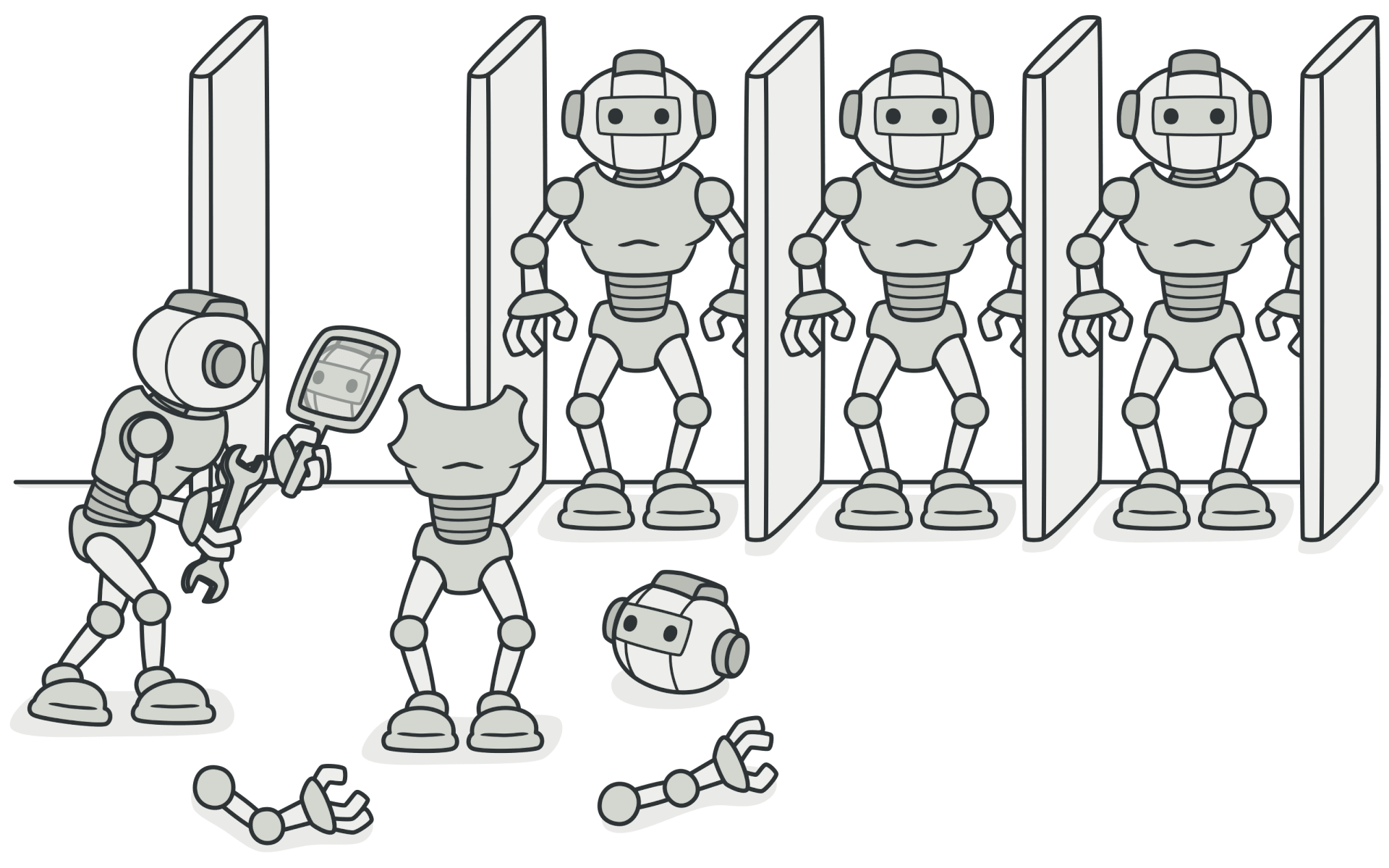
**Prototype Design Pattern - Trợ thủ đắc lực của Developers**



**1. Giới thiệu**

* Prototype là một design pattern thuộc nhóm Creational Pattern - những mẫu thiết kế cho việc khởi tạo object của lớp.
* Prototype quy định loại của các đối tượng cần tạo bằng cách dùng một đối tượng mẫu, tạo mới nhờ vào sao chép đối tượng mẫu này mà không làm cho code phụ thuộc vào các lớp của chúng.
* Prototype Pattern được dùng khi việc tạo một object tốn nhiều chi phí và thời gian trong khi bạn đã có một object tương tự tồn tại.
* Prototype Pattern cung cấp cơ chế để copy từ object ban đầu sang object mới và thay đổi giá trị một số thuộc tính nếu cần.
* Tần suất sử dụng: medium

**2. Mục đích ra đời**

**Problem**

Giả sử bạn có một object và bạn muốn tạo một bản sao chính xác của nó. Bạn sẽ làm điều này như thế nào? Đầu tiên, bạn phải tạo một object mới của cùng một lớp. Sau đó, bạn phải đi qua tất cả các field của object gốc và sao chép các giá trị của chúng sang object mới.

Không phải tất cả các object đều có thể được sao chép theo cách đó vì một số field của object có thể là phương thức private và không thể sử dụng từ bên ngoài object.

Diagram

Description automatically generated

**Solution**

Prototype pattern delegate (ủy nhiệm) quá trình cloning cho các object thực tế(realization) đang được clone lại. Pattern khai báo một giao diện chung cho tất cả các đối tượng hỗ trợ nhân bản. Giao diện này cho phép bạn sao chép một đối tượng mà không cần ghép mã của bạn với lớp của đối tượng đó. Thông thường, một giao diện như vậy chỉ chứa một phương thức sao chép duy nhất.

Việc triển khai phương thức clone là rất giống nhau trong tất cả các lớp. Phương thức này tạo một object của lớp hiện tại và chuyển tất cả các giá trị field của object cũ sang một object mới. Thậm chí có thể sao chép các private field vì hầu hết các ngôn ngữ lập trình đều cho phép các object truy cập vào các private field của các object khác thuộc cùng một lớp.

Một object hỗ trợ sao chép được gọi là prototype. Khi các object có hàng chục field và hàng trăm configuration, thì việc cloning chúng có thể là một giải pháp thay thế cho phân lớp con.

A picture containing diagram

Description automatically generated

Tạo một tập hợp các object, được xác định cấu hình theo nhiều cách khác nhau. Khi cần một object giống như object bạn đã cấu hình, chỉ cần clone một prototype thay vì xây dựng một object mới từ đầu.

**3. Kiến trúc**

Diagram

Description automatically generated

Các thành phần trong mô hình:

* Prototype: khai báo một đối tượng cho việc sao chép của nó.
* ConcretePrototype: thực thi một hoạt động cho việc sao chép của nó.
* Client: tạo ra một đối tượng mới bằng việc yêu cầu một kiểu mẫu để sao chép của nó.

Tránh việc tạo nhiều lớp con cho mỗi đối tượng tạo như của Abstract Factory Pattern.

Giảm chi phí để tạo ra một đối tượng mới theo “chuẩn”, tức là việc này tăng Performance so với việc sử dụng từ khóa new để tạo đối tượng mới.

Để cài đặt Prototype Pattern, tạo ra một Clone() Method ở lớp cha, và triển khai ở các lớp con. Cách cloning trong Prototype có 2 kiểu:

* *Shallow copy:* Các object con bên trong chỉ được copy reference. Nghĩa là chỉ nhân bản được value type. (Object ban đầu và object tạo mới đều trỏ tới chung 1 object con bên trong)
* *Deep copy:* Các object con bên trong cũng được copy lại toàn bộ các thuộc tính. Nghĩa là nhân bản được value type và reference type.

**4. Ưu & nhược điểm**

Prototype ẩn đi những lớp con rời rạc từ phía client, do vậy làm giảm đi số lớp con mà client cần biết. Hơn thế nữa pattern này làm cho client hoạt động với những lớp con cụ thể mà không phải thay đổi gì.

**Ưu điểm**

* *Thêm và loại bỏ lớp concrete lúc run-time:* Prototype hợp nhất quá trình khởi tạo một object mới vào trong hệ thống đơn giản chỉ bằng cách đăng ký một thực thể nguyên mẫu với client. Điều đó linh động hơn một chút so với các creational pattern khác bởi vì client có thể cài đặt và loại bỏ các pattern tại thời điểm run-time.
* *Khởi tạo object mới bằng cách thay đổi một vài attribute của object (các object có ít điểm khác biệt nhau):* Một hệ thống linh động sẽ để cho chúng ta tự define một hành động nào đó thông qua sự kết hợp với một object (nghĩa là một phương thức của một class) hơn là define một class mới. Client có thể thể hiện một tác động khác bằng cách ủy quyền cho lớp prototype. Đồng thời cách thức khởi tạo này cũng giúp cho chúng ta khai báo một “lớp mới” mà không phải lập trình gì cả. Thực tế thì việc copy một nguyên mẫu giống như việc khởi tạo một object từ một class mới. Prototype pattern giúp giảm số lớp mà hệ thống cần dùng.
* *Khởi tạo object mới bằng cách thay đổi cấu trúc:* Rất nhiều ứng dụng xây dựng hệ thống từ nhiều phần và các phần con. Các phần con lại khởi tạo từ nhiều phần con khác (chia nhỏ bài toán). Prototype pattern cũng hỗ trợ điều này. Nghĩa là các phần đó có thể được khởi tạo từ việc copy một nguyên mẫu từ một “cấu trúc” khác. Miễn là các phần kết hợp đều thể hiện Clone() và được sử dụng với cấu trúc khác nhau làm nguyên mẫu.
* *Giảm việc phân lớp:* Đôi khi hệ thống quá phức tạp vì có quá nhiều class, và cây thừa kế của lớp khởi tạo có quá nhiều lớp song song cùng mức. Prototype pattern rõ ràng làm giảm số lớp và sự phức tạp của cây thừa kế (class hierarchy). Một điểm hạn chế của prototype pattern là đôi khi việc phải implement mỗi phương thức Clone() của mỗi lớp con của Prototype sẽ gặp khó khăn. Ví dụ như việc implement Clone() sẽ khó khăn khi mà những object nội hàm của chúng không hỗ trợ việc copy hoặc có một reference không copy được reference.

**Nhược điểm**

* Clone các object phức tạp có phụ thuộc vòng (Circular Reference) có thể rất khó.

**5. Khi nào thì sử dụng**

* Giống như những mẫu thiết kế tạo lập khác (Builder, Abstract Factory và Factory Method), mẫu thiết kế Prototype ẩn việc tạo đối tượng từ client. Tuy nhiên, thay cho việc tạo ra một đối tượng không được thiết lập, nó trả về một đối tượng mới đã được thiết lập với các giá trị mà nó đã sao chép từ một đối tượng kiểu mẫu.
* Mẫu thiết kế Prototype không được sử dụng phổ biến trong việc xây dựng các ứng dụng nghiệp vụ (business application). Nó thường được sử dụng trong các kiểu ứng dụng xác định như đồ họa máy tính, CAD (Computer Assisted Drawing), GIS (Geographic Information Systems) và các trò chơi.
* Mẫu thiết kế Prototype tạo ra các bản sao của các đối tượng mẫu tồn tại trước đó. Cách tốt nhất để thực hiện việc này trong .NET là sử dụng iCloneable interface có sẵn trên các đối tượng được dùng như các kiểu mẫu. ICloneable interface có một phương thức gọi là Clone trả về một đối tượng là một bản sao của đối tượng gốc.
* Khi thực hiện chức năng Clone, bạn cần chú ý đến 2 kiểu khác nhau sau: deep copy và shallow copy. Shallow copy thì dễ dàng hơn nhưng chỉ sao chép các trường dữ liệu trong bản thân đối tượng – không phải các đối tượng mà kiểu mẫu đưa ra. Deep copy sao chép đối tượng kiểu mẫu và tất cả đối tượng nó đưa ra. Shallow copy thì dễ thực thi vì lớp Object có một phương thức MemberwiseClone trả về một shallow copy của đối tượng. Chiến lược sao chép cho deep copy có thể phức tạp hơn – một số đối tượng không được sao chép dễ dàng (chẳng hạn như Threads, các kết nối cơ sở dữ liệu, …).
* Tính linh hoạt nhờ vào sự đa xạ của phương thức clone(), ta có thể tạo lập 1 đối tượng mới mà ko hề cần viết mã gọi hàm tạo lập một cách cụ thể (việc cần tạo cụ thể 1 kiểu nào đó thì ta giao cho các quá trình tiền xử lý khác, và các quá trình tiền xử lý này hẳn cũng ko phụ thuộc vào hàm tạo lập cụ thể, một lần nữa ta lại được giải phóng…)
* Tránh việc tạo nhiều lớp con cho mỗi tượng như của Abstract Factory Pattern
* Giảm chi phí đáng kể so với tạo lập 1 đối tượng theo phương thức chuẩn, gọi toán tử new (gọi hàm tạo lập cụ thể)
* Ta có thể tùy chỉnh các thuộc tính đối tượng mẫu để tạo ra những đối tượng mẫu mới. Hay có thể hiểu là ta tạo ra một “chuẩn” class mới từ class có sẵn mà không cần viết code để định nghĩa.
* Ngoài ra còn rất nhiều lợi ích khác từ Design pattern: Prototype mà ta có thể ứng dụng vào trong việc lập trình.

**6. Source code minh họa với C#**

public class Person

{

public int Age;

public DateTime BirthDate;

public string Name;

public IdInfo IdInfo;

//Các object con bên trong chỉ được copy reference. Nghĩa là chỉ nhân bản được value type.

public Person ShallowCopy()

{

return (Person)this.MemberwiseClone();

}

//Các object con bên trong cũng được copy lại toàn bộ các thuộc tính.

public Person DeepCopy()

{

Person clone = (Person)this.MemberwiseClone();

clone.IdInfo = new IdInfo(IdInfo.IdNumber);

clone.Name = String.Copy(Name);

return clone;

}

}

public class IdInfo

{

public int IdNumber;

public IdInfo(int idNumber)

{

this.IdNumber = idNumber;

}

}

class Program

{

static void Main(string[] args)

{

Person p1 = new Person();

p1.Age = 42;

p1.BirthDate = Convert.ToDateTime("1977-01-01");

p1.Name = "Jack Daniels";

p1.IdInfo = new IdInfo(666);

// Thực thi shallow copy của P1 và truyền nó cho P1

Person p2 = p1.ShallowCopy();

// Thực thi deep copy của P1 và truyền cho P3

Person p3 = p1.DeepCopy();

// Hiển thị giá trị P1, P2, P3

Console.WriteLine("Original values of p1, p2, p3:");

Console.WriteLine(" p1 instance values: ");

DisplayValues(p1);

Console.WriteLine(" p2 instance values:");

DisplayValues(p2);

Console.WriteLine(" p3 instance values:");

DisplayValues(p3);

// Thay đổi giá trị của thuộc tính P1 và hiển thị giá trị P1, P2, P3

p1.Age = 32;

p1.BirthDate = Convert.ToDateTime("1900-01-01");

p1.Name = "Frank";

p1.IdInfo.IdNumber = 7878;

Console.WriteLine("\nValues of p1, p2 and p3 after changes to p1:");

Console.WriteLine(" p1 instance values: ");

DisplayValues(p1);

Console.WriteLine(" p2 instance values (reference values have changed):");

DisplayValues(p2);

Console.WriteLine(" p3 instance values (everything was kept the same):");

DisplayValues(p3);

}

public static void DisplayValues(Person p)

{

Console.WriteLine(" Name: {0:s}, Age: {1:d}, BirthDate: {2:MM/dd/yy}",

p.Name, p.Age, p.BirthDate);

Console.WriteLine(" ID#: {0:d}", p.IdInfo.IdNumber);

}

}

**7. Design Pattern liên quan**

* Nhiều thiết kế bắt đầu bằng cách sử dụng Factory Method (ít phức tạp hơn và có thể tùy chỉnh nhiều hơn thông qua các lớp con) và phát triển theo hướng Abstract Factory, Prototype hoặc Builder (linh hoạt hơn nhưng phức tạp hơn).
* Các lớp Abstract Factory thường dựa trên một tập hợp các Factory Methods, nhưng cũng có thể sử dụng Prototype để soạn các phương thức trên các lớp này.
* Prototype có thể hữu ích khi bạn cần lưu các bản sao của Commands vào lịch sử.
* Các thiết kế sử dụng nhiều Composite và Decorator thường có thể được hưởng lợi từ việc sử dụng Prototype. Áp dụng pattern cho phép clone các cấu trúc phức tạp thay vì xây dựng lại chúng từ đầu.
* Prototype không dựa trên tính kế thừa, vì vậy nó không có nhược điểm. Mặt khác, Prototype yêu cầu khởi tạo cloned object phức tạp. Factory Method dựa trên kế thừa nhưng không yêu cầu bước khởi tạo.
* Abstract Factories, Builders và Prototype đều có thể được triển khai dưới dạng các Singleton.

Bài viết của mình đến đây là kết thúc, cảm ơn các bạn đã theo dõi. Nếu các bạn thấy có ích có thể khám phá thêm [Series Design Patterns - Trợ thủ đắc lực của Developers](https://viblo.asia/s/design-patterns-tro-thu-dac-luc-cua-developers-Q75wqJ67ZWb) của mình!!

**Tài liệu tham khảo**

[1] Refactoring.Guru. <https://refactoring.guru/design-patterns>

[2] Design Patterns for Dummies, Steve Holzner, PhD

[3] Head First, Eric Freeman

[4] Gang of Four Design Patterns 4.0

[5] Dive into Design Pattern