package cn.dlb.bim.action;

import java.io.ByteArrayOutputStream;

import java.io.IOException;

import org.bson.types.ObjectId;

import org.eclipse.emf.ecore.EClass;

import org.springframework.web.socket.BinaryMessage;

import org.springframework.web.socket.TextMessage;

import org.springframework.web.socket.WebSocketSession;

import com.google.gson.Gson;

import cn.dlb.bim.component.PlatformServer;

import cn.dlb.bim.dao.entity.ConcreteRevision;

import cn.dlb.bim.ifc.database.DatabaseException;

import cn.dlb.bim.ifc.database.queries.om.Include;

import cn.dlb.bim.ifc.database.queries.om.Query;

import cn.dlb.bim.ifc.database.queries.om.QueryException;

import cn.dlb.bim.ifc.database.queries.om.QueryPart;

import cn.dlb.bim.ifc.emf.IfcModelInterfaceException;

import cn.dlb.bim.ifc.emf.PackageMetaData;

import cn.dlb.bim.ifc.serializers.SerializerException;

import cn.dlb.bim.ifc.shared.ProgressReporter;

import cn.dlb.bim.ifc.stream.message.BinaryGeometryMessagingStreamingSerializer;

import cn.dlb.bim.ifc.stream.query.QueryContext;

import cn.dlb.bim.ifc.stream.query.QueryObjectProvider;

import cn.dlb.bim.vo.ProgressVo;

/\*\*

\* @author shenan4321

\* 该类用于向前端输送几何信息

\*/

public class StreamingGeometryQueryAction extends LongAction {

private final PlatformServer server;

private final QueryContext queryContext;

private final ConcreteRevision concreteRevision;

private int lastPercentProcess = 0;

/\*\*

\* 向前端输送几何信息的对象构造

\* @param webSocketSession 对应的会话

\* @param server 服务对象

\* @param queryContext 查询上下文

\* @param concreteRevision 版本

\*/

public StreamingGeometryQueryAction(WebSocketSession webSocketSession, PlatformServer server,

QueryContext queryContext, ConcreteRevision concreteRevision) {

super(webSocketSession);

this.server = server;

this.queryContext = queryContext;

this.concreteRevision = concreteRevision;

}

@Override

public void execute() throws DatabaseException, IfcModelInterfaceException {

@SuppressWarnings("all")

ProgressReporter progressReporter = new ProgressReporter() {

private String title = "";

private long progress;

private long max;

@Override

public void update(long progress, long max) {

this.progress = progress;

this.max = max;

ProgressVo msg = new ProgressVo();

msg.setTitle(title);

msg.setProgress(progress);

msg.setMax(max);

sendWebSocketMessage(msg);

}

@Override

public void setTitle(String title) {

this.title = title;

ProgressVo msg = new ProgressVo();

msg.setTitle(title);

msg.setProgress(progress);

msg.setMax(max);

sendWebSocketMessage(msg);

}

};

try {

PackageMetaData packageMetaData = queryContext.getPackageMetaData();

Query query = new Query(packageMetaData);

QueryPart queryPart = query.createQueryPart();

EClass geometryInfoElcass = packageMetaData.getEClassIncludingDependencies("GeometryInfo");

queryPart.addType(geometryInfoElcass, false);

Include include = queryPart.createInclude();

include.addType(geometryInfoElcass, false);

include.addField("data");

QueryObjectProvider queryObjectProvider = new QueryObjectProvider(queryContext.getPlatformService(), server,

query, queryContext.getRid(), packageMetaData);

BinaryGeometryMessagingStreamingSerializer serializer = new BinaryGeometryMessagingStreamingSerializer();

serializer.init(queryObjectProvider, packageMetaData, concreteRevision);

ByteArrayOutputStream byteOutputStream = new ByteArrayOutputStream();

boolean write = true;

do {

write = serializer.writeMessage(byteOutputStream, progressReporter);

if (byteOutputStream.toByteArray().length > 0) {

BinaryMessage message = new BinaryMessage(byteOutputStream.toByteArray());

webSocketSession.sendMessage(message);

}

byteOutputStream.reset();

} while (write);

webSocketSession.close();

} catch (IOException e) {

e.printStackTrace();

} catch (QueryException e) {

e.printStackTrace();

} catch (SerializerException e) {

e.printStackTrace();

}

}

/\*\*

\* 传输socket信息

\* @param msg

\*/

public void sendWebSocketMessage(ProgressVo msg) {

if (webSocketSession == null || !webSocketSession.isOpen()) {

return;

}

int curPercent = 0;

if (msg.getMax() != 0) {

curPercent = (int) Math.floor(Double.valueOf(msg.getProgress()) / msg.getMax() \* 100);

}

if (lastPercentProcess != curPercent) {

lastPercentProcess = curPercent;

Gson gson = new Gson();

String jsonStr = gson.toJson(msg);

TextMessage message = new TextMessage(jsonStr);

try {

webSocketSession.sendMessage(message);

} catch (Exception e) {

try {

webSocketSession.close();

} catch (IOException e1) {

e1.printStackTrace();

}

e.printStackTrace();

}

}

}

}

package cn.dlb.bim.ifc.database;

import java.nio.ByteBuffer;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.eclipse.emf.ecore.EClass;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.google.common.collect.Iterators;

import cn.dlb.bim.cache.ModelCacheManager;

import cn.dlb.bim.dao.IfcModelDao;

import cn.dlb.bim.dao.entity.IdEObjectEntity;

import cn.dlb.bim.dao.entity.IfcModelEntity;

import cn.dlb.bim.ifc.database.binary.IfcModelBinary;

import cn.dlb.bim.ifc.database.binary.TodoList;

import cn.dlb.bim.ifc.emf.IdEObject;

import cn.dlb.bim.ifc.emf.IdEObjectImpl;

import cn.dlb.bim.ifc.emf.IdEObjectImpl.State;

import cn.dlb.bim.ifc.emf.IfcModelInterface;

import cn.dlb.bim.ifc.emf.IfcModelInterfaceException;

import cn.dlb.bim.ifc.emf.MetaDataManager;

import cn.dlb.bim.ifc.emf.PackageMetaData;

import cn.dlb.bim.ifc.emf.QueryInterface;

import cn.dlb.bim.ifc.model.BasicIfcModel;

import cn.dlb.bim.ifc.shared.ProgressReporter;

import cn.dlb.bim.models.geometry.GeometryData;

import cn.dlb.bim.models.geometry.GeometryInfo;

import cn.dlb.bim.service.PlatformService;

import cn.dlb.bim.vo.ModelInfoVo;

/\*\*

\* @author shenan4321

\* 该类用于存储模型数据到MongoDB

\*/

public class IfcModelDbSession extends IfcModelBinary {

private static final Logger LOGGER = LoggerFactory.getLogger(IfcModelDbSession.class);

private final IfcModelDao ifcModelDao;

private final MetaDataManager metaDataManager;

private ProgressReporter progressReporter;

private ModelCacheManager modelCacheManager;

public IfcModelDbSession(IfcModelDao ifcModelDao, MetaDataManager metaDataManager, PlatformService ifcDataBase,

ProgressReporter progressReporter, ModelCacheManager modelCacheManager) {

super(ifcDataBase);

this.ifcModelDao = ifcModelDao;

this.metaDataManager = metaDataManager;

this.progressReporter = progressReporter;

this.modelCacheManager = modelCacheManager;

}

/\*\*

\* 保存Ifc模型到数据库

\* @param model 模型对象

\* @param modelInfo 模型元信息

\* @throws DatabaseException

\*/

public void saveIfcModel(IfcModelInterface model, ModelInfoVo modelInfo) throws DatabaseException {

IfcModelEntity ifcModelEntity = new IfcModelEntity();

final Integer revisionId = ifcDataBase.newRevisionId();

model.getModelMetaData().setRevisionId(revisionId);

model.fixOids(ifcDataBase);

ifcModelEntity.setModelMetaData(model.getModelMetaData());

ifcModelEntity.setRid(revisionId);

ifcModelEntity.setPid(modelInfo.getPid());

ifcModelEntity.setApplyType(modelInfo.getApplyType());

ifcModelEntity.setName(modelInfo.getName());

ifcModelEntity.setFileName(modelInfo.getFileName());

ifcModelEntity.setFileSize(modelInfo.getFileSize());

ifcModelEntity.setUploadDate(modelInfo.getUploadDate());

Map<Long, IdEObject> geometries = new HashMap<>();

EClass productClass = (EClass) model.getPackageMetaData().getEClassifierCaseInsensitive("IfcProduct");

List<IdEObject> projectList = model.getAllWithSubTypes(productClass);

for (IdEObject ifcProduct : projectList) {

GeometryInfo geometryInfo = (GeometryInfo) ifcProduct.eGet(ifcProduct.eClass().getEStructuralFeature("geometry"));

if (geometryInfo != null && !geometries.containsKey(geometryInfo.getOid())) {

geometries.put(geometryInfo.getOid(), geometryInfo);

GeometryData geometryData = geometryInfo.getData();

if (geometryData != null && !geometries.containsKey(geometryData.getOid())) {

geometries.put(geometryData.getOid(), geometryData);

}

}

}

Iterator<IdEObject> iterator = Iterators.concat(model.getObjects().values().iterator(),

geometries.values().iterator());

List<IdEObjectEntity> idEObjectEntityList = new ArrayList<>();

while (iterator.hasNext()) {

IdEObject object = iterator.next();

ByteBuffer valueBuffer = ByteBuffer.allocate(16);

valueBuffer = convertObjectToByteArray(object, valueBuffer,

metaDataManager.getPackageMetaData(object.eClass().getEPackage().getName()));

IdEObjectEntity idEObjectEntity = new IdEObjectEntity();

idEObjectEntity.setOid(object.getOid());

idEObjectEntity.setRid(revisionId);

idEObjectEntity.setObjectBytes(valueBuffer.array());

idEObjectEntityList.add(idEObjectEntity);

}

ifcModelDao.insertIfcModelEntity(ifcModelEntity);

ifcModelDao.insertAllIdEObjectEntity(idEObjectEntityList);

ifcDataBase.syncOid();

modelCacheManager.cacheModel(revisionId, model);

}

/\*\*

\* 从数据库查询并恢复模型数据

\* @param packageMetaData 包元信息

\* @param rid 模型版本编号

\* @param query 查询接口

\* @return

\* @throws DatabaseException

\* @throws IfcModelInterfaceException

\*/

public IfcModelInterface get(PackageMetaData packageMetaData, int rid, QueryInterface query)

throws DatabaseException, IfcModelInterfaceException {

if (modelCacheManager.contains(rid)) {

return modelCacheManager.getIfcModel(rid);

}

IfcModelInterface model = new BasicIfcModel(packageMetaData);

TodoList todoList = new TodoList();

progressReporterTitle("Querying ifcmodel ...");

IfcModelEntity modelEntity = ifcModelDao.queryIfcModelEntityByRid(rid);

if (modelEntity == null) {

return null;

}

List<IdEObjectEntity> idEObjectEntitys = ifcModelDao.queryIdEObjectEntityByRid(rid);

Long total = Long.valueOf(idEObjectEntitys.size());

progressReporterTitle("Reading objects.");

progressReporterUpdate(0l, total);

Long doneObjectCount = 0l;

for (IdEObjectEntity objectEntity : idEObjectEntitys) {

get(objectEntity, model, query, todoList);

progressReporterUpdate(++doneObjectCount, total);

}

model.setModelMetaDataValue(modelEntity.getModelMetaData());

modelCacheManager.cacheModel(rid, model);

return model;

}

/\*\*

\* 将从数据库获取的单条数据转为ifc对象

\* @param objectEntity 数据库中的单条数据

\* @param model 模型数据

\* @param query 查询接口

\* @param todoList 未完成的ifc对象

\* @return

\* @throws DatabaseException

\*/

@SuppressWarnings("unchecked")

public <T extends IdEObject> T get(IdEObjectEntity objectEntity, IfcModelInterface model, QueryInterface query,

TodoList todoList) throws DatabaseException {

IdEObjectImpl cachedObject = (IdEObjectImpl) objectCache.get(objectEntity.getOid());

if (cachedObject != null) {

if (cachedObject.getLoadingState() == State.LOADED && cachedObject.getRid() != Integer.MAX\_VALUE) {

cachedObject.load();

return (T) cachedObject;

}

}

ByteBuffer valueBuffer = ByteBuffer.wrap(objectEntity.getObjectBytes());

EClass eClass = getEClassForOid(objectEntity.getOid());

int rid = model.getModelMetaData().getRevisionId();

T convertByteArrayToObject = (T) convertByteArrayToObject(eClass, eClass, objectEntity.getOid(), valueBuffer,

model, rid, query, todoList);

objectCache.put(convertByteArrayToObject.getOid(), convertByteArrayToObject);

return convertByteArrayToObject;

}

/\*\*

\* 更新进度标题

\* @param title

\*/

public void progressReporterTitle(String title) {

if (progressReporter != null) {

progressReporter.setTitle(title);

}

}

/\*\*

\* 更新进度

\* @param progress

\* @param max

\*/

public void progressReporterUpdate(Long progress, Long max) {

if (progressReporter != null) {

progressReporter.update(progress, max);

}

}

}

package cn.dlb.bim.ifc.database.binary;

import java.nio.BufferOverflowException;

import java.nio.BufferUnderflowException;

import java.nio.ByteBuffer;

import java.nio.ByteOrder;

import java.util.Date;

import java.util.List;

import org.eclipse.emf.common.util.AbstractEList;

import org.eclipse.emf.common.util.BasicEList;

import org.eclipse.emf.common.util.EList;

import org.eclipse.emf.common.util.Enumerator;

import org.eclipse.emf.ecore.EAttribute;

import org.eclipse.emf.ecore.EClass;

import org.eclipse.emf.ecore.EClassifier;

import org.eclipse.emf.ecore.EDataType;

import org.eclipse.emf.ecore.EEnum;

import org.eclipse.emf.ecore.EEnumLiteral;

import org.eclipse.emf.ecore.EObject;

import org.eclipse.emf.ecore.EReference;

import org.eclipse.emf.ecore.EStructuralFeature;

import org.eclipse.emf.ecore.EcorePackage;

import org.eclipse.emf.ecore.impl.EEnumImpl;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.google.common.base.Charsets;

import cn.dlb.bim.ifc.database.DatabaseException;

import cn.dlb.bim.ifc.database.ObjectCache;

import cn.dlb.bim.ifc.emf.IdEObject;

import cn.dlb.bim.ifc.emf.IdEObjectImpl;

import cn.dlb.bim.ifc.emf.IdEObjectImpl.State;

import cn.dlb.bim.ifc.emf.IfcModelInterface;

import cn.dlb.bim.ifc.emf.IfcModelInterfaceException;

import cn.dlb.bim.ifc.emf.PackageMetaData;

import cn.dlb.bim.ifc.emf.QueryInterface;

import cn.dlb.bim.models.geometry.GeometryPackage;

import cn.dlb.bim.models.ifc2x3tc1.Ifc2x3tc1Package;

import cn.dlb.bim.models.ifc4.Ifc4Package;

import cn.dlb.bim.models.store.StorePackage;

import cn.dlb.bim.service.PlatformService;

import cn.dlb.bim.utils.BinUtils;

/\*\*

\* @author shenan4321

\*/

public class IfcModelBinary {

private static final Logger LOGGER = LoggerFactory.getLogger(IfcModelBinary.class);

protected PlatformService ifcDataBase;

protected final ObjectCache objectCache = new ObjectCache();

public static final int STORE\_PROJECT\_ID = 1;

public IfcModelBinary(PlatformService ifcDataBase) {

this.ifcDataBase = ifcDataBase;

}

/\*\*

\* 将字节码转为对象

\* @param originalQueryClass 源类

\* @param eClass ifc类型

\* @param oid 对象编号

\* @param buffer 字节码对象

\* @param model 模型数据

\* @param rid 模型版本编号

\* @param query 查询接口

\* @param todoList 未完成列表

\* @return

\* @throws DatabaseException

\*/

public IdEObject convertByteArrayToObject(EClass originalQueryClass, EClass eClass, long oid, ByteBuffer buffer, IfcModelInterface model, Integer rid, QueryInterface query, TodoList todoList) throws DatabaseException {

try {

IdEObject idEObject = todoList.get(oid);

todoList.remove(oid);

if (idEObject == null) {

idEObject = createInternal(eClass, query);

((IdEObjectImpl) idEObject).setOid(oid);

((IdEObjectImpl) idEObject).setPid(query.getPid());

}

if (idEObject.eClass().getEAnnotation("wrapped") == null

&& idEObject.eClass().getEAnnotation("hidden") == null) {

try {

model.addAllowMultiModel(oid, idEObject);

} catch (IfcModelInterfaceException e) {

throw new DatabaseException(e);

}

}

((IdEObjectImpl) idEObject).setRid(rid);

((IdEObjectImpl) idEObject).useInverses(false);

if (StorePackage.eINSTANCE == idEObject.eClass().getEPackage()) {

LOGGER.info("Read: " + idEObject.eClass().getName() + " pid=" + query.getPid() + " oid=" + oid);

}

((IdEObjectImpl) idEObject).setLoadingState(State.LOADING);

objectCache.put(oid, idEObject);

int unsettedLength = model.getPackageMetaData().getUnsettedLength(eClass);

byte[] unsetted = new byte[unsettedLength];

buffer.get(unsetted);

int fieldCounter = 0;

for (EStructuralFeature feature : eClass.getEAllStructuralFeatures()) {

try {

if (model.getPackageMetaData().useForDatabaseStorage(eClass, feature)) {

boolean isUnsetted = (unsetted[fieldCounter / 8] & (1 << (fieldCounter % 8))) != 0;

if (isUnsetted) {

if (feature.isUnsettable()) {

idEObject.eUnset(feature);

} else if (feature.isMany()) {

// do nothing

} else if (feature.getDefaultValue() != null) {

idEObject.eSet(feature, feature.getDefaultValue());

}

} else {

if (!query.shouldFollowReference(originalQueryClass, eClass, feature)) {

// we have to do some reading to maintain a correct

// index

fakeRead(buffer, feature);

} else {

Object newValue = null;

if (feature.isMany()) {

newValue = readList(idEObject, buffer, model, query, todoList, feature);

} else {

if (feature.getEType() instanceof EEnum) {

int enumOrdinal = buffer.getInt();

if (enumOrdinal == -1) {

newValue = null;

} else {

EClassifier eType = feature.getEType();

EEnumLiteral enumLiteral = ((EEnumImpl) eType).getEEnumLiteral(enumOrdinal);

if (enumLiteral != null) {

newValue = enumLiteral.getInstance();

}

}

} else if (feature.getEType() instanceof EClass) {

// EReference eReference = (EReference) feature;

buffer.order(ByteOrder.LITTLE\_ENDIAN);

short cid = buffer.getShort();

buffer.order(ByteOrder.BIG\_ENDIAN);

if (cid == -1) {

// null, do nothing

} else if (cid < 0) {

// non minus one and negative cid means value is embedded in record

EClass referenceClass = ifcDataBase.getEClassForCid((short) (-cid));

if (feature.getEAnnotation("dbembed") != null) {

newValue = readEmbeddedValue(feature, buffer, referenceClass, query);

} else {

newValue = readWrappedValue(feature, buffer, referenceClass, query);

}

} else if (cid > 0) {

// positive cid means value is reference to other record

EClass referenceClass = ifcDataBase.getEClassForCid(cid);

if (referenceClass == null) {

throw new DatabaseException("No eClass found for cid " + cid);

}

// readReference is going to read a long, which includes the 2 bytes for the cid

buffer.position(buffer.position() - 2);

newValue = readReference(buffer, model, idEObject, feature, referenceClass, query, todoList);

// if (eReference.getEOpposite() != null &&

// ((IdEObjectImpl)

// newValue).isLoadedOrLoading()) {

// newValue = null;

// }

}

} else if (feature.getEType() instanceof EDataType) {

newValue = readPrimitiveValue(feature.getEType(), buffer, query);

}

}

if (newValue != null) {

idEObject.eSet(feature, newValue);

}

}

}

fieldCounter++;

}

} catch (StringIndexOutOfBoundsException e) {

throw new DatabaseException("Reading " + eClass.getName() + "(" + oid + ")." + feature.getName(), e);

} catch (BufferUnderflowException e) {

throw new DatabaseException("Reading " + eClass.getName() + "(" + oid + ")." + feature.getName(), e);

} catch (BufferOverflowException e) {

throw new DatabaseException("Reading " + eClass.getName() + "(" + oid + ")." + feature.getName(), e);

}

}

((IdEObjectImpl) idEObject).setLoaded();

((IdEObjectImpl) idEObject).useInverses(true);

if (idEObject.getRid() > 100000 || idEObject.getRid() < -100000) {

LOGGER.debug("Improbable rid " + idEObject.getRid() + " - " + idEObject);

}

return idEObject;

} catch (BufferUnderflowException e) {

throw new DatabaseException("Reading " + eClass.getName(), e);

} catch (BufferOverflowException e) {

throw new DatabaseException("Reading " + eClass.getName(), e);

}

}

/\*\*

\* 将对象转为字节码

\* @param object ifc对象

\* @param buffer 字节码

\* @param packageMetaData 类库包

\* @return

\* @throws DatabaseException

\*/

public ByteBuffer convertObjectToByteArray(IdEObject object, ByteBuffer buffer, PackageMetaData packageMetaData) throws DatabaseException {

int bufferSize = getExactSize(object, packageMetaData, true);

if (bufferSize > buffer.capacity()) {

LOGGER.debug("Buffer too small (" + bufferSize + ")");

buffer = ByteBuffer.allocate(bufferSize);

}

int unsettedLength = packageMetaData.getUnsettedLength(object.eClass());

byte[] unsetted = new byte[unsettedLength];

int fieldCounter = 0;

for (EStructuralFeature feature : object.eClass().getEAllStructuralFeatures()) {

if (packageMetaData.useForDatabaseStorage(object.eClass(), feature)) {

if (useUnsetBit(feature, object)) {

unsetted[fieldCounter / 8] |= (1 << (fieldCounter % 8));

}

fieldCounter++;

}

}

buffer.put(unsetted);

EClass eClass = getEClassForOid(object.getOid());

if (!eClass.isSuperTypeOf(object.eClass())) {

throw new DatabaseException("Object with oid " + object.getOid() + " is a " + object.eClass().getName() + " but it's cid-part says it's a " + eClass.getName());

}

for (EStructuralFeature feature : object.eClass().getEAllStructuralFeatures()) {

if (packageMetaData.useForDatabaseStorage(object.eClass(), feature)) {

if (!useUnsetBit(feature, object)) {

if (feature.isMany()) {

writeList(object, buffer, packageMetaData, feature);

} else {

Object value = object.eGet(feature);

if (feature.getEType() instanceof EEnum) {

if (value == null) {

buffer.putInt(-1);

} else {

EEnum eEnum = (EEnum) feature.getEType();

EEnumLiteral eEnumLiteral = eEnum.getEEnumLiteralByLiteral(((Enum<?>) value).toString());

if (eEnumLiteral != null) {

buffer.putInt(eEnumLiteral.getValue());

} else {

LOGGER.error(((Enum<?>) value).toString() + " not found");

buffer.putInt(-1);

}

}

} else if (feature.getEType() instanceof EClass) {

if (value == null) {

buffer.order(ByteOrder.LITTLE\_ENDIAN);

buffer.putShort((short) -1);

buffer.order(ByteOrder.BIG\_ENDIAN);

} else {

IdEObject referencedObject = (IdEObject) value;

EClass referencedClass = referencedObject.eClass();

if (feature.getEAnnotation("dbembed") != null) {

writeEmbeddedValue(object.getPid(), object.getRid(), value, buffer, packageMetaData);

} else if (referencedClass.getEAnnotation("wrapped") != null) {

writeWrappedValue(object.getPid(), object.getRid(), value, buffer, packageMetaData);

} else {

writeReference(object, value, buffer, feature);

}

}

} else if (feature.getEType() instanceof EDataType) {

writePrimitiveValue(feature, value, buffer);

}

}

}

}

}

if (buffer.position() != bufferSize) {

throw new DatabaseException("Value buffer sizes do not match for " + object.eClass().getName() + " " + buffer.position() + "/" + bufferSize);

}

return buffer;

}

/\*\*

\* 从字节码中解析引用

\* @param buffer 字节码

\* @param model 模型对象

\* @param object ifc类对象

\* @param feature 属性

\* @param eClass ifc类

\* @param query 查询接口

\* @param todoList 未完成列表

\* @return

\* @throws DatabaseException

\*/

protected IdEObject readReference(ByteBuffer buffer, IfcModelInterface model, IdEObject object, EStructuralFeature feature, EClass eClass,

QueryInterface query, TodoList todoList) throws DatabaseException {

// TODO next bit seems to make no sense, why detect a deleted record when reading a reference??

if (buffer.capacity() == 1 && buffer.get(0) == -1) {

buffer.position(buffer.position() + 1);

return null;

}

buffer.order(ByteOrder.LITTLE\_ENDIAN);

long oid = buffer.getLong();

buffer.order(ByteOrder.BIG\_ENDIAN);

IdEObject foundInCache = objectCache.get(oid);

if (foundInCache != null) {

return foundInCache;

}

if (model.contains(oid)) {

return model.get(oid);

}

IdEObjectImpl newObject = createInternal(eClass, query);

newObject.setOid(oid);

if (perRecordVersioning(newObject)) {

newObject.setPid(STORE\_PROJECT\_ID);

} else {

newObject.setPid(query.getPid());

}

newObject.setRid(query.getRid());

try {

newObject.setModel(model);

} catch (IfcModelInterfaceException e) {

LOGGER.error("", e);

}

objectCache.put(oid, newObject);

if (query.isDeep() && object.eClass().getEAnnotation("wrapped") == null) {

// if (feature.getEAnnotation("nolazyload") == null) {

todoList.put(oid, newObject);

// }

} else {

if (object.eClass().getEAnnotation("wrapped") == null) {

try {

model.addAllowMultiModel(oid, newObject);

} catch (IfcModelInterfaceException e) {

throw new DatabaseException(e);

}

}

}

return newObject;

}

/\*\*

\* 从字节码中解析list

\* @param idEObject ifc类对象

\* @param buffer 字节码

\* @param model 模型对象

\* @param query 查询接口

\* @param todoList 未完成列表

\* @param feature 对象属性

\* @return

\* @throws DatabaseException

\*/

@SuppressWarnings("unchecked")

protected Object readList(IdEObject idEObject, ByteBuffer buffer, IfcModelInterface model, QueryInterface query, TodoList todoList,

EStructuralFeature feature) throws DatabaseException {

if (feature.getEType() instanceof EEnum) {

} else if (feature.getEType() instanceof EClass) {

if (buffer.capacity() == 1 && buffer.get(0) == -1) {

buffer.position(buffer.position() + 1);

} else {

/\*

\* TODO There still is a problem with this,

\* when readReference (and all calls beyond

\* that call) alter (by opposites) this

\* list, this list can potentially grow too

\* large

\*

\* Only can happen with non-unique

\* references

\*/

int listSize = buffer.getInt();

AbstractEList<Object> list = (AbstractEList<Object>) idEObject.eGet(feature);

for (int i = 0; i < listSize; i++) {

if (feature.getEAnnotation("twodimensionalarray") != null) {

IdEObjectImpl newObject = createInternal((EClass) feature.getEType(), query);

Object result = readList(newObject, buffer, model, query, todoList, newObject.eClass().getEStructuralFeature("List"));

if (result != null) {

newObject.eSet(newObject.eClass().getEStructuralFeature("List"), result);

}

list.addUnique(newObject);

} else {

IdEObject referencedObject = null;

buffer.order(ByteOrder.LITTLE\_ENDIAN);

short cid = buffer.getShort();

buffer.order(ByteOrder.BIG\_ENDIAN);

if (cid == -1) {

// null, do nothing

} else if (cid < 0) {

// negative cid means value is

// embedded

// in record

EClass referenceClass = ifcDataBase.getEClassForCid((short) (-cid));

if (referenceClass == null) {

throw new DatabaseException("No class found for cid " + (-cid));

}

referencedObject = readWrappedValue(feature, buffer, referenceClass, query);

} else if (cid > 0) {

// positive cid means value is a

// reference

// to another record

EClass referenceClass = ifcDataBase.getEClassForCid(cid);

if (referenceClass == null) {

throw new DatabaseException("Cannot find class with cid " + cid);

}

buffer.position(buffer.position() - 2);

referencedObject = readReference(buffer, model, idEObject, feature, referenceClass, query, todoList);

}

if (referencedObject != null) {

if (!feature.getEType().isInstance(referencedObject)) {

throw new DatabaseException(referencedObject.getClass().getSimpleName() + " cannot be stored in list of type "

+ feature.getEType().getName() + " for feature " + feature.getName());

}

if (feature.isUnique()) {

list.add(referencedObject);

} else {

list.addUnique(referencedObject);

}

}

}

}

}

} else if (feature.getEType() instanceof EDataType) {

int listSize = buffer.getInt();

BasicEList<Object> list = new BasicEList<Object>(listSize);

for (int i = 0; i < listSize; i++) {

Object reference = readPrimitiveValue(feature.getEType(), buffer, query);

if (reference != null) {

list.addUnique(reference);

}

}

return list;

}

return null;

}

/\*\*

\* 从字节码中解析内嵌对象

\* @param feature 对应属性

\* @param buffer 字节码

\* @param eClass ifc类

\* @param query 查询接口

\* @return

\*/

protected IdEObject readEmbeddedValue(EStructuralFeature feature, ByteBuffer buffer, EClass eClass, QueryInterface query) {

IdEObject eObject = createInternal(eClass, query);

for (EStructuralFeature eStructuralFeature : eClass.getEAllStructuralFeatures()) {

if (eStructuralFeature.isMany()) {

// Not implemented

} else {

Object primitiveValue = readPrimitiveValue(eStructuralFeature.getEType(), buffer, query);

((IdEObjectImpl) eObject).setLoaded();

eObject.eSet(eStructuralFeature, primitiveValue);

}

}

return eObject;

}

/\*\*

\* 从字节码中读取封装属性值，如IFCValue这类

\* @param feature 对应属性

\* @param buffer 字节码

\* @param eClass ifc类

\* @param query 查询接口

\* @return

\*/

protected IdEObject readWrappedValue(EStructuralFeature feature, ByteBuffer buffer, EClass eClass, QueryInterface query) {

EStructuralFeature eStructuralFeature = eClass.getEStructuralFeature("wrappedValue");

Object primitiveValue = readPrimitiveValue(eStructuralFeature.getEType(), buffer, query);

IdEObject eObject = createInternal(eClass, query);

((IdEObjectImpl) eObject).setLoaded(); // We don't want to go lazy load

// this

eObject.eSet(eStructuralFeature, primitiveValue);

if (eStructuralFeature.getEType() == EcorePackage.eINSTANCE.getEDouble() || eStructuralFeature.getEType() == EcorePackage.eINSTANCE.getEDoubleObject()) {

EStructuralFeature strFeature = eClass.getEStructuralFeature("wrappedValueAsString");

Object stringVal = readPrimitiveValue(EcorePackage.eINSTANCE.getEString(), buffer, query);

eObject.eSet(strFeature, stringVal);

}

return eObject;

}

/\*\*

\* 隐去该属性不读取

\* @param buffer 字节码

\* @param feature 对应属性

\* @throws DatabaseException

\*/

public void fakeRead(ByteBuffer buffer, EStructuralFeature feature) throws DatabaseException {

boolean wrappedValue = feature.getEType().getEAnnotation("wrapped") != null;

if (feature.isMany()) {

if (feature.getEType() instanceof EEnum) {

} else if (feature.getEType() instanceof EClass) {

if (buffer.capacity() == 1 && buffer.get(0) == -1) {

buffer.position(buffer.position() + 1);

} else {

int listSize = buffer.getInt();

for (int i = 0; i < listSize; i++) {

buffer.order(ByteOrder.LITTLE\_ENDIAN);

short cid = buffer.getShort();

buffer.order(ByteOrder.BIG\_ENDIAN);

if (cid != -1) {

if (wrappedValue) {

EClass eClass = (EClass) feature.getEType();

fakePrimitiveRead(eClass.getEStructuralFeature("wrappedValue").getEType(), buffer);

} else {

buffer.position(buffer.position() + 6);

}

}

}

}

} else if (feature.getEType() instanceof EDataType) {

int listSize = buffer.getInt();

for (int i = 0; i < listSize; i++) {

fakePrimitiveRead(feature.getEType(), buffer);

}

}

} else {

if (feature.getEType() instanceof EEnum) {

buffer.position(buffer.position() + 4);

} else if (feature.getEType() instanceof EClass) {

if (buffer.capacity() == 1 && buffer.get(0) == -1) {

buffer.position(buffer.position() + 1);

} else {

buffer.order(ByteOrder.LITTLE\_ENDIAN);

short cid = buffer.getShort();

buffer.order(ByteOrder.BIG\_ENDIAN);

if (wrappedValue) {

fakePrimitiveRead(feature.getEType(), buffer);

} else {

if (cid != -1) {

buffer.position(buffer.position() + 6);

}

}

}

} else if (feature.getEType() instanceof EDataType) {

fakePrimitiveRead(feature.getEType(), buffer);

}

}

}

/\*\*

\* 隐去基础属性位的读取

\* @param classifier 对应类型

\* @param buffer 字节码

\* @throws DatabaseException

\*/

protected void fakePrimitiveRead(EClassifier classifier, ByteBuffer buffer) throws DatabaseException {

if (classifier == EcorePackage.eINSTANCE.getEString()) {

int length = buffer.getInt();

if (length != -1) {

buffer.position(buffer.position() + length);

}

} else if (classifier == EcorePackage.eINSTANCE.getEInt() || classifier == EcorePackage.eINSTANCE.getEIntegerObject()) {

buffer.position(buffer.position() + 4);

} else if (classifier == EcorePackage.eINSTANCE.getELong() || classifier == EcorePackage.eINSTANCE.getELongObject()) {

buffer.position(buffer.position() + 8);

} else if (classifier == EcorePackage.eINSTANCE.getEFloat() || classifier == EcorePackage.eINSTANCE.getEFloatObject()) {

buffer.position(buffer.position() + 4);

} else if (classifier == EcorePackage.eINSTANCE.getEDouble() || classifier == EcorePackage.eINSTANCE.getEDoubleObject()) {

buffer.position(buffer.position() + 8);

} else if (classifier == EcorePackage.eINSTANCE.getEBoolean() || classifier == EcorePackage.eINSTANCE.getEBooleanObject()) {

buffer.position(buffer.position() + 1);

} else if (classifier == EcorePackage.eINSTANCE.getEDate()) {

buffer.position(buffer.position() + 8);

} else if (classifier == EcorePackage.eINSTANCE.getEByteArray()) {

int length = buffer.getInt();

if (length != -1) {

buffer.position(buffer.position() + length);

}

} else {

throw new DatabaseException("Unimplemented " + classifier);

}

}

/\*\*

\* 读取基础类型数值

\* @param classifier 对于类型

\* @param buffer 字节码

\* @param query 查询接口

\* @return

\*/

public Object readPrimitiveValue(EClassifier classifier, ByteBuffer buffer, QueryInterface query) {

if (classifier == EcorePackage.eINSTANCE.getEString()) {

int length = buffer.getInt();

if (length != -1) {

return BinUtils.readString(buffer, length);

} else {

return null;

}

} else if (classifier == EcorePackage.eINSTANCE.getEInt() || classifier == EcorePackage.eINSTANCE.getEIntegerObject()) {

return buffer.getInt();

} else if (classifier == EcorePackage.eINSTANCE.getELong() || classifier == EcorePackage.eINSTANCE.getELongObject()) {

return buffer.getLong();

} else if (classifier == EcorePackage.eINSTANCE.getEFloat() || classifier == EcorePackage.eINSTANCE.getEFloatObject()) {

return buffer.getFloat();

} else if (classifier == EcorePackage.eINSTANCE.getEDouble() || classifier == EcorePackage.eINSTANCE.getEDoubleObject()) {

return buffer.getDouble();

} else if (classifier == EcorePackage.eINSTANCE.getEBoolean() || classifier == EcorePackage.eINSTANCE.getEBooleanObject()) {

return buffer.get() == 1;

} else if (classifier == EcorePackage.eINSTANCE.getEDate()) {

long val = buffer.getLong();

if (val == -1L) {

return null;

}

return new Date(val);

} else if (classifier == EcorePackage.eINSTANCE.getEByteArray()) {

int size = buffer.getInt();

byte[] result = new byte[size];

buffer.get(result);

return result;

} else if (classifier.getName().equals("Tristate")) {

int ordinal = buffer.getInt();

EEnum tristateEnum = query.getPackageMetaData().getEEnum("Tristate");

return tristateEnum.getEEnumLiteral(ordinal).getInstance();

} else if (classifier instanceof EEnum) {

int ordinal = buffer.getInt();

EEnum eEnum = (EEnum) classifier;

return eEnum.getEEnumLiteral(ordinal).getInstance();

} else {

throw new RuntimeException("Unsupported type " + classifier.getName());

}

}

/\*\*

\* 将基础类型数值写入字节码

\* @param classifier 对应类型

\* @param buffer 字节码

\* @param query 查询接口

\* @return

\*/

public byte[] readPrimitiveBytes(EClassifier classifier, ByteBuffer buffer, QueryInterface query) {

if (classifier == EcorePackage.eINSTANCE.getEString()) {

int length = buffer.getInt();

if (length != -1) {

byte[] result = new byte[length];

buffer.get(result, 0, length);

return result;

} else {

return null;

}

} else if (classifier == EcorePackage.eINSTANCE.getEInt() || classifier == EcorePackage.eINSTANCE.getEIntegerObject()) {

byte[] result = new byte[4];

buffer.get(result, 0, 4);

return result;

} else if (classifier == EcorePackage.eINSTANCE.getELong() || classifier == EcorePackage.eINSTANCE.getELongObject()) {

byte[] result = new byte[8];

buffer.get(result, 0, 8);

return result;

} else if (classifier == EcorePackage.eINSTANCE.getEFloat() || classifier == EcorePackage.eINSTANCE.getEFloatObject()) {

byte[] result = new byte[4];

buffer.get(result, 0, 4);

return result;

} else if (classifier == EcorePackage.eINSTANCE.getEDouble() || classifier == EcorePackage.eINSTANCE.getEDoubleObject()) {

byte[] result = new byte[8];

buffer.get(result, 0, 8);

return result;

} else if (classifier == EcorePackage.eINSTANCE.getEBoolean() || classifier == EcorePackage.eINSTANCE.getEBooleanObject()) {

byte[] result = new byte[1];

buffer.get(result, 0, 1);

return result;

} else if (classifier == EcorePackage.eINSTANCE.getEDate()) {

byte[] result = new byte[8];

buffer.get(result, 0, 8);

return result;

} else if (classifier == EcorePackage.eINSTANCE.getEByteArray()) {

int size = buffer.getInt();

byte[] result = new byte[size];

buffer.get(result);

return result;

} else {

throw new RuntimeException("Unsupported type " + classifier.getName());

}

}

/\*\*

\* 写入内嵌ifc对象值

\* @param pid 项目编号

\* @param rid 模型版本编号

\* @param value 内嵌对象

\* @param buffer 字节码

\* @param packageMetaData 类库包元数据

\* @throws DatabaseException

\*/

protected void writeEmbeddedValue(int pid, int rid, Object value, ByteBuffer buffer, PackageMetaData packageMetaData) throws DatabaseException {

IdEObject wrappedValue = (IdEObject) value;

Short cid = ifcDataBase.getCidOfEClass(wrappedValue.eClass());

buffer.order(ByteOrder.LITTLE\_ENDIAN);

buffer.putShort((short) -cid);

buffer.order(ByteOrder.BIG\_ENDIAN);

for (EStructuralFeature eStructuralFeature : wrappedValue.eClass().getEAllStructuralFeatures()) {

if (eStructuralFeature.isMany()) {

writeList(wrappedValue, buffer, packageMetaData, eStructuralFeature);

} else {

writePrimitiveValue(eStructuralFeature, wrappedValue.eGet(eStructuralFeature), buffer);

}

}

}

/\*\*

\* 写入List

\* @param object 对应ifc类对象

\* @param buffer 字节码

\* @param packageMetaData ifc类库元数据

\* @param feature 属性

\* @throws DatabaseException

\*/

protected void writeList(IdEObject object, ByteBuffer buffer, PackageMetaData packageMetaData, EStructuralFeature feature) throws DatabaseException {

if (feature.getEType() instanceof EEnum) {

// Aggregate relations to enums never occur... at this

// moment

} else if (feature.getEType() instanceof EClass) {

EList<?> list = (EList<?>) object.eGet(feature);

buffer.putInt(list.size());

for (Object o : list) {

if (o == null) {

buffer.order(ByteOrder.LITTLE\_ENDIAN);

buffer.putShort((short) -1);

buffer.order(ByteOrder.BIG\_ENDIAN);

} else {

IdEObject listObject = (IdEObject) o;

if (listObject.eClass().getEAnnotation("wrapped") != null || listObject.eClass().getEStructuralFeature("wrappedValue") != null) {

writeWrappedValue(object.getPid(), object.getRid(), listObject, buffer, packageMetaData);

} else if (feature.getEAnnotation("twodimensionalarray") != null) {

EStructuralFeature lf = listObject.eClass().getEStructuralFeature("List");

writeList(listObject, buffer, packageMetaData, lf);

} else {

writeReference(object, listObject, buffer, feature);

}

}

}

} else if (feature.getEType() instanceof EDataType) {

EList<?> list = (EList<?>) object.eGet(feature);

buffer.putInt(list.size());

for (Object o : list) {

writePrimitiveValue(feature, o, buffer);

}

}

}

/\*\*

\* 写入基础数据类型值

\* @param feature 对应属性

\* @param value 对应值

\* @param buffer 字节码

\* @throws DatabaseException

\*/

protected void writePrimitiveValue(EStructuralFeature feature, Object value, ByteBuffer buffer) throws DatabaseException {

EClassifier type = feature.getEType();

if (type == EcorePackage.eINSTANCE.getEString()) {

if (value == null) {

buffer.putInt(-1);

} else {

String stringValue = (String) value;

byte[] bytes = stringValue.getBytes(Charsets.UTF\_8);

if (bytes.length > Integer.MAX\_VALUE) {

throw new DatabaseException("String value too long (max length is " + Integer.MAX\_VALUE + ")");

}

buffer.putInt(bytes.length);

buffer.put(bytes);

}

} else if (type == EcorePackage.eINSTANCE.getEInt() || type == EcorePackage.eINSTANCE.getEIntegerObject()) {

if (value == null) {

buffer.putInt(0);

} else {

buffer.putInt((Integer) value);

}

} else if (type == EcorePackage.eINSTANCE.getEDouble() || type == EcorePackage.eINSTANCE.getEDoubleObject()) {

if (value == null) {

buffer.putDouble(0D);

} else {

buffer.putDouble((Double) value);

}

} else if (type == EcorePackage.eINSTANCE.getEFloat() || type == EcorePackage.eINSTANCE.getEFloatObject()) {

if (value == null) {

buffer.putFloat(0F);

} else {

buffer.putFloat((Float) value);

}

} else if (type == EcorePackage.eINSTANCE.getELong() || type == EcorePackage.eINSTANCE.getELongObject()) {

if (value == null) {

buffer.putLong(0L);

} else {

buffer.putLong((Long) value);

}

} else if (type == EcorePackage.eINSTANCE.getEBoolean() || type == EcorePackage.eINSTANCE.getEBooleanObject()) {

if (value == null) {

buffer.put((byte) 0);

} else {

buffer.put(((Boolean) value) ? (byte) 1 : (byte) 0);

}

} else if (type == EcorePackage.eINSTANCE.getEDate()) {

if (value == null) {

buffer.putLong(-1L);

} else {

buffer.putLong(((Date) value).getTime());

}

} else if (type.getName().equals("Tristate")) {

Enumerator eEnumLiteral = (Enumerator) value;

buffer.putInt(eEnumLiteral.getValue());

} else if (value instanceof Enumerator) {

Enumerator eEnumLiteral = (Enumerator) value;

buffer.putInt(eEnumLiteral.getValue());

} else if (type == EcorePackage.eINSTANCE.getEByteArray()) {

if (value == null) {

buffer.putInt(0);

} else {

byte[] bytes = (byte[]) value;

buffer.putInt(bytes.length);

buffer.put(bytes);

}

} else {

throw new RuntimeException("Unsupported type " + type.getName());

}

}

/\*\*

\* 写入引用类型数据

\* @param object 对应ifc对象

\* @param value 被引用ifc对象

\* @param buffer 字节码

\* @param feature 对应属性

\* @throws DatabaseException

\*/

protected void writeReference(IdEObject object, Object value, ByteBuffer buffer, EStructuralFeature feature) throws DatabaseException {

IdEObject idEObject = (IdEObject) value;

if (idEObject.getOid() < 0) {

throw new DatabaseException("Writing a reference with oid " + idEObject.getOid() + ", this is not supposed to happen, referenced: " + idEObject.getOid() + " " + value + " from " + object.getOid() + " " + object);

}

buffer.order(ByteOrder.LITTLE\_ENDIAN);

buffer.putLong(idEObject.getOid());

buffer.order(ByteOrder.BIG\_ENDIAN);

}

/\*\*

\* 写入包装类数值

\* @param pid 项目编号

\* @param rid 模型版本编号

\* @param value 写入的数值

\* @param buffer 字节码

\* @param packageMetaData ifc类库元数据

\* @throws DatabaseException

\*/

protected void writeWrappedValue(int pid, int rid, Object value, ByteBuffer buffer, PackageMetaData packageMetaData) throws DatabaseException {

IdEObject wrappedValue = (IdEObject) value;

EStructuralFeature eStructuralFeature = wrappedValue.eClass().getEStructuralFeature("wrappedValue");

Short cid = ifcDataBase.getCidOfEClass(wrappedValue.eClass());

buffer.order(ByteOrder.LITTLE\_ENDIAN);

buffer.putShort((short) -cid);

buffer.order(ByteOrder.BIG\_ENDIAN);

writePrimitiveValue(eStructuralFeature, wrappedValue.eGet(eStructuralFeature), buffer);

if (eStructuralFeature.getEType() == EcorePackage.eINSTANCE.getEDouble() || eStructuralFeature.getEType() == EcorePackage.eINSTANCE.getEDoubleObject()) {

EStructuralFeature fe = wrappedValue.eClass().getEStructuralFeature("wrappedValueAsString");

writePrimitiveValue(fe, wrappedValue.eGet(fe), buffer);

}

if (wrappedValue.eClass().getName().equals("IfcGloballyUniqueId")) {

EClass eClass = packageMetaData.getEClass("IfcGloballyUniqueId");

if (wrappedValue.getOid() == -1) {//TODO

((IdEObjectImpl) wrappedValue).setOid(ifcDataBase.newOid(eClass));

}

ByteBuffer valueBuffer = convertObjectToByteArray(wrappedValue, ByteBuffer.allocate(getExactSize(wrappedValue, packageMetaData, true)), packageMetaData);

ByteBuffer keyBuffer = createKeyBuffer(pid, wrappedValue.getOid(), rid);

// try {

// database.getKeyValueStore().storeNoOverwrite(eClass.getEPackage().getName() + "\_" + eClass.getName(),

// keyBuffer.array(), valueBuffer.array(), this);

// database.incrementCommittedWrites(1);

// } catch (BimserverLockConflictException e) {

// LOGGER.error("", e);

// }

}

}

/\*\*

\* 获取需要预留的字节码长度

\* @param idEObject ifc类对象

\* @param packageMetaData ifc类库元信息

\* @param useUnsetBits 是否用非可设置位

\* @return

\*/

protected int getExactSize(IdEObject idEObject, PackageMetaData packageMetaData, boolean useUnsetBits) {

int size = 0;

int bits = 0;

if (idEObject.getExpressId() == 15723) {

System.out.println();

}

for (EStructuralFeature eStructuralFeature : idEObject.eClass().getEAllStructuralFeatures()) {

if (packageMetaData.useForDatabaseStorage(idEObject.eClass(), eStructuralFeature)) {

bits++;

if (!useUnsetBits || !useUnsetBit(eStructuralFeature, idEObject)) {

Object val = idEObject.eGet(eStructuralFeature);

if (eStructuralFeature instanceof EAttribute) {

EAttribute eAttribute = (EAttribute) eStructuralFeature;

if (eAttribute.isMany()) {

size += 4;

for (Object v : ((List<?>) val)) {

size += getPrimitiveSize(eAttribute.getEAttributeType(), v);

}

} else {

size += getPrimitiveSize(eAttribute.getEAttributeType(), val);

}

} else if (eStructuralFeature instanceof EReference) {

EReference eReference = (EReference) eStructuralFeature;

if (eReference.isMany()) {

size += 4;

for (Object v : ((List<?>) val)) {

size += getWrappedValueSize(v, eReference, packageMetaData);

}

} else {

size += getWrappedValueSize(val, eReference, packageMetaData);

}

}

}

}

}

if (useUnsetBits) {

size += (int) Math.ceil(bits / 8.0);

}

return size;

}

/\*\*

\* 是否用非可设置位

\* @param feature 对应属性

\* @param object 对应ifc类对象

\* @return

\*/

protected boolean useUnsetBit(EStructuralFeature feature, IdEObject object) {

// TODO non-unsettable boolean values can also be stored in these bits

Object value = object.eGet(feature);

if (feature.isUnsettable()) {

if (!object.eIsSet(feature)) {

return true;

}

if (feature.isMany() && ((List<?>)value).isEmpty()) {

return true;

}

} else {

if (feature.isMany() && ((List<?>)value).isEmpty()) {

return true;

}

if (feature.getDefaultValue() == value || (feature.getDefaultValue() != null && feature.getDefaultValue().equals(value))) {

return true;

}

}

return false;

}

/\*\*

\* 计算基础类型占用字节码位数

\* @param eDataType 数据类型

\* @param val 数值

\* @return

\*/

protected int getPrimitiveSize(EDataType eDataType, Object val) {

if (eDataType == EcorePackage.eINSTANCE.getEInt() || eDataType == EcorePackage.eINSTANCE.getEIntegerObject()) {

return 4;

} else if (eDataType == EcorePackage.eINSTANCE.getEFloat() || eDataType == EcorePackage.eINSTANCE.getEFloatObject()) {

return 4;

} else if (eDataType == EcorePackage.eINSTANCE.getEBoolean() || eDataType == EcorePackage.eINSTANCE.getEBooleanObject()) {

return 1;

} else if (eDataType == EcorePackage.eINSTANCE.getEDate()) {

return 8;

} else if (eDataType == EcorePackage.eINSTANCE.getELong() || eDataType == EcorePackage.eINSTANCE.getELongObject()) {

return 8;

} else if (eDataType == EcorePackage.eINSTANCE.getEDouble() || eDataType == EcorePackage.eINSTANCE.getEDoubleObject()) {

return 8;

} else if (eDataType == EcorePackage.eINSTANCE.getEString()) {

if (val != null) {

return 4 + ((String) val).getBytes(Charsets.UTF\_8).length;

}

return 4;

} else if (eDataType == EcorePackage.eINSTANCE.getEByteArray()) {

if (val != null) {

return 4 + ((byte[]) val).length;

}

return 4;

} else if (eDataType instanceof EEnum) {

return 4;

}

throw new RuntimeException("Unimplemented: " + eDataType);

}

/\*\*

\* 计算包装类占用字节位数

\* @param val 数值

\* @param eReference 应用类型

\* @param packageMetaData ifc类库元信息

\* @return

\*/

protected int getWrappedValueSize(Object val, EReference eReference, PackageMetaData packageMetaData) {

if (val == null) {

return 2;

}

if (val instanceof EObject) {

IdEObject eObject = (IdEObject) val;

if (eReference.getEAnnotation("twodimensionalarray") != null) {

int refSize = 4;

EStructuralFeature eStructuralFeature = eObject.eClass().getEStructuralFeature("List");

List<?> l = (List<?>)eObject.eGet(eStructuralFeature);

for (Object o : l) {

if (o instanceof EObject) {//linfujun: i changed here

IdEObject subEObject = (IdEObject) o;

if (subEObject.eClass().getEAnnotation("wrapped") != null) {

IdEObject wrappedValue = (IdEObject) subEObject;

EStructuralFeature wrappedValueFeature = wrappedValue.eClass().getEStructuralFeature("wrappedValue");

Object wrappedVal = subEObject.eGet(wrappedValueFeature);

refSize += 2 + getPrimitiveSize((EDataType) wrappedValueFeature.getEType(), wrappedVal);

if (wrappedValueFeature.getEType() == EcorePackage.eINSTANCE.getEDouble() || wrappedValueFeature.getEType() == EcorePackage.eINSTANCE.getEDoubleObject()) {

EStructuralFeature wrappedStringFeature = wrappedValue.eClass().getEStructuralFeature("wrappedValueAsString");

String str = (String) subEObject.eGet(wrappedStringFeature);

refSize += getPrimitiveSize(EcorePackage.eINSTANCE.getEString(), str);

}

} else {

refSize += 8;

}

// refSize += 8;

} else {

refSize += getPrimitiveSize((EDataType) eStructuralFeature.getEType(), o);

}

}

return refSize;

} else if (eReference.getEAnnotation("dbembed") != null) {

int refSize = 2;

refSize += getExactSize(eObject, packageMetaData, false);

return refSize;

} else if (eObject.eClass().getEAnnotation("wrapped") != null) {

IdEObject wrappedValue = (IdEObject) val;

EStructuralFeature wrappedValueFeature = wrappedValue.eClass().getEStructuralFeature("wrappedValue");

Object wrappedVal = eObject.eGet(wrappedValueFeature);

int refSize = 2 + getPrimitiveSize((EDataType) wrappedValueFeature.getEType(), wrappedVal);

if (wrappedValueFeature.getEType() == EcorePackage.eINSTANCE.getEDouble() || wrappedValueFeature.getEType() == EcorePackage.eINSTANCE.getEDoubleObject()) {

EStructuralFeature wrappedStringFeature = wrappedValue.eClass().getEStructuralFeature("wrappedValueAsString");

String str = (String) eObject.eGet(wrappedStringFeature);

refSize += getPrimitiveSize(EcorePackage.eINSTANCE.getEString(), str);

}

return refSize;

} else {

return 8;

}

} else {

throw new RuntimeException("Programming error, cannot happen");

}

}

/\*\*

\* 创建ifc类对象

\* @param eClass 对应ifc类

\* @param queryInterface 查询接口

\* @return

\*/

protected IdEObjectImpl createInternal(EClass eClass, QueryInterface queryInterface) {

IdEObjectImpl object = (IdEObjectImpl) eClass.getEPackage().getEFactoryInstance().create(eClass);

object.setQueryInterface(queryInterface);

return object;

}

/\*\*

\* 根据oid查询属于哪个ifc类

\* @param oid 对象编号

\* @return

\* @throws DatabaseException

\*/

public EClass getEClassForOid(long oid) throws DatabaseException {

return ifcDataBase.getEClassForOid(oid);

}

/\*\*

\* 填入键字节码

\* @param buffer 字节码

\* @param object ifc类对象

\* @return

\*/

@SuppressWarnings("unused")

protected ByteBuffer fillKeyBuffer(ByteBuffer buffer, IdEObject object) {

if (object.getRid() > 100000 || object.getRid() < -100000) {

LOGGER.debug("Improbable rid: " + object.getRid() + " - " + object);

}

return fillKeyBuffer(buffer, object.getPid(), object.getOid(), object.getRid());

}

/\*\*

\* 填入键字节码

\* @param buffer 字节码

\* @param pid 项目编号

\* @param oid 对象编号

\* @param rid 模型版本编号

\* @return

\*/

protected ByteBuffer fillKeyBuffer(ByteBuffer buffer, int pid, long oid, int rid) {

buffer.position(0);

buffer.putInt(pid);

buffer.putLong(oid);

buffer.putInt(-rid);

return buffer;

}

/\*\*

\* 创建并填入键字节码

\* @param pid 项目编号

\* @param oid 对象编号

\* @param rid 模型版本编号

\* @return

\*/

protected ByteBuffer createKeyBuffer(int pid, long oid, int rid) {

ByteBuffer keyBuffer = ByteBuffer.allocate(16);

fillKeyBuffer(keyBuffer, pid, oid, rid);

return keyBuffer;

}

/\*\*

\* 创建并填入键字节码

\* @param pid 项目编号

\* @param oid 对象编号

\* @return

\*/

protected ByteBuffer createKeyBuffer(int pid, long oid) {

ByteBuffer keyBuffer = ByteBuffer.allocate(12);

fillKeyBuffer(keyBuffer, pid, oid);

return keyBuffer;

}

/\*\*

\* 填入键字节码

\* @param buffer 字节码

\* @param pid 项目编号

\* @param oid 对象编号

\* @return

\*/

protected ByteBuffer fillKeyBuffer(ByteBuffer buffer, int pid, long oid) {

buffer.position(0);

buffer.putInt(pid);

buffer.putLong(oid);

return buffer;

}

public boolean perRecordVersioning(IdEObject idEObject) {

return perRecordVersioning(idEObject.eClass());

}

public static boolean perRecordVersioning(EClass eClass) {

return eClass.getEPackage() != Ifc2x3tc1Package.eINSTANCE && eClass.getEPackage() != Ifc4Package.eINSTANCE && eClass.getEPackage() != GeometryPackage.eINSTANCE;

}

}

package cn.dlb.bim.lucene;

import java.io.File;

import java.io.IOException;

import java.io.StringReader;

import java.util.List;

import org.apache.lucene.analysis.Analyzer;

import org.apache.lucene.analysis.TokenStream;

import org.apache.lucene.document.Document;

import org.apache.lucene.index.DirectoryReader;

import org.apache.lucene.index.IndexReader;

import org.apache.lucene.index.IndexWriter;

import org.apache.lucene.index.IndexWriterConfig;

import org.apache.lucene.index.Term;

import org.apache.lucene.queryparser.classic.QueryParser;

import org.apache.lucene.search.IndexSearcher;

import org.apache.lucene.search.Query;

import org.apache.lucene.search.ScoreDoc;

import org.apache.lucene.search.highlight.Highlighter;

import org.apache.lucene.search.highlight.InvalidTokenOffsetsException;

import org.apache.lucene.search.highlight.QueryScorer;

import org.apache.lucene.search.highlight.SimpleHTMLFormatter;

import org.apache.lucene.store.Directory;

import org.apache.lucene.store.FSDirectory;

import org.apache.lucene.util.Version;

import org.wltea.analyzer.lucene.IKAnalyzer;

public abstract class AbstractLuceneSearch<T> {

public File indexDir; // 存放索引文件的目录

protected static Analyzer analyzer = new IKAnalyzer(); // 分词器

public static Version LUCENE\_VERSION = Version.LUCENE\_47;

public AbstractLuceneSearch(File indexDir) {

this.indexDir = indexDir;

}

/\*\*

\* 为数据库检索数据创建索引

\*

\* @param <T>

\*/

public void createIndex(List<T> items) {

Directory directory = null;

IndexWriter indexWriter = null;

try {

directory = FSDirectory.open(indexDir);

IndexWriterConfig config = new IndexWriterConfig(LUCENE\_VERSION, analyzer);

config.setOpenMode(IndexWriterConfig.OpenMode.CREATE\_OR\_APPEND);// 设置打开索引模式为创建或追加

indexWriter = new IndexWriter(directory, config);

// 装配成document

List<Document> docs = getDoc(items);

for (Document doc : docs) {

indexWriter.addDocument(doc);

}

} catch (IOException e) {

e.printStackTrace();

} finally {

destoryIndexWriter(indexWriter);

destoryFSDirectory(directory);

}

}

/\*\*

\* 更新索引

\* @param items

\* @param tern 需要更新的term

\*/

public void insertOrUpdateIndex(List<T> items, Term tern) {

Directory directory = null;

IndexWriter indexWriter = null;

try {

directory = FSDirectory.open(indexDir);

IndexWriterConfig config = new IndexWriterConfig(LUCENE\_VERSION, analyzer);

config.setOpenMode(IndexWriterConfig.OpenMode.CREATE\_OR\_APPEND);// 设置打开索引模式为创建或追加

indexWriter = new IndexWriter(directory, config);

// 装配成document

List<Document> docs = getDoc(items);

indexWriter.updateDocuments(tern, docs);

} catch (IOException e) {

e.printStackTrace();

} finally {

destoryIndexWriter(indexWriter);

destoryFSDirectory(directory);

}

}

/\*\*

\* 搜索索引

\*

\* @param queryStr

\* @param queryField

\* @return

\*/

public List<T> search(String queryStr, String queryField, int limit) {

List<T> hitItem = null;

IndexReader reader = null;

IndexSearcher indexSearcher = null;

try {

indexSearcher = getIndexSearcher();

QueryParser parser = new QueryParser(LUCENE\_VERSION, queryField, analyzer);

Query query = parser.parse(queryStr);

ScoreDoc[] hits = indexSearcher.search(query, limit).scoreDocs;

hitItem = toBean(indexSearcher, query, hits);

} catch (Exception e) {

e.printStackTrace();

} finally {

if (reader != null)

try {

reader.close();

} catch (IOException e) {

e.printStackTrace();

}

}

return hitItem;

}

/\*\*

\* 删除索引

\* @param query

\*/

public void delete(Term delTerm) {

Directory directory = null;

IndexWriter indexWriter = null;

try {

directory = FSDirectory.open(indexDir);

IndexWriterConfig config = new IndexWriterConfig(LUCENE\_VERSION, analyzer);

config.setOpenMode(IndexWriterConfig.OpenMode.CREATE\_OR\_APPEND);// 设置打开索引模式为创建或追加

indexWriter = new IndexWriter(directory, config);

indexWriter.deleteDocuments(delTerm);

} catch (IOException e) {

e.printStackTrace();

} finally {

destoryIndexWriter(indexWriter);

destoryFSDirectory(directory);

}

}

private void destoryIndexWriter(IndexWriter writer) {

if (writer == null) {

return;

}

try {

writer.close();

} catch (IOException e) {

e.printStackTrace();

}

}

private void destoryFSDirectory(Directory directory) {

if (directory == null) {

return;

}

try {

directory.close();

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* 装配成document对象

\*

\* @param items

\* @return

\*/

public abstract List<Document> getDoc(List<T> items);

/\*\*

\* 将搜索结果还原成Bean

\*

\* @param indexSearcher

\* @param query

\* @param hits

\* @return

\*/

public abstract List<T> toBean(IndexSearcher indexSearcher, Query query, ScoreDoc[] hits);

/\*\*

\* 高亮设置

\*

\* @param query

\* @param doc

\* @param field

\* @return

\*/

protected String toHighlighter(Query query, Document doc, String field) {

try {

SimpleHTMLFormatter simpleHtmlFormatter = new SimpleHTMLFormatter("<font color=\"blue\">", "</font>");

Highlighter highlighter = new Highlighter(simpleHtmlFormatter, new QueryScorer(query));

TokenStream tokenStream1 = analyzer.tokenStream("text", new StringReader(doc.get(field)));

String highlighterStr = highlighter.getBestFragment(tokenStream1, doc.get(field));

return highlighterStr == null ? doc.get(field) : highlighterStr;

} catch (IOException e) {

e.printStackTrace();

} catch (InvalidTokenOffsetsException e) {

e.printStackTrace();

}

return null;

}

private IndexSearcher getIndexSearcher() {

IndexSearcher search = null;

String key = indexDir.getPath();

if (LuceneCacheHelper.exist(key)) {

search = LuceneCacheHelper.get(key);

} else {

IndexReader reader;

try {

reader = DirectoryReader.open(FSDirectory.open(indexDir));

search = new IndexSearcher(reader);

LuceneCacheHelper.insert(key, search);

} catch (IOException e) {

e.printStackTrace();

}

}

return search;

}

}

package cn.dlb.bim.lucene;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import org.apache.lucene.document.Document;

import org.apache.lucene.document.Field;

import org.apache.lucene.document.FieldType;

import org.apache.lucene.search.IndexSearcher;

import org.apache.lucene.search.Query;

import org.apache.lucene.search.ScoreDoc;

/\*\*

\* 对ifcProduct的记录进行全文索引

\* @author Administrator

\*

\*/

public class IfcProductRecordTextSearch extends AbstractLuceneSearch<IfcProductRecordText> {

public static final String Key\_Oid = "oid";

public static final String Key\_Location = "location";

public static final String Key\_Type = "type";

public static final String Key\_Name = "name";

public static final String Key\_Detail = "detail";

/\*\*

\* 构造方法

\* @param indexDir 索引文件

\*/

public IfcProductRecordTextSearch(File indexDir) {

super(indexDir);

}

/\*\*

\* 添加索引字段

\*/

@Override

public List<Document> getDoc(List<IfcProductRecordText> items) {

List<Document> docs = new ArrayList<Document>();

FieldType storedType = new FieldType();

storedType.setIndexed(true);

storedType.setStored(true);

storedType.setTokenized(true);

FieldType unTokeType = new FieldType();

unTokeType.setIndexed(true);

unTokeType.setStored(true);

unTokeType.setTokenized(false);

for (IfcProductRecordText record : items) {

Document doc = new Document();

Field oid = genStringFieldCheckNull(Key\_Oid, record.getOid(), unTokeType);

Field location = genStringFieldCheckNull(Key\_Location, record.getLocation(), storedType);

Field type = genStringFieldCheckNull(Key\_Type, record.getType(), storedType);

Field name = genStringFieldCheckNull(Key\_Name, record.getName(), storedType);

Field detail = genStringFieldCheckNull(Key\_Detail, record.getDetail(), storedType);

doc.add(oid);

doc.add(location);

doc.add(type);

doc.add(name);

doc.add(detail);

docs.add(doc);

}

return docs;

}

/\*\*

\* 将索引恢复成数据

\*/

@Override

public List<IfcProductRecordText> toBean(IndexSearcher indexSearcher, Query query, ScoreDoc[] hits) {

List<IfcProductRecordText> hitRecords = new ArrayList<IfcProductRecordText>();

try {

for (int i = 0; i < hits.length; i++) {

ScoreDoc scoreDoc = hits[i];

Document hitDoc = indexSearcher.doc(scoreDoc.doc);

String oid = hitDoc.get(Key\_Oid);

String location = hitDoc.get(Key\_Location);

String type = hitDoc.get(Key\_Type);

String name = hitDoc.get(Key\_Name);

String detail = hitDoc.get(Key\_Detail);

IfcProductRecordText record = new IfcProductRecordText();

record.setOid(oid);

record.setLocation(location);

record.setType(type);

record.setName(name);

record.setDetail(detail);

hitRecords.add(record);

}

} catch (IOException e) {

e.printStackTrace();

}

return hitRecords;

}

private Field genStringFieldCheckNull(String fieldName, String fieldValue, FieldType type) {

if (fieldValue != null) {

return new Field(fieldName, fieldValue, type);

} else {

return new Field(fieldName, "", type);

}

}

}

package cn.dlb.bim.ifc.stream;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.LinkedHashMap;

import java.util.List;

import java.util.Map;

import java.util.Set;

import org.eclipse.emf.ecore.EAttribute;

import org.eclipse.emf.ecore.EClass;

import org.eclipse.emf.ecore.EStructuralFeature;

import org.springframework.data.annotation.Transient;

import org.springframework.data.mongodb.core.index.Indexed;

import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "VirtualObject")

public class VirtualObject implements MinimalVirtualObject {

@Indexed

private Long oid;

@Indexed

private Integer rid;

@Indexed

private final Short eClassId;

private final Map<String, Object> features;

/\*\*

\* 用于标注是否序列化输出，但不影响数据库存储

\*/

@Transient

private final Map<EStructuralFeature, Object> useForSerializationFeatures;

@Transient

private EClass eClass;

public VirtualObject(Integer rid, Short eClassId, Long oid, EClass eClass) {

this.rid = rid;

this.eClassId = eClassId;

this.oid = oid;

features = new LinkedHashMap<>();

useForSerializationFeatures = new LinkedHashMap<>();

this.eClass = eClass;

}

public Long getOid() {

return oid;

}

public void setOid(Long oid) {

this.oid = oid;

}

public Integer getRid() {

return rid;

}

public void setRid(Integer rid) {

this.rid = rid;

}

public Short getEClassId() {

return eClassId;

}

public Object eGet(EStructuralFeature feature) {

return features.get(feature.getName());

}

public void eUnset(EStructuralFeature eStructuralFeature) {

features.remove(eStructuralFeature.getName());

}

public boolean eIsSet(EStructuralFeature feature) {

return features.containsKey(feature.getName());

}

public void setReference(EStructuralFeature eStructuralFeature, Long oid) {

features.put(eStructuralFeature.getName(), oid);

}

public void setAttribute(EStructuralFeature eStructuralFeature, Object value) {

if (value != null) {

features.put(eStructuralFeature.getName(), value);

}

}

@SuppressWarnings({ "rawtypes", "unchecked" })

public void setListItem(EStructuralFeature structuralFeature, int index, Object value) {

List list = getOrCreateList(structuralFeature, index + 1);

list.set(index, value);

}

@SuppressWarnings({ "rawtypes", "unchecked" })

public void setListItemReference(EStructuralFeature structuralFeature, int index, Long referencedOid) {

List list = getOrCreateList(structuralFeature, index + 1);

list.set(index, referencedOid);

}

@SuppressWarnings({ "rawtypes", "unchecked" })

private List getOrCreateList(EStructuralFeature structuralFeature, int minSize) {

List list = (List<?>) features.get(structuralFeature.getName());

if (list == null) {

list = new ArrayList(minSize == -1 ? 0 : minSize);

features.put(structuralFeature.getName(), list);

}

while (list.size() < minSize) {

list.add(null);

}

return list;

}

public Map<String, Object> getFeatures() {

return features;

}

public Boolean has(String key) {

return features.containsKey(key);

}

public Object get(String key) {

return features.get(key);

}

@SuppressWarnings("unchecked")

public boolean useFeatureForSerialization(EStructuralFeature feature, int index) {

if (feature instanceof EAttribute) {

return true;

}

if (useForSerializationFeatures.containsKey(feature)) {

Object object = useForSerializationFeatures.get(feature);

if (object instanceof Set) {

Set<Integer> set = (Set<Integer>) object;

if (set.contains(index)) {

return true;

}

} else {

return object == Boolean.TRUE;

}

}

return false;

}

public boolean useFeatureForSerialization(EStructuralFeature feature) {

if (feature instanceof EAttribute) {

return true;

}

return useForSerializationFeatures.containsKey(feature);

}

public void addUseForSerialization(EStructuralFeature eStructuralFeature) {

if (eStructuralFeature.getEContainingClass().isSuperTypeOf(eClass)) {

useForSerializationFeatures.put(eStructuralFeature, true);

} else {

throw new IllegalArgumentException(eStructuralFeature.getName() + " does not exist in " + eClass.getName());

}

}

@SuppressWarnings("unchecked")

public void addUseForSerialization(EStructuralFeature eStructuralFeature, int index) {

if (eStructuralFeature.getEContainingClass().isSuperTypeOf(eClass)) {

Set<Integer> set = (Set<Integer>) useForSerializationFeatures.get(eStructuralFeature);

if (set == null) {

set = new HashSet<>();

useForSerializationFeatures.put(eStructuralFeature, set);

}

set.add(index);

} else {

throw new IllegalArgumentException(eStructuralFeature.getName() + " does not exist in " + eClass.getName());

}

}

public EClass eClass() {

return eClass;

}

}

package cn.dlb.bim.cache;

import java.util.LinkedHashMap;

import java.util.Map;

import java.util.Set;

/\*\*

\*

\* @author shenan4321

\*

\* @param <K>

\* @param <V>

\*/

public class LRUCache<K, V> {

private final int MAX\_CACHE\_SIZE;

private final float DEFAULT\_LOAD\_FACTOR = 0.75f;

private Map<K, V> map;

/\*\*

\* 缓存构造

\* @param cacheSize

\*/

@SuppressWarnings({ "unchecked", "serial", "rawtypes" })

public LRUCache(int cacheSize) {

MAX\_CACHE\_SIZE = cacheSize;

//根据cacheSize和加载因子计算hashmap的capactiy，+1确保当达到cacheSize上限时不会触发hashmap的扩容，

int capacity = (int) Math.ceil(MAX\_CACHE\_SIZE / DEFAULT\_LOAD\_FACTOR) + 1;

map = new LinkedHashMap(capacity, DEFAULT\_LOAD\_FACTOR, true) {

@Override

protected boolean removeEldestEntry(Map.Entry eldest) {

return size() > MAX\_CACHE\_SIZE;

}

};

}

public synchronized boolean contains(K key) {

return map.containsKey(key);

}

public synchronized void put(K key, V value) {

map.put(key, value);

}

public synchronized V get(K key) {

return map.get(key);

}

public synchronized void remove(K key) {

map.remove(key);

}

public synchronized Set<Map.Entry<K, V>> getAll() {

return map.entrySet();

}

public synchronized int size() {

return map.size();

}

public synchronized void clear() {

map.clear();

}

@SuppressWarnings("rawtypes")

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

for (Map.Entry entry : map.entrySet()) {

sb.append(String.format("%s:%s ", entry.getKey(), entry.getValue()));

}

return sb.toString();

}

}

package cn.dlb.bim.ifc.collada;

import java.io.OutputStream;

import java.io.PrintWriter;

import java.nio.ByteBuffer;

import java.nio.ByteOrder;

import java.nio.DoubleBuffer;

import java.text.DateFormat;

import java.text.DecimalFormat;

import java.text.Format;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.HashMap;

import java.util.HashSet;

import java.util.LinkedHashMap;

import java.util.List;

import java.util.Map;

import java.util.Set;

import org.apache.commons.lang.StringUtils;

import org.eclipse.emf.common.util.Enumerator;

import org.eclipse.emf.ecore.EClass;

import org.eclipse.emf.ecore.EStructuralFeature;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import cn.dlb.bim.ifc.emf.IdEObject;

import cn.dlb.bim.ifc.emf.IfcModelInterface;

import cn.dlb.bim.ifc.emf.PackageMetaData;

import cn.dlb.bim.ifc.emf.ProjectInfo;

import cn.dlb.bim.ifc.engine.RenderEngineException;

import cn.dlb.bim.ifc.engine.cells.Matrix;

import cn.dlb.bim.ifc.engine.cells.Vector3d;

import cn.dlb.bim.ifc.serializers.SerializerException;

import cn.dlb.bim.ifc.shared.ProgressReporter;

import cn.dlb.bim.models.geometry.GeometryData;

import cn.dlb.bim.models.geometry.GeometryInfo;

import cn.dlb.bim.models.store.SIPrefix;

import cn.dlb.bim.utils.UTF8PrintWriter;

/\*\*

\* @author shenan4321

\*/

public class ColladaSerializer extends AbstractGeometrySerializer {

private static final Logger LOGGER = LoggerFactory.getLogger(ColladaSerializer.class);

private static final Map<String, MaterialConvertor> convertors = new LinkedHashMap<String, MaterialConvertor>();

private final Map<String, Set<IdEObject>> converted = new HashMap<String, Set<IdEObject>>();

private SIPrefix lengthUnitPrefix;

/\*\*

\* 添加颜色转换

\* @param convertor

\*/

private static void addConvertor(MaterialConvertor convertor) {

convertors.put(convertor.getIfcType(), convertor);

}

static {

addConvertor(new MaterialConvertor("IfcRoof", new double[] { 70/255.0f, 30/255.0f, 30/255.0f }, 1.0f));

addConvertor(new MaterialConvertor("IfcSlab", new double[] { 160.0/255.0f, 150/255.0f, 104.0/255.0f }, 1.0f));

addConvertor(new MaterialConvertor("IfcWindow", new double[] { 0.2f, 0.2f, 0.8f }, 0.3f));

addConvertor(new MaterialConvertor("IfcSpace", new double[] { 0.5f, 0.4f, 0.1f }, 0.0f));

addConvertor(new MaterialConvertor("IfcDoor", new double[] { 0.637255, 0.603922, 0.670588 }, 1.0f));

addConvertor(new MaterialConvertor("IfcStair", new double[] { 0.0, 0.0f, 0.0f }, 1.0f));

addConvertor(new MaterialConvertor("IfcStairFlight", new double[] { 0.637255f, 0.603922f, 0.670588f }, 1.0f));

addConvertor(new MaterialConvertor("IfcFlowSegment", new double[] { 0.6f, 0.4f, 0.5f }, 1.0f));

addConvertor(new MaterialConvertor("IfcFurnishingElement", new double[] { 205.0/255, 104.0/255, 57.0/255f }, 1.0f));

addConvertor(new MaterialConvertor("IfcPlate", new double[] { 0.0f, 0.0f, 0.0f }, 0.4f));

addConvertor(new MaterialConvertor("IfcMember", new double[] { 0.137255, 0.203922, 0.270588 }, 1.0f));

addConvertor(new MaterialConvertor("IfcWallStandardCase", new double[] { 150/255.0, 150/255.0, 104.0/255.0f }, 1.0f));

addConvertor(new MaterialConvertor("IfcWall", new double[] { 146.0/255.0, 72.0/255.0, 38.0/255.0f }, 1.0f));

addConvertor(new MaterialConvertor("IfcCurtainWall", new double[] { 0.5f, 0.5f, 0.5f }, 0.5f));

addConvertor(new MaterialConvertor("IfcRailing", new double[] { 0.137255, 0.203922, 0.270588f }, 1.0f));

addConvertor(new MaterialConvertor("IfcColumn", new double[] { 0.0f, 0.0f, 0.0f }, 1.0f));

addConvertor(new MaterialConvertor("IfcBuildingElementProxy", new double[] { 0.0f, 0.8f, 0.5f }, 1.0f));

addConvertor(new MaterialConvertor("IfcBeam", new double[] { 0.137255, 0.403922, 0.870588f }, 1.0f));

// addConvertor(new Convertor<IfcBeamStandardCase>(IfcBeamStandardCase.class, new double[] { 0.137255f, 0.403922f, 0.870588f }, 1.0f));

addConvertor(new MaterialConvertor("IfcFlowTerminal", new double[] { 0.137255, 0.403922, 0.870588f }, 1.0f));

addConvertor(new MaterialConvertor("IfcProxy", new double[] { 0.0, 0.8, 0.5f }, 0.9f));

addConvertor(new MaterialConvertor("IfcSite", new double[] { 0.0, 102.0f/255.0f, 180.0f/255.0f }, 1.0f));

// addConvertor(new Convertor<IfcLightFixture>(IfcLightFixture.class, new double[] { 0.137255f, 0.403922f, 0.870588f }, 1.0f));

// addConvertor(new Convertor<IfcDuctSegment>(IfcDuctSegment.class, new double[] { 0.137255f, 0.403922f, 0.870588f }, 1.0f));

addConvertor(new MaterialConvertor("IfcDistributionFlowElement", new double[] { 0.8470588235f, 0.427450980392f, 0.0f }, 1.0f));

// addConvertor(new Convertor<IfcDuctFitting>(IfcDuctFitting.class, new double[] { 0.137255f, 0.403922f, 0.870588f }, 1.0f));

addConvertor(new MaterialConvertor("IfcPile", new double[] { 0.8470588235f, 0.427450980392f, 0.0f }, 1.0f));

// addConvertor(new Convertor<IfcAirTerminal>(IfcAirTerminal.class, new double[] { 0.8470588235f, 0.427450980392f, 0.0f }, 1.0f));

addConvertor(new MaterialConvertor("IfcCovering", new double[] { 0.8470588235f, 0.427450980392f, 0.0f }, 1.0f));

addConvertor(new MaterialConvertor("IfcTransportElement", new double[] { 0.8470588235f, 0.427450980392f, 0.0f }, 1.0f));

addConvertor(new MaterialConvertor("IfcFlowController", new double[] { 0.8470588235f, 0.427450980392f, 0.0f }, 1.0f));

addConvertor(new MaterialConvertor("IfcFlowFitting", new double[] { 0.8470588235f, 0.427450980392f, 0.0f }, 1.0f));

addConvertor(new MaterialConvertor("IfcRamp", new double[] { 0.8470588235f, 0.427450980392f, 0.0f }, 1.0f));

// addConvertor(new Convertor<IfcFurniture>(IfcFurniture.class, new double[] { 0.8470588235f, 0.427450980392f, 0.0f }, 1.0f));

addConvertor(new MaterialConvertor("IfcFooting", new double[] { 0.8470588235f, 0.427450980392f, 0.0f }, 1.0f));

// addConvertor(new Convertor<IfcSystemFurnitureElement>(IfcSystemFurnitureElement.class, new double[] { 0.8470588235f, 0.427450980392f, 0.0f }, 1.0f));

addConvertor(new MaterialConvertor("IfcBuildingElementPart", new double[] { 1.0f, 0.5f, 0.5f }, 1.0f));

addConvertor(new MaterialConvertor("IfcDistributionElement", new double[] { 1.0f, 0.5f, 0.5f }, 1.0f));

addConvertor(new MaterialConvertor("IfcProduct", new double[] { 0.8470588235, 0.427450980392, 0.0f }, 1.0f));

}

// Prepare a transformer for floating-point numbers into a strings, clipping extraneous zeros.

private static final DecimalFormat decimalFormat = new DecimalFormat("#.##########");

// Prepare a transformer for integers into strings.

private static final DecimalFormat intFormat = new DecimalFormat("#");

private Vector3d lowestObserved = new Vector3d();

private Vector3d highestObserved = new Vector3d();

@Override

public void init(IfcModelInterface model, ProjectInfo projectInfo, boolean normalizeOids) throws SerializerException {

super.init(model, projectInfo, normalizeOids);

this.lengthUnitPrefix = getLengthUnitPrefix(model);

}

/\*\*

\* 写dae文件流

\*/

@Override

public boolean write(OutputStream out, ProgressReporter progressReporter) throws SerializerException {

if (getMode() == Mode.BODY) {

PrintWriter writer = new UTF8PrintWriter(out);

try {

writer.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");

writer.println("<COLLADA xmlns=\"http://www.collada.org/2008/03/COLLADASchema\" version=\"1.5.0\">");

// Data sections.

writeAssets(writer);

writeCameras(writer);

writeLights(writer);

writeEffects(writer);

writeMaterials(writer);

writeGeometries(writer);

writeVisualScenes(writer);

writeScene(writer);

// End of root section.

writer.print("</COLLADA>");

writer.flush();

} catch (Exception e) {

LOGGER.error("", e);

}

writer.flush();

setMode(Mode.FINISHED);

return true;

} else if (getMode() == Mode.FINISHED) {

return false;

} else if (getMode() == Mode.HEADER) {

setMode(Mode.BODY);

return true;

}

return false;

}

// Provide a date formatter.

private static final DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd'T'hh:mm:ss'Z'");

private void writeAssets(PrintWriter out) {

// Produce a date based on now.

String date = dateFormat.format(new Date());

// Determine what a meter is and what to logically refer to that transformation as.

Number unitMeter = (lengthUnitPrefix == null) ? 1 : Math.pow(10.0, lengthUnitPrefix.getValue());

String unitName = (lengthUnitPrefix == null) ? "meter" : lengthUnitPrefix.name().toLowerCase();

// Write the asset block.

out.println(" <asset>");

out.println(" <contributor>");

out.println(" <author>" + (getProjectInfo() == null ? "" : "linfujun") + "</author>");

out.println(" <authoring\_tool>BIMplatform</authoring\_tool>");

out.println(" <comments>" + (getProjectInfo() == null ? "" : getProjectInfo().getDescription()) + "</comments>");

out.println(" <copyright>Copyright</copyright>");

out.println(" </contributor>");

out.println(" <created>" + date + "</created>");

out.println(" <modified>" + date + "</modified>");

out.println(" <unit meter=\"" + unitMeter + "\" name=\"" + unitName + "\"/>");

out.println(" <up\_axis>Z\_UP</up\_axis>");

out.println(" </asset>");

}

/\*\*

\* 写几何数据

\* @param out

\* @throws RenderEngineException

\* @throws SerializerException

\*/

private void writeGeometries(PrintWriter out) throws RenderEngineException, SerializerException {

// Prepare the section.

out.println(" <library\_geometries>");

// For each IfcProduct, get the geometry for each object in the product.

Set<IdEObject> convertedObjects = new HashSet<IdEObject>();

for (MaterialConvertor materialConvertor : convertors.values()) {

String ifcType = materialConvertor.getIfcType();

for (IdEObject ifcProduct : model.getAllWithSubTypes(model.getPackageMetaData().getEClass(ifcType))) {

if (!convertedObjects.contains(ifcProduct)) {

convertedObjects.add(ifcProduct);

setGeometry(out, ifcProduct, materialConvertor.getIfcType());

}

}

}

// Close the section.

out.println(" </library\_geometries>");

}

private void setGeometry(PrintWriter out, IdEObject ifcProductObject, String material) throws RenderEngineException, SerializerException {

// Mostly just skips IfcOpeningElements which one would probably not want to end up in the Collada file.

if (isInstanceOf(ifcProductObject, "IfcFeatureElementSubtraction") || isInstanceOf(ifcProductObject, "IfcSpace")) {

return;

}

//

// if (!(isInstanceOf(ifcProductObject, "IfcWall") || isInstanceOf(ifcProductObject, "IfcWallStandardCase")

// || isInstanceOf(ifcProductObject, "IfcCurtainWall"))) {

// return; //DEMO 输出窗

// }

GeometryInfo geometryInfo = (GeometryInfo) ifcProductObject.eGet(ifcProductObject.eClass().getEStructuralFeature("geometry"));

if (geometryInfo != null && geometryInfo.getTransformation() != null) {

GeometryData geometryData = geometryInfo.getData();

ByteBuffer indicesBuffer = ByteBuffer.wrap(geometryData.getIndices());

indicesBuffer.order(ByteOrder.LITTLE\_ENDIAN);

// TODO: In Blender (3d modeling tool) and Three.js, normals are ignored in favor of vertex order. The incoming geometry seems to be in order 0 1 2 when it needs to be in 1 0 2. Need more test cases.

// Failing order: (0, 1050, 2800), (0, 1050, 3100), (3580, 1050, 3100)

// Successful order: (0, 1050, 3100), (0, 1050, 2800), (3580, 1050, 3100)

List<Integer> list = new ArrayList<Integer>();

while (indicesBuffer.hasRemaining())

list.add(indicesBuffer.getInt());

indicesBuffer.rewind();

for (int i = 0; i < list.size(); i += 3)

{

Integer first = list.get(i);

Integer next = list.get(i + 1);

list.set(i, next);

list.set(i + 1, first);

}

// Positions the X or the Y or the Z of (X, Y, Z).

ByteBuffer positionsBuffer = ByteBuffer.wrap(geometryData.getVertices());

positionsBuffer.order(ByteOrder.LITTLE\_ENDIAN);

// Do pass to find highest Z for considered objects.

while (positionsBuffer.hasRemaining())

{

float x = positionsBuffer.getFloat();

float y = positionsBuffer.getFloat();

float z = positionsBuffer.getFloat();

// X

if (x > highestObserved.x)

highestObserved.x = x;

else if (x < lowestObserved.x)

lowestObserved.x = x;

// Y

if (y > highestObserved.y)

highestObserved.y = y;

else if (y < lowestObserved.y)

lowestObserved.y = y;

// Z

if (z > highestObserved.z)

highestObserved.z = z;

else if (z < lowestObserved.z)

lowestObserved.z = z;

}

positionsBuffer.rewind();

//

ByteBuffer normalsBuffer = ByteBuffer.wrap(geometryData.getNormals());

normalsBuffer.order(ByteOrder.LITTLE\_ENDIAN);

// Create a geometry identification number in the form of: geom-320450

long oid = ifcProductObject.getOid();

String id = String.format("geom-%d", oid);

// If the material doesn't exist in the converted map, add it.

if (!converted.containsKey(material))

converted.put(material, new HashSet<IdEObject>());

// Add the current IfcProduct to the appropriate entry in the material map.

converted.get(material).add(ifcProductObject);

// Name for geometry.

Object globalIdObj = ifcProductObject.eGet(ifcProductObject.eClass().getEStructuralFeature("GlobalId"));

String name = (globalIdObj == null) ? "[NO\_GUID]" : (String) globalIdObj;

// Counts.

int vertexComponentsTotal = positionsBuffer.capacity() / 4, normalComponentsTotal = normalsBuffer.capacity() / 4;

int verticesCount = positionsBuffer.capacity() / 12, normalsCount = normalsBuffer.capacity() / 12, triangleCount = indicesBuffer.capacity() / 12;

// Vertex scalars as one long string: 4.05 2 1 55.0 34.01 2

String stringPositionScalars = byteBufferToFloatingPointSpaceDelimitedString(positionsBuffer);

// Normal scalars as one long string: 4.05 2 1 55.0 34.01 2

String stringNormalScalars = byteBufferToFloatingPointSpaceDelimitedString(normalsBuffer); //doubleBufferToFloatingPointSpaceDelimitedString(flippedNormalsBuffer);

// Vertex indices as one long string: 1 0 2 0 3 2 5 4 6

String stringIndexScalars = listToSpaceDelimitedString(list, intFormat);

// Write geometry block for this IfcProduct (i.e. IfcRoof, IfcSlab, etc).

out.println(" <geometry id=\"" + id + "\" name=\"" + name + "\">");

out.println(" <mesh>");

out.println(" <source id=\"positions-" + oid + "\" name=\"positions-" + oid + "\">");

out.println(" <float\_array id=\"positions-array-" + oid + "\" count=\"" + vertexComponentsTotal + "\">" + stringPositionScalars + "</float\_array>");

out.println(" <technique\_common>");

out.println(" <accessor count=\"" + verticesCount + "\" offset=\"0\" source=\"#positions-array-" + oid + "\" stride=\"3\">");

out.println(" <param name=\"X\" type=\"float\"></param>");

out.println(" <param name=\"Y\" type=\"float\"></param>");

out.println(" <param name=\"Z\" type=\"float\"></param>");

out.println(" </accessor>");

out.println(" </technique\_common>");

out.println(" </source>");

out.println(" <source id=\"normals-" + oid + "\" name=\"normals-" + oid + "\">");

out.println(" <float\_array id=\"normals-array-" + oid + "\" count=\"" + normalComponentsTotal + "\">" + stringNormalScalars + "</float\_array>");

out.println(" <technique\_common>");

out.println(" <accessor count=\"" + normalsCount + "\" offset=\"0\" source=\"#normals-array-" + oid + "\" stride=\"3\">");

out.println(" <param name=\"X\" type=\"float\"></param>");

out.println(" <param name=\"Y\" type=\"float\"></param>");

out.println(" <param name=\"Z\" type=\"float\"></param>");

out.println(" </accessor>");

out.println(" </technique\_common>");

out.println(" </source>");

out.println(" <vertices id=\"vertices-" + oid + "\">");

out.println(" <input semantic=\"POSITION\" source=\"#positions-" + oid + "\"/>");

out.println(" <input semantic=\"NORMAL\" source=\"#normals-" + oid + "\"/>");

out.println(" </vertices>");

out.println(" <triangles count=\"" + triangleCount + "\" material=\"Material-" + oid + "\">");

out.println(" <input offset=\"0\" semantic=\"VERTEX\" source=\"#vertices-" + oid + "\"/>");

out.println(" <p>" + stringIndexScalars + "</p>");

out.println(" </triangles>");

out.println(" </mesh>");

out.println(" </geometry>");

}

}

@SuppressWarnings("unused")

private String doubleBufferToFloatingPointSpaceDelimitedString(DoubleBuffer buffer) {

// For each scalar in the buffer, turn it into a string, adding it to the overall list.

List<String> stringScalars = doubleBufferToStringList(buffer, decimalFormat);

// Send back a space-delimited list of the strings: 1 2.45 0

return StringUtils.join(stringScalars, " ");

}

private String byteBufferToFloatingPointSpaceDelimitedString(ByteBuffer buffer) {

// For each scalar in the buffer, turn it into a string, adding it to the overall list.

List<String> stringScalars = floatBufferToStringList(buffer, decimalFormat);

// Send back a space-delimited list of the strings: 1 2.45 0

return StringUtils.join(stringScalars, " ");

}

@SuppressWarnings("unused")

private String byteBufferToIntPointSpaceDelimitedString(ByteBuffer buffer) {

// Prepare to store integers as a list of strings.

List<String> stringScalars = intBufferToStringList(buffer, intFormat);

// Send back a space-delimited list of the strings: 1 2 0

return StringUtils.join(stringScalars, " ");

}

private List<String> doubleBufferToStringList(DoubleBuffer buffer, Format formatter) {

// Transform the array into a list.

List<Float> list = new ArrayList<Float>();

while (buffer.hasRemaining())

list.add(new Float(buffer.get()));

// Get the data as a list of String objects.

return listToStringList(list, formatter);

}

private List<String> floatBufferToStringList(ByteBuffer buffer, Format formatter) {

// Transform the array into a list.

List<Float> list = new ArrayList<Float>();

while (buffer.hasRemaining())

list.add(new Float(buffer.getFloat()));

// Get the data as a list of String objects.

return listToStringList(list, formatter);

}

private List<String> intBufferToStringList(ByteBuffer buffer, Format formatter) {

List<Integer> list = new ArrayList<Integer>();

while (buffer.hasRemaining())

list.add(new Integer(buffer.getInt()));

// Get the data as a list of String objects.

return listToStringList(list, formatter);

}

private String floatArrayToSpaceDelimitedString(float[] matrix) {

List<Float> floatMatrix = floatArrayToFloatList(matrix);

List<String> list = listToStringList(floatMatrix, decimalFormat);

// Get data as space-delimited string: 1.004 5.0 24.00145

return StringUtils.join(list, " ");

}

private String doubleArrayToSpaceDelimitedString(double[] matrix) {

List<Double> floatMatrix = doubleArrayToFloatList(matrix);

List<String> list = listToStringList(floatMatrix, decimalFormat);

// Get data as space-delimited string: 1.004 5.0 24.00145

return StringUtils.join(list, " ");

}

private List<Float> floatArrayToFloatList(float[] array) {

List<Float> list = new ArrayList<Float>();

for (float f : array)

list.add(new Float(f));

return list;

}

private List<Double> doubleArrayToFloatList(double[] array) {

List<Double> list = new ArrayList<Double>();

for (double f : array)

list.add(new Double(f));

return list;

}

private String listToSpaceDelimitedString(List<?> list, Format formatter) {

List<String> stringScalars = listToStringList(list, formatter);

return StringUtils.join(stringScalars, " ");

}

private List<String> listToStringList(List<?> list, Format formatter) {

// Prepare to store floating-points as a list of strings.

List<String> stringScalars = new ArrayList<String>();

// For each scalar in the buffer, turn it into a string, adding it to the overall list.

for (Object scalar : list) {

String scalarAsString = formatter.format(scalar);

stringScalars.add(scalarAsString);

}

return stringScalars;

}

private void writeScene(PrintWriter out) {

out.println(" <scene>");

out.println(" <instance\_visual\_scene url=\"#VisualSceneNode\"/>");

out.println(" </scene>");

}

//String leftLightLocationString = String.format("%f %f %f", leftLightLocation.x(), leftLightLocation.y(), leftLightLocation.z());

@SuppressWarnings("unused")

private void writeVisualScenes(PrintWriter out) {

// Open the section.

out.println(" <library\_visual\_scenes>");

out.println(" <visual\_scene id=\"VisualSceneNode\" name=\"VisualSceneNode\">");

// Write each IFC object as a node entry (maps to a displayed object).

for (String material : converted.keySet()) {

Set<IdEObject> ids = converted.get(material);

for (IdEObject product : ids) {

GeometryInfo geometryInfo = (GeometryInfo) product.eGet(product.eClass().getEStructuralFeature("geometry"));

if (geometryInfo != null && geometryInfo.getTransformation() != null) {

out.println(" <node id=\"node-" + product.getOid() + "\" name=\"node-" + product.getOid() + "\">");

printMatrix(out, geometryInfo);

out.println(" <instance\_geometry url=\"#geom-" + product.getOid() + "\">");

out.println(" <bind\_material>");

out.println(" <technique\_common>");

out.println(" <instance\_material symbol=\"Material-" + product.getOid() + "\" target=\"#" + material + "Material\"/>");

out.println(" </technique\_common>");

out.println(" </bind\_material>");

out.println(" </instance\_geometry>");

out.println(" </node>");

}

}

}

// Create convenience variables to simplify the perceived complexity of the equations.

float lx = (float) lowestObserved.x, ly = (float) lowestObserved.y, lz = (float) lowestObserved.z;

float hx = (float) highestObserved.x, hy = (float) highestObserved.y, hz = (float) highestObserved.z;

// Derive useful information from the observed boundary of the IFC objects.

Vector3d delta = new Vector3d(hx, hy, hz);

delta.sub(lowestObserved);

// Move the light left (-x) and back (-y) and up (+z) at 20% of the size of the observed objects.

Vector3d leftLightLocation = new Vector3d(lx - (0.2 \* delta.x), ly - (0.2 \* delta.y), hz + (0.2 \* delta.z));

float x = (float) leftLightLocation.x, y = (float) leftLightLocation.y, z = (float) leftLightLocation.z;

// Move left (-x) and back (-y) at 500% and up (+z) at 200% of the light (so that the objects are in sensing range of the camera).

Vector3d leftCameraLocation = new Vector3d(5 \* x, 5 \* y, 2 \* z);

float cx = (float) leftCameraLocation.x, cy = (float) leftCameraLocation.y, cz = (float) leftCameraLocation.z;

// TODO: Three.js doesn't seem to care about the camera and the light.

// Include the camera.

out.println(" <node id=\"Camera\" name=\"Camera\">");

out.println(" <translate>"+ cx + " " + cy + " " + cz + "</translate>");

out.println(" <rotate>"+ 0f + " " + 0f + " " + 1f + " " + -45f + "</rotate>");

out.println(" <rotate>"+ 1f + " " + 0f + " " + 0f + " " + 45f + "</rotate>");

out.println(" <instance\_camera url=\"#PerspCamera\"/>");

out.println(" </node>");

// Include the light.

out.println(" <node id=\"Light\" name=\"Light\">");

out.println(" <translate>"+ x + " " + y + " " + z + "</translate>");

out.println(" <rotate>"+ 0f + " " + 0f + " " + 1f + " " + 225f + "</rotate>");

out.println(" <rotate>"+ 0f + " " + 1f + " " + 0f + " " + 45f + "</rotate>");

out.println(" <instance\_light url=\"#light-lib\"/>");

out.println(" </node>");

// Close the section.

out.println(" </visual\_scene>");

out.println(" </library\_visual\_scenes>");

}

private void printMatrix(PrintWriter out, GeometryInfo geometryInfo) {

ByteBuffer transformation = ByteBuffer.wrap(geometryInfo.getTransformation());

transformation.order(ByteOrder.LITTLE\_ENDIAN);

DoubleBuffer doubleBuffer = transformation.asDoubleBuffer();

// Prepare to create the transform matrix.

double[] matrix = new double[16];

// Add the first 16 values of the buffer.

for (int i = 0; i < matrix.length; i++)

matrix[i] = doubleBuffer.get();

// Switch from column-major (x.x ... x.y ... x.z ... 0 ...) to row-major orientation (x.x x.y x.z 0 ...)?

matrix = Matrix.changeOrientation(matrix);

// List all 16 elements of the matrix as a single space-delimited String object.

out.println(" <matrix>" + doubleArrayToSpaceDelimitedString(matrix) + "</matrix>");

}

private void writeEffects(PrintWriter out) {

out.println(" <library\_effects>");

for (MaterialConvertor convertor : convertors.values())

writeEffect(out, convertor.getIfcType(), convertor.getColors(), convertor.getOpacity());

out.println(" </library\_effects>");

}

private void writeEffect(PrintWriter out, String name, double[] colors, double transparency) {

out.println(" <effect id=\"" + name + "-fx\">");

out.println(" <profile\_COMMON>");

out.println(" <technique sid=\"common\">");

out.println(" <phong>");

out.println(" <emission>");

out.println(" <color>0 0 0 1</color>");

out.println(" </emission>");

out.println(" <ambient>");

out.println(" <color>0 0 0 1</color>");

out.println(" </ambient>");

out.println(" <diffuse>");

out.println(" <color>" + colors[0] + " " + colors[1] + " " + colors[2] + " " + transparency + "</color>");

out.println(" </diffuse>");

out.println(" <specular>");

out.println(" <color>0.5 0.5 0.5 1</color>");

out.println(" </specular>");

if (transparency < 1.0) {

out.println(" <transparent opaque=\"RGB\_ZERO\">");

out.println(" <color>0.4980392 0.4980392 0.4980392 0.4980392</color>");

out.println(" </transparent>");

out.println(" <transparency>");

out.println(" <float>1</float>");

out.println(" </transparency>");

out.println(" <shininess>");

out.println(" <float>64</float>");//max 128

out.println(" </shininess>");

} else {

out.println(" <shininess>");

out.println(" <float>16</float>");

out.println(" </shininess>");

}

out.println(" <reflective>");

out.println(" <color>0 0 0 1</color>");

out.println(" </reflective>");

out.println(" <reflectivity>");

out.println(" <float>0.5</float>");

out.println(" </reflectivity>");

out.println(" <index\_of\_refraction>");

out.println(" <float>0</float>");

out.println(" </index\_of\_refraction>");

out.println(" </phong>");

out.println(" </technique>");

out.println(" </profile\_COMMON>");

out.println(" </effect>");

}

private void writeLights(PrintWriter out) {

// TODO: Lights defined in library\_lights of COLLADA file must be used in some visual\_scene to prevent crashes with collada2gltf (stand-alone OpenGL Transmission Format exporter).

out.println(" <library\_lights>");

out.println(" <light id=\"light-lib\" name=\"light\">");

out.println(" <technique\_common>");

out.println(" <directional>");

out.println(" <color>1 1 1</color>");

out.println(" </directional>");

out.println(" </technique\_common>");

out.println(" <technique profile=\"MAX3D\">");

out.println(" <intensity>1.000000</intensity>");

out.println(" </technique>");

out.println(" </light>");

out.println(" </library\_lights>");

}

private void writeCameras(PrintWriter out) {

out.println(" <library\_cameras>");

out.println(" <camera id=\"PerspCamera\" name=\"PerspCamera\">");

out.println(" <optics>");

out.println(" <technique\_common>");

out.println(" <perspective>");

out.println(" <yfov>37.8493</yfov>");

out.println(" <aspect\_ratio>1</aspect\_ratio>");

out.println(" <znear>10</znear>");

out.println(" <zfar>1000</zfar>");

out.println(" </perspective>");

out.println(" </technique\_common>");

out.println(" </optics>");

out.println(" </camera>");

out.println(" <camera id=\"testCameraShape\" name=\"testCameraShape\">");

out.println(" <optics>");

out.println(" <technique\_common>");

out.println(" <perspective>");

out.println(" <yfov>37.8501</yfov>");

out.println(" <aspect\_ratio>1</aspect\_ratio>");

out.println(" <znear>0.01</znear>");

out.println(" <zfar>1000</zfar>");

out.println(" </perspective>");

out.println(" </technique\_common>");

out.println(" </optics>");

out.println(" </camera>");

out.println(" </library\_cameras>");

}

private void writeMaterials(PrintWriter out) {

out.println(" <library\_materials>");

for (MaterialConvertor convertor : convertors.values())

writeMaterial(out, convertor.getIfcType());

out.println(" </library\_materials>");

}

private void writeMaterial(PrintWriter out, String materialName) {

out.println(" <material id=\"" + materialName + "Material\" name=\"" + materialName + "Material\">");

out.println(" <instance\_effect url=\"#" + materialName + "-fx\"/>");

out.println(" </material>");

}

private SIPrefix getLengthUnitPrefix(IfcModelInterface model) {

SIPrefix lengthUnitPrefix = null;

boolean prefixFound = false;

Map<Long, IdEObject> objects = model.getObjects();

for (IdEObject object : objects.values()) {

if (isInstanceOf(object, "IfcProject")) {

IdEObject unitsInContext = (IdEObject) object.eGet(object.eClass().getEStructuralFeature("UnitsInContext"));

// IfcUnitAssignment unitsInContext = ((IfcProject) object).getUnitsInContext();

if (unitsInContext != null) {

List units = (List) unitsInContext.eGet(unitsInContext.eClass().getEStructuralFeature("Units"));

// EList<IfcUnit> units = unitsInContext.getUnits();

for (Object unit : units) {

IdEObject ifcSIUnit = (IdEObject) unit;

if (isInstanceOf(ifcSIUnit, "IfcSIUnit")) {

EStructuralFeature feature = ifcSIUnit.eClass().getEStructuralFeature("UnitType");

Enumerator unitType = (Enumerator) ifcSIUnit.eGet(feature);

if (unitType.getLiteral() == "LENGTHUNIT") {

prefixFound = true;

Enumerator prefix = (Enumerator) ifcSIUnit.eGet(ifcSIUnit.eClass().getEStructuralFeature("Prefix"));

switch (prefix.getLiteral()) {

case "EXA":

lengthUnitPrefix = SIPrefix.EXAMETER;

break;

case "PETA":

lengthUnitPrefix = SIPrefix.PETAMETER;

break;

case "TERA":

lengthUnitPrefix = SIPrefix.TERAMETER;

break;

case "GIGA":

lengthUnitPrefix = SIPrefix.GIGAMETER;

break;

case "MEGA":

lengthUnitPrefix = SIPrefix.MEGAMETER;

break;

case "KILO":

lengthUnitPrefix = SIPrefix.KILOMETER;

break;

case "HECTO":

lengthUnitPrefix = SIPrefix.HECTOMETER;

break;

case "DECA":

lengthUnitPrefix = SIPrefix.DECAMETER;

break;

case "DECI":

lengthUnitPrefix = SIPrefix.DECIMETER;

break;

case "CENTI":

lengthUnitPrefix = SIPrefix.CENTIMETER;

break;

case "MILLI":

lengthUnitPrefix = SIPrefix.MILLIMETER;

break;

case "MICRO":

lengthUnitPrefix = SIPrefix.MICROMETER;

break;

case "NANO":

lengthUnitPrefix = SIPrefix.NANOMETER;

break;

case "PICO":

lengthUnitPrefix = SIPrefix.PICOMETER;

break;

case "FEMTO":

lengthUnitPrefix = SIPrefix.FEMTOMETER;

break;

case "ATTO":

lengthUnitPrefix = SIPrefix.ATTOMETER;

break;

case "NULL":

lengthUnitPrefix = SIPrefix.METER;

break;

}

break;

}

}

}

}

}

if (prefixFound)

break;

}

return lengthUnitPrefix;

}

private Boolean isInstanceOf(IdEObject originObject, String type) {

PackageMetaData packageMetaData = model.getPackageMetaData();

EClass eClass = packageMetaData.getEClass(type);

return eClass.isSuperTypeOf(originObject.eClass());

}

}

<!DOCTYPE html>

<html lang=*"en"*>

<head>

<meta charset=*"UTF-8"* />

<meta name=*"viewport"* content=*"width=device-width, initial-scale=1.0, user-scalable=0, minimum-scale=1.0, maximum-scale=1.0"* />

<meta name=*"screen-orientation"* content=*"portrait"* />

<meta name=*"x5-orientation"* content=*"portrait"* />

<meta name=*"full-screen"* content=*"yes"* />

<meta name=*"x5-fullscreen"* content=*"true"* />

<meta name=*"browsermode"* content=*"application"* />

<meta name=*"x5-page-mode"* content=*"app"* />

<title>东链博BIM应用Server</title>

<link rel=*"shortcut icon"* href=*"/favicon.ico"* type=*"image/x-icon"* />

<!-- build:css -->

<link rel=*"stylesheet"* href=*"public/css/format.css"* type=*"text/css"* />

<link rel=*"stylesheet"* href=*"public/css/bim.css"* type=*"text/css"* />

<link rel=*"stylesheet"* href=*"public/css/guest.css"* type=*"text/css"* />

<link rel=*"stylesheet"* href=*"public/css/index.css"* type=*"text/css"* />

<link rel=*"stylesheet"* href=*"public/css/jquery.minicolors.css"* type=*"text/css"* />

<!-- endbuild -->

<style>

**html,body,***#mySceneCanvas*{width:*100%*;height:*100%*;overflow:*hidden*}

</style>

</head>

<body ng-controller=*"myAppCtrl"*>

<div id=*"luopanBox1"* ng-hide=*"majorTypedata.length==0"*></div>

<div id=*"luopanBox"* class=*"ui-draggable"*>

<div class=*"luopanBar"*>

<svg xmlns=*"http://www.w3.org/2000/svg"* version=*"1.1"* id=*"luopanSvg"* width=*"192"* height=*"192"*>

<image x=*"22"* y=*"22"* width=*"148"* height=*"148"* id=*"luopanSvgImg1"* xmlns:xlink=*"http://www.w3.org/1999/xlink"* xlink:href=*"./public/images/luopan\_bg.png"*></image>

</svg>

</div>

</div>

<canvas id=*"mySceneCanvas"* width=*"100%"* height=*"100%"*></canvas>

<header class=*"app-header navbar"*>

<a class=*"navbar-brand"* href=*"#"*>

<svg xmlns=*"http://www.w3.org/2000/svg"* width=*"10"* height=*"10"* viewbox=*'-263 -263 526 526'* fill=*"#00ccff"* class=*"svg flash animated"*>

<defs>

<filter id=*"Gaussian\_Blur"*>

<feGaussianBlur in=*"SourceGraphic"* stdDeviation=*"10"*/>

</filter>

<radialGradient id=*"grey\_blue"* cx=*"50%"* cy=*"50%"* r=*"50%"* fx=*"50%"* fy=*"50%"*>

<stop offset=*"0%"* style="stop-color:*rgb(255, 255, 255)*;stop-opacity:*1*"/>

<stop offset=*"80%"* style="stop-color:*rgb(144, 245, 255)*;stop-opacity:*1*"/>

</radialGradient>

</defs>

<polygon id=*'star'* cx=*"100"* cy=*"100"* rx=*"100"* ry=*"100"* style="fill:*url(#grey\_blue)*;stroke:*#fff9b8*;stroke-width:*10*; filter:*url(#Gaussian\_Blur)*;"points=*'250,0 64,64 0,250 -64,64 -250,0 -64,-64 0,-250 64,-64'* />

</polygon>

</svg>

</a>

<div class=*"c-div div\_project"*>

<a class=*"btn action-save c-button button\_wUyZlQ2"* type=*"button"* href=*"./admin/index.html#!/"*>

项目管理

</a>

</div>

</header>

<div class=*"aside-menu-box"*>

<ul class=*"aside-box"*>

<li class=*"li"* data-name=*"fileCtrl"*> <a class=*"font"* data-section=*""*> <i class=*"iconfont"*></i> <i class=*"iconfont1"*></i> <p> 文件 </p> </a> </li>

<li class=*"li"* data-name=*"treeCtrl"*> <a class=*"font"* data-section=*""*> <i class=*"iconfont"*></i> <i class=*"iconfont1"*></i> <p> 场景树 </p> </a> </li>

<li class=*"li"* data-name=*"typeCtrl"*> <a class=*"font"* data-section=*""*> <i class=*"iconfont"*></i> <i class=*"iconfont1"*></i> <p> 类型 </p> </a> </li>

<li class=*"li"* data-name=*"pTableCtrl"* > <a class=*"font"* data-section=*""*> <i class=*"iconfont"*></i> <i class=*"iconfont1"*></i> <p> 属性 </p> </a> </li>

<li class=*"li"* data-name=*"searchCtrl"*> <a class=*"font"* data-section=*""*> <i class=*"iconfont"*></i> <i class=*"iconfont1"*></i> <p> 搜索 </p> </a> </li>

<li class=*"li"* data-name=*"floorCtrl"*> <a class=*"font"* data-section=*""*> <i class=*"iconfont"*></i> <i class=*"iconfont1"*></i> <p> 楼层 </p> </a> </li>

<li class=*"li"* data-name=*"colorCtrl"*> <a class=*"font"* data-section=*""*> <i class=*"icon iconfont"*>&#xe60a;</i> <i class=*"iconfont1"*></i> <p> 配色 </p> </a> </li>

<li class=*"li"* data-name=*"majorCtrl"*> <a class=*"font"* data-section=*""*> <i class=*"icon iconfont"*>&#xe636;</i> <i class=*"iconfont1"*></i> <p> 专业 </p> </a> </li>

</ul>

</div>

<div class=*"nav-slide"*>

<div class=*"nav-iconback"*></div>

<div class=*"nav-slide-o"*>

<div>

<div ng-include=*"'./treeTemp/file.html'"*></div>

</div>

</div>

<div class=*"nav-slide-o"*>

<div>

<ul class=*"tree-view"*>

<li ng-repeat=*"item in treeData"* data-oid=*{{item.oid}}*>

<span ng-class=*"{true:'tree-minus-bg',false:'tree-add-bg'}[item.show]"* ng-click=*"item.show=!item.show"* ng-init=*"item.show=false"*></span><span class=*"ifc"*></span>{{item.ifcClassType}}{{item.name}}

<ul ng-include = *"'./treeTemp/contains.html'"* ng-if=*"item.show"*></ul>

<ul ng-include = *"'./treeTemp/decomposition.html'"* ng-if=*"item.show"*></ul>

</li>

</ul>

</div>

</div>

<div class=*"nav-slide-o"*>

<div>

<div ng-include=*"'./treeTemp/typeView.html'"*></div>

</div>

</div>

<div class=*"nav-slide-o"*>

<div id=*"pTable"*>

<div ng-include=*"'./treeTemp/pTable.html'"*></div>

</div>

</div>

<div class=*"nav-slide-o"*>

<div>

<div ng-include=*"'./treeTemp/search.html'"*></div>

</div>

</div>

<div class=*"nav-slide-o"*>

<div>

<div ng-include=*"'./treeTemp/floor.html'"*></div>

</div>

</div>

<div class=*"nav-slide-o"*>

<div>

<div ng-include=*"'./treeTemp/color.html'"*></div>

</div>

</div>

<div class=*"nav-slide-o"*>

<div>

<div ng-include=*"'./treeTemp/major.html'"*></div>

</div>

</div>

<!--<div class="nav-slide-o"></div> -->

</div>

<script type=*"text/javascript"* src=*"public/js/jquery/2.1.4/jquery-2.1.4.min.js"*></script>

<!-- build:js1 -->

<script type=*"text/javascript"* src=*"public/js/progress.js"*></script>

<script type=*"text/javascript"* src=*"public/js/websoketArrayBuffer.js"*></script>

<!-- endbuild -->

<!-- build:js2 -->

<script src=*"public/js/ifc/Constants.js"*></script>

<script src=*"public/js/ifc/DataInputStreamReader.js"*></script>

<script src=*"public/js/ifc/StringView.js"* type=*"text/javascript"*></script>

<script src=*"public/js/ifc/GeometryLoader.js"* type=*"text/javascript"*></script>

<!-- endbuild -->

<script src=*"public/js/scenejs/scenejs.js"*></script>

<script src=*"public/js/angularjs/1.4.4/angular.min.js"*></script>

<!-- build:js3 -->

<script src=*"public/js/app/sceneTree.js"*></script>

<script src=*"public/js/app/renderBim2.js"*></script>

<!-- endbuild -->

<!-- build:js4 -->

<script src=*"public/js/app/dlbSvg.js"* type=*"text/javascript"*></script>

<script src=*"public/js/app/menu.js"*></script>

<script src=*"public/js/app/qaq.js"*></script>

<script src=*"public/js/html5validate.js"* type=*"text/javascript"*></script>

<script src=*"public/js/color/jquery.minicolors.js"* type=*"text/javascript"*></script>

<!-- endbuild -->

<script src=*"public/js/datgui/dat.gui.min.js"* type=*"text/javascript"*></script>

<script>

**function** GetQueryString(name){

**var** reg = **new** RegExp("(^|&)" + name + "=([^&]\*)(&|$)","i");

**var** r = window.location.search.substr(1).match(reg);

**if** (r!=**null**) **return** unescape(r[2]); **return** **null**;

}

**var** websocket = **new** WebsocketBim();

**var** gui; //全局的控制面板

window.hisPick = {}; //记录点过的东西

window.allPoint = [];

**var** string = GetQueryString("rid");

**if**(!string){

**var** pid= GetQueryString("pid");

**if** (pid){

$.ajax({

url:"./model/queryModelInfoByPid.do?pid="+pid,

type:"POST",

dataType:"json",

aysnc:**false**,

success:**function**(res){

string =res.data.length?res.data[0].rid:4;

**var** maxZoom = 1000;

**if**(!window.scene){

createScene();

}

setTimeout(start,800)

}

});

}**else**{

rid = 1

}

}**else**{

**var** maxZoom = 1000;

**if**(!window.scene){

createScene();

}

setTimeout(start,800)

}

/\*\*

\* 开始请求后端数据

\*\*/

**function** start(){

websocket.init({

urlPath:"ws/streamgeometry?rid="+string,

onopen:**function**(msg){

window.progress = $.progress();

},

onmessage:**function**(res){

**if**(**typeof**(res)=='string'){

progress.update(JSON.parse(res));

}**else**{

geomotryLoad.process(res);

}

},

onclose:**function**(){

progress.hide();

websocket = **null**;

geomotryLoad = **null**;

$('.aside-menu-box').show();

/\*var Menus = function () {

this.message = "Directional light";

this["alpha.a"] = 0.6;

var self = this;

var update = function () {

for(var i=0;i<allPoint.length;i++){

scene.getNode(allPoint[i]+'geometry',function(mt){

mt.setAlpha(self["alpha.a"]);

})

}

requestAnimationFrame(update);

};

update();

};

gui = new dat.GUI();

var menu = new Menus();

var color = gui.addFolder('设置总体透明度');

color.add(menu, 'alpha.a', 0.0, 1.0);

color.open(); \*/

initStart();

}

});

}

//监听窗口关闭事件，当窗口关闭时，主动去关闭websocket连接，防止连接还没断开就关闭窗口，server端会抛异常。

window.onbeforeunload = **function** () {

**if**(websocket){

websocket.send('close');

}

}

</script>

</body>

</html>

**function** GeometryLoader() {

**var** o = **this**;

**this**.models = {};

o.state = {

nrObjectsRead: 0,

nrObjects: 0

};

o.stats = {

nrPrimitives: 0,

nrVertices: 0,

nrNormals: 0,

nrColors: 0

};

o.nodes = [];

o.step = 1;

**var** flagsObj = {

transparent : **true**,

backfaces:**true**,

enable:**true**

};

**this**.setModels = **function**(data){

o.models = data;

}

**this**.getModels = **function**(){

**return** o.models;

}

/\*\*

\* 读取一块数据

\*/

**this**.readObject = **function**(data, geometryType) {

**var** ifcname = data.readUTF8();

**var** material = Ifc.Constants.materials[ifcname] || Ifc.Constants.materials['DEFAULT'];

**var** ifcProductOid = data.readLong();//不同的

data.align8();

**var** transformationMatrix = data.readDoubleArray(16);

**if** (geometryType == 1) {

**var** geometryDataOid = data.readLong();

**var** indices = data.readShortArray((data.readInt()));

data.align4();

**var** IndicesForLinesWireFrame = data.readShortArray((data.readInt()));

data.align4();

**var** vertices = data.readFloatArray((data.readInt()));

**var** normals = data.readFloatArray((data.readInt()));

**var** colors = data.readFloatArray((data.readInt()));

o.library.addNodes([{

coreId:'geo'+geometryDataOid,

type : "geometry",

primitive : "triangles",

positions: vertices,

indices: indices,

normals: normals

},{

coreId:'geoLine'+geometryDataOid,

type : "geometry",

primitive : "lines",

positions: vertices,

indices: IndicesForLinesWireFrame,

normals: normals

}]);

**this**.processGeometry(1,ifcProductOid, [geometryDataOid], material, transformationMatrix)

} **else** **if**(geometryType == 2){

**var** nrParts = data.readInt();

data.align8();

**var** nodes = [];

**for** (**var** i=0; i<nrParts; i++) {

**var** coreId = data.readLong();

**var** indices = data.readShortArray((data.readInt()));

data.align4();

**var** vertices = data.readFloatArray((data.readInt()));

**var** normals = data.readFloatArray((data.readInt()));

**var** colors = data.readFloatArray((data.readInt()));

o.library.addNode({

type:"geometry",

primitive:"triangles",

positions: vertices,

indices: indices,

coreId:'geo'+coreId,

normals: normals

});

}

}**else** **if**(geometryType == 3){

**var** geometryDataOid = data.readLong();

**this**.processGeometry(3,ifcProductOid, [geometryDataOid], material, transformationMatrix)

}**else**{

**var** arraySize = data.readInt();

**var** coreIds = [];

**for** (**var** i=0;i<arraySize;i++) {

**var** coreId = data.readLong();

coreIds.push(coreId)

}

**this**.processGeometry(4,ifcProductOid, coreIds, material, transformationMatrix);

}

o.state.nrObjectsRead++;

o.updateProgress();

};

/\*\*

\* 往node中塞geometry数据

\*/

**this**.processGeometry = **function**(type,ifcProductOid, coreIds, material, transformationMatrix){

**var** coreNodes = [];

**var** coreNodesLine = [];

coreIds.forEach(**function**(coreId){

coreNodes.push({

type: "geometry",

coreId: 'geo'+coreId

});

**if**(type==1||type==3){

coreNodesLine.push({

type: "geometry",

coreId: 'geoLine'+coreId

});

}

});

**var** flags = {

type : "flags",

flags : flagsObj,

id : "flags"+ifcProductOid,

nodes : [{

type : "name",

name : ifcProductOid,

nodes:[{

type: "matrix",

elements: transformationMatrix,

nodes : [{

type : "material",

baseColor:{r:material.r,g:material.g,b:material.b},

color:{r:material.r,g:material.g,b:material.b},

alpha: material.a,

id:ifcProductOid+"geometry",

nodes: coreNodes

}]

}]

}]

};

**if**(type==1||type==3){

flags.nodes[0].nodes[0].nodes[1] = {

type : "material",

color:{r:0,g:0,b:0},

alpha:0.5,

nodes: coreNodesLine

}

}

o.models.addNode(flags);

flags = **null**;

}

**this**.updateProgress = **function**() {

progress.update({title:'transferring',progress:o.state.nrObjectsRead,max:o.state.nrObjects});

};

/\*\*

\* 读取开始部分数据

\*/

**this**.readStart = **function**(data){

**var** start = data.readUTF8();

**if** (start != "BGS") {

console.log("Stream does not start with BGS (" + start + ")");

**return** **false**;

}

**var** version = data.readByte();

**if** (version != 6) {

console.log("Unimplemented version");

**return** **false**;

} **else** {

o.state.version = version;

}

data.align8();

**var** modelBounds = data.readDoubleArray(6);

o.modelBounds = {

min: {x: modelBounds[0], y: modelBounds[1], z: modelBounds[2]},

max: {x: modelBounds[3], y: modelBounds[4], z: modelBounds[5]}

};

o.center = {

x: (o.modelBounds.max.x + o.modelBounds.min.x) / 2,

y: (o.modelBounds.max.y + o.modelBounds.min.y) / 2,

z: (o.modelBounds.max.z + o.modelBounds.min.z) / 2,

};

**var** zoom = Math.abs(o.modelBounds.max.x - o.modelBounds.min.x);

**if** (window.scene) {

window.scene.getNode("lookAt",**function**(lookat){

**var** eye = { x: (o.modelBounds.max.x - o.modelBounds.min.x) \* 0.5, y: (o.modelBounds.max.y - o.modelBounds.min.y) \* -1, z: (o.modelBounds.max.z - o.modelBounds.min.z) \* 0.5 };

lookat.setEye(eye);

lookat.setLook(o.center);

});

window.scene.getNode("main-camera",**function**(maincamera){

**var** diagonal = Math.sqrt(Math.pow(o.modelBounds.max.x - o.modelBounds.min.x, 2) + Math.pow(o.modelBounds.max.y - o.modelBounds.min.y, 2) + Math.pow(o.modelBounds.max.z - o.modelBounds.min.z, 2));

**var** far = diagonal \* 18; // 5 being a guessed constant that should somehow coincide with the max zoom-out-factor

maincamera.setOptics({

type: 'perspective',

far: far,

near: far / 1000,

aspect: jQuery(window).width() / jQuery(window).height(),

fovy: 37.8493

});

});

}

o.library = scene.findNode("library");

**if** (o.library == **null**) {

o.library = scene.addNode({

id: "library",

type: "library"

});

}

o.state.nrObjects = data.readInt();

//o.step = o.state.nrObjects<100 ? 1 :~~(o.state.nrObjects/100);

};

/\*\*

\* 根据消息类型进行二进制数据解析

\*/

**this**.process = **function**(res){

inputStream = **new** DataInputStream(res);

**var** messageType = inputStream.readByte();

**if** (messageType == 0) {

o.readStart(inputStream);

}**else**{

o.readObject(inputStream, messageType);

}

};

}