Spring cache

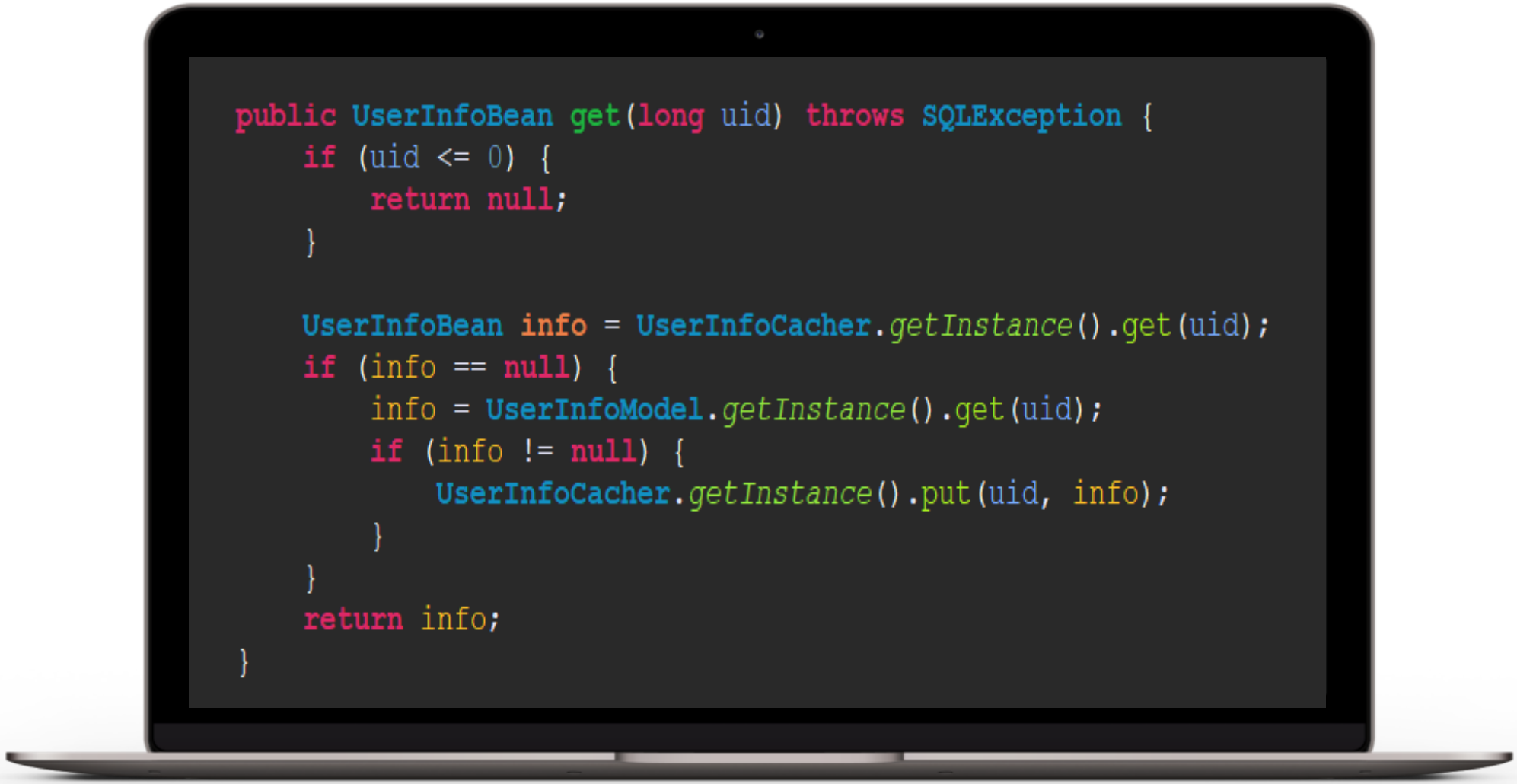
1、简单介绍

2、实现原理

3、AutoloadCache

01 简单介绍

相关概念 & 优缺点

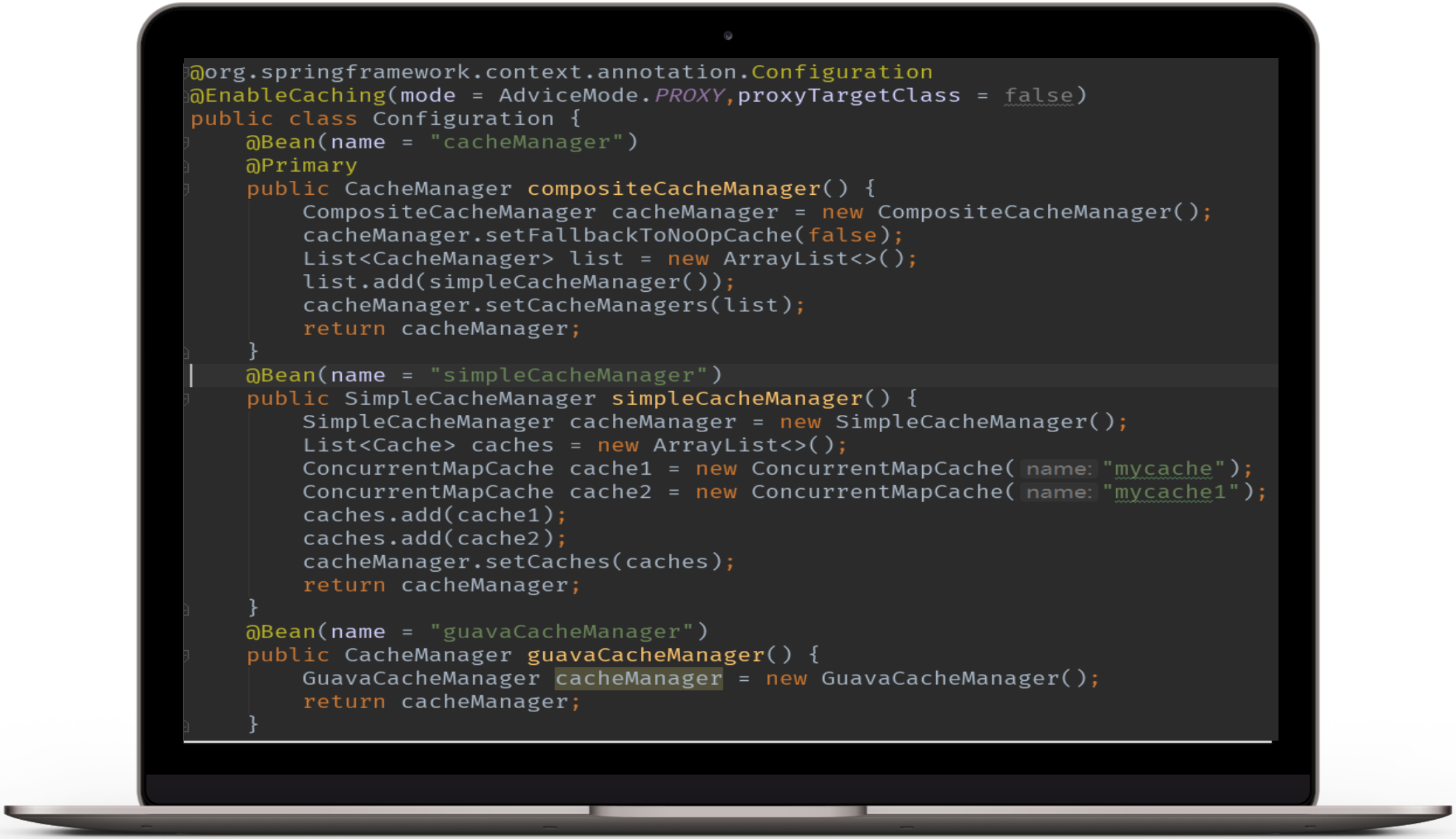

public UserInfoBean get(long uid) throws SQLException {
if (uid <= 0) {
return null;
}
UserInfoBean info = UserInfoCacher.getInstance().get(uid);
if (info == null) {
info = UserInfoModel.getInstance().get(uid);
if (info != null) {
UserInfoCacher.getInstance().put(uid, info);
}
}
return info;
}

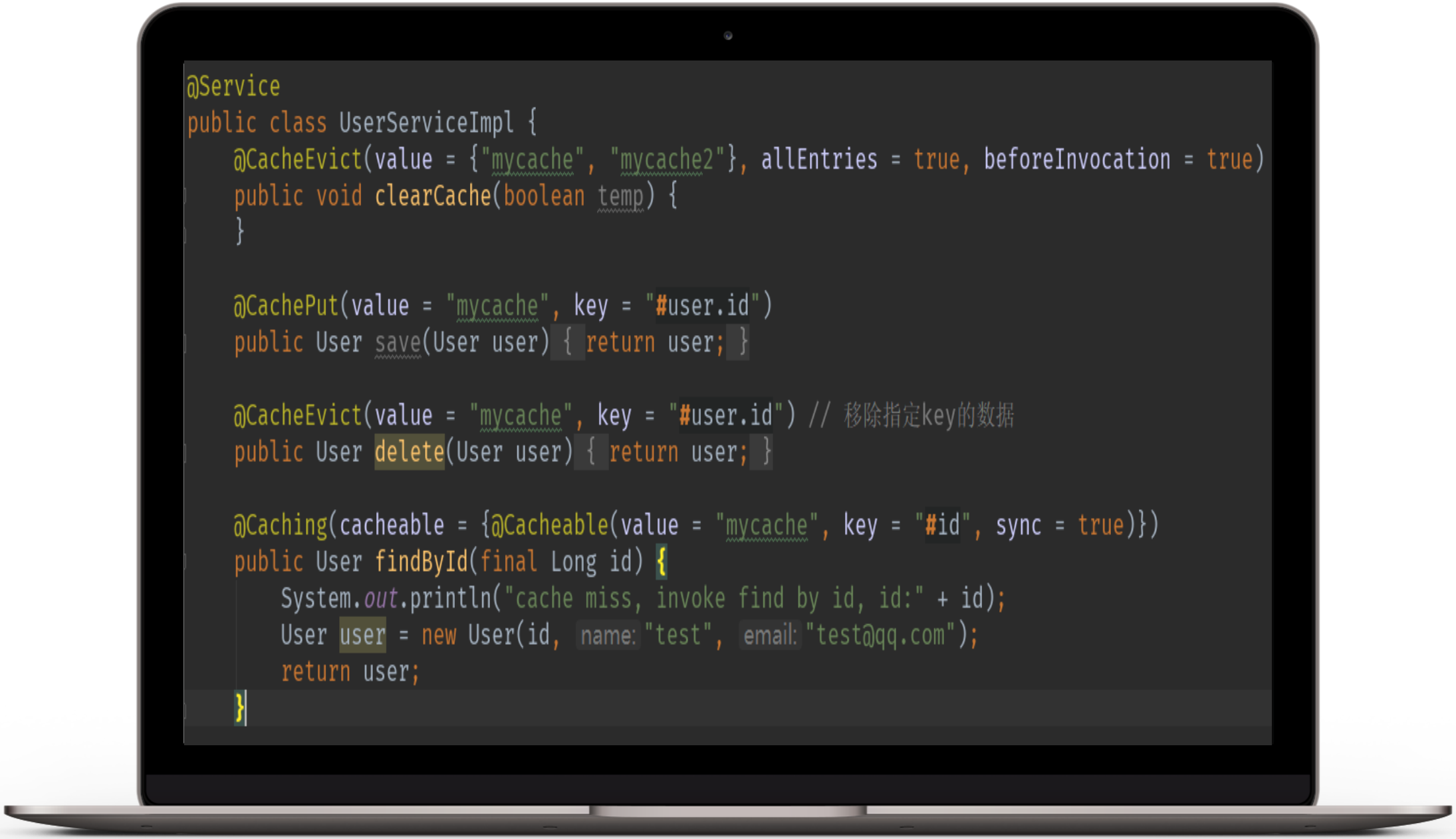
Spring cache是什么？

Since version 3.1, Spring Framework provides support for transparently adding caching into an existing Spring application. Similar to the transaction support, the caching abstraction allows consistent use of various caching solutions with minimal impact on the code.

At its core, the abstraction applies caching to Java methods, reducing thus the number of executions based on the information available in the cache. That is, each time a targeted method is invoked, the abstraction will apply a caching behaviour checking whether the method has been already executed for the given arguments. If it has, then the cached result is returned without having to execute the actual method; if it has not, then method is executed, the result cached and returned to the user so that, the next time the method is invoked, the cached result is returned. This way, expensive methods (whether CPU or IO bound) can be executed only once for a given set of parameters and the result reused without having to actually execute the method again.

Note that just like other services in Spring Framework, the caching service is an abstraction (not a cache implementation) and requires the use of an actual storage to store the cache data - that is, the abstraction frees the developer from having to write the caching logic but does not provide the actual stores.


@org.springframework.context.annotation.Configuration
@EnableCaching(mode = AdviceMode.PROXY,proxyTargetClass = false)
public class Configuration {
@Bean(name = "cacheManager")
@Primary
public CacheManager compositeCacheManager() {
CompositeCacheManager cacheManager = new CompositeCacheManager();
cacheManager.setFallbackToNoOpCache(false);
List<CacheManager> list = new ArrayList<>();
list.add(simpleCacheManager());
cacheManager.setCacheManagers(list);
return cacheManager;
}
@Bean(name = "simpleCacheManager")
public SimpleCacheManager simpleCacheManager() {
SimpleCacheManager cacheManager = new SimpleCacheManager();
List<Cache> caches = new ArrayList<>();
ConcurrentMapCache cache1 = new ConcurrentMapCache(name: "mycache");
ConcurrentMapCache cache2 = new ConcurrentMapCache(name: "mycache1");
caches.add(cache1);
caches.add(cache2);
cacheManager.setCaches(caches);
return cacheManager;
}
@Bean(name = "guavaCacheManager")
public CacheManager guavaCacheManager() {
GuavaCacheManager cacheManager = new GuavaCacheManager();
return cacheManager;
}


@Service
public class UserServiceImpl {
@CacheEvict(value = {"mycache", "mycache2"}, allEntries = true, beforeInvocation = true)
public void clearCache(boolean temp) {
}
@CachePut(value = "mycache", key = "#user.id")
public User save(User user) { return user; }
@CacheEvict(value = "mycache", key = "#user.id") // 移除指定key的数据
public User delete(User user) { return user; }
@Caching(cacheable = {@Cacheable(value = "mycache", key = "#id", sync = true)})
public User findById(final Long id) {
System.out.println("cache miss, invoke find by id, id:" + id);
User user = new User(id, name: "test", email: "test@qq.com");
return user;
}

优点

通过注解的方式使用缓存，对代码的侵入性比较低

支持开箱即用 Out-Of-The-Box

可灵活切换底层缓存的实现

支持 Spring Express Language

支持 AspectJ

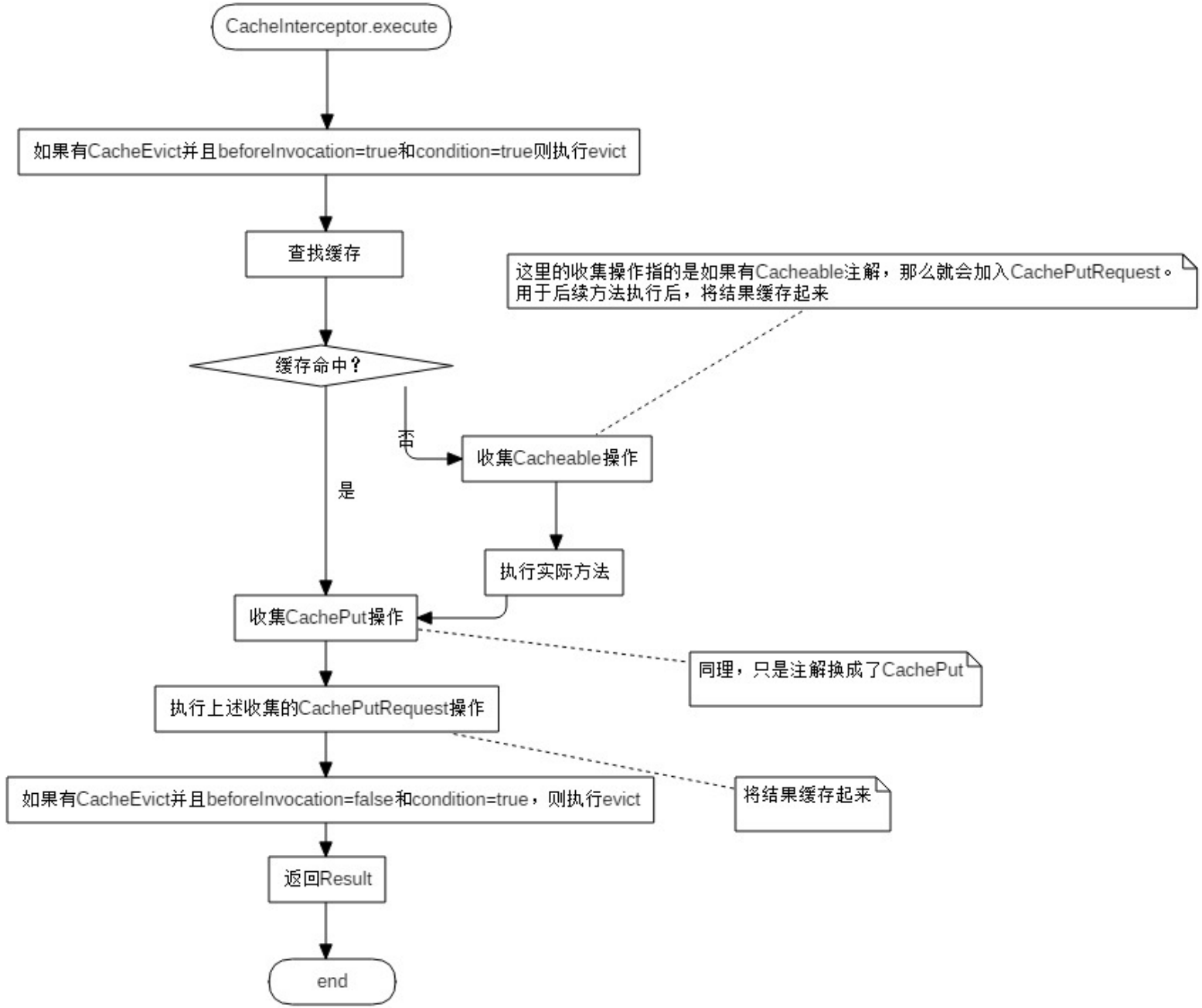
“拿来主义”机制

缺点

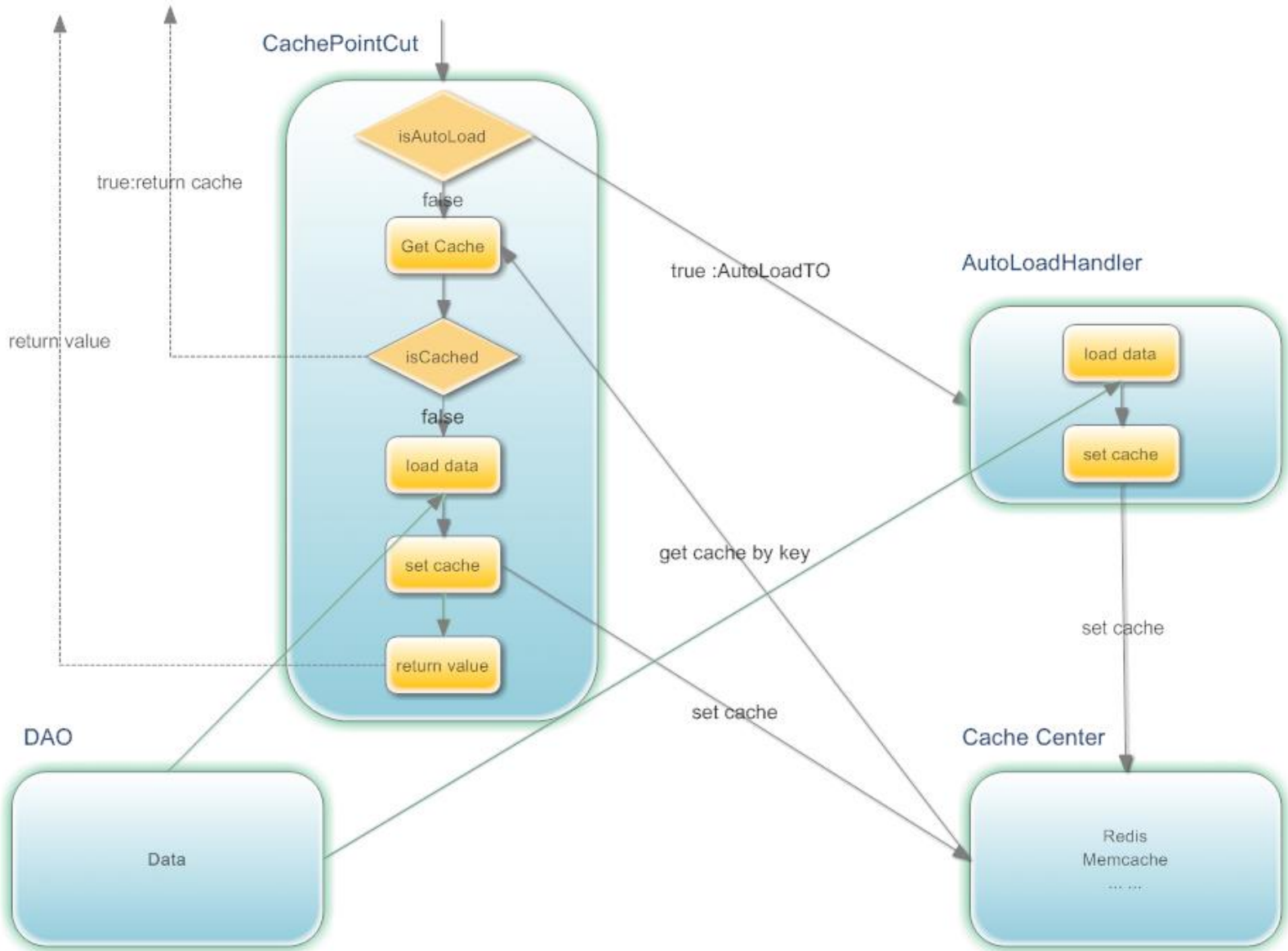
无法使用Spring EL表达式来动态生成Cache name

无法灵活设置缓存策略

02 实现原理


CacheInterceptor.execute
如果有CacheEvict并且beforeInvocation=true和condition=true则执行evict
查找缓存
这里的收集操作指的是如果有Cacheable注解，那么就会加入CachePutRequest。
用于后续方法执行后，将结果缓存起来
缓存命中？
否
是
收集Cacheable操作
执行实际方法
收集CachePut操作
同理，只是注解换成了CachePut
执行上述收集的CachePutRequest操作
将结果缓存起来
如果有CacheEvict并且beforeInvocation=false和condition=true，则执行evict
返回Result
end

AutoloadCache


CachePointCut
isAutoLoad
true:return cache
false
Get Cache
true :AutoLoadTO
AutoLoadHandler
return value
isCached
load data
false
load data
set cache
get cache by key
set cache
return value
set cache
DAO
Cache Center
Data
Redis
Memcache
... ...

相同点：

1、都是基于AOP+Annotation，将缓存与业务逻辑进行解耦；
2、都实现了拿来主义机制；
3、都支持事务；
4、可通过Spring EL表达式生成key和缓存条件
5、支持组合缓存管理方案（本地缓存+远程缓存）

不同点：

1、AutoLoadCache可以脱离spring生态使用；
2、AutoLoadCache实现了自动加载和异步刷新机制；
3、Spring Cache 使用 name 和 key 的来管理缓存，而 AutoLoadCache 使用的是 namespace + key + hfield 来管理缓存（更适合redis）；
4、AutoLoadCache可针对每个缓存key生成对应的过期时间，spring cache只可以设置统一过期时间；
5、AutoLoadCache支持扩展表达式解析器，支持sqel、ognl、javascript；
6、AutoLoadCache对缓存穿透问题提供了一些优化（默认将 null 值使用 CacheWrapper“包装”后进行缓存）；

Thank You