

服务定位器模式

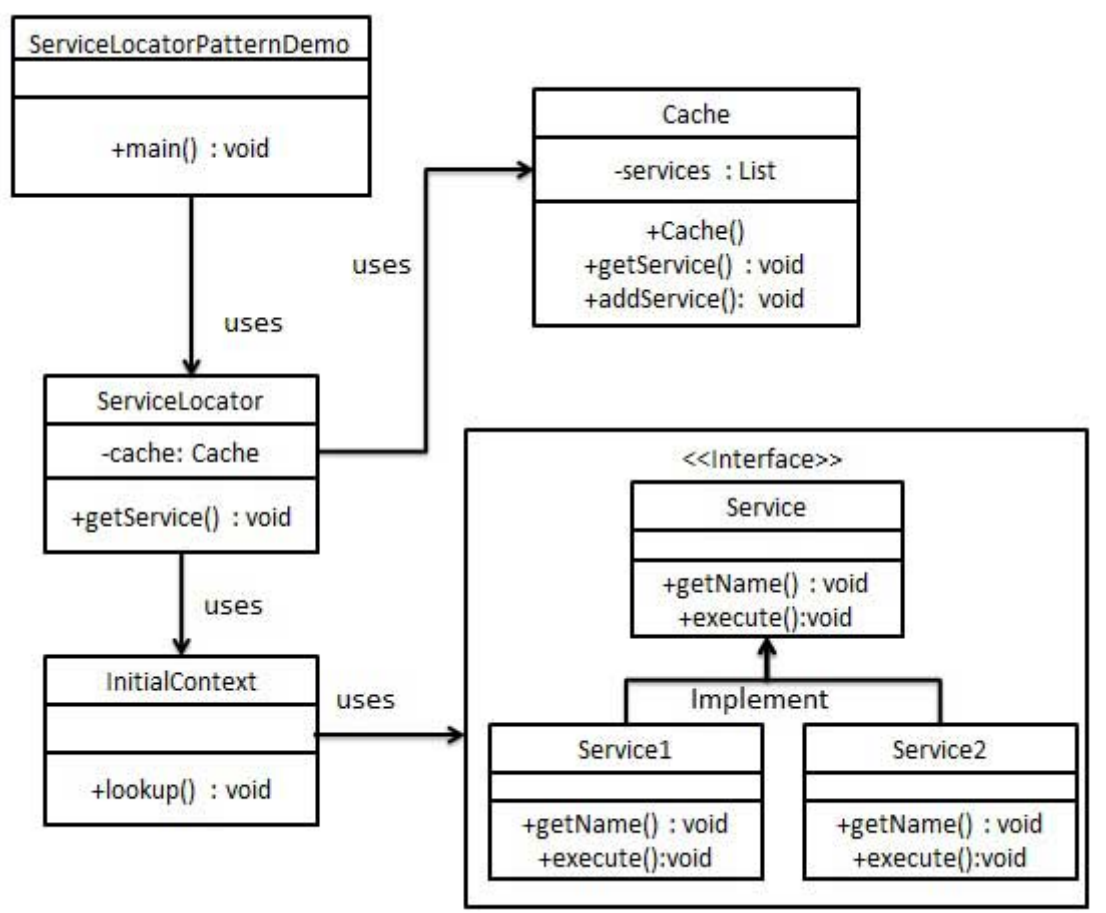
服务定位器模式 (Service Locator Pattern) 用在我们想使用 JNDI 查询定位各种服务的时候。考虑到为某个服务查找 JNDI 的代价很高，服务定位器模式充分利用了缓存技术。在首次请求某个服务时，服务定位器在 JNDI 中查找服务，并缓存该服务对象。当再次请求相同的服务时，服务定位器会在它的缓存中查找，这样可以在很大程度上提高应用程序的性能。以下是这种设计模式的实体。

- **服务 (Service)** - 实际处理请求的服务。对这种服务的引用可以在 JNDI 服务器中查找到。
- **Context / 初始的 Context** - JNDI Context 带有对要查找的服务的引用。
- **服务定位器 (Service Locator)** - 服务定位器是通过 JNDI 查找和缓存服务来获取服务的单点接触。
- **缓存 (Cache)** - 缓存存储服务的引用，以便复用它们。
- **客户端 (Client)** - Client 是通过 ServiceLocator 调用服务的对象。

实现

我们将创建 *ServiceLocator*、*InitialContext*、*Cache*、*Service* 作为表示实体的各种对象。*Service1* 和 *Service2* 表示实体服务。

ServiceLocatorPatternDemo，我们的演示类在这里是作为一个客户端，将使用 *ServiceLocator* 来演示服务定位器设计模式。



步骤 1

创建服务接口 Service。

Service.java

```
public interface Service {  
    public String getName();  
    public void execute();  
}
```

步骤 2

创建实体服务。

Service1.java

```
public class Service1 implements Service {  
    public void execute(){  
        System.out.println("Executing Service1");  
    }  
    @Override  
    public String getName() {  
        return "Service1";  
    }  
}
```

Service2.java

```
public class Service2 implements Service {  
    public void execute(){  
        System.out.println("Executing Service2");  
    }  
    @Override  
    public String getName() {  
        return "Service2";  
    }  
}
```

步骤 3

为 JNDI 查询创建 InitialContext。

InitialContext.java

```
public class InitialContext {  
    public Object lookup(String jndiName){  
        if(jndiName.equalsIgnoreCase("SERVICE1")){  
            System.out.println("Looking up and creating a new Service1 object");  
            return new Service1();  
        }else if (jndiName.equalsIgnoreCase("SERVICE2")){  
            System.out.println("Looking up and creating a new Service2 object");  
            return new Service2();  
        }  
        return null;  
    }  
}
```

```
}  
}
```

步骤 4

创建缓存 Cache。

Cache.java

```
import java.util.ArrayList;  
import java.util.List;  
public class Cache {  
    private List<Service> services;  
    public Cache(){  
        services = new ArrayList<Service>();  
    }  
    public Service getService(String serviceName){  
        for (Service service : services) {  
            if(service.getName().equalsIgnoreCase(serviceName)){  
                System.out.println("Returning cached "+serviceName+" object");  
                return service;  
            }  
        }  
        return null;  
    }  
    public void addService(Service newService){  
        boolean exists = false;  
        for (Service service : services) {  
            if(service.getName().equalsIgnoreCase(newService.getName())){  
                exists = true;  
            }  
        }  
        if(!exists){  
            services.add(newService);  
        }  
    }  
}
```

步骤 5

创建服务定位器。

ServiceLocator.java

```
public class ServiceLocator {  
    private static Cache cache;  
    static {  
        cache = new Cache();  
    }  
    public static Service getService(String jndiName){  
        Service service = cache.getService(jndiName);  
        if(service != null){  
            return service;  
        }  
    }  
}
```

```
InitialContext context = new InitialContext();
Service service1 = (Service)context.lookup(jndiName);
cache.addService(service1);
return service1;
}
```

步骤 6

使用 *ServiceLocator* 来演示服务定位器设计模式。

ServiceLocatorPatternDemo.java

```
public class ServiceLocatorPatternDemo {
    public static void main(String[] args) {
        Service service = ServiceLocator.getService("Service1");
        service.execute();
        service = ServiceLocator.getService("Service2");
        service.execute();
        service = ServiceLocator.getService("Service1");
        service.execute();
        service = ServiceLocator.getService("Service2");
        service.execute();
    }
}
```

步骤 7

执行程序，输出结果：

```
Looking up and creating a new Service1 object
Executing Service1
Looking up and creating a new Service2 object
Executing Service2
Returning cached Service1 object
Executing Service1
Returning cached Service2 object
Executing Service2
```

← 拦截过滤器模式

传输对象模式 →



1 篇笔记



写笔记



步骤1 和步骤2，主要表现接口之间的多态性，指定行为方式。

步骤3：也就是工厂模式的应用，通过类名来确定要实例化的对象。

步骤4：用的是下面的传输对象模式。对实体类集合进行操作，主要是在集合中 获取/添加 实体类对象。

步骤5定位器：使用步骤3来创建实例，使用步骤4来添加到集合，或者从集合中获取。（缓存中没有才会创建）

步骤6：调用步骤5得到实体类，并执行实体类的方法

慕言 7个月前 [08-13]