

*iserv*

ИНТЕРНЕТ-СЕРВИС

Более

17

лет

на рынке

RPC в NodeJS

# RPC - *Remote Procedure Call*

- action: string – имя сущности
- method: string – вызываемый метод сущности
- data: any[] - данные
- type: string – тип операции. По умолчанию “rpc”
- tid: number – идентификатор запроса

# Пример RPC запроса

```
[
  {
    "action": "users",
    "method": "Query",
    "data": [{
      "filter": [{
        "property": "login",
        "value": "user"
      }]
    }],
    "type": "rpc",
    "tid": 1
  }
]
```

Документация:

# RPC в браузере

## ▼ Request Headers

⚠ Provisional headers are shown

**Authorization:** Token cm9vdDowNzc5YTU2YWEzYmEwNDcw

**Content-Type:** application/json

**Origin:** http://localhost:1841

**Referer:** http://localhost:1841/

**User-Agent:** Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36

**X-Requested-With:** XMLHttpRequest

## ▼ Request Payload [view source](#)

```
▼ {action: "vw_users", method: "Query",...}
  action: "vw_users"
  ▼ data: [{page: 1, start: 0, limit: 25, filter: [{property: "login", value: "root"}]}]
    ▼ 0: {page: 1, start: 0, limit: 25, filter: [{property: "login", value: "root"}]}
      ▼ filter: [{property: "login", value: "root"}]
        ▼ 0: {property: "login", value: "root"}
          property: "login"
          value: "root"
          limit: 25
          page: 1
          start: 0
          method: "Query"
          tid: 3
          type: "rpc"
```

# Результат RPC

- meta
  - success: boolean – результат выполнения запроса
  - msg: string – текст ошибки, если success = false
- code: number – HTTP статус код
- result
  - records: any[] | any – массив данных или объект
  - total: number – число результатов
- time: number – время выполнения sql запроса
- sql
  - query: string – запрос к БД
- tid: number – идентификатор запроса
- type: string – тип запроса
- method: string – метод сущности
- action: string – имя сущности
- rpcTime: number – время выполнения RPC функции

# Результат RPC

```
1 [
2   {
3     "meta": {
4       "success": true,
5       "msg": "ok"
6     },
7     "code": 200,
8     "result": {
9       "records": [
10        {
11          "id": 1,
12          "name": "Главный",
13          "login": "root",
14          "claims": ".master.admin.filer.support.",
15          "email": null,
16          "tel": null,
17          "last_sign_in": null,
18          "icon_fileimage": null,
19          "description": null,
20          "address": null
21        }
22      ],
23      "total": 1
24    },
25    "time": 31,
26    "sql": {
27      "query": "SELECT * FROM public.vw_users as v WHERE v.login = 'root' LIMIT 25;",
28      "params": []
29    },
30    "tid": 3,
31    "type": "rpc",
32    "method": "Query",
33    "action": "vw_users",
34    "rpcTime": 33
35  }
36 ]
```

# RPC «СУЩНОСТЬ» В NodeJS

```
exports.[ИМЯ СУЩНОСТИ] = function(session) {  
  return {  
    isLocal: true, /// !!!  
  
    [ИМЯ СУЩНОСТИ]: function(data, callback) {  
      callback([результат]);  
    }  
  };  
}
```

# Пример RPC «сущности»

```
8  /**
9  | * объект для формирования ответа
10 | */
11 var result_layout = require('../postgresql-rpc-dbcontext/modules/result-layout');
12 var authorizeDb = require('../modules/authorize/authorizeDb');
13 var saltHash = require('../modules/authorize/saltHash');
14
15 /**
16 | * Объект с набором RPC функций
17 | */
18 exports.shell = function (session) {
19
20   return {
21     isLocal: true, // нужно указывать, иначе безопасность при создании meta не пропустит
22
23     /**
24     | * Получение серверного времени
25     | * @param {*} data
26     | * @param {*} callback
27     | * @example
28     | * // никаких параметров не нужно передавать
29     | * PN.shell.getServerTime({}, function(){});
30     | */
31     getServerTime: function (data, callback) {
32       callback(result_layout.ok([new Date()]));
33     },
```



# Регистрация созданной сущности в RPC

```
var shell = require('postgresql-rpc-shell');
```

```
var shellObject = {  
  db: db,  
  namespace: ns,  
  contexts: [  
    require('./modules/webapi'),  
    require('./modules/adminapi')  
  ],  
  ...  
};
```

```
app.use(shell(shellObject));
```

# postgresql-rpc-shell

Основная библиотека для работы с RPC в NodeJS

- Авторизация
- Обработка запросов
- Выполнение операции
- Фильтрация данных по безопасности

Документирование:

<https://docs.appcode.pw/projects?project=postgresql-rpc-shell>

# postgresql-rpc-shell

Основная библиотека для работы с RPC в NodeJS

- Авторизация
- Обработка запросов
- Выполнение операции
- Фильтрация данных по безопасности

Документирование:

<https://docs.appcode.pw/projects?project=postgresql-rpc-shell>

# Взаимодействие с БД

Запуск node bin/www



Подключение к БД  
и  
генерация схемы



Генерация  
dbcontext.js

```
...  
var render = require('dynamic-schema-reader'); // модуль генерации шаблонов
```

```
module.exports = function (callback) {  
  reader({  
    ...  
    connectionString: connectionString  
    autoRemove: true,  
    schemaList: ["core", "public", ...]  
  }, function (schemas) {  
    var content = ejs.compile(...);  
    fs.writeFileSync(path, content);  
    ...  
  });  
}
```

# Пример dbcontext.js

```
exports.chat = function (session) {
  return {
    Query: function (query_param, callback) {
      provider.select('public', 'chat', query_param, filter.security(session), callback);
    },
    Add: function (data, callback) {
      provider.insert('public', 'chat', data, callback);
    },
    Update: function (data, callback) {
      provider.update('public', 'chat', 'id', data, callback);
    },
    Delete: function (data, callback) {
      provider.delete('public', 'chat', 'id', data, callback);
    },
    Count: function (query_param, callback) {
      provider.count('public', 'chat', query_param, callback);
    }
  }
}
```

# postgresql-rpc-dbcontext

Основная библиотека для работы БД

Postgresql

- Выполнение CRUD операций с таблицами
- Вызов представлений и функций

Документирование:

<https://docs.appcode.pw/projects?project=postgresql-rpc-dbcontext>

# Примеры

- Создание собственной RPC функции и ее подключение
  - Способы возвращения результатов
- Добавление новой таблицы и обработка ее через RPC
- Просмотр данных в интерфейсе ExtJS
  - Интерфейс тестирования запросов
- Добавление безопасности, фильтрация данных
- Работа с websocket  
(<https://www.appcode.pw/?p=1256>)



*iserv*

ИНТЕРНЕТ-СЕРВИС

Спасибо за внимание!