لایه دسترسی به داده برای NodeJS به عنوان یک سرویس

گام دوم ، پیاده سازی فیچر های بیشتر

علی عسگری

فروردین 1401

عنوان : لایه دسترسی به داده برای NodeJS به عنوان یک سرویس گام دوم ، پیاده سازی فیچر های بیشتر

نگارش : علی عسگری

نام درس : معماری نرم افزار

استاد درس : دکتر مصطفی فخر احمد

کار با رویه (Procedures) های ذخیره شده

رویه های ذخیره شده بخش حیاتی هر پروژه با کیفیتی هستند، اجازه دهید ببینیم لایه دسترسی به داده ما به عنوان یک سرویس ، چگونه عملیات رایج رویه های ذخیره شده را مدیریت می کند.

هندل کردن پارامترهای ورودی

رویه‌های ذخیره‌شده پارامترهای ورودی را می‌پذیرند، ما رویه‌ای به نام SearchEmployee ساخته‌ایم که ورودی «Name» را می‌پذیرد.

```
router.get('/search', async (req, res) => {
    try {
        const result = await dataAccess.execute(`SearchEmployee`, [
            { name: 'Name', value: req.query.name }
        ]);
        const employees = result.recordset;

        res.json(employees);
    } catch (error) {
        res.status(500).json(error);
    }
});

//Ali.Askari @Shiraz.University
```

تصویر 1 - search به کمک سرویس dataAccess و Stored procedures

برای رویه های ذخیره شده، از روش execute سرویس دسترسی به داده استفاده می کنیم. متد execute بسیار شبیه به روش query است، با این تفاوت که نام رویه‌ها را به عنوان اولین پارامتر می‌پذیرد.

هندل کردن پارامترهای خروجی

Procedure های ذخیره شده پارامترهای خروجی را می پذیرند، ما رویه ای به نام GetEmployeesStatus ساخته ایم که تعدادی از پارامترهای خروجی را برمی گرداند.

```
router.get('/status', async (req, res) => {
    try {
        const result = await dataAccess.execute(`GetEmployeesStatus`, [],
[
            { name: 'Count', value: 0 },
            { name: 'Max', value: 0 },
            { name: 'Min', value: 0 },
            { name: 'Average', value: 0 },
            { name: 'Sum', value: 0 },
        ]);
        const status = {
            Count: +result.output.Count,
            Max: +result.output.Max,
            Min: +result.output.Min,
            Average: +result.output.Average,
            Sum: +result.output.Sum
        };

        res.json(status);
    } catch (error) {
        res.status(500).json(error);
    }
});

//Ali.Askari @Shiraz.University
```

تصویر 2 - GetEmployeesStatus

در اینجا، ما آرایه پارامتر خروجی را به عنوان پارامتر سوم عرضه می کنیم. پس از اجرا، این ویژگی خروجی آبجکت یا همان شیئ نتیجه را با مقادیر مناسب به روز می کند.

سر و کار داشتن با چندین مجموعه رکورد

رویه‌های ذخیره‌شده می‌توانند مجموعه‌های نتیجه متعددی را برگردانند، ما رویه‌ای به نام GetSalarySummary ساخته‌ایم که مجموعه‌های نتایج Department و Job را برمی‌گرداند.

```
router.get('/summary', async (req, res) => {
    try {
        const result = await dataAccess.execute(`GetSalarySummary`);
        const summary = {
            Department: result.recordsets[0],
            Job: result.recordsets[1],
        };

        res.json(summary);
    } catch (error) {
        res.status(500).json(error);
    }
});
//Ali.Askari @Shiraz.University
```

تصویر 3 - GetSalarySummary

در اینجا، ویژگی recordsets شی نتیجه حاوی نتایج متعددی است که از اجرا برگردانده شده‌اند، که می‌توان به آن‌ها با شاخص‌های آرایه دسترسی داشت.

هندل کردن پارامترهای جدول (Table)

رویه‌های ذخیره‌شده می‌توانند جدول را به‌عنوان پارامتر ورودی بپذیرند، ما رویه‌ای به نام AddEmployees ساخته‌ایم که Employees را به‌عنوان پارامتر با ارزش جدول می‌گیرد.

```
router.post('/many', async (req, res) => {
    try {
        const employees = req.body;
        const employeesTable = dataAccess.generateTable([
            { name: 'Code', type: dataAccess.mssql.TYPES.VarChar(50) },
            { name: 'Name', type: dataAccess.mssql.TYPES.VarChar(50) },
            { name: 'Job', type: dataAccess.mssql.TYPES.VarChar(50) },
            { name: 'Salary', type: dataAccess.mssql.TYPES.Int },
            { name: 'Department', type:
dataAccess.mssql.TYPES.VarChar(50) }
        ], employees);

        const result = await dataAccess.execute(`AddEmployees`, [
            { name: 'Employees', value: employeesTable }
        ]);
        const newEmployees = result.recordset;
        res.json(newEmployees);
    } catch (error) {
        console.log(error)
        res.status(500).json(error);
    }
});

//Ali.Askari @Shiraz.University
```

تصویر 4 - AddEmployees

در اینجا، ما از متد generateTable برای ایجاد یک ورودی مقدار جدول به نام workingsTable استفاده می کنیم. بعداً، با وارد کردن کارکنان Table به عنوان ورودی، با رویه AddEmployees تماس می گیریم.

خب اکثر سناریوهای رایج را با لایه دسترسی به داده خود پوشش داده ایم. دیدیم که چگونه از روش های مختلفی مانند generatorTable، execute ، queryEntity ، query و غیره برای رسیدگی به موارد مختلف استفاده می شود.

این پایان فیچر های افزون بر CURD ما بود ، اما در بحث های آینده اندکی از سادگی CURD در سرویس ساخته شده میکاهیم تا بدون نوشتن کوئری در فایل employees-variant که در حال استفاده از سرویس dataAccess ماست ، بتوانیم فرآیند های CURD را هندل کنیم.

```sql
CREATE PROCEDURE [dbo].[SearchEmployee]
    @Name VARCHAR(100)
AS
BEGIN
    SELECT * FROM Employee WHERE LOWER(Name) LIKE '%' + LOWER(@Name) + '%'
END

//Ali.Askari @Shiraz.University
```

```sql
CREATE PROCEDURE [dbo].[GetEmployeesStatus]
    @Count      INT OUTPUT,
    @Max        INT OUTPUT,
    @Min        INT OUTPUT,
    @Average    INT OUTPUT,
    @Sum        INT OUTPUT
AS
BEGIN
    SELECT  @Count      = COUNT(1),
            @Max        = MAX(Salary),
            @Min        = MIN(Salary),
            @Average    = AVG(Salary),
            @Sum        = SUM(Salary)
    FROM Employee;
END

//Ali.Askari @Shiraz.University
```

تصاویر 5 ، 6 - مربوط به رویه ها

```sql
CREATE PROCEDURE [dbo].[GetSalarySummary]
AS
BEGIN
    -- get department wise salary summary
    SELECT
        Department,
        COUNT(1) EmployeeCount,
        SUM(Salary) AS Salary,
        SUM(Salary) * 12 AS Annual
    FROM
        Employee
    GROUP BY
        Department
    ORDER BY
        SUM(Salary) DESC;

    -- get job wise salary summary
    SELECT
        Job,
        COUNT(1) EmployeeCount,
        SUM(Salary) AS Salary,
        SUM(Salary) * 12 AS Annual
    FROM
        Employee
    GROUP BY
        Job
    ORDER BY
        SUM(Salary) DESC;
END

//Ali.Askari @Shiraz.University
```

تصویر 7 - مربوط به رویه ها

```sql
CREATE TYPE [dbo].[EmployeeType] AS TABLE(
    [Code] [varchar](50) NOT NULL,
    [Name] [varchar](50) NULL,
    [Job] [varchar](50) NULL,
    [Salary] [int] NULL,
    [Department] [varchar](50) NULL
)
//Ali.Askari @Shiraz.University
```

```sql
CREATE PROCEDURE [dbo].[AddEmployees]
    @Employees EmployeeType READONLY
AS
BEGIN
    DECLARE @lastId INT;

    SET @lastId = (SELECT MAX(Id) AS LastId FROM Employee);

    INSERT INTO Employee (Code, [Name], Job, Salary, Department)
    SELECT * FROM @Employees;

    SELECT * FROM Employee WHERE Id > @lastId;
END
//Ali.Askari @Shiraz.University
```

تصاویر 8، 9 - مربوط به رویه ها

پایان گام دوم

با احترام و تشکر به خاطر وقتی که برای مطالعه گذاشتید.

همچنین از کارکردِ سیستم ویدیوی کوتاهی در فایل پروژه قرار گرفته است.

علی عسگری - فروردین 1400