组件总结

组件化的必要性

在机器学习的全流程中，分析人员的精力应该注重在机器学习流程本身而不是在程序的开发运行上。（为什么？业务场景？）在机器学习程序的编写过程中，大部分功能代码块都是可以在机器学习程序中通用的看（为什么？全部通用吗？）。因此，如果将这些可通用的功能代码以低耦合、高内聚（什么是低內聚？高耦合？例子？）的方式划分为不同的功能组件（现在不是功能组件？），使用者可以通过拖拽计算组件的方式动态编排机器学习程序流程（这是一个解决路径？），便可以降低使用者的学习和开发成本、提高应用创建效率，使其精力主要放在机器学习本身（这是最终目标吗？）而不是程序的开发上。

组件

组件是对逻辑的封装，是软件系统中的功能插件（哪里参考的定义？）。可以将软件流程中一个或者是多个小的功能封装为一个组件，同时也可以将一个功能中的一小部分封装为一个组件，组件封装的粒度大小取决于用户（在自定义组件中）（真是这样吗？有依据吗？）。这里可以将组件的封装粒度分为三类（定义的依据是什么？有量化标准吗？大小的定义是什么？）：

* 粗粒度：封装粒度较大，一般粗粒度可以认为是一个完整的模型，用户只需要输入训练数据来训练模型。复杂度较低，自定义程度低。
* 中等粒度：封装细粒度介于粗粒度和细粒度之间的，既可以定义模型，又有较为合适的复杂度。
* 细粒度：封装粒度较小，也就是语句级（这种就是细粒度？）的封装。复杂度较高，自定义程度高。

在当前组件的自动开发平台上，推荐使用中等粒度的组件化进行组件封装（为什么有这样的推荐？）。

在组件的使用时，不需要考虑组件的内部实现，只需要满足组件的接口规范即可。 组件内部的属性可以通过接口进行自定义。组件具有较高的可重用性，组件的数据接口，具有较高的兼容性（ ？），能够适配大多数的使用情况（如何证明能够适配大多数情况？）。每一个组件在平台上运行时都是一个单独的环境独立运行（为什么要这样？怎么就突然出现环境独立运行了？），互不干扰，并且不同的组件可以有不同的依赖环境（为什么要这样？）。（上面在讲组件的定义、分类，后面要讲划分，这里插播的运行环境很不合逻辑～）

组件的划分

在当前平台中（平台在哪里？），对机器学习代码的组件化是将实现特定功能的方法封装为一个组件（属于哪个粒度的封装？），功能方法的细粒度推荐（什么意思？）等于第三方库中功能方法实现的细粒度（为什么？它们的封装就是细粒度？）。例如将sklearn中的DecisionTreeClassifier封装为一个组件，当需要当前类的实例对象时便可以直接使用当前组件获得实例对象（这与前面细粒度的说明相符？）。（这段跟组件的划分有什么关系？）

在进行功能的划分（功能还是组件？）时，组件之间的数据传输应该是纯数据的方式，不应该存在其他引用或者方法传递的方式（为什么？）。在组件之间传输流动的只能够是数据而非其他（为什么？）。（上述内容均与组建划分无关）

组件化

组件化主要是指解耦复杂的代码时将多个功能模块拆分、重组的过程。（例如？）组件化就是将一个复杂的代码结构分解为多个完成独立功能的组件，组件与组件之间具有低耦合性、组件内部具有高内聚性、组件的边界划分明确（这好像又回到了前面的组件？）。组件是具有一定功能并且能够独立运行的（“的”什么？），将各个组件以数据流的方式前后串联起来便可以组成一个完整的机器学习全流程（怎么又跑到机器学习全流程了？）。组件的开发和组件的使用相对独立（与前文有关系？）。

组件化的Lee（完整形式是什么？需不需要大写？）优点：（写这些优点的目的是什么？）

* 降低整个系统耦合度
* 可维护性高
* 组件重复使用率高，开发效率高

组件化的思路（是想讲机器学习pipeline如何组件化实现？）

在组件化时应该遵循

* 标准化的前提即（“前提即”是什么？），所有的方法在进行组件化的时候应该遵循当前平台的组件化标准（当前平台有标准？）。
* 向外提供服务为基础（谁向外提供服务？），组件是从整个机器学习流程中抽象出来的功能单一且性能较为稳定的部分封装成的公共方法，并将所有的公共方法统一管理，以便于向外提供稳定可靠的计算服务。
* 以组合式应用为目的，软件的核心（后面是“软件”的核心？这里的软件又指什么？）是实现动态的拼接已有的组件形成可执行的机器学习全流程Pipeline，并将Pipeline送入调度器中进行调度运行。（这是要遵循的“标准”？“规范”？）

组件分为哪些类型（前文不是讲过了？）

在整个平台的运行当中组件的类型是可以包含任意类型的。

在实践中，机器学习流程包括一系列的阶段，包括数据预处理、特征处理、模型拟合以及结果验证或预测。例如，将一组文本文档进行分类包括分词、清理、特征提取、训练分类模型以及输出分类结果。这些阶段都可以看作是黑盒过程，并且可以包装成组件。当然，不同的实现方式便是一个不同的组件，实现同一个功能不同实现方式的组件便可以归为一类。（按照前文的分类划分，是否合适？）

组件的来源（应该从去不同视角来总结，如实验应用视角、代码库管理视角、迭代视角等）

在当前开发的平台中，组件来源主要是分为三类，第一类是用户手动编写组件。第二类是扫描第三方开源库，将第三方开源库按照一定的规则进行组件化。第三类是扫描第三方开源项目，将第三方开源项目的源码按照一定的规则进行组件化。

自动组件化的方法（这部分还差太远，先把前面的问题理解清楚吧，细化后再说针对哪种类型的组件自动化生成）

采用自动化的方法，对用户手写的组件方法代码进行组件化代码格式的转换、对第三方开源库对API方法代码进行组件化格式的转换。这种方式主要是可以比较快速的完成代码的转换，在现有工程的基础之上，能够快速的将一个第三方工程集成到平台上。

其他组件化的方式（下面描述的是组件化的方式？下面不是该描述其他类型的组件化方式吗？下面描述的是pipeline实现工具吧？）

* 使用训练好的模型部署，以API的方式提供预测服务，如华为文本识别、文档识别和在线文本翻译等采用训练好的模型进行部署，当数据来时直接进行预测。
* 指定模型，用户自定义训练集来训练模
* Amazon SageMaker for MLOps： 需要使用一定的代码来编排基础的pipeline，或者是使用已有的pipeline；
* Azure machine Learning（AML）:微软在公有云Azure上推出的基于Web使用的一项机器学习服务，其内置了多种基础的机器学习算法，但AML只能够在Azure上使用，不能够单独部署。AML的组件是基于单个算法的组件，一个算法是一个组件。组件划分的粒度是算法级别。
* TensorFlow Hub：将开发好的模型以模块化的方式放在公共的模型仓库中，当需要使用模型时，只需要根据模型的URL进行模型加载即可。

组件的管理方式（管理是如此简单的吗？还是要分吧？组件的基本信息管理、组件对象管理、组件权限管理等）

组件应该分类别的统一放在组件仓库中集中管理，当需要使用到组件时，客户端程序可以从组件仓库中获取组件。当组件需要更新时，只需要对组件仓库中对应的组件进行上传即可。在使用组件进行机器学习流程编排时，可以选择不同版本的组件。

机器学习组件化搭载平台（应该是基于上面的平台进行总结）

在当前平台中，需要一种组件的搭载平台，该平台需要具有以下功能：

* Pipeline的调度功能、调度过程中元数据的管理功能
* Pipeline编排功能
* 组件动态加载功能
* 调度运行Pipeline中的组件时，环境相对独立且隔离
* 分布式调度功能

在满足以上功能条件的基础之上（为什么要满足上述功能条件？），我们找到了Kubeflow和Airflow两个调度框架，这两个调度框架都从不同程度和不同方向上满足了以上功能需求。

AirFlow

AirFlow是一个使用Python语言编写的data pipeline调度和监控工作流平台。AirFlow是通过Pipeline来管理任务流程的任务调度工具。在完整的生产环境中，AirFlow包含以下组件：

* 元数据库：这个数据库存储有关任务的状态信息。
* 调度器：对Pipeline中的组件进行调度。
* 执行器：用于确定实际执行每个组件的工作进程。
* workers:实际执行组件逻辑的进程。

Kubeflow

Kubeflow运行在Kubernetes上。kubeflow具有云原生的特性：弹性伸缩、高可用、DevOps等.其集成了大量机器学习所用到的工具，例如Jupyter、Tensorflow/Pytorch、Pipelines等。

对于当前平台来讲，在Kubeflow中，主要是使用Kubeflow的核心组件Pipelines，Kubeflow的主要目的是简化在Kubernetes上运行机器学习任务流程，并以一套完整可用的Pipeline来实现机器学习全流程。而Pipelines的功能便是对机器学习工作流的编排，能够编译部署机器学习的工作流。

在整体上可以将Pipelines分为八部分：

* Python SDK：用于创建kubeflow pipelines 组件的特定语言（DSL）
* DSL compiler：将Python代码转换成YAML静态配置文件
* Pipeline Web Server：Pipeline的前端服务，用于展示pipeline相关的各种信息
* Pipeline Service：pipeline的后端服务，调用k8s服务从YAML创建pipeline运行
* Kubernetes Resources：创建CRDs运行pipeline
* Machine Learning Metadata Service:监视Pipeline Service创建的k8s资源，并将这些资源的额状态持久化道元数据服务中
* Artifact Storage:存储metadata和Artifact。Kubeflow Pipelines将元数据存储在MySQL数据库中，将工件存储在[Minio服务器](https://link.zhihu.com/?target=https%3A//docs.minio.io/)或[Cloud Storage](https://link.zhihu.com/?target=https%3A//cloud.google.com/storage/docs/)等工件存储中。
* Orchestration Controllers：任务编排，比如Argo Wrokflow控制器，他可以协调任务驱动的工作流。

综合Airflow和Kubeflow的具体用途，可以有如下几点对比：

* 在组件调度的过程当中，Airflow的组件需要组件之间依赖的第三方包版本、环境一致，不能够实现每一个组件都是运行在一个独立的环境当中。
* Airflow是一个通用的任务编排平台，而Kubeflow特别专注于机器学习任务，两者都是用Python定义Pipeline和组件，但是Kubeflow是在k8s上进行任务运行，对分布式节点的调度更加的方便快捷，不需要进行额外的配置。而在对Airflow进行分布式运行时，其数据需要同步到所有的worker节点，但Airflow不提供数据同步的功能，此功能需要用户自己实现。
* airflow在Pipeline调度执行期间对组件的日志监控不如Kubeflow对日志监控全面。kubeflow还可以在日之内绘制图表等。

综合来看，使用kubeflow作为平台的搭载和调度工具较为合适。（是因为kubeflow简单？已实现了很多功能？）