Thoughts on Caching Strategies

Posted At : October 7, 2020 1:00 PM | Posted By : Steve
Related Categories: ColdFusion

I've been thinking about caching strategies for different scenarios lately. I wanted to write down a some thoughts on what seem to have been working for me. Hopefully this will help me organize my thoughts and maybe others can help tell me if my thinking makes sense or if I am missing something.

What is Caching?

According to the **AWS page on caching**:

> In computing, a cache is a high-speed data storage layer which stores a subset of data, typically transient in nature, so that future requests for that data are served up faster than is possible by accessing the data's primary storage location. Caching allows you to efficiently reuse previously retrieved or computed data.

This page goes on to mention RAM as a frequent storage mechanism for caching. The upshot, though, is that the data is stored somewhere that is more convenient (faster) for access. Regardless of the means of storage, though, you can expect some trade-off for how much you have stored in cache. Caching absolutely everything will not likely be the most efficient approach.

So, caching speeds up data retrieval but we don't want to put everything in the cache. So, how do we come up with a good strategy for what to cache and how long to cache it?

Factors to Consider

With data retrieval, we need to consider a few factors:

- **Time to Retrieve Data**: The longer it takes to retrieve data, the more benefit we gain from caching. The faster it can be retrieved, the less benefit caching affords us.
- **Amount of Data Retrieved**: The more data being retrieved, the higher the cost to cache it. The less data being retrieved, the lower the cost to cache it.
- **The Frequency of Data Access**: The more frequently data is accessed, the more we benefit from caching it. The less frequently it is access, the less benefit we gain from caching it.
- **The frequency with with the data changes**: Our caching strategy should minimize retrievals from the source while keeping the data from being overly stale.

If we have data that takes a long time to retrieve, is used often, and is a moderate amount of data, then that is a perfect candidate for caching. Data that can be retrieved quickly and is accessed rarely (especially if the data amount is large) is a poor candidate for caching. Everything else in between is a gray area.

Of course, the gray area is where we live. I'll cover my thoughts on that in just a bit. First, though, we should think a bit about how to manage frequency of data changes.

Managing Data Changes

As I mentioned, the challenge with caching is minimizing data retrieval without allowing data to get stale.

This can be managed by interval-based caching. In ColdFusion, this could be done with a <cfquery> "cachedwithin" attribute or a "timespan" attribute in <cfcache> or in caching functions.

Don't get too caught up in the fact that I am using ColdFusion for examples, by the way. The concepts should cross languages.

The advantage of this strategy is that it is very easy to implement. The drawback is that you have to guess the time period and your data will be potentially that stale while also potentially retrieving data much more often than you need to.

Another strategy is to track any time that the data is changed and update the cache.

In ColdFusion, this could mean setting a variable any time that the data is changed and use that for the "cachedafter" attribute of or it could mean manually flushing the cache (which my own **MRECache** makes easy and flexible).

I'm often surprised that the "cachedafter" attribute in doesn't get more attention, by the way. Set the value of a variable every time that you update the data and it will stay cached until you update it again. How great and easy is that!

Updating the cache when the data updates ensures that it is fresh and not retrieved more often than needed. This is true regardless of the language or mechanism.

Balancing Cache Storage

Of course, if I am suggesting caching data forever then doesn't that go against my earlier suggestion that we don't store more in the cache than we need? Potentially, yes.

I think that suggestion still applies well for data that is accessed frequently.

In general, though, I would suggest a combination approach. Cache the data with a time limit, but refresh it as needed. Again, MRECache is built to solve this, but it would be easy enough to do with most any caching solution.

The other challenge comes when dealing with data that is accessed unevenly. For example, let's say that you are caching every category record that you retrieve. Maybe you have 100 category records. But maybe 3 of those categories are accessed very frequently, but the other 97 hardly at all. How do you efficiently cache the records in this case.

This is where the "idleTime" comes in very handy. You can set that low enough that the other categories will likely frequently be purged from the cache, but that the popular categories will stay in it. This may mean that categories are frequently being put in, and removed from, the cache.

That is OK, though, so long as the cost of doing so is more than offset by the savings of so frequently retrieving the popular categories. Which takes us back to the consideration of time to retreive data. If data retrieval is too fast or too infrequent, then that data may not be a good candidate for caching.

Summary

By using a combination of timeSpan, idleTime, and manual updates to the cache data you should be able to come up with a caching strategy that is maximally efficient.

Ideally, I would want to clear cached data whenever it is updated. If I can't do that, then I would to set the timeSpan to balance the speed and the degree to which the data is stale. If I can clear the cache when data is updated, then that frees me to have a more generous timeSpan.

The idleTime can be set to balance the time to retrieve the data with its popularity. The higher the data retrieval time and the fewer instances of the data, the longer I can set my idleTime. The lower the data retrieval time or the more instances of the data, the lower I would set my idleTime.

If anything I've said is unclear, or if my thinking is mistaken in any way, please do let me know in the comments. As I've said, this has been working for me, but I am always looking to learn and improve.