**KỸ THUẬT LẬP TRÌNH C#**

**MỘT SỐ HÀM BỔ TRỢ LÀM VIỆC TRÊN TẬP DỮ LIỆU**

**Phan Minh Tài** *| Founder of TitiSoft|CafeT.vn|ChuyenTin.vn|…*

**MỞ ĐẦU**

Trong các ứng dụng thực tế, cỡ vài chục đối tượng trong một bài toán là khá phổ biến, và thường thì chúng ta sẽ phải tương tác/xử lý khá nhiều với các kiểu dữ liệu mang tính tập hợp như Collection, List, Stack, …

Bài này chúng ta bàn luận một vài hàm bổ trợ để dễ xử lý trên các đối tượng này

**CÁC PHƯƠNG THỨC THƯỜNG GẶP**

Đối với các kiểu dữ liệu mang tính tập hợp, chúng ta thường nghe đến khái niệm Generic, Collection, Enums, … và thường hay gặp các thao tác sau đây:

* Foreach ()
* OderBy
* Filter
* Select
* ….

Nhưng còn một vài thao tác khác chúng ta cũng thường dùng nhưng không có cài đặt sẵn, chẳng hạn:

* Xét tập T là một BO (BusinessObject), có nhiều trường.
* Ta thực hiện:
  + Lấy tất cả các

Products – là danh sách các sản phẩm

* GetAll()
* GetById()
* FilterBy()
* GetNews()

Trừu tượng hơn, ta có:

* Items – tập các đối tượng
  + Lấy tất cả các đối tượng có CreatedBy = “Phan Minh Tài”
    - Nếu đối tượng không có trường CreatedBy

**CỤ THỂ HƠN**

* Ta muốn bổ sung một hàm HasProperty(string name) nhằm kiểm tra một lớp bất kỳ có thuộc tính “name” hay không và áp dụng cho tất cả các đối tượng có trong bài toán. Ta nên dùng abstract.

Ta tiếp tục với một lớp mẫu khác, ở đây là Article (Bài viết), vì rằng hiện tại CafeT.vn – Một website chuyên đề về Công nghệ và Ứng dụng, chia sẻ nhiều bài viết/nghiên cứu có giá trị, cho nên cái Article cũng cần phải thiết kế sao cho trọn vẹn. Chúng ta sẽ đi qua mấy điểm chính:

* ApplicationDbContext – đương nhiên là đại diện cho Database trong ứng dụng, sử dụng Code First, chúng ta có thể tương tác với CSDL thông quan Entity Framework. Ngoài ra, để thuận tiện và đảm bảo các nguyên tắc thiết kế, ta tiếp tục sử dụng Generic Repository, Unit Of Work (đang tạm sử dụng một thư viện mẫu do LongLe xây dựng).
* Khai báo Article, thừa kế từ BaseObject

public class ArticleModel : Article

{

public string EnglishContent { set; get; }

public string VietnameseContent { set; get; }

public int? Points { set; get; }

public virtual ImageModel Avatar { set; get; }

public Guid? CourseId { set; get; }

public Guid? ProjectId { set; get; }

public Guid? CompanyId { set; get; }

public Guid? JobId { set; get; }

[Display(Name = "Phân loại")]

public Guid? CategoryId { set; get; }

public virtual ArticleCategory Category { set; get; }

public string Followers { set; get; }

public string FromUrl { set; get; }

public IEnumerable<QuestionModel> Questions { set; get; }

public IEnumerable<CommentModel> Comments { set; get; }

public IEnumerable<FileModel> Files { set; get; }

public ArticleModel() : base()

{

Points = 0;

}

//…………………

}

Ta tiếp tục để ý, ArticleModel – là đối tượng (đang dùng) trong ứng dụng, còn Article là đối tượng mà tôi sử dụng chung tại CafeT.Frameworks.BussinessObjects – tập hợp các khai báo những đối tượng hay dùng, thừa kế từ BaseObject

**MỘT VÀI BÀI TOÁN KHÁC**

**Tạo bản sao cho một đối tượng bất kỳ (Clone)**

* Đây cũng là một bài toán hay gặp, chẳng hạn trong ngữ cảnh của ứng dụng hiện nay tôi có nhu cầu khá rõ nét:
  + Giả sử bài viết ban đầu (do người dùng biên tập) sẽ được giữ nguyên và đánh dấu là Layer[0] – hình dung theo nghĩa trải bài viết lên một mặt phẳng.
  + Tôi muốn phân tích sâu hơn bài viết hoặc xử lý nó theo nhiều tình huống khác nhau tùy theo bối cảnh, chẳng hạn ở đây:
    - Layer[1] – Tôi hiệu chỉnh chính tả.
    - Layer[2] – Tôi đánh dấu chỉ số (dưới) của tất cả các từ có trong văn bản.
    - Layer[3] – Tôi tiến hành dịch nó sang một ngôn ngữ nào đó (chẳng hạn tiếng Anh) – thường thì tôi dùng công cụ dịch như Google hoặc Microsoft, sau đó lưu kết quả lại, khi cần thì dùng.
    - Layer[4] – Tôi đánh dấu tất cả các từ chưa biết (so khớp với từ điển cá nhân), …
    - ….

Như vậy thì các Layer này sẽ được tổ chức như thế nào ?



**Viewers – Người xem đối tượng**

* Đối với một đối tượng bất kỳ (mang tính bussiness) thường thì có mấy hành động cụ thể đối với nó:
  + Xóa
  + Sửa
  + Tạo
  + Xem
  + (và hành động không phải trực tiếp – Tìm kiếm)
* Việc theo dõi xem ai đã (Xem, Sửa, Xóa, Tạo) và đảm bảo an toàn khi thực thi các hành động trên đối với từng đối tượng thực sự cần thiết.
* Một biểu mẫu lấy yêu cầu cụ thể có thể như sau:
  + Lớp/Đối tượng
  + Ý nghĩa
  + Mục đích sử dụng
  + Tương tác
    - Xem: {Ai, như thế nào, ảnh hưởng ra sao}
    - Xóa {Ai, như thế nào, …}
* Như vậy về bản chất đối tượng sẽ phải có:
  + Các hành động kiểm tra trước khi tương tác với đối tượng nên để cho ObjectManager<T> thực thi.
  + Bản thân nội tại nó không tự nó thực thi được các hành động như CRUD, tuy nhiên nó có thể tự quyết định được việc (Xem) sẽ được thực hiện như thế nào cũng như quá trình xem nó thì xem được những gì, và khi bị xem nó phản ứng như thế nào?
    - ObjectManager<T>.GetToView()
    - ObjectManager<T>.GetToDelete()
    - ….

**Biến hình các đối tượng (Transform)**

**LỚP UrlObject**

Cũng là một lớp các đối tượng thường gặp trong các bài toán liên quan đến Data/Web Minning, và trong thực tế tôi cũng sử dụng UrlObject để khai thác dữ liệu cho các website.

UrlObject

* string url: Là địa chỉ đầu vào, việc khởi tạo nó cần kiểm tra, nếu url.IsUrl() (– hàm mở rộng tự xây dựng) thì mới khởi tạo thành công, còn nếu không thì không tạo được.