**面向对象程序设计源码分析报告**

**——知识图谱嵌入框架 DGL-KE**

顾丰铭

2018K80099370007

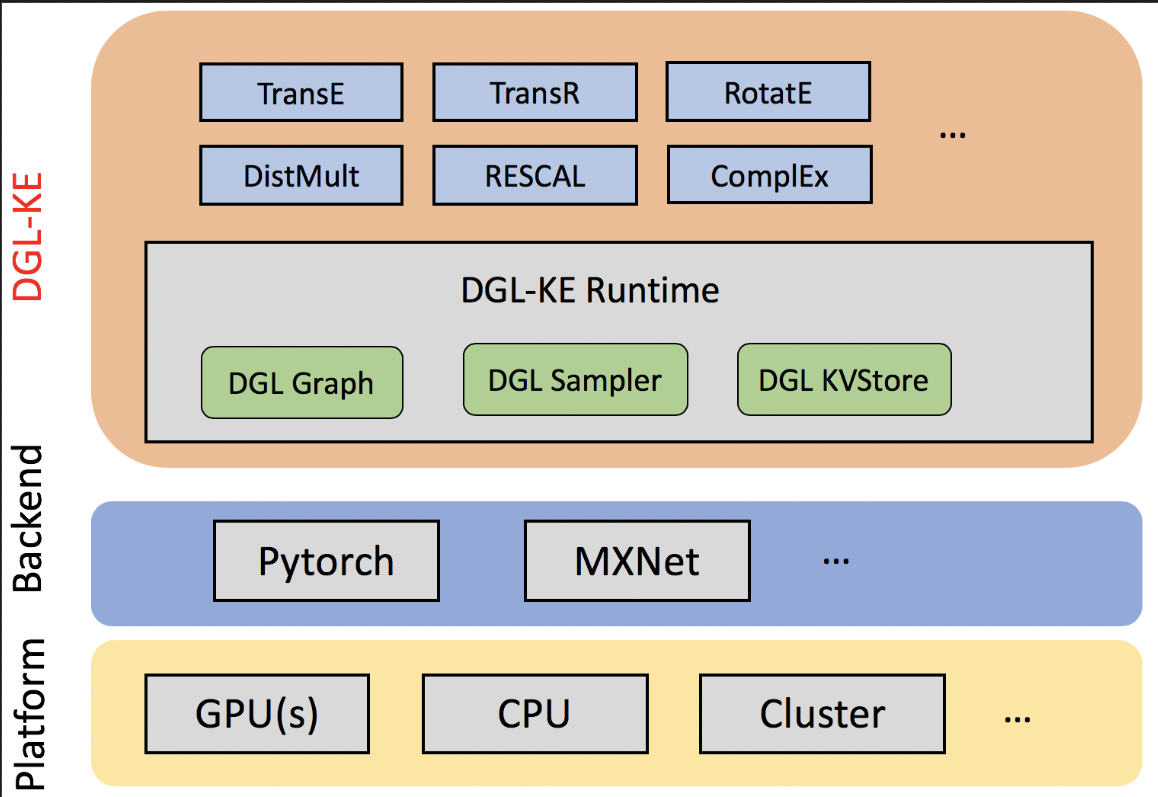
1. **DGL-KE简介**
2. **什么是知识图谱？**

图是用于表示事物及其关系的结构。它由节点集（也称为顶点）和边缘集（也称为弧）。每个边缘本身连接一对节点，表明它们之间存在关系。在同构图中，所有节点表示相同类型的实例，所有边均表示相同类型的关系；在异构图中，节点和边可以具有不同的类型。而知识图谱（KG）则是有向异构图，知识图的顶点通常称为实体，有向边通常称为三元组，并表示为（h，r，t）元组，其中h为头实体，t为尾实体，r为关系将头部与尾部实体相关联。

1. **什么是DGL-KE**

Deep Graph Library（DGL）是纽约大学、纽约大学上海分校、AWS上海研究院以及AWS MXNet Science Team共同开源了一个面向图神经网络及图机器学习的全新框架，这个框架具有良好的性能，能够既快又好地部署深度神经网络。

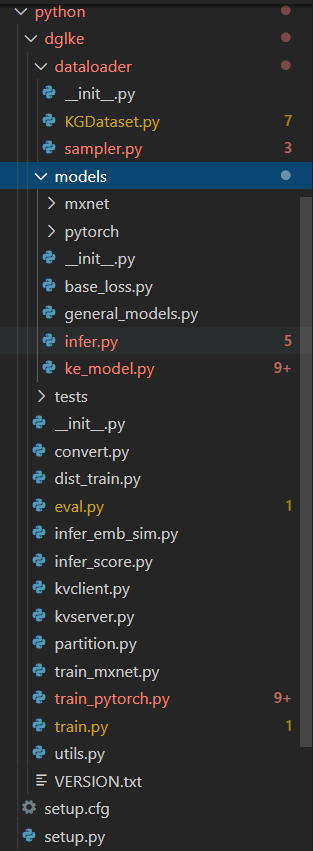
DGL-KE指的是知识图谱的嵌入训练，简单来讲，就是通过从现有图结构中概率性地推断缺失弧来完成知识图的任务。该软件包运行的位置位于DGL的顶部，如图figure1是官方给出的总体架构图。

****

**Figure 1**

1. **DGL-KE的优点**
2. 支持各种主流知识图谱表示学习算法，包括 TransE、ComplEx、DistMult、TransR、RESCAL、RotatE 等；
3. 已有开源框架中唯一支持多核 CPU，多卡 GPU，CPU-GPU 混合训练，以及分布式训练的知识图谱嵌入表示框架；
4. 高性能且可扩展。根据 DGL-KE 发布的 Freebase 数据集 （超过 8600 万节点，3 亿条边）Benchmark 显示，在 AWS EC2 平台上，一台 p3.16xlarge （8 GPUs）可以在 100 分钟内完成训练。4 台 r5dn.24xlarge （4\*48 CPUs）可以在 30 分钟内完成训练，并且达到线性加速比。这一结果比当前最快的同类系统（如 Facebook 发布 Pytorch-BigGraph）快 2-5 倍。
5. **整体框架与主要功能分析**
6. **代码组件与功能分类**

DGL-KE支持在单台或多台计算机的CPU及GPU上进行训练，同时支持推理和评估功能，总体上相对不算很复杂。

****

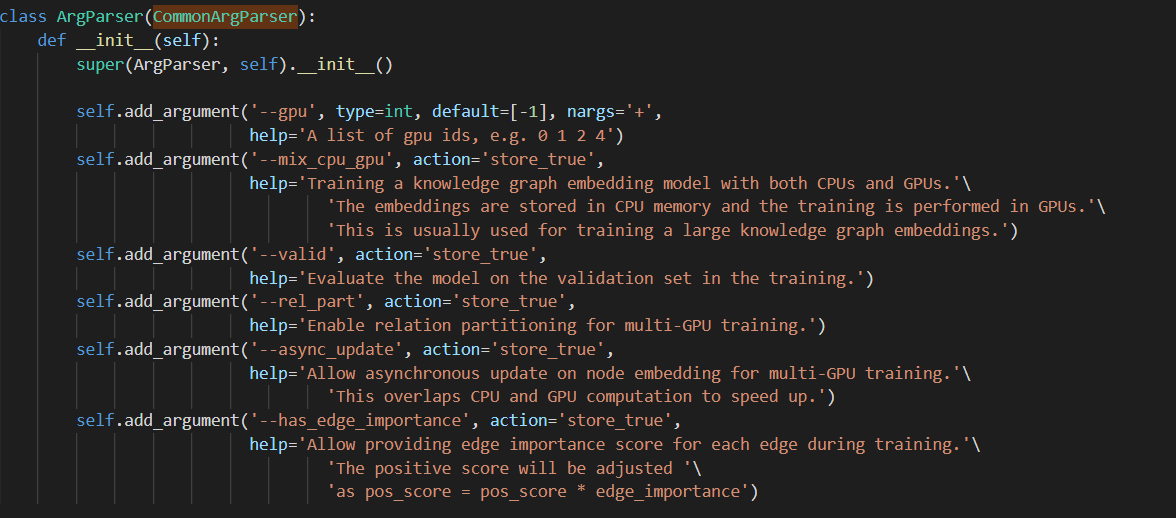
如左图figure2，基本包含了这个项目所有的核心部件。Dataloader目录下主要包括了训练数据集加载和规范化输入的类和函数。Mxnet和pytorch二选一的依赖环境。Infer.py和model.py分别保存推理类和模型类，以import的方式被调用。在主处理函数中，train.py和dist\_train.py用于启动训练功能，infer\_score.py和infer\_emb\_sim.py用于执行推理功能，eval.py用于评分，并在训练结束后进行加载，utils.py用于保存一些通用函数，其他文件基本只负责封装函数和维护服务器交互。总的来讲，对于不同功能，其对外暴露的接口往往比较少，模块之间满足松耦合关系，具体依赖将在下文进行详细阐述。

**Figure 2**

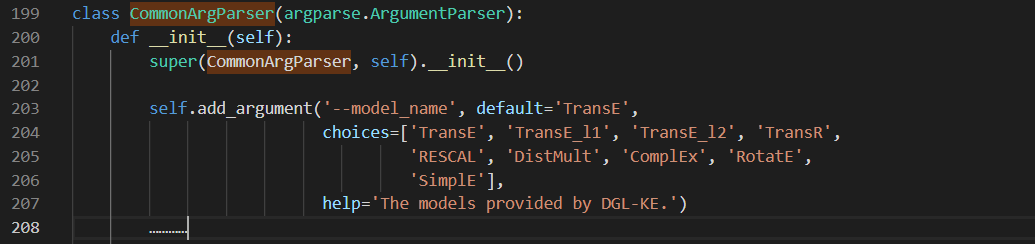
**2.模块间依赖需求分析**

**（1）dgl\_train**

对于训练功能，以非分布式的dgl\_train为例，其需求是根据命令行输入，加载训练模型和训练集，并输出训练后的模型。其中涉及的对象主要包括有：命令行输入，模型，数据集。其实直接这样说还是比较笼统，每一个模块中类的调用方式并不尽相同，而且对于高复用的模块，项目中也设计了一些封装和组合，以追溯一次训练的完整过程为例，希望可以初窥到这个项目中一些面向对象的思想。

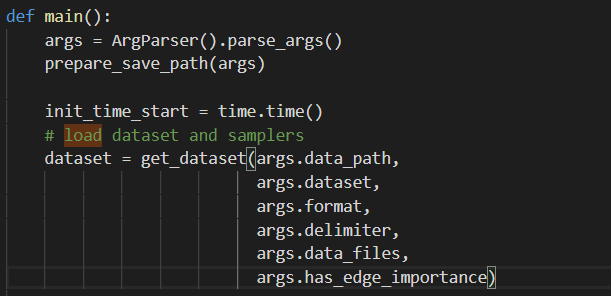


**Figure 3**



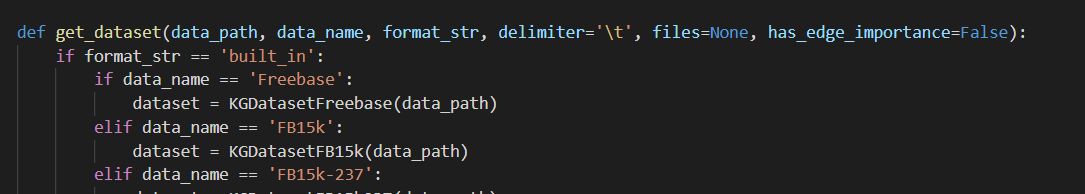
**Figure 4**

对于一次训练，首先需要对命令行输入进行解析，如图figure3，ArgParser继承了父类CommonArgParser，而CommonArgParser继承了python包自带的parse函数，并增添了大量参数属性，如图figure4，这里这样定义一层继承关系的原因是这个新增了大量参数的parser在本项目中被大量引用。



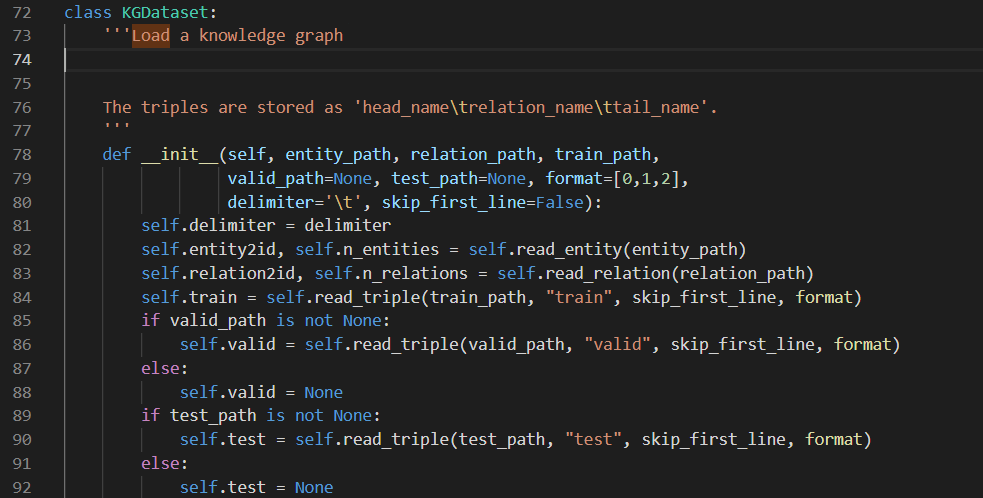
**Figure 5**

在处理完输入后，需要使用get\_dataset载入训练集，如图figure5.

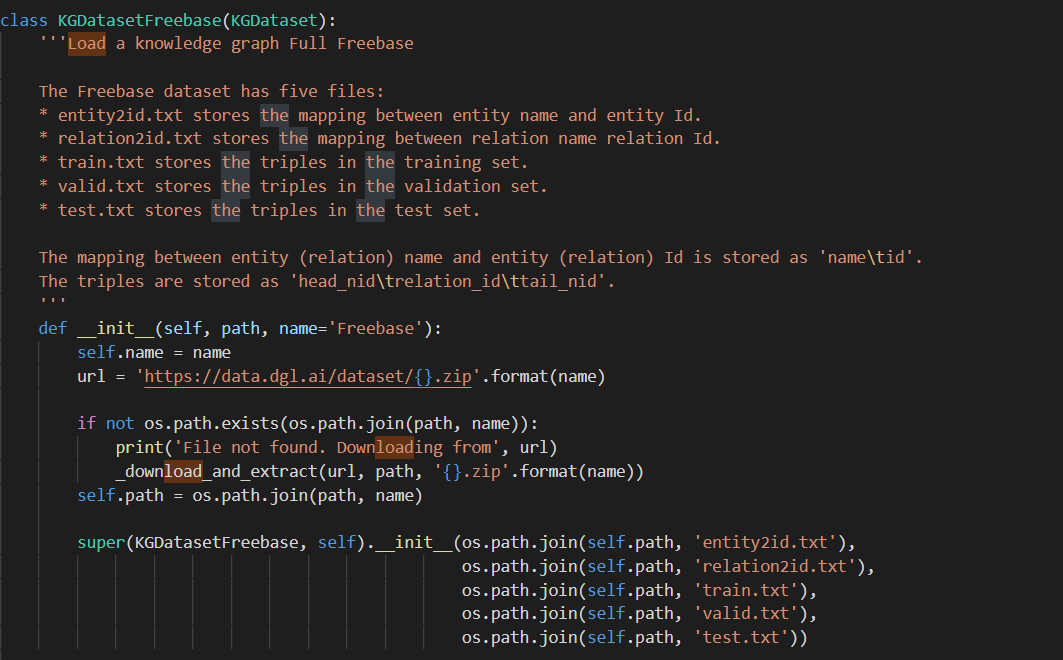


**Figure 6**

追溯get\_dataset，如图figure6，这里使用了判断语句选择数据集的引入，继续追溯数据集是否使用类结构：

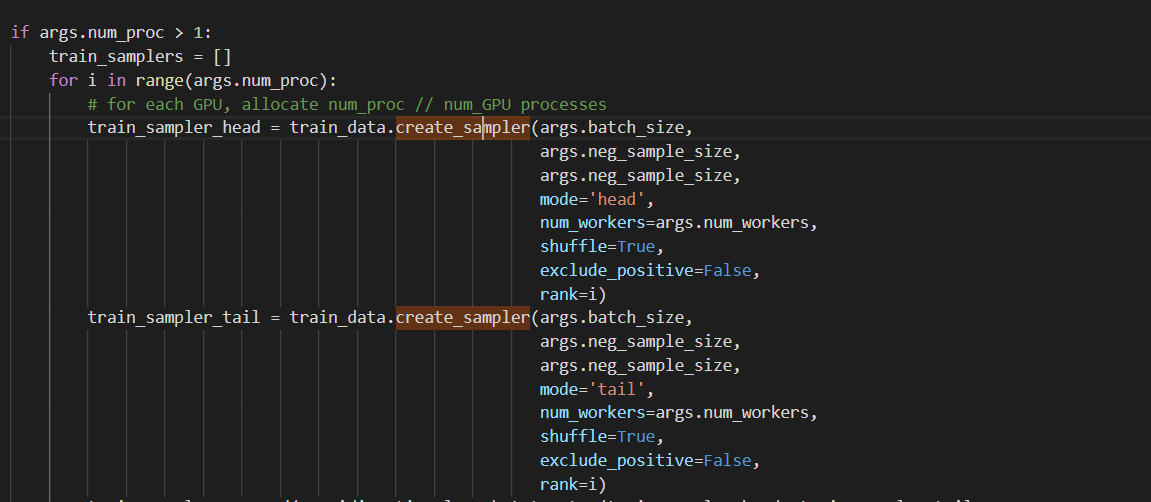


**Figure 7**

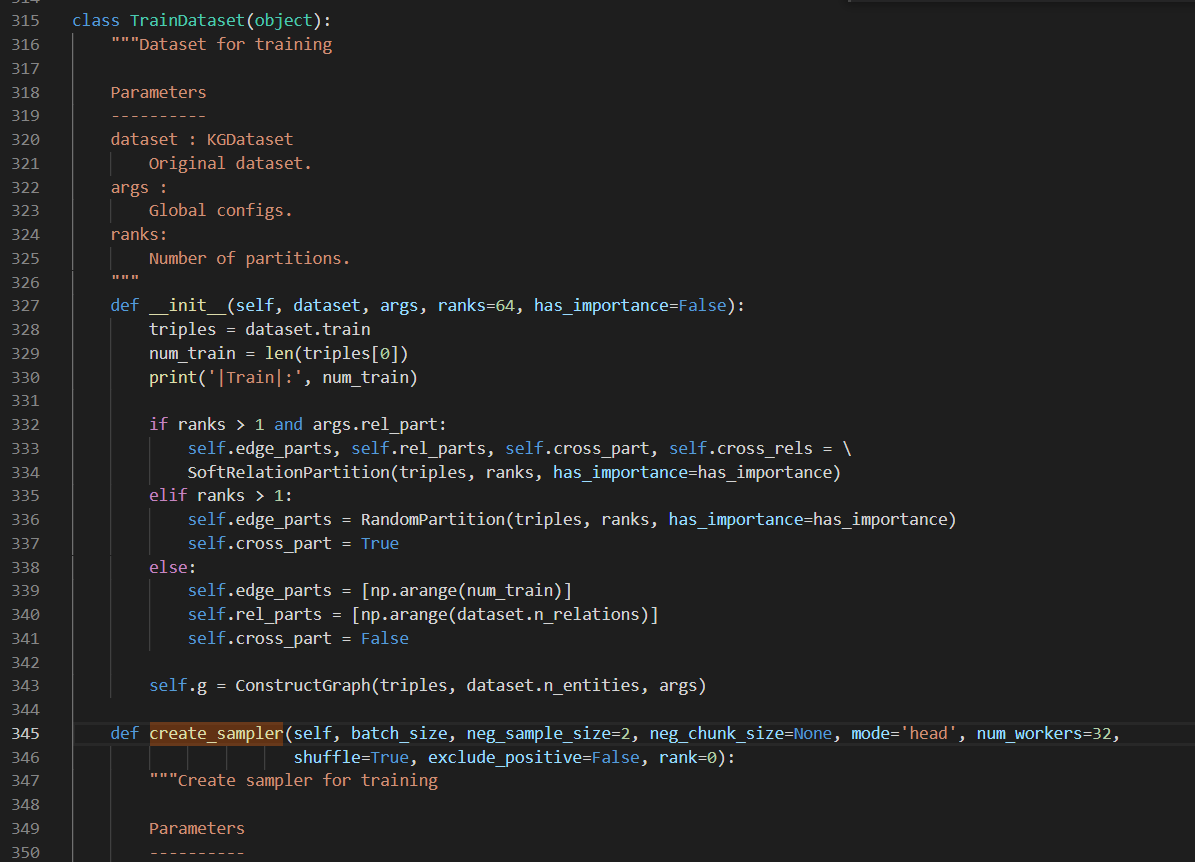


**Figure 8**

如图figure7和figure8，这里KGDataset作为基类，被许多的子数据类继承，呈现出面向对象的核心思想之一——多态。但是，从另一个角度来看，其实这里的KGDataset类很像是一种多继承的接口的概念，理论上可以单独地把它写成一个接口，对于不同的dataset子类，可以独立地重写其中的各种方法实现，但是查阅资料发现，python语法中并没有类似于java中interface关键词类，因此这里仅仅作为一种形式上的近似。

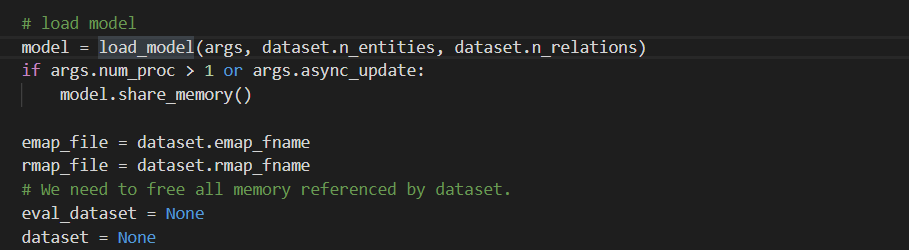


**Figure 9**

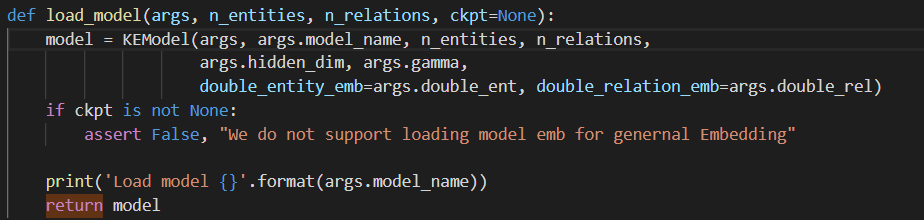


**Figure 10**

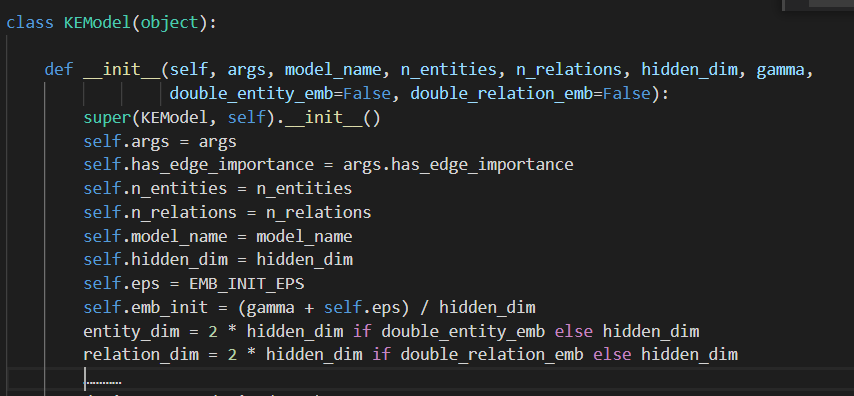
如图figure9和figure10，在获取到数据集之后，需要将其切分成sampler（样例），方便进行CPU/GPU的任务分配。这里的调用关系也很简单，直接调用Traindataset的createsampler方法。Create\_sampler的具体作用是将输入的数据集标准化为可以输入给训练的标准三元组，并进行相关设定，包括内存共享，模型update方式等。



**Figure 11**

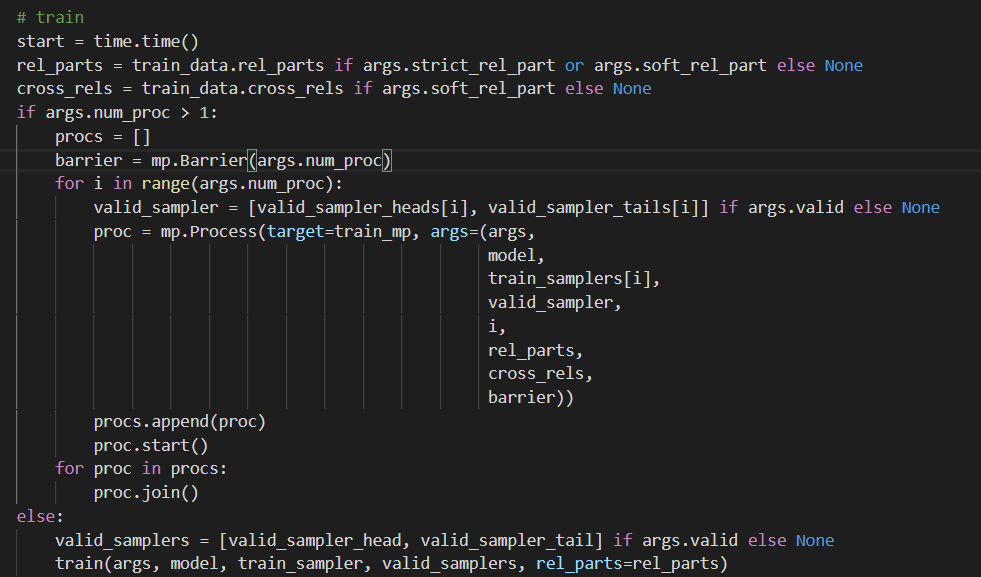


**Figure 12**



**Figure 13**

如图figure11-13，最后还需要加载训练模型，这里loadmodel函数调用了KEModel类进行实现。其实这里的代码风格感觉上更偏向于C语言类似的面向过程方式而不是面向对象方式，如果是规范的面向对象方式，则一定会定义一个Model类，每次调用时通过类中方法进行调用，考虑到Model参数类型是类似的，模型的产生可以采取工厂模式进行。



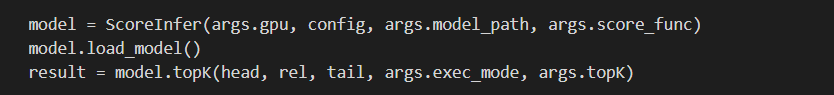
**Figure 14**

如图figure14，数据准备完成，开始训练，train函数同样仅仅是一个函数而不是一个类，具体的训练算法不是我们所关注的重点，不在此赘述。

总的来讲，分析一次训练过程，已经可以发现，整个框架并不是完全面向对象的，随处可见各种工具类函数或者程序入口函数的调用，当然，这和需求本身的过程复杂性有一定关联，但如果后期不断扩展该框架可能会有维护上的困难。

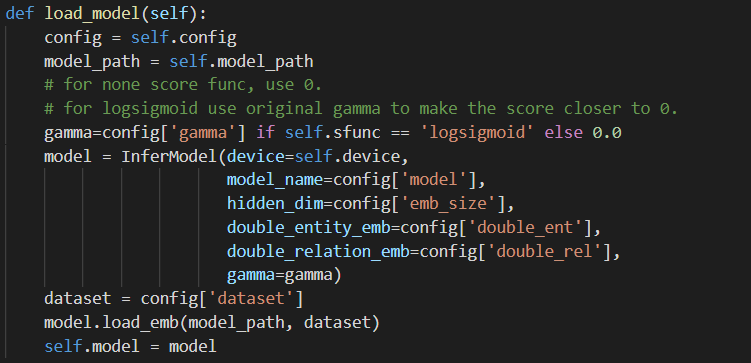
**（2）dgl\_predict**

dglke\_predict的输出是排名最高的候选（h，r，t）三元组及其预测分数的列表。相对于训练过程，预测仅仅需要输出最匹配的前几项。



**Figure 15**

Dgl\_predict同样需要先解析输入，但这里的模型加载方式却和dgl\_train大为不同。如图figure15，这里的输入来自于实例化的ScoreInfer类。



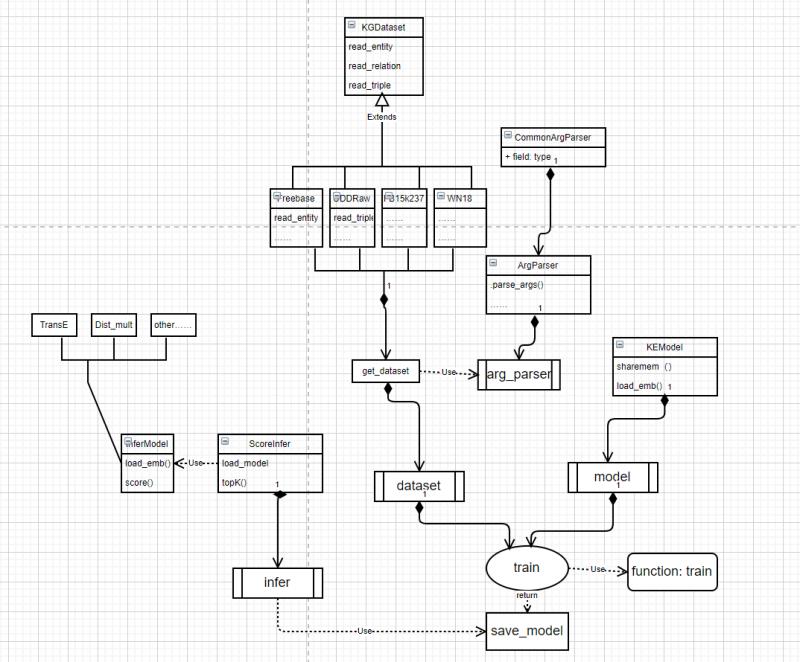
**Figure 16**

如图figure16，load\_model方法中又实例化了一个Infermodel的对象，而ScoreInfer中的其他方法（比如TopK）依赖于这个对象，即可以认为这个InferModel类是与ScoreInfer类形成了组合关系。

回到figure15，最后，调用model对象中的topK方法输出模型中匹配度最高的K个三元组。

总的来讲，dgl\_predict功能的调用关系比较清晰，但是类中方法的构建细节个人认为还有诸多可优化之处，将在下文进行阐释。

1. **核心模块与流程**
2. **类间关系**

****

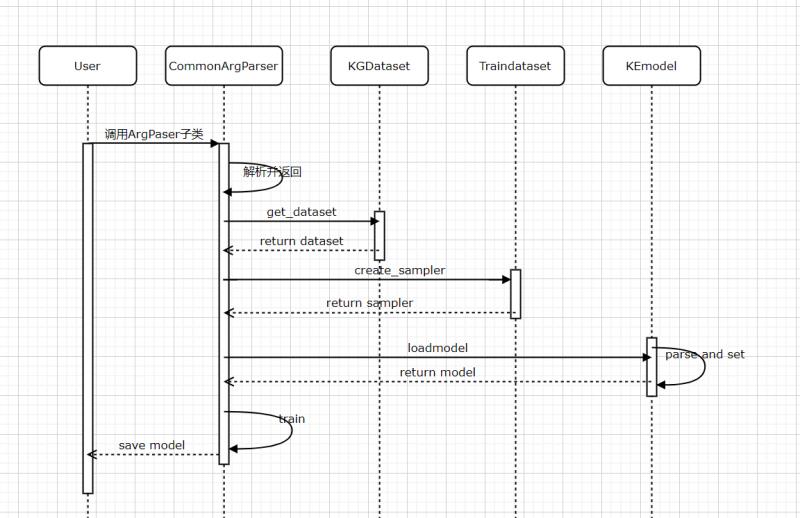
**Figure 17**

如图figure17，是之前所述所有业务的总体类图，可以看到，总体上模块之间，不同功能部件之间呈现松耦合关系，模块内部内聚度高。程序的主要功能被设计成了类（使用python语言），比如比较核心的模型类（dglke/models/ke\_model.py）和infering的代码（gdlke/models/infer.py）等。

代码框架也符合组合/聚合复用原则：尽量使用组合/聚合达到复用而非继承。项目的几种功能几乎完全并行存在，每种功能由各个相应组件，包括模型，数据集，命令行参数等组合/聚合而成。

除此以外，对于庞大的子类，开闭原则，即“面向扩展开放，面向修改关闭”也得到了较好的体现。整个框架的一大特色就是其强大的可伸缩性，对于最顶层的model，dataset等对象，可以很方便地进行扩展，只需要符合接口调用的规则，这样有助于快速适应图神经网络领域的变更而不需要改的底层代码，这也符合所谓的依赖反转。

1. **时序图**

****

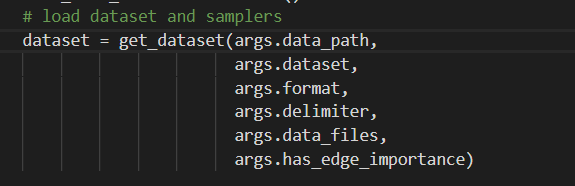
**Figure 18**

如图figure18，可以看到，总体上训练的流程在准备阶段基本是串行执行的，而train的输入则组合了准备的各个组件，在内部进行复杂的并行化运算处理（其中的分布式算法其实是dgl-ke的一大亮点，但是具体过程过于复杂，也没有体现出面向对象的思想，所以这里不展开探讨）。

1. **高级设计意图**
2. **Adapter模式**

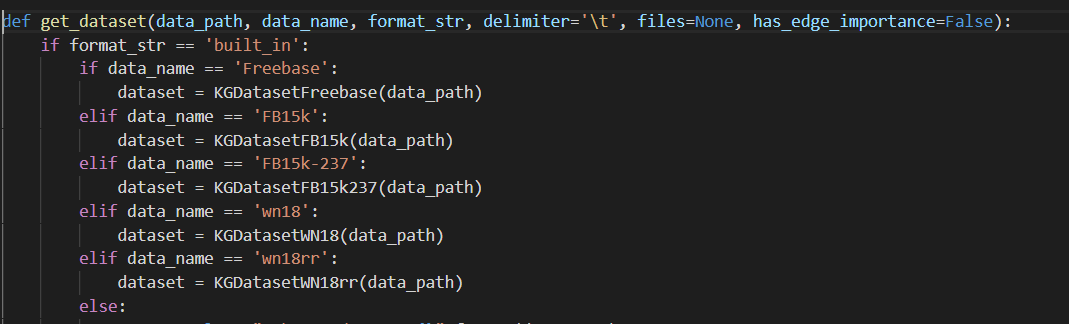
Adapter模式的目的是将一个类的接口转换成客户希望的另一个接口。这样使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

在dgl-ke中，虽然python语法中不支持接口类的修饰词，但是仔细分析依然能找出适配器模式这一设计思想的痕迹。比如下面这个例子：



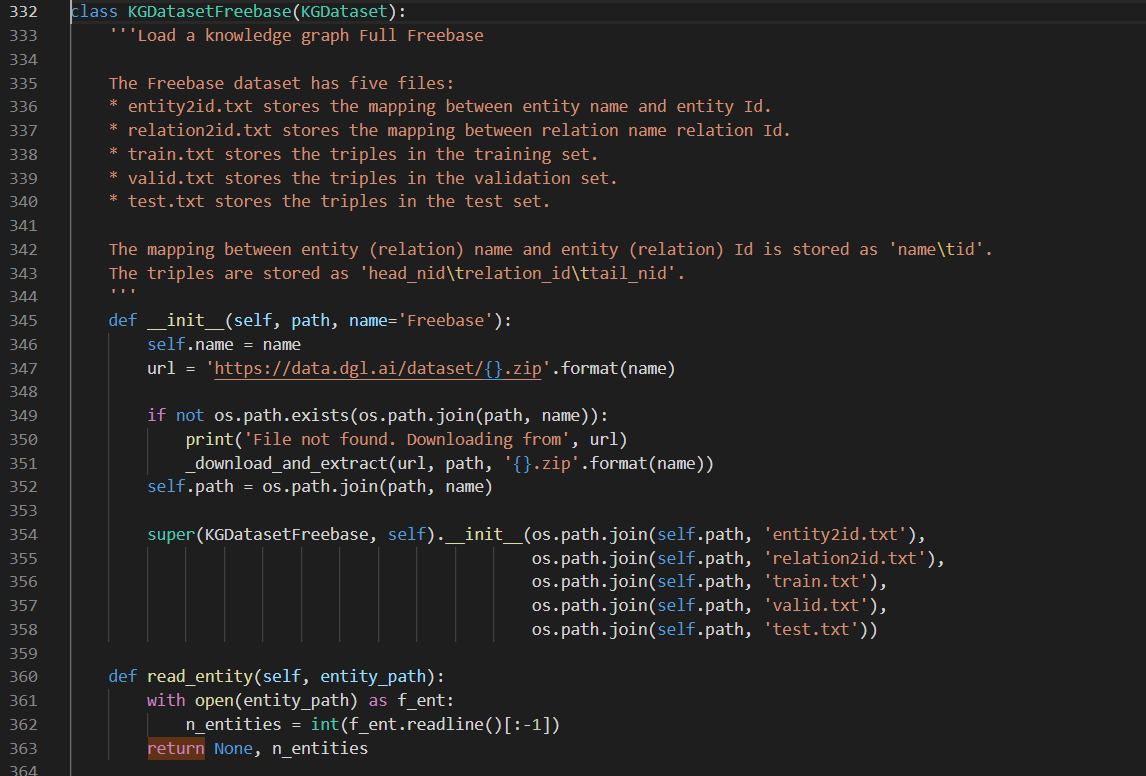
**Figure 19**

如图figure19，dataset是根据arg参数实例化出的某个dataset子类的对象，但是此时的对象，但是此时的输入是以文件路径的形式输入的，但真正训练所需要的数据是以三元组的实体或者ID的形式。

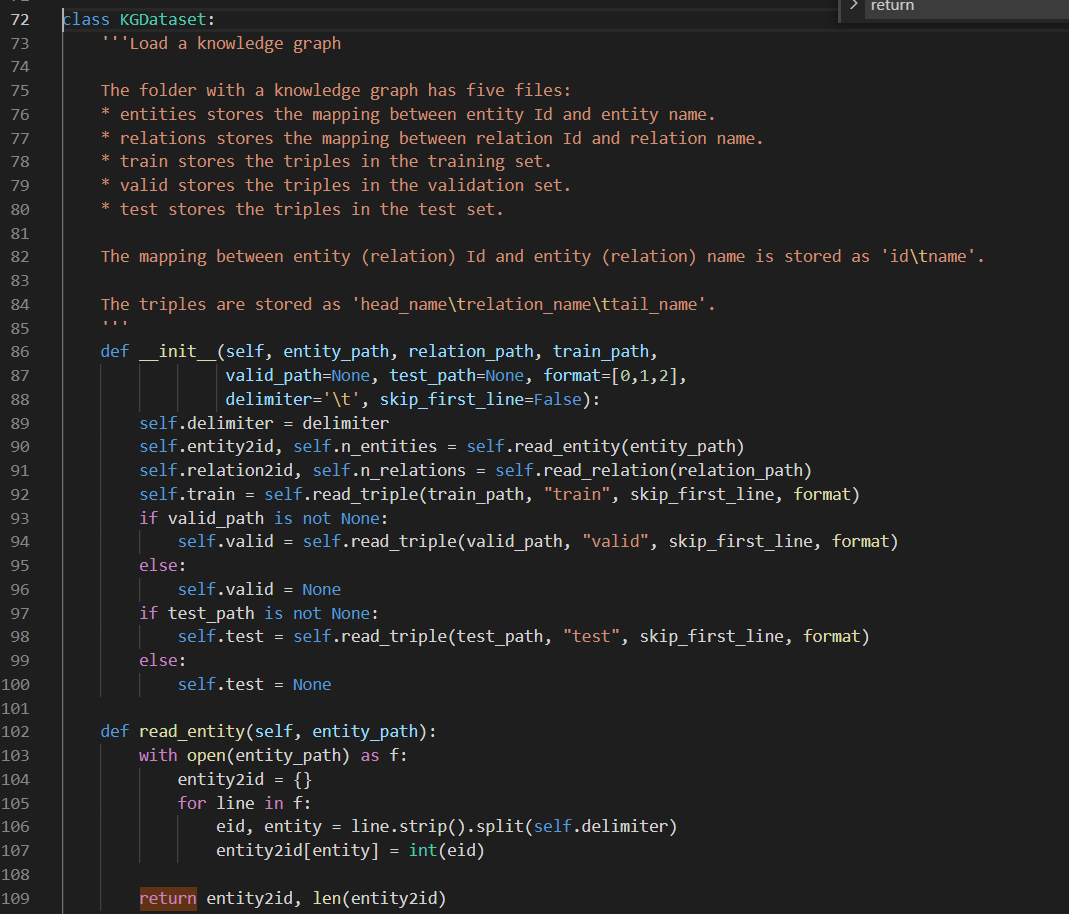


**Figure 20**

如图figure20，追溯get\_dataset的调用类。发现这里是根据参数名去实例化对应的KGDatasetxxx对象。



**Figure 21**



**Figure 22**

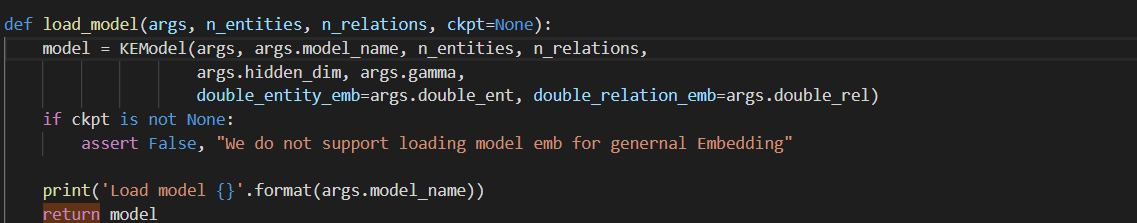
如图figure21和figure22，这里的KGDataset类可以认为是一种adaptee，KGDatasetFreebase子类作为adapter，重定义了包括read\_entity在内的一些行为.这么做的原因是我们希望使用一个已经存在的KGDataset类，但是它的接口（以及方法）并非对于所有数据集都是通用的，对于不同的数据集，将其处理为底层dgl库所能识别的实体类的方法也不同，因此，对于每一个数据集，都采用一个这样的Adapter来进行输入吻合。

即使这样，这里依然有一些不那么完美的地方，比如这里缺失的target接口可能会增加dataset与train输入的耦合度，比如大部分KGDataset的子类其实都不需要重写方法，也许可以将这部分子类直接抽象成一个通用类。

从宏观角度来讲，dgl-ke框架整体本身也可以看作是一个适配器，框架中预留了对外的非封闭接口，可以用于适配用户的训练输入到dgl底层库的转换，大大增加了dgl的可扩展性。

1. **策略模式**

策略模式其实往往是非常常用的设计模式，“策略”提供了一种用多个行为中的一个行为来配置一个类的方法。将每一个算法封装到具有共同接口的独立的类中，从而使得它们可以相互替换是策略模式的核心。在dgl-ke框架中，同样存在大量这样的情况。比如训练用的模型类，对于不同的模型，其区别仅仅是内部的算法差异，模型之间完全是可相互替代的。

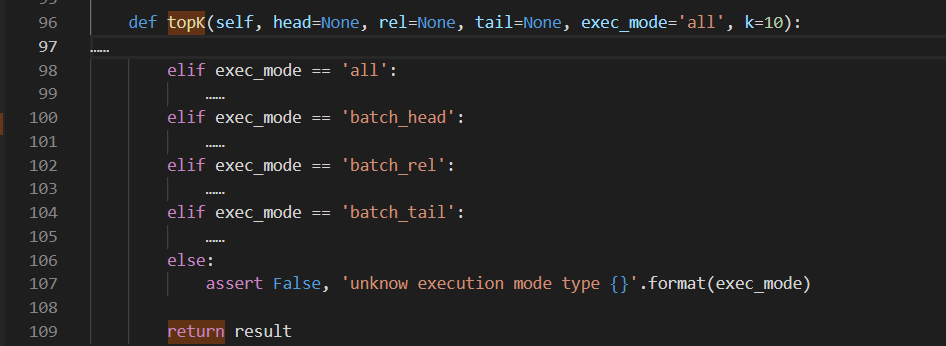


**Figure 23**

如图figure23，这里的KEModel类逻辑上等价于策略模式中的接口，可以调用TransE，Multdist，Rescal，RotaE等不同的算法，但是这里它的做法并不太规范，KEModel类中采用了组合的方式进行罗列各个策略。总体而言，这依然算的上是策略模式的一种实现方式，是的架构总体较为清晰且良好符合开闭原则。

1. **可能的改进**
2. **过程->对象的优化**

之前的分析中已经发现，这个项目的涉及并不是完全面向对象的，直接的例子就是许多方法体和函数都是独立于类而存在的，而有定义的类也往往偏复杂，具体表现有时是一个类中的方法体包含较多的if-else。比如：



**Figure 24**

图figure24中在topK方法中采用了大量的条件选择语句（具体执行过程省略），topK的作用是根据exec\_mode输出result，这里可以使用策略模式抽象一层接口类来实现算法的选择，这样当需要加入新的算法模式时，我们可以仅增加策略子类实现对模块的扩展，使其具有满足那些改变的新行为。

1. **Python中接口类的实现**

接口类Interface中可以定义一些接口名（就是函数名）且并未实现接口的功能，子类继承接口类，并且实现接口中的功能。但python语法不支持interface关键字，可以通过外部模块“abc”进行实现。比如对于上述策略模式的接口可以这样写：

from abc import ABCMeta,abstractmethod

class Exec\_mode(metaclass=ABCMeta):

@abstractmethod

def pay(self,mode):

pass

class batchhead(mode):

def exec(self,mode):

do ……

……

……

p = batchhead()

1. **结语**

到这里，对于DGL-KE这个项目中面向对象的思想分析就基本完成了，之前发邮件给AWS的纪树平老师，他也说这个框架确实不是完全面向对象的，一方面是由于python本身语言的限制，另一方面框架本身目前规模还不算大，完全采用面向对象的思想或许暂时不会对代码的整体架构有很大的提升。在查阅资料的过程中，我了解到Graph Embedding是当下机器学习中一个比较热门且有价值的一个方向，其工具链在最近几年之内也丰富完善得很快，也许随着复杂度不断提高，作者本人也会考虑用面向对象的方法重构代码。

报告中的不足和错误或许存在，非常希望能得到指正和交流的机会，谢谢！

注：DGL-KE开源地址：https://github.com/awslabs/dgl-ke