Program Listing for File app_driver.hpp

Return to documentation for file (`include/holoscan/core/app_driver.hpp`)

```
/* * SPDX-FileCopyrightText: Copyright (c) 2023 NVIDIA CORPORATION & AFFILIATES.
All rights reserved. * SPDX-License-Identifier: Apache-2.0 * * Licensed under the
Apache License, Version 2.0 (the "License"); * you may not use this file except in
compliance with the License. * You may obtain a copy of the License at * *
http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law
or agreed to in writing, software * distributed under the License is distributed on an
"AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
express or implied. * See the License for the specific language governing
permissions and * limitations under the License. */ #ifndef
HOLOSCAN_CORE_APP_DRIVER_HPP #define HOLOSCAN_CORE_APP_DRIVER_HPP
#include <future> #include <memory> #include <queue> #include <string>
#include <unordered_map> #include <utility> // for std::pair #include <vector>
#include "holoscan/core/common.hpp" #include "holoscan/core/application.hpp"
#include "holoscan/core/fragment_scheduler.hpp" #include
"holoscan/core/graphs/flow_graph.hpp" #include "holoscan/core/io_spec.hpp"
namespace holoscan { // Forward declarations struct AppWorkerTerminationStatus;
enum class AppWorkerTerminationCode; struct ConnectionItem {
ConnectionItem(std::string name, IOSpec::IOType io_type, IOSpec::ConnectorType
connector_type, ArgList args) : name(std::move(name)), io_type(io_type),
connector_type(connector_type), args(std::move(args)) {} std::string name;
IOSpec::IOType io_type; IOSpec::ConnectorType connector_type; ArgList args; }; class
AppDriver { public: explicit AppDriver(Application* app); virtual ~AppDriver(); static
bool get_bool_env_var(const char* name, bool default_value = false); static int64_t
get_int_env_var(const char* name, int64_t default_value = 0); static uint64_t
parse_memory_size(const std::string& size_str); static void
set_ucx_to_exclude_cuda_ipc(); static void exclude_cuda_ipc_transport_on_igpu();
enum class AppStatus { kNotInitialized, kNotStarted, kRunning, kFinished, kError, };
enum class DriverMessageCode { kCheckFragmentSchedule,
kWorkerExecutionFinished, kTerminateServer, }; struct DriverMessage {
DriverMessageCode code; std::any data; }; void run(); std::future<void> run_async();
CLIOptions* options(); AppStatus status(); FragmentScheduler*
fragment_scheduler(); MultipleFragmentsPortMap* all_fragment_port_map(); void
submit_message(DriverMessage&& message); void process_message_queue();
private: friend class service::AppDriverServer; bool
need_to_update_port_names(std::shared_ptr<holoscan::FragmentEdgeDataElementTy
port_map); bool update_port_names(std::string src_frag_name, std::string
target_frag_name, std::shared_ptr<holoscan::FragmentEdgeDataElementType>&
port_map); bool collect_connections(holoscan::FragmentGraph& fragment_graph);
void correct_connection_map(); void connect_fragments(holoscan::FragmentGraph&
fragment_graph, std::vector<holoscan::FragmentNodeType>& target_fragments);
bool check_configuration(); void collect_resource_requirements(const Config&
app_config, holoscan::FragmentGraph& fragment_graph);
SystemResourceRequirement parse_resource_requirement(const YAML::Node&
node); SystemResourceRequirement parse_resource_requirement( const
std::string& fragment_name, const YAML::Node& node, const
SystemResourceRequirement& base_requirement); SystemResourceRequirement
parse_resource_requirement( const std::string& fragment_name, const
SystemResourceRequirement& base_requirement); void
check_fragment_schedule(const std::string& worker_address = ""); void
check_worker_execution(const AppWorkerTerminationStatus& termination_status);
void terminate_all_workers(AppWorkerTerminationCode error_code); void
launch_app_driver(); void launch_app_worker(); std::future<void>
launch_fragments_async(std::vector<FragmentNodeType>& target_fragments);
Application* app_ = nullptr; CLIOptions* options_ = nullptr; bool
need_health_check_ = false; bool need_driver_ = false; bool need_worker_ = false;
bool is_local_ = false; AppStatus app_status_ = AppStatus::kNotInitialized;
std::unordered_map<std::shared_ptr<Fragment>,
std::vector<std::shared_ptr<ConnectionItem>>> connection_map_;
std::unordered_map<std::string, std::vector<std::pair<int32_t, uint32_t>>>
receiver_port_map_; std::unordered_map<int32_t, std::string> index_to_ip_map_;
std::unordered_map<int32_t, uint32_t> index_to_port_map_;
std::unique_ptr<service::AppDriverServer> driver_server_;
std::unique_ptr<FragmentScheduler> fragment_scheduler_; std::mutex
message_mutex_; std::queue<DriverMessage> message_queue_;
std::unique_ptr<MultipleFragmentsPortMap> all_fragment_port_map_ =
std::make_unique<MultipleFragmentsPortMap>(); }; } // namespace holoscan
#endif/* HOLOSCAN_CORE_APP_DRIVER_HPP */
```