5. Praktické použite JAPI v príkladoch

5.1. Test správnosti adresovania RPC alebo SBA

Volanie RPC v D2000 obsahuje viacero úrovní adresácie:

- HOBJ procesu, na ktorý sa správa doruí (EVH, HIP, JAPI *Session*),
- HOBJ Event-u, na ktorom je RPC definovaná (v prípade JAPI Session má hodnotu 0),
- íslo inštancie Event-u v prípade inštanného Event-u alebo 0,
- „use Java" – logická hodnota rozlišujúca, i je RPC definovaná v D2000 Java Environment,
- HOBJ ESL Interface objektu, v ktorom je RPC deklarovaná v prípade implementácie interface procedúry alebo 0 v prípade, že je RPC deklarovaná priamo,
- meno RPC.

Zoznam parametrov nie je súasou adresy, ale pokia sa RPC zavolá s iným potom alebo typmi parametrov, ako je deklarovaná, ide o chybu.

V prípade volania SBA je adresácia jednoduchšia (vypadli „use Java", HOBJ ESL Interface a parametre volania majú implicitnú štruktúru).

Vzhadom na dynamickú povahu konfigurácie aplikácie v D2000 nie je možné zarui, že bude RPC vždy dostupná na rovnakom mieste. Aktuálny stav je však možné zisti volaním metód `D2Session.testRPC` a `D2Session.testSBA`.

> **Pozor**: Metódy zisujú stav v momente zavolania (resp. doruenia žiadosti na príslušný proces) a tento stav sa môže v reálnom ase zmeni. V prípade kladnej odpovede sa preto na túto skutonos do budúcnosti nedá celkom spoahnú. Negatívna odpove je však dobrý mechanizmus ako detegova, že klientská aplikácia nemá uspokojené všetky svoje požiadavky voi D2000 aplikácii.

Nasledujúci príklad demonštruje test, i je možné zavola:

- RPC s názvom `setData`,
- s parametrami:
  1. vstupný parameter typu `INT` (meno parametra nie je dôležité),
  2. vstupný parameter typu `RECORD NOALIAS(SD.Data)`,
  3. vstupno-výstupný parameter typu `BOOL`,
- implementovaný v ESL asti Event-u `E.Service`,
- bežiaci v procese `SELF.EVH`,
- nie je to inštanný Event, ani RPC neimplementuje ESL Interface.

Budúci výsledok vo forme Future je uložený v premennej upcomingResult.

```
FutureEvent<Integer> upcomingProcessHobj =
    D2SessionUtils.getFutureObjectHobjByName(session, "SELF.EVH");

FutureEvent<Integer> upcomingEventHobj =
    D2SessionUtils.getFutureObjectHobjByName(session, "E.Service");

int structureDefinitionHobj = session.getStructDefInfo("SD.Data").getHobj();

FutureEvent<Boolean> upcomingResult = session.testRPC(
    false,
    upcomingProcessHobj.get(),
    upcomingEventHobj.get(),
    0, // eventInstanceId
    0, // ESL interface HOBJ
    "SetData",
    new RpcParameterType[]
    {
        RpcParameterTypeFactory.createParameter(false, UnivalType.integer),
        RpcParameterTypeFactory.createParameter(false, structureDefinitionHobj),
        RpcParameterTypeFactory.createParameter(true, UnivalType.bool)
    });
```

5.2. ítanie dát z archívu

V nasledovnom príklade je uvedená statická metóda, ktorá umožní asynchrónne ítanie dát z archívu. Pre jednoduchos sú vypustené niektoré parametre prístupu do archívu. Maximálny poet preítaných záznamov z archívu je možné limitova parametrom *limitDataLength* volania *getArchiveValues*. Hodnota parametra -1 znamená, že ítanie je bez obmedzenia. Nie je však možné preíta viac záznamov, ako sa zmestí do pamäte procesu *arc.exe*.

V príklade je názorne použitá trieda *AdjustableFuture*, ktorá implementuje rozhranie `java.util.concurrent.Future` a zárove umožuje asynchrónne nastavi hodnotu ako aj prida asynchrónneho spracovatea hodnoty. Taktiež je v príklade použitý objekt typu *ObjectInfoCache*, ktorý po zvolenú dobu udržiava preklady názvov objektov na ich HOBJ.

```
public static FutureEvent<ArchiveResult> loadArchiveData(
        final D2Session session,
        final IndexedObjectName indexedObjectName,
        final Date beginTime,
        final Date endTime,
        final int limitDataLength) {

    final AdjustableFuture<ArchiveResult> result = new AdjustableFuture<>();

    final ObjectInfoCache cache = session.getConnector().getDefaultObjectInfoCache();

    final FutureEvent<Integer> futureHobj
            = cache.getHobj(indexedObjectName.getObjectName(), session);

    futureHobj.addEventHandler(
            new HobjHandler(result, session, indexedObjectName, beginTime, endTime,
                    limitDataLength));

    return result;
}
```

Trieda *ArchiveResult* obauje preítané hodnoty a popis archívneho objektu.

```
public class ArchiveResult {

    private final ArchiveObjectDescription description;
    private final List<UnivalValue<?>> values;

    private ArchiveResult(
            List<UnivalValue<?>> values,
            ArchiveObjectDescription description) {
        this.values = values;
        this.description = description;
    }

    public ArchiveObjectDescription getDescription() {
        return description;
    }

    public List<UnivalValue<?>> getValues() {
        return values;
    }
}
```

Trieda *HobjHandler* spracuje udalos získania HOBJ na základe mena objektu. V prípade neúspechu indikuje neúspech aj do výsledku pôvodného volania – objekt *result*. V prípade úspechu spustí ítanie dát z archívu so zvolenými parametrami.

```
private static class HobjHandler
        implements FutureEventHandler<Integer> {

    private final Date beginTime;
    private final Date endTime;
    private Integer hobj;
    private final IndexedObjectName indexedObjectName;
    private final int limitDataLength;
    private final AdjustableFuture<ArchiveResult> result;
    private final D2Session session;

    public HobjHandler(
            AdjustableFuture<ArchiveResult> result,
            D2Session session,
            IndexedObjectName indexedObjectName,
            Date beginTime,
            Date endTime,
            int limitDataLength) {
        this.result = result;
        this.session = session;
        this.indexedObjectName = indexedObjectName;
        this.beginTime = beginTime;
        this.endTime = endTime;
        this.limitDataLength = limitDataLength;
    }

    @Override
    public void onException(Throwable error) {
        result.setException(error);
    }

    @Override
    public void onResult(Integer hobj) {
        this.hobj = hobj;
        session.getArchiveValues(hobj,
                indexedObjectName.getRowIndex(),
                indexedObjectName.getColumnIndex(),
                beginTime,
                endTime,
                0,
                limitDataLength,
                new DataListener());
    }
```

Spracovanie dát z archívu je implementované v triede *DataListener*, ktorá je vnorenou triedou triedy *HobjHandler*. Implementuje rozhranie *ArchiveDataListener2*, ktoré je rozšírením rozhrania *ArchiveDataListener*.

Archív ako prvé pošle popis archívneho objektu. Následne posiela naítané dáta v balíkoch maximálne po 1000 záznamoch, ktoré sú zoradené vzostupne poda asovej znaky. Posledný balík dát je indikovaný hodnotou *true* parametra *noMoreData*. Vtedy je možné ítanie ukoni a výsledok uloži do premennej *result*.

Prípadnú chybu indikuje archív chybovým kódom. Chyba môže nasta kedykovek pred doruením posledného balíka dát – t.j. na zaiatku aj po prijatí jedného alebo viacerých balíkov.

V prípade, že zoznam prijatých hodnôt z archívu bol orezaný poda parametra *limitDataLength* – v archívnej databáze sa v asovom intervale od *beginTime* po *endTime* nachádzajú ešte nejaké alšie hodnoty – tak bude pred posledným balíkom dát zavolaná metóda *onMoreDataInArchive*. Vtedy je možné v ítaní pokraova nadväzujúcou požiadavkou na archív.

```
private class DataListener
        implements ArchiveDataListener2 {

    private ArchiveObjectDescription description = null;
    private boolean moreDataInArchive = false;
    private boolean skipFirst = false;
    private final List<UnivalValue<?>> values = new ArrayList<>();

    @Override
    public void onArchiveData(UnivalValue<?>[] data, boolean noMoreData) {
        List<UnivalValue<?>> list = Arrays.asList(data);
        if (this.skipFirst) {
            list = list.subList(1, data.length);
            this.skipFirst = false;
        }
        values.addAll(list);
        if (noMoreData) {
            if (this.moreDataInArchive) {
                this.moreDataInArchive = false;
                this.skipFirst = true;
                session.getArchiveValues(hobj,
                        indexedObjectName.getRowIndex(),
                        indexedObjectName.getColumnIndex(),
                        data[data.length - 1].getValueTime(),
                        endTime,
                        0,
                        limitDataLength,
                        this);
            } else {
                result.setValue(new ArchiveResult(values, description));
            }
        }
    }

    @Override
    public void onArchiveObjectDescription(ArchiveObjectDescription description) {
        this.description = description;
    }

    @Override
    public void onError(D2JapiErrorCode errorCode) {
        result.setException(new D2JapiException(errorCode));
    }

    @Override
    public void onMoreDataInArchive() {
        this.moreDataInArchive = true;
    }
}
```