdf 最小可用集及 OWL 逻辑谓词能力，定义 SPG 主体模型最小语义单元并扩展 SPG 在谓词语义、逻辑规则等方面的表达。

4.1.1 主体分类模型扩展

为了更好的增强 LPG 中对于节点类型的语义表达，SPG 在 LPG 的节点类型和边类型之上扩展并引入更多主体分类模型对节点类型进行扩充以兼容更加多元的知识表示，扩展的主体类型分为标准类型、概念类型、实体类型、事件类型等，领域分类模型如图 25 所示。

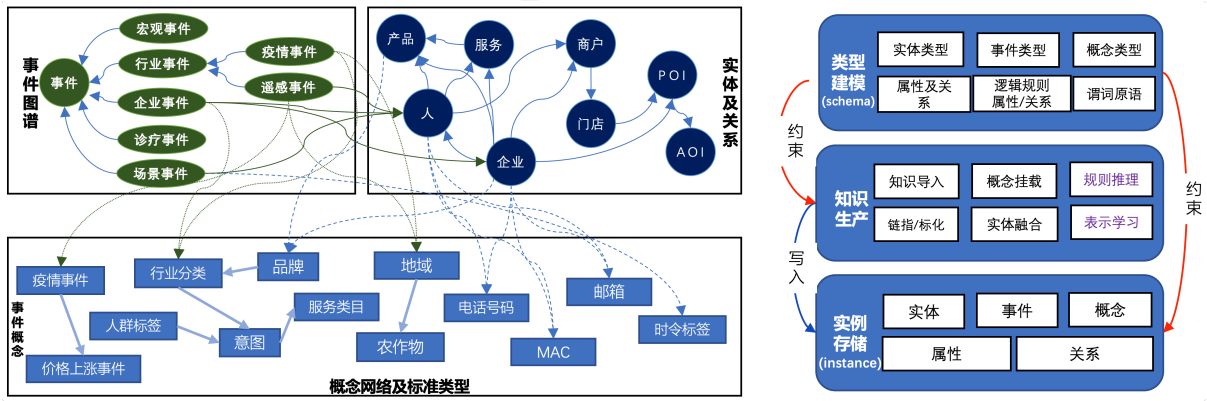

图 25 SPG 领域分类模型

SPG 主体分类模型的简要解释如下：

- **实体：**业务相关性比较强的客观对象，通多属性、多关系刻画的多元复合结构类型，如用户、企业、商户等。
- **概念：**实体从具体到一般的抽象，表述的是一组实体实例或事件实例的集合，是一种分类体系。相对静态，也是常识知识，具有较强复用性，如人群标签、事件分类、行政区划分类等。为简化企业应用，标准类型也放到了常识概念中。

- **事件：**加入时间、空间等约束的时空多元类型，如通过 NLP、CV 等抽取出来的行业事件、企业事件、诊疗事件或因购买、核销、注册等行为产生的用户行为事件。
- **属性：**属性是实体、事件、概念等的组成要素，用以表述一个复杂结构的各个独立要素，每个属性要素又会关联为一个具体的简单或复杂结构，如基础类型、标准类型、概念类型等。
- **关系：**关系的定义和属性基本一致，表达同一个复杂对象与其他对象之间的关联，关系和属性的区别是，若关联对象为实体类型则为关系。

1. 实体类型

实体类型是 SPG 的基础类型单元，它是有多个属性和关系定义组成的复合数据类型，它是在 LPG 的 Node 类型上直接扩展得到的。在第 2 章中，我们深入分析了目前 LPG 知识管理方面存在的困难。为解决实体类型数据准备成本高、属性类型语义能力缺失等问题，SPG-Schema 在 LPG Node 类型上扩展了属性取值类型的表达，取值类型可为标准类型、概念类型、实体类型等。为实现实体类型的继承和复用，我们借鉴并扩展了 subClassOf 主体谓词语义，实现子类对父类的属性和关系继承。针对实体类型异构性导致相同实体类型命名不一致的问题，我们支持扩展融合实体类型，并通过实体链指和实体归一算子来逻辑对齐不同图谱中的实体类型，未来，我们将在 SPG 白皮书 2.0 中重点发布算子绑定部分。

2. 概念和事件

针对领域图谱 Schema 存在的设计主观性问题，由于不同的业务领域中内部诉求的不同，会导致不同业务对同一类实体的命名和粒度需求不同而产生多种相似类型，而这些相似类型的数据均来自同一份源数据，这严重影响和阻碍了知识的传播也造成知识和数据的不一致。为了避免这种不一致性，SPG-Schema 引入概念类型来对相似类型进行分类，通过概念和基本实体类型进行联动以化解知识异构问题。

另外，事理图谱中的事件模型存在时空多元关联，并且在建模事理层时也需要同时对因果、顺承、共现、结构等简单或复杂逻辑进行关联关联，为了解决 LPG 无法完美表达这类需求的问题，SPG-Schema 引入事件概念来对分类模型进行扩充来更好的表达事件之间、事件和实体之间以及事件和概念之间横向、纵向的关联。

概念事件四象限图按照领域知识的由动态到静态、由特殊到一般、由实例到概念的原则描述了实体类型、概念和事件之间的关联，如图 26 所示。概念事件四象限图具体到定义与实例化层面可以被分为抽象实体、具体实体、抽象事件、具体事件这四块内容，物理上是联通的，是一张图。

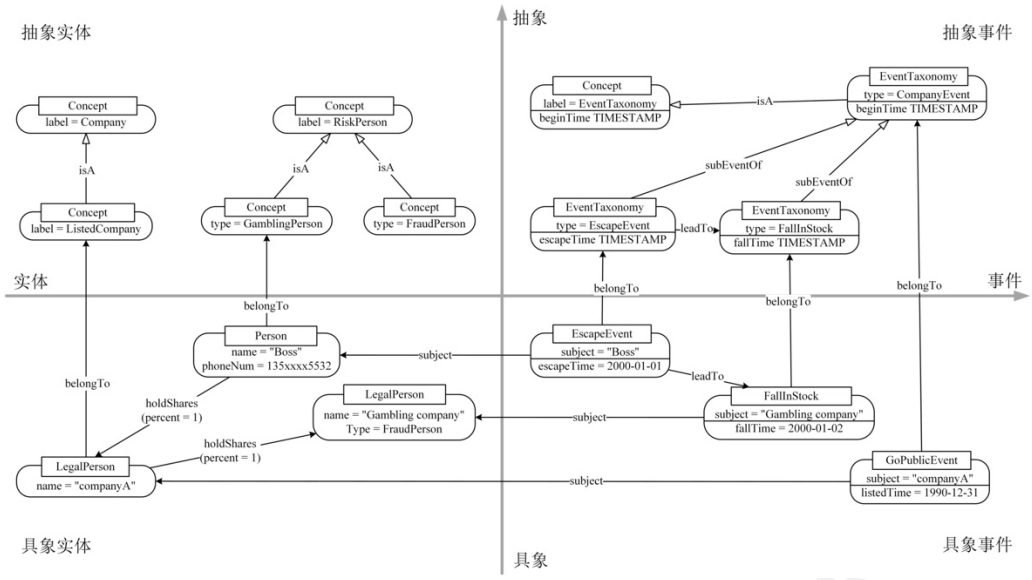

图 26 概念事件四象限图

具体来说四象限左右划分为实体域和事件域，上下可以划分为抽象域和具象域，上下也可以称之为概念域和实例域。其中抽象实体类型表达的是对具象实体类型的抽象概念，具象实体是实体类型的具体实例化；而抽象事件表示为抽象事件概念以表达事件之间的因果和顺承，具象事件则是对应事件类型的具体实例化。

3. 标准属性

在 RDF/OWL 模型中，每类实体、关系、属性都需要独立建模，而属性图语法元素虽然相较于 RDF 来说简单，但仅是数据结构的声明，无法有效利用属性知识进行知识传播。在实际的业务落地过程中，却往往需要利用实体属性进行知识传播和分析，针对 LPG 中没有明确领域类型约束的属性只是单纯字面上的文本或者数字，既无法保障属性内容的完整性和正确性，又难以基于结构实现有效的查询及关联传导。正是这种需求导致了我们需要为属性进行额外的建模，并由于属性量的差异带来的较大的数据准备成本，会随着图谱关联分析的需求越来越高而产生的更高的成本。

为了更好的平衡 RDF 图和 LPG 之间对于属性的取舍，并有效减少数据准备成本，SPG-Schema 引入标准属性类型简化数据依赖关系。标准属性的应用，可以自动显式的将属性图中的文本属性物化成关系，这增加了知识的可传播能力和隐式关联；由于使用标准属性替代关系建模，此处无需显式定义关系，通过标准属性的语义传播实现实体类型之间的关系传导。

4.1.2 语义及规则推理能力扩充

LPG 在通用的建模过程中仅包含节点 (Node)和关系 (Edge)两种元素，而这两种元素中的属性却常见于文本/字符串形式，而正是这种过为宽松的约束导致在实际业务场景中反而存在很多问题。为了进一步在经过拓展过后的 5 类主题分类模型之上实现更多的语法语义以实现更高效的知识传播和推导，SPG-Schema 添加一系列语义谓词对 LPG 进行约束，具体的语义语法分层图可见图 27。

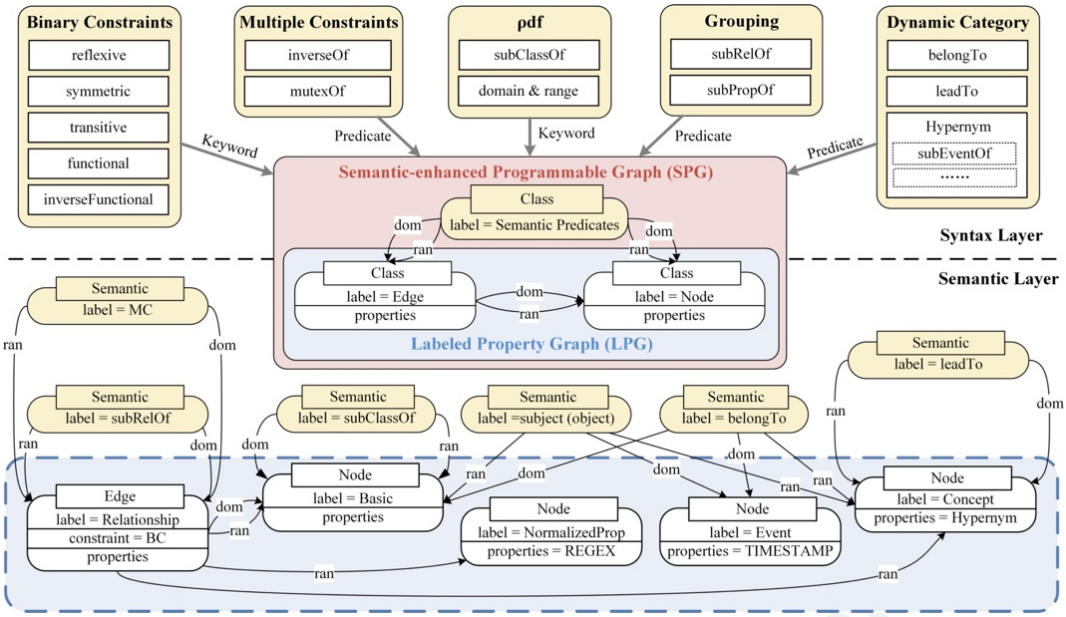

图 27 SPG-Schema 语法语义分层图

- 语法层 (Syntax Layer)：在该层定义了 SPG-Schema 中的语法内容。主要将 SPG-Schema 用于语义推理的相关语法一共可以划分为最小约束集合 (ρdf)、单二元关系约束 (Binary Constraints，BC)、多二元关系约束 (Multiple Constraints，MC)、关系分组约束以及动态类型共 5 类，适当的根据使用情景采用关键字 (Keywords)形式和标准命名空间 std 中的内置谓词 (Predicate)形式作用于 SPG-Schema 之上。
- 语义层 (Semantic Layer)：在该层定义了 SPG-Schema 定义相关语法推理能力的具体定义域 (domain，简写为 dom)和作用于 (range，简写为 ran)，提现内置推理规则谓词与细化后实体分类模型之间的关联。

4.1.3 SPG-Schema 四层架构

对于 SPG-Schema 的整体框架来说，从四个基本内容出发，逐步扩展分解需求，从语义完备性和实际工业需求中逐渐确定 SPG-Schema Core 所包含内容。尽可能采用轻量级语法，避免高复杂度，使 SPG-Schema 的复杂度不超过 PTIME，保障在工业级落地中的效率，平衡好语义复杂度和业务应用成本。

MOF 体系结构是分层的原数据体系结构，根据该结构我们可以将 SPG-Schema 整体建模层次同样划分为四层，如图 28 所示：

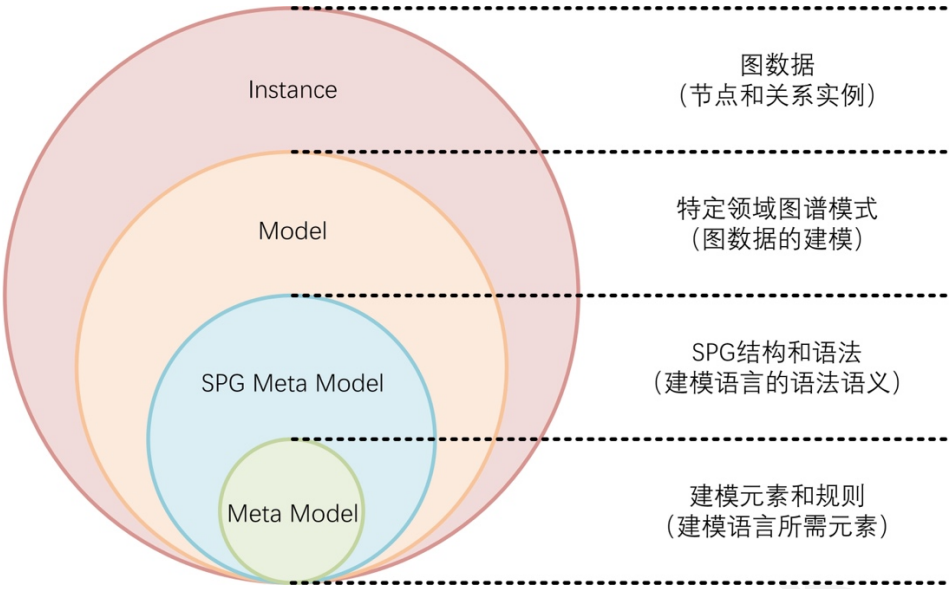

图 28 SPG-Schema 四层结构图

基于上述总结的 SPG-Schema 四层结构图，结合黑产图谱案例和事理图谱案例作为基本场景支撑，SPG-Schema 总体四层架构图如图 29 所示：

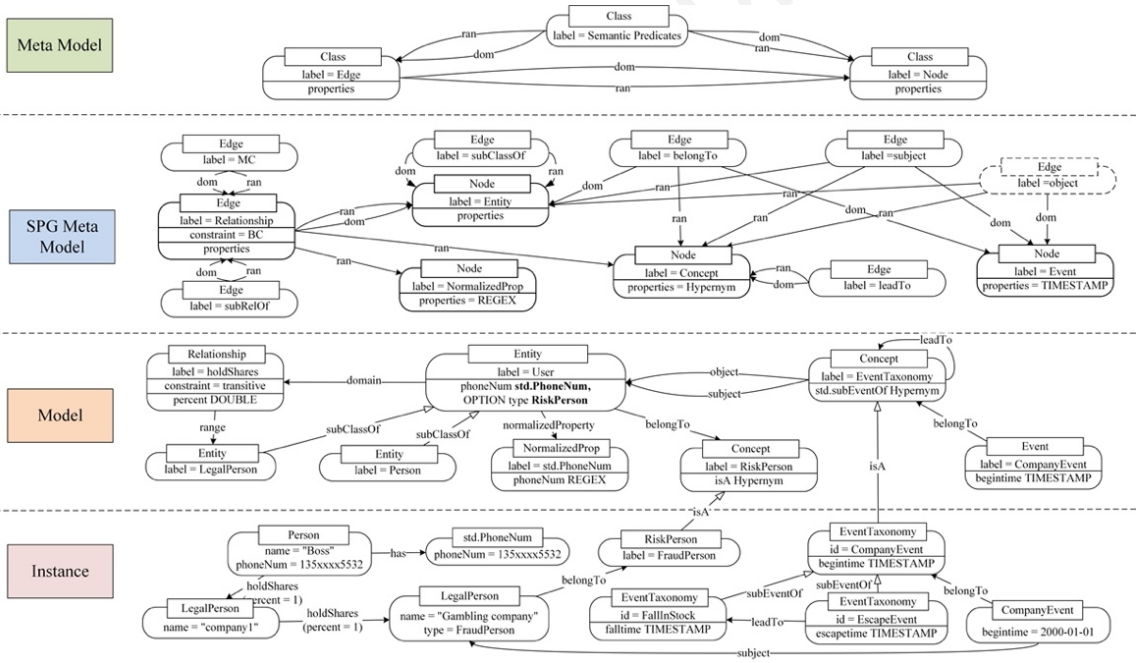

图 29 SPG-Schema 四层架构示意图

四层架构的解释为：

- Meta Model: 该层是对 SPG Meta Model 层范式的总体定义，定义了在通过 SPG-Schema 建模时所需要的元素，这层对一般用户透明。

- SPG Meta Model: 该层是对于 Meta Model 层的实例化，是对 Model 层范式的总体定义，通过额外对类型和谓词的定义进行更加细致的分类，定义相关的建模语言的结构和语法，以对 Model 层建模加以更多语义。
- Model：该层定义了一个具体的系统模型，用户可通过语法对于实例 Instance 层能感知的类、属性、关系结构及其语义进行定义。Model 层中存储的是 Instance 层的数据，是对 Instance 层面的建模，而 Instance 层的数据是 Model 层中模型的实例。
- Instance: SPG-Schema 采用的是 Instance-Class 分离范式，每个 Instance 都必须严格符合 Model 层 Schema 定义的约束。该层是数量最大、最为具像化的一层，Instance 层的内容是具体的实例。

4.2 节点和边的语义增强

SPG-Schema 的扩展方案的主体语法设计通过在 PG-TYPES[16]方案基础上引入额外的关键字来实现，因此 SPG-Schema 主体语义同样分成 NodeType，EdgeType，GraphType 三大类，分别对应于属性图 LPG 中对于节点 (Node)，关系 (Edge)和视图的定义。主体语义的基本定义位于四层架构图中 SPG Meta Model 层，通过对 LPG 节点类型进行更加细致的分类来为 Meta Model 层添加更多相关语义。

4.2.1 NodeType 节点语法语义

基于 4.1.1 节定义，我们将常用的知识图谱中的节点类型划分为了实体类型、标准类型、概念类型以及事件类型。接下来给出这四类节点类型的基本语法语义。

1. 实体类型

通过 CREATE ENTITY TYPE 的语法来定义节点类型 Class 以及在该节点类型中可以出现的标签和属性类型，可以通过如下语法创建 User 类。

```
// 定义 User 类型
CREATE ENTITY TYPE ( User {
  phoneNum std.PhoneNum,
  OPTIONAL taxonomy RiskPerson,
  OPEN
});
```

在上述例子中定义了拥有两个属性 phoneNum 和 taxonomy 的 User 类型，其中 phoneNum 的类型为 std.PhoneNum，taxonomy 的类型为 RiskPerson。OPTIONAL 关键字表示 taxonomy 这一属性可选；OPEN 关键字则表示在定义该 Model 向下填充实例数据时，可以添加额外属性字段。

在某些情况下，也可能需要将类型声明为抽象类型，即抽象类型不能直接实例化。如上述例子中 User 类型可以通过 ABSTRACT 关键字被标注为抽象实体类型，于是在填充实例数据的时候不能够直接在该类别下进行填充，而是填充到继承了该抽象父类的子类型当中。就其本身而言，这种类型可能不是很有用，但是对于多个子类的共用属性的复用会有帮助。

除上述三个关键字外，借鉴 PG-Keys[17]中关键字约束给节点定义添加更多的语义信息和约束。通过 EXCLUSIVE、MANDATORY、SINGLETON 三个关键字来分别代表唯一、至少、至多三种约束，进一步补充 SPG-Schema 对于实体属性中约束进行属性的语义增强。下列例子我们重新修改 User 类型为抽象类型，且通过三个属性约束关键字加以约束。

```
// 扩充定义 User 类型为抽象类型
CREATE ENTITY TYPE ABSTRACT ( User {
  EXCLUSIVE idcard STRING,                          // 每个实例的身份证号码应当各不相同
  MANDATORY name STRING,                            // 每个实例至少应当有一个姓名
  SINGLETON birthday DATE,                          // 每个实例最多只有一个生日
  OPTIONAL phoneNum std.PhoneNum,                   // 可选是否添加具有标准属性的 email 字段
  OPTIONAL SINGLETON taxonomy RiskPerson,           // 可选择是否需要概念分类属性
  OPEN
});
```

当三个关键词与 OPTIONAL 关键字同时作用于类型约束中时，前者用于约束该类型在实例层中填充数据时的约束，而后者则对应于约束实例是否可以使用该属性。这二者互相不冲突，例如 OPTIONAL SINGLETON type RiskPerson 则可以用来表示数据可以不包括概念分类属性 taxonomy，但一旦添加该属性，那么该实例最多只能拥有一个分类属性。

2. 标准类型

为了更好的区分开标准类型和用户自定类型，引入命名空间概念，以 std 命名空间体现标准类型。其中标准类型均定义在 std 命名空间之下，通过正则表达式创建。由于标准类型更多与属性标准化搭配使用，由属性转化而来，因此通常标准类型只存在一个属性。通过 CREATE NOMALIZED TYPE 语法定义一个标准属性后的标准属性，可以直接使用，也可以通过 SET PROP 语句和 NORMALIZED 来将某一类型下的属性进行标化。需要注意的是，标准属性需搭配 REGEX 关键字引导的正则表达式进行使用，对该属性的值的模式/格式进行约束。

```
// 定义标准类型 Email
CREATE NOMALIZED TYPE (std.Email {
  value STRING REGEX '[a-zA-Z0-9_\-\.]+@[a-zA-Z0-9_\-\.]+\.[a-zA-Z]{2,3}'
});
// 定义标准类型 phoneNum
CREATE NOMALIZED TYPE (std.PhoneNum {
  value STRING REGEX '/^(13[0-9]|14[01456879]|15[0-35-9]|16[2567]|17[0-8]|18[0-9]|19[0-35-9])\d{8}$/'
});
// 修改 User 类中 phoneNum 属性为标准属性
SET PROP (User.phoneNum) NORMALIZED std.PhoneNum;
```

标准类型也是 SPG-Schema 中用于属性标化的最重要的内容之一，与普通 LPG 属性不同，当 SPG 中某一属性的类型为标准类型的时候，若该标准类型语义上是可传播的，将会默认将该属性转换成关系处理，而该属性的值也将作为标准类型下的一个实例，因而会产生更多有意义的信息用于语义推理。在该例中，我们将 User 类型下的 phoneNum 属性修改成为事先定义好的标准属性 std.PhoneNum，除此之外，用户还可以直接在创建节点类型的时候将属性指定为标准类型。

3. 概念类型

创建概念类型时，通过使用 OPTIONAL 关键字进行标注对可选属性进行要求，其余字段默认为必选字段。通过该类型语义，可以将节点、属性等关联到相关业务概念域中，从而通过概念域进行更多业务推理。

```
// 定义公司分类概念
CREATE CONCEPT TYPE (CompanyTaxonomy {
  isA std.Hypernym,
  OPTIONAL beginTime TIMESTAMP,
  OPEN
});
```

在上例中，创建了公司分类概念 CompanyTaxonomy 作为一个概念域，后续在填充概念实例时，都将归属于该概念类型之下。其中 std.Hypernym 类型的属性 isA 为该概念域中的一个必选属性，表示该概念域中概念实例之间使用的上下位词为 isA，而 begintime 字段为可选属性。关于 std.Hypernym 类型的更多细节，我们将在 4.3.5 节进行更加细致的讨论。

4. 事件类型

创建事件类型时，需要内置对时间、主体、客体的必选要求和可选要求。其中使用关键字 REQUIRED 来表示对于字段必选要求，其余字段为可选要求。需要注意的是，事件类型默认必须拥有时间戳属性以及指定的主体类型。因此我们采用如下的语法定义事件。

```
// 定义公司运营事件
CREATE EVENT TYPE (CompanyEvent {
  {REQUIRED occurrenceTime TIMESTAMP, OPEN}
  REQUIRED SUBJECT (Company | Person),
  OBJECT (Company | Person)
});
```

在上述定义中定义公司运营事件，其中包括一个必选的 occurrenceTime 的时间戳属性，以及一个必选的主体类型为 Company 类型或是 Person 类型。而一个事件本身我们应当视为一个图结构，例如上述例子中定义的公司运营事件 CompanyEvent，其中定义了一个 Event 类型节点，而该节点通过 SUBJECT 关键字自动使用内置谓词 std.subject 指向 Company 和 Person 类型表示该事件的主体应该是公司或者人，这将是一个必选字段，而客体可以为空。

4.2.2 EdgeType 关系语法语义

EdgeType 语义指定了边类型中可以出现的标签和属性以及属性值的类型，并且指定了允许的源节点类型和目标节点类型。使用关系创建语句 CREATE EDGE TYPE 时，要求源节点类型和目标节点类型均已定义，否则将出现悬空挂载的情况，导致关系错误，这种情况将不予创建。

```
// 定义持股关系
CREATE EDGE TYPE
(Person)-[holdShares {percent DOUBLE}]->(LegalPerson);
```

很多情况下使用者不希望在使用整条关系时采用三元组的表示方式，因此我们可以设置关系别名来直接指代一条三元组关系。别名的设置主要有两种方式：直接定义，后期修改。对于刚刚定义的持股关系 holdShares 来说，前者通过 AS <> 的语句来直接在定义时指定别名，而后者通过 ALTER 关键字为忘记设置别名的关系补充别名。

```
// 定义时指定别名 holdSharesType
CREATE EDGE TYPE
(Person)-[holdShares {percent DOUBLE}]->(LegalPerson)
AS <holdSharesType>;

//使用 ALTER 对于上述持股关系设置别名
ALTER EDGE TYPE
(Person)-[holdShares {percent DOUBLE}]->(LegalPerson)
AS <holdSharesType>;
```

在边的定义中第一次出现<>用于表示边的别名。在 SPG-Schema 中，我们使用<>来快速指代一个图，对于关系来说，实质则是点边点的关系构成的一个图类型。

除了基本的定义之外，与上述提到的属性约束类似，EXCLUSIVE 关键字将同样可以作用于关系约束中，我们称为跨类型约束。由于关系定义默认每条关系实例都有且唯一的源节点和目标节点，因此不将 MANDATORY、SINGLETON 关键字用以约束关系。

```
// 作用于源节点 => 一个实体不能同时拥有多个相同关系的出边
CREATE EDGE TYPE (EXCLUSIVE Class1)-[ Type { propClause }]->(Class2);

// 作用于目标节点 => 一个实体不能同时拥有多个相同的关系
CREATE EDGE TYPE (Class1)-[ Type { propClause }]->(EXCLUSIVE Class2);
```

通过对边的源节点和目标节点加以约束，我们变能轻松实现关系数据库中的关系约束:

```
// 一对一
CREATE EDGE TYPE (EXCLUSIVE Class1)-[ Type { propClause }]->(EXCLUSIVE Class2);

// 一对多
CREATE EDGE TYPE (EXCLUSIVE Class1)-[ Type { propClause }]->(Class2);

//多对多
CREATE EDGE TYPE (Class1)-[ Type { propClause }]->(Class2);
```

除此之外，我们还额外为关系添加了一类指定的约束，称之为 Binary Constraints (BC) ，这类约束限定了某一个具体关系的全部实例个体拥有的特性，我们将在 4.3.2 章中着重介绍这类约束。

4.3 谓词及约束的语义增强

为了在 SPG-Schema 中更好的管理最小谓词和和拓展更多内置谓词，与 RDF 一样采用了命名空间机制。在使用标准类型的时候就已经使用过该机制，通过引入 std 命名空间，可以将一些内置谓词进行归类，使得现有的基础谓词保证为最小的谓词集合，后续面向不同领域知识图谱的 Schema 构建的时候，可以有针对性的添加不同领域的命名空间。除此之外，通过定义不同层面的命名空间，能够从不同层面丰富和完善 Schama Core 的语义和能力，对于不同产业环境下的落地应用，将产生更大的灵活性和可用空间。

Model 层定义用户可感知的类、属性、关系结构及其语义，由于 SPG Meta Model 层作为 Model 层的抽象，需要通过一系列约束来约定模式结构，因此为了更好的表达实体的语义，我们从最小约束集合 ρdf 出发，继 4.2 节中介绍 SPG-Schema 的主体语义后在本节着重介绍我们在 SPG Meta Model 层为 Meta Model 层面补充的内置谓词和约束语义，将 Model 层面会出现的约束分为单二元关系约束、多二元关系约束、分组规则和动态类型。目前我们讨论的谓词和约束均存放于标准空间 std 之下。

4.3.1 最小约束集合——ρdf

ρdf[18]的概念来源于 RDF 中最小的谓词集合，为了更好的适应 SPG 的需要，特地做出调整保留 subClassOf、domain 和 range 作为 SPG-Schema 的最小约束集合。

1. 实体类型层次 subClassOf

subClassOf 谓词通过定义节点类型之间的层级关系来补充类型继承的语义，在定义节点类型 Class 时，在后面跟上 SUBCLASSOF 关键字引导的类型名称即可实现继承。

```
// 定义 User 类的两个子类
CREATE ENTITY TYPE (Person {age INT, OPTIONAL father Person}) SUBCLASSOF (User);
CREATE ENTITY TYPE (LegalPerson {amount INT, legalId STRING}) SUBCLASSOF (User);
```

通过继承自抽象节点类型 User 类，除了这两个子类均会自动包含 User 类中定义的属性和约束外，Person 类型将额外包括年龄属性 age 以及一个可选的 Person 类型的父亲属性 father，而 LegalPerson 类型额外添加持股数量 amount 属性以及法人编号 legalId 属性。

需要注意的是，在使用 subClassOf 关键字时，需要保证子类节点不包含与父类节点重名的属性，否则将面临重写和覆盖的问题。这同时涵盖两种情况：名称和类型均相同；名称相同但类型不同。我们将这两种情况视为错误情况不予处理。

2. 关系定义域和值域 domain & range

在构建关系时，由于需同时指定关系的源节点类型和目标节点类型，为了减少实体转换等带来的冗余计算成本，对于 DOMAIN 和 RANGE 的修改应待采取“只增不改”的原则。例如在先前例子中构建了包括一个持股百分比 (percent)属性的持股关系 (holdShares)，但由于初始定义不当，发现存在 (LegalPerson)-[holdShares]->(LegalPerson)这类额外的需求，需要通过相应的修改令该关系值域添加 LegalPerson。

```
// 上述定义持股关系<holdSharesType>
(Person)-[holdShares {percent DOUBLE}]->(LegalPerson)
// 为其添加定义域 LegalPerson
ALTER <hold_share> DOMAIN (LegalPerson);
// 操作过后的关系实际应为
(Person | LegalPerson)-[holdShares {percent DOUBLE}]->(LegalPerson)
```

4.3.2 单二元关系约束 (Binary Constraints, BC)

在 PG-Schemas 中，对于 BC 特性有如下定义：

Binary constraints (BC), i.e., defined to be (ir)reflexive, (in)transitive, (a)cyclic, (a/anti)symmetric, etc.

该类约束更加注重的是某个二元关系其本身的性质，通常在定义关系时一起定义，因此我们将这类约束作为关键字进行定义。在 Model 层为关系定义了 BC 类约束之后，该关系对应的全部 Instance 层实体数据都应当遵循该约束。根据上述定义，SPG-Schema 中 BC 约束应当具备最基本的自反性，对称性和传递性。

现假设已经实现定义好的几个节点类型 Class1 和 Class2。我们通过 REFLEXIVE、SYMMETRIC 和 TRANSITIVE 三个关键字分别创建自反关系、对称关系和传递关系，并且给出了该定义下的语义推导。

```
// 定义自反关系 edgeA
CREATE EDGE TYPE REFLEXIVE (Class1)-[ edgeA { prop STRING }]->(Class1);
// 意味着
(a:Class1)-[p:edgeA]->(a:Class1)
---------------------------------
// 定义对称关系 edgeB
CREATE EDGE TYPE SYMMETRIC (Class1)-[ edgeB { prop STRING }]->(Class2);
// 意味着
(a:Class1)-[p:edgeB]->(b:Class2),
---------------------------------
(b:Class2)-[p:edgeB]->(a:Class1)
// 定义传递关系 edgeC
CREATE EDGE TYPE TRANSITIVE (Class1)-[ edgeC { prop STRING }]->(Class1);
// 意味着
(a:Class1)-[p1:edgeC]->(b:Class1),
(b:Class1)-[p2:edgeC]->(c:Class1)
---------------------------------
(a:Class1)-[p3:edgeC]->(c:Class1)
```

除了上述三个基本的约束（自反、对称、传递性）以外，为了更好的实现语义完备性，额外根据 OWL 语法添加函数式的关系和非函数式的关系，通过 FUNCTIONAL 和 INVERSE_FUNCTIONAL 关键字引导，其定义如下：

$$FunctionalProperty\text{:}\ \top \sqsubseteq (\leqslant 1\, r.\top)\ \ (e.g., \top \sqsubseteq (\leqslant 1\, hasMother))$$
$$InverseFuntionalProperty: \top \sqsubseteq (\leqslant 1\, Inv(r).\top)\ \ (e.g., \top \sqsubseteq (\leqslant 1\, isMotherOf^{-}))$$

```
// 定义函数式关系 edgeD
CREATE EDGE TYPE FUNCTIONAL (Class1)-[ edgeD { prop STRING }]->(Class2);
// 意味着
(a:Class1)-[p:edgeD]->(b:Class2),
(a:Class1)-[p:edgeD]->(c:Class2)
---------------------------------
 (b:Class2) = (c:Class2)
// 定义反函数式关系 edgeE
CREATE EDGE TYPE INVERSE_FUNCTIONAL (Class1)-[ edgeE { prop STRING }]->(Class2);
// 意味着
(b:Class2)-[p:edgeE]->(a:Class1),
(c:Class2)-[p:edgeE]->(a:Class1)
---------------------------------
 (b:Class2) = (c:Class2)
```

4.3.3 多二元关系约束 (Multiple Constraints, MC)

为了使 SPG-Schema 增加额外的语义推导功能，额外添加 MC 谓词支持。该类谓词更多面向于的是两个二元关系之间的关系推导，使用 SET REL 的方式进行设置，谓词部分整体采用内置命名空间 std 下的内置谓词作为该类谓词。

现假设已经存在了已定义的两个关系，别名分别为<Pred1>和<Pred2>。我们通过 std.inverseOf、std.mutexOf 这两个内置谓词分别定义关系互反和关系互斥，并且给出了该定义下的语义推导。

```
// 两个关系为互反关系
SET REL <Pred1>-[std.inverseOf]-<Pred2>;
```

inverseOf 定义的为互反关系，如果定义两个关系为互反关系，那么实质上将默认产生的为一对等价的逆关系。例如“上司”关系可以和“下属”关系互为一对逆关系，在进行推理时，可利用“上司”关系自动反向推理“下属”从而解决复杂语义推理时带来的工业问题。在上述例子中，(Class1)-[Pred1]->(Class2)和(Class2)-[Pred2]->(Class1)为一对互反的关系。

```
// 两个关系为互斥关系
SET REL <Pred1>-[std.mutexOf]-<Pred2>;
```

mutexOf 定义的互斥关系代表一个实例关系，只能从定义的两个关系类型中进行选择，即在上述语法中提到的例子，对于同一个关系实例 s，不能同时拥有 Pred1 关系和 Pred2 关系，即(Class1)-[Pred1]->(Class2)和(Class1)-[Pred2]->(Class3)只能进行二选一。

在 MC 类谓词中，第一次出现用尖括号的表示形式< >-[ ]-< >，这种方式会更加符合原本 Cypher 用户的使用习惯，<>在此处是在定义关系时定义的别名，例如<Pred1>实质上可视为三元组关系(Class1)-[Pred1]->(Class2)，这样就可以大大简化在书写多个二元关系之间关系的复杂度，让整体语法更加简洁和易懂。

4.3.4 关系分组

由于在真实案例场景中，往往需要通过一个抽象化的关系来查询一类相似的关系，因此需要引入关系分组谓词来帮助用户构建分组，以节约查询成本。整体的可以将分组视为关系分组和属性分组两个部分，由于属性在一定程度上可以视为一种关系，因此统一采用 SET REL 语法进行定义关系分组。

1. 关系分组

分类的第一个应用场景为关系分组，主要采用内置谓词 std.subRelOf 进行定义。在构建分组时，首先应当确保存在一个顶层关系，该顶层关系在使用 CREATE EDGE TYPE 语句创建时必须使用 ABSTRACT 关键字进行标注以表明该抽象分组关系下将不可拥有实例，任何装载的实体关系都应当从属于该分组下的具体关系中来。

关系分组实质上同样可以视为两个二元关系之间的关系，但又与上述 MC 谓词不同关系分组谓词跟关系类型定义在语法上有所区分。MC 关系谓词定义的两个二元关系之间为等价关系，但关系分组中两个关系会存在明显的上下位关系，因此定义的语法为有箭头的三元组形式< >-[ ]->< >。

```
// 创建亲属关系分组，使用 ABSTRAT 关键字进行标注
CREATE EDGE TYPE ABSTRACT (Person)-[kinship]->(Person) AS <kinship>;

// 定义父亲、母亲、夫妻三个亲属关系
CREATE EDGE TYPE (Person)-[isFatherOf]->(Person) AS <father>;
CREATE EDGE TYPE (Person)-[isMatherOf]->(Person) AS <mother>;
CREATE EDGE TYPE (Person)-[conjugality]->(Person) AS <conjugality>;

// 定义关系分组
SET REL <father>-[std.subRelOf]-><kinship>;
SET REL <mother>-[std.subRelOf]-><kinship>;
SET REL <conjugality>-[std.subRelOf]-><kinship>;
```

通过以上定义，我们将父子关系、母子关系和夫妻关系均视为家庭关系的某一种具体关系，可以通过 kinship 关系直接获取到该分组关系下的这三个关系。而在装载实例时，将不会存在任意一个三元组属于家庭关系，而是应当从属于父子关系、母子关系和夫妻关系三种关系中的具体一种。

2. 属性分组

属性在进行属性标化之后，也可以将属性视为一种具体的关系可以使用 subRelOf 进行分组。但是这样的需求显然不符合一个完整的完备的语义定义，若是对于非标化属性来说，依旧会存在属性分组的需求。

因此属性分组的语法定义类似于 subRelOf，属于对属性的分组，但是二者的区别在于顶层分组的属性可能会存在自身的实例数据，并且属性都是装载到实体当中。因此无需提前通过 ABSTRACT 定义抽象的顶层属性，并且采用 (类型.属性) 的模式来代替<>所表示的关系，从而获取指定类型下的指定属性。

```
// Person 类型中对交易聚合值分组
SET REL (Person.1_day_complaint_rate)-[std.subPropOf]->(Person.day_complaint_rate);
SET REL (Person.7_day_complaint_rate)-[std.subPropOf]->(Person.day_complaint_rate)
```

4.3.5 动态类型

1. 事件概念上下位谓词 Hypernym

由于概念领域存在多样性，不同的概念域中会采用不同的上下位词，因此我们支持在定义事件的同时通过"Hypernym" 来表达一类上下位谓词，从而支持在不同概念域上对事件的概念层级进行划分。该类谓词通过在定义概念类型时候同时定义，例如在风险人员分类概念中，我们将上位词设置为 isA，而可以在城市分类概念 CityTaxonomy 中可以设置上位词为 locateAt。

```
// 定义风险人员分类概念
CREATE CONCEPT TYPE (RiskPerson {
  isA std.Hypernym,
  OPEN
});

// 定义城市分类概念
CREATE CONCEPT TYPE (CityTaxonomy {
  locateAt std.Hypernym
});
```

通过这样定义，风险人员分类概念中可以存在概念实例：“赌徒”isA“风险人员”，而城市分类概念中，可以存在：“成都”locateAt “四川”，“四川” locateAt “中国”。

除此之外，事件类型是 Schema 中比较特殊的一类节点类型，其本质是一个图并且大多会关联事件分类概念，因此有必要对于事件类型概念使用特殊的谓词来达到对事件类型的约束。std.subEventOf 则是 Hypernym 谓词中专用于事件概念分层的上下位谓词，在定义事件概念类型时必须采用 std.subEventOf 作为上下谓词。

```
// 定义公司运营事件概念
CREATE CONCEPT TYPE (CompanyOperationTaxonomy {
   std.subEventOf std.Hypernym,
   OPTIONAL beginTime TIMESTAMP,
OPEN
 });

// 实例层有：高管出逃概念从属于公司运营概念
<EscacapeEvent:CompanyOperationTaxonomy>-[std.subEventOf]-><CompanyEvent:CompanyOperationTaxonomy>;

// 实例层有：股价下跌事件从属于公司运营事件概念
<FallInStock:CompanyOperationTaxonomy>-[std.subEventOf]-><CompanyEvent:CompanyOperationTaxonomy>;
```

该谓词作用于 Instance 层实例化的事件概念上。并且通过事件概念分层之后，子事件的主客体必须为父事件主客体类型的子类，并且可以拥有父亲事件之外的其他属性。例如上述例子中指定“高管出逃概念 EscacapeEvent”和“股价下跌事件 FallInStock”都从属于“公司运营事件概念 CompanyEvent”。

2. 动态类型谓词 belongTo

动态类型是指当实体实例或者事件实例和具体概念实例关联时，可以将概念实例的名字作为该实例的类型。我们主要通过 SET REL 语法和 belongTo 关键字来指定具体的实体实例（包括实体类型实例和事件类型实例）及其从属的具体概念实例。

```
// 基本实体类型实例会从属于风险人员分类概念
SET REL (User)-[std.belongTo]-><RiskPerson>;
```

首先可以让实体类型与概念相关联。在前文中我们定义了风险人员分类概念 RiskPerson 和 User 实体类型，通过 std.belongTo 关键字可以将 User 类型关联到风险人员分类概念域中。

```
// 公司运营事件定义
CREATE EVENT TYPE (CompanyEvent {
  {REQUIRED begintime TIMESTAMP}
  SUBJECT (Company | Person),
  OBJECT (Company | Person)
});
// 公司运营事件实例会从属于公司运行事件概念
SET REL <CompanyEvent>-[std.belongTo]-><CompanyOperationTaxonomy>;
```

根据前文定义的公司运营事件概念分类 CompanyOperationTaxonomy，将新定义的公司运营事件类型 CompanyEvent 通过 std.belongTo 关键字进行关联，从而定义公司运营事件中实例可以归属于公司运营事件分类概念下的具体公司运营事件概念实例。

3. 概念推导谓词 leadTo

leadTo 与 belongTo 不同，从事件概念域上提现因果关系，利用该规则会更多的添加自动推理事件类型中的相关关系。但二者同样采用相同的语法，我们通过重新定义高管出逃事件类型和股价下跌事件类型来进一步描述 SPG-Schema 通过 leadTo 来进行语义推导。

```
// 定义某一公司运营事件会导致另一公司运营事件
SET REL <CompanyOperationTaxonomy>-[std.leadTo]-><CompanyOperationTaxonomy>;
// 若 A 公司高管出逃事件实例 a 从属于高管出逃事件概念
// 且有高管出逃事件概念实例导致股价下跌事件实例
<a:EscapeEvent>-[std.belongTo]-><EscapeEvent:CompanyOperationTaxonomy>;
<EscapeEvent:CompanyOperationTaxonomy>-[std.leadTo]-><FallInStock:CompanyOperationTaxonomy>
---------------------------------------------------------------------
// 自动产生 FallInStock 事件实例 b,关联至 A 公司股价下跌事件
<b:FallInStock>-[std.belongTo]-><FallInStock:CompanyOperationTaxonomy>;
```

通过std.leadTo定义的关系仅表示建模层面的关联，但作用于Instance层实例化的事件概念上。例如上述例子中出现的实例：高管出逃事件 会导致 股价下跌，是属于事理上的定义，当发生了一个事件 A 公司高管出逃，该事件被 belongTo 了到概念域中高管出逃事件，那么可直接产生一个新的事件实例，A 公司股价下跌。

4.4 规则定义的语义增强

为了更好的形成完备的语义体系，除了基本的谓词和约束之外，还应当引入规则定义部分进行补充。但规则定义大多不是靠语法定义，实现需要放于 Programming 中实现，因此在本节仅介绍规则定义的语法并提供相关例子以作参考。

4.4.1 自定义关系属性规则

自定义关系属性规则是业务中常见的一类需求，当满足一定条件之后才会产生某一个具体的关系。因此我们通过 RULE 关键字进一步扩充 EdgeType 语法来为规则定义部分留出空间。由于在规则定义时通常会作用于某一个具体的实例中，因此在表达关系三元组的时候，可以通过(instance:Class)的形式来指代，但在创建关系的时候，如果没有 RULE 引导的规则块，并不会产生额外的影响。

```
// 定义单链路上的控股比例(递归定义）
CREATE EDGE TYPE
(s:Person)-[p:hold_share_rate]->(o:LegalPerson)
RULE {
  STRUCTURE {
    (s)-[p1:hold_share_rate]->(c:LegalPerson),
    (c)-[p2:hold_share_rate]->(o)
   },
CONSTRAINT {
            real_rate("相乘得到实际持股比例") = p1.real_hold_share_rate*p2.real_hold_share_rate
            p.real_hold_share_rate = real_rate
  }
};
```

常见的规则可以分为结构规则和约束条件规则，这两部分规则将在规则块中分别使用 Structure 和 Constraint 引导的子规则块进行区分，前者是由三元组关系构成的实例图结构，后者是规则代码。

4.4.2 条件互反规则

```
// 定义关系——证书被法人有效持有
CREATE EDGE TYPE (Cert)-[effect_owned_by]->(LegalPerson) AS <effect_owned_by>;
// 实例当证书有效时才存在互反关系——法人拥有证书
CREATE EDGE TYPE (o:LegalPerson)-[p2:has]->(s:Cert) CONDINVERSEROF <effect_owned_by>
RULE {
  STRUCTURE {
  },
  CONSTRAINT {
    s.is_effect == true
  }
} AS <has>;
```

条件逆关系整体采用 CONDINVERSEOF 结合 RULE 关键字进行引导，通过花括号引导若干条件规则语句，只有在满足条件规则的情况下可以正确将两个关系视为逆关系，当条件规则置空时，将默认与条件互反是等价的。在上述例子中，首先定义了基本的一条关系<effect_owned_by>，通过关系互反，定义只有当满足证书还在有效期的时候，才会产生另一条"法人拥有证书"的关系，并使用别名<has>标注。

4.5 SPG-Schema 与 PG-Schemas 的关系

PG-Schemas[16]的目的是弥补属性图数据库管理的不足和现有系统对模式支持能力的缺失，增强类型定义和提高数据完整性约束，以提供更灵活的类型管理，支持一定程度的类型继承和复用。通过 PG-Keys[17]的形式化定义，实体、关系和属性可以在属性图上进行形式化表达，通过建立一个灵活而强大的定义关键约束的框架增强了 Schema 不同要素之间的逻辑关联和一致性，但图构建和使用的复杂度并没有降低，用户仍需准备大量数据工作。并且，用 PG-Schemas 直接表示 SPG 还存在以下问题：

- 类别缺失业务语义。PG-Schemas 中允许用户根据意图通过操作符&（和）和|（或）动态组合节点类型，这提供更加丰富的类型表达能力，但无法有效体现类型的业务语义，包括类型内部语义结构及类型间语义表示，并且类标签之间的层次关系模糊也导致在实际业务落地中结构控制困难。
- 缺乏逻辑依赖支持。PG-Keys 定义了一个定义关键约束的框架作为全局约束来加强属性图数据完整性，而对于特定查询语言的验证和维护的复杂性，蕴涵和推理问题却仍需进一步探索。SPG 知识管理希望实现逻辑规则与事实知识的有机融合，刻画知识之间的逻辑依赖，构建知识的分层衍生机制，减少无效重复构建，保障逻辑一致性。
- 支持特性部分缺失。PG-Schemas 文章中提到，在现阶段的版本中，尚未完全支持二元约束(Binary Constraints，BC)以及 introspection (IS)特性，并且通过作者对比实验发现，RDFS、SHACL、ShEx 等工作中同样缺乏 BC 特性。

总体而言，PG-Schemas 的本质还是数据库管理，增强类型定义、强化 Key 约束等，如第 1/2/3 章所述，SPG 面向知识的逻辑依赖、知识分层、知识构建、可编程等方面构建知识管理能力，和 PG-Schemas 面向数据库做能力增强是两个不同的视角，可以做较好的互通和增强，SPG 的目标是降低用户使用图谱门槛，减少用户对图谱知识生产的介入，PG-Schemas 可以提供全局一致性的校验，该特点可以应用到 SPG 生产出的知识图谱上。结合 PG-Schemas 的官方描述，SPG 和 PG-Schemas 的关系如图 30 所示：

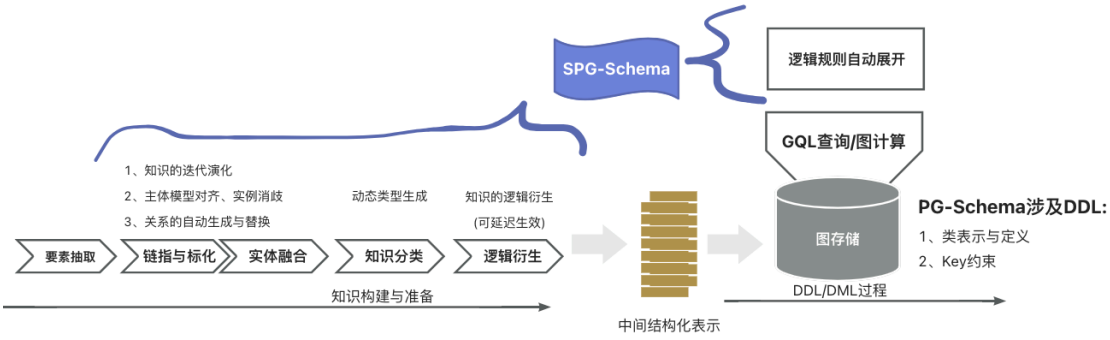

图 30 SPG 与 PG-Schemas 的关系

4.6 SPG-Schema 总结

本章对 SPG-Schema 的整体架构及语法语义设计展开描述，并详细说明主体模型基于一般属性图做出的拓展细节。当前版本重点描述 SPG DC 针对业务需求而衍生出的概念、事件及标准类型三类主体分类模型，并针对扩充后的分类模型设计和形成了用于 SPG Reasoning 语义增强可推理的逻辑语法和相关谓词。在后续的版本中会对当前语法的语义完备性进一步严格证明，以形成一套易用的完备语法语义体系，更详细的谓词逻辑语义及语法介绍，会陆续以连载文章形式发布在 SPG 技术社区和公众号。

蚂蚁集团 x OpenKG

第 5 章 SPG-Engine 层

本章主要聚焦在 SPG 语法实际执行过程的实现，我们称之为 SPG-Engine 层。SPG-Engine 层是将 SPG 的推理和计算转换到实际的 LPG 系统中执行的模块。SPG 底层依赖通常包括图存储、图查询、图计算等基础能力，这样的底层能力通常是由 LPG 的图服务厂商提供的。本章描述了 SPG-Engine 层的整体架构，按照 SPG 语义下的图模型定义、图数据导入、图查询和计算等功能模块进行划分，分别给出了如何对接到底层 LPG 处理系统的方式。

5.1 SPG-Engine 架构

SPG-Engine 层是将 SPG 的推理和计算转换到实际的 LPG 系统中执行的模块。SPG 底层依赖通常包括图存储、图查询、图计算等基础能力，由 LPG 的图服务厂商提供。为了满足基于 SPG 的知识图谱推理和服务能力的要求，我们将对引擎能力的要求分为基础能力和进阶能力。

GQL[19]是属性图查询语言的 ISO 国际标准，将于 2024 年发布。它定义了基于属性图的查询语言规范，并兼容了弱类型的 Label 和强类型的 Type 方式。SPG 方案不对底层的图服务使用 Label 还是 Type 进行限制，只要能实现 SPG-Engine LPG Adapter 的接口即可接入 SPG 引擎。SPG-Engine LPG Adapter 提供了第三方属性图系统接入 SPG 系统的方式，可以使用 GQL 语言或自定义函数/过程进行实现，也可以使用 HTAP[20]图数据库系统单一实现或 OLTP 的图数据库系统结合 OLAP 的图计算系统组合实现。

结合图 24 所示架构图，SPG Engine 各模块详细功能如图 31 所示：

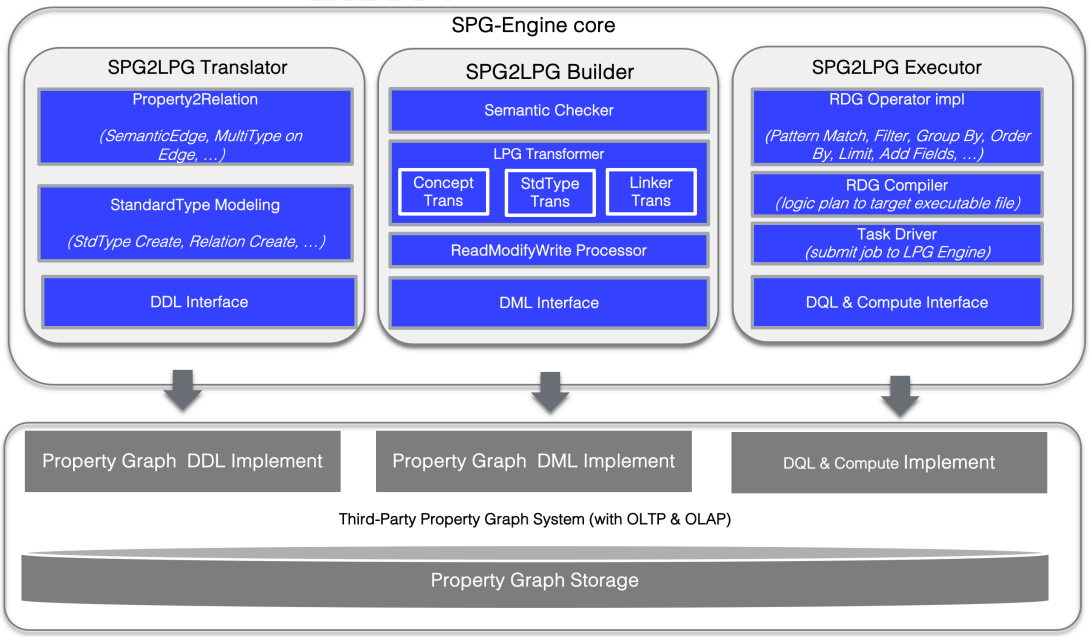

图 31 SPG-Engine 的整体架构

SPG-Engine Core 层是实现 SPG 和 LPG 相互转换的功能模块，以依赖包形式运行在 SPG-Controller 进程内。第三方属性图系统作为独立的服务进程，承载实际的 SPG 数据存储、查询和计算任务，通过 DDL 接口、DML 接口、查询和计算接口与 SPG-Controller 中的 SPG2LPG Translator、SPG2LPG Builder、SPG2LPG Executor 进行对接。第三方属性图系统需要满足 SPG-Engine 规范中的基本要求，并实现 SPG-Engine LPG Adapter 的对接接口。对于进阶要求，应以配置文件方式描述，未能实现的应提供空的接口实现。

作为支持核心功能的模块，SPG-Engine 的性能和弹性部署能力至关重要。性能优异的第三方属性图系统不仅可以处理大量数据，还可以确保系统稳定性和响应速度。弹性部署能力使系统更加灵活适应各种应用场景和需求变化，提高用户满意度和业务适应性。我们可以采取压力测试、基准测试和实际业务场景模拟等方法评估性能和弹性部署能力。在当前 SPG 1.0 版本中，我们更关注功能对接和实现，确保核心功能得到完整体现。在未来版本中，我们将完善和强化性能和弹性部署能力描述和实现，并进行更严格的评估以满足用户期望。

5.2 SPG2LPG Translator

SPG-Schema 章节描述了 SPG-Schema 和 LPG-Schema 关系，如第 4 章图 29 所示。SPG 在 LPG 基础上增加语义谓词、类型扩展、逻辑规则等，本章节需将 Semantic Layer 所表示 Schema 转换对应到 LPG 引擎 Schema。SPG Meta Model 可以和 Meta Model 进行互相转换，参考 Schema 模型分层。SPG2LPG Translator 主要负责将 SPG 的 Schema 转换为 LPG 的 Schema 格式。SPG Schema 与 LPG Schema 最大的差异之一在于属性类型，转换框架需要将 SPG 的语义化属性类型转换为 LPG 文本/数字数据类型，并生成对应的关系。同时，语义约束也需要进行翻译，如继承、动态类型、子属性等。此外，SPG Schema 内置的标准属性类型也需要转换成单独的实体类型并添加约束，并生成对应的关系。SPG2LPG Translator 根据 SPG Meta Model 到 Meta Model 的映射关系，主要分为三层：

- Property2Relation 层，将属性转换为属性图中的边，包括标准属性、概念和事件等。
- StandardType Modeling 层，将标准属性、概念和事件转换成对应的点模型。
- DDL 接口层，将 1、2 层的转换内容映射到 DDL 接口上，需要底层属性图支持该能力。

表 7 展示了为完成 SPG Meta Model 到 LPG Meta Model 的翻译转换所需的类型/关系映射。

表 7 SPG Meta Model 到 LPG Meta Model 的翻译转换

SPG Meta Model	作用	LPG Meta Model
Class	实体类型	Node
Concept	概念	Node
NormalizedProp	标准属性	Node
Event	事件	Node
leadTo	概念间的因果关系谓词	Edge
hypernym	概念间的上下位谓词	Edge
object	事件的客体	Edge
subject	事件的主体	Edge
subClassOf	实体类型的层级关系	Edge
RelationShip	关系类型	Edge
subRelOf	关系类型间的层级关系	Edge
MC	关系的多元约束	Edge

SPG Schema 中定义实体类型/概念类型支持继承，定义关系支持反向边，这些定义都需要进行翻译。以图 33，图 34 举例说明。

1. 实体类型的语义转换

对于通过 subClassOf（继承）谓词定义的实体类型，需要先读取父类的属性，再合并上子类的属性，以此作为子类的属性集合。需要注意的是，父类属性和子类属性的名称不能重复，这是对 subClassOf 的约束。此外，所有实体类型的顶层父类都从根类型 Thing 继承而来。Thing 类型包含三个基本属性：主键 ID、实体名称和描述。

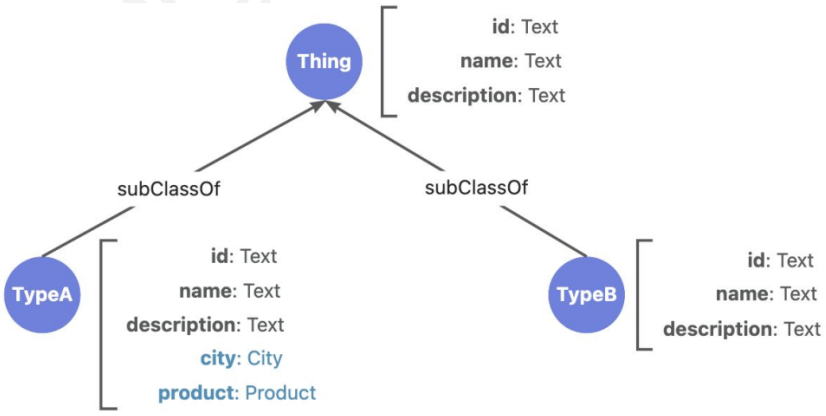

图 32 subClassOf 语义示意图

2. 属性类型为实体类型/概念类型的转换

当 SPG Schema 的属性为实体类型或概念类型时，需要进行以下转换行为：

- 将属性的类型翻译为文本类型。

- 新增一个名为 rawOf+propertyName 的文本类型属性，用于保存属性的原始值。
- 添加一个从当前实体类型指向目标实体类型或概念类型的关系，关系名称与属性名称保持一致。
- 如果属性上还有子属性，需要将子属性同步创建到关系上，作为关系的属性存在。

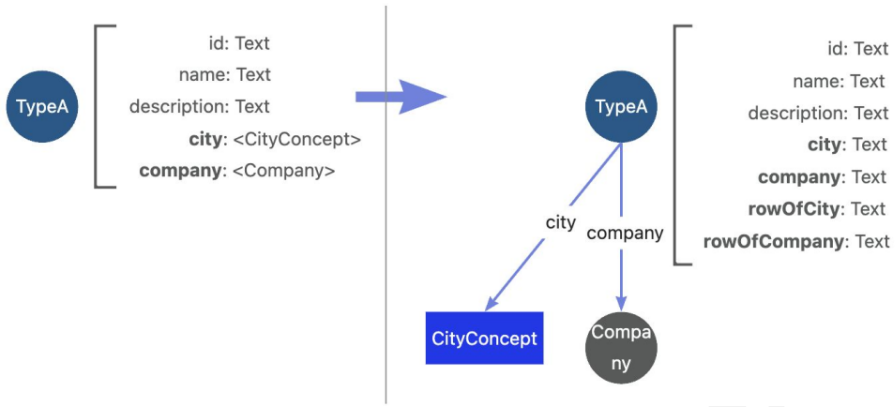

图 33 SPG 语义化属性到 LPG 的无损冗余适配过程

5.3 SPG2LPG Builder

本模块主要解决将 SPG 格式的数据转换为 LPG 格式的数据时，针对实体数据变更可能产生的一条或多条实体/关系的新增/删除操作进行处理。数据变更包括导入实体、删除实体、导入关系、删除关系、导入概念和删除概念等。SPG Meta Model 转换成 LPG Model 的 Node 和 Edge，整体子模块如图 34 所示，它包括三部分：

- Semantic Checker，该层为语义检查模块，检查输入内容是否满足 SPG 定义的约束规范
- LPG Transformer，该层将 SPG 数据具体转换映射为实际属性图存储模式，下面会针对这功能详细转换详细介绍
- ReadModifyWriter Processor，该层为读写同步层，保证写入一致性。

从 SPG 到 LPG 的转换适配过程，有实体转换、关系转换、概念转换等。其中，实体转换对 DML 的要求有：新增或者更新节点 (UpsertNode/UpsertVertex)、删除节点 (DeleteNode/DeleteVertex)、新增关系(AddEdge)、删除关系 (DeleteEdge)、查询关系 (GetEdge)等。首先，判断实体属性值是否符合相应的类型定义，查询当前实体的属性名对应的关系，再删除查询出来的关系：

- 当属性类型为实体类型时，将原始值写入 raw 属性，如果策略是 ID 相等，直接从生成一条从当前实体到属性值所示 ID 的实体的边，如果策略是算子，则运行算子逻辑后再生成一条从当前实体到链指结果 ID 的实体的边。
- 当属性类型为概念类型时，原始值写入 raw 属性，生成一条从当前实体到属性指定的概念类型的 ID 为属性值的边。

- 当属性类型为标准类型时，原始值写入 raw 属性，新增一个 ID 为属性值的标准类型实体，生成一条从当前实体到指向上一步生成的实体的边。

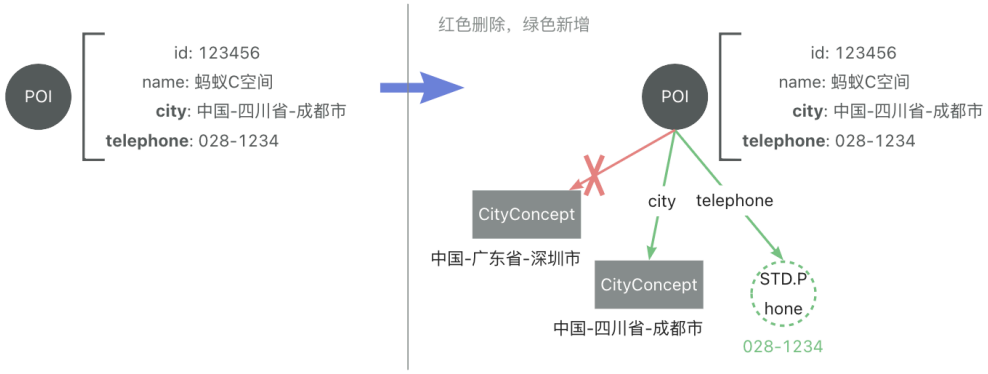

图 34 SPG 实体实例变更时 LPG 子图事务更新过程

关系转换对 LPG DML 的要求有：新增/更新节点 (UpsertNode/UpsertVertex)、新增关系 (AddEdge)、删除关系 (DeleteEdge)、查询节点 (GetNode)、查询关系 (GetEdge)等，转换逻辑如图 37 所示，检查关系的变更是否符合语义约束，新增/删除关系后，需同步将关系等价的属性上的值进行更新。

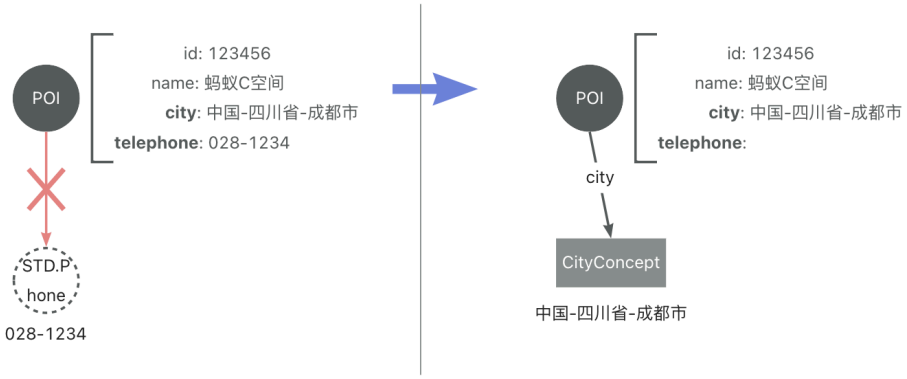

图 35 实体实例更新时 SPG 语义约束检查

5.4 SPG2LPG Executor

SPG2LPG Executor 主要执行由 SPG-Reasoner 下发的基于 RDG 算子 (Resilient Distributed Graph, RDG)组成的执行计划，RDG 模型设计思路来源于 Spark 中 RDD[21]。如同 RDD 思路，RDD 通过抽象出 Map、Filter、ReduceByKey 等算子操作，简化了原始 MapReduce 数据操作表达复杂性的问题。数据操作问题同样在图上存在，故抽象出 RDG 模型，将需要的对图操作转换为算子操作，以表达复杂计算过程表达，执行计划组织方式同为树状结构，按照后序遍历方式一次执行树上算子。为实现如上目标，整个 SPG2LPG Executor 分为三个部分，每个部分作用如下

- RDG Operator Impl，RDG 模型算子实现层，依据底层的 LPG 引擎分别实现各个算子定义的功能，例如 Pattern Match、Filter 等

- RDG Compiler，RDG 编译器，将 SPGReasoner 下发的执行计划转换成底层 LPG 可执行的二进制文件
- Task Driver，将 RDG Compiler 转换成的二进制文件提交到 LPG Engine 执行，该模块需要和具体引擎接口对接

1. 执行计划生成

执行计划表达的为数据处理过程，以判断用户是否为一个多设备用户为例，KGDSL 规则表达如下。

```
Define (s:Person)-[p:belongTo]->(o:UserClass/ManyDeviceUser) {
  Structure {
    (s)-[t:has]->(u:Device)
  }
  Constraint {
    has_device_num("持有设备数目") = group(s).count(u.id)
    R1("持有设备超过 100 个"): has_device_num > 100
    R2("年龄大于 18 岁"): s.age > 18
  }
}
```

经过 SPG-Reasoner 转换后，形成如下算子树。

```
└──DDL(ddlOp=Set(AddPredicate(PredicateElement(belongTo,p,(s:Person),EntityElement(ManyDeviceUser,UserClass))
  └──Filter(rule=LogicRule(R2,"年龄大于 18 岁",BinaryOpExpr(name=BGreaterThan)))
    └──Filter(rule=LogicRule(R1,"持有设备超过 100 个",BinaryOpExpr(name=BGreaterThan)))
      └──GroupByAndAgg(group=Set(NodeVar(s,null))
        └──PatternMatch(pattern=PartialGraphPattern(s,Map(s -> (s:Person), u -> (u:Device)),Map(s -> Set((s)->[t:has]-
```

算子树以 PatternMatch 节点开始执行，到 DDL 节点结束，树中每一个节点为一个 RDG 算子。

2. RDG 算子

在 1）执行计划执行中讲解了 KGDSL 执行算子编排顺序，本节主要介绍 RDG 中对算子定义，下表为算子列表。

表 8 RDG Operator 列表一览

编号	算子名	算子作用
1	patternMatch	给定一个子图模式和起点实例，获得该起点的子图实例（RDG），当不满足时返回null
2	expandInto	从已获取的子图实例（RDG）中，挑选一个实体实例作为起点实例
3	filter	给定一个判断表达式，判断当前获取的子图实例是否满足，若不满足，则该子图实例丢弃
4	groupByAndAgg	聚合计算操作，针对已有子图实例，进行聚合统计等行为计算，例如统计某个点的邻居数目
5	orderByAndLimit	排序并截断操作
6	limit	不排序，截断操作
7	shuffleAndFilter	根据不同点类型视角拆分和聚合子图数据
8	addFields	Rule中计算的临时变量存储
9	dropFields	移除无用的临时变量
10	join	将不同的子图实例（RDG）合并在一起
11	linkedExpand	调用三方服务做链指
12	ddl	实现对点、边的定义
13	select	输出指定的子图结果
14	cache	缓存当前RDG，为内部计算时使用算子

3. 生成可执行代码

RDG 算子表达的是对一个 RDG 的原子操作，将 RDG 算子树转换成底层引擎可执行代码，还需要配合执行计划树经过 Execution Plan Generator 生成。如图 36 所示。

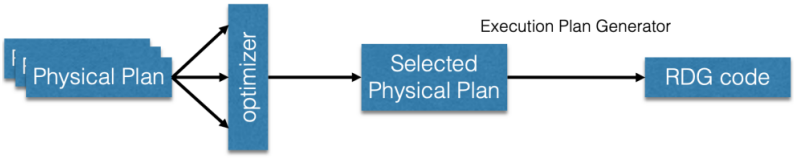

图 36 可执行 code 流程示意图

Compiler 将 Physical Plan 及 RDG Operator 生成可执行代码，Operator 节点伪码示意如下。

```
abstract class PhysicalOperator[T <: RDG[T]: TypeTag]
extends AbstractTreeNode[PhysicalOperator[T]] {
 /**
  * The context during physical planner executing
  * @return
  */
 implicit def context: PhysicalPlannerContext[T] = children.head.context
 /**
  * The output of the current operator
  * @return
  */
 def RDG: T = children.head.RDG
 /**
  * The meta of the output of the current output
```

```
   * @return
   */
  def meta: List[Var]
}
```

以 RDG 算子为节点形成树状结构，按照后序遍历执行，例如 PatternMatch 算子。

```
final case class PatternMatch[T <: RDG[T]: TypeTag](
    in: PhysicalOperator[T],
    pattern: Pattern,
    meta: List[Var])
  extends PhysicalOperator[T] {
  override def rdg: T = in.rdg.patternMatch(pattern)
}
```

输入为子节点的 RDG，调用完成 patternMatch 后返回新的 RDG 数据，以 1 中例子为例，会生成如下代码（Neo4j 客户端为例）。

```
(new Neo4jRDG(driver)).patternMatch(pattern)
                      .GroupByAndAgg(s, COUNT)
                      .filter(expr("持有设备超过 100 个"))
                      .filter("年龄大于 18 岁")
                      .ddl(new AddPredicate("s", "o", "belongTo"))
```

4. 任务执行

根据 LPG 引擎接口类型不同，分为如下两种不同场景。

- 场景 1：LPG 提供类似 Cypher[22]等 DQL 查询语言
- 场景 2：LPG 提供类似 Spark RDD 等计算编程框架，可通过嵌入用户自定义算子实现计算

针对场景 1，生成的可执行文件为 DQL，与 LPG 查询引擎通信完成查询和修改目标。

针对场景 2，可将 4.3 章节生成代码打包成为一个插件，提交到 LPG 引擎中。如图 37 流程。

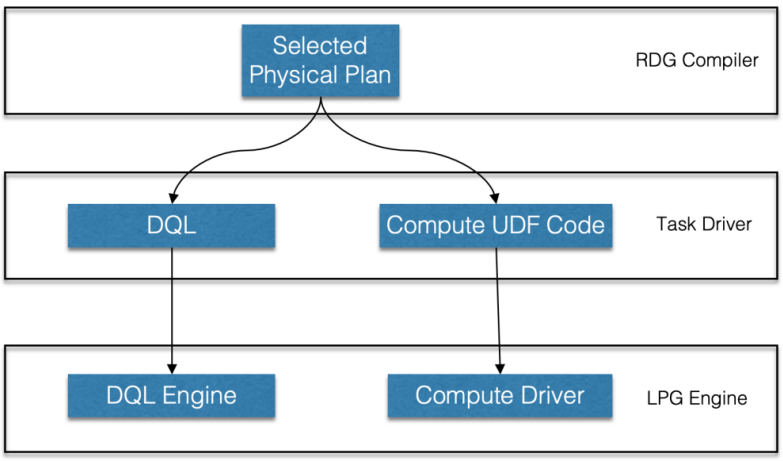

图 37 Task driver 流程示意图

5.5 SPG-Engine 对属性图系统的基础要求

第三方属性图系统，为实际承载 SPG 数据的存储、查询、计算等任务的系统。该系统以属性图为数据模型，即使用节点、边和属性来表示和存储数据，提供图结构语义的查询和计算能力。在对于图中的点和边进行分类时，该系统可以使用弱类型的 Label，也可以使用强类型的 Type，在这里统称为 Labelled Property Graph，即 LPG。

第三方属性图系统是一个独立的服务进程，应支持分布式部署，具备独立的集群安装、部署、管理、监控、运维方式，以提供基于 web 的 ui 界面为佳。该图系统通过一组适配接口和 SPG-Controller 所在进程进行交互，这组适配接口即为 SPG-Engine LPG Adapter。

第三方属性图系统既需要包括图数据的存储和查询，也需要包括在图数据上的分析和计算。通常来说可以有两种实现方式：使用单一的具备 HTAP 能力的底层系统，或者分别使用不同的 TP 和 AP 系统合起来满足要求。不论使用哪种实现方式，都应实现 SPG 对第三方属性图系统的基础要求、宜实现 SPG 对第三方属性图系统的进阶要求，并按照 SPG-Engine Core 中所描述的接口规范完成适配。

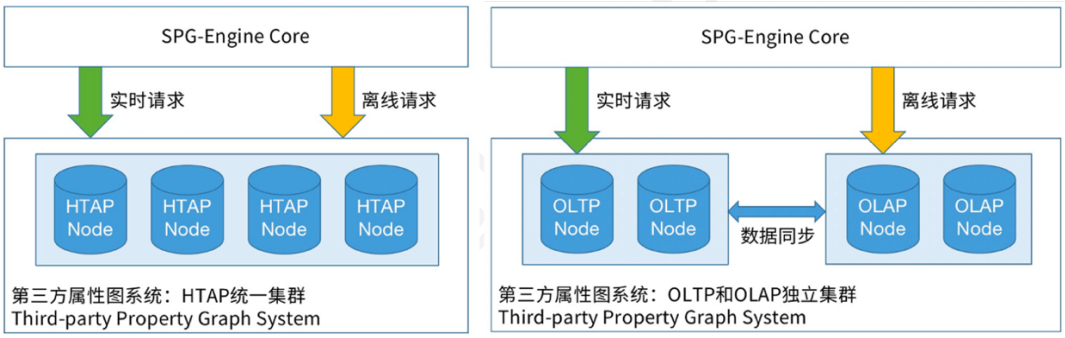

图 38 单一 HTAP 系统及不同 TP/AP 系统的接入架构图

基础要求能力一览如表 9 所示。

表 9 第三方属性图系统基础功能能力要求

能力模型	子能力	详细描述
图模型	支持属性图模型	提供点（Vertex）、边（Edge）、属性（Property）的存储和查询，点、边具备类型（Type）或者标签（Label）
	支持边特性	有向边和无向边，支持两点间存在同类型的多条边
	支持属性类型	支持基本属性类型定义：整型、浮点型、字符串等基本类型 支持复杂属性类型定义：高精度数值（BigDecimal）、时间戳（DateTime）、地理坐标（GeographicPoint）等复杂属性类型 支持容器属性类型定义：列表（List）、集合（Set）、映射（Map）、属性组（PropertyGroup）等
	支持图隔离要求	在同一个图服务中支持多个图的存储、查询、计算
图数据导入	数据导入方式	支持全量数据、增量数据批量导入；支持流式导入
	数据来源支持	支持从现有关系库中，通过JDBC连接进行导入，可自定义用于导入的SELECT语句； 支持常规文件数据源，如CSV格式文件、JSON格式文件 支持常见大数据存储文件格式，如HDFS上的Parquet格式文件
	映射方式	支持从单一原始表映射多个实体和关系的能力
	导入约束	进行批量导入时无须停止图引擎服务
属性索引	支持常见类型索引	对常见属性类型（整型、浮点型、字符串型、时间戳型等）进行索引
	支持组合索引	具备对多个属性字段进行组合索引
	支持模糊索引	对字符串型属性创建和使用全文索引
事务要求	支持事务的ACID特性	提供不低于读已提交（Read Committed）的隔离级别，应用层通过显式加锁的方式保证执行顺序的可序列化（Serializable）隔离级别
图查询语言	具备DQL能力	支持标准化查询语言GQL或者支持提供DQL查询能力
部署模式	支持灵活部署模式	对于较小的数据量，支持主备模式的部署方式；对于较大的数据量，支持分布式分片（Sharding）的部署方式

5.6 SPG-Engine 对属性图系统的高级要求

5.6.1 系统能力

- 支持触发器 (Trigger)，触发器是一种特殊类型的存储过程，它在数据库中设置了一个自动化的事件响应机制。当特定的数据库操作（例如插入、更新或删除）发生时，Trigger 会自动触发并执行预定义的代码或 GQL / Cypher 语句。
- 支持用户自定义函数/过程/算法，自定义函数/过程/算法是一种扩展图查询的机制，它允许用户通过自定义的方式来扩展数据库的功能，并且可以直接访问数据库内部 API。
- 支持时序数据的存储和查询，通过时间戳字段判定点或边的有效性，通过时间戳判定属性值的有效性。
- 多层级图谱的能力，通过一个甚至多个物理图根据业务逻辑映射出多个逻辑图：支持实体到实体的映射、支持属性到实体的映射、支持属性到属性的映射、支持直接在逻辑图上使用 GQL/ Cypher 进行查询等
- 跨图的实体/关系类型映射和转换，支持多个图（包括物理图和逻辑图）的联合查询。

5.6.2 语义功能

- 持特定属性转边，通过 Trigger 并指定存储方式（内存缓存或物理落盘），进行自动的插入和更新。

表 10 属性转边能力

需求内容	SPG-Engine能力	功能简介
标准属性	属性转边	特定属性创建时，转换为边
概念属性	属性转为多跳边	特定属性转换时，转换为多跳边的路径

```
// 实体类型属性支持
create (n:设备 {id:0, 设备名:"devid****001"})

// 标准属性创建时，Trigger 触发执行语句将其转换为边
match (m:wifi {id:"TP_LIN*****0011"}), (n:设备 {名称:"A"}) create (n)-[:use_wifi]->(m)

// 概念属性创建时，Trigger 触发执行语句转换为多跳边
match (m1:国家 {名称:"中国"})<--(m2:省 {名称:"四川"})<--(m3:市 {名称:"成都"}), (n:人 {姓名:"张三"}) create (n)-[:livi_in]->(m3)
```

- 支持知识分层，基于自定义逻辑规则，通过 Trigger 进行自动的插入和更新。

表 11 SPG 类型分类能力

需求内容	SPG-Engine能力	功能简介
实体类型	类型约束	创建点边时，符合约束规则的点将被创建一条与标记点相连的边
关系类型	类型约束	创建点边时，符合约束规则的点将被创建一条与标记点相连的边
属性类型	属性约束	属性修改时，符合约束规则的点的属性相应修改

```
// 定义实体类型即添加类型约束，创建 Person 类型时，检测是否符合约束条件，符合则创建 belongTo 边
// 创建 Person 点
create (n:Person {id:"2088****0001"})
// 触发服务端约束检测，符合约束则创建相应边
match (n:Person {id:" 2088****0001"})-[:has]->(D:Device)-[:has_wifi]->(W:WIFI)<-[:has_wifi]-(D2:Device)<-[:has]-(s)-[p:belongTo]->(o:Fraudster) create (n:Person {id:" 2088****0001"})-[p:belongTo]->(o:Fraudster)

// 定义关系类型即添加类型约束,创建设备点时，检测是否符合约束条件，符合则创建 same_wifi 边
create (n:Device {id:" devid****001"})
// 触发服务端约束检测，符合约束则创建相应边
match (n:Device {id:" devid****001"})--[:has_wifi]->(W:WIFI)<-[:has_wifi]-(o:Device) create (n)-[:same_wifi]->(o)


// 定义属性类型即添加属性约束，App 点属性修改时，符合约束规则的点的属性相应修改
match (n:App {id:"appid*****0012"}) set n.mark = "black"
// 触发服务端约束检测，符合约束则修改相应点属性
match (n:App {id:" appid*****0012"})<-[:release]-(m:公司) set m.mark = "black"
```

- 支持内置谓词，内置谓词需要多种 LPG 进阶能力，对应关系见表 12。

表 12 LPG 进阶能力要求

需求内容	LPG进阶能力	功能简介
传递性（transitive）	路径的条件匹配	输入点边及路径条件，进行路径匹配
同一性（sameAs）	图映射	根据条件匹配子图，并进行数据计算聚合操作，再用结果在当前图创建新子图
条件互反（conditionInverseOf）	谓词匹配	谓词表达条件互反关系
互反（inverseOf）	谓词匹配	谓词表达互反关系
子属性（subPropertyOf）	属性组值类型支持	属性组内部可以有多个子属性，但子属性不能为属性组类型
标准属性（normalizedProperty）	见 “自定义逻辑规则”	
属性等价（equivalentProperty）	属性映射	点上的多个属性相等约束
互斥（mutexOf）	主键约束	某类型点主键唯一

```
// 传递性,需要图库支持路径条件匹配
// 用户画像聚集，与张三拥有同样爱好口味的人群聚集
match (n:人 {姓名:"张三"})-[:爱好|isA*3]->(m:口味)<-[:爱好|isA*3]-(m2:人) return m as 口味, collect(m2) as 人群
// 最终实控人挖掘
match (n:公司 {id:" 4201151234****ABC" ) <-[r:控股 *1..5]-(m:人)
return m as 控股人, sum(reduce(total = 1, h IN r | total * h.持股比例 / 100.0 ) ) as 持股比例 order by  持股比例

// 条件互反，谓词匹配
// 通过 A 查找其全资母公司
match (A:公司 {id:" 4201151234****ABC"})-[:子公司 {holding_rate:1}]->(B) return B
// 通过 B 查找其全资子公司
match (B:公司 {id:" 4201151234****ABC"})<-[:子公司 {holding_rate:1}]-(A) return A
```

5.7 本章总结

本章介绍了 SPG-Engine 层的架构和实现。SPG-Engine 层包含 SPG-Engine Core 和第三方属性图系统两大部分。SPG-Engine Core 提供了 SPG 语义下的图模型定义、图数据导入、图查询和计算的功能模块，并调用第三方属性图提供的接口进行执行。第三方属性图由 LPG 图服务厂商提供，可以是 HTAP 的统一集群，也可以是 OLTP 图数据库集群和 OLAP 图计算集群两个独立集群共同完成能力支撑。本章还描述了对接到底层 LPG 处理系统的方式和第三方属性图系统应具备的功能，使得 SPG 可以运行在各种常见的属性图系统之上，并且也给属性图厂商提供了提升 SPG 性能的优化方向，更多细节及 RDG 的实现案例我们会在 SPG 公众号连载文章中持续发布。

第 6 章 SPG-Controller 层

SPG-Controller 是 SPG 框架的控制层，负责统一分析、调用和管理服务和任务的执行。作为 SPG 框架的核心枢纽，它与其他模块之间紧密关联，共同完成从用户输入到结果返回的全任务流程。SPG-Controller 通过从 SPG-LLM 或 SPG-Programming 接收请求，进行解析编译并生成任务规划，对任务进行分发和调用，选择对应的能力来完成具体执行过程，包括从注册部署的 SPG-Engine、SPG-Index 或外部能力中选择对应的 Runtime。最后，它将任务执行结果返回给调用者。本章概要式介绍了 SPG-Controller 的整体架构和工作流程，并对任务编译规划、任务分发调用以及知识查询、构建、推理、搜索等服务进行了总体说明，详细介绍待结合相关子系统在 2.0，3.0 白皮书逐步展开。

6.1 SPG-Controller 架构与工作流

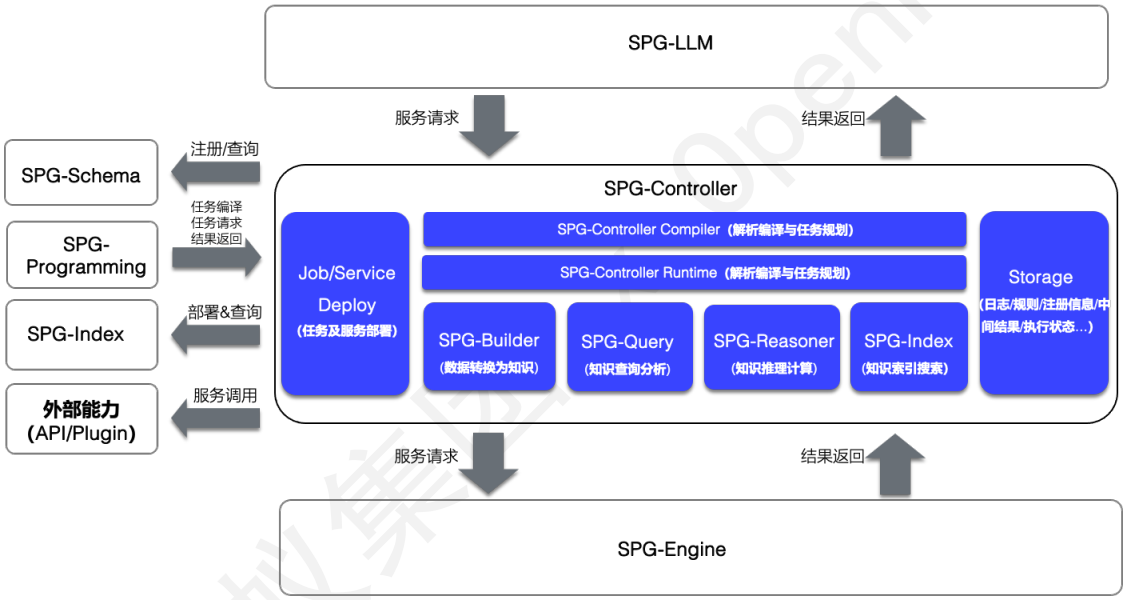

图 39 SPG-Controller 的架构与工作流

SPG-Controller 是 SPG 框架中的核心中控枢纽，主要职责包括以下方面：1）解析编译及任务规划：根据上层调用输入进行解析与编译，对应到各子系统的能力、命令和相关配置及参数。然后，根据解析编译的结果，进行任务的规划，以实现任务执行方式、过程和周期等的编排。2）任务分发与调用：根据任务规划的结果，进行任务的执行和调用服务及任务，在相应的 Runtime 中执行。3）SPG-Builder：提供知识构建相关的 Runtime Engine，以实现 SPG 中图谱构建的数据到知识的转换。4）SPG-Query：提供知识查询的 Runtime Engine，实现 SPG 中查询与图基础挖掘分析相关的接口。5）SPG-Reasoner：提知识推理计算的 Runtime Engine，以实现 SPG 中各种推理能力，包括逻辑规则推理和神经网络推理。6）SPG-Index：提供 SPG 索引及搜索相关的 Runtime Engine，以实现 SPG 中向量搜索、全文搜索等能力。7） Job/Service Deploy：提供 SPG 各 Controller 对应 Runtime 中的服务或任务注册与部署，使其能力被有效调用。

在整个系统架构中，首先，SPG-Controller 作为 SPG-Schema 子模块的 Runtime Engine，对外提供 CRUD Python/Java 接口。其次，它为 SPG-Programming 提供任务编译及任务分发执行的能力。任务编译包括语法检查、算法/任务有效性检查与复用、执行计划及字节码生成等。此外，它还接受子引擎模块可插拔式添加到注册信息模块。最后，它接受任务请求的输入，编译执行、任务分发，将任务分发到各子引擎完成任务计算后接收结果，并将结果组装返回给用户。

6.2 解析编译与任务规划

输入解析模块从 SPG-LLM 或 SPG-Programming 接收输入，并对其进行解析，以理解用户输入中的指令，从而识别出需要执行的任务类型、执行逻辑及过程等。输入主要包含两部分：任务类型和任务主体。任务类型确定执行哪一类型的任务，任务主体包括调用的接口及其参数。以下是主要的任务类型及其定义：1）Schema 类任务：对应图谱 Schema 的基础增、删、改、查接口。任务主体中包括调用的接口及其参数。接口的定义及实现在 SPG-Schema 模块中。2） 查询类任务：对应图谱基础查询，调用 SPG-Engine 中的图查询能力。任务主体为查询语言 GQL。3）推理类任务：对应图谱推理类的任务。任务主体为规则语言 KGDSL 或神经网络推理任务。4）搜索类任务：对应搜索任务，分为向量索引及全文索引等。包括索引的创建、分词索引/向量数据写入和文本检索/向量搜索等。任务主体为带条件的 query 检索任务。5）构建类任务：对应 SPG 图谱构建任务，包括结构化数据到知识的映射、非结构化及多模态知识的抽取、知识的融合等。

任务规划依据解析编译的结果，进行任务的规划，以实现任务执行方式、过程及周期等的编排。

6.3 任务分发与调用

根据任务规划的结果，实现任务的分发，并在相应的 Runtime 中执行，包括已注册的各服务和任务对应的 Runtime 以及整体任务执行的 Pipeline。在任务执行过程中，SPG 提供微服务调用框架和任务调度引擎，以实现服务的调用和任务的执行。同时，SPG-Controller 提供了存储功能，以存储任务执行过程中的相关信息，包括任务日志数据、任务执行中间及结果数据、任务执行状态与调度数据等。这些信息的存储和溯源追踪有助于对任务执行过程进行监控和管理，以保证任务执行的可靠性和稳定性。

6.4 图谱构建

SPG-Builder 实现了从数据到知识的转换，包括结构化数据、半结构化数据和多模态非结构化数据的知识抽取，主要能力包括 1） ER2SPG：该功能数据从数据库或大数据平台中转换映射得到，实现数据到 SPG 知识的转换。在原始数据转 ER (Data to ER，D2R)的基础上，修改输入为 SPG 规范的输入。2）半结构化数据知识抽取：该功能用于对半结构化数据进行知识抽取，首先抽取得到

结构化要素。该功能将在二期添加。3）文本知识抽取：该功能后续会实现基于大模型的抽取，用于对文本数据进行知识抽取。该功能将在二期添加。4）多模态知识抽取：该功能用于对多模态非结构化数据进行知识抽取，包括图像、音频、视频等多种形式的数据。该功能将在二期添加。

通过这些功能的支持，SPG-Builder 可以实现从各种形式的数据中提取出有价值的知识，充分激活大数据体系沉淀的海量数据，并将其存储在 SPG 仓储中，以便于后续的应用、分析和推理任务。

6.5 知识查询

SPG-Query 提供图谱查询及分析服务的功能，主要包括图谱数据的基础增删改查、图谱的基础搜索，以及图的分析挖掘。具体查询功能包括：1）基础查询：支持实体、概念、属性的精确查询，例如在黑产图谱中查询账号为 2088****0001 账户。2）高级查询：支持实体、概念、属性的非精确查询，包括模糊查询和全文搜索等，例如在事理图谱中按事件名、企业名等模糊检索。3）图遍历查询：支持广度、深度优先算法进行图遍历查询，例如在黑产图谱中查询某账户有交易记录的账户。4）模式匹配查询：支持满足指定模式的子图查询，如在黑产图谱中满足 A-B-C-A 转账模式的查询等等。

此外，SPG-Query 还提供多种图分析算法，包括：1）社区类算法：如 LPA、WCC、SCC、Louvain 等，可用于在黑产图谱中发现频繁交易的团伙。2）权威度计算算法：如 PageRank、HITS、度中心性、中介中心性、亲密中心性等，可用于在黑产图谱中计算每个节点的权重值。3）其它算法：如三角形计数等，可用于在黑产图谱中计算某交易子图的交易频繁程度。这些功能的支持可以帮助用户更好地理解和分析 SPG 中的图数据，并从中挖掘出有价值的信息。

6.6 图谱推理

SPG-Reasoner 提供调用知识图谱推理能力，包括常用的知识图谱推理方法。具体推理包括：1）规则推理算法：可用于推理事理图谱中的风险传导规则、黑产图谱中涉黑/涉诈专家规则定义。2）图嵌入学习算法：包括图神经网络、随机游走、翻译距离等，可用于对 SPG 中的知识图谱进行嵌入学习和表示学习。3）提示学习算法：用于构建 SPG-LLM 中大模型提示的学习算法，该功能将在未来发布中重点规范实现。

为了支持这些推理能力，SPG-Controller 提供了推理规则、推理算法的注册信息以及算法参数配置等的存储管理，有助于对推理过程进行监控和管理。

蚂蚁集团 Ant Group

6.7 全文搜索和向量搜索

SPG-Index 提供搜索服务，包括向量搜索和全文搜索等常规搜索。SPG-Controller 与外挂的 SPG-Index 进行调用交互，用户可以方便地进行数据的搜索和查询。具体功能包括：1）索引的创建与管理：支持创建和管理索引。2）索引数据的写入与更新：支持将数据写入索引并进行更新。3 基于向量的搜索：支持基于向量的搜索，以便于根据相似度进行数据的搜索和查询。

6.8 服务与任务部署

Job/Service Deploy 实现各核心模块对应 Runtime 的服务或任务注册与部署，具体提供的功能包括：1）注册：支持服务及任务的注册，以便于在 SPG-Controller 中被发现和调用。2）管理：支持微服务或任务的管理，包括上下线、配置等。3）执行：支持在 Runtime 中执行服务或任务。4）监控：支持监测运行的状态，包括资源的监控，以便于及时发现和解决问题。

具体实现上，SPG-Controller 中存储了注册部署的服务及任务，并在任务执行调用时进行调度。为了实现任务调度，可以使用任务调度引擎来管理精确至时分秒定时执行的作业，并通过动态设置分片参数来设置任务的并发大小数。而为了实现微服务调用，可以使用微服务治理框架来管理和调用微服务，并通过微服务网关对外提供服务。

6.9 本章总结

本章对 SPG-Controller 的整体架构及工作流程进行了概要式描述。在具体的任务方面，当前版本主要针对结构化数据构建图谱、图谱基础查询与分析、图谱推理及索引搜索作了简要描述说明，在后续的版本中会对当前的任务进一步细化与扩展，包括非结构化数据构建图谱、多模态数据构建图谱，表示学习推理、复合索引等方面；同时会进一步扩展其它类型的任务，包括外部插件系统、指令系统等的调用集成等。此外，在后续版本中会进一步完善 SPG-Controller 的管理类功能，包括统一鉴权管理、统一异常处理等方面。

第 7 章 SPG-Programming 层

领域模型与基础引擎分层解耦是 AI 基础引擎必须具备的基础能力，通过可编程 SDK 和算子框架将业务领域模型从基础引擎中剥离出来，帮助开发人员快速构建和部署自定义图谱算法、Schema 模型和推理能力，帮助业务快速构建和部署知识图谱应用，提高应用的效率和可扩展性。基础引擎也更加聚焦在 I/O、调度、性能、吞吐等通用能力的优化。SPG 引擎在架构上也分为三层：核心基础引擎层、SDK 及算子框架层、业务应用层。

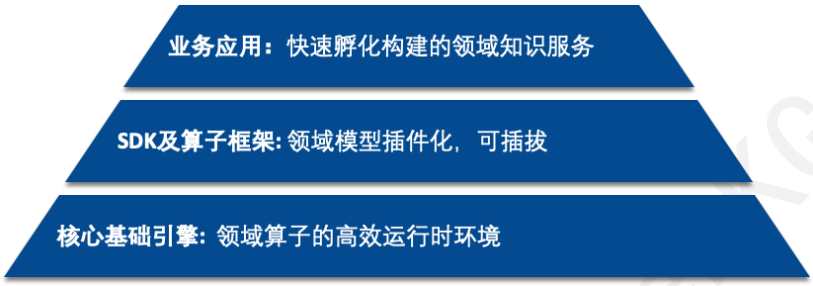

图 40 SPG-Programming 领域分层

本章节仅对 SPG-Programming 做简单概要介绍，更完整的介绍及 KGDSL 的完整语法表示会在白皮书 2.0 版本正式发布，也会通过系列文章详细介绍。

7.1 SPG 语义可编程架构

可编程子模块的主要功能是为用户提供可编程的交互界面，支持算法/业务工程师按照 SPG 提供的编程框架实现领域图谱的建模、构建和推理，并支持领域图谱的持续迭代。

图 41 SPG 可编程分层框架

在可编程能力方面，包括以下几个方面：

- 知识生产框架：基于结构化、非/半结构化数据构建领域图谱的算子框架。
- 逻辑规则编程：包括逻辑规则属性/关系、KGDSL 规则推理等。
- 表示学习推理：以图谱数据为基础，实现图神经网络类、逻辑引导类等图谱推理。

7.2 数据到知识的生产转换

如第 1 章图 1 所示，借助 SPG 框架，通过可编程算子实现数据到知识的转换，包括如下几类算子：1) **信息抽取算子：**实现所有非、半结构化数据的结构化，得到知识要素。2) **主体挂载算子：**包括实体、事件、概念类型挂载，及实体/事件属性的类型挂载。3) **实体融合算子：**解决实体构建多源异构更新及跨图谱知识融合等问题。**4）动态分类算子：**为达到支持业务动态定义类型颗粒度而通过逻辑规则定义的分类表达。

通过标准化算子框架和属性要素，可以显著降低原始数据准备成本。算子只与目标实体、概念、事件类型相关，因此也可以极大减少重复开发成本。引入属性要素标准化后，Schema 建模时属性的类型也从文本(Text)、整型(Long)、浮点数(Double)转变为领域模型建模，基于 LPG 建模的伪码表示：

```
class User {
  id TEXT(文本);
  name TEXT(文本);
  phoneNo LONG(整数);
  poc     TEXT(文本);
  homeAddr TEXT(文本);
}
```

基于 SPG 建模的伪码表示为:

```
class User {
  id UserNormId(用户标准 ID);
  name TEXT(文本);
  phoneNo ChnMobilePhone(中国大陆电话号码);
  poc     AdminArea/省/市/区(行政区划);
  homeAddr POI(标准 POI);
}
```

通过准备一张 User 表，用户可以构建出四类实体和四类关系，从而大幅降低原始数据准备成本。引入知识生产算子后，将算子框架和算子实现分离，并定义知识构建过程为标准组件，为这些组件提供统一的运行时框架。算法开发者可以基于 Python 算子框架快速实现知识生产算子，并将其绑定到目标类型。同时，可以为每个类型指定 Link Function/Fuse Function 或 Normalize Function，

方便图谱的持续迭代，并解决工业应用下非完备数据持续构建演化、实例归一化问题。算子定义简单示例如下伪代码所示，可以通过 bind_to 绑定到具体的类型上

```
# -*- coding: utf-8 -*-

from knext.api import EntityLinkOp
from knext.api.base import BaseOp
from knext.models.runtime.vertex import Vertex

@BaseOp.register("AdminAreaNormOp", bind_to="AdminArea", is_api_iface=True)
class AdminAreaNormOp(PropertyNormalizeOp):
    def eval(self, property: str, record: Vertex) -> EvalResult[str]
        traces, errors = [], []
        result = ""
        try:
            result = adminNorm(property)
        except Exception as e:
            errors.append(f"property:{property}, error_msg:{e.__repr__()}")
        return EvalResult(result, traces, errors)
```

为降低用户使用成本，SPG 支持属性、关系在查询阶段的自适应，在用户的 GQL/KGDSL 表达中需要传播时会自动展开，若不需要传播，默认提取标化之后的属性值。未来在 KGDSL 详细语法文章中会详细介绍。

7.3 逻辑规则编程

谓词语义是实现 SPG 逻辑规则编程的关键基础，通过谓词语义可以将 SPG 翻译成机器可理解的形态，构建机器自动推理能力。在能力定义上包括如下几层：**1）系统内置谓词**，以确定的语义定义基础谓词能力，它不具备业务语义，但可被上层规则引用。**2）逻辑规则知识**，以属性/关系存在的逻辑规则，并基于逻辑规则实现实体的动态分类。**3）推理决策规则，**以子图、结构、路径等形态获取，支持规则决策、知识注入等。图 42 说明了整体的分层结构，同时为了平衡规则管理成本和计算复杂度，也明确了内置谓词只能用于定义逻辑规则知识，知识推理层的应用则只依赖基础事实知识和逻辑规则知识。

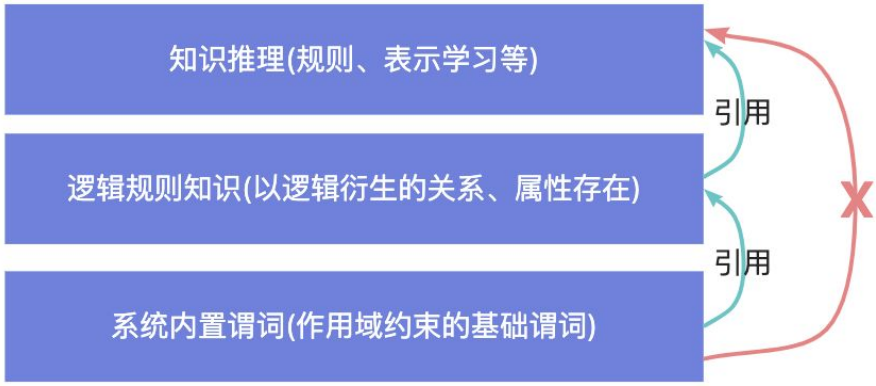

图 42 推理决策依赖关系

通过谓词与逻辑规则定义知识之间的依赖第 4 章已经有详细的介绍，本章不在赘述。逻辑规则编程主要包括两部分，通过逻辑规则定义知识依赖， 生成逻辑衍生属性/关系和通过 DSL/GQL 定义复杂的端到端规则决策，如下为 KGDSL 决策代码示例。

```
Structure {
  (s:User)
  (e1:TradeEvent)-[ps1:std.subject]->(su1:User)
  (e1:TradeEvent)-[pp1:std.object]->(sp1:PID)
  (e2:TradeEvent)-[ps2: std.subject]->(su2:User)
  (e2:TradeEvent)-[pp2: std.object]->(sp2:PID)
  (su1)-[has]->(sp2)
  (su2)-[has]->(sp1)
  (e2)-[pb:belongTo]->(o:/TaxoOfTradeEvent/单笔交易金额高)
}
Constraint {
   s.id == su1.id
   e1.ts < e2.ts and hour(current_time()) - hour(e1.ts) < 24
   group(s).count() > 10
}
Action {
    createEdgeInstance(
                src=s,
                dst=o:TaxoOfUser/交易风险/返款交易多,
                type="belongTo",
                value= { time=now() }  )
}
```

7.4 图谱表示学习

图谱表示学习框架解决的核心问题是图谱特征、子图提取问题，并适配到主流的深度学习框架如 Tensorflow/Pytorch 上，并进一步转换成对应图学习算法需要的 Tensor 结构。

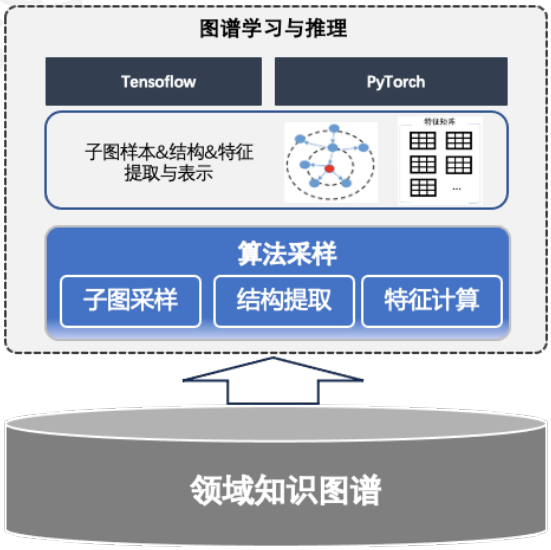

图 43 图表示学习框架

图谱表示学习通过采样算子模块实现图学习与图谱数据的联动与解耦，目前主要的图采样算子包括：**1）子图采样**，主要应用于 GCN 类的多跳子图采样，包括正/负样本生成，在 SPG 大规模动态异构范式下支持带权采样、时间过滤等。**2）结构提取**，应用于符号规则引导的图学习、规则挖掘等结构感知型推理任务。**3）特征计算**，主要包括通过复杂图谱结构可提取的 Page Rank，度中心性等子图特征。如下伪码为 GCN 类算法多跳子图采样的示意，通过 Python 采样算子实现算法对图谱数据高效读取。

```
# -*- coding: utf-8 -*-
import libkg_client
from kgrl.conf import KgrlConstants # noqa
from kgrl.data import KGExpression # noqa
from kgrl.data.sampler import KGStateCacheBaseSampler

in_degree = KGExpression.SourceNodeInDegreeKey()
out_degree = KGExpression.SourceNodeOutDegreeKey()
node_version = KGExpression.SourceNodeVersionKey()
edge_version = KGExpression.EdgeVersionKey()
v_begin = 30
v_end   = 40
def get_filters(v_begin, v_end):
   return {
      KgrlConstants.NEIGHBORHOOD_SAMPLING_FILTER_NAME: f"{edge_version}<{v_begin} and
{edge_version}>{v_end}",
      KgrlConstants.NODE_SAMPLING_FILTER_NAME: f"{node_version}==0",
      KgrlConstants.EDGE_SAMPLING_FILTER_NAME: f"{edge_version}<{v_begin} and {edge_version}>{v_end}",
   }
def get_weights(v_begin, v_end):
   return {
      KgrlConstants.NEIGHBORHOOD_SAMPLING_WEIGHT_NAME: f"abs({edge_version}-
{v_begin})*log2({edge_version}+{v_end})",
      KgrlConstants.NODE_SAMPLING_WEIGHT_NAME: f"({out_degree}+{in_degree})",
      KgrlConstants.EDGE_SAMPLING_WEIGHT_NAME: f"abs({edge_version}-
{v_begin})*log2({edge_version}+{v_end})",
   }
sampler_conf = {
    "client_conf": {...},
    "gen_data_conf": {
        "random": True, "fanouts": [50, 20], "buffer_size": 2, "filters": get_filters(10, 20), "weights": get_weights(10, 20),
   },
}
sampler = NodeSubGraphSampler.from_params(sampler_conf)
```

7.5 本章总结

本章节概要式介绍 SDK 可编程框架的分层抽象，预计在 《SPG 语义增强可编程知识图谱框架》白皮书 2.0 中重点发布完整的可编程框架。

第 8 章 SPG-LLM 层

2023 年伊始，大模型展现出其强大的能力，在语言理解、对话生成方面表现的尤其亮眼。而知识图谱则擅长大模型所无法解决的事实性“幻觉”和复杂推理问题。有效结合知识图谱和大语言模型各自的优势，充分发挥各自的特长，可以提供更优质的人工智能服务和产品。

在 SPG 的基础上，借力 LLM 结构、语义、逻辑理解能力，形成 SPG+LLM 的双轮驱动。基于 SPG 强 Schema、逻辑约束、符号化的表达能力，进一步为提升领域知识构建与推理效率，加速知识图谱的产业落地，结合用户自然语言表达的意图理解/意图扩散、任务构造、可控生成等实现自然语言交互式的图谱查询和推理，是我们持续探索的方向。本章节简要介绍 LLM 与 SPG 自然语言交互架构，并结合达观科技的实践介绍基于 LLM 的知识抽取。

8.1 SPG-LLM 自然语言交互架构

结合图 24 的总体架构定义，大语言模型交互主要分为四部分：大模型适配接口(LLM Adapter Interface)、知识图谱的自动抽取&构建(SPG Constructor)、基于大模型实现 SPG 的自然语言查询(SPG NL Query)和推理(SPG NL Reasoner)。

8.2 自动抽取和图谱自动化构建

引入 LLM 之后，知识图谱的构建过程如图 44 所示:

图 44 知识图谱构建链

业务理解和 Schema 设计：知识图谱的 Schema 的设计和实现需要对领域内的知识有深入的理解和抽象，同时也需要考虑数据源的质量和可获取性，以及应用场景的需求和限制。这个过程通常由多方合作完成，在珠峰书中梳理了一系列实践经验，并总结为“六韬法”（如图 44 所示）。在 Schema 设计过程中，需要充分应用 SPG-Schema 的内容，并在其基础上，进一步扩展 SPG-Schema 的内容，引入标准化自然语言注释。SPG-Schema 的自然语言注释有关的内容会在后续版本的白皮书中引入。这里面可以参考的内容是 Ontology（本体） 中的一些定义，它可以定义概念、属性、关系、约束和规则等元素，并支持推理和验证。如 schema.org，FIBO，GO 等知名本体库可以参考或者复用，来优化 Schema 的设计。其目标是保证所设计的 Schema 的规范性、一致性和通用性。

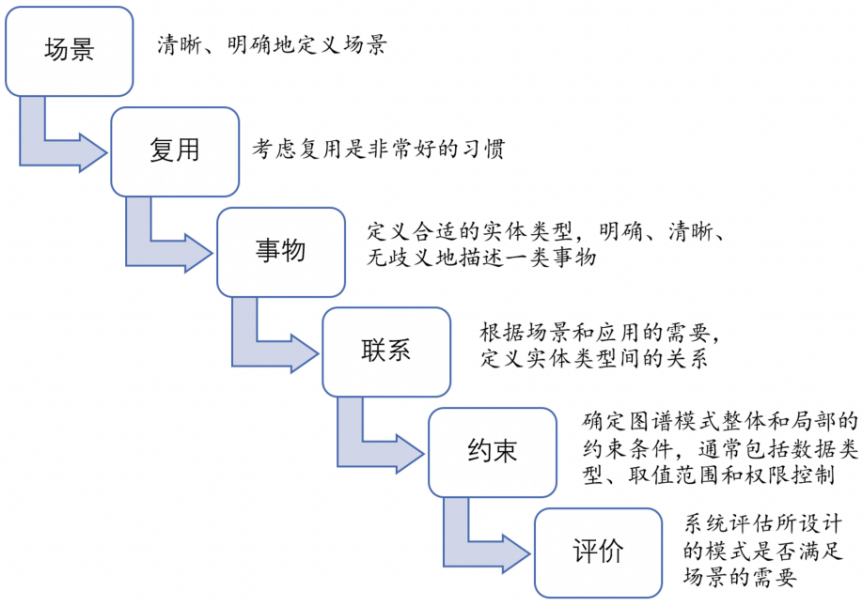

图 2-13　六韬法——六韬瀑布模型

图 45 珠峰书《知识图谱：认知智能理论与实战》第二章[6]

人工梳理样例和自动/人工编写 Prompt：基于所设计的 Schema 进行提示工程的工作，来实现实体、关系和属性的自动抽取，进而构建出知识图谱。自动化生成 Prompt 的引擎，也可以参考本体中的推理引擎来实现。这里自动生成 prompt 会依赖于 Schema 中的自然语言注释，以及人工梳理的样例。在实践中，通过人工梳理样例或使用 LLM 自动生成抽取样例，有助于使用少样本学习，来提升 LLM 抽取的准确性。

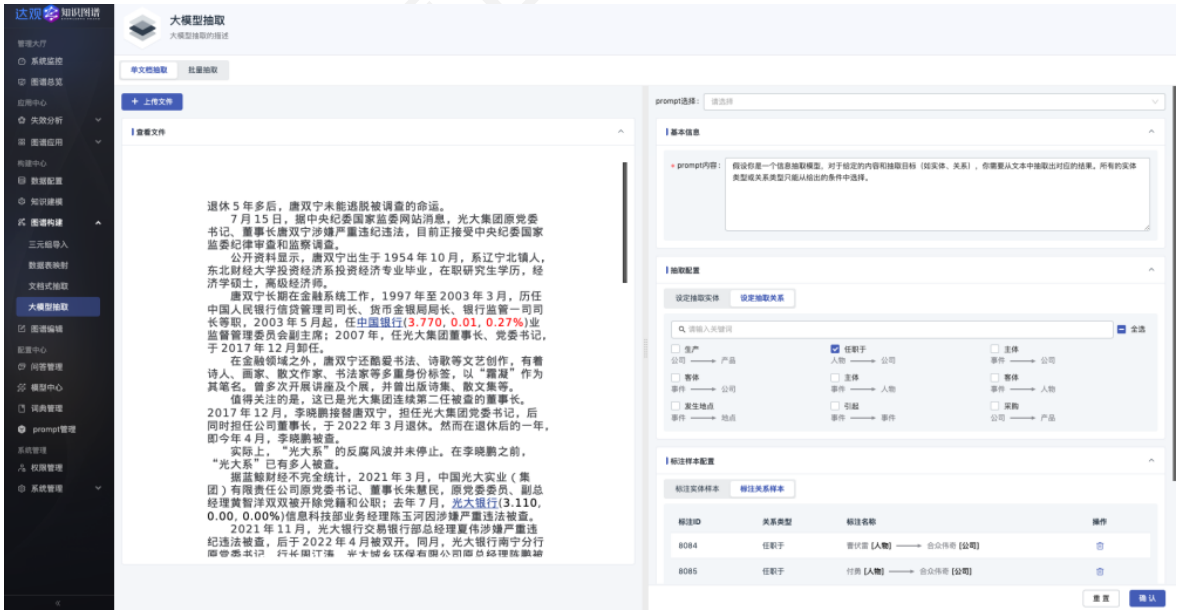

图 46 大模型抽取示例

LLM 抽取和可选的人工审核：利用 LLM 来构建知识图谱，同时，在必要的情况下，提供人工审核来确保所构建知识图谱的准确性。

图 47 大模型抽取审核示例

图谱构建：将 LLM 抽取的结果融合进已有的知识图谱中。基于 LLM 进行实体抽取、关系提取等方式,从大量文本中构建出知识图谱的核心在于知识图谱 Schema 中定义了知识图谱中的实体类型、关系类型和属性类型等元素的规范，特别是相关的自然语言注释。这与 SPG-Schema 的规范强相关在 Schema 中提供自然语言注释，有助于将其转化为大模型抽取和交互的 Prompt。这里面应当有三个层次的内容：

- 实体类型、关系类型和属性类型本身的自然语言注释
- 概念层次结构、语义关联和逻辑规则等层面的自然语言注释
- 对上述两类注释的标准化，这有助于实现公共的自动化生成 prompt 的库（引擎）

Schema 的自然语言注释，一方面能够实现 Prompt 的自动生成，另一方面在利用大模型进行知识图谱构建时，可以利用大模型来自动生成少样本学习的样本。在实践中，在关系抽取中，少样本学习是非常重要的，零样本要实现好的关系抽取非常难，而少样本学习能够大幅提升关系抽取的效果。

8.3 基于大模型的领域知识补全

使用大模型进行知识补全可以帮助中小机构获取更丰富的专业知识。相较于仅仅依赖企业内部积累的知识，大模型使用数量巨大的语料来获取完善的常识和领域知识。通过特殊方法从大模型中萃取知识并保存到知识图谱，可以为企业提供更高效的知识积累和使用。与传统将已存在图谱中的隐性知识显性化过程的知识挖掘，大模型的知识补全更侧重于萃取 LLM 中的专业知识并融入到知识图谱中，提供不存在于知识图谱中的知识。本次白皮书仅提出大模型“知识补全”的概念，更多的实现方法、例子、意义等敬请期待后续版本。

8.4 自然语言知识查询与智能问答

传统知识图谱对自然语言理解能力较弱，大语言模型正好可以弥补这一缺陷,它经过数百亿参数的训练，拥有接近人类的语言理解和生成能力。将两者有机结合,可以让知识图谱理解用户的自然语言查询，并利用其内在知识提供准确答案。大语言模型负责语义分析，知识图谱提供结构化知识来检索答案，两者互为补充。这种结合既发挥了知识图谱的结构化知识优势，也利用了大语言模型对自然语言的理解力，从而提供更人性化的问答服务。大语言模型分析查询语句的真正意图，知识图谱则提供丰富的背景知识，帮助挖掘更准确、更相关的搜索结果。在对话系统中,知识图谱也为会话提供了丰富的常识性知识来源,使对话更加智能化和接近人类交流。大语言模型负责自然的语言交互,知识图谱则补充相关知识,使机器人拥有更强的上下文感知能力。

结合大语言模和向量检索的强大能力，将自然语言交互和知识图谱结合，形成可控、可信、可靠的问答，解决大模型自身所无法解决的“幻觉”问题，为产业应用解决“最后一公里”实现落地，如图 48 所示。

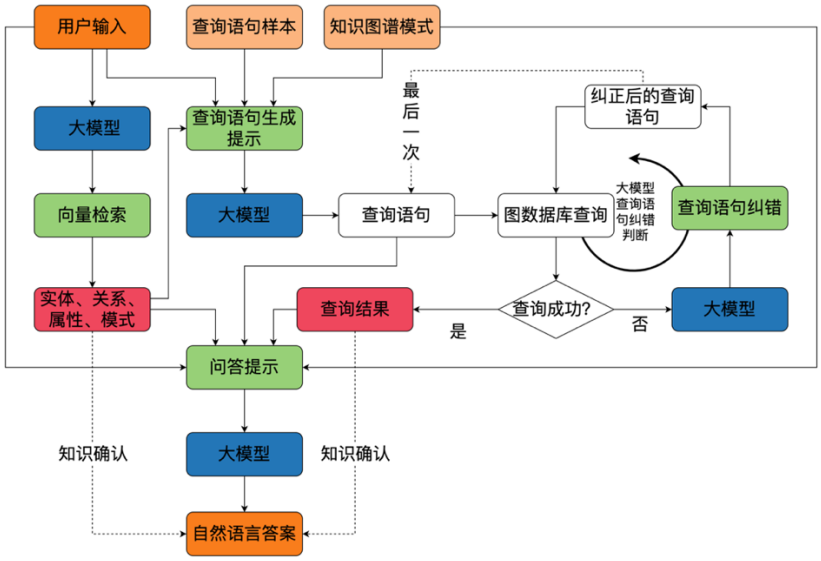

图 48 大模型+知识图谱实现可控可信可靠问答的架构(初稿)

NL2GQL/NL2KGDSL 通过人工标注的万条级别的自然语言-GQL/KGDSL 对即可对 LLM 进行 SFT，进而实现自然语言知识查询与智能问答。本次白皮书发布内容主要集中在原理性的探讨。

在后续版本中，会在 GQL 或 KGDSL 更为成熟之后，发布相关的数据集，以及基于开源大模型的 SFT 规范、代码仓库以及模型等，敬请期待后续更新。

8.5 本章总结

本章节概要式介绍 SPG-LLM 层的基本原理、框架型内容，也提出“知识补全”的概念，预计在《SPG 语义增强可编程知识图谱框架》白皮书未来发布中重点完整介绍的 SPG-LLM 层的内容。

第 9 章 SPG 驱动的新一代认知应用案例

在第 2 章中总结和分析了金融事理、黑产风控基于属性图的图谱构建和应用上存在的问题，本章节结合第 2 章中提出的问题，阐述基于 SPG 是如何解决的，并给出整体的解决方案。

9.1 SPG 驱动的金融事理图谱

本章节以 2.3 章节中提到的 2019 年发生的巴西淡水河谷的溃坝事件为例。这个事件造成了铁矿石价格上涨，从而导致下游企业炼钢成本上涨。在整个事件影响链中，与淡水河谷同属于一个产业下的企业（即竞争关系企业）受益，利润有所上升。但对产业链的下游造成了负面影响，原材料成本上涨导致企业利润下降。针对 2.3 章节提出的事理图谱在属性图上应用的问题，我们提出 SPG 的解决方案。

1. 通过可派生的概念应对事件动态分类的要求

事理图谱涉及事件在概念层面的演化推理，因此首先需要将事件实例映射到相应的事件概念，通过 belongTo 谓词将事件实例指向相应概念。由于事件类型众多，难以全部预先定义，因此 SPG 支持使用概念通过特定组合规则派生出新的概念，从而实现对事件实例的动态分类。一个具体的案例如图 49 所示。

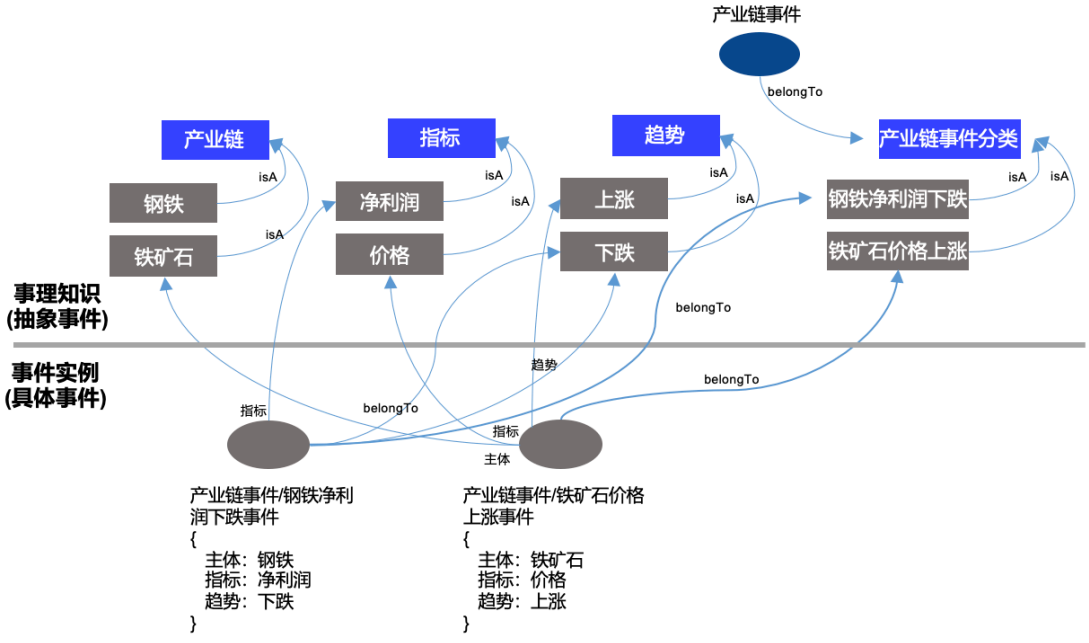

图 49 组合概念

本例中涉及“钢铁净利润下跌”和“铁矿石价格上涨”两个事件实例。其中，前者属于产业链事件分类，通过“指标：净利润”和“趋势：下跌”的组合可以派生出“净利润下跌”的概念。同理，后者也属于产业链事件分类，通过“产业链：铁矿石”、“指标：价格”和“趋势：上涨”的组合，可以派生出

“铁矿石价格上涨”的概念。值得注意的是，SPG 不会立即将所有可能的概念组合全部派生，而是针对实际发生的事件实例，派生相应的事件概念，从而避免无意义或违反事实逻辑的概念体系。

2. 通过概念事理层建模解决无法表达整个事件脉络

在解决了 1. 的问题基础上,SPG 则可在概念分类体系上建立事理因果关系,用于表达事件脉络,如图 50 所示，其中红色虚线表示事理知识层 leadTo 被激活后在事件实例层自动产生的事件传导。

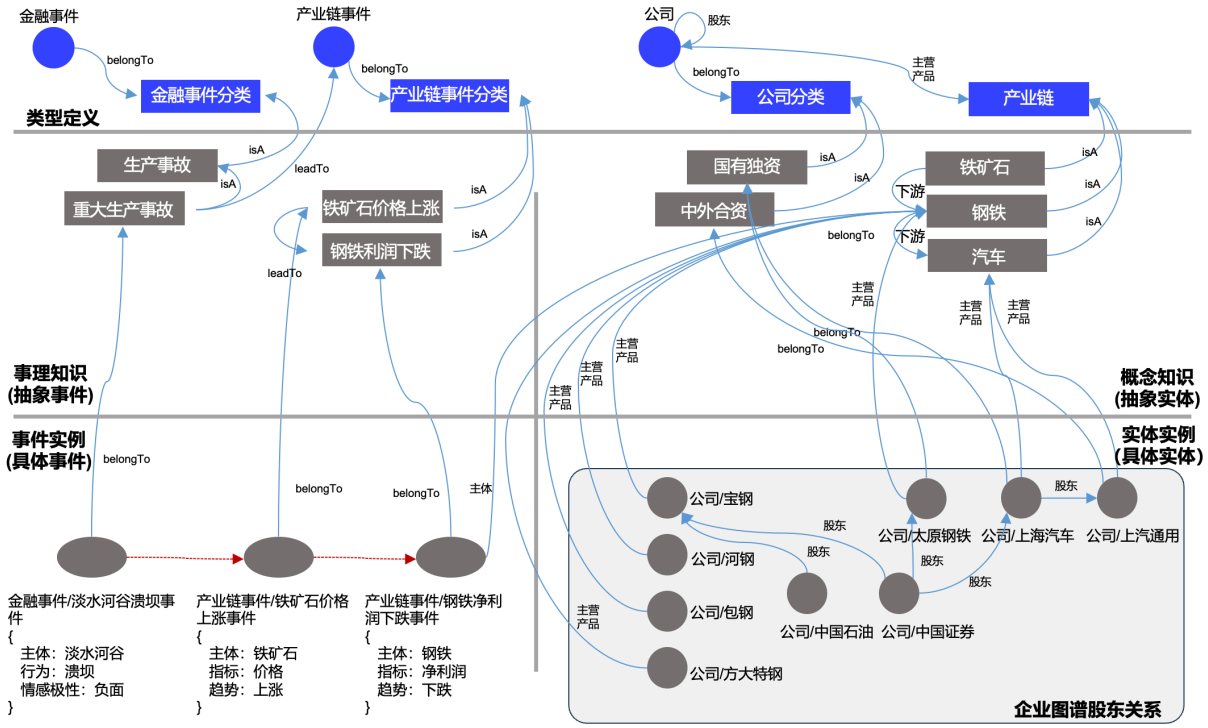

图 50 事理演绎

图 50 中，从下至上分为事件实例和事理知识两层。事件实例层表示的就是具体的事件实例，例如“淡水河谷溃坝事件”，可根据 1）中的方法映射到事理知识层，事理知识层具有事理上的因果关系表达能力。在本例中，在事理知识层中已经定义：企业的“重大生产事故”会导致企业所属产业链出现“铁矿石价格上涨”或者“钢铁净利润下跌”等。当发生淡水河谷溃坝事件且该事件被归类到“重大生产事故”概念时，事件推理就从事件实例层转移到了事理知识层继续进行。根据已定义的事理知识，事件传导机制会在事件实例层生成一个新的产业链事件实例，并且新产生的事件实例又会被归类到“铁矿石价格上涨”概念，从而让新产生的事件实例成为“铁矿石价格上涨事件”。事理知识可从对事件进行归纳或者从事理模式泛化而来，用于指导具体事件实例分类和传导。例如“重大生产事故”会导致“产业链事件”发生，“产业链事件”属于“铁矿石价格上涨”概念或者“钢铁净利润下跌”概念，则规则模板为：

RULE1：重大生产事故 -leadTo-> 产业链事件

RULE2：产业链事件 -belongTo-> 铁矿石价格上涨

RULE3：产业链事件 -belongTo-> 钢铁净利润下跌

此时可将事件归类到派生概念，让满足规则的派生概念形成事理因果关系，最终形成事理知识层的事理因果关系。这种表达方式从上至下指导事件实例层的分类和传导约束，实现了在事理知识层和事件实例层对整个事件脉络的表达。在上例中，事理知识层表达了事故导致涉事主体所属行业的下游企业利润下跌，具化到事件实例层就是“淡水河谷溃坝事件”导致了若干钢铁公司利润下跌。

值得注意的是，SPG 提供的是一种事理描述框架,事理间因果关系还是需要根据业务特点由使用者进行创建，概念归纳方法更详细的案例会在未来 SPG 系列文章中针对性发布相关实践。

3. 逻辑表达内置解决数据不受推理逻辑约束问题

SPG 将概念分类逻辑使用逻辑约束进行表达,以溃坝事件为例,需要对该事件分类,可定义如下规则。

```
Define (s:FinancialEvent)-[p:belongTo]->(o:`FinancialEventTaxonomy/重大生产事故`) {
  Structure {
        (s)-[:std.subject]->(company:Company) //关联公司实例
  }
  Constraint {
    R1("主体发生生产事故"): s.behavior == '生产事故'
    R2("主体公司市场占有率超过 x%,具有重大影响"): company.marketShare > x%
  }
}
```

上述规则含义为:当发生一个安全事故时,且该公司市场占有率超过 x%,那么该事件被归纳到对应行业的生产重大生产事故。同样，事理之间的 leadTo 关系也可进行逻辑表达,当发生了一个 e1 事件后,会产生并激活另一个 e2 事件。

```
Define (s:`FinancialEventTaxonomy/重大生产事故`)-[p:leadTo]->(o:`IndustryChainEventTaxonomy/价格上涨`) {
  Structure {
        (s)-[:std.subject]->(company:Company)-[:industry]->[I:Industry) //获得行业
  }
  Constraint {
  }
  Action {
    createNode(
        type=IndustryChainEvent
        value={
          subject=I.name
          index='价格'
          trend='上涨'
```

```
        }
    )
  }
}
```

事件传导产生一个新的事件实例后，新的事件实例可再次触发对该事件的分类，事件的分类可以使用组合概念进行分类。以下举例使用“产业链(IndustryChain)”、“指标(Index)”、“趋势(Trend)”三个概念类型作为组合概念对上述的产业链事件进行分类的定义。

```
Define (s: IndustryChainEvent)-[p:belongTo]->(o: `IndustryChainEventTaxonomy`/`IndustryChain` + `Index` + `Trend`) {
  Structure {
  }
  Constraint {
    o = s.subject + s.index + s.trend
  }
}
```

4. 逻辑属性、关系解决外部辅助数据依赖的问题

在上述例子中，Company 存在一个属性 marketShare（市场占有率），在实践中该数据可能来自其他系统，不存在于图谱系统中，此时可以使用逻辑规则定义该数据来源，如下示例。

```
Define (s:Compnay)-[p:marketShare]->(o:Float) {
 Structure {
  (s)
 }
 Constraint {
  o = callForMarketShares(s.id, 'marketShare')
 }
}
```

第 6 行代码 callForMarketShares 为一个 udf 算子，可向其他系统获取市场占有率信息。和之前所有数据均需要导入图谱方式相比，这里不需要额外拷贝其他系统数据，可以保证逻辑上数据一致，解决金融事理图谱场景下依赖外部数据决策情况。

5. SPG 解决推理结论可解释性不足的问题

本例中，SPG 通过事件概念定义解耦了事理层面和实例层面的传导性问题,且保证了逻辑规则和数据的一致性，按照四象限思路，保证以下四个方面的可解释

- **事件概念体本体的泛化与派生逻辑的可解释**

在事理模式层，定义各类事件及实体类型的结构化表示 Scheme 模式；同时自顶向下的定义实体概念本体（如产品分类）和事件本体的概念上下位体系。每一个更细粒度的事件概念，是基于事件槽位值填充，对上一层级事件概念的具象化（例如，图 50 中的“产品价格变化事件”相对于“产业链事件”，是对“指标”这个槽位，填充为特定值“价格”；而“净利润上涨”是对“净利润变化”在“趋

势”这个槽位值的进一步约束）。使用槽位定义->自顶向下槽位值具象化的方式，通过槽位值的属性组合，实现概念语义的可解释。上下位概念之间，拥有明确的泛化与派生逻辑。

- **事理常识逻辑关系的可解释**

通过定义 RULE 模式，实现概念事件间的因果、顺承、时空关系的逻辑定义。“产品价格上涨”不一定导致“产品利润下降”，通过对领域专家知识总结或对大量实际案例的分析，由于产业链上下游的供需关系影响，能够得到“上游原料产品的价格上涨，导致下游产品净利润下跌”的规则。进而基于已知的产业链上下游关系，能够批量的推导和生产具体产品间价格上涨-净利润下跌的事理逻辑。如“铁矿石价格上涨->钢铁净利润下跌”、“钢铁价格上涨->汽车净利润下跌”。对多个事理常识生成规则的定义和组合应用，能够生成针对特定产业、场景的可解释的事理常识体系。

- **事件实例之间的事实因果、时空顺承关系的可追溯**

组成事实链条的各子事件的事件主体、事件发生的时间、地点，及其这些要素间的事实关联、语义关联、时空共现或顺承关联，为事件实例间关系的成立和归因溯源提供依据。如图 51 中，2019 年 1 月 25 日淡水河谷溃坝事件、当年 7 月铁矿石价格上涨至高位，宝钢股份、宝钢股份、方大特钢的净利润下跌是有新闻报道、财报披露的事实。通过每个实例事件-概念事件的 belongTo 关系，事理知识层概念事件的 leadTo 关系，铁矿石-钢铁-汽车间的产业链关系，可以清晰的解释，这几个事件不是独立的，而是可被产业链事理关系解释的事实链条。

- **事实关系-事理常识间归纳演绎的逻辑可解释**

事理层面，定义了从抽象到具体的本体概念、概念间的事理逻辑关系，常识关系；在事实实例，定义了实例知识的结构化和语义标准化表示，及事件间的事实关联。事实关系-事理常识的表示方式结耦的同时，用 SPG 的 belongTo、isA、isInstanceOf 等标准谓词，对概念事件到具体事实的演绎，具体事实关系到事理逻辑的归纳提供了统一的表示方法。这种在抽象概念和具体事实间的关联和逻辑解释，能够帮助在具体场景，使用已有事实关系样本对事理关系做正确性验证，及用事理逻辑，辅助对隐藏的事件间因果、顺承关系的挖掘。例如：利用事理模式生成的“钢铁价格上涨-汽车净利润下跌”及融合企业图谱中已知的汽车企业及汽车企业的股权穿透关系，挖掘出有利润下跌风险的汽车企业。

9.2 金融事理图谱 SPG 与 LPG 的对比

表 13 事理图谱场景下 SPG 与 LPG 能力对比

应用阶段	应用的问题	LPG一般解决方案	SPG解决方案
事理定义及传导	事理定义	一般使用业务系统替代	基于派生概念支持事理定义
	事件在事理层面之间传导关系		基于特定推理谓词leadTo描述事理定义
事件定义	事件和实体图谱如何关联	主要存在两种方式 1）按子图定义事件，对建模和数据准备均有一定要求 2）业务系统中直接召回，该方式需要业务系统开发	支持原生事件类型定义，简化建模和数据准备，也无需业务系统开发
事件分类	事件单分类和多分类	一般是建立业务系统，对该事件进行分类打标，或者是不做分类，直接使用分析任务替代分类结论	支持事件到概念的动态分类
	事件分类不可穷举的问题		使用组合概念派生出新概念（抽象事件）
事件传导	发送一个事件如何描述其影响	一般在业务系统进行任务形式进行定制处理	基于事件和概念联动实现事件传导

从上面中，SPG 提供了一种事理表达框架，和 LPG 对比，可有效的将事件传导链路表达清晰，为金融事件影响快速分析响应提供一种新的实践。

9.3 SPG 驱动的黑产知识图谱

第 2 章中提到黑产风险图谱应用的问题，核心在于数据的维护和管理以及理解成本过高，可将第 2 章中提到的点边按照数据生成方式分为两类：1） 基础数据，即来源于原始表数据。如 Person、Phone、Cert、Device、App 等实体，Person-has->Phone、Person-has->Cert 等关系可直接从原始表中转换得出的。2）派生数据，由基础数据或者派生数据生产得出的数据。如 Person-samePhone->Person、Person-developed-App 等由逻辑派生而来。下面分别从问题出发详细论述 SPG 解决方案在黑产图谱上应用。

1. 解决原始数据转换成图谱数据后数据膨胀和成本增加的问题。基础数据和原始表数据相比相差较大。例如，原始表只提供了用户表和应用表，并无设备、证书、电话表，该信息均为用户表和应用表的字段存在，基于 LPG 的构建一般需要用户进行额外数据转换工作，或提供映射操作。SPG 提供标准属性能力可简化用户数据建模，减少数据清洗成本。如下将手机、设备和证书做成标准属性：

```
CREATE TYPE (std.Phone {
        value STRING REGEX '^1([38]\d|5[0-35-9]|7[3678])\d{8}$'
});
CREATE TYPE (std.Cert {
        value STRING REGEX '^[a-f0-9]{32}$'
});
CREATE TYPE (std.Device {
        value STRING REGEX '^([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})$'
});
```

其余点定义：

```
CREATE NODE ( User {
  id STRING,                    //用户主键
  name STRING,                  //用户名
  type STRING,                  //用户类型，自然人 or 法人
  hasPhoneNum std.Phone,        //此处使用标准属性
  hasCert std.Cert,             // 此处使用标准属性
  hasDevice std.Device          //此处使用标准属性
});

CREATE NODE ( App {
  id STRING,
  riskType STRING,              // 风险标记
  hasCert std.Cert,             //此处使用标准属性
  installDevice std.Device      //此处使用标准属性
});
```

由于全部使用标准属性替代关系建模，此处无关系显式定义，只需要导入用户表信息以及应用信息表即可。可以发现，使用 SPG 的建模能力，简化设备、证书等实体的建模成本，也减少用户数据清洗成本。

2. 解决因业务特点不同导致重复数据准备的问题，支持跨业务的图谱复用。黑产图谱需要使用到转账数据和股权数据，SPG 提供图谱融合能力，可将其他场景图谱实体关系引用到本图谱中，通过自定义归一算子，以满足本图谱场景使用。这部分可以来自于已有的资金图谱和股权图谱，例如图 51 所示将资金图谱和黑产图谱融合，其中实例中文字结构为：类型/实例属性名=属性值。

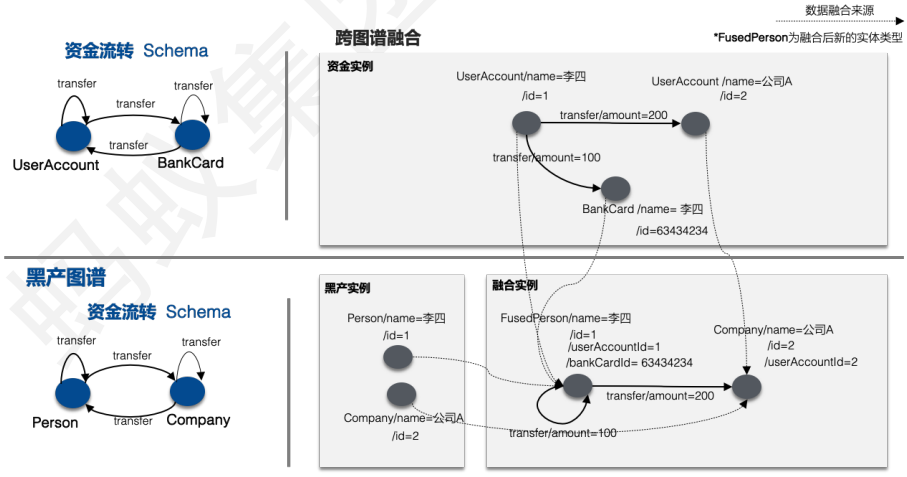

图 51 跨图谱知识融合

FusedPerson 由资金图谱中的 UserAccount 和黑产图谱中的 Person 链指、归一合并而来，要想完成这样的同一关系，需要定义实体链指、实体归一两个阶段。值得说明的是，FusedPerson 只是类型的声明和算子的关联，不生成实际的融合实例，能大大节约计算和存储成本。

1）实体链指。主要定义以什么样的链指算子将 UserAccount 实例和 Person 实例进行对应，可以是一一对应，也可以是多对一，本例中为多对一，只需要基于规则的链指就可以实现跨图谱的实体链接，算子接口定义伪码如下:

```
@BaseOp.register("FusedPersonLinkOp", bind_to="FusedPerson", is_api_iface=True)
class FusedPersonLinkOp(EntityLinkingOp):
  def eval(self, record: Vertex) -> EvalResult[List[Vertex]]:
    pass
```

2）实体归一。本例中基于图谱归一算子实现，基于归一条件表达式规则，对成功映射的 UserAccount、Person 实例实现属性、关系筛选及操作处理，算子接口定义伪码如下:

```
@BaseOp.register("PersonFuseOp", bind_to="FusedPerson", is_api_iface=True)
class FusedPersonFuseOp(EntityFuseOp):
  def eval(
    self, source_vertex: Vertex, target_vertexes: List[Vertex]
  ) -> EvalResult[List[Vertex]]:
    pass
```

3. 解决关系数据之间因存在逻辑依赖带来的不一致问题。SPG 提供派生关系、派生数据能力，如下，以同手机号、同人为例。

```
Define (s:Person)-[p:samePhone]->(o:Person) {
  Structure {
    (s)-[:hasPhoneNum]->(w:std.Phone]<-[:hasPhoneNum]-(o)
  }
 Constraint {  }
}
```

KGDSL 的逻辑规则可以复用已定义的关系类型。如同人关系为两个人同时拥有同手机号和同设备。

```
Define (s:Person)-[p:sameUser]->(o:Person) {
  Structure {
    (s)-[:hasPhoneNum]->(o),  (s)-[:hasDevice]->(o)
  }
  Constraint {  }
}
```

针对于复杂的实际控股关系也可通过 transitive 定义，如图 52 中图谱实例，其中图中文字结构为实例的类型/属性名=属性值。

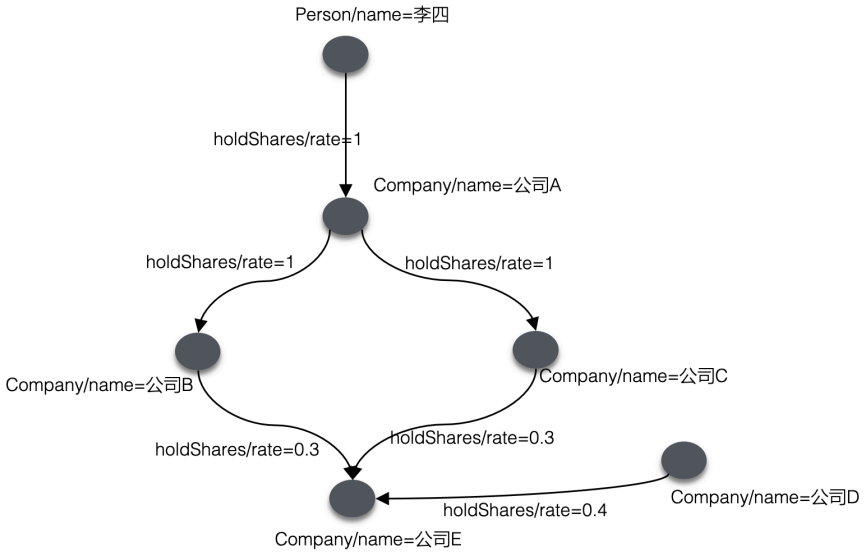

图 52 股权关系实例图

```
//先定义控股比例，transitive 必须要求 GraphStructure 中类型一样
Define transitive (s:Company)-[p:holdShares]->(o:Company) {
  Structure {
    // 图结构中必须是下面结构，表达传递性
    (s)-[p1:holdShares]->(c:Company), (c)-[p2:holdShares]->(o)
  }
  Constraint {
    // 以 s，o 点进行分组聚合，得到所有实际股权
    real_rate("相乘得到实际持股比例") = group(s,o).sum(p1.shares*p2.shares)
    p.shares = real_rate  // 赋值，以控股公司 A 对公司 E 为例，实际份额为 1*0.3 + 1*0.3=0.6
  }
}
Define (s:Person)-[p: indirectHolding]->(o:Company) {
  Structure {
    (s)-[p1:holdShares]->(c:Company)-[p2:holdShares]->(o)   // 通过公司 c，对 o 公司进行间接控股
  }
  Constraint {
    R1("直接控股股权比例必须大于 50%"): p1.shares > 0.5
    R2("间接控股比例必须大于 50%"): p2.shares > 0.5
  }
}
```

依次类推，可基于专家规则将所有关系补全。

4. 克服业务迭代演化导致 schema 及数据持续膨胀最终不可维护的问题

在以往的 LPG 图中，数据和 schema 强绑定，若业务发生变化，则需要 schema 配合改变，这样的改动成本较高，SPG 基于概念提供了动态分类能力，可在概念层进行业务上的扩展，如图 53 所示。

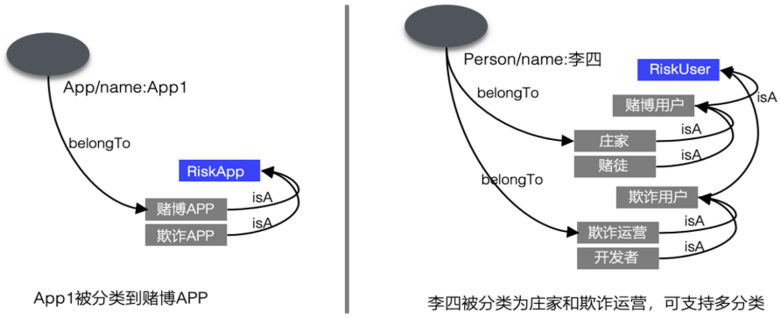

图 53 黑产图谱基于概念的实体动态分类

新增推理谓词 belongTo，将符合规则要求的实体和概念链接起来，比如，当一个 App 确认为欺诈 App，则其开发者为欺诈可疑人员。以 Fraudster 为例，实现动态分类的规则如下：

```
Define (s:Person)-[p:belongTo]->(o:TaxonomyOfRiskUser/Fraudster) {
  Structure {
    // 开发欺诈应用的人为欺诈者
    (A:App)-[:developer]->(s)
  }
  Constraint {
    R1("App 为欺诈应用"): A.type == '欺诈'
  }
}
```

可通过专家规则来描述概念和实体的关联，解决业务和实际数据耦合过于紧密的问题。避免业务发生变动从而更改底层数据，也可尽量避免业务应用层对底层数据发生变化的直接感知。我们使用概念和业务进行强绑定，对于业务人员，可以将概念当成类型进行应用，以上面例子举例，将 Fraudster 当做类型，例如下方式。

```
MATCH
  (u:TaxonomyOfRiskUser/Fraudster)
RETURN
  u
```

其他场景下基于 SPG 图谱的黑产业务应用

1. 直接查询使用

当出现某个 App 被标记成为赌博应用（可能来自用户投诉，也可能来自其他安全事件触发），此时我们要找出该 App 背后团伙，可通过如下查询语句

```
MATCH
   (a:App)-[:developer|boss]->(u:Person)
WHERE
  a.id = '赌博应用 1'
RETURN
  u
```

2. 神经符号的融合学习

将深度学习与规则结合一直是学术研究的热点，也是难点。深度学习可以解决很多表征学习问题，比如图像分类任务；而规则（符号逻辑）能够处理很多显式的推理问题。

现有的神经符号结合的推理主要有两类应用大方向。

1）规则和神经融合方式

从规则先验的角度对这些方法进行分类，可以分为两类：

第一类：用规则约束模型结构，比较典型的方法有 DeepProbLog[23]方法、neuro-symbolic forward reasoning (NSFS)[24]、Logical Neural Networks (LNN)[25]以及 LogicMP[26]等

第二类：用规则约束目标，这里把规则作为一个先验知识，先验加在目标函数上，作为一个惩罚项，比较典型的方法有 SemanticLoss[27]、NCLF[28]等方法

不管是第一类还是第二类，均需要一阶谓词方式作为规则输入形式，本文中提的各类规则可和一阶谓词互相转换，例如

```
Define (s:Person)-[p:belongTo]->(o:Fraudster) {
  Structure {
    (A:App)-[:developer]->(s)
  }
  Constraint {
    R1("App 为欺诈应用"): A.type == '欺诈'
  }
}
```

转换一阶谓词如下：

```
forall s: exits a:  developer(a, s) & type(a) == '欺诈' -> belongTo(s) == Fraudster
```

此外，由于业务专家经验属于 hard rule，很容易导致出现召回率低的问题，LogicMP 中可将具体规则内容进行软化，反向提高规则覆盖率。

2）将符号间的关系表示成图结构，通过图算法进行推理

本例中，可以把用户通过逻辑规则的定义生成的图形式，输入到图算法中进行训练，该方法可以通过图的形式解耦神经和符号两种不同的方法，保证可扩展性和灵活性

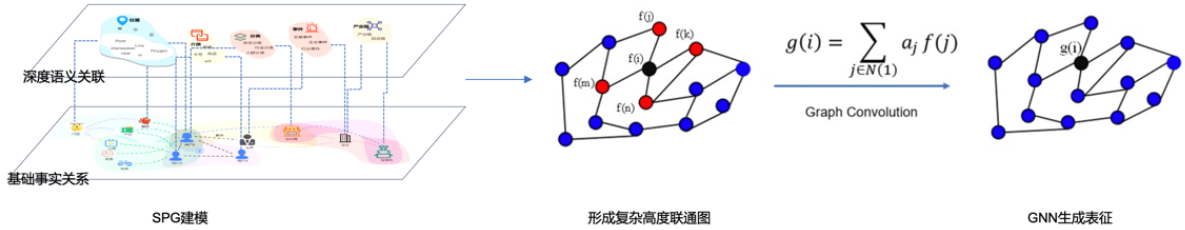

图 54 SPG + GCN 类图算法结合

9.4 黑产知识图谱 SPG 与 LPG 的对比

从知识生产、知识应用、知识演化三个角度分别对 SPG 和 LPG 的存在的优劣势对比。

表 14 黑产图谱 SPG 与 LPG 能力对比

应用阶段	应用方式	LPG解决方案	SPG解决方案
知识生产	点、边、属性导入	提供mapping方案，点/边/属性均需指定mapping	使用语义化建模替代mapping方案，支持Table2SPG；使用逻辑属性/关系替代派生数据导入
	数据一致性维护	定义schema约束，典型方案PG-Key	使用逻辑依赖维护数据的一致性，可有效降低数据不一致的风险
	图谱数据融合	不支持，只能导入一张图或重新导入	支持跨图谱融合，并形成本图谱所需数据，无需用户额外增加数据处理成本
知识应用	图分析	提供图查询语言，但用户需要从基础数据进行编写分析语言	提供图查询语言，可使用逻辑关系、逻辑属性屏蔽复杂底层数据，用户只需关注需要的信息即可，减少用户使用者门槛，也易于维护
知识演化	新增或删除修改实体/关系分类	需要变更schema，且需重新导入数据，变动成本较高	基于概念的类型表达能力，变更概念即可达到实体分类目标，无需变动schema和重导数据。

SPG 在黑产风控类上着重解决了图谱使用者的使用成本问题，分别在知识生产、知识应用、知识演化各个阶段均能有效提示用户使用效率、降低使用成本。

9.5 本章总结

本章主要讲述 SPG 在金融事理图谱和黑产风险图谱的应用。在金融事理图谱中，SPG 提供一种事件表达框架，可有效的描述事件的影响传导关系，最终得到及时有效的结论，补充了 LPG 在事理图谱中应用的不足；在黑产风控图谱中，SPG 主要从用户使用成本和效率出发，解决了 LPG 图在实践中的数据一致性难以保证、图谱演化成本高、图谱理解成本高等问题。

第 10 章 紧跟新时代认知智能的 SPG

企业数字化、智能化升级的未来趋势是基于企业大数据体系积累的海量业务数据构建知识，促进数据知识化。通过业务数据与 AI 体系的有机融合，实现业务智能化。属性图具有与大数据体系兼容的优势，SPG 基于属性图构建核心能力，旨在加速数据知识化、知识与 AI 体系的有机融合。本章节结合前几章介绍的核心能力及两个案例，总结分析 SPG 的优势、不足、机遇和挑战。

10.1 SPG 相比于属性图的 SWOT

综合上文描述，我们从 S (Strengths)、W (Weaknesses)、O (Opportunities)、T (Threats)四象限分析下 SPG 得优势、弱点、机遇与挑战。

- **SPG 的优势分析。**1）SPG 低成本兼容大数据架构，在企业级应用中可以基于业务积累的结构化数据快速构建领域图谱。2） SPG 分级语义模型支持非完备图谱的持续演化，满足工业应用中业务快速落地、数据持续积累完善、技术应用由浅入深的要求。3）SPG 克服了 LPG 语义能力缺失的短板，兼容 LPG 点/边结构有效衔接大数据，SPG 语义增强更好的衔接 AI 技术体系。

- **SPG 的弱点分析。**SPG 还处于成长阶段，能力设计上存在部分妥协也有一定弱点，后期是我们需要持续克服的。 1）动态分类如何实现继承扩展，目前动态分类模型有效解决了类型颗粒度问题，但对应用有一定限制，难以在新的子类下扩展属性。2）需要持续完善主体内置语义结构，目前主体模型内置语义还不够丰富，只有事件主体/客体/时间、概念分层/上下位等定义和约束，如何基于可控生成及可解释推理的诉求，清晰的表达主体内置语义结构，需要结合下游应用持续的完善。3）实例-概念的联动推理模型，目前 SPG 具备了一定的实例到概念的归纳推理能力，但概念与实例的协同传导，概念到实例的演绎推理的能力还有很大提升空间，还需要结合更多类事理图谱的应用持续的优化打磨。

- **SPG 的机遇分析。**1）填补知识图谱企业应用语义框架缺失的空白，RDF/OWL 因其复杂性未有效在企业落地，建设企业级应用的事实标准，方便跨主体的知识语义对齐，更方便促进知识的流通、互通、交换、共享。2）驱动构建知识图谱通用引擎架构，推动图谱技术的平民化、普惠化。每个技术领域的大规模应用都离不开标准化、框架化。像搜索引擎、深度学习、云计算等。3）大模型时代实现图谱与大模型的双向驱动、衔接互补。各企业基于 Transformer 或开源 LLM 可快速孵化/fine-tuning 新的预训练基座，通过 SPG 标准符号有助于在预训练、SFT/RLHF 及推理阶段实现高效的知识注入、提示联想、知识查询等，并形成稳定的范式实现图谱与大模型的联动。同时，通过数据知识化，构建与 LLM 神经网络化知识体系互补对等的符号化世界领域知识体系。

- **SPG 的挑战分析。**1）规模化应用的性能挑战，尤其是在构建阶段，因抽取模型、实体链指较大的性能开销，严重影响大规模图谱构建效率。2）系统能力还需要更多应用打磨，SPG 系统能力还需要结合更多业务和场景持续优化，3）语义化的用户心智培养，一方面需要持续提升用户对语义的理解，另一方面需要持续降低用户对语义的感知。

图 55 SPG 的 SWOT 分析

10.2 第 2 章问题解决情况和遗留问题

表 15 基于 LPG 的图谱知识管理存在的基础问题和 SPG 的解决状态

LPG的典型问题		SPG的解决方案	SPG状态
类型管理	主体语义缺失	事件、实体、概念分类	已支持
	类型颗粒度问题	动态类型	已支持
	属性/关系抉择困难	标准属性、概念类型	已支持
	实体同类不同名	知识融合	已支持
非完备图谱构建	多源数据实体构建	实体链指、实体归一	部分支持
	多图谱知识复用	知识融合	已支持
逻辑依赖	逻辑谓词语义缺失	带规则条件的语义谓词	部分支持
	定义与逻辑分离带来不一致	逻辑衍生	已支持
	窗口类属性/关系爆炸	窗口标准类型，变参属性	已支持
	属性逻辑不一致	逻辑属性	已支持
	关系逻辑不一致	逻辑关系	已支持
事理逻辑	事件逻辑传导受阻	事理演绎时激活新事件实例	已支持
	事件分类	动态类型	已支持
	概念组合语义	基于事件、实体事实的组态组合	已支持

特别说明，表 15 主要列举基于属性图知识管理存在的基础问题和 SPG 的解决状态，主要体现为主体语义及逻辑谓词语义部分，可编程框架和复杂知识推理时构建在良性循环的知识管理框架之上的，不属于知识管理基本能力范畴，未在此表格中列出，但会在第 11 章未来发布计划中进一步描述。

第 11 章 展望 SPG 的未来

本白皮书从企业级知识管理面临的问题出发，介绍了企业级图谱应用因需求范式的变更对知识语义表示与引擎框架都提出了更高的要求。本文第 1 章总结了在知识图谱技术的发展过程中，仍存在着的一些主要问题：

- 缺少统一语义表示。目前，强语义的知识图谱并未实现 RDF/OWL 的工业落地，而弱语义的 LPG 属性图在工业级图谱中却应用广泛。
- 工具多但不统一。为每种数据集定制开发的抽取算法/链指算法、依托图数据库的图谱存储、表示学习工具、模糊检索工具、知识问答工具等，使得知识图谱技术的应用存在着较大的分散性和不便利性。

立足当下，任何复杂技术的大规模产业化应用，都需要统一的技术框架，屏蔽复杂的技术细节以支持新业务的快速部署；都需要可插拔的分层架构，实现领域模型与主体引擎的分层解耦以实现新领域的快速迁移。知识图谱亦是如此，知识图谱的技术发展需要与时俱进，SPG 为强语义的知识图谱定义了工业级易用的知识语义框架，帮助企业进行海量数据知识化的加速构建，通过 SPG 知识引擎统一的技术框架和引擎架构，真正实现知识图谱技术的框架化、平民化、普惠化。

面向未来，知识图谱有着广阔的应用空间。一方面，知识图谱作为结构化数据的最佳建模实践，可以实现机器、算法、工程、业务、运营等不同视角数据建模的统一，可以构建符合 data fabric 思想的下一代数据架构，加速企业级海量数据的知识化，连接数据孤岛，发现更多隐式关联，充分激活数据价值，降低找/用数据的成本，为业务带来更大的增长空间。另一方面，知识图谱强事实、弱泛化、可解释性强、计算成本低、构建成本高的特点，与大模型弱事实、强泛化、可解释性差、计算成本高、语义理解强形成完美互补。未来，期望通过统一的知识符号表示和引擎架构和大模型形成高效的联动和互补，通过大模型技术进一步降低图谱构建成本加速数据知识化，也为大模型的可控生成提供更多领域知识的补充。通过海量常识级领域知识库的建设，加速推进通用人工智能进程。知识图谱与大模型联动互补的实现强烈依赖于完整的知识图谱和大模型技术栈，目前大模型技术已趋于成熟，基于 SPG 所定义的强语义知识图谱框架，有望形成可与大模型无缝配合的知识图谱应用框架，并在未来实现工业级可用的基于知识图谱和大模型的易泛化、高鲁棒、可解释的综合人工智能技术。

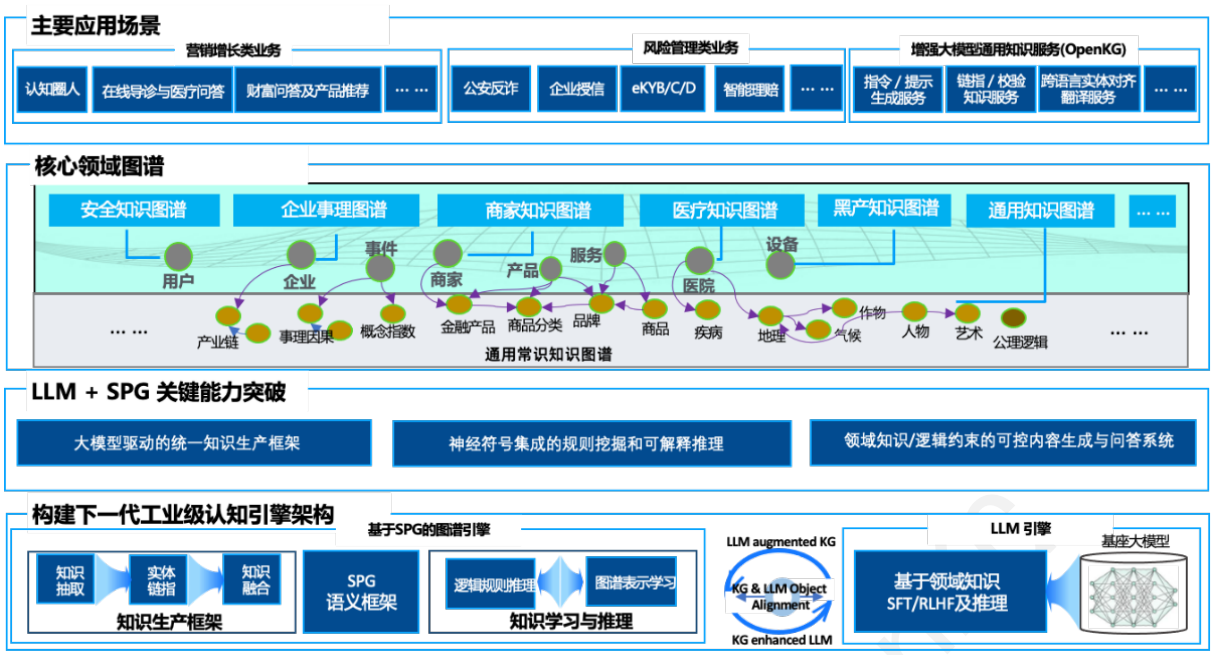

图 56 未来 SPG 与 LLM 双驱技术范式

符号逻辑与神经网络的融合一直是业界的研究热点。其中，一种常见的方法是使用神经网络来学习符号逻辑中的规则和关系，使其能够更好地处理复杂的逻辑问题；另一种方法则是使用符号逻辑来指导神经网络的学习过程，以提高其学习准确性和可解释性。知识图谱作为符号逻辑的典型代表，在结构表示、语义刻画、知识关联等方面有独特的优势。通过 SPG 为其构建的统一语义框架可以为其提供更强的生命力。目前，神经网络与符号逻辑的融合主要发生在知识推理阶段。随着大模型的出现，为符号逻辑与神经网络的融合提供了新的思路。一方面，知识图谱作为符号逻辑的语义表示与知识数据管理的底层支撑，可以借助大模型强大的语义理解能力和知识图谱的强结构、强语义的自动 prompts 和样本构建，帮助知识图谱形成统一的知识抽取框架，加速数据的知识化。另一方面，在内容生成阶段，施加强语义约束的领域知识数据可以有效避免大模型幻觉和胡说八道的问题。这些问题都有望在 SPG + LLM 范式下加速解决。我们还将结合产业实践不断提升 SPG 表达能力，并通过 SPG 增强 LLM 实现客观事实的对齐，有效避免/减少大模型幻觉。同时，LLM 也将增强 SPG，以提升数据知识化的转化效率。我们致力于构建 SPG 与 LLM 相互驱动、相互增强的下一代人工智能引擎。

表 16 SPG 未来发布计划

阶段	发布内容	发布子项
阶段一	《语义增强可编程知识图谱（SPG ）白皮书》V1.0 --- 新一代工业级知识语义框架 介绍SPG知识语义框架的背景、规范、框架、案例等	1.SPG 整体框架 2.两个内容丰富的案例 3.SPG-Schema Core 4.SPG-Engine 的基本要求 5.SPG-controller 的基本要求 6.SPG-Programming 的基础框架
	SPG与欢迎社区成立	1.社区运营机制成立
	SPG引擎单机版 v1.0开源	1.满足白皮书1.0版本的开源引擎建设 2.基于引擎打造开源社区 3.默认适配Tugraph技术栈，支持厂商扩展
	推荐SPG 的标准化【行业、国家、国际】	1.接触各个标准化组织，推进标准化立项
阶段二	《语义增强可编程知识图谱（SPG ）白皮书》V2.0 --- 新一代工业级知识语义框架 介绍SPG可编程框架Operator、SDK、案例等，开放完整的知识构建框架	1.完善的SPG-schema 2.SPG可编程框架的算子、框架完整定义 3.支持映射式【可编程】图谱构建、支持搜索引擎、向量检索、模型服务等 4.基本的推理方案，包含构建、查询和推理的基本的查询语言 5.完整的存储方案【单机】 6.SPG+LLM 的基本要求及在知识抽取/NL2KGDSL的探索
	满足语义和可编程要求的查询语言发布	1.能够满足 SPG 应用基本要求的GQL 或者KGDSL 发布 2.SPG 的增删改查 3.支持基本的自然语言查询
	开源框架V2.0	1.包含白皮书2.0的完整的SPG 框架开源实现 2.查询语言实现模块【GQL 或 KGDSL】 3.可编程 SDK
	发布SPG行业、国际标准	1.标准稿定稿 2.标准稿公示
阶段三	《语义增强可编程知识图谱（SPG ）白皮书》V3.0 --- 新一代工业级知识语义框架 介绍SPG推理引擎规则、图学习、规则引导可解释推理等，开放完整的知识推理能力	1.可编程模块沉淀可复用的model hub，layer hub等，支持开箱即用 2.SPG针对领域，如时空、医疗等的行业扩展 3.规则推理引擎完备语法集 4.SPG与AGL联动支持知识推理、可解释推理等
	开源框架v3.0	1.完全匹配标准稿和 SPG V3.0 2.扩展模块
	基于SPG的常识知识建设和开源方案	依托OpenKG建设常识知识良性增长的开源、众包、服务社区
阶段四	《语义增强可编程知识图谱（SPG ）白皮书》V4.0 --- 新一代工业级知识语义框架 介绍SPG + LLM双驱动的范式，构建可控LLM技术体系	1.SPG + LLM的完整范式和系统框架 2.包含构建、查询和推理的基本的自然语言接口 3.自然语言交互模块【基于 LLM】 4.构建模块支持抽取式构建（基于 LLM）
	基于SPG的常识知识服务	构建增强大模型知识服务，如指令/提示生成服务、链指/校验知识服务等，促进知识要素交换、生长

未来，我们也将持续升级 SPG，表 16 是我们未来计划发布的内容，发布的时间计划会更新在 SPG 公众号：语义增强可编程图谱框架上，欢迎关注、交流，一起探索工业级知识图谱架构范式。

参考文献

[1] Martin, S., Szekely, B., Allemang, D. (2021). The Rise of the Knowledge Graph. O'Reilly.

[2] 王昊奋, 丁军, 胡芳槐, & 王鑫. (2020). 大规模企业级知识图谱实践综述. 计算机工程, 46(7), 13.

[3] 王昊奋, 漆桂林, 陈华钧 (2019). 知识图谱：方法、实践与应用. 电子工业出版社

[4] 中国知识图谱行业研究报告 [OL]. 艾瑞咨询, 2022.

[5] 陆锋,诸云强,张雪英.时空知识图谱研究进展与展望[J].地球信息科学学报,2023,25(6):1091-1105. [Lu F, Zhu Y Q, Zhang X Y.Spatiotemporal knowledge graph: Advances and perspectives[J]. Journal of Geo-information Science, 2023,25(6):1091-1105.] DOI:10.12082/dqxxkx.2023.230154

[6] 王文广. (2022). 知识图谱：认知智能理论与实战. 电子工业出版社.

[7] Colas, Anthony, M. Alvandipour , and D. Z. Wang . "GAP: A Graph-aware Language Model Framework for Knowledge Graph-to-Text Generation." (2022).

[8] 王昊奋, 王萌. "神经+符号"：从知识图谱角度看认知推理的发展[J].中国计算机学会通讯,2020, 16(8), 52.

[9] Yang, L., Chen, H., Li, Z., Ding, X., & Wu, X. (2023). ChatGPT is not Enough: Enhancing Large Language Models with Knowledge Graphs for Fact-aware Language Modeling. *arXiv preprint arXiv:2306.11489*.

[10] Pan, S., Luo, L., Wang, Y., Chen, C., Wang, J., & Wu, X. (2023). Unifying Large Language Models and Knowledge Graphs: A Roadmap. *arXiv preprint arXiv:2306.08302*.

[11] 王文广，王昊奋. 融合大模型的多模态知识图谱及在金融业的应用[j]. 人工智能,2023(02).

[12] Bretto A. Hypergraph theory[J]. An introduction. Mathematical Engineering. Cham: Springer, 2013, 1.

[13] Ferraz de Arruda G, Tizzani M, Moreno Y. Phase transitions and stability of dynamical processes on hypergraphs[J]. Communications Physics, 2021, 4(1): 24.

[14] RDF-star Working Group Charter. https://www.w3.org/2022/08/rdf-star-wg-charter/

[15] 白硕. 事理图谱六问六答 [OL]. 理深科技时评,2019.

[16] RenzoAngles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Bei Li, Jan Hidders, Alastair Green, Leonid Libkin, Victor Marsault,Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savković, Michael Schmidt, Juan Sequeda, Sławek Staworko, Dominik Tomaszuk, Hannes Voigt, Domagoj Vrgoč, Mingxi Wu, and Dušan Živković. 2023. PG-Schemas: Schemas for Property Graphs. In Proceedings of the 2023 International Conference on Management of Data (SIGMOD '23), June 18–23, 2023, Seattle, USA. ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3589778

[17] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Keith W. Hare, Jan Hidders, Victor E. Lee, Bei Li, Leonid Libkin, Wim Martens, Filip Murlak, Josh Perryman, Ognjen Savković, Michael Schmidt, Juan Sequeda, Sławek Staworko, and Dominik Tomaszuk. 2021. PG-Keys: Keys for Property Graphs. In Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China. ACM, 2423-2436.

[18] Munoz-Venegas S , Perez J ,Gutiérrez, Claudio.Simple and Efficient Minimal Rdfs[J].Social Science Electronic Publishing[2023-08-12].DOI:10.2139/ssrn.3199430.

[19] The GQL Standards Website, https://www.gqlstandards.org/

[20] Jana Giceva and Mohammad Sadoghi. 2019. Hybrid OLTP and OLAP. In Encyclopedia of Big Data Technologies. Springer. https://doi.org/10.1007/ 978-3-319-63962-8_179-1

[21] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A {Fault-Tolerant} abstraction for {In-Memory} cluster computing[C]//9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). 2012: 15-28.

[22] Francis N, Green A, Guagliardo P, et al. Cypher: An evolving query language for property graphs[C]//Proceedings of the 2018 international conference on management of data. 2018: 1433-1445.

[23] Manhaeve R , Dumani S , Kimmig A ,et al.Neural probabilistic logic programming in DeepProbLog[J].Artificial Intelligence, 2021:103504.DOI:10.1016/j.artint.2021.103504.

[24] Shindo H, Dhami D S, Kersting K. Neuro-Symbolic Forward Reasoning[J]. 2021.DOI:10.48550/arXiv.2110.09383.

[25] Riegel R, Gray A, Luus F, et al.Logical Neural Networks[J]. 2020.DOI:10.48550/arXiv.2006.13155.

[26] Weidi Xu, Jianshan He, Jingwei Wang, Hongting Zhou, Xiaopei Wan, Taifeng Wang, Ruopeng Li, Wei Chu, An Efficient Mean-field Approach to High-Order Markov Logic. https://openreview.net/forum?id=7UrHaeZ5Ie7

[27] Xu J , Zhang Z , Friedman T ,et al.A Semantic Loss Function for Deep Learning with Symbolic Knowledge[J]. 2017.DOI:10.48550/arXiv.1711.11157.

[28] Zhang Y, Chen X, Yang Y, et al. Efficient Probabilistic Logic Reasoning with Graph Neural Networks[J]. arXiv preprint arXiv:2001.11850, 2020.

蚂蚁集团 x OpenKG