基于Kubeflow的机器学习服务组件化开发方法

# 引言

机器学习是一个近年来迅速崛起的领域，得到了广大开发人员的关注。然而，开发出一个新的机器学习模型并不容易，首先需要深厚的机器学习理论知识，其次需要大量的训练数据（对于深度学习，尤其如此），最后，训练一个模型往往需要消耗大量的运算力，特别是对GPU的要求非常高。

相对于专业级的数据科学家，非专业级数据科学家一般不具有数据科学类学科的专业背景，编写代码的能力较弱，不参与数据科学的全流水线活动，而是利用自己在某一领域的知识和经验优势，主要借助数据科学工具完成数据科学流水线的某一或少数活动。近年来随着数据科学进入各个领域，非专业级数据科学家的数量逐步递增，非专业级机器学习可视化流程编排工具的需求随之而来。因此需要对机器学习流程进行组件化的开发，并以组件的形式向外提供服务以简化机器学习流程。

当然，机器学习也属于软件设计领域，所以采用组件化模式也是自然而然的方法。机器学习可以在三个层次上进行模块化：

* + 1. 应用层：应用层可以提供高度封装好的功能接口，使用者甚至无需任何机器学习背景知识，对外接口也可以看不出采用了机器学习的方法。
    2. 中间层：在中间层可以提供训练好的模型，使用者可以在项目中直接加载模型，也可以在自己的数据集上对模型进行再训练，实现个性化需求。这种方法不再需要海量数据，就可以获得不错的性能。
    3. 框架层：框架层可以提供梯度递减算法、softmax、卷积运算等等，这些是构建深度学习必要的基础组件，但开发者必须设计模型，需要有深厚的机器学习背景知识，以及大量的数据。

组件所在的层级越高，接口越简单，使用越方便，但灵活度越低，只适合非常通用的场景。层级越低，灵活性越高，但编写和训练模型就越复杂。所以处在中间层级的模型组件化能够在复杂性和灵活性之间取得很好的平衡。

基于Kubeflow的机器学习服务组件化开发方法主要是在机器学习服务组件化开发的基础之上，使用Kubeflow作为所开发组件的载体，向外提供服务，同时使用Kubeflow作为组件流程编排工具和组件调度工具，在机器学习中，将组件以pipeline的方式进行编排，编排完成之后输入Kubeflow对编排好的流程进行调度和执行组件并返回最终的结果。在流程调度过程中，数据的传输以及元数据的管理也是使用Kubeflow作为载体进行管理。

# 机器学习服务组件化开发方法

## 机器学习服务

服务性导向的机器学习（Machine Learning as a Service, MLaaS）是自动化和半自动化云平台的一个概括性定义，这个云平台范围很广，包含数据预处理、模型训练、模型评估以及预测。而且预测结果可以通过REST API与其他IT系统进行交互。其旨在以接口的方式向外提供功能调用，简化机器学习流程，降低机器学习使用门槛。亚马逊机器学习服务，微软Azure机器学习和谷歌云AI是三个领先的云MLaaS服务，在只有少量甚至没有数据科学专业知识的用户也能够利用这些服务进行快速模型训练和部署。

## 组件化与模块化

组件：组件的最初的目的是代码重用，功能相对单一或者独立。在整个系统的代码层次上位于最底层，被其他代码所依赖，所以说组件化是纵向分层。

模块：最初的目的是将同一类型的代码整合在一起，所以模块的功能相对复杂，但都同属于一个业务。不同模块之间也会存在依赖关系，但大部分都是业务性的互相跳转，从地位上来说它们都是平级的。

总体而言，组件化开发和模块化开发实际上有各自的适用领域。组件化开发更多被应用于技术底层的实际实现，而模块化开发则需要结合实际的业务功能。组件化和模块化的具体区别如表1。在项目的实际开发过程中，两种开发模式往往是并存的，结合使用的。

表1 组件化和模块化的具体区别

| **类别** | **目的** | **特点** | **接口** | **成果** | **架构定位** |
| --- | --- | --- | --- | --- | --- |
| 组件 | 重用、解耦 | 高重用、松耦合 | 无统一接口 | 基础库、基础组件 | 纵向分层 |
| 模块 | 隔离/封装 | 高内聚、松耦合 | 统一接口 | 业务框架、业务模块 | 横向分块 |

## 服务的组件化

当机器学习平台需要向外提供服务的时候，为了划分不同的功能对外部提供服务，所以需要采用组件化的方式对不同的功能进行低耦合的开发。组件化的开发可以使得机器学习流程中的服务都是以单独的方式向其上下游提供服务，不会依赖于其他组件，组件之间的运行互不影响。

## 组件化的开发方法

组件化的开发主要是将一定细粒度的机器学习方法封装为一个组件，封装好的组件能够直接的向外提供服务。将机器学习中需要执行和调用的方法封装为各个组件，每一个组件有其特定的功能。组件是最基本的能够重用的功能封装，不需要统一的接口，因此在开发上没有过多的要求。

# 基于Kubeflow的机器学习服务组件化平台架构

Kubeflow旨在支持多种机器学习框架运行在 Kubernetes 之上，比如 Tensorflow, Pytorch, Caffe 等常见框架。它包含了 operator、pipeline、超参数调优、serving 等诸多模块。它通过提供对应的 operator，基于 Kubernetes 的 Pod/headless Service 等基础资源为框架提供与之相配的更高层次的资源。比如 tf-operator 为 Tensorflow 提供了 job 维度的生命周期管理能力，以满足 Tensorflow 分布式训练的资源和拓扑需求，达到了一键式部署 Tensorflow 训练任务的效果。

机器学习服务组件化平台主要是使用Kubeflow的pipeline功能，Pipeline是一个基于Argo实现了面向机器学习场景的工作流项目，提供机器学习流程的创建、编排调度和管理，还提供了一个Web UI。

pipeline是由多个组件(Component)组成，每一个组件都可以是以往标准机器学习代码中的一个方法。该平台采用组件堆叠的方式动态的组成一条pipeline，一条pipeline就是一个机器学习从数据处理到模型训练的全流程。全流程中不同的功能由不同的组件来实现，pipeline的构建者可以动态的选择不同的组件来以不同的方式完成同一个步骤。整个平台的架构图如下所示：



基于Kubeflow的机器学习服务组件化开发平台架构

在整个平台的架构中，从外而内可以分为pipeline可视化编排层、pipeline解析层和Kubeflow执行调度层。在调度层之后还有pipeline结合的adapter适配器层，组件开发和生成的组件生成层。每一层的具体功能和设计如下所示：

* 1. Pipeline 编排层：可视化图形的方式或者是脚本编写的方式对机器学习流程进行编排。通过拖拉组件的方式进行机器学习流程进行编排，并生成pipeline。生成pipeline之后传递到后端进行pipeline的解析工作。
  2. Pipeline解析层：在接收到Pipeline可视化编排层编排好的Pipeline脚本后，解析工作便Pipeline解析层就开始对脚本进行解析，其解析工作是将Pipeline脚本转换为Pipeline开发SDK标准的python脚本代码。最最后运行Python代码生成Pipeline的yaml文件，并将文件上传到Pipeline仓库中进行存储。
  3. Kubeflow调度中心：在Kubeflow调度中心中，需要对pipeline执行时，使用pipeline的管理界面从pipeline仓库中拉取需要运行的pipeline并创建一次运行，运行起来之后pipeline的所有操作数据和返回数据都存储在元数据管理中心。
  4. 组件仓库：组件仓库是对组件进行统一集中管理的地方，所有生成的组件都上传到组件仓库。pipeline编排层会在组件仓库拉取组件生成pipeline，Kubeflow调度中心会在组件仓库拉取组件进行调度工作。
  5. 组件生成层：组件的生成主要是有三种方式，其最终目标都是生成一个功能组件的yaml描述文件，三种组件生成的方式如下所示：
     1. 人工组件开发：人工使用SDK对机器学习功能组件进行开发，开发完成之后手动编译得到组件的yaml文件并上传至组件管理中心。使用人工开发的方式速度较慢，但人工直接按照标准的组件开发方法进行开发，组件出错的概率小，组件的质量能够得到保证。
     2. 对第三方库进行转换：对第三方的库代码进行转换有两种方式，第一种方式是直接对第三方库代码的源码进行转换，获取到源码之后直接扫描所有源码将源码的每一个方法按照组件开发标准进行转换。这种方法需要将当前转换方法所调用的方法都递归的添加到当前方法的方法体中，此时会出现递归深度过大的问题。第二种转换方式是在第三方库的API方法外使用一层方法进行调用包裹，之后再对包裹方法进行转换，这种转换方式只需要引入第三方库，不需要递归的将API实现方法内所调用的方法递归的引入进去，因此转换速度快，不会出现递归深度过大的问题，但API方法的识别便是第二种转换方法的一个难点。两种转换方法转换完成之后对转换之后的代码进行编译得到组件的yaml文件，将yaml文件上传到组件管理中心进行集中管理。