Кэширование в асинхронном Spring приложении

August Vilakia

AlfaBank

@augustlakia

@avgvi

JPoint

August Vilakia

avgvi

Bio

- Больше 4-х лет опыта разработки на Java. Работаю над бэкендом мобильного приложения Альфа-Банка

О ЧЁМ ПОГОВОРИМ

01 **Необходимость кэширования асинхронных запросов**

02 Проблемы с асинхронным кэшированием в Spring

03 Возможные решения для асинхронного кэширования

04 Решение проблемы в Альфа-Банке

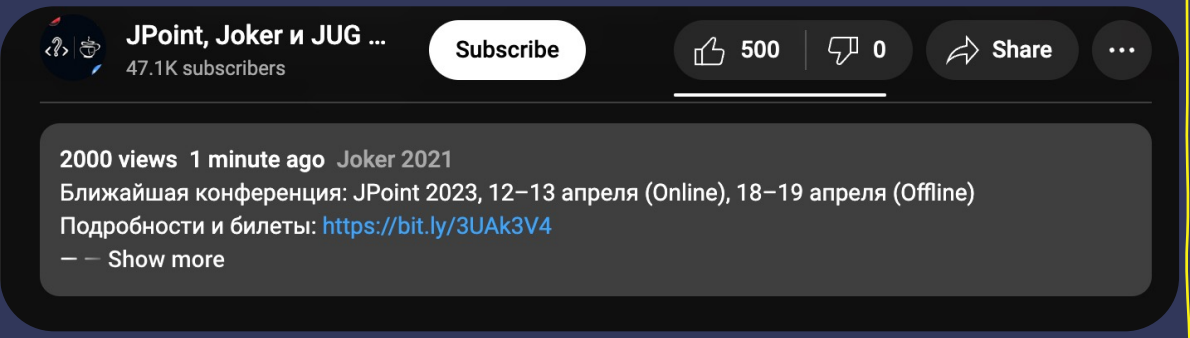
05 Вывод

ИН-МЕМОРИ КЕШ

In-memory cache, простыми словами хэш мапа с eviction policy

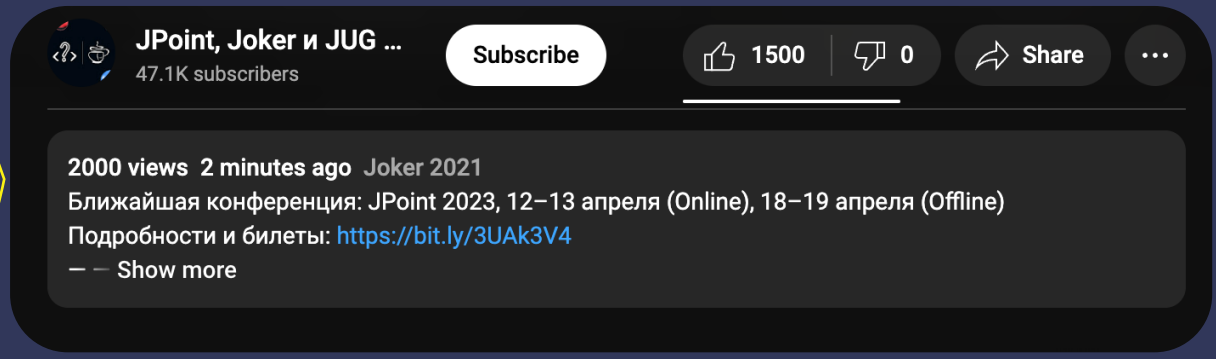
РАСПРЕДЕЛЕННЫЙ КЕШ?

1)



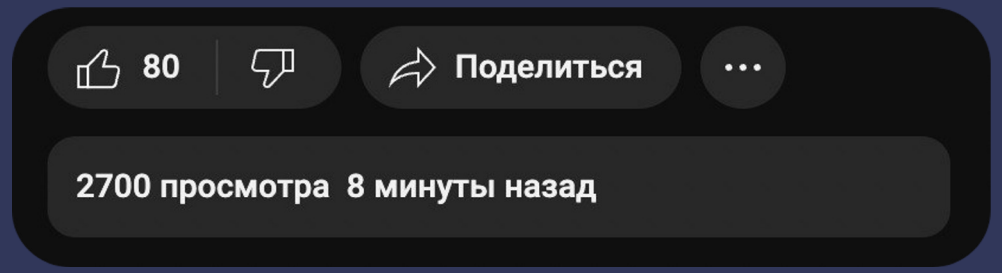
А ПОЧЕМУ ТАК?

2)

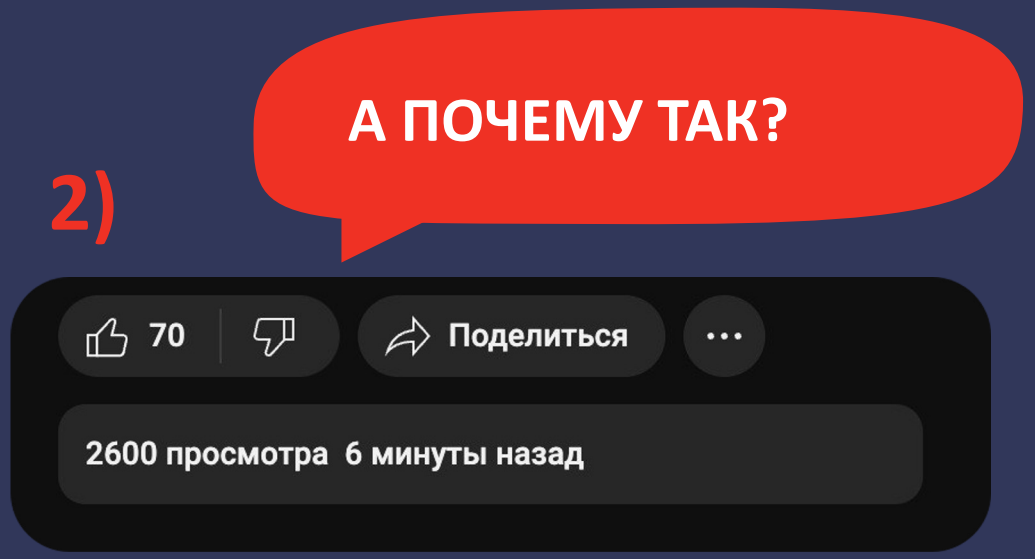


РАСПРЕДЕЛЕННЫЙ КЕШ?

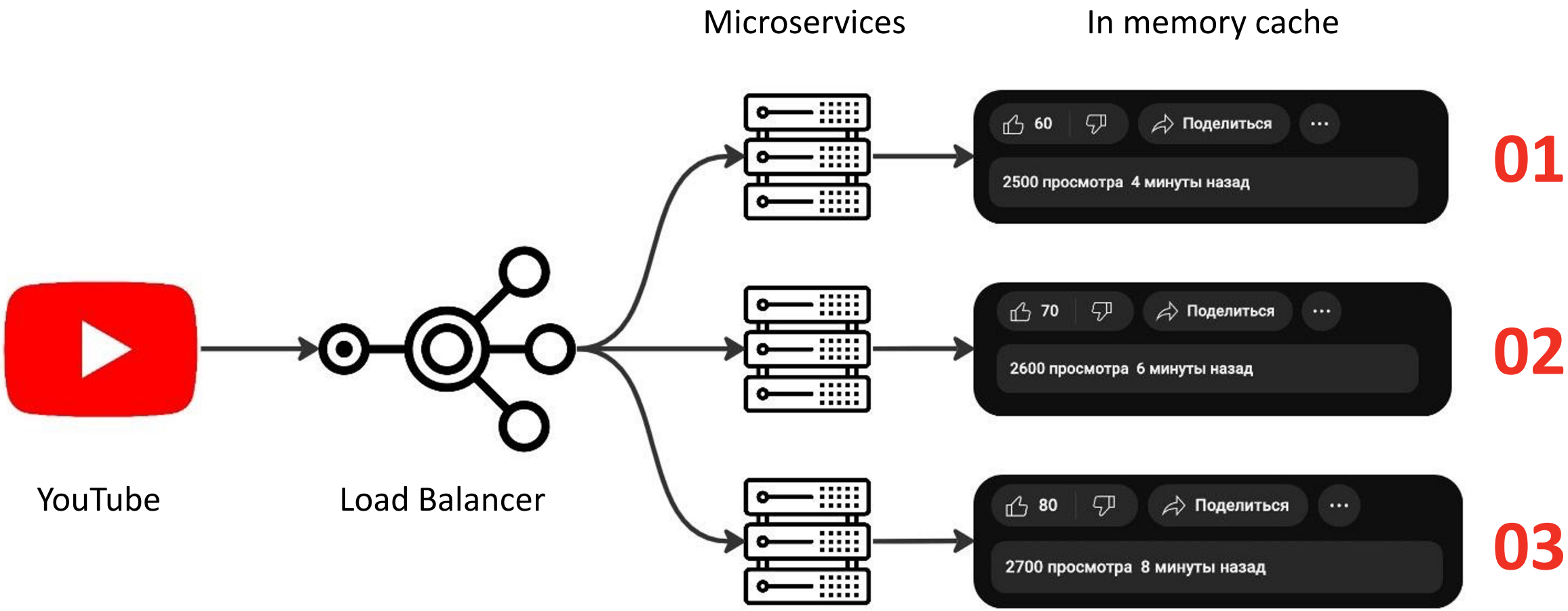
1)



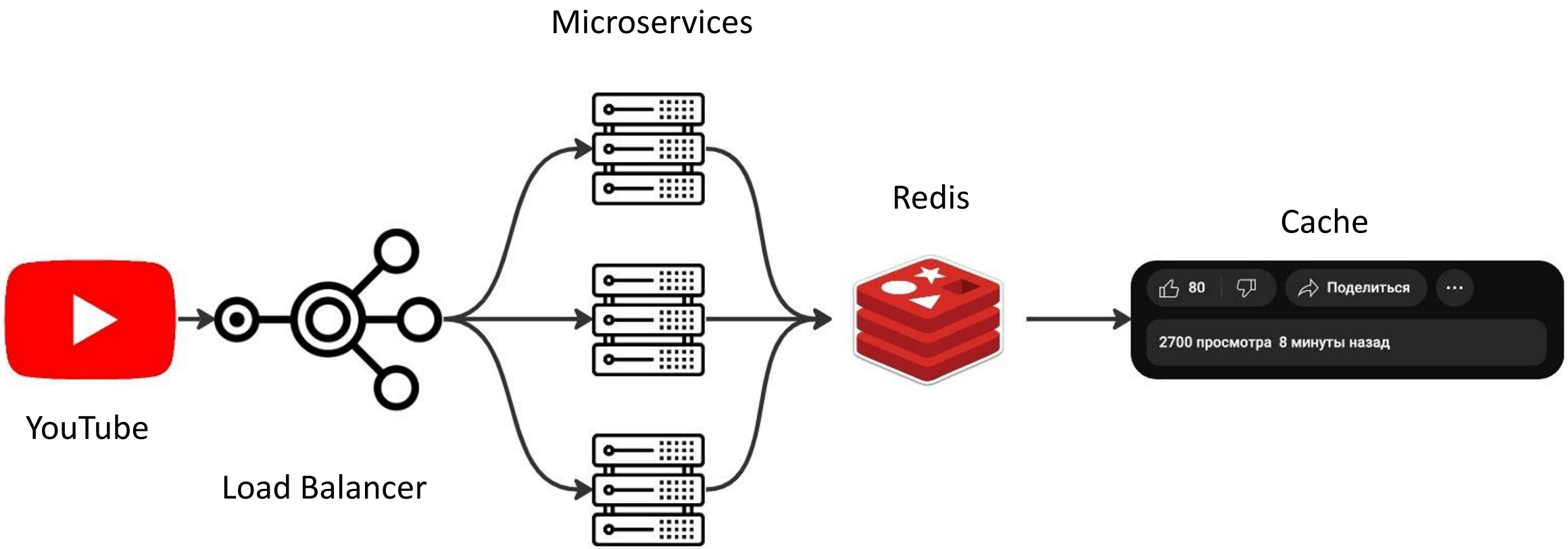
2)



РАСПРЕДЕЛЕННЫЙ КЕШ?

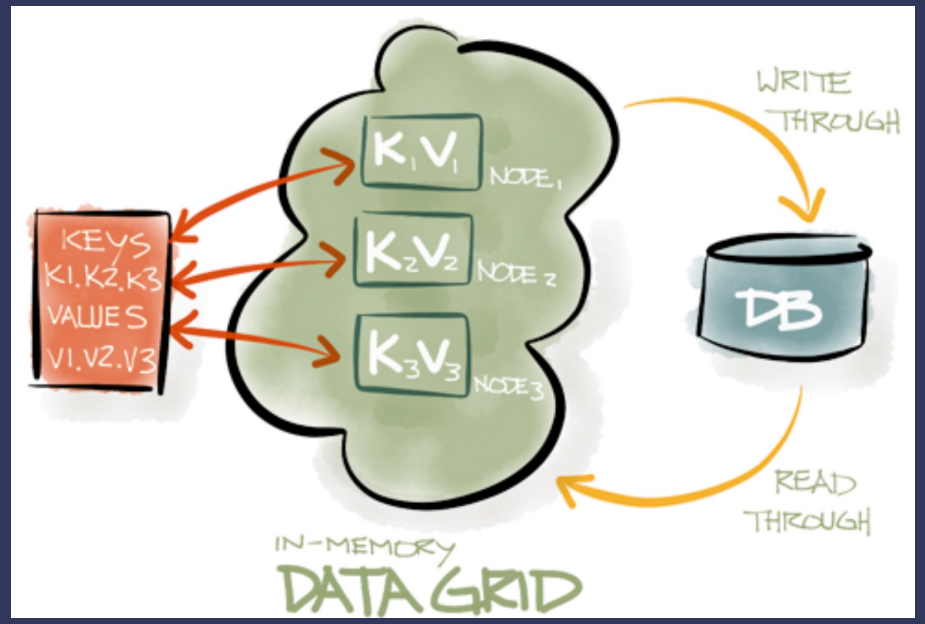


РАСПРЕДЕЛЕННЫЙ КЕШ?



fastest database query is the one you don’t have to do at all

ЗАЧЕМ ИСПОЛЬЗОВАТЬ КЕШ?



Killer-features

- Снижение нагрузки с базы данных, зависимых систем
- Улучшение времени ожидания (latency)
- Evicition Policies (LRU, LFU, TTL)

53% of users abandon a site if it takes more than 3 seconds to load

Асинхронность?

The Reactive Manifesto 01

Servlet 3.1 Non-blocking IO 02

WebFlux 4x раза быстрее когда зависимые сервисы меленные

WebFlux 20% быстрее чем NIO Servlet с CompletableFuture

Устойчивость к пикам нагрузки, backpressure

Таймауты зависимых сервисов

УПС!!!

```
t version: 5.2.1
2023-02-23 18:29:41.351  INFO 30265 --- [-query.thread-1] c.h.i.p.in
2023-02-23 18:29:42.795  INFO 30265 --- [nio-8080-exec-1] o.a.c.c.C.
2023-02-23 18:29:42.796  INFO 30265 --- [nio-8080-exec-1] o.s.web.se
2023-02-23 18:29:42.797  INFO 30265 --- [nio-8080-exec-1] o.s.web.se
2023-02-23 18:29:42.880 ERROR 30265 --- [nio-8080-exec-1] o.a.c.c.C.

java.io.NotSerializableException: kotlinx.coroutines.reactor.MonoCor
        at java.base/java.io.ObjectOutputStream.writeObject0(Object
        at java.base/java.io.ObjectOutputStream.defaultWriteFields(
        at java.base/java.io.ObjectOutputStream.writeSerialData(Obje
        at java.base/java.io.ObjectOutputStream.writeOrdinaryObject(
```

risovach.ru

О ЧЁМ ПОГОВОРИМ

01 Необходимость кэширования асинхронных запросов

02 **Проблемы с асинхронным кэшированием в Spring**

03 Возможные решения для асинхронного кэширования

04 Решение проблемы в Альфа-Банке

05 Вывод

Добавляем Spring WebFlux в существующие сервисы, в которых используется кэшированаие

Пробуем Spring WebFlux + Hazelcast

```java
@Cacheable("products")
public Mono<String> getProductByID(String ID) {
    return findById(ID);
}

private Mono<String> findById(String ID) {

    try {
        Thread.sleep( millis: 5000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    return Mono.just( data: "JPoint Hazelcast key " + ID);
}
```

Результат

WebFlux

```
023-02-25 14:53:40.647  INFO 31384 --- [           main] com.jpoint.demo.DemoApplication          : Started DemoApplication in 3.461 seconds (JVM running for 3.
2023-02-25 14:53:49.531  INFO 31384 --- [ctor-http-nio-2] c.h.i.p.impl.PartitionStateManager       : [192.168.8.203]:5701 [dev] [4.2.7] Initializing cluster partiti
2023-02-25 14:53:54.592 ERROR 31384 --- [ctor-http-nio-2] a.w.r.e.AbstractErrorWebExceptionHandler : [c66d79b6-1]  500 Server Error for HTTP GET "/jpoint/cache/123"

com.hazelcast.nio.serialization.HazelcastSerializationException Create breakpoint : Failed to serialize 'reactor.core.publisher.MonoJust'
    at com.hazelcast.internal.serialization.impl.SerializationUtil.handleSerializeException(SerializationUtil.java:115) ~[hazelcast-all-4.2.7.jar:4.2.7]
    Suppressed: reactor.core.publisher.FluxOnAssembly$OnAssemblyException:
Error has been observed at the following site(s):
    *__checkpoint ⇢ HTTP GET "/jpoint/cache/123" [ExceptionHandlingWebHandler]
Original Stack Trace:
        at com.hazelcast.internal.serialization.impl.SerializationUtil.handleSerializeException(SerializationUtil.java:115) ~[hazelcast-all-4.2.7.jar:4.2.7]
        at com.hazelcast.internal.serialization.impl.AbstractSerializationService.toBytes(AbstractSerializationService.java:178) ~[hazelcast-all-4.2.7.jar:4.2.7]
        at com.hazelcast.internal.serialization.impl.AbstractSerializationService.toBytes(AbstractSerializationService.java:154) ~[hazelcast-all-4.2.7.jar:4.2.
```

Результат

Kotlin Coroutines

```
2023-02-25 15:17:40.768 ERROR 34114 --- [nio-8081-exec-1] o.a.c.c.C.[.[.[/].[dispatcherServlet]    : Servlet.service() for servlet [dispatcherServlet] in context wi

java.io.NotSerializableException Create breakpoint : kotlinx.coroutines.reactor.MonoCoroutine
    at java.base/java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java:1187) ~[na:na]
    at java.base/java.io.ObjectOutputStream.defaultWriteFields(ObjectOutputStream.java:1572) ~[na:na]
    at java.base/java.io.ObjectOutputStream.writeSerialData(ObjectOutputStream.java:1529) ~[na:na]
    at java.base/java.io.ObjectOutputStream.writeOrdinaryObject(ObjectOutputStream.java:1438) ~[na:na]
    at java.base/java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java:1181) ~[na:na]
    at java.base/java.io.ObjectOutputStream.defaultWriteFields(ObjectOutputStream.java:1572) ~[na:na]
    at java.base/java.io.ObjectOutputStream.writeSerialData(ObjectOutputStream.java:1529) ~[na:na]
    at java.base/java.io.ObjectOutputStream.writeOrdinaryObject(ObjectOutputStream.java:1438) ~[na:na]
```

Результат Spring WebFlux + Redis

```
@Cacheable("CACHE_NAME")
suspend fun cacheSomething() = coroutineScope { this: CoroutineScope
    //Имитируем выполнение долгой сложной логики
    delay( timeMillis: 10000)
    "CACHE_VALUE" ^coroutineScope
}
```

```
@Cacheable("CACHE_NAME")
public Mono<String> cacheSomething(String key) {
    return Mono
            .just( data: "CACHE_VALUE")
            //Имитируем выполнение долгой сложной логики
            .delayElement(Duration.ofMillis(500));
}
```

Результат redis

Reactive

```
127.0.0.1:6379> keys *
1) "reactive::SIMPLE_KEY"
127.0.0.1:6379> get "reactive::SIMPLE_KEY"
"{\"@class\":\"reactor.core.publisher.MonoJust\",\"scanAvailable\":true}"
```

Coroutine

```
127.0.0.1:6379> keys *
1) "coroutine::SimpleKey [SIMPLE_KEY,Continuation at kotlin.reflect.
full.KCallables.callSuspend(KCallables.kt:56)]"
127.0.0.1:6379> get "coroutine::SimpleKey [SIMPLE_KEY,Continuation a
t kotlin.reflect.full.KCallables.callSuspend(KCallables.kt:56)]"
"\"CACHED_VALUE\""
```

Причина возникновения

01

Результаты асинхронных вычислений не Serializable

```
public abstract class Mono<T> implements CorePublisher<T> {
    static final BiPredicate EQUALS_BIPREDICATE = Object::equals;
```

```
public class CompletableFuture<T> implements Future<T>, CompletionStage<T> {
```

02

Spring Cache внутри использует асинхронные интерфейсы, что делает бессмысленным и не эффективным

О ЧЁМ ПОГОВОРИМ

01 Необходимость кэширования асинхронных запросов

02 Проблемы с асинхронным кэшированием в Spring

03 **Возможные решения для асинхронного кэширования**

04 Решение проблемы в Альфа-Банке

05 Вывод

ЧТО ДЕЛАТЬ?

Самое популярное решение

CacheFlux / CacheMono

Reactive Addons

CacheFlux / CacheMono

3 новые операции

I. lookup() – пытается найти Mono<V> по ключу в кеше

II. onCacheMissResume() – в случае отсутствия в кеше ищет в основном хранилище

III. andWriteWith() – операция для записи в кеш

CacheFlux / CacheMono

```
public Mono<String> getProductById(String id) {



    return CacheMono
            .lookup( k -> Mono.fromCompletionStage(cache.getAsync(k)).map(Signal::next), id)

            .onCacheMissResume(findByIdMongoReactiveMethod(id))

            .andWriteWith((k, signal) -> Mono.fromCompletionStage(cache.putAsync(k, signal.get())).then());
}
```

Использование реактивного API провайдеров (Redis, Hazelcast, etc.)

Reactive
API

No Spring Cache (Cache, CacheManager, etc.)

BLOCKING

```java
@Autowired
CacheManager cacheManager;

public Mono<String> getProductById(String id) {
    Cache cache = cacheManager.getCache( name: "products");
    Optional<String> value = Optional.ofNullable(cache.get(id, String.class));

    return value.map(Mono::just)
            .orElseGet(() ->  {
                return findByIdMongoReactiveMethod(id).doOnSuccess(v -> cache.put(id, v));
            });
```

No Blocking Hazelcast

IMap API example

```
@Autowired
IMap<String, String> cache;

public Mono<String> getProductById(String id) {

    return Mono.fromCompletionStage(() -> cache.getAsync(id))

            .switchIfEmpty(findByIdMongoReactiveMethod(id)

                    .doOnNext(value -> cache.putAsync(id, value)));
```

No Blocking Hazelcast

Redis Reactive API example

```
@Autowired
ReactiveRedisConnectionFactory reactiveRedisConnectionFactory;

@Autowired
RedisCacheConfiguration redisConfiguration;

public Mono<String> getProductById(String id) {

    RedisElementWriter<String> writer = redisConfiguration.getKeySerializationPair().getWriter();
    RedisElementReader<String> reader = redisConfiguration.getKeySerializationPair().getReader();


    return Mono.using(reactiveRedisConnectionFactory::getReactiveConnection,

                it-> it.stringCommands().get(writer.write(id)).map(reader::read),

                ReactiveRedisConnection::closeLater);
}
```

Поискать уже готовые решения для имплементации

Библиотеки с GitHub

Какие варианты?

spring-kotlin-coroutine

Watch 31 | Fork 65 | Star 414

последний коммит – 30 мая 2019 г.

redis-reactive-cache

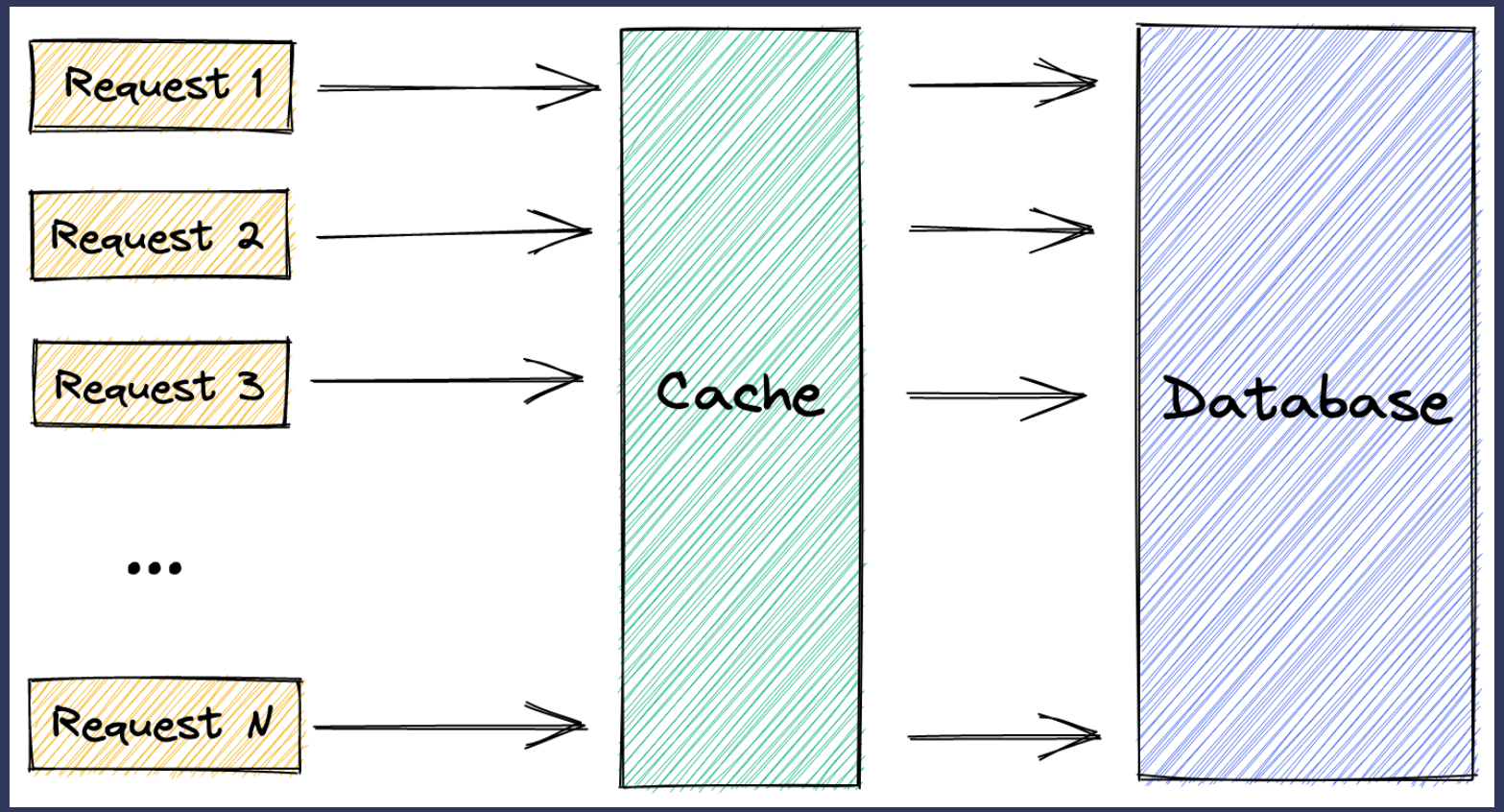
Watch 4 | Fork 4 | Star 19

последний коммит – 5 ноября 2021 г.

Проблемы данных решений?

01 Reactive-Addons – отсутствие обработки Cache Stampede

Cache Stampede что это?



Проблемы данных решений?

01 Reactive-Addons – отсутствие обработки Cache Stampede

02 Reactive-Addons – кеширование ошибок

Проблемы данных решений?

01 Reactive-Addons – отсутствие обработки Cache Stampede

02 Reactive-Addons – кеширование ошибок

03 Reactive API провайдеров – отсутствие структуры приведет к большому копированию команда в рамках компании

Проблемы данных решений?

01 Reactive-Addons – отсутствие обработки Cache Stampede

02 Reactive-Addons – кеширование ошибок

03 Reactive API провайдеров – отсутствие структуры приведет к большому копированию команда в рамках компании

04 Open Source решения – не поддерживаются, последние коммиты были очень давно

ЭТО
НИКОМУ НЕ
НУЖНО

Что говорят в разработчики VMWare, которые работают над Project Reactor и Spring?

“There is indeed a missing piece in the cache abstraction, but one caveat I could gather from looking at these various projects and issues is that unless the backing cache provider supports async use cases (and especially async write-through), it is going to be difficult to come up with a reliable abstraction”

© Spring Engineering team member at VMware, Spring Framework, ex-Reactor project lead

“Right now there isn't the engineering capacity to take this on (given its current priority). I'm going to leave this open as it is an interesting feature. Wait to the imminent release of Loom the timing is also terrible on this kind of thing. If this is going to happen it will probably happen after the landscape has settled around Reactive/Async code in the wake of Loom and its impacts on the overhead of heavy threading”

© Spring Engineering team member at VMware

О ЧЁМ ПОГОВОРИМ

redis

HAZELCAST

spring

01 Необходимость кэширования асинхронных запросов

02 Проблемы с асинхронным кэшированием в Spring

03 Возможные решения для асинхронного кэширования

04 **Решение проблемы в Альфа-Банке**

05 Вывод

Дополнить Spring Cache?

Идеальный кандидат

Аннотированное кеширование для реактива/корутин

Дополнить Spring Cache?

Идеальный кандидат

- Аннотированное кеширование для реактива/корутин
- Эффективная работа с асинхронным API провайдера

Дополнить Spring Cache?

Идеальный кандидат

- Аннотированное кеширование для реактива/корутин
- Эффективная работа с асинхронным API провайдера
- Изолированный уровень абстракции, для переиспользования

Маркерный интерфейс

для корутин и реактора

```
interface Cache<K, V> {
    fun withCondition(predicate: (V) -> Boolean): Cache<K, V>
}
```

Кеш интерфейс для Корутин

```
interface CoroutineCache<K, V, N> : Cache<K, V> {
    suspend fun get(key: K): V?
    suspend fun put(key: K, value: V)

    fun getNativeCache(): N
}

suspend fun <K, V> CoroutineCache<K, V, *>.executeGet(key: K, valueSupplier: suspend () -> V?): V? =
    get(key) ?: executePut(key, valueSupplier)

suspend fun <K, V> CoroutineCache<K, V, *>.executePut(key: K, valueSupplier: suspend () -> V?): V? =
    valueSupplier()?.also { put(key, it) }
```

Зачем executeGet?

```java
package org.springframework.cache;

import ...

public interface Cache {
    String getName();

    Object getNativeCache();

    @Nullable
    ValueWrapper get(Object key);

    @Nullable
    <T> T get(Object key, @Nullable Class<T> type);

    @Nullable
    <T> T get(Object key, Callable<T> valueLoader);
```

Зачем executeGet?

```
public <T> T get(Object key, Callable<T> valueLoader) {
    Cache.ValueWrapper result = this.get(key);
    return result != null ? result.get() : this.getSynchronized(key, valueLoader);
}

private synchronized <T> T getSynchronized(Object key, Callable<T> valueLoader) {
    Cache.ValueWrapper result = this.get(key);
    if (result != null) {
        return result.get();
    } else {
        Object value;
        try {
            value = valueLoader.call();
        } catch (Exception var6) {
            throw new Cache.ValueRetrievalException(key, valueLoader, var6);
        }

        this.put(key, value);
        return value;
    }
}
```

Так же паттерн для реактора

```
interface ReactorCache<K, V, N> : Cache<K, V> {
    fun get(key: K): Mono<V>
    fun put(key: K, value: V): Mono<Void>

    fun getNativeCache(): N
}

fun <K, V> ReactorCache<K, V, *>.executeGet(key: K, invocationResult: Mono<V>): Mono<V> =
    get(key).switchIfEmpty(
        Mono.defer { executePut(key, invocationResult) }
    )

fun <K, V> ReactorCache<K, V, *>.executePut(key: K, invocationResult: Mono<V>): Mono<V> =
    invocationResult.flatMap { put(key, it).thenReturn(it) }
```

Общий интерфейс для корутин и реактора

```
interface InvocationCacheContext<K, V> {
    fun invokeCacheable(key: K, cache: Cache<K, V>): Any?
}
```

Общий интерфейс для корутин и реактора

```
interface InvocationCacheContext<K, V> {
    fun invokeCacheable(key: K, cache: Cache<K, V>): Any?
}
```

Реализация для корутин

```
uppress( ...names: "UNCHECKED_CAST")
open class CoroutineInvocationCacheContext<K, V>(jp: JoinPoint) :
    InvocationCacheContext<K, V>,
    CoroutineInvocationContext<Any>(jp) {

    override fun invokeCacheable(key: K, cache: Cache<K, V>): V? = ::executeGet.invokeSuspend(key, cache)

    private suspend fun executeGet(key: K, cache: CoroutineCache<K, V, *>): V? = cache.executeGet(key) { await() as V }

    private fun <K, R> KFunction<R>.invokeSuspend(key: K?, cacheParam: Any): R =
        javaMethod!!.invokeSuspend( targetObj: this@CoroutineInvocationCacheContext, key, cacheParam, cont = continuation)
```

Реализация для реактора

```
@Suppress( ...names: "UNCHECKED_CAST")
class ReactorInvocationCacheContext<K, V>(private val pjp: ProceedingJoinPoint) : InvocationCacheContext<K, V> {

    override fun invokeCacheable(key: K, cache: Cache<K, V>): Mono<V> =
        (cache as ReactorCache<K, V, *>).executeGet(key, Mono.defer { pjp.proceed() as Mono<V> })


    companion object : KLogging()
}
```

Аспект

```
@Aspect
class CacheableAsyncAspect(
    processor: CacheAnnotationProcessor,
    private val invocationContextFactory: InvocationContextFactory,
    private val invocationDataExtractor: InvocationDataExtractor,
) {

    private val getOperationContext = processor::createCacheableOperation.memoize()

    @Around("@annotation(cacheableAsync)")
    fun aroundSuspend(pjp: ProceedingJoinPoint, cacheableAsync: CacheableAsync): Any? {
        val cacheOperation = getOperationContext(cacheableAsync)
        val invocationContext = invocationContextFactory.createInvocationContext<Any, Any>(pjp)
        val invocationData = invocationDataExtractor.extractInvocationData(pjp, invocationContext)
        val key = cacheOperation.generateKey(invocationData)
        val cache = cacheOperation.getCache(invocationData)
        return invocationContext.invokeCacheable(key, cache)
    }
}
```

```
@Aspect
class CacheableAsyncAspect(
    processor: CacheAnnotationProcessor,
    private val invocationContextFactory: InvocationContextFactory,
    private val invocationDataExtractor: InvocationDataExtractor,
) {

3) private val getOperationContext = processor::createCacheableOperation.memoize()

1) @Around("@annotation(cacheableAsync)")
    fun aroundSuspend(pjp: ProceedingJoinPoint, cacheableAsync: CacheableAsync): Any? {
2)      val cacheOperation = getOperationContext(cacheableAsync)
4)      val invocationContext = invocationContextFactory.createInvocationContext<Any, Any>(pjp)
        val invocationData = invocationDataExtractor.extractInvocationData(pjp, invocationContext)
        val key = cacheOperation.generateKey(invocationData)
        val cache = cacheOperation.getCache(invocationData)
5)      return invocationContext.invokeCacheable(key, cache)
    }
}
```

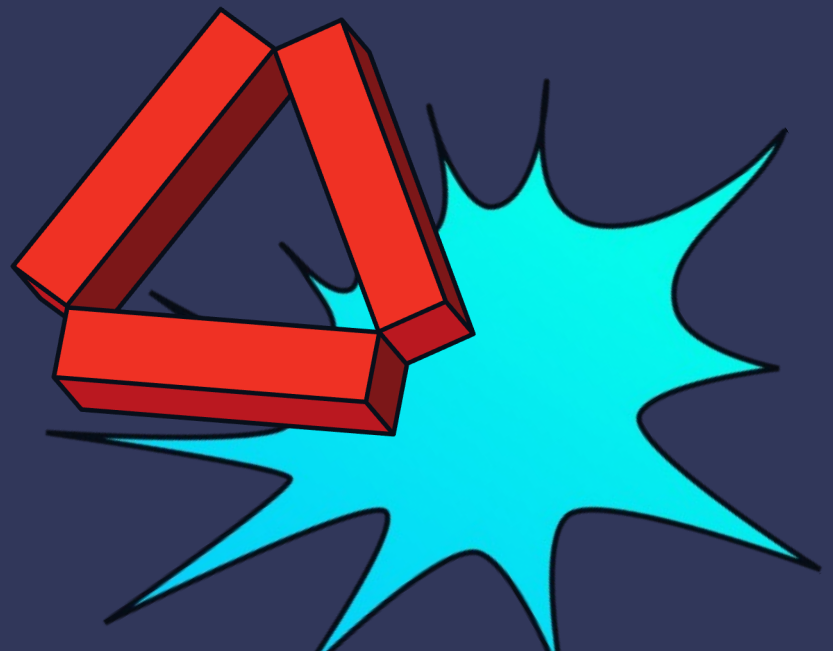
```kotlin
fun <K, V> ((K) -> V).memoize(
    memoizationMap: ConcurrentHashMap<K, V> = ConcurrentHashMap()
): (K) -> V =
    { key -> memoizationMap.computeIfAbsent(key, mappingFunction: this) }
```

```
@Aspect
class CacheableAsyncAspect(
    processor: CacheAnnotationProcessor,
    private val invocationContextFactory: InvocationContextFactory,
    private val invocationDataExtractor: InvocationDataExtractor,
) {

3) private val getOperationContext = processor::createCacheableOperation.memoize()

1) @Around("@annotation(cacheableAsync)")
    fun aroundSuspend(pjp: ProceedingJoinPoint, cacheableAsync: CacheableAsync): Any? {
2)      val cacheOperation = getOperationContext(cacheableAsync)
4)      val invocationContext = invocationContextFactory.createInvocationContext<Any, Any>(pjp)
        val invocationData = invocationDataExtractor.extractInvocationData(pjp, invocationContext)
        val key = cacheOperation.generateKey(invocationData)
        val cache = cacheOperation.getCache(invocationData)
5)      return invocationContext.invokeCacheable(key, cache)
    }
}
```

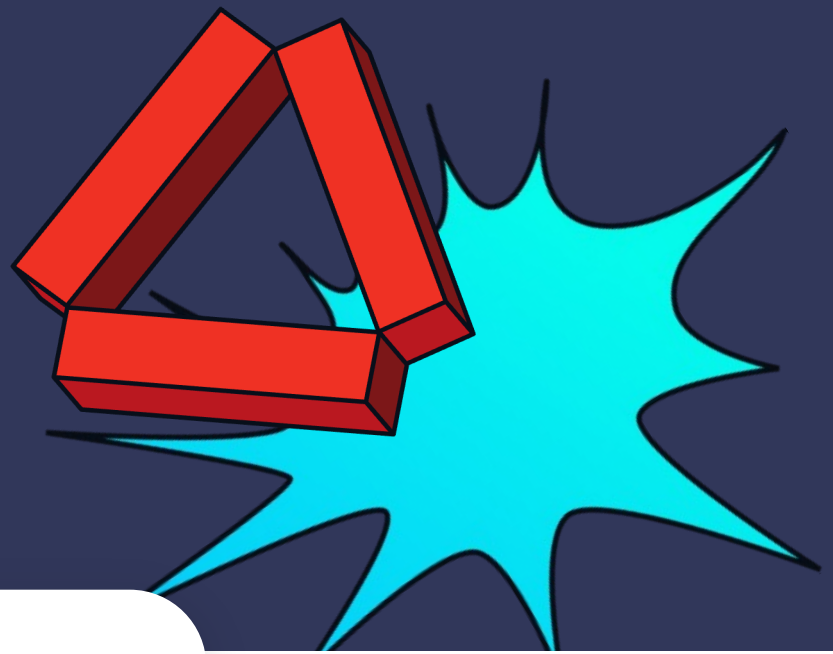
ЧТО
В ИТОГЕ



1

Абстрактного слоя кеша для асинхронных запросов в Spring не предвидится

ЧТО
В ИТОГЕ



1

Абстрактного слоя кеша
для асинхронных запросов в Spring не предвидится

2

Сделать свое решение
Не сложно и получаем гармонизированное решение для компании

ЧТО
В ИТОГЕ

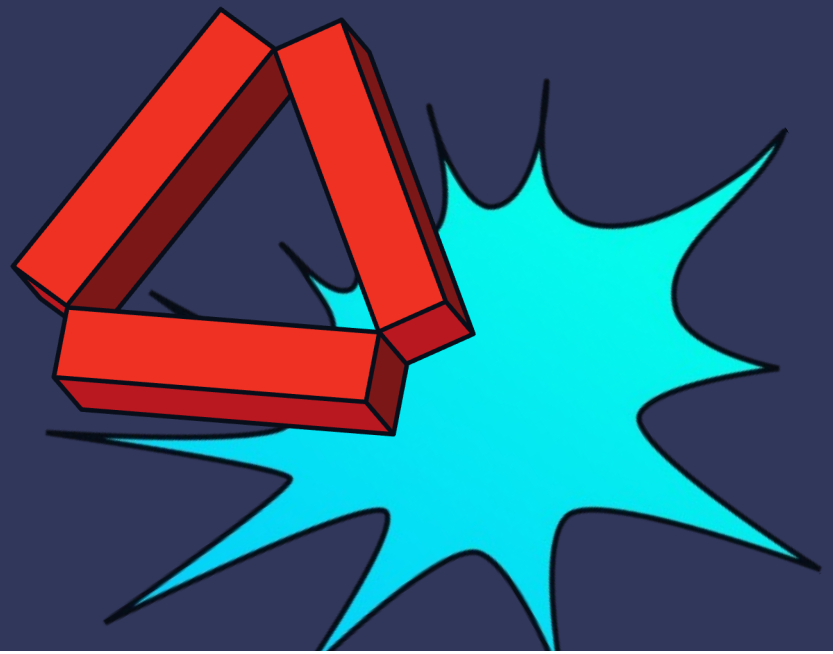
1

Абстрактного слоя кеша
для асинхронных запросов в Spring не предвидится

2

Сделать свое решение
Не сложно и получаем гармонизированное решение для компании

3

Использовать несовершенные готовые решения
Ведет к неочевидным проблемам

Спасибо
ВСЕМ!