Table of Contents

Eine IStorage wird in JVx verwendet um beliebige tabellarische Daten zwischen Client und Server, standardisiert, zu übertragen. In JVx existiert bereits eine Implementierung für Datenbanken, die DBStorage, und für dynamisch zusammengestellte Daten, die AbstractMemStorage. Am Client wird auf die Daten über RemoteDataBooks zugegriffen.

Damit die gewohnten Features wie Meta Daten Caching, Server-Side Triggers, Sortierung, Master/Detail und Export nicht erneut implementiert werden müssen, können die vorhandenen Basisklassen wiederverwendet werden.

Die Klasse AbstractStorage implementiert die Handhabung der Server-Side Triggers. Die AbstractCachedStorage ist eine Ableitung von AbstractStorage und implementiert den Meta Daten Cache.

Die DBStorage leitet von AbstractCachedStorage ab und implementiert die notwendigen Methoden:

```
public abstract void writeCSV(...) throws Exception;

public abstract MetaData executeGetMetaData() throws DataSourceException;

public abstract Object[] executeRefetchRow(...) throws DataSourceException;

public abstract List<Object[]> executeFetch(...) throws DataSourceException;

public abstract Object[] executeInsert(...) throws DataSourceException;

public abstract Object[] executeUpdate(...) throws DataSourceException;

public abstract void executeDelete(...) throws DataSourceException;
```

speziell für Datenbanken.

Wenn die Daten dynamisch zusammengestellt werden sollen, kann auf die AbstractMemStorage zurückgegriffen werden. Diese Klasse verwendet ein MemDataBook um die Daten zu speichern.

Ein Anwendungsbeispiel hierfür wäre die Abfrage von E-Mails. Die Daten könnten in einer EMailStorage für den Client zur Verfügung gestellt werden. Ein weiteres Beispiel wäre eine TwitterStorage.

Ein Auszug aus einer möglichen TwitterStorage Implementierung:

```
@Override
public RowDefinition getRowDefinition() throws ModelException
{
   RowDefinition rowdef = new RowDefinition();
   rowdef.addColumnDefinition(new ColumnDefinition("ID", new
BigDecimalDataType()));
   rowdef.addColumnDefinition(new ColumnDefinition("USER", new
StringDataType()));
   rowdef.addColumnDefinition(new ColumnDefinition("IMAGE", new
BinaryDataType()));
   rowdef.addColumnDefinition(new ColumnDefinition("MESSAGE", new
StringDataType()));
```

```
   rowdef.setPrimaryKeyColumnNames(new String[] {"ID"});

   rowdef.setColumnView(null, new ColumnView(rowdef));

   return rowdef;
}

@Override
public void loadData(MemDataBook pBook, ICondition pFilter) throws
ModelException
{
   pBook.close();
   pBook.open();

   try
   {
      List<Status> liStati = twitter.getUserTimeline(new Paging(1,
iMaxMessages));

       for (Status status : liStati)
       {
          pBook.insert(false);
          pBook.setValues(new String[] {"ID", "USER", "IMAGE", "MESSAGE"},
                          new Object[] {BigDecimal.valueOf(status.getId()),
                                        status.getUser().getName(),
getImage(status.getUser().getProfileImageURL()),
                                        status.getText()});
       }
    }
    catch (Exception ex)
    {
       throw new ModelException("Load data failed!", ex);
    }
}

@Override
public void insert(DataBookEvent pEvent) throws ModelException
{
   try
   {
      Status status =
twitter.updateStatus((String)pEvent.getChangedDataBook().
                                           getValue("MESSAGE"));

      pEvent.getChangedDataBook().setValues(new String[] {"ID", "USER",
"IMAGE"},
                                            new Object[]
{BigDecimal.valueOf(
status.getId()),
status.getUser().getName(),
```

```
getImage(status.getUser())});
    }
    catch (TwitterException te)
    {
       throw new ModelException("Insert failed", te);
    }
}

@Override
public void update(DataBookEvent pEvent) throws ModelException
{
    throw new ModelException("Update is not supported!");
}

@Override
public void delete(DataBookEvent pEvent) throws ModelException
{
    BigDecimal bdId = (BigDecimal)pEvent.getOriginalDataRow().getValue("ID");

    try
    {
       twitter.destroyStatus(bdId.longValue());
    }
    catch (TwitterException te)
    {
       throw new ModelException("Delete failed!", te);
    }
}
```