Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2019, 6(5):267-271**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Implementing a Caching Framework in Salesforce Apex

Chirag Amrutlal Pethad

PetSmart.com, LLC, Stores and Services,
Phoenix, Arizona, USA
ChiragPethad@live.com, ChiragPethad@gmail.com, Cpethad@petsmart.com

Abstract: Caching is a technique used in computing to store copies of data in a temporary storage location, or cache, so that future requests for that data can be served faster. The main purpose of caching is to improve the performance and efficiency of data retrieval operations by reducing the time and resources required to access frequently used data.

Keywords: Latency, Performance, Cache Hits, Cache Miss, Cache Eviction, TTL – Time-to-Live, LRU – Least-Recently-Used, LFU – Least-Frequently-Used, FIFO – First-In-First-Out, Persistent Storage, Frameworks

Introduction

Efficiency and performance are paramount in any enterprise application development. Salesforce, as a leading customer relationship management (CRM) platform, provides a rich set of functionalities, but even the most optimized systems can benefit from strategic use of caching. Implementing a caching framework in Salesforce Apex can significantly reduce the load on databases / backend, decrease response times, and improve overall system performance. This white paper explores the principles, benefits, and best practices for implementing a caching framework in Salesforce Apex. This framework provides a systematic and standardized way to cache data that doesn't change frequently.

Key Concepts

A. Cache Storage

A cache is typically a high-speed temporary storage layer that can be located in various places, such as in-memory (RAM), on disk, or distributed across a network. The choice of storage depends on the specific requirements and use cases.

B. Cache Hit and Cache Miss

When requested data is found in the cache, it is served directly from the cache, is called a Cache Hit and results in a faster response.

When requested data is not found in the cache, called as Cache Miss, it must be retrieved from the original source (e.g. Database, file system). The retrieved data is then stored in the cache for future requests.

C. Cache Eviction

Caches have limited storage capacity, so when the cache becomes full, older or less frequently accessed data is removed to make room for new data. Common eviction policies include:

1. Least Recently Used (LRU): Removes the least recently accessed items first.
2. First In, First Out (FIFO): Removes the oldest items first.
3. Least Frequently Used (LFU): Removes the least frequently accessed items first.

D. Cache Consistency

Ensuring that the data in the cache remains consistent with the data in the original source. This can be achieved through various strategies, such as setting expiration times (TTL - Time To Live) for cached data or using cache invalidation techniques.

E. Frameworks

A framework in software development is a structured platform that provides a foundation for building applications. It comprises a collection of pre-written code, libraries, tools, and best practices that developers can use to streamline the development process and ensure consistency across projects. Frameworks are designed to help developers focus on the specific functionality of their applications rather than dealing with the underlying infrastructure and repetitive tasks.

Importance of caching

Caching is a technique used to temporarily store data in a high-speed memory storage area to reduce the time it takes to access data. The key benefits of caching include:

A. Improved Performance

Caching reduces the load on the database and speeds up data retrieval. Retrieving data from the cache is much faster than through an API call. When comparing SOQL to cache retrieval times, the cache is also much faster than SOQL queries. Figure 1 & 2 shows the retrieval times, in milliseconds, of data through an API call and the cache. It is easy to notice the huge performance gain when fetching data locally through the cache, especially when retrieving data in multiple transactions. In the sample used for the graph, the cache is hundreds of times faster than API calls. Cache retrieval times are just a few milliseconds but appear as almost zero due to the scale used for the time value. Keep in mind that this chart is a sample test, and actual numbers might vary for other apps.

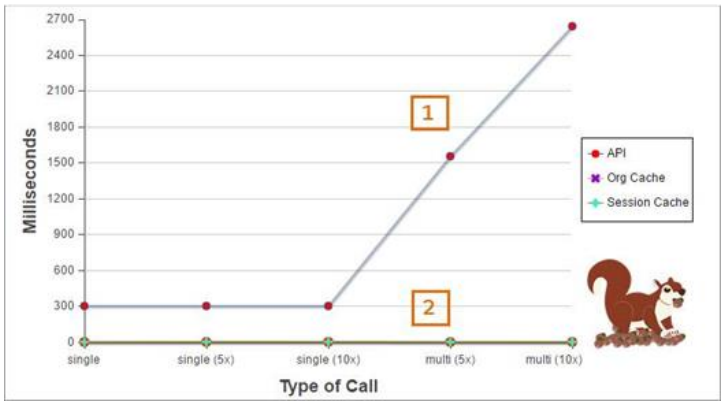

***Figure 1**: Making API Calls to External Services Is Slower (1) Than Getting Data from the Cache (2)*

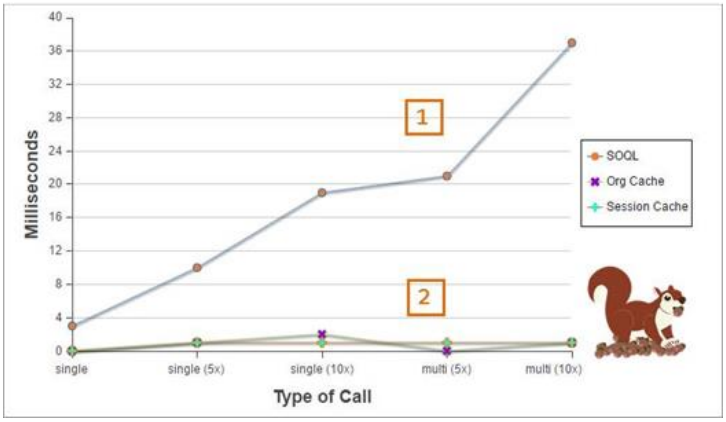

***Figure 2:** Getting Data through SOQL queries Is Slower (1) Than Getting Data from the Cache (2)*

B. Other Benefits

- **Scalability**: Enhances system's ability to handle increased load and traffic.
- **Reduced Latency**: Minimizes the time required to fetch the data, leading to faster response times.
- **Cost Efficiency**: Lowers operational costs by decreasing database read operations.
- **Avoiding Limits:** Helps avoid hitting existing Salesforce limits like 100 SOQL query limits in a transaction.

Options For Caching in Salesforce

A. Custom Setting
B. Custom Metadata Types
C. Static Object
D. Platform Cache

Both Custom Setting and Custom Metadata types are used to store application configuration information which is structured data. They both internally cache the data after the first data retrieval. However, we cannot cache runtime data using Custom Setting or Custom Metadata Types. Both are used for permanent cache.

Static variable can also be used to cache items for a transaction, but we must declare lots of variables for specific type to store runtime data. Also, the cached data will be used in only specific transaction and cannot be used outside the scope of transaction. Platform Cache can be used to store any type of data at runtime and it can be used for specific session or by all users.

Types Of Platform Caching In Salesforce

Salesforce provides several options for caching which fall under the umbrella term called Platform Cache. Platform Cache is the memory layer that stores Salesforce data. The options include:

A. Org Cache

Org Cache is a Salesforce feature to store data that can be shared across users, sessions and transactions.

B. Session Cache

Session Cache stores data that is specific to a user's session.

C. Custom Cache

A custom cache solution that stores data only for a single transaction.

Implementation Of Caching Framework In Apex

A. Org Cache

The primary use case for using Org cache is for caching static data or data that does change often. Data that never change or changes only once a day or few days are ideal candidates. There are 2 ways to implement Org Cache:

Option 1: We start by creating an Apex Class as follows

force-app > main > default > classes > ProductsCache.cls > ...

```
public with sharing class ProductsCache implements Cache.CacheBuilder{

    private static ProductsCache selfInstance;
    public static ProductsCache getInstance() {
        if(selfInstance == null) {
            selfInstance = new ProductsCache();
        }
        return selfInstance;
    }

    public Object getFromCache(String cacheKey) {
        return Cache.Org.get(ProductsCache.class, cacheKey);
    }

    public Object doLoad(String cacheKey){
        return [Select Id, Name, ProductCode from Product2
                Where ProductCode = :cacheKey LIMIT 1];
    }
}

```

Option 1: Usage Example

scripts > apex > UsageExample.apex

Execute | Debug

```
List<Product2> cachedProducts = (List<Product2>)ProductsCache.getInstance().getFromCache('GC3020');
System.debug('Product record from ProductsCache - CacheBuilder >>' + cachedProducts[0]);

```

Option 2: By Creating custom Implementation

force-app > main > default > classes > ICache.cls > ...

```
public interface ICache {
    void cacheData(String key, Object value);
    Object getCachedData(String key);
}

```

force-app > main > default > classes > ProductsCache2.cls > ...

```
public with sharing class ProductsCache2 implements ICache{

    private static ProductsCache2 selfInstance;
    public static ProductsCache2 getInstance() {
        if(selfInstance == null) {
            selfInstance = new ProductsCache2();
        }
        return selfInstance;
    }

    public void cacheData(String cacheKey, Object value){
        Cache.Org.put(cacheKey, value);
    }

    public Object getCachedData(String cacheKey){
        return Cache.Org.get(cacheKey);
    }
}
```

Option 2: Usage Example

scripts > apex > UsageExample.apex

Execute | Debug

```
Product2 product = [Select Id, Name, ProductCode from Product2 Where ProductCode = 'GC3020' LIMIT 1];

//Cache the Product Data Manually
ProductsCache2.getInstance().cacheData('GC3020', product);

//Retrieve the Product Data from Cache
Product2 cachedProduct = (Product2)ProductsCache2.getInstance().getCachedData('GC3020');
System.debug('Product record from ProductsCache2 >> ' + cachedProduct);
```

B. Session Cache

force-app > main > default > classes > ICache.cls > ...

```
public interface ICache {
    void cacheData(String key, Object value);
    Object getCachedData(String key);
}
```

force-app > main > default > classes > ProductSessionCache.cls > ...

```
public with sharing class ProductSessionCache  implements ICache{

    private static ProductSessionCache selfInstance;
    public static ProductSessionCache getInstance() {
        if(selfInstance == null) {
            selfInstance = new ProductSessionCache();
        }
        return selfInstance;
    }

    public void cacheData(String cacheKey, Object value){
        Cache.Session.put(cacheKey, value);
    }

    public Object getCachedData(String cacheKey){
        return Cache.Session.get(cacheKey);
    }
}
```

Usage Example

scripts > apex > UsageExample.apex

Execute | Debug

```
Product2 product = [Select Id, Name, ProductCode from Product2 Where ProductCode = 'GC3020' LIMIT 1];

//Cache the Product Data Manually
ProductSessionCache.getInstance().cacheData('GC3020', product);

//Retrieve the Product Data from Cache
Product2 cachedProduct = (Product2)ProductSessionCache.getInstance().getCachedData('GC3020');
System.debug('Product record from Session Cache >> ' + cachedProduct);
```

C. Custom Cache

force-app > main > default > classes > ICacheManager.cls > ...

```
public interface ICacheManager {

    Object get(String key);
    void put(String key, Object value);

}
```

force-app > main > default > classes > ContextManager.cls > ...

```
public class ContextManager implements ICacheManager{

    private static Map<String, Object> contextCache {
        get {
            if (contextCache == null) contextCache = new Map<String, Object>();
            return contextCache;
        } set;
    }

    public Object get(String key) {
        return contextCache.get(key);
    }

    public void put(String key, Object value) {
        contextCache.put(key, value);
    }
}
```

Usage Example

scripts > apex > UsageExample.apex

Execute | Debug

```
Product2 product = [Select Id, Name, ProductCode from Product2 Where ProductCode = 'GC3020' LIMIT 1];

ICacheManager manager = new ContextManager();
manager.put('GC3020', product);

//Retrieve the Product Data from Cache
Product2 cachedProduct = (Product2)manager.get('GC3020');
System.debug('Product record from Local Cache >> ' + cachedProduct);
```

B. Cache is short lived and per default visible and mutable.
C. Expect the Cache to fail you.
D. What's your strategy to invalidate cache?
E. Don't use Cache as temporary storage for transactional data.
F. Don't rely on the integrity of cached items as they are not immutable.
G. Do not store Items larger than 100kb.
H. Don't use Cache as a fast, limit-free database.
I. Cached items are not persisted. So do not rely on them being there.
J. Cached data goes stale. So plan to rebuild the cache using Trggers.

Best Practices for Implementing Caching

A. Determine What to Cache

Identify frequently accessed data that is expensive to retrieve from the database and cache this data to improve performance.

B. Set Appropriate Expiration Time

Set appropriate expiration times (TTL - Time to Live) for cached data to ensure data consistency and prevent stale data.

C. Monitor Cache Usage

Regularly monitor cache usage and performance to ensure that caching is providing the expected benefits and adjust as needed.

D. Handle Cache Misses

Implement logic to handle cache misses gracefully by falling back to retrieving data from the database.

E. Secure Cached Data

Ensure that cached data is properly secured, and that sensitive information is not exposed through the cache.

F. Clear Expired Cache Entries

Implement mechanisms to clear expired cache entries to maintain cache efficiency and avoid unnecessary storage usage.

G. How much to Cache

Cache lists or maps of objects rather than single objects. Ensure the list or map does not exceed 100kb limit.

Conclusion

Implementing a caching framework in Salesforce Apex can significantly enhance the performance and scalability of your applications. By strategically caching frequently accessed data, reducing database load, and minimizing response times, businesses can improve user experience and operational efficiency. Whether using Platform / Org Cache, Session Cache, or a custom caching solution, the principles and best practices outlined in this white paper provide a solid foundation for building a robust caching framework in Salesforce Apex. With careful planning and implementation, caching can become a powerful tool in your Salesforce development toolkit, driving better performance and scalability for your applications.

References

[1]. Salesforce Developer Guide - https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_cache_namespace_overview.htm
[2]. Salesforce Trailhead - https://trailhead.salesforce.com/content/learn/modules/platform_cache
[3]. Salesforce Apex Reference Guide - https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex_namespace_cache.htm
[4]. Salesforcecodex - https://salesforcecodex.com/salesforce/enhance-apex-performance-with-platform-caching/
[5]. ApexHours - https://www.apexhours.com/platform-cache