10/30 Design Patterns Activity

Team 6

1. Factory

https://github.com/django/django/blob/6452112640081ac8838147a8ba192c45879203d8/django/db/migrations/serializer.py

```
class BaseSerializer:
    def __init__(self, value):
        self.value = value

    def serialize(self):
        raise NotImplementedError('Subclasses of BaseSerializer must implement the serialize() method.')


class BaseSequenceSerializer(BaseSerializer):
    def _format(self):
        raise NotImplementedError('Subclasses of BaseSequenceSerializer must implement the _format() method.')

    def serialize(self):
        imports = set()
        strings = []
        for item in self.value:
            item_string, item_imports = serializer_factory(item).serialize()
            imports.update(item_imports)
            strings.append(item_string)
        value = self._format()
        return value % (", ".join(strings)), imports


class DictionarySerializer(BaseSerializer):
    def serialize(self):
        imports = set()
        strings = []
        for k, v in sorted(self.value.items()):
            k_string, k_imports = serializer_factory(k).serialize()
            v_string, v_imports = serializer_factory(v).serialize()
            imports.update(k_imports)
            imports.update(v_imports)
            strings.append((k_string, v_string))
        return "{%s}" % (", ".join("%s: %s" % (k, v) for k, v in strings)), imports
```

```
def serializer_factory(value):
    if isinstance(value, Promise):
        value = str(value)
    elif isinstance(value, LazyObject):
        # The unwrapped value is returned as the first item of the arguments
        # tuple.
        value = value.__reduce__()[1][0]

    if isinstance(value, models.Field):
        return ModelFieldSerializer(value)
    if isinstance(value, models.manager.BaseManager):
        return ModelManagerSerializer(value)
    if isinstance(value, Operation):
        return OperationSerializer(value)
    if isinstance(value, type):
        return TypeSerializer(value)
    # Anything that knows how to deconstruct itself.
    if hasattr(value, 'deconstruct'):
        return DeconstructableSerializer(value)
    for type_, serializer_cls in Serializer._registry.items():
        if isinstance(value, type_):
            return serializer_cls(value)
    raise ValueError(
        "Cannot serialize: %r\nThere are some values Django cannot serialize into "
        "migration files.\nFor more, see https://docs.djangoproject.com/en/%s/"
        "topics/migrations/#migration-serializing" % (value, get_docs_version())
    )
```

베이스가 되는 BaseSerializer Class 를 정의하고, 이를 상속하여 BaseSequenceSerializer, DictionarySerializer 등의 Class 를 만들고 BaseSerializer 에 정의된 method 를 다르게 정의했습니다. 소스코드 아래의 serializer_factory 함수에서는 이렇게 정의된 다양한 serializer class 들을 조건에 따라 적절한 것으로 return 하고 있습니다.

3. Singleton

https://github.com/pytorch/pytorch/blob/master/torch/csrc/distributed/autograd/context/dist_autograd_container.cpp
- line 21~64

pytorch/csrc/distributed/autograd/context/dist_autograd_context.h 에서 singleton 설명과 pytorch/csrc/distributed/autograd/context/dist_autograd_context.cpp 에서 그 구현을 확인할 수 있습니다. dist_autograd 는 worker 마다 함수의 기울기(differentiation)를 찾게 해주는 class 입니다. worker 마다 하나의 모듈만 있으면 충분하기 때문에 worker 에 대해서 static 하게 하나의 instance 를 가져옵니다.

```
DistAutogradContainer::DistAutogradContainer()
    : next_context_id_(0),
      worker_id_(0),
      initialized_(false),
      next_autograd_message_id_(0),
      max_id_(0) {}

DistAutogradContainer& DistAutogradContainer::init(int64_t worker_id) {
  std::lock_guard<std::mutex> guard(dist_container_init_lock_);

  TORCH_CHECK(
      worker_id >= 0 && worker_id <= kMaxWorkerId,
      "worker_id needs to be in the range [0, 65535]")

  auto& container = getInstanceInternal();
  TORCH_CHECK(
      !container.initialized_,
      "Container is already initialized! Cannot initialize it twice!");

  container.worker_id_ = worker_id;
  container.next_context_id_ = static_cast<int64_t>(worker_id)
      << kAutoIncrementBits;
  container.next_autograd_message_id_ = static_cast<int64_t>(worker_id)
      << kAutoIncrementBits;
  container.max_id_ =
      (kAutoIncrementMask |
       (static_cast<int64_t>(worker_id) << kAutoIncrementBits));
  container.initialized_ = true;
  return container;
}
```

DistAutogradContainer 는 worker_id 에 대한 정보를 받아서 init 함수에서 instance 를 생성합니다. 한편 init 함수 첫줄에서 lock 이 걸려 여러 인스턴스가 만들어지는 것을 방지합니다.

```
DistAutogradContainer& DistAutogradContainer::getInstance() {
  auto& instance = getInstanceInternal();
  TORCH_CHECK(
      instance.initialized_,
      "Need to initialize distributed autograd using "
      "torch.distributed.autograd.init()");
  return instance;
}

DistAutogradContainer& DistAutogradContainer::getInstanceInternal() {
  static DistAutogradContainer container;
  return container;
}
```

instance 는 getInstance 와 getInstanceInternal 에서 받아오는데 getInstanceInternal 은 내부적으로 인스턴스를 받아올 때 쓰고, getInstance 는 외부에서 해당 객체를 사용할 때 쓰게됩니다. getInstanceInternal 에서 객체가 static 으로 정의된 것을 확인할 수 있습니다.

https://github.com/pytorch/pytorch/blob/043530a9b90f26da0fd0469dcc8ab330f252ad65/torch/csrc/distributed/autograd/utils.cpp

```
DistAutogradContext* addRecvRpcBackward(
    const AutogradMetadata& autogradMetadata,
    std::vector<torch::Tensor>& tensors,
    rpc::worker_id_t fromWorkerId) {
  // Initialize autograd context if necessary.
  auto& autogradContainer = DistAutogradContainer::getInstance();
  DistAutogradContext& autogradContext =
      autogradContainer.getOrCreateContext(autogradMetadata.autogradContextId);

  if (!tensors.empty()) {
    TORCH_INTERNAL_ASSERT(
        torch::autograd::compute_requires_grad(tensors),
        "Received tensors do not require grad, addRecvRpcBackward should not be called");

    // Attach the tensors as inputs to the autograd function.
    auto grad_fn = std::make_shared<RecvRpcBackward>(
        autogradMetadata, autogradContext, fromWorkerId);
    for (auto& tensor : tensors) {
      torch::autograd::set_history(tensor, grad_fn);
    }

    // Now update the autograd context with the necessary information.
    autogradContext.addRecvFunction(
        grad_fn, autogradMetadata.autogradMessageId);
  }

  return &autogradContext;
}
```

위 코드는 다른 클래스에서 DistAutogradContainer 를 쓴 예시입니다. getInstance 함수를 통해 곧바로 객체를 가져온 것을 확인할 수 있습니다.

4. **Dependency Injection**

https://github.com/apache/reef/blob/f98f20bb67290e74e8f6ccd57aef539f673237d3/lang/java/reef-io/src/main/java/org/apache/reef/io/network/group/impl/driver/CommunicationGroupDriverFactory.java
- line 47~89

```
@Inject
private CommunicationGroupDriverFactory(
    @Parameter(DriverIdentifier.class) final String driverId,
    @Parameter(GroupCommSenderStage.class) final EStage<GroupCommunicationMessage> senderStage,
    @Parameter(GroupCommRunningTaskHandler.class)
        final BroadcastingEventHandler<RunningTask> groupCommRunningTaskHandler,
    @Parameter(GroupCommFailedTaskHandler.class)
        final BroadcastingEventHandler<FailedTask> groupCommFailedTaskHandler,
    @Parameter(GroupCommFailedEvalHandler.class)
        final BroadcastingEventHandler<FailedEvaluator> groupCommFailedEvaluatorHandler,
    final GroupCommMessageHandler groupCommMessageHandler) {
  injector = Tang.Factory.getTang().newInjector();
  injector.bindVolatileParameter(GroupCommSenderStage.class, senderStage);
  injector.bindVolatileParameter(DriverIdentifier.class, driverId);
  injector.bindVolatileParameter(GroupCommRunningTaskHandler.class, groupCommRunningTaskHandler);
  injector.bindVolatileParameter(GroupCommFailedTaskHandler.class, groupCommFailedTaskHandler);
  injector.bindVolatileParameter(GroupCommFailedEvalHandler.class, groupCommFailedEvaluatorHandler);
  injector.bindVolatileInstance(GroupCommMessageHandler.class, groupCommMessageHandler);
}

/**
 * Instantiates a new CommunicationGroupDriver instance.
 * @param groupName specified name of the communication group
 * @param topologyClass topology implementation
 * @param numberOfTasks minimum number of tasks needed in this group before start
 * @param customFanOut fanOut for TreeTopology
 * @return CommunicationGroupDriver instance
 * @throws InjectionException
 */
public CommunicationGroupDriver getNewInstance(
    final Class<? extends Name<String>> groupName,
    final Class<? extends Topology> topologyClass,
    final int numberOfTasks,
    final int customFanOut) throws InjectionException {

  final Injector newInjector = injector.forkInjector();
  newInjector.bindVolatileParameter(CommGroupNameClass.class, groupName);
  newInjector.bindVolatileParameter(TopologyClass.class, topologyClass);
  newInjector.bindVolatileParameter(CommGroupNumTask.class, numberOfTasks);
  newInjector.bindVolatileParameter(TreeTopologyFanOut.class, customFanOut);
  return newInjector.getInstance(CommunicationGroupDriver.class);
}
```

https://github.com/apache/reef/blob/f98f20bb67290e74e8f6ccd57aef539f673237d3/lang/java/reef-io/src/main/java/org/apache/reef/io/network/group/impl/driver/GroupCommDriverImpl.java
- line 178~183

```
try {
  commGroupDriverFactory = injector.getInstance(CommunicationGroupDriverFactory.class);
} catch (final InjectionException e) {
  throw new RuntimeException(e);
}
```

앞의 코드 부분에서 @Inject 키워드를 이용해서 injector 에서 사용할 생성자를 설정하였습니다. Injector 내부에서는 bindVolatileParameter 을 이용해서 parameter 들을 바인드 해줍니다.
뒷부분의 코드에서 실제로 injector 가 사용되는 예시를 볼 수 있습니다.