POLITÉCNICO COLOMBIANO
JAIME ISAZA CADAVID

Patrón de diseño

ITERATOR

Realizado por
Sebastian Zapata Castaño

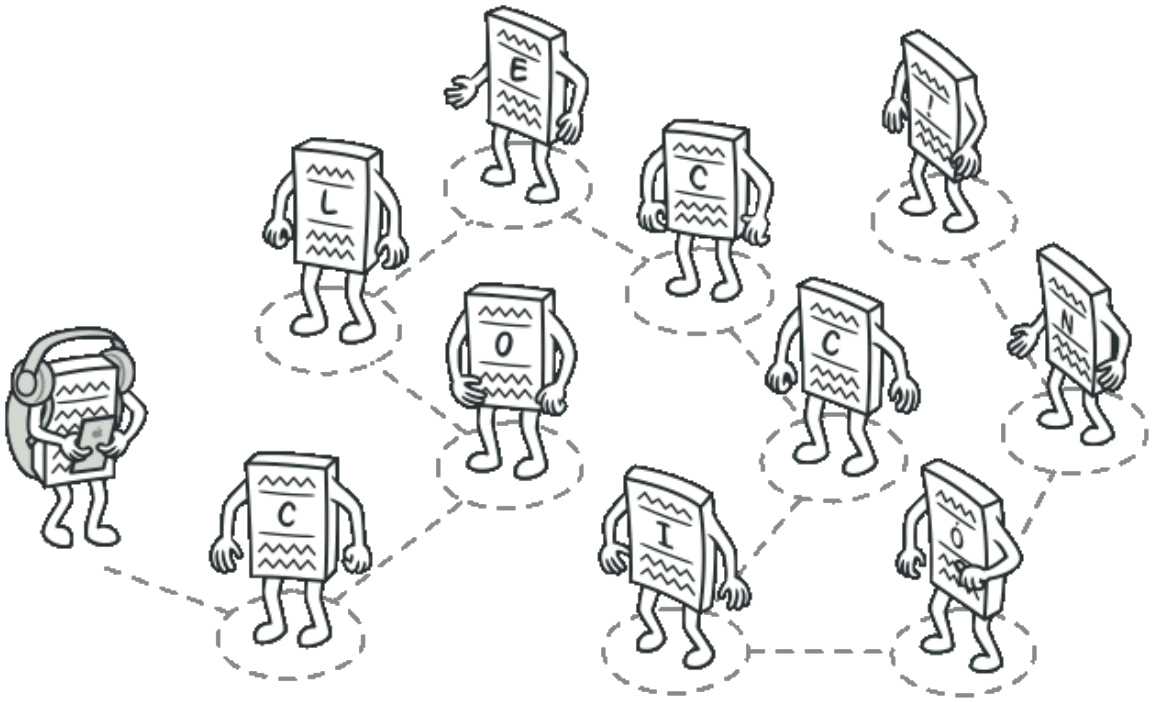
Dirigido por
Hector Manuel Vanegas Solis

Universidad
Politecnico Colombiano Jaime Isaza Cadavid

Febrero, 2022

Descripcion general

Este patrón de diseño hace posile atravesar una estructura de datos sin tener que conocer su estructura interna. Es especialmente útil cuando estamos traajando con estructuras de datos complejas ya que nos permite iterar a través de sus elementos usando Iterator Iterator es una interfaz que proporciona los métodos necesarios para iterar sore los elementos de una estructura de datos. Iterator es un patrón de diseño de comportamiento que te permite recorrer elementos de una colección sin exponer su representación subyacente (lista, pila, árbol, etc.).



¿Que soluiona este patrón de diseño?

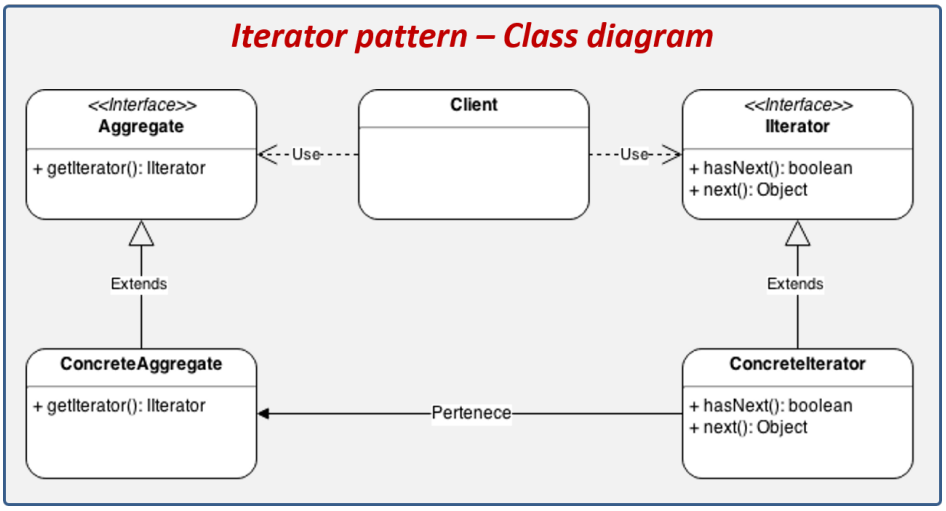
Las colecciones son muy usadas en el mundo de la programacion, solo bastaría con una ciclo para recorrerlos, pero ¿que pasa para recorrer de forma secuencial un arbol? Entonces, la idea central del patrón Iterator es extraer el comportamiento de recorrido de una colección y colocarlo en un objeto independiente llamado iterador.

Además de implementar el propio algoritmo un ojeto iterador encapsula todos los detalles del recorrido como la posición actual y cuántos elementos quedan hasta el final. Deido a esto varios iteradores pueden recorrer la misma colección al mismo tiempo independientemente los unos de los otros.

Normalmente los iteradores aportan un método principal para extraer elementos de la colección. El cliente puede continuar ejecutando este método hasta que no devuelva nada lo que significa que el iterador ha recorrido todos los elementos.

Todos los iteradores deen implementar la misma interfaz. Esto hace que el código del cliente sea compatile con cualquier tipo de recopilación o algoritmo transversal siempre que exista un iterador adecuado. Si necesitara un método específico para iterar sore un conjunto crearía una nueva clase de iterador sin tener que modificar el conjunto o el cliente.

Iterator como diagrama UML



Ventajas y desventajas de Estrategia

0.0.1. Ventajas del patrón de diseño iterator

Principio de responsabilidad única. Puedes limpiar el código cliente y las colecciones extrayendo algoritmos de recorrido voluminosos y colocándolos en clases independientes.

Principio de abierto/cerrado. Puedes implementar nuevos tipos de colecciones e iteradores y pasarlos al código existente sin descomponer nada.

Puedes recorrer la misma colección en paralelo porque cada objeto iterador contiene su propio estado de iteración.

Por la misma razón, puedes retrasar una iteración y continuar cuando sea necesario.

0.0.2. Desventajas del patrón de diseño iterator

Aplicar el patrón puede resultar excesivo si tu aplicación funciona únicamente con colecciones sencillas.

Utilizar un iterador puede ser menos eficiente que recorrer directamente los elementos de algunas colecciones especializadas.

Ejemplo en codigo del patron de diseño Iterator

```java
package refactoring_guru.iterator.example.iterators;

import refactoring_guru.iterator.example.profile.Profile;

public interface ProfileIterator {
    boolean hasNext();

    Profile getNext();

    void reset();
}
```

```java
package refactoring_guru.iterator.example.iterators;

import refactoring_guru.iterator.example.profile.Profile;
import refactoring_guru.iterator.example.social_networks.Facebook;

import java.util.ArrayList;
import java.util.List;

public class FacebookIterator implements ProfileIterator {
    private Facebook facebook;
    private String type;
    private String email;
    private int currentPosition = 0;
    private List<String> emails = new ArrayList<>();
    private List<Profile> profiles = new ArrayList<>();

    public FacebookIterator(Facebook facebook, String type, String email) {
        this.facebook = facebook;
        this.type = type;
        this.email = email;
    }

    private void lazyLoad() {
        if (emails.size() == 0) {
            List<String> profiles = facebook.requestProfileFriendsFromFacebook(this.emai
            for (String profile : profiles) {
                this.emails.add(profile);
                this.profiles.add(null);
            }
        }
    }
```

```
    @Override
    public boolean hasNext() {
        lazyLoad();
        return currentPosition < emails.size();
    }

    @Override
    public Profile getNext() {
        if (!hasNext()) {
            return null;
        }

        String friendEmail = emails.get(currentPosition);
        Profile friendProfile = profiles.get(currentPosition);
        if (friendProfile == null) {
            friendProfile = facebook.requestProfileFromFacebook(friendEmail);
            profiles.set(currentPosition, friendProfile);
        }
        currentPosition++;
        return friendProfile;
    }

    @Override
    public void reset() {
        currentPosition = 0;
    }
}
```

```java
package refactoring_guru.iterator.example.iterators;

import refactoring_guru.iterator.example.profile.Profile;
import refactoring_guru.iterator.example.social_networks.LinkedIn;

import java.util.ArrayList;
import java.util.List;

public class LinkedInIterator implements ProfileIterator {
    private LinkedIn linkedIn;
    private String type;
    private String email;
    private int currentPosition = 0;
    private List<String> emails = new ArrayList<>();
    private List<Profile> contacts = new ArrayList<>();

    public LinkedInIterator(LinkedIn linkedIn, String type, String email) {
        this.linkedIn = linkedIn;
        this.type = type;
        this.email = email;
    }

    private void lazyLoad() {
        if (emails.size() == 0) {
            List<String> profiles = linkedIn.requestRelatedContactsFromLinkedInAPI(this.
            for (String profile : profiles) {
                this.emails.add(profile);
                this.contacts.add(null);
            }
        }
    }
```

```
    @Override
    public boolean hasNext() {
        lazyLoad();
        return currentPosition < emails.size();
    }

    @Override
    public Profile getNext() {
        if (!hasNext()) {
            return null;
        }

        String friendEmail = emails.get(currentPosition);
        Profile friendContact = contacts.get(currentPosition);
        if (friendContact == null) {
            friendContact = linkedIn.requestContactInfoFromLinkedInAPI(friendEmail);
            contacts.set(currentPosition, friendContact);
        }
        currentPosition++;
        return friendContact;
    }

    @Override
    public void reset() {
        currentPosition = 0;
    }
}
```

```
package refactoring_guru.iterator.example.social_networks;

import refactoring_guru.iterator.example.iterators.ProfileIterator;

public interface SocialNetwork {
    ProfileIterator createFriendsIterator(String profileEmail);

    ProfileIterator createCoworkersIterator(String profileEmail);
}
```

```java
package refactoring_guru.iterator.example.social_networks;

import refactoring_guru.iterator.example.iterators.FacebookIterator;
import refactoring_guru.iterator.example.iterators.ProfileIterator;
import refactoring_guru.iterator.example.profile.Profile;

import java.util.ArrayList;
import java.util.List;

public class Facebook implements SocialNetwork {
    private List<Profile> profiles;

    public Facebook(List<Profile> cache) {
        if (cache != null) {
            this.profiles = cache;
        } else {
            this.profiles = new ArrayList<>();
        }
    }

    public Profile requestProfileFromFacebook(String profileEmail) {
        // Here would be a POST request to one of the Facebook API endpoints.
        // Instead, we emulates long network connection, which you would expect
        // in the real life...
        simulateNetworkLatency();
        System.out.println("Facebook: Loading profile '" + profileEmail + "' over the ne

        // ...and return test data.
        return findProfile(profileEmail);
    }
```

```
    public List<String> requestProfileFriendsFromFacebook(String profileEmail, String co
        // Here would be a POST request to one of the Facebook API endpoints.
        // Instead, we emulates long network connection, which you would expect
        // in the real life...
        simulateNetworkLatency();
        System.out.println("Facebook: Loading '" + contactType + "' list of '" + profile

        // ...and return test data.
        Profile profile = findProfile(profileEmail);
        if (profile != null) {
            return profile.getContacts(contactType);
        }
        return null;
    }

    private Profile findProfile(String profileEmail) {
        for (Profile profile : profiles) {
            if (profile.getEmail().equals(profileEmail)) {
                return profile;
            }
        }
        return null;
    }

    private void simulateNetworkLatency() {
        try {
            Thread.sleep(2500);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
    }
```

```
    @Override
    public ProfileIterator createFriendsIterator(String profileEmail) {
        return new FacebookIterator(this, "friends", profileEmail);
    }

    @Override
    public ProfileIterator createCoworkersIterator(String profileEmail) {
        return new FacebookIterator(this, "coworkers", profileEmail);
    }

}
```

```java
package refactoring_guru.iterator.example.social_networks;

import refactoring_guru.iterator.example.iterators.LinkedInIterator;
import refactoring_guru.iterator.example.iterators.ProfileIterator;
import refactoring_guru.iterator.example.profile.Profile;

import java.util.ArrayList;
import java.util.List;

public class LinkedIn implements SocialNetwork {
    private List<Profile> contacts;

    public LinkedIn(List<Profile> cache) {
        if (cache != null) {
            this.contacts = cache;
        } else {
            this.contacts = new ArrayList<>();
        }
    }

    public Profile requestContactInfoFromLinkedInAPI(String profileEmail) {
        // Here would be a POST request to one of the LinkedIn API endpoints.
        // Instead, we emulates long network connection, which you would expect
        // in the real life...
        simulateNetworkLatency();
        System.out.println("LinkedIn: Loading profile '" + profileEmail + "' over the ne

        // ...and return test data.
        return findContact(profileEmail);
    }
```

```
    public List<String> requestRelatedContactsFromLinkedInAPI(String profileEmail, Strin
        // Here would be a POST request to one of the LinkedIn API endpoints.
        // Instead, we emulates long network connection, which you would expect
        // in the real life.
        simulateNetworkLatency();
        System.out.println("LinkedIn: Loading '" + contactType + "' list of '" + profile

        // ...and return test data.
        Profile profile = findContact(profileEmail);
        if (profile != null) {
            return profile.getContacts(contactType);
        }
        return null;
    }

    private Profile findContact(String profileEmail) {
        for (Profile profile : contacts) {
            if (profile.getEmail().equals(profileEmail)) {
                return profile;
            }
        }
        return null;
    }
```

```
    private void simulateNetworkLatency() {
        try {
            Thread.sleep(2500);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
    }

    @Override
    public ProfileIterator createFriendsIterator(String profileEmail) {
        return new LinkedInIterator(this, "friends", profileEmail);
    }

    @Override
    public ProfileIterator createCoworkersIterator(String profileEmail) {
        return new LinkedInIterator(this, "coworkers", profileEmail);
    }
}
```

```
package refactoring_guru.iterator.example.profile;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Profile {
    private String name;
    private String email;
    private Map<String, List<String>> contacts = new HashMap<>();

    public Profile(String email, String name, String... contacts) {
        this.email = email;
        this.name = name;

        // Parse contact list from a set of "friend:email@gmail.com" pairs.
        for (String contact : contacts) {
            String[] parts = contact.split(":");
            String contactType = "friend", contactEmail;
            if (parts.length == 1) {
                contactEmail = parts[0];
            }
            else {
                contactType = parts[0];
                contactEmail = parts[1];
            }
            if (!this.contacts.containsKey(contactType)) {
                this.contacts.put(contactType, new ArrayList<>());
            }
            this.contacts.get(contactType).add(contactEmail);
        }
    }
```

```
    public String getName() {
        return name;
    }

    public List<String> getContacts(String contactType) {
        if (!this.contacts.containsKey(contactType)) {
            this.contacts.put(contactType, new ArrayList<>());
        }
        return contacts.get(contactType);
    }
}
```

```
package refactoring_guru.iterator.example.spammer;

import refactoring_guru.iterator.example.iterators.ProfileIterator;
import refactoring_guru.iterator.example.profile.Profile;
import refactoring_guru.iterator.example.social_networks.SocialNetwork;

public class SocialSpammer {
    public SocialNetwork network;
    public ProfileIterator iterator;

    public SocialSpammer(SocialNetwork network) {
        this.network = network;
    }

    public void sendSpamToFriends(String profileEmail, String message) {
        System.out.println("\nIterating over friends...\n");
        iterator = network.createFriendsIterator(profileEmail);
        while (iterator.hasNext()) {
            Profile profile = iterator.getNext();
            sendMessage(profile.getEmail(), message);
        }
    }

    public void sendSpamToCoworkers(String profileEmail, String message) {
        System.out.println("\nIterating over coworkers...\n");
        iterator = network.createCoworkersIterator(profileEmail);
        while (iterator.hasNext()) {
            Profile profile = iterator.getNext();
            sendMessage(profile.getEmail(), message);
        }
    }
```

```java
package refactoring_guru.iterator.example;

import refactoring_guru.iterator.example.profile.Profile;
import refactoring_guru.iterator.example.social_networks.Facebook;
import refactoring_guru.iterator.example.social_networks.LinkedIn;
import refactoring_guru.iterator.example.social_networks.SocialNetwork;
import refactoring_guru.iterator.example.spammer.SocialSpammer;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

/**
 * Demo class. Everything comes together here.
 */
public class Demo {
    public static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        System.out.println("Please specify social network to target spam tool (default:F
        System.out.println("1. Facebook");
        System.out.println("2. LinkedIn");
        String choice = scanner.nextLine();

        SocialNetwork network;
        if (choice.equals("2")) {
            network = new LinkedIn(createTestProfiles());
        }
        else {
            network = new Facebook(createTestProfiles());
        }
```

```
        SocialSpammer spammer = new SocialSpammer(network);
        spammer.sendSpamToFriends("anna.smith@bing.com",
                "Hey! This is Anna's friend Josh. Can you do me a favor and like this po
        spammer.sendSpamToCoworkers("anna.smith@bing.com",
                "Hey! This is Anna's boss Jason. Anna told me you would be interested in
    }

    public static List<Profile> createTestProfiles() {
        List<Profile> data = new ArrayList<Profile>();
        data.add(new Profile("anna.smith@bing.com", "Anna Smith", "friends:mad_max@ya.co
        data.add(new Profile("mad_max@ya.com", "Maximilian", "friends:anna.smith@bing.co
        data.add(new Profile("bill@microsoft.eu", "Billie", "coworkers:avanger@ukr.net")
        data.add(new Profile("avanger@ukr.net", "John Day", "coworkers:bill@microsoft.eu
        data.add(new Profile("sam@amazon.com", "Sam Kitting", "coworkers:anna.smith@bing
        data.add(new Profile("catwoman@yahoo.com", "Liza", "friends:anna.smith@bing.com"
        return data;
    }
}
```

Referencias

- Iterator en Java. (n.d.). Refactoring.guru. Retrieved February 9, 2022, from https://refactoring.guru/es/design-patterns/iterator/java/example
- Strategy. (n.d.). Refactoring.guru. Retrieved January 27, 2022, from https://refactoring.guru/es/design-patterns/strategy
- Nesteruk, D. (2018). Iterator. In Design Patterns in Modern C++ (pp. 205–215). Apress.