视频结构化系统Worker设计

术语解释

Model

是做Inference所需要素材，包含了模型文件，label.txt等文件，也包含相应的子程序。

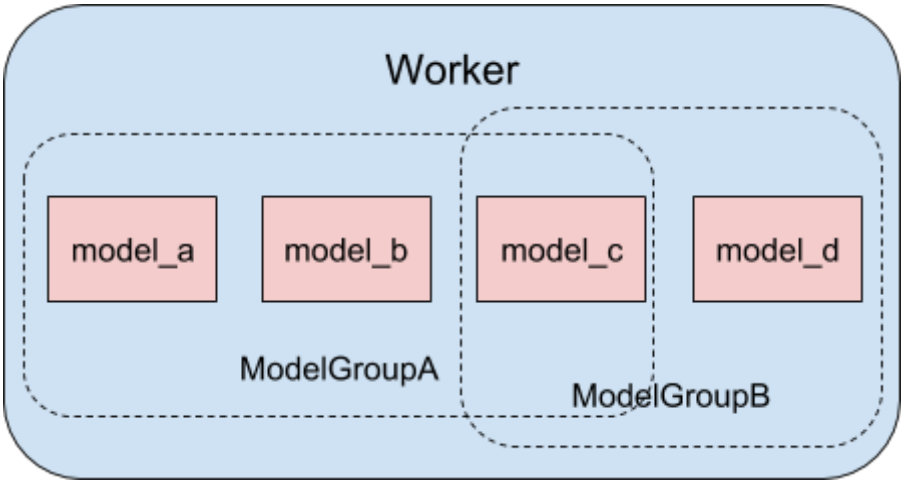
ModelGroup

是Model的集合，是为了特定的任务需求绑定在一起的Model，为了MessageQueue传递消息而对ModelGroup设定Topic消息。

Worker

是分布式系统上任务的具体执行者。 在启动Worker时会加载多个Model，这些Model根据配置组合成一个或多个ModelGroup。组合完成后Worker将监听这些ModelGroup对应的Topic消息。

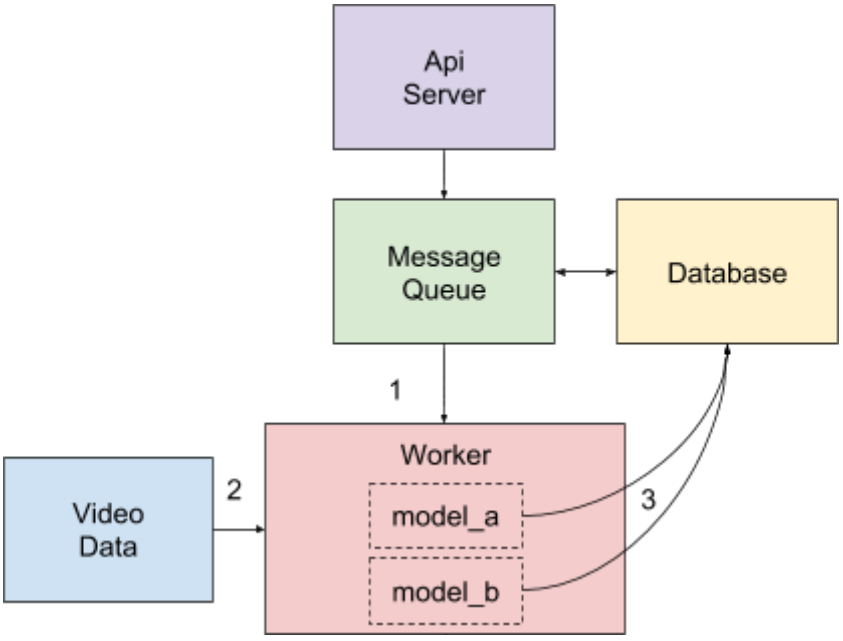
Model/ModelGroup/Worker概念的关系



Worker的工作流程

在系统设计中Worker会根据ModelGroup(s)的配置监听相应的Topic的消息，在ApiServer获取了video request后，MessageQueue将分解video request到ModelGroup这个层级，发送消息给监听该Topic消息的Worker。Worker在接受到消息后将在其内部建立task并写入TaskQueue

，按照TaskQueue顺序调用VideoData和Models，Models各自将其Inference结果写入数据库中。



1. MessageQueue向Worker传递的消息，Worker获取相应要处理的model_id(s)和video_path，将其写入TaskQueue；
2. Worker根据从TaskQueue中取得task配置信息获取VideoData；
3. Worker的各model执行任务做inference，并各自将结果写入数据库；

MessageQueue消息传递

由于Topic对应的是ModelGroup，因此在监听到消息后，Worker需要根据消息的内容创建task。MessageQueue传递的消息的proto：

```
message TopicMsg {
  request_id = 1; // 写数据库需要
  repeated model_ids = 2; // 需要使用哪些模型
  video_path = 3; // 视频文件路径
}
```

Inference结果输出

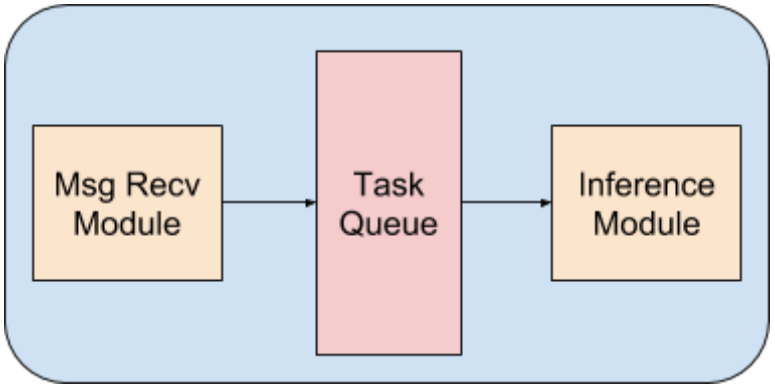
Inference的结果由Model这个层级直接输出写数据库，因此该输出的结构化数据的通用proto为：

```
message Box {
  int32 x = 1;
  int32 y = 2;
  int32 width = 3;
  int32 height = 4;
}

message Detail {
  repeated string class = 1;  // 分类结果
```

```
  repeated float confidence  = 2;  // 分类结果的置信度
  Box bbox = 3;  // bbox检测结果，只做分类则不存在bbox这个field
  string id = 4; // 如果做识别或者跟踪，则具备有在视频全局的id或特殊id
}
// 关键帧信息
message FrameMetadata {
  int64 frame_num = 1;
  repeated Detail details = 2;
}
// 模型的输出，由关键帧信息组成
message ModelOutput {
  repeated FrameMetadata metadatas = 1;
}
```

Worker功能模块划分



针对视频文件处理，在Worker内部形成一个生产消费模型，MsgRecvModule写task到TaskQueue，InferenceModule从TaskQueue获取task做inference，输出写入数据库。

TaskQueue

一个Task的FIFO数据结构，MsgRecvModule接受到MessageQueue消息后生成TaskInfo，Put到Queue中，InferenceModule则在处理完上个任务后从TaskQueue中get新的TaskInfo。在TaskQueue上实现生产消费模型，保证TaskQueue线程安全。

MsgRecvModule

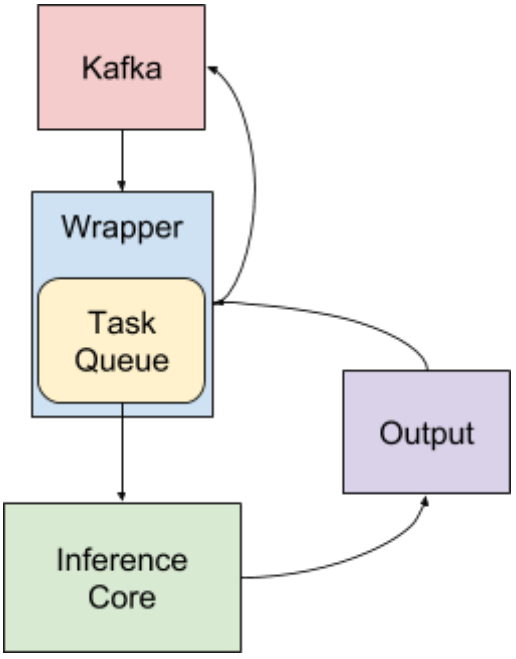
MsgRecvModule在Worker中负责监听MessageQueue的消息，生成TaskInfo写入TaskQueue中，也就是生产者的角色。考虑到Worker对应多个ModelGroup的Topic，需要启用多个线程的监听。

InferenceModule

InferenceModule读取TaskQueue中的TaskInfo信息，获取video_path，由单独的线程做视频的解码，将解码结果写入FrameQueue中，另一个线程获取frame并执行model的Inference，将结果写入数据库。

Worker实现方案

Worker实现实际上是由MsgRecvModule和InferenceModule围绕TaskQueue形成的生产消费模型。由C++实现InferenceCore和Go来实现Wrapper，其中Wrapper实现与外部的通信和任务分配，即TaskQueue由Wrapper维持，InferenceCore完成Inference的功能。两者之间用grpc通信。



因此Wrapper对Kafka消息进行监听，在监听到消息后将任务写入TaskQueue中，当InferenceCore通知Wrapper可以处理新的任务时，Wrapper通过grpc将任务发送给InferenceCore，并以stream形式接受InferenceCore传回的Output。Wrapper将output发送Kafka或者数据库。在InferenceCore发送Output时会发送进度信息给Wrapper，用来传递给Kafka，并被前端使用。因此需要定义InferenceCore和Wrapper之间的grpc接口，Wrapper传递给Kafka消息的接口。

```
message AnnotateVideoRequest {
    string request_id = 1;
    string video_path = 2;
}

message AnnotateVideoResponse {
    int64 total_num = 1;
    int64 progress_num = 2;
    repeated storage.Framemetadata metadatas = 3;
}

message CheckInferenceCoreRequest {}

message CheckInferenceCoreResponse {
    bool is_able = 1;
}
```

```
Service Core {
  rpc AnnotateVideo(AnnotateVideoRequest) returns (stream AnnotateVideoResponse) {}
  rpc CheckInferenceCore(CheckInferenceCoreRequest) returns
(CheckInferenceCoreResponse) {}
}
```