1. CacheProvider:定义了创建,配置,获取,管理和控制多个CacheManager.一个应用可以在访问期可以访问多个CacheProvider
2. CacheManager: 定义了创建,配置,获取,管理和控制多个唯一命名的cache.这些cache存在于CacheManger的应用上下文中，一个CacheManager紧被一个CacheProvider所拥有
3. Cache:是一个类似于Map的数据结构并临时存储以key为索引的值，一个Cache紧被一个CacherManager所拥有
4. Entry: 是一个存在Cache的Key-Value的对
5. Expiry:每一个存储在Cache中的条目有一个定义的时期，一旦超过这个时期，条目过期的状态，一旦过期，条目将不可访问，更新，删除，缓存的有效期可以通过ExpiryPolicy设置

**@CachePut**

应用到写数据的方法上，如新增/修改方法，调用方法时会自动把相应的数据放入缓存会先执行方法把数据进行(新增/修改)然后再执行**注解@CachePut**

**(@CachePut：**主要针对方法的配置，能够根据方法的请求参数对其结果进行缓存，和@Cacheable不同的是，它每次都会触发真实方法的调用。**)**

1. 例子
2. @CachePut(value = "user", key = "#user.id")
3. **public** User save(User user) {
4. users.add(user);
5. **return** user;
6. }

### @Cacheable

应用到读取数据的方法上，即可缓存的方法，如查找方法：先从缓存中读取，如果没有再调用方法获取数据，然后把数据添加到缓存中：

1. 例子:
2. @Cacheable(value = "user", key = "#id")
3. **public** User findById(**final** Long id) {
4. **for** (User user : users) {
5. **if** (user.getId().equals(id)) {
6. **return** user;
7. }
8. }
9. **return** **null**;
10. }

生成keyde实例:

1. **public** **interface** KeyGenerator {
2. Object generate(Object target, Method method, Object... params);
3. }

**clean:** 清除target目录;

**install:** 将工程打包到本地仓库，这时本地项目可以依赖，别人是依赖不了的;

**package:**将项目中的各种文件，比如源代码、编译生成的字节码、配置文件、文档，按照规范的格式生成归档，最常见的当然就是JAR包和WAR包;

**@CacheEvict**：主要对方法配置，用来标记要清空缓存的方法，当这个方法被调用并满足一定条件后，即会清空缓存。

**参数解析**：

* value：缓存的位置，不能为空。
* key：缓存的key，默认为空。
* condition：触发的条件，只有满足条件的情况才会清楚缓存，默认为空，支持SpEL。

|  |
| --- |
|  |