Eclipse Modeling Framework (EMF)

Altran, Bangalore, India
19’th - 22’nd March 2020

EMF Runtime framework

- Notification and Adapter
- Persistence framework
- Proxy Resolution and lazy loading
- EPackage, Resource Factory, and Registries.

Notification and Adapter

Notification

- The **EMF Notification framework** makes it possible for every EObject instance to send notifications whenever the value of its structural feature changes
- The interface **`EObject`** extends from the **`Notifier`** interface
- The method **`eNotify(Notification)`** delivers the notification to the notification observers (AKA **Adapter**). The parameter **`Notification`** describes the change.

Adapter

- An EMF **Adapter** (AKA notification Observer) listens to the changes made to the features of the Notifier instance
- The delivery of the notifications to the attached Adapter happens via the `Adapter#notifyChanged()` method
- An Adapter instance can be attached to the target EObject using the `eObject.eAdapters().add(adapter)` method.

Do's and Don'ts

- Adapters should be attached and disposed of correctly as the object hierarchy changes. If not done correctly, it might lead to **memory leaks**
- The `EObject#eAdapters()` list can contain duplicate adapters. Adding the same adapter instances multiple times to the list should be avoided
- The implementation of the `Adapter#notifyChanged()` must be lightweight. It should not impact the user experience in any way.

Exercise

- Inspect the state change methods and the notification logic in the generated code
- Enhance one of the generated **`AdapterFactory#createXXAdapter()`** methods to return a specific Adapter instance
- Create a Junit test for the demonstrate the following
  - Use the above Enhanced Adapter factory to get a specific Adapter instance
  - Attach the returned Adapter instance to a model instance
  - Perform modifications to the features of the model instance.

EMF Persistence framework

Resourceset, Resource, and URI

- EMF **`Resource`** is a container of one or more objects that are persisted together, along with their contents
- EMF **`ResourceSet`** is a container of one or more Resources. It manages the references between the objects in different resources
- EMF uses **Uniform Resource Identifier (`URI`)** for the following purposes
  - Uniquely identify an EPackage
  - Uniquely identifying a Resource
  - Reference objects within a Resource
- **`Resource#save(Map)`** persists the contained objects. By default, XML Meta Interchange (XMI) 2.0 format used for persistence. EMF supports XML serialization out of the box

Do's and Don'ts

- Running the application in the **Standalone** mode requires explicit registration of the Resource Factory and EPackage instance in the appropriate registries
- Running the application **in Eclipse** does not require the above registration. Explicit registration might lead to incorrect application behavior.

Exercise

- Create Junit test to demonstrate the following
  - Using the generated Factory for creating model instances
  - API for Saving a Resource to the local file system
  - API for Loading a Resource in the memory
  - Persisting objects having cross-references into multiple resources.

EMF Proxy resolution

EMF Proxy and Proxy resolution

- A Proxy is a placeholder object that contains the **URI** to the target object
- When loading a resource, all **cross-document references** use proxies instead of the actual target objects. The proxies are replaced by the target objects on their first use (AKA Proxy resolution)
- The proxy is retained if the resolution fails for some reason.

Non-containment references and Proxy resolution

- In the case of non-containment references, by default, the source and the target objects are contained in two separate resources
- Non-containment references have proxy resolution **enabled by default**. At first, all cross-document references use proxies instead of the target object
- Setting the **Resolve Proxies** attribute of a EReference to **false** enables storing the two objects in the same resource, thereby disabling proxy resolution.

Containment references and Proxy resolution

- In the case of Containment references, the target object is contained in a container object. The container and the contained objects reside in the same resource by default
- Containment references have proxy resolution **disabled by default**
- Setting the **Containment Proxies** attribute to **true** enables storing the contained object in a different resource than its container.

Do's and Don'ts

- API `ECoreUtil#resolveAll()` resolves all the cross-references and loads the complete object hierarchy in the memory. It must be used with caution especially if the Object hierarchy is deep
- For non-containment references, the value of Resolve Proxies attribute should be set to false if a single resource is used for persisting the source and the target objects
- Setting the value of the Containment Proxies to true will make all the contained references proxy resolving. The value of the Resolve Proxies attribute should be set to false for those containment references for which proxy resolution should be disabled.

Exercise

- Create JUnit test to demonstrate
  - EMF Proxies and the Proxy resolution mechanism
  - `ECoreUtil#resolveAll()` and other utility API.

EPackage and EPackage Registry

EPackage

- An EMF EPackage contains the definition of a model. It consists of **`EClasses, EAttributes, EDataTypes, EAnnotations, EOperations`**
- It provides the API for clients to access the above metadata, for example, **`CreditCardPackage#eINSTANCE#getProduct()`** returns the **`Product EClass`**
- It programmatically builds the EPackage, its contents, and registers the same in the global EPackage registry
- While loading a resource, instance creation of the serialized objects happens through the generated EPackage. Method **`EPackage#getXXFactory()`** returns the required factory.

EPackage Registry

- A central **registry for storing** EPackage instances
- The registry is of following types
  - **Global** - `EPackage.Registry.INSTANCE`
  - **Local** - `ResourceSet#getPackageRegistry()`
- The searching of a requested EPackage happens in the local registry and, if not found, the global registry. Throws the `PackageNotFound` exception if the EPackage cannot be found in either of the above registries.

Do's and Don'ts

- Running the application in **Standalone** mode requires explicit registration of the EPackage before loading a Resource in memory
- Running the application in **Eclipse** does not require explicit EPackage registration. It happens through the extension point defined in your plugin.xml file.

Exercise

- Inspect the following methods in the generated model EPackage class
  - `EPackage#init()`
  - `EPackage#createPackageContents()`
- Create a JUnit test to demonstrate the following
  - EPackage registration in the Global EPackage registry
  - EPackage registration in the Local EPackage registry.

Resource Factory and Resource Factory Registry

Resource Factory

- A factory for creating Resource instances
- The method **`ResourceSet#createResource(URI)`** internally uses the factory for resource creation
- The serialization format depends entirely on the returned Resource. **`XMIResource`** has XMI as its serialization format.

Resource Factory Registry

- A central registry for storing Resource Factory instances
- The registry is of following types
  - **Global** - `Resource.Factory.Registry.INSTANCE`
  - **Local** - `ResourceSet#getResourceFactoryRegistry()`
- The searching of a requested Resource Factory happens in the local registry and, if not found, the global registry. Returns a null value if the factory cannot be found.

Do's and Don'ts

- Running the application in **Standalone** mode requires explicit registration of the Resource Factory before creating a Resource using the Resource Set
- Running the application in **Eclipse** does not require explicit EPackage registration. It happens through the extension point defined in your plugin.xml file.

Exercise

- Inspect the following methods in the EMF API
  - `ResourceSetImpl#createResource()`
  - `ResourceSetImpl#getResourceFactoryRegistry()`
- Create a JUnit test for the following
  - Demo Resource Factory registration in the Global EPackage registry
  - Demo Resource Factory registration in the Local EPackage registry
  - Demo the result of no Resource Factory registration.