عنوان: **نگاهی به افزونه‌های کار با اسناد در RavenDB**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۶/۱۹ ۰:۳۷
آدرس: www.dotnettips.info
برچسب‌ها: NoSQL, RavenDB

توانمندی‌های RavenDB جهت کار با اسناد، صرفا به ذخیره و ویرایش آن‌ها محدود نمی‌شوند. در ادامه، مباحثی مانند پیوست فایل‌های باینری به اسناد، نگهداری نگارش‌های مختلف آن‌ها، حذف آبشاری اسناد و وصله کردن آن‌ها را مورد بررسی قرار خواهیم داد. تعدادی از این قابلیت‌ها توکار هستند و تعدادی دیگر توسط افزونه‌های آن فراهم شده‌اند.

پیوست و بازیابی فایل‌های باینری

امکان پیوست فایل‌های باینری نیز به اسناد RavenDB وجود دارد. برای مثال به کلاس سؤالات قسمت اول این سری، خاصیت FileId را اضافه کنید:

```
public class Question
{
    public string FileId { set; get; }
}
```

اکنون برای ذخیره فایلی و همچنین انتساب آن به یک سند، به روش ذیل باید عمل کرد:

```
using (var store = new DocumentStore
            {
                Url = "http://localhost:8080"
            }.Initialize())
            {
                using (var session = store.OpenSession())
                {
                    store.DatabaseCommands.PutAttachment(key: "file/1",
                                                         etag: null,
                                                         data:
System.IO.File.OpenRead(@"D:\Prog\packages.config"),
                                                         metadata: new RavenJObject
                                                         {
                                                            { "Description", "توضیحات فایل" }
                                                         });

                    var question = new Question
                    {
                        By = "users/Vahid",
                        Title = "Raven Intro",
                        Content = "Test....",
                        FileId = "file/1"
                    };
                    session.Store(question);

                    session.SaveChanges();
                }
            }
```

کار متد store.DatabaseCommands.PutAttachment، ارسال اطلاعات یک استریم به سرور RavenDB است که تحت کلید مشخصی ذخیره خواهد شد. متد استاندارد System.IO.File.OpenRead روش مناسبی است برای دریافت استریم‌ها و ارسال آن به متد PutAttachment. در قسمت metadata این فایل، توسط شیء RavenJObject، یک دیکشنری از key-valueها را جهت درج اطلاعات اضافی مرتبط با هر فایل می‌توان مقدار دهی کرد. پس از آن، جهت انتساب این فایل ارسال شده به یک سند، تنها کافی است کلید آن‌را به خاصیت FileId انتساب دهیم.
در این حالت اگر به خروجی دیباگ سرور نیز دقت کنیم، مسیر ذخیره سازی این نوع فایل‌ها مشخص می‌شود:

```
Request # 2: PUT   - 200 ms - <system> - 201 - /static/file/1
```

بازیابی فایل‌های همراه با اسناد نیز بسیار ساده است:

```
using (var store = new DocumentStore
            {
                Url = "http://localhost:8080"
            }.Initialize())
            {
                using (var session = store.OpenSession())
                {
                    var question = session.Load<Question>("questions/97");
                    var file1 = store.DatabaseCommands.GetAttachment(question.FileId);
                    Console.WriteLine(file1.Size);
                }
            }
```

فقط کافی است سند را یکبار Load کرده و سپس از متد store.DatabaseCommands.GetAttachment برای دستیابی به فایل پیوست شده استفاده نمائیم.

وصله کردن اسناد

سند سؤالات قسمت اول و پاسخ‌های آن، همگی داخل یک سند هستند. اکنون برای اضافه کردن یک آیتم به این لیست، یک راه، واکشی کل آن سند است و سپس افزودن یک آیتم جدید به لیست پاسخ‌ها و یا در این حالت، جهت کاهش ترافیک سرور و سریعتر شدن کار، RavenDB مفهوم Patching یا وصله کردن اسناد را ارائه داده است. در این روش بدون واکشی کل سند، می‌توان قسمتی از سند را وصله کرد و تغییر داد.

```
using (var store = new DocumentStore
            {
                Url = "http://localhost:8080"
            }.Initialize())
            {
                using (var session = store.OpenSession())
                {
                    store.DatabaseCommands.Patch(key: "questions/97",
                                                 patches: new[]
                                                          {
                                                             new PatchRequest
                                                             {
                                                                Type = PatchCommandType.Add,
                                                                Name = "Answers",
                                                                Value = RavenJObject.FromObject(new
Answer{ By= "users/Vahid", Content="data..."})
                                                             }
                                                          });

                }
            }
```

برای وصله کردن اسناد از متد store.DatabaseCommands.Patch استفاده می‌شود. در اینجا ابتدا Id سند مورد نظر مشخص شده و سپس آرایه‌ای از تغییرات لازم را به صورت اشیاء PatchRequest ارائه می‌دهیم. در هر PatchRequest، خاصیت Type مشخص می‌کند که حین عملیات وصله کردن چه کاری باید صورت گیرد؛ برای مثال اطلاعات ارسالی اضافه شوند یا ویرایش و امثال آن. خاصیت Name، نام خاصیت در حال تغییر را مشخص می‌کند. برای مثال در اینجا می‌خواهیم به مجموعه پاسخ‌های یک سند، آیتم جدیدی را اضافه کنیم. خاصیت Value، مقدار جدید را دریافت خواهد کرد. این مقدار باید با فرمت JSON تنظیم شود؛ به همین جهت از متد توکار RavenJObject.FromObject برای اینکار استفاده شده است.

افزونه‌های RavenDB

قابلیت‌های ذکر شده فوق جهت کار با اسناد به صورت توکار در RavenDB مهیا هستند. این سیستم افزونه پذیر است و تاکنون افزونه‌های متعددی برای آن تهیه شده‌اند که در اینجا به آن‌ها Bundles گفته می‌شوند. برای استفاده از آن‌ها تنها کافی است فایل DLL مرتبط را درون پوشه Plugins سرور، کپی کنیم. دریافت آن‌ها نیز از طریق NuGet پشتیبانی می‌شود؛ و یا سورس آن‌ها را دریافت کرده و کامپایل کنید. در ادامه تعدادی از این افزونه‌ها را بررسی خواهیم کرد.

حذف آبشاری اسناد

```
PM> Install-Package RavenDB.Bundles.CascadeDelete -Pre
```

فایل افزونه حذف آبشاری اسناد را از طریق دستور نیوگت فوق می‌توان دریافت کرد. سپس فایل Raven.Bundles.CascadeDelete.dll دریافتی را درون پوشه plugins کنار فایل exe سرور RavenDB کپی کنید تا قابل استفاده شود. استفاده مهم این افزونه، حذف پیوست‌های باینری اسناد و یا حذف اسناد مرتبط با یک سند، پس از حذف سند اصلی است (که به صورت پیش فرض انجام نمی‌شود).
یک مثال:

```
var comment = new Comment
{
   PostId = post.Id
};
session.Store(comment);

session.Advanced.GetMetadataFor(post)["Raven-Cascade-Delete-Documents"] = RavenJToken.FromObject(new[]
{ comment.Id });
session.Advanced.GetMetadataFor(post)["Raven-Cascade-Delete-Attachments"] =
RavenJToken.FromObject(new[] { "picture/1" });

session.SaveChanges();
```

برای استفاده از آن باید از متد session.Advanced.GetMetadataFor استفاده کرد. در اینجا شیء post که دارای تعدادی کامنت است، مشخص می‌شود. سپس با مشخص سازی Raven-Cascade-Delete-Documents و ذکر Id کامنت‌های مرتبطی که باید حذف شوند، تمام این اسناد با هم پس از حذف post، حذف خواهند شد. همچنین دستور Raven-Cascade-Delete-Attachments سبب حذف فایل‌های مشخص شده با Id مرتبط با یک سند، می‌گردد.

نگهداری و بازیابی نگارش‌های مختلف اسناد

```
PM> Install-Package RavenDB.Bundles.Versioning
```

فایل افزونه Versioning اسناد را از طریق دستور نیوگت فوق می‌توان دریافت کرد. سپس فایل dll دریافتی را درون پوشه plugins کنار فایل exe سرور RavenDB کپی کنید تا قابل استفاده شود. فایل Raven.Bundles.Versioning.dll باید در پوشه افزونه‌ها کپی شود و فایل Raven.Client.Versioning.dll به برنامه ما ارجاع داده خواهد شد.
با استفاده از قابلیت document versioning می‌توان تغییرات اسناد را در طول زمان، ردیابی کرد؛ همچنین حذف یک سند، این سابقه را از بین نخواهد برد.
تنظیمات اولیه آن به این صورت است که توسط شیء VersioningConfiguration به سشن جاری اعلام می‌کنیم که چند نگارش از اسناد را ذخیره کند. اگر Exclude آن به true تنظیم شود، اینکار صورت نخواهد گرفت.

```
session.Store(new VersioningConfiguration
{
  Exclude = false,
  Id = "Raven/Versioning/DefaultConfiguration",
  MaxRevisions = 5
});
```

تنظیم Id به Raven/Versioning/DefaultConfiguration، سبب خواهد شد تا VersioningConfiguration فوق به تمام اسناد اعمال شود. اگر نیاز است برای مثال تنها به BlogPosts اعمال شود، این Id را باید به Raven/Versioning/BlogPosts تنظیم کرد.
بازیابی نگارش‌های مختلف یک سند، صرفا از طریق متد Load میسر است و در اینجا شماره Id نگارش به انتهای Id سند اضافه می‌شود. برای مثال "blogposts/1/revisions/1" به نگارش یک مطلب شماره یک اشاره می‌کند.
برای بدست آوردن سه نگارش آخر یک سند باید از متد ذیل استفاده کرد:

```
var lastThreeVersions = session.Advanced.GetRevisionsFor<BlogPost>(post.Id, 0, 3);
```