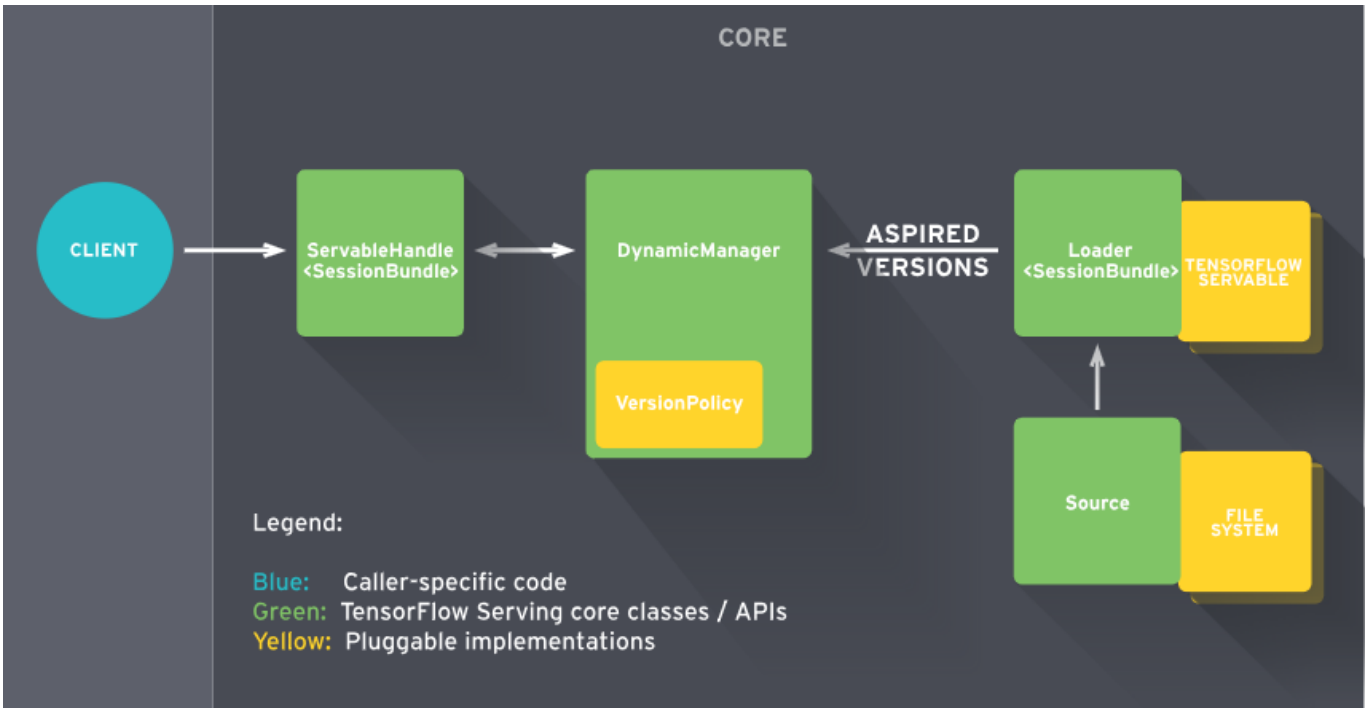
TFS Studying

1. Architecture Overview



1. How TFS found new servable?
2. How TFS load/unload servable?
3. How TFS provide predict service?
4. Discussion about what TFS is capable of.

NOTE: Parts 2 represents main works that TFS components do. If you are not interested and only focus on discussions above, just skip this Parts.

2. Key Concepts

1. **Servables**: Servables are the underlying objects that clients use to perform computation (for example, a lookup or inference). *Typical Servables includes*: `a SavedModelBundle(tensorflow::Session)` and `a lookup table for embedding`
    - **Servable Versions**: TensorFlow Serving can handle one or more versions of a servable over the lifetime of a single server instance.
    - **Models**: TensorFlow Serving represents a model as one or more servables.

      You can represent a **composite model** as either of the following:

      - multiple independent servables
      - single composite servable

      A servable may also correspond to a fraction of a model. For example, a large lookup table could be sharded across many TensorFlow Serving instances.

2. **Loaders**:

   > **Loaders** manage a servable's life cycle. The Loader API enables common infrastructure independent from specific learning algorithms, data or product use-cases involved. Specifically, Loaders standardize the APIs for loading and unloading a servable.

3. **Sources**: Sources are plugin modules that find and provide servables. Each Source provides zero or more servable streams. For each servable stream, a Source supplies one Loader instance for each version it makes available to be loaded. *Sources can maintain state that is shared across multiple servables or versions. This is useful for servables that use delta (diff) updates between versions.*

4. **Aspired Versions**: Aspired versions represent the set of servable versions that should be loaded and ready.

5. **Managers**: Managers handle the full lifecycle of Servables, including

   - loading Servables
   - serving Servables
   - unloading Servables

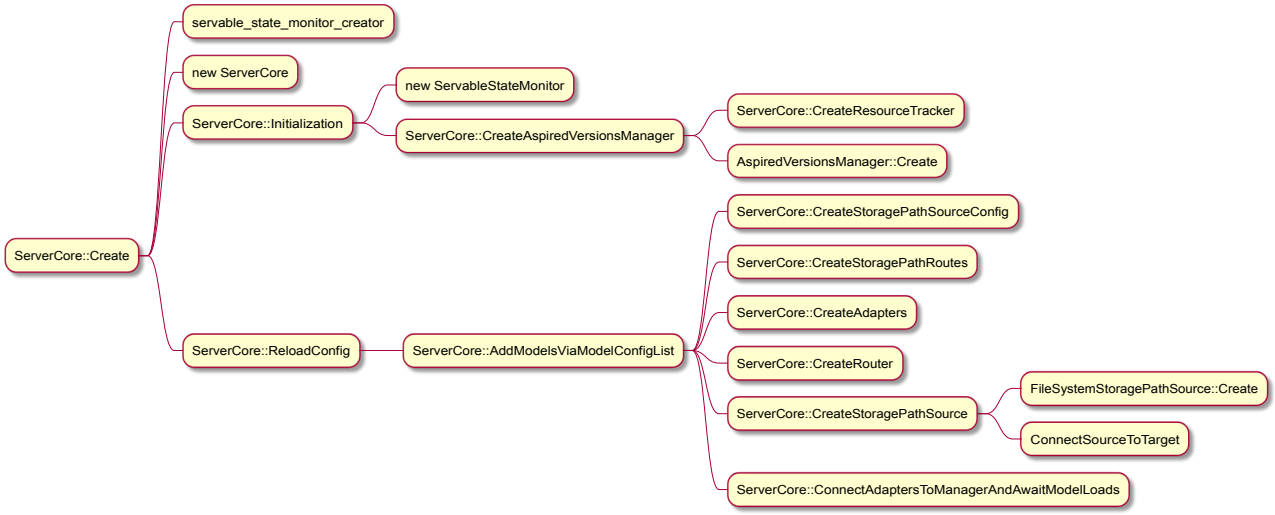
   TensorFlow Serving Managers provide a simple, narrow interface -- `GetServableHandle()` -- for clients to access loaded servable instances.

6. **core**: Using the standard TensorFlow Serving APis, TensorFlow Serving Core manages the following aspects of servables:

   - lifecycle
   - metrics

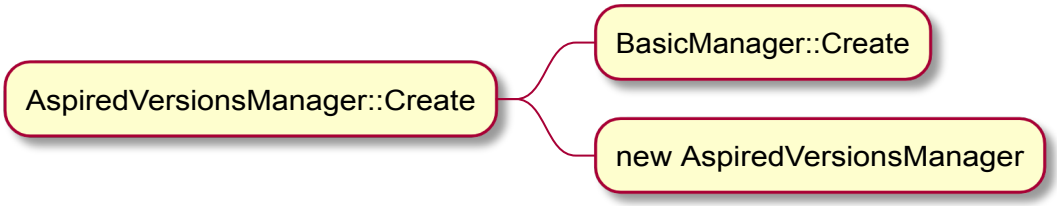
2.1 ServerCore

- Create components of model server

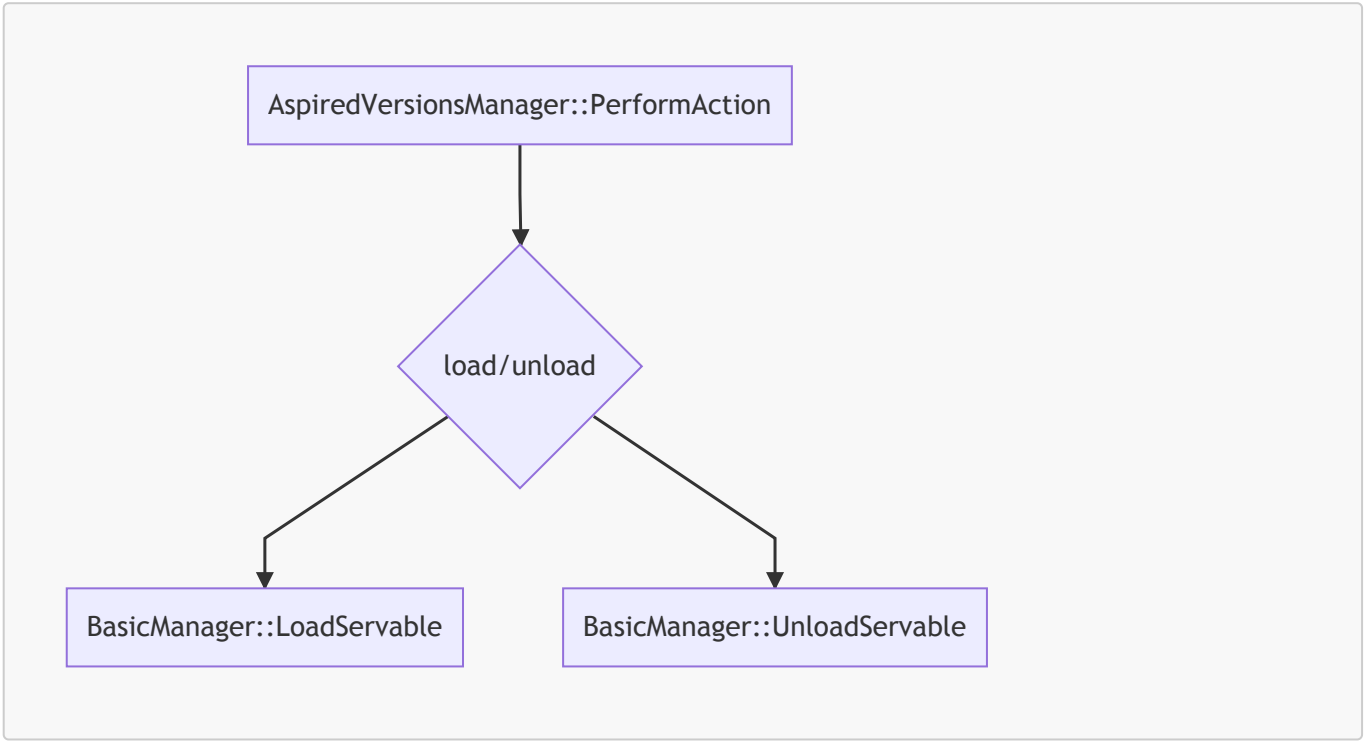


2.2 AspiredVersionManager

- create BasicManager

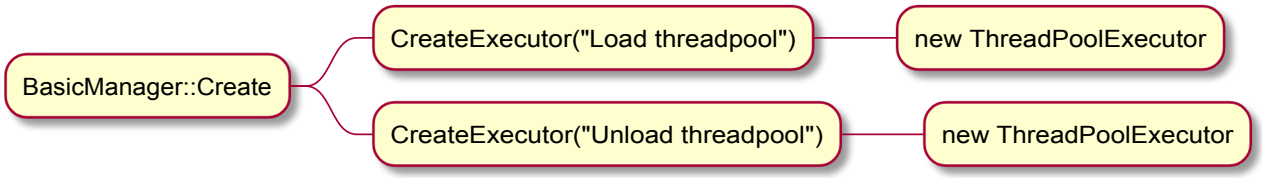


- periodically perform servable action (load or unload servable)



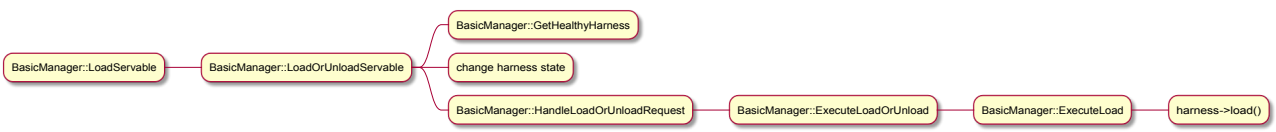
2.3 BasicManager

- create load/uload thread pool



- manage servable

- load or unload servable



2.4 LoaderHarness

```
/// LoaderHarness is a widget the Manager uses to hold on to and talk to a
/// Loader while it owns it. It tracks the overall state of a Servable such that
/// Manager can determine which state transitions to make at what times.
///
/// A manager implementation can also add some additional state with each
/// harness. For example, a manager could put ACL or lifecycle metadata here.
/// The ownership is maintained by the harness.
///
/// This class is thread-safe.
class LoaderHarness final {
 public:
```

- state transition

```
Status LoaderHarness::TransitionState(const State from, const State to) {
  if (state_ != from) {
    const Status error = errors::Internal(
        "Illegal request to transition from state ", StateDebugString(state_),
        " to ", StateDebugString(to));
    DCHECK(false) << error;
    ErrorInternal(error);
    return error;
  }
  state_ = to;
  LOG(INFO) << "LoaderHarness::TransitionState() " << id()
      << " from " << state_2_str(from) << " to " << state_2_str(to);
  return Status::OK();
}
```

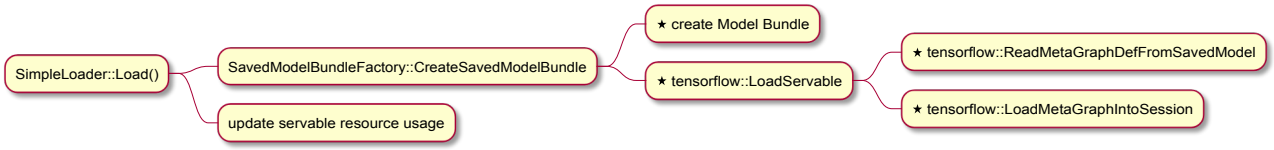
harness操作都伴随着state的transition，合法的状态转换如下：
kNew-->kLoading-->kReady-->kQuiescing-->kQuiesced-->kUnloading-->kDisabled

- load source

- unload source
（略）

2.4 Loader

- load

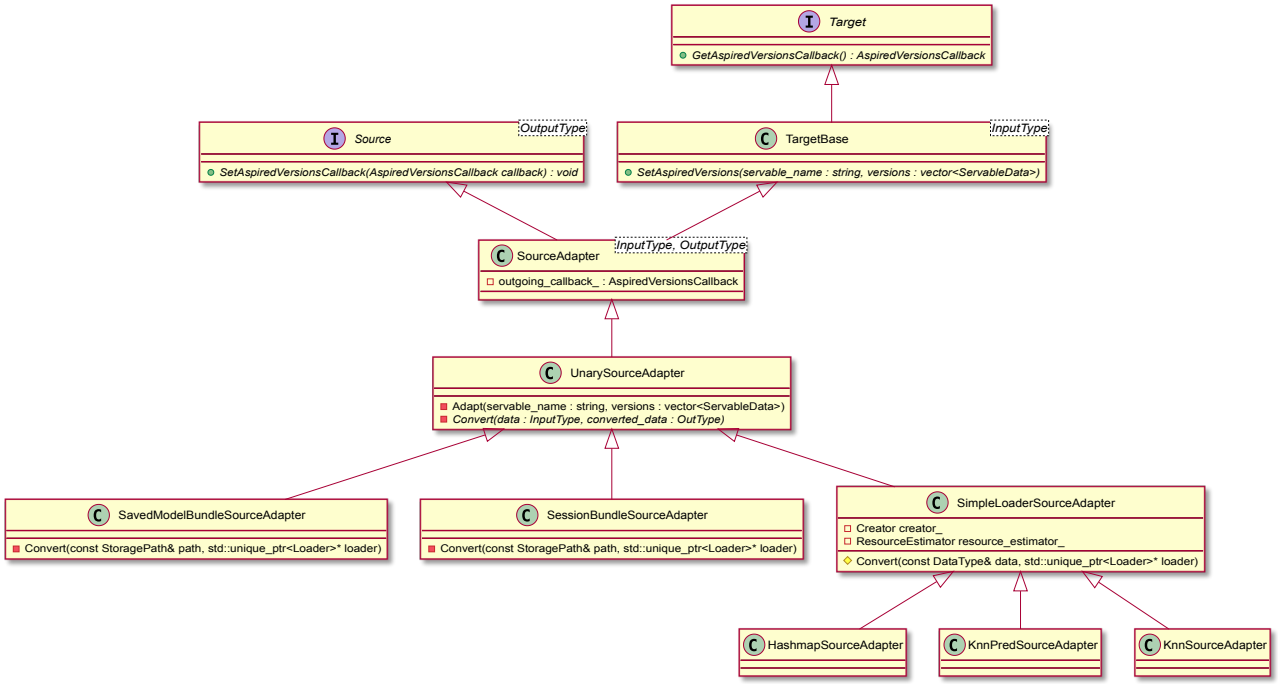


- unload
略

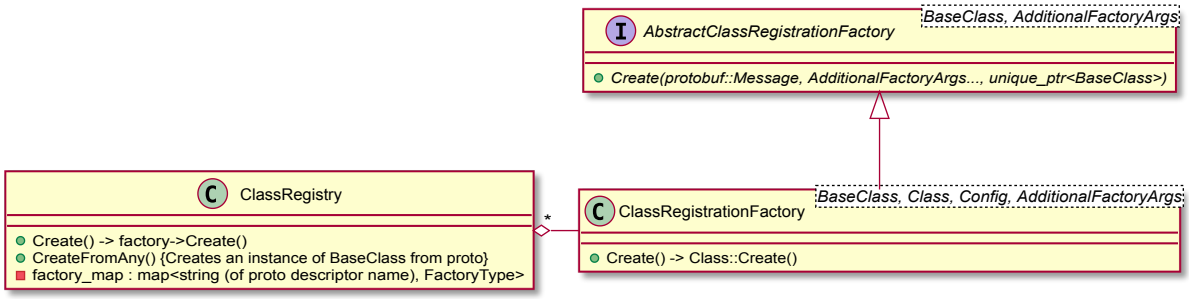
2.5 SourceAdapter

SourceAdapter is an abstraction for a module that receives aspired-version `callbacks` with data of type InputType and `converts` them into calls with data of type OutputType.
It's so complicated that we explain it with charts.

- adpater class diagram



- adapter registration
  - registration class diagram



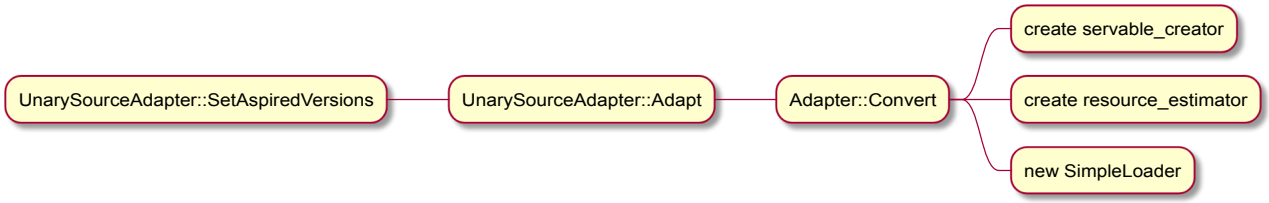
- class registration

```
// Registers a factory that creates subclasses of BaseClass by calling
// ClassCreator::Create().
#define REGISTER_CLASS(RegistryName, BaseClass, ClassCreator, config_proto, \
                       ...)                                                 \
  REGISTER_CLASS_UNIQ_HELPER(__COUNTER__, RegistryName, BaseClass,          \
                             ClassCreator, config_proto, ##__VA_ARGS__)

#define REGISTER_CLASS_UNIQ_HELPER(cnt, RegistryName, BaseClass, ClassCreator, \
                                   config_proto, ...)                          \
  REGISTER_CLASS_UNIQ(cnt, RegistryName, BaseClass, ClassCreator,              \
                      config_proto, ##__VA_ARGS__)

#define REGISTER_CLASS_UNIQ(cnt, RegistryName, BaseClass, ClassCreator,    \
                            config_proto, ...)                             \
  static ::tensorflow::serving::internal::ClassRegistry<                   \
      RegistryName, BaseClass, ##__VA_ARGS__>::MapInserter                 \
      register_class_##cnt(                                                \
          (config_proto::default_instance().GetDescriptor()->full_name()), \
          (new ::tensorflow::serving::internal::ClassRegistrationFactory<  \
              BaseClass, ClassCreator, config_proto, ##__VA_ARGS__>));
```

adapter creater registration的过程就是向ClassRegistry的map结构中增加 string 到Adapter的Factory类的映射。 举个栗子：

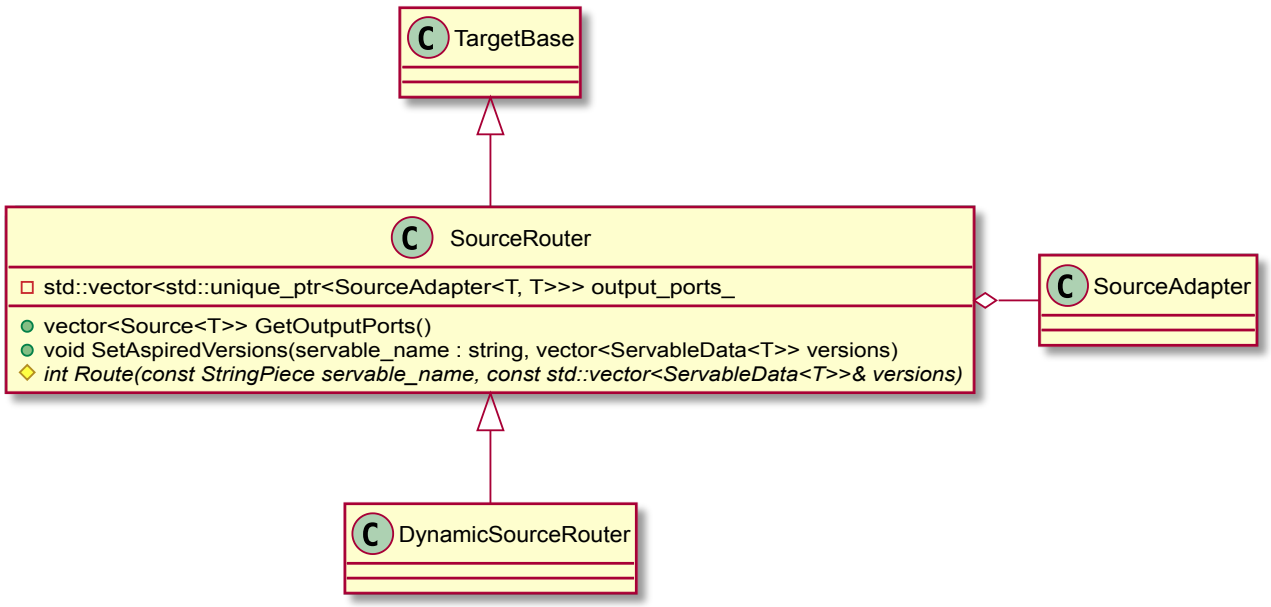
```
platform_configs : {      You, 4 months ago • model server no need to restart when (un)loading model(s).
    key : "tensorflow",
    value : {
        source_adapter_config : {
            type_url: "type.googleapis.com/tensorflow.serving.SavedModelBundleSourceAdapterConfig",
            value: "\302>\002\022\000"
        }
    }
}
```

- Convert (path -> Loader)



2.6 SourceRouter

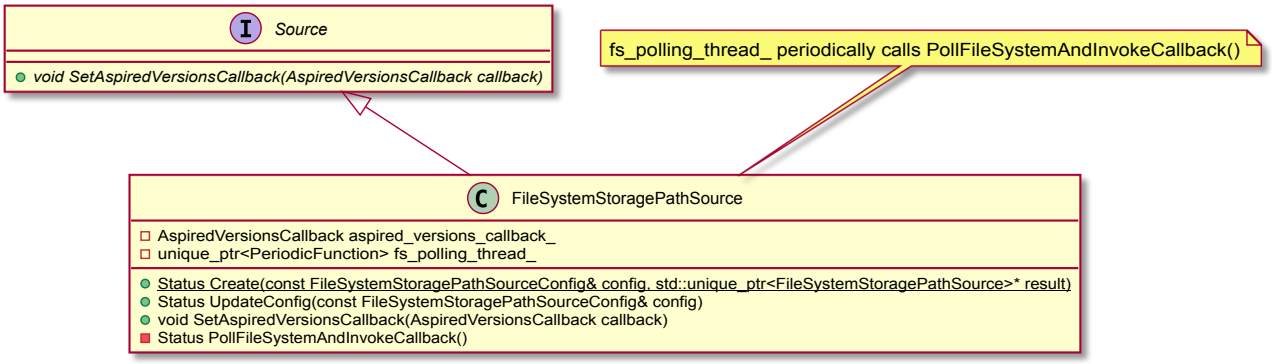
- class diagram



- trigger Adapter

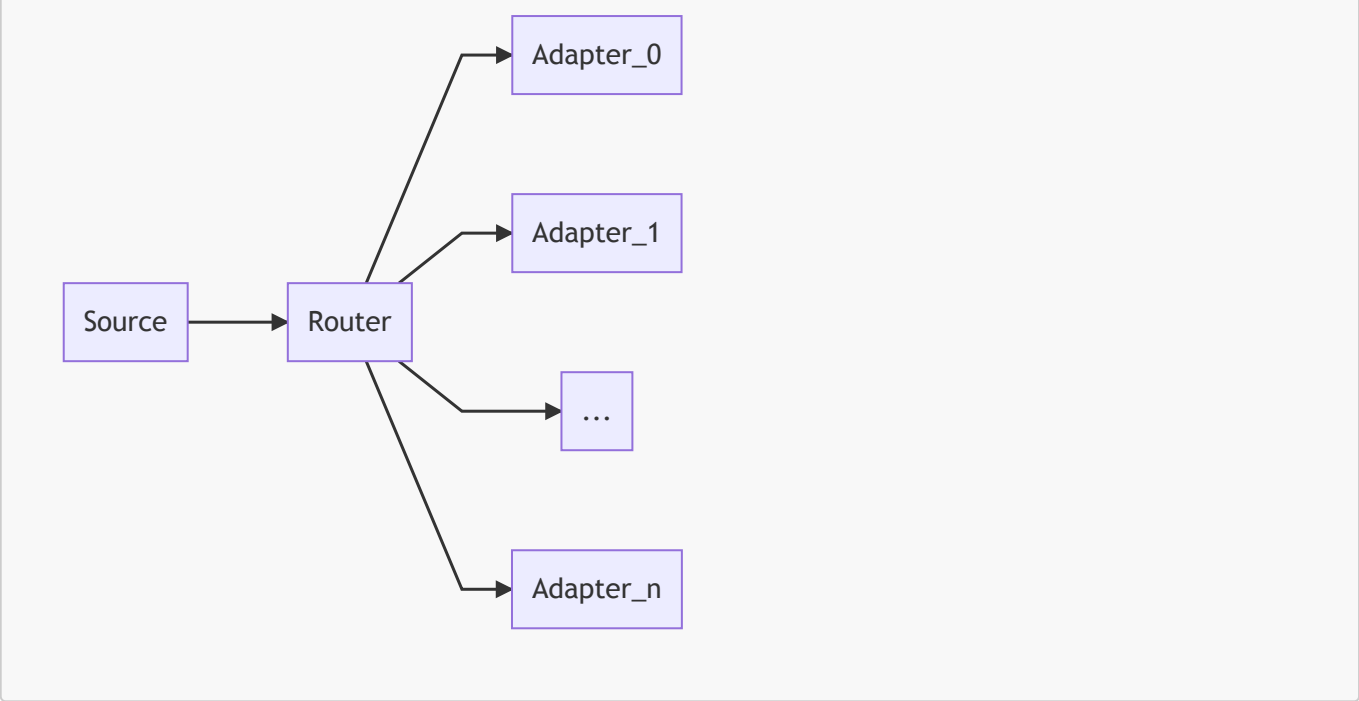
2.7 Source

- class diagram



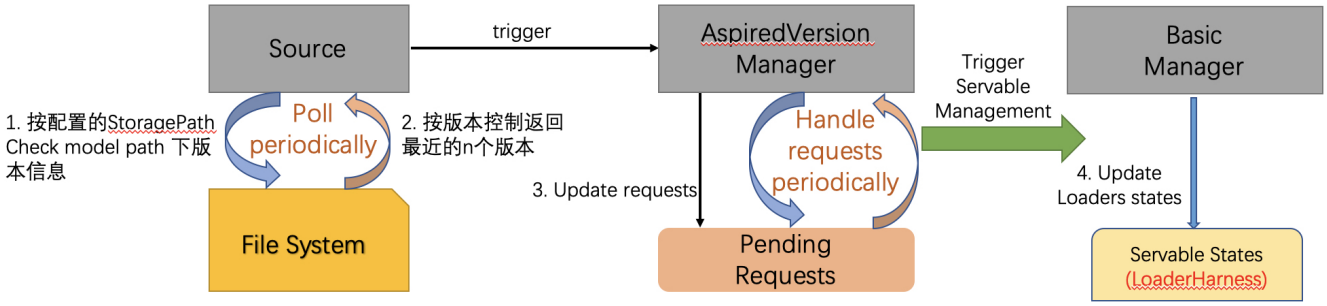
2.8 Source, Router & Adapter

- topology

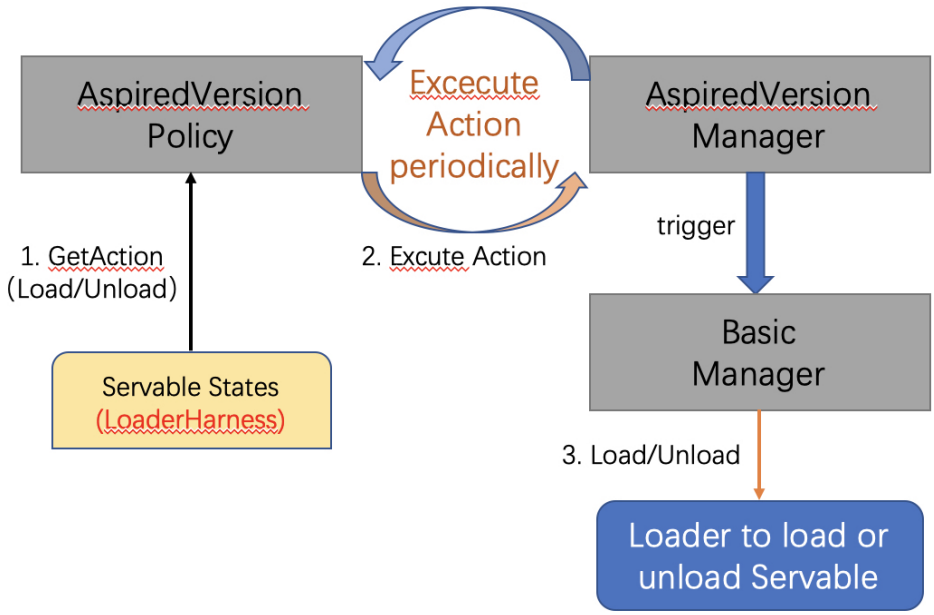


Source的target是Router， Router中包含多个Adapter

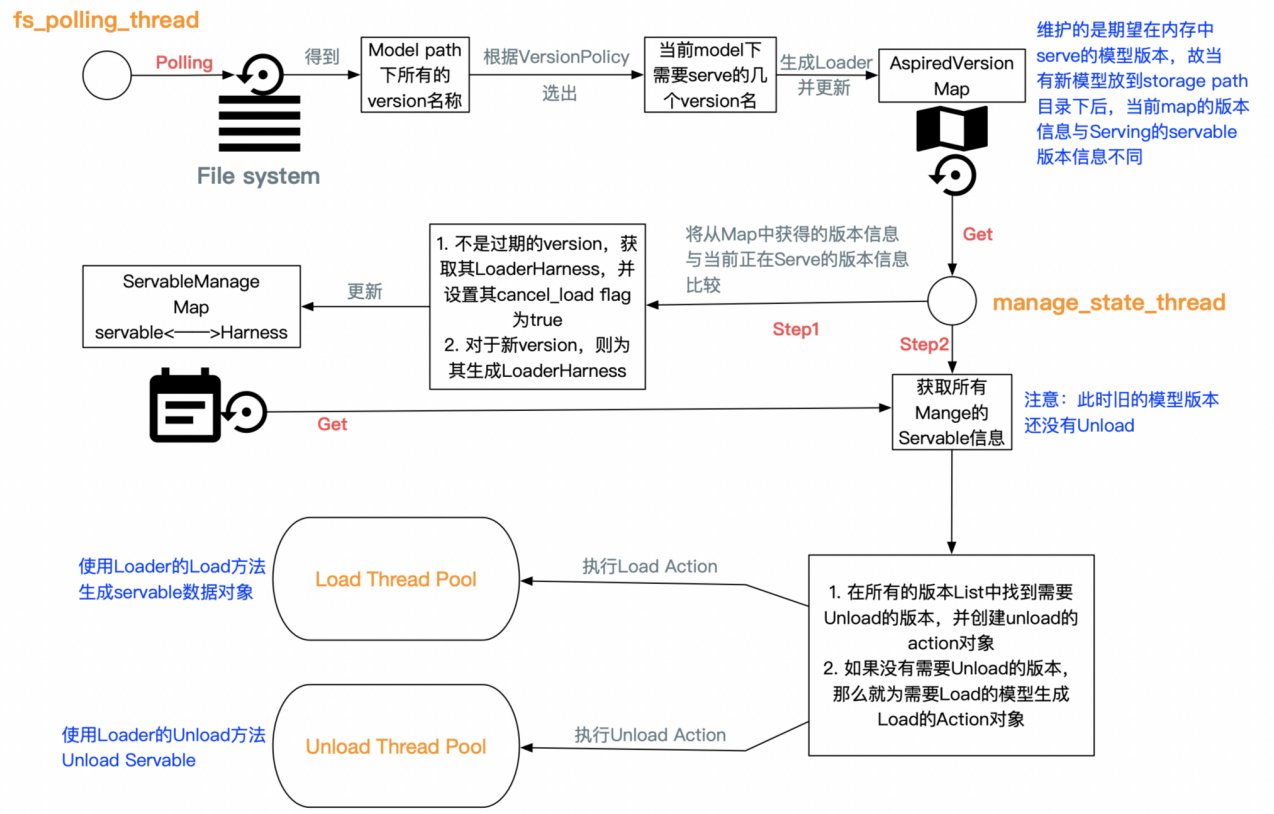
3. How TFS found new servable?



4. How TFS load/unload servable?



- 把上面的两个过程连接起来，得到下面的图



5. How TFS provide predict service?

TFS provide grpc/http service for tensorflow model serving. ServerCore provides an interface `GetServableHandle` for getting servable handle.

6. Discussion about what TFS is capable of

Firstly, `TFS is a server that provides what servables do`. From this perspective, TFS's capability is what we implement servable. Servable can be either tf model calculation or any other computation process. Secondly, `TFS provides multi-servable service`. This is a good way to provide multiple server in same platform, so we can consider it as micro-service provider.
Features:

- online server updating. (version updating)
- computation intensive.

7. Implementation of `AddServable`

This section is a example to implement a simple servable with which we can get a sum of 2 integers.

- Add Service proto

```
/* in "add_service.proto"
 *
* python -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=.
add_service.proto
* protoc -I. --cpp_out=/tmp/1 add_service.proto
```

```
*/
syntax = "proto3";
package tensorflow.serving;
option cc_enable_arenas = true;
/* bazel编译用这个 */
import "tensorflow_serving/apis/model.proto";
/* 单独编译用这个 */
/* import "model.proto"; */

message AddSearchRequest {
    int32 a = 1;
    int32 b = 2;
    ModelSpec model_spec = 3;
}

message AddSearchResult {
    int64 c = 1;
}

service AddService {
rpc AddSearch(AddSearchRequest) returns (AddSearchResult);
}
```

- AddServable

```
/* in "add_servable.h" under $TFS/servables/tests/ */
#pragma once

#include <cstdint>
#include <iostream>

namespace tests {

// A servable that can "add" 2 integers.
class AddServable final {
 public:
    int64_t add(const int& a, const int& b);
};

int64_t AddServable::add(const int& a, const int& b) {
    return a + b;
}

}  // namespace tests
```

- Add Service implement

```
/* in "add_service_impl.h" under $TFS/servables/tests/ */
#pragma once
```

```
#include <iostream>
#include "grpc/grpc.h"
#include "grpcpp/server.h"
#include "grpcpp/server_context.h"
#include "grpcpp/support/status.h"
#include "tensorflow_serving/apis/add_service.grpc.pb.h"
#include "tensorflow_serving/apis/add_service.pb.h"
#include "tensorflow_serving/core/servable_handle.h"
#include "tensorflow_serving/model_servers/grpc_status_util.h"
#include "tensorflow_serving/model_servers/server_core.h"
#include "tensorflow_serving/servables/tests/add_servable.h"

class AddServiceImpl final : public
tensorflow::serving::AddService::Service {
 public:
    using ServerCore = tensorflow::serving::ServerCore;
    using ServerContext = grpc::ServerContext;
    using Request = tensorflow::serving::AddSearchRequest;
    using Response = tensorflow::serving::AddSearchResult;
 public:
    explicit AddServiceImpl(ServerCore *core) : core_(core) {}

    grpc::Status AddSearch(ServerContext* context, const Request* request,
                           Response* response) override;

 private:
    ServerCore* core_;
};

grpc::Status AddServiceImpl::AddSearch(ServerContext* context, const
Request* request,
                                       Response* response) {
    tensorflow::serving::ServableHandle<tests::AddServable> handle;
    auto status = core_->GetServableHandle(request->model_spec(),
&handle);
    if(status == tensorflow::Status::OK()) {
        int64_t c = handle->add(request->a(), request->b());
        std::cout << "c:" << c << std::endl;
        if (response != nullptr) {
            response->set_c(c);
        }
    }
    return grpc::Status();
}
```

- AddSource Adapter and Registration

```
#pragma once

#include <string>
#include "tensorflow_serving/core/simple_loader.h"
```

```
#include "tensorflow_serving/core/source_adapter.h"
#include "tensorflow_serving/core/storage_path.h"
#include "tensorflow_serving/servables/tests/add_source_adapter.pb.h"
#include "tensorflow_serving/servables/tests/add_servable.h"

namespace tensorflow {
namespace serving {

class AddSourceAdapter final
        : public SimpleLoaderSourceAdapter<StoragePath,
tests::AddServable> {
 public:
    explicit AddSourceAdapter();
    ~AddSourceAdapter() override;
private:
    friend class AddSourceAdapterCreator;
private:
    TF_DISALLOW_COPY_AND_ASSIGN(AddSourceAdapter);
};

AddSourceAdapter::AddSourceAdapter()
                : SimpleLoaderSourceAdapter<StoragePath,
tests::AddServable>(
        [](const ServableId& servable_id, const StoragePath& path,
           std::unique_ptr<tests::AddServable>* pServable) -> Status {
               pServable->reset(new tests::AddServable());
               return Status::OK();
        },
        SimpleLoaderSourceAdapter<StoragePath,
tests::AddServable>::EstimateNoResources()) {}


AddSourceAdapter::~AddSourceAdapter() {
    Detach();
}

class AddSourceAdapterCreator {
public:
    static Status Create(const AddSourceAdapterConfig& config, // not used
in our scenario
                std::unique_ptr<SourceAdapter<StoragePath,
std::unique_ptr<Loader>>>* adapter) {
        adapter->reset(new AddSourceAdapter());
        return Status::OK();
    }
};
REGISTER_STORAGE_PATH_SOURCE_ADAPTER(AddSourceAdapterCreator,
                                     AddSourceAdapterConfig);

void loadAddServable() {
    LOG(INFO) << "Registering add servable...";
}
```

```
}   // namespace serving
}   // namespace tensorflow
```

- Add Source Adapter Config proto

```
/* in "add_source_adapter.proto" under $TFS/servables/tests/ */
syntax = "proto3";

package tensorflow.serving;

// Config proto for AddSourceAdapter.
message AddSourceAdapterConfig {
    string conf_data = 1;
}
```

- BUILD Files
    - generate new BUILD File under $TFS/servables/tests/

```
cc_library(
name = "add_servable",
srcs = [
    "add_servable.h",
],
hdrs = [
    "add_servable.h",
] + select({
    ":use_gpu": ["gpu_res.h"],
    "//conditions:default": [],
}),
visibility = [
    "//visibility:public",
],
deps = [
    "//tensorflow_serving/core:servable_id",
    "//tensorflow_serving/model_servers:global_vars",
    "@com_jd_third_party//:boost_lexical_cast",
    "@com_jd_third_party//:boost_string_algo",
    "//tensorflow_serving/apis:add_service_proto",
],)
cc_library(
name = "add_source_adapter",
srcs = [
    "add_source_adapter.h",
],
hdrs = [
    "add_source_adapter.h",
],
visibility = [
```

```
    "//visibility:public",
],
deps = [
    ":add_source_adapter_proto",
    ":add_servable",
    "//tensorflow_serving/core:simple_loader",
    "//tensorflow_serving/core:source_adapter",
    "//tensorflow_serving/core:storage_path",
    "@org_tensorflow//tensorflow/core:lib",
    "@org_tensorflow//tensorflow/core:tensorflow",
],)
cc_library(
name = "add_service_impl",
srcs = [
    "add_service_impl.h",
],
hdrs = [
    "add_service_impl.h",
],
visibility = [
    "//visibility:public",
],
deps = [
    ":add_servable",
    "//tensorflow_serving/apis:add_service_proto",
    "//tensorflow_serving/core:servable_handle",
    "//tensorflow_serving/model_servers:server_core",
    "//tensorflow_serving/model_servers:grpc_status_util",
    "//tensorflow_serving/model_servers:grpc_context_metadata",
],)
load("//tensorflow_serving:serving.bzl", "serving_proto_library")
serving_proto_library(
name = "add_source_adapter_proto",
srcs = ["add_source_adapter.proto"],
cc_api_version = 2,
)
```

- add follows into BUILD file under $TFS/apis

```
serving_proto_library(
name = "add_service_proto",
srcs = ["add_service.proto"],
deps = [
    ":model_proto",
],
has_services = 1,
cc_api_version = 2,
cc_grpc_version = 1,
java_api_version = 2,)
```

- append belows to `TENSORFLOW_DEPS` in BUILD file under $TFS/model_servers

```
"//tensorflow_serving/servables/tests:add_source_adapter",
"//tensorflow_serving/servables/tests:add_service_impl",
```

- Regist Add Service

```
// in "main.cc"
#include "tensorflow_serving/servables/tests/add_source_adapter.h"
#include "tensorflow_serving/servables/tests/add_service_impl.h"

// add belows in Function 'RunServer'
tensorflow::serving::loadAddServable();
AddServiceImpl add_service(core.get());
builder.RegisterService(&add_service);
```

- Server Configs
  - paltform.conf

```
// Add below into platform.conf
// Here we define a platform with key "tests"
// It's adapter config is "AddSourceAdapterConfig"
platform_configs : {
    key: "tests",
    value : {
        source_adapter_config : {
            type_url:
"type.googleapis.com/tensorflow.serving.AddSourceAdapterConfig"
        }
    }
}
```

  - tfserv.conf

```
// Add config content below into tfserv.conf
// we defined a model with name "test_add" with
// platform is new-defined "tests"
// and base_path is "$SOMEWHERE/test_add".
// NOTICE: we should aware of that add_servable in our test does not
// need any MODEL-FILE(Extra Data). We can simply mkdir under path
// "$SOMEWHERE/test_add" with sequence number( > 0) to make servable
work.
// For my instance, I put an empty subdir named "1" under
"$SOMEWHERE/test_add".
config : {
```

```
model_platform : "tests",
name : "test_add",
base_path : "$SOMEWHERE/test_add",
model_version_policy :{
latest :{
    num_versions : 3
}
}
},
```

- testing code & result