캐싱(Caching)은 데이터를 미리 저장해두고 동일한 데이터 요청 시 저장된 데이터를 활용하여 서버와의 통신을 최소화하고 성능을 향상시키는 기술이다. 이를 통해 웹 페이지 로딩 속도를 개선하고 서버 부하를 줄이는 효과를 얻는다.

브라우저 캐싱

웹 사이트 방문 시 브라우저가 서버에서 받은 리소스(이미지, CSS, JavaScript 파일 등)를 사용자의 로컬 저장소에 임시로 저장해 두는 방식이다. 이 캐싱을 통해 동일한 웹 페이지나 리소스를 다시 요청할 때 서버에 재요청하지 않고 브라우저에 저장된 데이터를 사용하여 페이지 로딩 속도를 빠르게 할 수 있다. 이는 웹 성능을 크게 개선하고 서버 부하를 줄이는 효과가 있다.

브라우저 캐싱의 동작 방식

1. 초기 요청 : 사용자가 웹 페이지를 처음 방문할 때 브라우저는 서버에 요청을 보내고 필요한 리소스(HTML, CSS, JavaScript, 이미지 등)를 다운로드한다.
2. 캐시 저장 : 서버에서 받은 리소스는 브라우저의 로컬 캐시에 저장된다. 이 캐시는 사용자의 하드 드라이브나 메모리에 저장되며, 다음 요청에 재사용할 수 있다.
3. 재요청 시 캐시 사용 : 동일한 리소스를 다시 요청할 경우 브라우저는 먼저 로컬 캐시를 확인한다. 캐시된 리소스가 유효하다면 서버에 다시 요청하지 않고 캐시에서 리소스를 불러온다. 이를 캐시 히트(cache hit)라고 하며 서버에 다시 요청할 필요가 없으므로 네트워크 트래픽을 줄이고 로딩 속도를 개선한다.
4. 캐시 미스: 만약 캐시된 리소스가 없거나 유효하지 않다면 브라우저는 서버에 재요청을 보내 최신 리소스를 받아온다 이를 캐시 미스(cache miss)라고 하며 새로운 리소스는 다시 브라우저 캐시에 저장된다.

브라우저 캐싱 설정 방법

브라우저 캐싱은 주로 서버에서 설정한 HTTP 헤더를 통해 관리되며, 중요한 헤더로는 `Cache-Control`, `Expires`, `ETag` 등이 있다. 각각의 헤더는 캐시의 유효 기간이나 업데이트 조건 등을 제어한다.

1. `Cache-Control` 헤더
   `Cache-Control`은 캐시 정책을 정의하는 주요 HTTP 헤더이다. 이를 통해 브라우저가 리소스를 얼마나 오래 캐시해야 하는지 캐시된 데이터를 사용할지 여부 등을 결정할 수 있다.
   1. `max-age` : 캐시의 유효 기간을 초 단위로 지정하다. `Cache-Control: max-age=3600`은 브라우저가 리소스를 1시간 동안(3600초) 캐시해야 한다는 의미이다.

    2. `no-cache` : 캐시된 리소스를 사용할 수 있지만 매번 서버에 확인 요청을 보내 최신 버전인지 체크한다.
    3. `no-store` : 리소스를 캐시하지 않고 매번 서버에서 데이터를 받아온다.
    4. `public` : 리소스가 모든 캐시(브라우저뿐만 아니라 중간 서버도 포함)에서 캐시될 수 있음을 나타낸다.
    5. `private` : 리소스가 브라우저 캐시에만 저장되며 중간 서버에서는 캐시되지 않음을 나타낸다.
2. `Expires` 헤더
   `Expires`는 특정 시점을 기준으로 캐시된 리소스가 만료되는 시간을 설정하는 헤더이다. `Expires: Wed, 21 Oct 2023 07:28:00 GMT`는 해당 시간 이후에는 브라우저가 캐시를 사용하지 않고 서버에서 다시 리소스를 요청하게 한다. 그러나 현재는 `Cache-Control`의 `max-age`가 더 많이 사용되며 `Expires`는 구식 방식이다.
3. `ETag` 헤더
   `ETag`는 서버가 리소스의 고유 식별자를 제공하여 브라우저가 캐시된 리소스가 여전히 최신 상태인지 확인하는 데 사용된다. 브라우저가 서버에 캐시된 리소스가 여전히 유효한지 요청할 때 서버는 `ETag` 값을 비교하여 동일한 경우 캐시된 리소스를 그대로 사용하고 변경된 경우 새로운 리소스를 제공할 수 있다.
4. `Last-Modified` 헤더
   `Last-Modified`는 리소스가 마지막으로 수정된 시간을 서버가 브라우저에 알려주는 헤더이다. 브라우저는 이 값을 기억하고 서버에 재요청할 때 `If-Modified-Since` 헤더로 확인 요청을 보낸다. 만약 리소스가 변경되지 않았다면 서버는 304 Not Modified 응답을 보내고 브라우저는 캐시된 리소스를 계속 사용한다.

브라우저 캐싱은 웹 성능을 개선하고 사용자 경험을 향상시키는 중요한 기술이다. 적절한 HTTP 헤더 설정을 통해 캐싱 정책을 잘 관리하면 웹 페이지의 응답 속도를 크게 높이고 서버 부하를 줄일 수 있다.

CDN 캐싱(Content Delivery Network Caching)

콘텐츠 전달 네트워크(CDN)를 활용해 데이터를 사용자에게 더 가까운 위치에 캐시하는 방식이다. CDN은 전 세계 여러 지역에 분산된 서버 네트워크로 구성되며 사용자와 가까운 서버(에지 서버)에 콘텐츠를 저장해 두었다가 빠르게 제공함으로써 성능을 최적화한다.

CDN 캐싱의 동작 방식

1. 콘텐츠 요청 : 사용자가 웹 사이트에 접속하면, 브라우저는 필요한 리소스(이미지, CSS, JavaScript 파일 등)를 요청합니다.

2. 에지 서버 확인 : 사용자의 요청은 CDN 네트워크 내에서 가장 가까운 서버 에지 서버로 전달된다. 이 서버는 요청된 콘텐츠가 캐시되어 있는지 확인한다.
3. 캐시 히트 : 에지 서버에 해당 콘텐츠가 이미 캐시되어 있는 경우 캐시 히트(cache hit)가 발생하며 서버는 빠르게 캐시된 콘텐츠를 사용자에게 제공한다. 이때 원래 웹 서버까지 가지 않고 가까운 위치에서 콘텐츠가 제공되므로 대기 시간(Latency)이 줄어든다.
4. 캐시 미스 : 만약 에지 서버에 콘텐츠가 없는 경우 캐시 미스(cache miss)가 발생한다. 에지 서버는 원본 서버로부터 콘텐츠를 받아오고 이를 사용자에게 전달하면서 동시에 자신의 캐시에 저장해 둔다. 이후 같은 콘텐츠를 요청하는 사용자에게는 에지 서버가 캐시된 콘텐츠를 제공한다.

CDN 캐싱의 장점

1. 속도 향상: 사용자와 가까운 에지 서버에서 데이터를 제공하기 때문에 웹 사이트 로딩 속도가 크게 향상된다.
2. 서버 부하 감소 : 원본 서버로의 요청이 줄어들어 서버에 가해지는 부하가 감소한다. 이는 대규모 트래픽을 처리할 때 서버 다운이나 속도 저하를 방지하는 데 유리하다.
3. 전 세계적 접근성 향상 : CDN은 다양한 지역에 분산된 서버를 통해 글로벌 사용자에게도 빠른 서비스 제공이 가능하다. 지역 간의 네트워크 지연을 최소화한다.
4. 대역폭 비용 절감 : CDN을 활용하면 서버로 가는 직접적인 트래픽이 줄어들어 대역폭 비용이 절감될 수 있다.
5. 보안 강화 : 많은 CDN 제공업체는 DDoS 방어, SSL 인증서 지원 등 다양한 보안 기능을 제공하여 원본 서버를 보호하고 안전한 콘텐츠 전송을 보장한다.

CDN 캐싱 설정

CDN 캐싱은 서버의 설정이나 콘텐츠 제공 방법에 따라 다양한 방식으로 구현된다.

1. TTL(Time To Live) : 콘텐츠의 유효 기간을 설정하는 방식이다. CDN에 캐시된 콘텐츠는 1시간 동안 유효하도록 설정할 수 있다. 유효 기간이 지나면 CDN 서버는 본 서버로부터 새로운 콘텐츠를 요청한다.
2. `Cache-Control` 헤더 : 이 헤더를 사용하여 CDN과 브라우저에 캐시 정책을 정의할 수 있다. 캐시의 최대 유효 시간을 지정하거나 캐시하지 말아야 할 리소스를 설정할 수 있다.

CDN 캐싱은 사용자 경험을 개선하고 서버의 성능을 높이는 중요한 기술이다. 특히 전 세계적으로 분산된 트래픽을 효율적으로 처리해야 하는 웹 사이트나 애플리케이션에서 필수적인 역할을 한다.

`Cache-Control`은 HTTP 헤더 중 하나로 웹 브라우저나 CDN 같은 캐시 서버가 어떻게 리소스를 캐싱해야 하는지에 대한 지침을 제공한다. 이를 통해 서버가 웹 페이지 또는 리소스의 캐싱 동작을 제어할 수 있으며 클라이언트(브라우저)와 서버 간의 효율적인 데이터 전송을 할 수 있다. `Cache-Control` 헤더는 캐싱 기간, 캐싱 여부, 캐시된 데이터를 재검증하는 방식 등 다양한 캐싱 정책을 정의할 수 있다.

`Cache-Control`의 주요 디렉티브

`Cache-Control` 헤더는 여러 디렉티브를 포함할 수 있으며 각각 캐시의 동작 방식을 제어을 할 수 있다.

1. `max-age=<seconds>`
    1. 설명 : 리소스를 캐시할 최대 기간(초 단위)을 정의한다. 이 기간 동안 브라우저는 서버에 재요청하지 않고 캐시된 데이터를 사용한다.
    2. 예시 : `Cache-Control: max-age=3600`은 1시간 동안(3600초) 캐시된 데이터를 사용하도록 설정한다.
2. `s-maxage=<seconds>`
    1. 설명 : 공유 캐시(CDN, 프록시 서버 등)에서의 캐싱 유효 기간을 지정한다. 브라우저의 `max-age`와는 별도로 공유 캐시 서버의 캐싱 정책을 설정할 수 있다.
    2. 예시 : `Cache-Control: s-maxage=86400`은 CDN이나 프록시 서버가 24시간(86400초) 동안 캐시된 데이터를 사용할 수 있게 한다.
3. `public`
    1. 설명 : 리소스가 공유 캐시(CDN, 프록시 서버 등)과 브라우저 캐시에 저장될 수 있음을 나타낸다. 캐시가 모든 사용자에게 공유될 수 있음을 의미한다.
    2. 예시 : `Cache-Control: public`은 해당 리소스가 공유 캐시와 브라우저 캐시에 저장될 수 있도록 허용한다.
4. `private`
    1. 설명 : 리소스가 브라우저 캐시에만 저장되고 공유 캐시에는 저장되지 않음을 나타낸다. 개인화된 데이터나 사용자별 데이터를 캐시할 때 주로 사용된다.
    2. 예시 : `Cache-Control: private`은 해당 리소스가 브라우저에만 캐시되고 프록시나 CDN 같은 공유 캐시에는 저장되지 않도록 한다.
5. `no-cache`
    1. 설명 : 캐시된 리소스를 사용하기 전에 서버에 재검증을 요청해야 함을 나타낸다. 서버에서 최신 리소스인지 확인한 후 캐시된 리소스를 사용할 수 있다.
    2. 예시 : `Cache-Control: no-cache`는 캐시된 데이터를 사용할 수는 있지만 매번 서버에 확인을 요청해야 한다.
6. `no-store`
    1. 설명 : 리소스를 캐시하지 않도록 설정한다. 브라우저나 중간 캐시 서버에 데이터를 저장하지 않으며 매번 서버에서 리소스를 다시 받아온다. 민감한 정보(예: 개인 정보, 금융 정보)를 다룰 때 사용된다.

    2. 예시 : `Cache-Control: no-store` 는 캐시를 완전히 금지하여 리소스가 브라우저나 중간 서버에 저장되지 않도록 한다.
7. `must-revalidate`
    1. 설명 : 캐시된 리소스의 유효 기간이 끝나면 서버에 재검증을 요청해야 한다. 브라우저가 만료된 캐시를 사용할 수 없도록 한다.
    2. 예시 : `Cache-Control: must-revalidate` 는 캐시된 데이터가 만료된 경우 서버에 재검증을 요청해야 한다.
8. `proxy-revalidate`
    1. 설명 : 프록시 서버가 캐시된 리소스를 재검증하도록 설정한다. `must-revalidate` 와 비슷하지만 이 설정은 브라우저가 아닌 공유 캐시(프록시 서버)에만 적용된다.
    2. 예시 : `Cache-Control: proxy-revalidate` 는 프록시 서버가 캐시된 리소스를 사용할 때 재검증을 요구한다.
9. `no-transform`
    1. 설명 : 중간 캐시 서버나 프록시 서버가 리소스를 변환하지 못하도록 설정한다. 이미지의 포맷을 변경하거나 데이터를 압축하지 않도록 한다.
    2. 예시 : `Cache-Control: no-transform` 은 이미지나 리소스가 중간 서버에 의해 변경되지 않도록 보장한다.
10. `immutable`
11. 설명 : 리소스가 변경되지 않음을 보장한다. 리소스는 절대 변경되지 않으므로 브라우저는 서버에 재검증 요청 없이 항상 캐시된 리소스를 사용할 수 있다. 주로 버전이 명확하게 관리되는 파일(CSS, JS)에서 사용된다.
12. 예시 : `Cache-Control: immutable` 은 브라우저가 리소스를 캐시에 저장하고 절대 서버에 다시 요청하지 않도록 설정한다.
13. `stale-while-revalidate=<seconds>`
    1. 설명 : 캐시된 리소스가 만료되었더라도 새로운 리소스를 받아오는 동안 잠시 만료된 캐시를 사용할 수 있도록 허용한다. 서버에서 데이터를 다시 받는 동안 사용자는 캐시된 데이터를 받아볼 수 있어 로딩 시간을 줄이는 데 도움이 된다.
    2. 예시 : `Cache-Control: stale-while-revalidate=60` 은 리소스가 만료된 후 60초 동안은 만료된 캐시를 사용하며 백그라운드에서 새로운 데이터를 받아온다.
14. `stale-if-error=<seconds>`
    1. 설명 : 서버에서 오류가 발생할 경우 캐시된 리소스가 만료되었더라도 오류 발생 시 캐시된 데이터를 사용할 수 있도록 허용한다. 서버 다운이나 네트워크 장애 시 유용하게 사용된다.
    2. 예시 : `Cache-Control: stale-if-error=300` 은 서버에서 오류가 발생할 경우 5분간은 만료된 캐시를 사용할 수 있게 한다.

`Cache-Control` 의 예시

1. 일반 캐싱

```
Cache-Control: public, max-age=3600
```

이 설정은 리소스를 1시간 동안 캐시하며 브라우저와 CDN 같은 공유 캐시 서버에서도 사용할 수 있도록 허용한다.

2. 민감한 데이터 캐싱 금지

```
Cache-Control: no-store
```

이 설정은 브라우저나 공유 캐시 서버에 절대 리소스를 저장하지 않고 매번 서버에서 새롭게 데이터를 가져오도록 강제한다.

3. 재검증 설정

```
Cache-Control: no-cache, must-revalidate
```

이 설정은 브라우저가 캐시된 리소스를 사용하기 전에 항상 서버에 재검증을 요청하고 만료된 리소스는 반드시 재검증해야 한다는 의미이다.

`Cache-Control` 헤더는 웹 페이지 성능 최적화에서 매우 중요한 역할을 한다. 이를 통해 리소스가 얼마나 오래 어디에서 어떻게 캐시되어야 하는지를 명확하게 정의할 수 있으며 적절한 캐싱 정책을 적용함으로써 웹 성능을 크게 개선하고 서버 부하를 줄일 수 있다. 캐시 정책은 리소스의 성격 업데이트 빈도 민감도에 따라 세밀하게 설정되어야 한다.