The preceding code means that we apply a full-page cache to the index action. The page will be cached for a year and the cache will refresh if one of the dependency data changes. Therefore, in general, the dependency works as follows:

• The first run gets the fresh data as described in the dependency, saves it for future reference, and updates the cache

• It gets the fresh data as described in dependency, gets the saved data, and then compares the two

• If they are equal, it uses the cached data

• If not, it updates the cache, uses the fresh data, and saves the fresh dependency data for future reference

In our case, two dependency types are used—tag and DB. A tag dependency marks data with the custom string tag and checks it to decide if we need to invalidate the cache, while a DB dependency uses the SQL query result for the same purpose.

The question that you have now is probably, “Why have we used DB for one case and tags for another?” That is a good question!

The goal of using the DB dependency is to replace heavy calculations and select a light query that gets as little data as possible. The best thing about this type of dependency is that we don’t need to embed any additional logic in the existing code. In our case, we can use this type of dependency for account operations, but cannot use it for articles as the article content can be changed. Therefore, for articles, we set a global tag named article which basically means that we can manually call the following when we want to invalidate total the article cache:

TagDependency::invalidate(\Yii::$app->cache, 'articles');

See also

In order to learn more about caching and using cache dependencies, refer to <http://www.yiiframework.com/doc-2.0/guide-caching-overview.html>