缓存

> 使用spring cache的注解，并使用redis作为缓存实现。
>
> 通过使用注解的方式自动添加缓存到redis中，默认失效时间为1天。

代码样例

配置类

```
@EnableCaching
@Configuration
public class RedisCacheConfiguration extends AbstractCacheConfiguration {

    @Override
    protected Map<String, Long> defaultExpiredTime() {
        Map<String, Long> map = new HashMap<>();
        map.put("DataDictionary", 60 * 24 * 7L);
                map.put("DataDictionary-DropDown", 60 * 24 * 7L);
        return map;
    }

}
```

> 注意需要在类上面加上@EnableCaching @Configuration，并保证能被ComponentScan扫描到目录

业务代码

```
        @Cacheable(value = "DataDictionary", key = "#id") DataDictionaryid
spelid
    public DataDictionaryVO findOne(String id) {
        return this.doFindOne(id)
                .setDataDictionaryValueVOs(dataDictionaryValueService.
findAllData(id));
    }


        @Cacheable("DataDictionary-DropDown") DataDictionary-DropDownkeykey
    public List<DropDownBoxValue> getDropDownValues(String fieldEngName,
String fieldChName) {
        return this.service.findByAbbCode(fieldEngName, fieldChName).
stream().map(s -> {
            DropDownBoxValue v = new DropDownBoxValue();
            v.setFieldValue(s.getFieldValue());
            v.setFieldName(MessagesResourceHelper.isEnLanguage() ? s.
getDisplayNameEn()
                    : s.getDisplayNameCn());
            return v;
        }).collect(Collectors.toList());
    }



        @CachePut(value = "DataDictionary", key = "#id") key
    @CacheEvict(value = "DataDictionary-DropDown", allEntries = true)
    public DataDictionaryVO put(String id, DataDictionaryVO vo) {
        this.service.update(this.transfer2POForPut(id, vo),
                ArafBeanUtils.copyListProperties(vo.
getDataDictionaryValueVOs(),
                        new ArrayList<DataDictionaryValue>(),
DataDictionaryValue.class));
        return this.doFindOne(id);
    }
```

数据字典中的查询方法，对于数据变动不是很频繁的数据项，可以直接用上面的逻辑通过注解进行缓存管理。

缓存中key自动生成的格式为：

```
{cacheNames}:{spring.application.name}:{methodName}:{输入参数}
```

redis中value数据为二进制格式

缓存annotation的说明

对于缓存声明，spring的缓存提供了一组java注解:

- @Cacheable:触发缓存写入。
- @CacheEvict:触发缓存清除。

- @CachePut:更新缓存(不会影响到方法的运行)。
- @Caching:重新组合要应用于方法的多个缓存操作。

@CachePut

@CachePut:当需要更新缓存而不干扰方法的运行时 ，可以使用该注解。也就是说，始终执行该方法，并将结果放入缓存，注解参数与@Cacheable相同。 以下是一个简单的例子:

```
@CachePut(cacheNames="book", key="#isbn") public Book updateBook(ISBN
isbn, BookDescriptor descriptor)
```

通常强烈建议不要对同一方法同时使用@CachePut和@Cacheable注解，因为它们具有不同的行为。可能会产生不可思议的BUG哦。

@CacheEvict

@CacheEvict:删除缓存的注解,这对删除旧的数据和无用的数据是非常有用的。这里还多了一个参数(allEntries),设置allEntries=true时，可以对整个条目进行批量删除。 以下是个简单的例子:

```
@CacheEvict(cacheNames="books") public void loadBooks(InputStream batch) //
cacheNames
@CacheEvict(cacheNames="books", allEntries=true) public void loadBooks
(InputStream batch)
```

@Caching

@Caching:在使用缓存的时候，有可能会同时进行更新和删除，会出现同时使用多个注解的情况.而@Caching可以实现。 以下是个简单的例子：

```
@Caching(evict = { @CacheEvict("primary"), @CacheEvict(cacheNames="
secondary", key="#p0") }) public Book importBooks(String deposit, Date
date)
```

@CacheConfig

@CacheConfig:缓存提供了许多的注解选项，但是有一些公用的操作，我们可以使用@CacheConfig在类上进行全局设置。 以下是个简单的例子：

```
@CacheConfig("books") public class BookRepositoryImpl implements
BookRepository { @Cacheable public Book findBook(ISBN isbn) {...} }
```