Транзакционное хранилище типа ”ключ-значение” на базе событийно-ориентированной архитектуры

Студент:
Мария Давыдова
Руководитель:
Вадим Гуров, JetBrains

СПбАУ, 2012

Определения

- **Встраиваемые СУБД –** СУБД, которые поставляются как часть программного обеспечения и чаще всего представляют собой подключаемые библиотеки.
- **Событийно-ориентированная архитектура –** шаблон архитектуры программного обеспечения, позволяющий создание, определение, потребление и реакцию на события.

Введение в Node.js

Callback

```
object.asyncAction args, (error, result) =>
  someCallbackOperation
```

Этот механизм ограничен размером стека вызовов и позволяет добавлять только одного обработчика на событие.

Event

```
object.on 'event',  (result) =>
  someCallbackOperation
object.asyncAction
```

Цели

- Проверить гипотезу о возможности написания транзакционного хранилища типа ”ключ-значение” на базе событийно-ориентированной архитектуры.
  - Разработать соответствующие паттерны.

Exodus

- **Log**
  - виртуальное адресное пространство
  - взаимодействие с библиотекой ввода-вывода
- **BTree**
  - логическая структура данных
- **Environment**
  - внешний интерфейс
  - транзакции
  - восстановление

Java vs JavaScript (on Node.js)

Структурное программирование
vs
Событийно-ориентированное программирование

→ **Архитектурные паттерны**

Статическая типизация
(=> перегрузка функций есть)
vs
Динамическая типизация
(=> перегрузки функций нет)

→ **Языковые паттерны**

Класс-ориентированное ООП
vs
Прототипно-ориентированное ООП

Пример архитектурного паттерна с чисто асинхронной функцией

```
1. res = obj.syncOp()
2. result = nextInstr res
3. return result
```

```
1. onAsyncOp = (res) =>
2.   result = nextInstr res
3.   @emit 'newEvent', result
4. obj.once 'asyncOp', onAsyncOp
5. obj.asyncOp()
```

Примеры языковых паттернов

- Функции:
  - `foo() → foo()`
  - `foo(int v) → foo$int(v)`
  - `foo(int v, String s) → foo$int$String(v, s)`
- Конструкторы:
  - `A(T1 v1, T2 v2)` →

```
@create$T1$T2: (v1, v2, o) ->
  # initialization
  if !o? then o = new A
  # initialization
  return o
```

- Вызов родительского конструктора:
  - `super(...) → o = ParentClassName.create$...(..., o)`

Exodus on Node.js

- В качестве уровня структуры данных – связный список.
- Интерфейс эквивалентен Exodus.
- Поддержка множественных хранилищ, транзакций и курсоров.
- git://github.com/mariyafomkina/test_db.git
- **npm install exodus**

Спасибо за внимание!

Встраиваемые БД

- Снижение стоимости проектирования (одна БД, а не несколько версий).
- Устранение стоимости БД из общей стоимости продукта.
- Нет сложной логики запросов к внешней БД.
- Высокая производительность + *относительно* небольшой объём данных + хранение данных на одной машине.

Событийно-ориентированная модель

- Программа состоит из блоков.
- Блок состоит из выполняемых независимо (параллельно) однотипных операций, которые могут принимать параметры.
- Одна операция:

```
result = operation : case result
                         value1: goto Block1
                         value2: goto Block2
                         …
                         default: goto BlockN
```

- Переход от блока к блоку осуществляется путём создания и обработки событий (event).

Модель машины Тьюринга

```
BlockReadSI :
  result = read -> case result :
                       value1 : move to BlockWriteS1
                                with value1, D1
                       value2 : move to BlockWriteS2
                                with value2, D2

BlockWriteSI :
  result = write valuei, Di -> case result :
                                   true : move to
                                          BlockReadSI
```

Пример архитектурного паттерна с частично асинхронной функцией

```
1. res = obj.syncOp()
2. result = nextInstr res
3. return result
```

```
1.  emit = false
2.  onAsyncOp = (res) =>
3.    result = nextInstr res
4.    if emit then @emit 'newEvent', result
5.    else return result
6.  res = obj.asyncOp()
7.  if res?
8.    result = onAsyncOp res
9.    return result
10.else
11.   emit = true
12.   obj.once 'asyncOp', onAsyncOp
13.   return undefined
```