BizTemplates

Предназначение

BizTemplates - Java библиотека призванная помочь в построении бизнес логики приложения, задумывалась как набор шаблонов на базе интерфейсов. Библиотека не использует никаких контейнеров и фреймворков (это принципиальный подход), доступ к ним происходит через модуль интеграции. С появлением Java 8 возможности интерфейсов были расширены и по сути они превратились в абстрактные классы без данных, появилась возможность использовать `default` методы, это позволило сократить объем кода в классах. Использование интерфейсов это классический подход, позволяющий подключать в классах любой набор требуемой функциональности, в стиле множественного наследования C++ и дающий требуемый уровень абстракции. Построение хорошей domain model ('Шаблоны Корпоративных Приложений' Мартин Фаулер) имеет определяющее значение для продолжительной жизни и легкой поддержки любого приложения.

Краткое описание

Примеры взяты из исходных кодов и демо проекта, некоторые взяты из реализации библиотек расположенных в папке lib (на них не распостраняется лицензия Apache 2.0). Вся реализация не приводится для сокращения объема текста, для просмотра реализации в библиотеках можно воспользоваться декомпелятором, конечно не нарушая авторских прав.

Содержание

1. Шаблоны проектирования.
2. Собственные шаблоны.
3. Шаблонные и вспомогательные классы.
4. Утилитные классы.
5. Интеграционный модуль.

1. Шаблоны проектирования

В библиотеке сделана попытка оформления в интерфейсах общеизвестных шаблонов проектирования. Наименования интерфейсов определяют описанный шаблон. Не все шаблоны реализованы в библиотеке, если будет потребность в определении отсутствующих то возможно их включение в последующие реализации библиотеки.

- `Builder`
- `Composite`

- `Factory`
- `Mediator`
- `ObjectAdapter`
- `Observer`
- `Strategy`
- `Observer`
- `UnitOfWork`
- `Visitor`

Описание шаблонов можно найти в специализированной литературе, в частности в книге 'Применение Шаблонов JAVA' Стивен Стелтинг, Олав Маассен.

2. Собственные шаблоны

Шаблоны определенные в библиотеке, призванные дополнить стандартные шаблоны.

`Indexed`

Определяет уникальный типизированный идентификатор.

```
public interface Component extends Indexed<String> {
  ...
}
```

`Component`

Предназначен для индексирования класса строковым уникальным идентификатором, используется для определения уникального объекта на уровне приложения в runtime режиме. В нем определена функциональность по умолчанию для задания значения идентификатора. Значение идентификатора можно задать аннотацией `#ComponentId`, переопределением константы `COMPONENT_ID` (по умолчанию она равна `null`) или переопределить метод `default String Componen.getId()` вернув требуемое уникальное значение.

```
@ComponentId(TestComponent.COMPONENT_ID)
public interface TestComponent extends Component {

  String COMPONENT_ID = "testComponent";

  String test();

}
```

```
@Service(SpringComponentsResourceAdapter.BEAN_NAME)
@ComponentId(SpringComponentsResourceAdapter.COMPONENT_ID)
public final class SpringComponentsResourceAdapter extends

                                AbstractObjectResourceAdapter<String,Object>
```

```
                implements ObjectResource<String, Object>, ObjectResourceResolver,
ApplicationContextAware {

    public static final String BEAN_NAME = "springComponentsResourceAdapter";
    public static final String COMPONENT_ID = BEAN_NAME;
    public static final ObjectResourceQualifier
                    RESOURCE_QUALIFIER =
                    new ObjectResourceQualifier(COMPONENT_ID, object.class, String.class);
    ...
}

public interface ObjectResourceResolver extends
    Resolver<ObjectResource<?, ?>, ObjectResourceQualifier>, Component {
  ...
}
```

`DataObjectManager`

Предназначен для построения DAO класса к определенному entity объекту.

```
public abstract class EntityDataManager<T> implements DataObjectManager<T> {
    ...
    public abstract EntityManager getEntityManager();
    ...
}
```

`Resolver`

Предназначен для предоставления доступа к объектам определённого типа по определённому аргументу.

```
public interface FactoryResolver extends Resolver<Factory<?>, Class<?>> {
  ...
}
```

`Mapper`

Предназначен для определения преобразователя одного объекта в другой.

```
public class CustomerMapper implements Mapper<net.biztemplates.demo.common.Customer,
 net.biztemplates.demo.persistence.entities.Customer> {

  @Override
  public Customer map(net.biztemplates.demo.persistence.entities.Customer value) {
    return new net.biztemplates.demo.common.Customer(
        value.getInn(),
        value.getFirstName(),
        value.getLastName(),
        value.getMiddleName());
  }
  ...
}
```

`ObjectStateManager`

Предназначен для определения интерфейса сервиса отвечающего за сохранение и чтения состояния бизнес объекта в(из) хранилища данных.

```
public interface CustomerStateManager extends
 ObjectStateManager<net.biztemplates.demo.common.Customer> {

  @Override
  default Class<? extends Customer> getObjectType() {
    return Customer.class;
  }

  Customer[] loadAllCustomers();

  void updateCustomer(Customer customer);

  void loadCustomer(Customer customer);
}

//Бизнес класс адаптер, управляющий состоянием бизнес объекта.
public class CustomerRepository extends ObjectStateManagerAdapter<CustomerStateManager,
net.biztemplates.demo.common.Customer> {
  ...
  public void save(Customer customer) {
    ...
  }

  public void load(Customer customer) {
    ...
  }

  public void delete(Customer customer) {
    ...
  }
  ...
}
```

```
//Класс JPA DAO, реализующий управление состоянием entity и бизнес объектов.
public abstract class CustomerManager extends
EntityDataManager<net.biztemplates.demo.persistence.entities.Customer>
    implements
    CustomerStateManager {
    ...
}

//Spring сервис класс, обеспечивающий JPA контекст и транзакции.
@Repository
public class CustomerRepositiry extends CustomerManager implements CustomerStateManager {
   ...
}
```

`Integrator`

Предназначен для переноса и извлечения части данных в (из) объект(а).

`ParamsHolder`

Предназначен для определения объекта который содержит параметры какого нибудь процесса.

```
public interface Query<T, R> extends ParamsHolder, Component {
  ...
}
```

`Query`

Предназначен для определения объекта запроса к определенному источнику данных. Источником данных может быть любой объект.

```
public abstract class EntityQuery<T> implements Query<T, EntityManager> {
  ...
}
```

`Queryable`

Предназначен для определения объекта предоставляющего источник данных для набора `Query` запросов.

`ValueEditor`

Предназначен для определения объекта устанавливающего определённое значение и дающий требуемый уровень абстракции.

3. Шаблонные и вспомогательные классы

Классы реализующие шаблонные интерфейсы, классы предназначенные для использования в других шаблонах или вспомогательные классы используемые в приложении.

`@ComponentId`

Предназначен для определения ключа для определённого объекта.

`AbstractVisitor`

Базовый класс для реализации `Visitor`.

`CompositeBuilderImpl`

Реализует шаблон `CompositeBuilder`.

`Configuration`

Предназначен для чтения конфигурационных данных из *.properties файлов.

```
public class ObjectStateManagerAdapter<M extends ObjectStateManager<T>, T>
    extends AbstractObjectAdapter<ObjectStateManagerResolverComponent, String> implements
    ObjectStateManager<T> {

  public static final String PROP_OBJECT_STATE_MANAGER_RESOURSE =
"biztemplates.integration.object_states_resource";
    ...
    public ObjectStateManagerAdapter(Class<M> managerType) {
    super(new ObjectResourceQualifier(
        Configuration.property(PROP_OBJECT_STATE_MANAGER_RESOURSE),
        Object.class,
        String.class), ObjectStateManagerResolverComponent.COMPONENT_ID);
    ...
  }
  ...
}
```

`EventsRecorder`

Предназначен для сохранения последовательности событий для их последующей обработки. Единицей информации является объект `Event`.

`FactoryResolverImpl`

Реализует `FactoryResolver`, использует `ConcurrentHashMap<Class<?>, Factory<?>>` для хранения объектов.

`QueryQualifier`

Идентификатор описывающий запрос `Query`.

`QueryResolverImpl`

Реализация `QueryResolver`, использует `ConcurrentHashMap<QueryQualifier, Query<?, ?>>` для хранения объектов.

`ReflectionPropertyEditor`

Реализация `PropertyEditor` использующая Reflection.

4. Утилитные классы.

Классы из пакета `utils`, реализующие функции не относящиеся к прямому назначению библиотеки. Наименование классов описывают их функции, примеры использования можно найти в исходных кодах.

- `PropertiesUtil`
- `ReflectionUtil`
- `ReverseIterator`
- `SerializationUtil`
- `StringUtil`
- `UrlUtil`

5. Интеграционный модуль

Библиотека предназначена для реализации бизнес уровня приложения который должен быть переносим с платформы на платформу, поэтому идеологически для него платформа это внешняя система, тем более одновременно могут использоваться несколько фреймворков. Для реализации прозрачного подключения к внешнему источнику и предназначен интеграционный модуль, являющийся при таком подходе неотъемлемой частью BL (бизнес уровня).

Фундаментом является класс `ObjectResources`, являющийся инструментом для поиска нужного внешнего ресурса. Для регистрации внешнего контекста как ресурса используется `AbstractObjectResourceAdapter`. Для прозрачного встраивания в бизнес модель как её объект используется `AbstractObjectAdapter`. Детальный пример реализован в демо проекте, ниже лишь поверхностное описание.

Данный модуль также может быть использован и для прозрачного подключения к бизнес логике каких то своих разработок, любой код по логике являющийся внешним к BL может быть таким образом подключен.

`ObjectResources` Предназначен для регистрации и поиска внешних ресурсов - объектов реализующих определённые сервисные интерфейсы.

```
String testResourceName = "myResource";
String testResolverName = "myResolver";

ComponentsResource components = new ComponentsResource(testResourceName);
TestClass obj = new TestClass("test");
components.add(obj);
InMemoryObjectResourceResolver resolver = new InMemoryObjectResourceResolver(testResolverName);
resolver.add(components);
ObjectResources.registerResolver(resolver);

ObjectResource<String, Component> resource = ObjectResources.firstResource(
  new ObjectResourceQualifier(testResourceName, Component.class, String.class));
TestComponent comp = null;
comp = ObjectResources.object(TestComponent.class, TestComponent.COMPONENT_ID);
comp = ObjectResources.component(TestComponent.class);
```

`ObjectResource`

Определяет функции поиска объектов в определенном источнике, например контексте Spring.

```
public class ComponentsResource implements ObjectResource<String, Component>,
    Composite<String, Component> {
  ...
}
```

`ObjectResourceQualifier`

Предназначен для идентификации регистрируемого ресурса.

`ObjectResourceResolver`

Предназначен для поиска ресурса `ObjectResource` по его идентификатору `ObjectResourceQualifier` и регистрации в `ObjectResources`.

```
public class InMemoryObjectResourceResolver implements ObjectResourceResolver,
    Set<ObjectResource<?, ?>> {
  ...
}
```

`AbstractObjectResourceAdapter`

Базовый класс для реализации адаптера к внешнему фреймворку, реализующий одновременно `ObjectResource` и `ObjectResourceResolver` для большего удобства.

```
@Service(SpringComponentsResourceAdapter.BEAN_NAME)
@ComponentId(SpringComponentsResourceAdapter.COMPONENT_ID)
public final class SpringComponentsResourceAdapter extends
                                AbstractObjectResourceAdapter<String, Object>
                implements ObjectResource<String, Object>, ObjectResourceResolver,
ApplicationContextAware {
  ...
}
```

`AbstractObjectAdapter` Базовый класс для реализации бизнес объекта адаптера к `ObjectResources` дающий прозрачность доступа к модулю, при желании внутренную реализацию можно будет полностью заменить не меняя бизнес логику.

```
public class TestComponentAdapter extends AbstractObjectAdapter<TestComponent, String>
implements
    TestComponent {
  ...
}
```

`ObjectStateManagerAdapter`

Частный случай `ObjectAdapter` для реализации бизнес объекта `ObjectStateManager` адаптера к `ObjectResources`.

`InMemoryObjectResourceResolver`

Реализует `ObjectResourceResolver` с хранением объектов `ObjectResource` в `ConcurrentHashMap`.

```
public class InMemoryObjectResourceResolver implements ObjectResourceResolver,
    Set<ObjectResource<?, ?>> {

  private final Set<ObjectResource<?, ?>> resources = ConcurrentHashMap.newKeySet(0);
  ...
}
```

Заключение

Этот документ не вкоем случае не является исчерпывающей инструкцией по эксплуатации, это лишь краткое описание призванное дать представление о функциях библиотеки, основным источником информации являются сами исходные коды и демо проект.

Это не функциональная библиотека а лишь инструмент для структурирования BL. Идея такой библиотеки возникла при разработке стандартных прикладных программ в которых по мере увеличения функциональности объем кода увеличивается в разы и дальнейшая поддержка превращается в ад. На настоящий момент появилось большое количество фреймворков и обеспечение абстрагирования от них BL является несомненно тоже важной задачей.