Link:https://nureddineraslan.medium.com/caching-in-flutter-an-effective-approach-to-speed-up-your-app-77e688d3fbe5

Flutter is one of the most popular cross-platform mobile app development solutions today. However, accessing data quickly and efficiently within your app can sometimes be challenging. This is where caching techniques come into play. With caching methods provided by Flutter, you can enhance the speed and performance of your application. In this article, we will explore caching in Flutter, discuss various caching strategies, and provide code examples.

1. Understanding Caching: Caching involves temporarily storing frequently used data to make it readily available for reuse. In Flutter, caching refers to storing data on disk or in memory and retrieving it from there when needed. This improves performance by avoiding repetitive network operations.
2. Caching Strategies: a. Memory Cache: Flutter offers the MemoryCache class for storing data temporarily in memory. This method is ideal for caching data that needs to be quickly accessed. For example, by caching the results of API calls in memory, you can retrieve the same data rapidly without repeating the network request.

```
import 'package:flutter_cache_manager/flutter_cache_manager.dart';

void main() {
  final cache = DefaultCacheManager();

  // Cache the result of an API call
  cache.putFile('api_response', http.Response('{"message": "Hello, World!"}', 200));

  // Retrieve the cached data from memory
  final cachedData = cache.getFileFromMemory('api_response');
  print(cachedData.data);
}
```

b. Disk Cache: Disk caching involves storing data on the device's disk for long-term retention. This method is suitable for caching data that doesn't require network access every time the app launches. For instance, you can cache user profile images to the disk and load them quickly instead of downloading them again.

```
import 'package:flutter_cache_manager/flutter_cache_manager.dart';

void main() {
  final cache = DefaultCacheManager();

  // Cache the image to disk
  cache.downloadFile('https://example.com/profile.jpg');

  // Load the image from disk cache
  final cachedImage = cache.getFile('https://example.com/profile.jpg');
```

```
  print(cachedImage.path);
}
```

Removing Data from Cache: Caches can grow over time and hold unnecessary data. Therefore, it's important to regularly clean up cached data. Flutter provides various methods to remove data from memory and disk cache.

```
import 'package:flutter_cache_manager/flutter_cache_manager.dart';

void main() {
  final cache = DefaultCacheManager();

  // Remove the cached data
  cache.removeFile('api_response');

  // Clear the memory cache
  cache.emptyCache();

  // Clear the disk cache
  cache.emptyCacheDisk();
}
```

Conclusion: Caching in Flutter is an effective approach to improve your app's performance and provide quick access to data. In this article, we discussed memory and disk caching strategies and provided relevant code examples. By employing these methods, you can enhance the speed of your application and deliver an improved user experience.

This is sample app:

```
import 'package:flutter/material.dart';
import 'package:flutter_cache_manager/flutter_cache_manager.dart';
import 'dart:convert';

void main() {
  runApp(MyApp());
}

class MyApp extends StatefulWidget {
  @override
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  final cache = DefaultCacheManager();
  List<String> cachedData = [];

  @override
```

```
void initState() {
 super.initState();
 fetchData();
}

Future<void> fetchData() async {
 final file = await cache.getFile('api_data');

 if (file != null && file.file.existsSync()) {
  final jsonData = await file.file.readAsString();
  final data = json.decode(jsonData) as List<dynamic>;

  setState(() {
   cachedData = data.map((item) => item.toString()).toList();
  });
 } else {
  final response = await DefaultCacheManager().downloadFile('https://api.example.com/data');

  if (response.statusCode == 200) {
   final jsonData = await response.file.readAsString();
   final data = json.decode(jsonData) as List<dynamic>;

   await cache.putFile('api_data', response.file);

   setState(() {
    cachedData = data.map((item) => item.toString()).toList();
   });
  } else {
   print('API request failed');
  }
 }
}

@override
Widget build(BuildContext context) {
 return MaterialApp(
  title: 'Cache Example',
  theme: ThemeData(
   primarySwatch: Colors.blue,
  ),
  home: Scaffold(
   appBar: AppBar(
    title: Text('Cache Example'),
   ),
```

```
    body: Center(
      child: ListView.builder(
        itemCount: cachedData.length,
        itemBuilder: (context, index) {
          return ListTile(
            title: Text(cachedData[index]),
          );
        },
      ),
    ),
  ),
 );
}
```