프록시 패턴과 프록시 서버

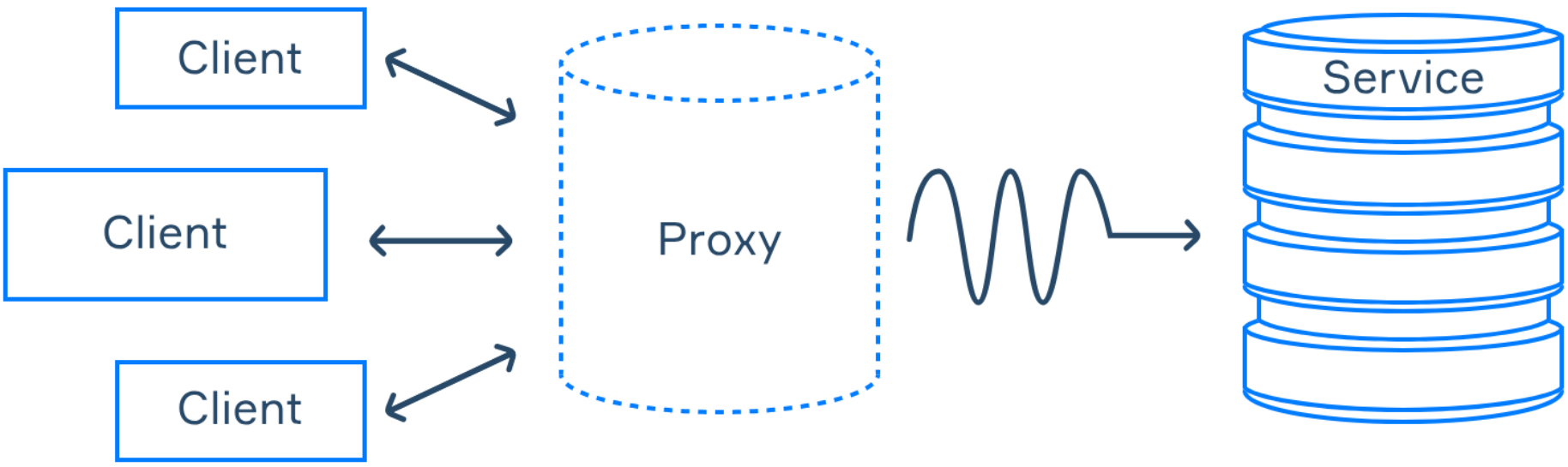
프록시 패턴

개요

구조 패턴

Proxy: 대리, 대리인

프록시 패턴은 대상 객체에 접근하기 전 그 접근에 대한 흐름을 가로채 대상 객체 앞단의 인터페이스 역할을 하는 디자인 패턴입니다.



프록시 패턴 사용이유

- **보안(Security)** : 프록시는 클라이언트가 작업을 수행할 수 있는 권한이 있는지 확인하고 검사 결과가 긍정적인 경우에만 요청을 대상으로 전달한다.
- **캐싱(Caching)** : 프록시가 내부 캐시를 유지하여 데이터가 캐시에 아직 존재하지 않는 경우에만 대상에서 작업이 실행되도록 한다.
- **데이터 유효성 검사(Data validation)** : 프록시가 입력을 대상으로 전달하기 전에 유효성을 검사한다.
- **지연 초기화(Lazy initialization)** : 대상의 생성 비용이 비싸다면 프록시는 그것을 필요로 할때까지 연기할 수 있다.
- **로깅(Logging)** : 프록시는 메소드 호출과 상대 매개 변수를 인터셉트하고 이를 기록한다.
- **원격 객체(Remote objects)** : 프록시는 원격 위치에 있는 객체를 가져와서 로컬처럼 보이게 할 수 있다.

프록시 패턴 장단점

장점:

1. **보안 강화:**
   - 보호 프록시를 사용하면 클라이언트가 직접 접근하지 않고 프록시를 통해 간접적으로 리소스에 접근하므로, 보안을 강화할 수 있습니다. 프록시는 접근 권한을 확인하고 부여할 수 있습니다.
2. **성능 최적화:**
   - 캐싱 프록시를 사용하여 이전에 요청된 결과를 저장하고 재사용함으로써 성능을 최적화할 수 있습니다. 또한, 스마트 프록시를 사용하여 필요한 경우에만 리소스를 로딩하거나 추가 로직을 수행할 수 있습니다.
3. **코드 분리:**
   - 프록시 패턴을 사용하면 실제 객체와 프록시 객체 간에 인터페이스를 정의함으로써 코드를 분리할 수 있습니다. 이로써 유지보수성이 향상되고 확장성이 높아집니다.
4. **리소스 접근 제어:**
   - 프록시를 사용하여 리소스에 대한 접근을 제어하면, 부하 분산, 로그 기록, 권한 부여 등과 같은 다양한 관리 작업을 수행할 수 있습니다.

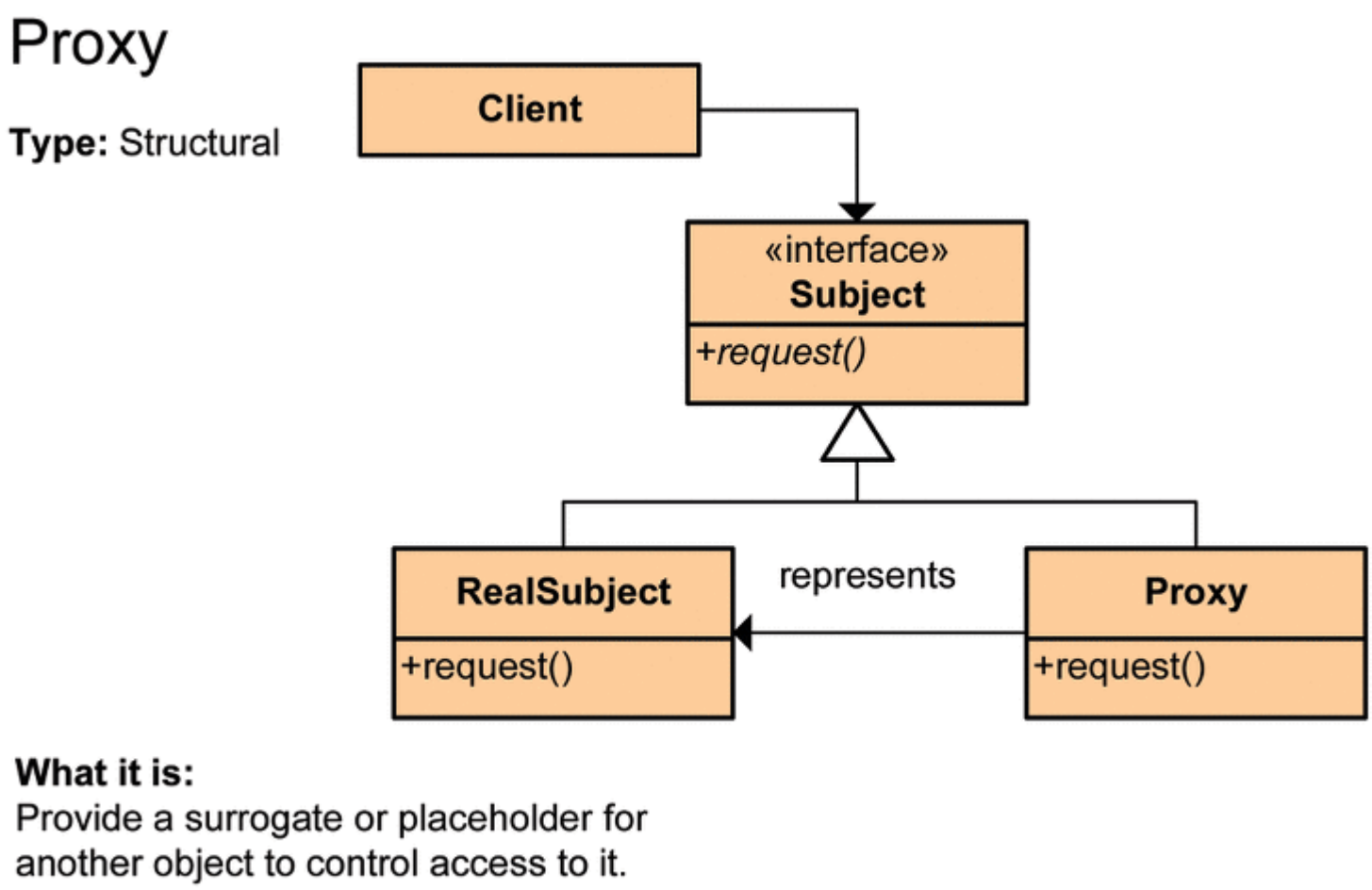
단점:

1. **복잡성 증가:**
    - 프록시 패턴을 과도하게 사용하면 코드가 복잡해질 수 있습니다. 많은 수의 프록시 클래스가 추가될 경우 유지보수가 어려워질 수 있습니다.
2. **성능 저하:**
    - 너무 많은 프록시를 사용하면 성능에 오히려 부정적인 영향을 미칠 수 있습니다. 특히, 스마트 프록시나 로깅과 같이 추가 작업이 필요한 경우에는 성능 저하가 발생할 수 있습니다.
3. **디버깅 어려움:**
    - 디버깅 시에 프록시 패턴은 객체 간의 중간 계층을 도입하기 때문에 디버깅이 어려울 수 있습니다. 프록시와 실제 객체 간의 통신 문제를 해결하기 위해서는 추가적인 노력이 필요합니다.
4. **패턴 이해 어려움:**
    - 처음에는 패턴 자체의 이해가 어려울 수 있습니다. 특히, 각 프록시 종류마다 역할이 다르기 때문에 어떤 종류의 프록시를 사용해야 하는지 결정하는 것이 중요합니다.

프록시 패턴 예시

- **가상 프록시**
    - 이미지 로딩 최적화**:** 웹 페이지에서 많은 양의 이미지를 미리 모두 로딩하는 것은 효율적이지 않습니다. 대신, 가상 프록시를 사용하여 실제로 이미지가 필요한 시점에만 로딩할 수 있습니다.
- **원격 프록시**
    - 파일 시스템 접근 제어**:** 특정 사용자에게만 파일 시스템에 접근 권한을 부여하는 보호 프록시를 사용할 수 있습니다.
- **보호 프록시**
    - 웹 페이지 캐싱**:** 웹 페이지의 내용을 캐싱하여 동일한 페이지에 대한 요청이 들어올 때 캐시된 내용을 반환할 수 있습니다.

프록시 패턴 구현




```
// 공통 인터페이스
interface Image {
    void display();
```

```
}

// 실제 이미지 로딩을 담당하는 클래스
class RealImage implements Image {
    private String filename;

    public RealImage(String filename) {
        this.filename = filename;
        loadImageFromDisk();
    }

    private void loadImageFromDisk() {
        System.out.println("Loading image from disk: " + filename);
    }

    public void display() {
        System.out.println("Displaying image: " + filename);
    }
}

// 가상 프록시 클래스
class ProxyImage implements Image {
    private RealImage realImage;
    private String filename;

    public ProxyImage(String filename) {
        this.filename = filename;
    }

    public void display() {
        if (realImage == null) {
            realImage = new RealImage(filename);
        }
        realImage.display();
    }
}
```

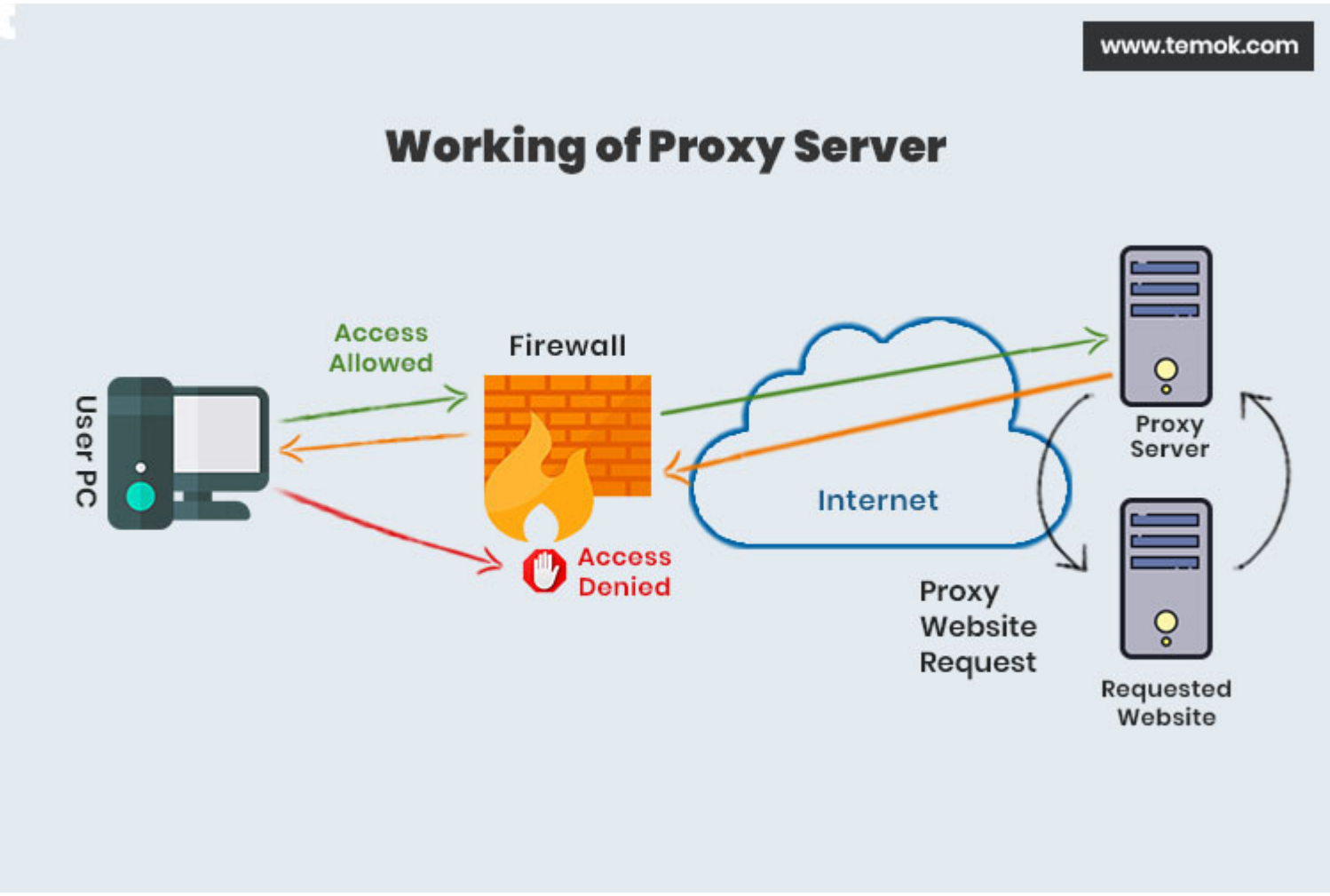
프록시 서버

개요

프록시 서버(Proxy Server)는 클라이언트와 다른 서버 간에 중개자 역할을 하는 서버입니다. 이는 주로 네트워크 서비스에 대한 중계 및 보안 목적으로 사용됩니다. 프록시 서버는 클라이언트의 요청을 받아서 해당 요청을 다른 서버로 전달하고, 서버의 응답을 클라이언트에게 반환합니다.

프록시 서버 주요기능

1. **캐싱 (Caching):**
    - 프록시 서버는 이전에 받았던 요청에 대한 응답을 캐싱하여, 동일한 요청이 다시 들어왔을 때 서버에 요청을 전달하지 않고 캐시된 응답을 반환할 수 있습니다. 이는 네트워크 대역폭을 절약하고 응답 속도를 향상시킵니다.
2. **보안 (Security):**
    - 프록시 서버는 클라이언트와 서버 간의 통신을 중개하므로, 클라이언트의 실제 IP 주소를 서버에게 노출시키지 않고 자체 IP 주소를 노출시킬 수 있습니다. 이는 클라이언트의 익명성을 보장하고 보안을 강화할 수 있습니다.
3. **접근 제어 (Access Control):**
    - 특정 웹 사이트나 서비스에 대한 접근을 제어하고 차단하는데 사용됩니다. 기업이나 학교에서는 프록시 서버를 통해 특정 웹 사이트에 대한 접근을 제한하는 등의 정책을 설정할 수 있습니다.
4. **로드 밸런싱 (Load Balancing):**
    - 여러 서버에 대한 요청을 분산하여 부하를 고르게 분배함으로써 서버의 성능을 최적화합니다. 클라이언트는 프록시 서버에게 요청을 하고, 프록시 서버는 적절한 백엔드 서버로 요청을 전달합니다.



프록시 서버 예시

1. **웹 프록시 서버:**
    - 사용자들이 웹 브라우저를 통해 인터넷에 접속할 때, 웹 프록시 서버를 거쳐서 웹 페이지에 접근하는 경우가 많습니다. 이를 통해 웹 캐싱, 보안, 로드 밸런싱 등의 기능을 수행할 수 있습니다.
2. **콘텐츠 필터링 프록시:**
    - 기업이나 학교에서는 인터넷 사용에 대한 정책을 시행하기 위해 콘텐츠 필터링 프록시를 사용할 수 있습니다. 이를 통해 특정 사이트의 접근을 차단하거나 특정 콘텐츠를 제한할 수 있습니다.
3. **로드 밸런서 프록시:**
    - 여러 웹 서버에 대한 부하를 분산하기 위해 로드 밸런서 프록시를 사용할 수 있습니다. 프록시는 클라이언트의 요청을 각 서버로 전달하고, 서버의 응답을 클라이언트에게 반환합니다.

예상 질문:

1. 프록시 패턴이 무엇이며 언제 사용하는 것이 좋다고 생각하시나요?
2. 프록시 패턴을 사용하여 어떤 문제를 해결할 수 있나요?
3. 프록시 패턴을 사용한 실제 프로젝트 경험이나 예시를 제공할 수 있나요?
4. 프록시 서버는 무엇이며 언제 사용하는 것이 좋은가요?

출처:

https://youtu.be/4pnOhP-nfmM?si=Q6lyJEElN8p2CgVE

https://limkydev.tistory.com/79

https://inpa.tistory.com/entry/GOF-💠-프록시Proxy-패턴-제대로-배워보자

https://surfshark.com/ko/blog/proxy-server

https://velog.io/@jangwonyoon/Proxy-Server프록시-서버란

https://refactoring.guru/design-patterns/proxy