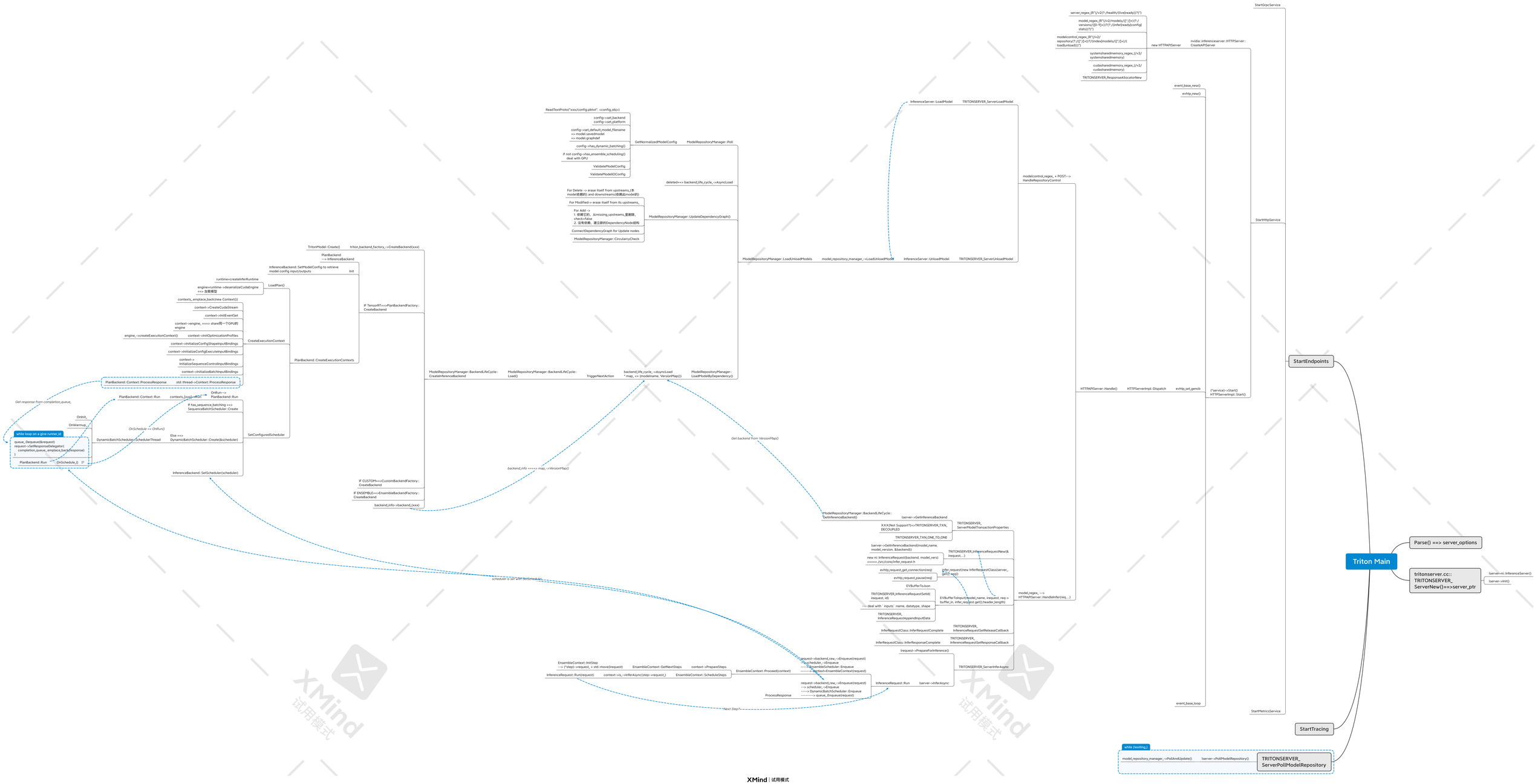
Triton Main
Parse() ==> server_options
tritonserver.cc::
TRITONSERVER_
ServerNew()==>server_ptr
(server-> Init()
StartEndpoints
StartTracing
StartGrpcService
StartHttpService
StartMetricsService
server_regex,R"(/v2(?:/health/(live|ready)(?)?)"
model_regex,R"(/v2/models/([^/]+)(?:/(versions/([0-9]+))?)?)",(/(infer|ready|config|stats)(?)?)
modelcontrol_regex,R"(/v2/repository(?:/(?:([^/]+)?)(?:/index|models/([^/]+)/(load|unload))?)"
systemsharedmemory_regex,(/v2/systemsharedmemory)
cudasharedmemory_regex,(/v2/cudasharedmemory)
TRITONSERVER_ResponseAllocatorNew
new HTTPAPIServer
nvidia::inferenceserver::HTTPServer::
CreateAPIServer
event_base_new()
evhttp_new()
modelcontrol_regex, + POST--> HandleRepositoryControl
HTTPAPIServer::Handle()
HTTPServerImpl::Dispatch
evhttp_set_gencb
(*service)->Start()
HTTPServerImpl::Start()
event_base_loop
InferenceServer::LoadModel
TRITONSERVER_ServerLoadModel
InferenceServer::UnloadModel
TRITONSERVER_ServerUnloadModel
ReadTextProto("xxx/config.pbtxt", <<config_info>>)
config->set_backend
config->set_platform
config->set_default_model_filename
=> model.savedmodel
=> model.graphdef
config->has_dynamic_batching()
if not config->has_ensemble_scheduling()
deal with GPU
ValidateModelConfig
ValidateModelIOConfig
GetNormalizedModelConfig
ModelRepositoryManager::Poll
deleted==> backend_life_cycle_->AsyncLoad
For Delete -> erase itself from upstreams,(本model依赖的) and downstreams(依赖此模型的)
For Modified-> erase itself from its upstreams,
For Add -->
1. 依赖它的，从missing_upstreams_里删除，
check=false
2. 没有依赖，建立新的DependencyNode结构
ConnectDependencyGraph for Update nodes
ModelRepositoryManager::CircularcyCheck
ModelRepositoryManager::UpdateDependencyGraph()
ModelRepositoryManager::LoadUnloadModels
model_repository_manager_->LoadUnloadModel
TritonModel::Create()
triton_backend_factory_->CreateBackend(xxx)
PlanBackend
--> InferenceBackend
InferenceBackend::SetModelConfig to retrieve
model config input/outputs
Init
runtime=createInferRuntime
engine=runtime->deserializeCudaEngine
==> 加载模型
LoadPlan()
contexts_.emplace_back(new Context())
context->CreateCudaStream
context->InitEventSet
context->engine_ ==> share同一个GPU卡
engine
engine_->createExecutionContext()
context->InitOptimizationProfiles
context->InitializeConfigShapeInputBindings
context->InitializeConfigExecuteInputBindings
context->
InitializeSequenceControlInputBindings
context->InitializeBatchInputBindings
CreateExecutionContext
PlanBackend::CreateExecutionContexts
IF TensorRT==>PlanBackendFactory::
CreateBackend
PlanBackend::Context::ProcessResponse
std::thread->Context::ProcessResponse
Get response from completion_queue_
PlanBackend::Context::Run
contexts_[xxx]->Run()
OnRun-->
PlanBackend::Run
OnInit_
OnWarmup_
OnSchedule -> OnRun()
while loop on a give runner_id
queue_.Dequeue(&request)
request->SetResponseDelegator(
completion_queue_.emplace_back(response)
)
PlanBackend::Run
-->OnSchedule_()
DynamicBatchScheduler::SchedulerThread
if has_sequence_batching ==>
SequenceBatchScheduler::Create
Else ==>
DynamicBatchScheduler::Create(&scheduler)
SetConfiguredScheduler
InferenceBackend::SetScheduler(scheduler)
schedule = std::bind SetScheduler
IF CUSTOM==>CustomBackendFactory::
CreateBackend
IF ENSEMBLE==>EnsembleBackendFactory::
CreateBackend
backend_info->backend_(xxx)
backend_info ====> map_->VersionMap()
ModelRepositoryManager::BackendLifeCycle::
CreateInferenceBackend
ModelRepositoryManager::BackendLifeCycle::
Load()
TriggerNextAction
backend_life_cycle_->AsyncLoad
*map_ <= (modelname, VersionMap())
ModelRepositoryManager::
LoadModelByDependency()
Get backend from VersionMap()
ModelRepositoryManager::BackendLifeCycle::
GetInferenceBackend()
lserver->GetInferenceBackend
XXX(Not Support?)==>TRITONSERVER_TXN_
DECOURLED
TRITONSERVER_TXN_ONE_TO_ONE
TRITONSERVER_
ServerModelTransactionProperties
lserver->GetInferenceBackend(model_name,
model_version, &backend)
new ::InferenceRequest(backend, model_vers)
<====./src/core/infer_request.h
TRITONSERVER_InferenceRequestNew(&
inrequest,...)
evhttp_request_get_connection(req)
evhttp_request_pause(req)
Infer_request=new InferRequestClass(server_,
get2_req)
EVBufferToJson
TRITONSERVER_InferenceRequestSetId(
inrequest, id)
EVBufferToInputJson(model_name, inrequest, req->
buffer_in, infer_request.get()).header_length()
--> deal with `inputs`:name, datatype, shape
TRITONSERVER_
InferenceRequestAppendInputData
model_regex, -->
HTTPAPIServer::HandleInfer(req,...)
InferRequestClass::InferRequestComplete
TRITONSERVER_
InferenceRequestSetReleaseCallback
InferRequestClass::InferResponseComplete
TRITONSERVER_
InferenceRequestSetResponseCallback
Inrequest->PrepareForInference()
request->backend_raw_->Enqueue(request)
--> scheduler_->Enqueue
--> EnsembleScheduler::Enqueue
-------> context->Enqueue(request)
TRITONSERVER_ServerInferAsync
EnsembleContext::InitStep
--> (*step)->request_ = std::move(request)
EnsembleContext::GetNextSteps
context->PrepareSteps
EnsembleContext::Proceed(context)
InferenceRequest::Run(request)
context->is_->InferAsync(step->request_)
EnsembleContext::ScheduleSteps
request->backend_raw_->Enqueue(request)
--> scheduler_->Enqueue
-----> DynamicBatchScheduler::Enqueue
--------> queue_.Enqueue(request)
InferenceRequest::Run
lserver->InferAsync
ProcessResponse
Next Step?
while (!exiting_)
model_repository_manager_->PollAndUpdate()
lserver->PollModelRepository()
TRITONSERVER_
ServerPollModelRepository

XMind | 试用模式