Data Fetching

Server Actions 사용법
fetch API와 캐싱 전략

(1) 기존 fetch와 Next.js fetch 비교

구분	기존 fetch()	Next.js fetch()
위치	클라이언트/서버 모두	주로 서버
캐싱 제어	수동 캐시 제어 필요	`cache`, `revalidate` 옵션 제공
성능 최적화	직접 구현 필요	자동 캐시 + 정적 최적화
태그 활용	불가능	`tags`로 정밀 무효화 가능

(2) 렌더링 방식에 따른 특징

방식	설명	실행 위치	주요 특징
Static Generation (SSG)	빌드 시 생성	서버	빠름, 정적
Server-Side Rendering (SSR)	매 요청마다 생성	서버	실시간 반영
ISR (점진적 정적 생성)	일정 주기로 갱신	서버	성능 + 최신성

(3) API 캐싱 전략

1. cache : 'force-cache'

```
await fetch('https://api.example.com/data', {
  cache: 'force-cache',
});
```

2. cache : 'no-store'

```
await fetch('https://api.example.com/data', {
  cache: 'no-store',
});
```

3. revalidate : N

```
await fetch('https://api.example.com/data', {
  next: { revalidate: 60 }, // 60초 간격으로 재검증
});
```

4. tag + revalidateTag()

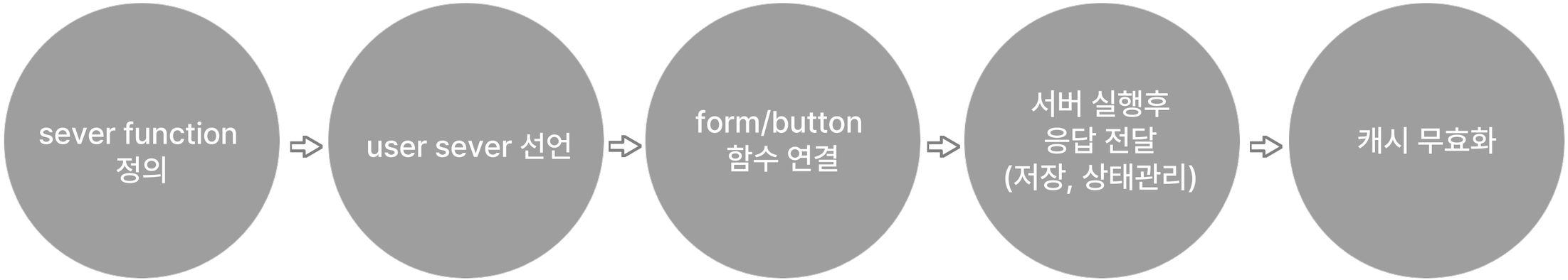
```
// fetch 시
await fetch('https://api.example.com/posts', {
  next: { tags: ['posts'], revalidate: 30 },
});

// 서버 액션 또는 이벤트 후
import { revalidateTag } from 'next/cache';
await revalidateTag('posts');
```

(4) 렌더링 방식에 따른 API 캐싱 전략 정리

상황	전략	설명
항상 최신 데이터	`cache: 'no-store'`	SSR
일정 주기 갱신	`revalidate: 60`	ISR
변경 거의 없음	`force-cache`	SSG
갱신 필요 제어	`tags + revalidateTag()`	동적 무효화

(5) Sever Actions 사용법 순서도



(5) Sever Actions 사용법 - Form 형식

- 액션 호출

```tsx
// actions.ts
'use server';

export async function createPost(formData: FormData) {
  const title = formData.get('title');
  // DB 저장 등 처리
}
```

```tsx
// app/page.tsx (서버 컴포넌트)
import { createPost } from './actions';

export default function Page() {
  return (
    <form action={createPost}>
      <input name="title" />
      <button type="submit">등록</button>
    </form>
  );
}
```

- 상태 관리 / 캐시 무효화

```tsx
// actions.ts
'use server';
import { revalidatePath } from 'next/cache';

export async function createPost(prevState: any, formData: FormData) {
  const title = formData.get('title')?.toString();
  if (!title) return { success: false, message: '제목이 필요합니다' };

  // DB 처리 등
  await revalidatePath('/posts'); // 경로 캐시 무효화
  return { success: true, message: '등록 완료' };
}
```

```tsx
// FormComponent.tsx
'use client';
import { useActionState } from 'react';
import { createPost } from '../actions';

export default function FormComponent() {
  const [state, formAction] = useActionState(createPost, { success: false, message: '' });

  return (
    <form action={formAction}>
      <input name="title" />
      <button type="submit">등록</button>
      <p>{state.message}</p>
    </form>
  );
}
```

(5) Sever Actions 사용법 - Button 형식

- 액션 호출

```
// actions.ts
'use server';

export async function createPost(formData: FormData) {
  const title = formData.get('title');
  // DB 저장 등 처리
}
```

```
// components/PostButton.tsx
'use client';

import { createPost } from '../actions';

export default function PostButton() {
  const handleClick = async () => {
    const formData = new FormData();
    formData.append('title', '예시 제목');
    await createPost(formData);
  };

  return <button onClick={handleClick}>등록</button>;
}
```

- 상태 관리 / 캐시 무효화

```
'use server';
import { revalidateTag } from 'next/cache';

export async function createPost(formData: FormData) {
  const title = formData.get('title')?.toString();
  if (!title) throw new Error('제목 필요');

  // 데이터 처리
  await revalidateTag('posts');
}
```

```
'use client';
import { useState } from 'react';
import { createPost } from '../actions';

export default function ButtonComponent() {
  const [message, setMessage] = useState('');

  const handleClick = async () => {
    const formData = new FormData();
    formData.append('title', '테스트 제목');

    try {
      await createPost(formData);
      setMessage('등록 성공');
    } catch {
      setMessage('에러 발생');
    }
  };

  return (
    <>
      <button onClick={handleClick}>등록</button>
      <p>{message}</p>
    </>
  );
}
```

(7) Form/Button 방식에 따른 Server Actions 사용법 정리

항목	Form 기반 방식	Button(onClick) 기반 방식
액션 호출 방식	`<form action={serverAction}>` 자동 제출	`onClick={() => serverAction()}` 수동 호출
`use server` 위치	액션 함수 내부	액션 함수 내부
`use client` 필요 여부	❌ (기본), ✅ `useActionState` 사용 시	✅ 항상 필요
상태 관리	`useActionState` 가능	`useState`, `useTransition` 등 수동 관리
캐시 무효화 방식	`serverAction` 내 `revalidatePath` or `revalidateTag`	동일 (`serverAction` 내에서 호출)
UI 반영 방식	자동 갱신 (form 제출 후 반영)	수동 fetch 필요 가능성 있음