《Java 性能优化实战 21 讲》

李国 前京东、陌陌高级架构师

— 拉勾教育出品 —

09 | 案例分析：池化对象的应用场景

案例分析：池化对象的应用场景

在编码中，通常会将一些对象保存起来，这主要考虑的是**对象的创建成本**

显著的特征——通过轻量级的重置工作，可以循环、重复地使用

可以使用一个虚拟的池子，将这些资源保存起来，当使用的时候，从池子里快速获取一个

在 Java 中，常见的**池化技术**有数据库连接池、线程池等

公用池化包 Commons Pool 2.0

GenericObjectPool是对象池的核心类

通过传入一个对象池的配置和一个对象的工厂，即可快速创建对象池

```
public GenericObjectPool(
        final PooledObjectFactory<T> factory,
        final GenericObjectPoolConfig<T> config)
```

公用池化包 Commons Pool 2.0

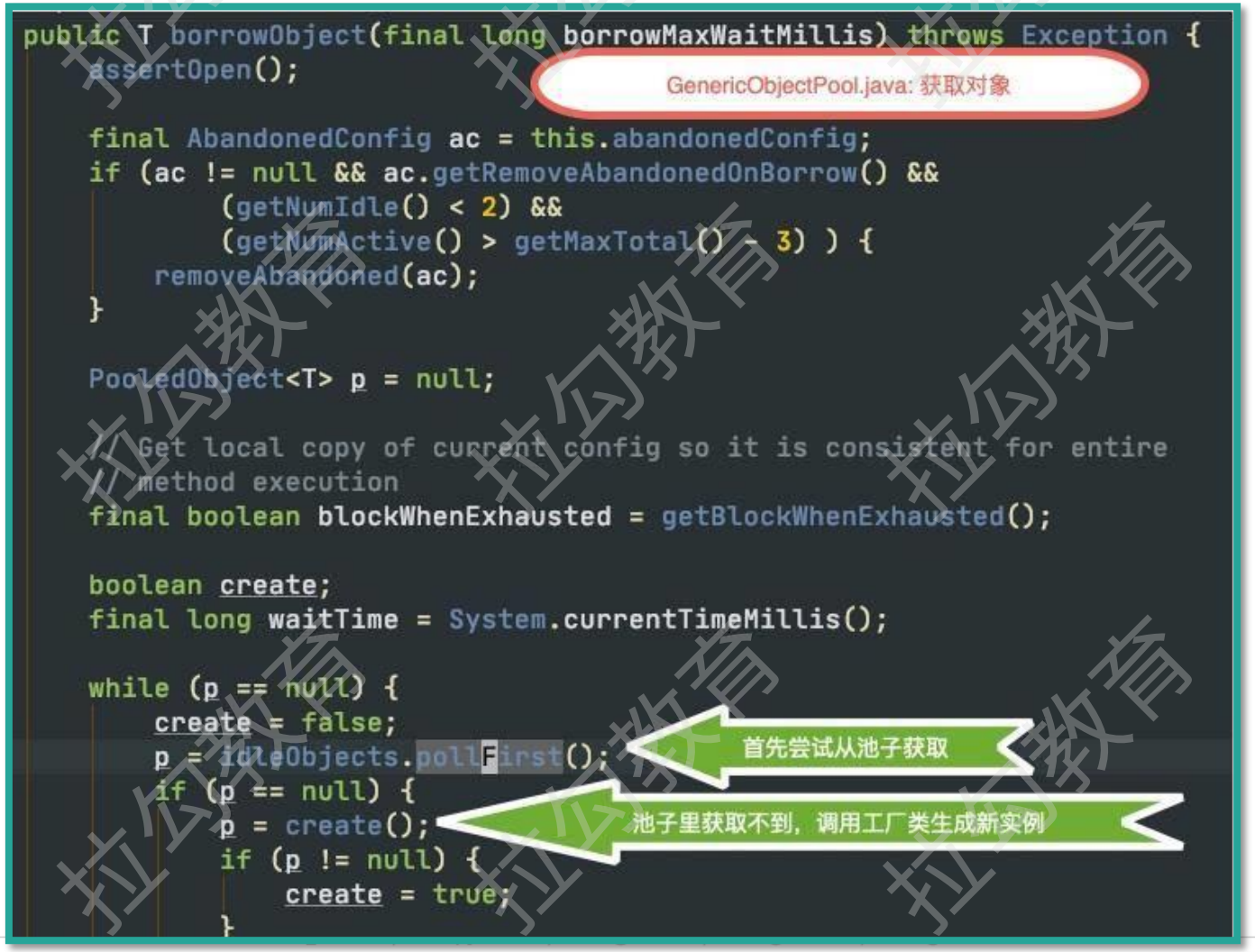

```java
@Override
public PooledObject<Jedis> makeObject() throws Exception {
  final HostAndPort hp = this.hostAndPort.get();
  final Jedis jedis = new Jedis(hp.getHost(), hp.getPort(), connectionTimeout, soTimeout,
      ssl, sslSocketFactory, sslParameters, hostnameVerifier);
  try {
    jedis.connect();
    if (user != null) {
      jedis.auth(user, password);
    } else if (password != null) {
      jedis.auth(password);
    }
    if (database != 0) {
      jedis.select(database);
    }
    if (clientName != null) {
      jedis.clientSetname(clientName);
    }
  } catch (JedisException je) {
    jedis.close();
    throw je;
  }

  return new DefaultPooledObject<>(jedis);
}
```

JedisFactory.java: Jedis使用工厂创建对象

耗时操作

返回包装对象

公用池化包 Commons Pool 2.0




```
public T borrowObject(final long borrowMaxWaitMillis) throws Exception {
    assertOpen();

    final AbandonedConfig ac = this.abandonedConfig;
    if (ac != null && ac.getRemoveAbandonedOnBorrow() &&
            (getNumIdle() < 2) &&
            (getNumActive() > getMaxTotal() - 3) ) {
        removeAbandoned(ac);
    }

    PooledObject<T> p = null;

    // Get local copy of current config so it is consistent for entire
    // method execution
    final boolean blockWhenExhausted = getBlockWhenExhausted();

    boolean create;
    final long waitTime = System.currentTimeMillis();

    while (p == null) {
        create = false;
        p = idleObjects.pollFirst();
        if (p == null) {
            p = create();
            if (p != null) {
                create = true;
            }
```

公用池化包 Commons Pool 2.0

对象是存在什么地方的呢？

由 LinkedBlockingDeque的结构来承担的

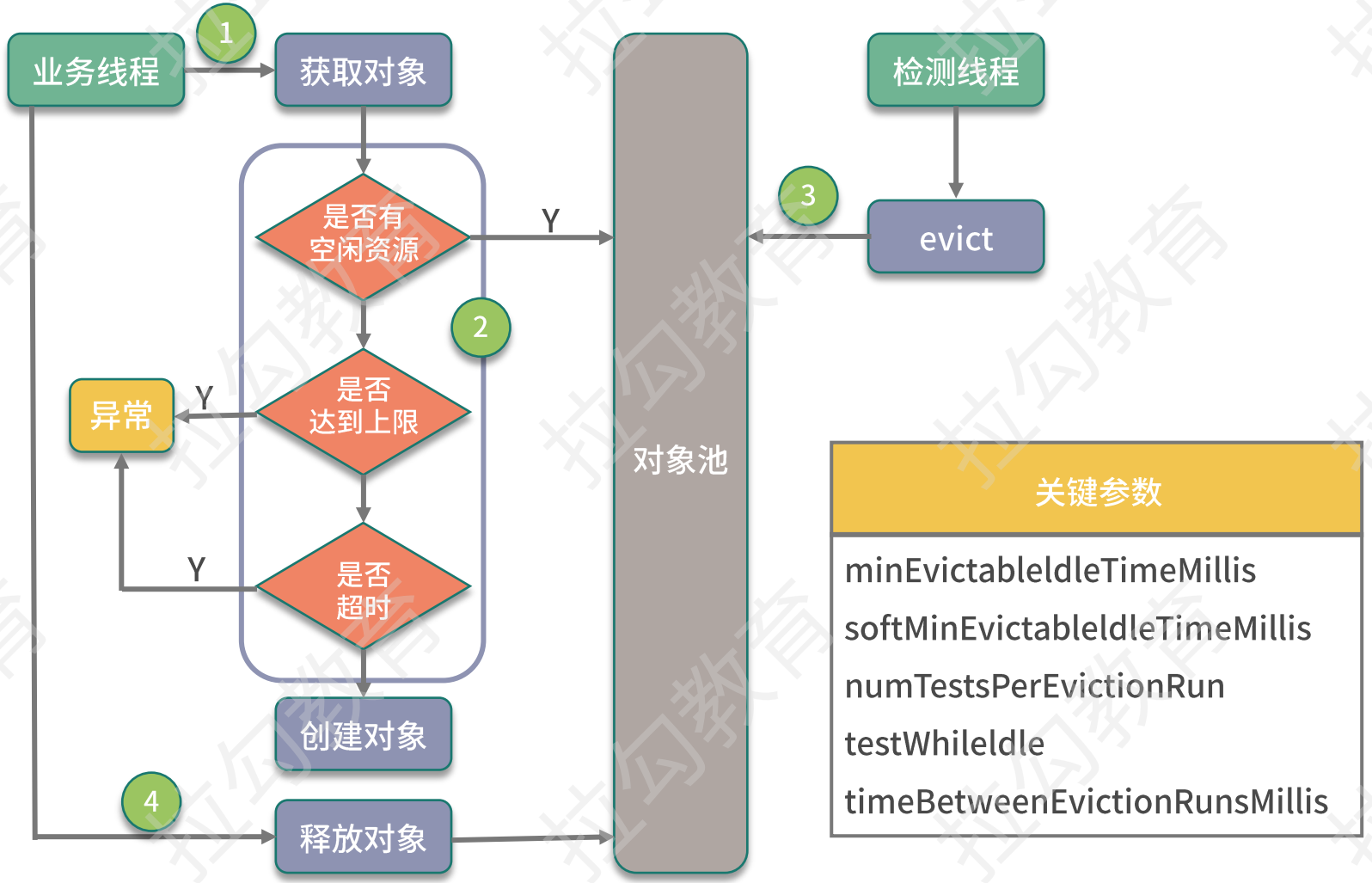
它是一个双向的队列

公用池化包 Commons Pool 2.0

拉勾教育
互联网人实战大学

```
private int maxTotal = DEFAULT_MAX_TOTAL;
private int maxIdle = DEFAULT_MAX_IDLE;
private int minIdle = DEFAULT_MIN_IDLE;

private boolean lifo = DEFAULT_LIFO;
private boolean fairness = DEFAULT_FAIRNESS;
private long maxWaitMillis = DEFAULT_MAX_WAIT_MILLIS;
private long minEvictableIdleTimeMillis = DEFAULT_MIN_EVICTABLE_IDLE_TIME_MILLIS;
private long evictorShutdownTimeoutMillis = DEFAULT_EVICTOR_SHUTDOWN_TIMEOUT_MILLIS;
private long softMinEvictableIdleTimeMillis = DEFAULT_SOFT_MIN_EVICTABLE_IDLE_TIME_MILLIS;
private int numTestsPerEvictionRun = DEFAULT_NUM_TESTS_PER_EVICTION_RUN;
private EvictionPolicy<T> evictionPolicy = null; // Only 2.6.0 applications set this
private String evictionPolicyClassName = DEFAULT_EVICTION_POLICY_CLASS_NAME;
private boolean testOnCreate = DEFAULT_TEST_ON_CREATE;
private boolean testOnBorrow = DEFAULT_TEST_ON_BORROW;
private boolean testOnReturn = DEFAULT_TEST_ON_RETURN;
private boolean testWhileIdle = DEFAULT_TEST_WHILE_IDLE;
private long timeBetweenEvictionRunsMillis = DEFAULT_TIME_BETWEEN_EVICTION_RUNS_MILLIS;
private boolean blockWhenExhausted = DEFAULT_BLOCK_WHEN_EXHAUSTED;
```

公用池化包 Commons Pool 2.0



关键参数
minEvictableIdleTimeMillis
softMinEvictableIdleTimeMillis
numTestsPerEvictionRun
testWhileIdle
timeBetweenEvictionRunsMillis

公用池化包 Commons Pool 2.0

对象池在进行初始化时，要指定三个主要的参数：

- **maxTotal**——对象池中管理的对象上限
- **maxIdle**——最大空闲数
- **minIdle**——最小空闲数

公用池化包 Commons Pool 2.0

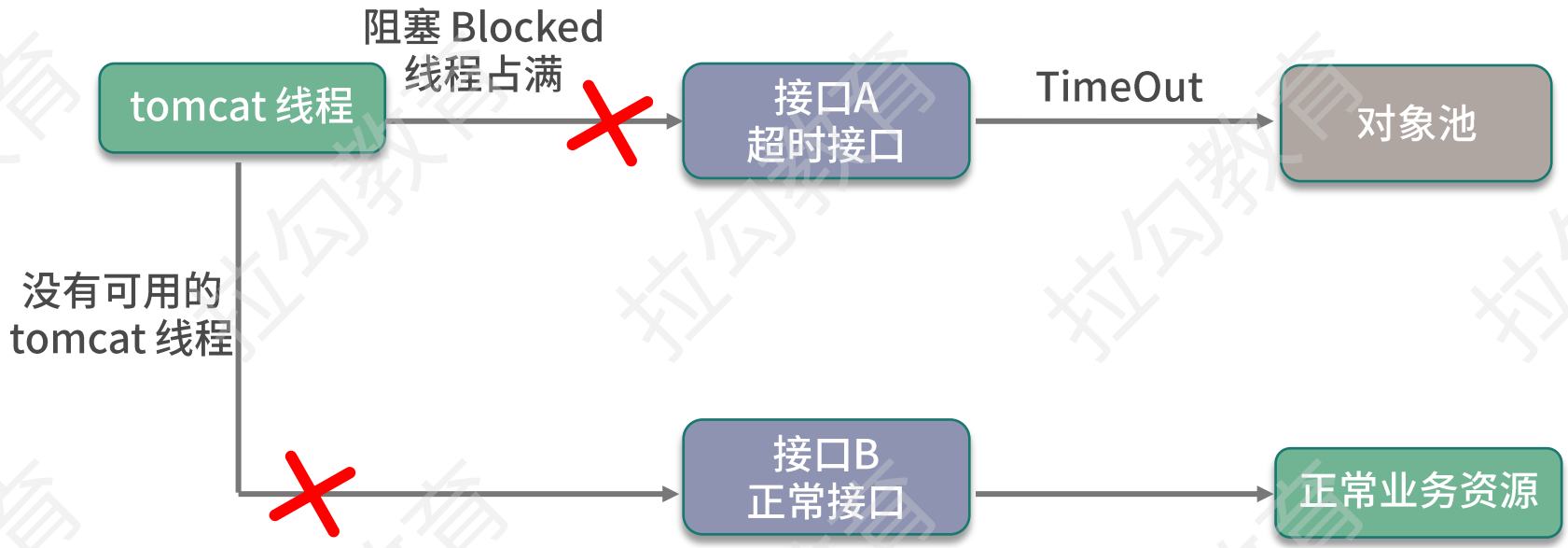
当业务线程获取对象时，会首先检测是否有空闲的对象

- 如果有，则返回一个
- 否则进入创建逻辑。如果池中个数已经达到了最大值，会创建失败，返回空对象

最大等待时间（maxWaitMillis）

该参数默认为 -1，表示永不超时，直到有对象空闲

公用池化包 Commons Pool 2.0



公用池化包 Commons Pool 2.0

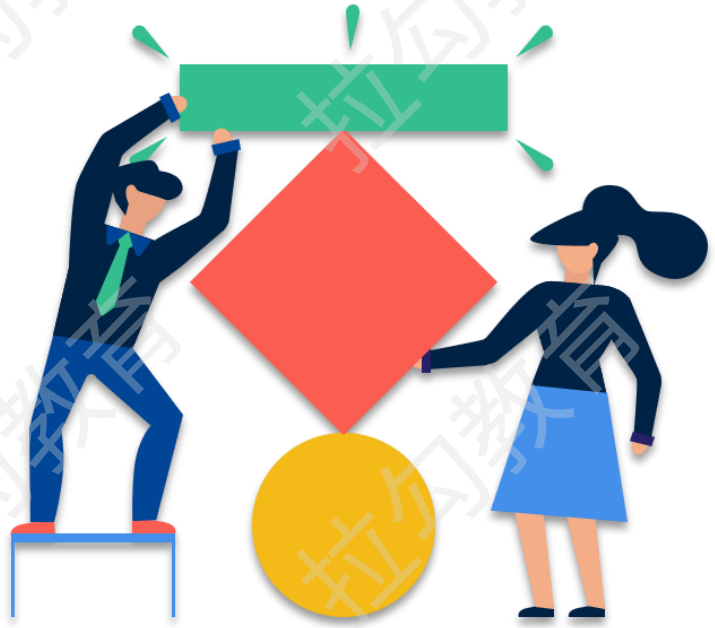
公用池化包 Commons Pool 2.0

把最大等待时间，设置成接口可以忍受的最大延迟

比如，一个正常服务响应时间 10ms 左右，达到 1 秒钟就会感觉到卡顿

参数可以设置成 500~1000ms

超时之后，会抛出 NoSuchElementException 异常，请求会快速失败



公用池化包 Commons Pool 2.0

带有 evcit 字样的参数——处理对象逐出

连接池会占用多条连接，线程池会增加调度开销等

业务在突发流量下，会申请到超出正常情况的对象资源放在池子中，等对象不再被使用，把它清理掉

超出 minEvictableIdleTimeMillis 参数指定值的对象，会被强制回收掉，值默认是 30 分钟

softMinEvictableIdleTimeMillis 参数在当前对象数量大于 minIdle 时会执行移除

公用池化包 Commons Pool 2.0

4 个 test 参数：testOnCreate、testOnBorrow、testOnReturn、testWhileIdle

分别指定了在创建、获取、归还、空闲检测时，是否对池化对象进行有效性检测

生产环境上，建议只将 testWhileIdle 设置为 true

并通过调整空闲检测时间间隔（timeBetweenEvictionRunsMillis），来保证资源的可用性和效率

Jedis JMH 测试

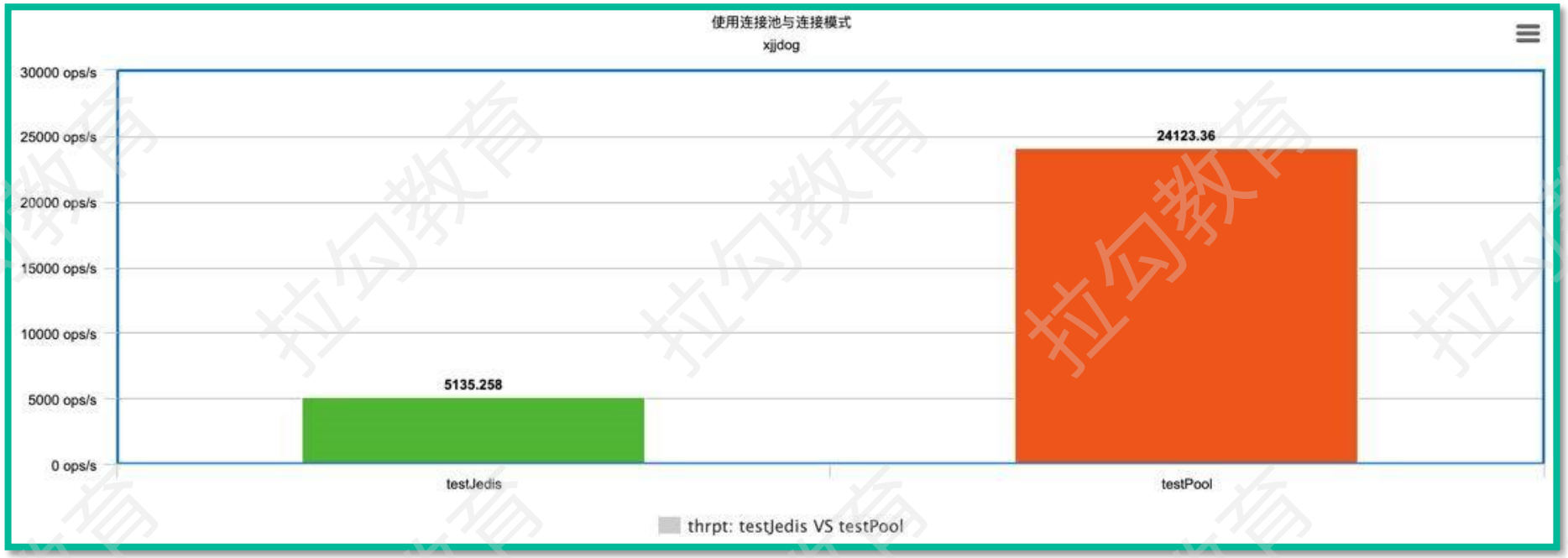
使用连接池和不使用连接池

它们之间的性能差距到底有多大呢？

Jedis JMH 测试


```
@Fork(2)
@State(Scope.Benchmark)
@Warmup(iterations = 5, time = 1)
@Measurement(iterations = 5, time = 1)
@BenchmarkMode(Mode.Throughput)
public class JedisPoolVSJedisBenchmark {
    JedisPool pool = new JedisPool("localhost", 6379);

    @Benchmark
    public void testPool() {
        Jedis jedis = pool.getResource();
        jedis.set("a", UUID.randomUUID().toString());
        jedis.close();
    }

    @Benchmark
    public void testJedis() {
        Jedis jedis = new Jedis("localhost", 6379);
        jedis.set("a", UUID.randomUUID().toString());
        jedis.close();
    }
...
```

Jedis JMH 测试

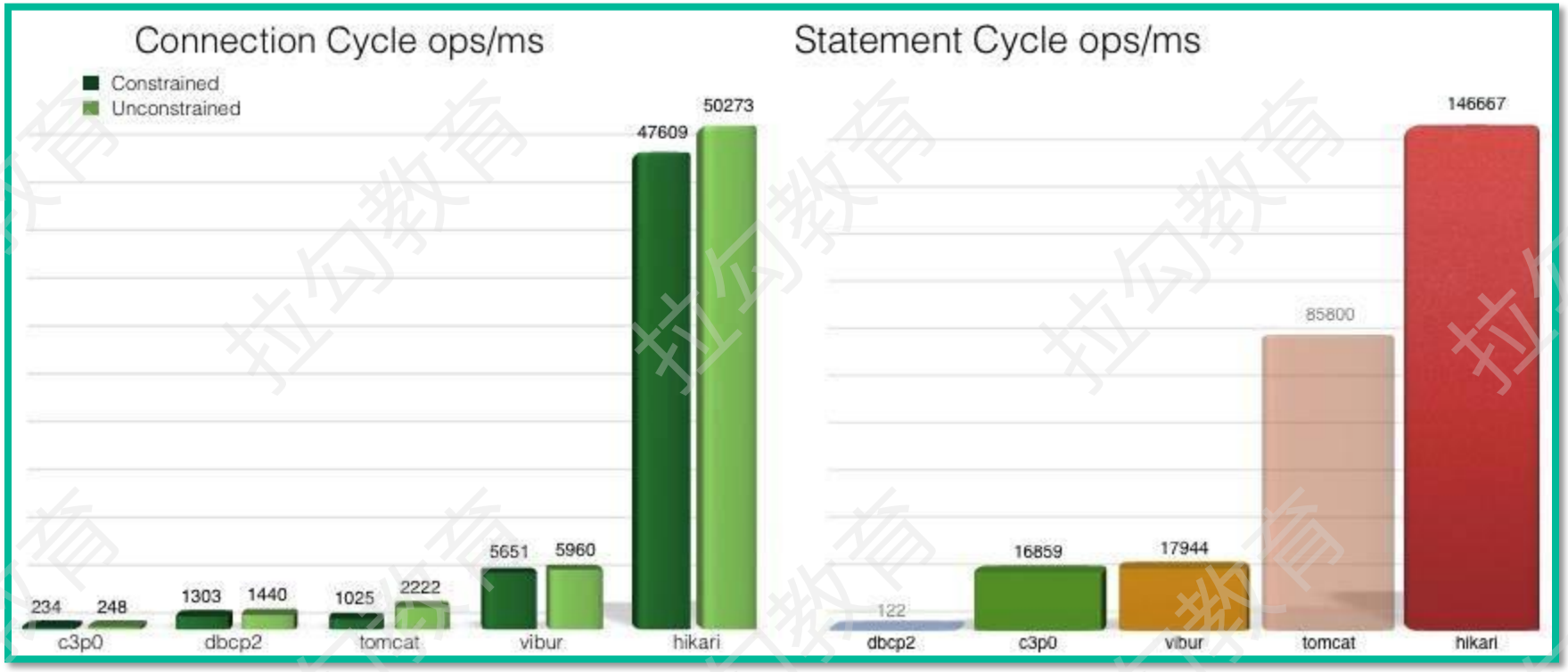


数据库连接池 HikariCP

HikariCP 是 SpringBoot 中默认的数据库连接池

它可以有效地减少数据库连接创建、销毁的资源消耗

数据库连接池 HikariCP



数据库连接池 HikariCP

数据库连接池 HikariCP

- 它使用 FastList 替代 ArrayList，通过初始化的默认值，减少了越界检查的操作
- 优化并精简了字节码，通过使用 Javassist，减少了动态代理的性能损耗

  比如使用 invokestatic 指令代替 invokevirtual 指令
- 实现了无锁的 ConcurrentBag，减少了并发场景下的锁竞争

数据库连接池 HikariCP

连接池的大小设置得越大越好

甚至把这个值设置成 1000 以上

数据库连接池 HikariCP

拉勾教育
互联网人实战大学

minimumIdle 值被默认设置成和 maximumPoolSize 一样的大小

如果数据库 Server 端连接资源空闲较大，可以去掉连接池的动态调整功能

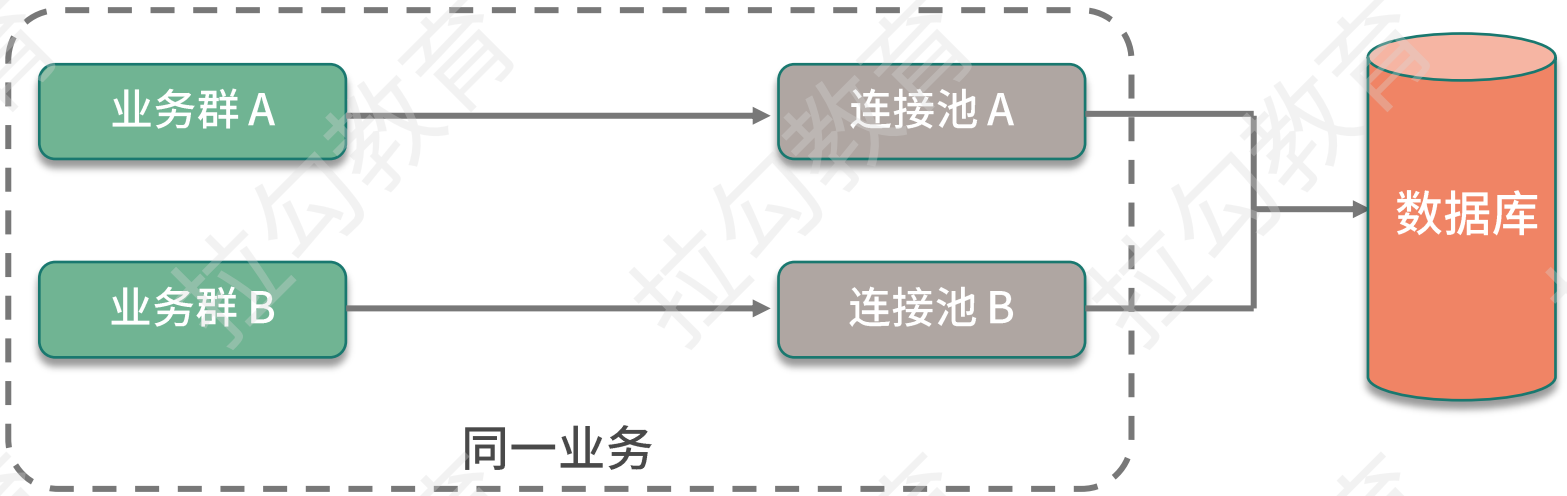
根据数据库查询和事务类型，一个应用中可以配置多个数据库连接池的

数据库连接池 HikariCP

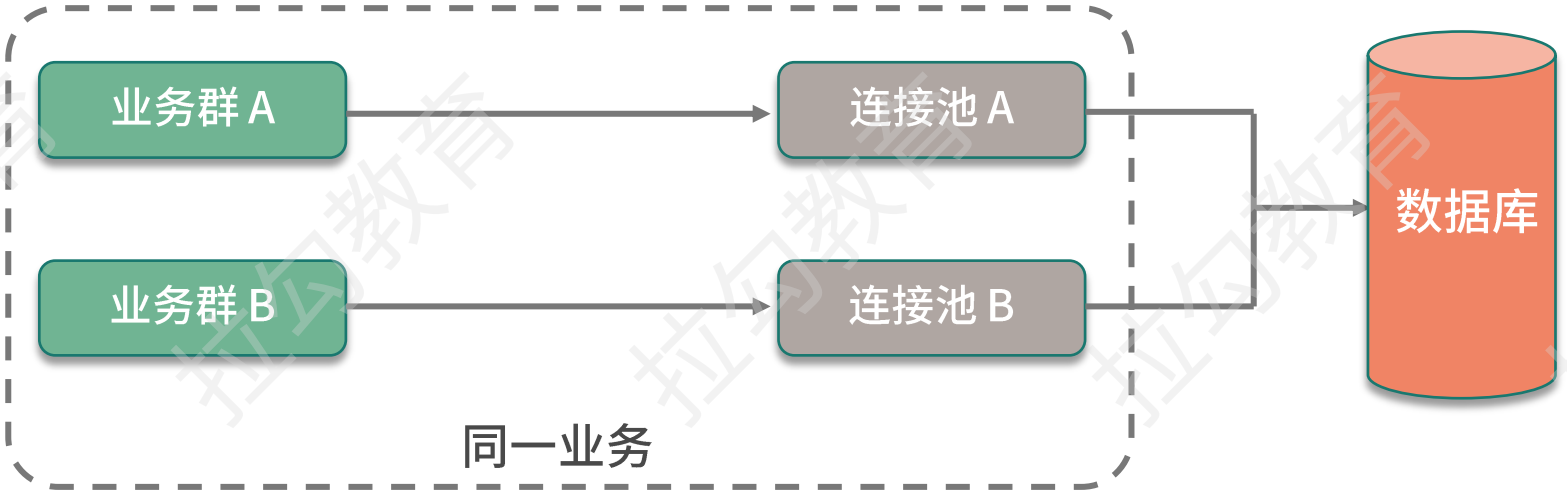
业务类型通常有两种：

- 需要快速的响应时间，把数据尽快返回给用户
- 可以在后台慢慢执行，耗时比较长，对时效性要求不高

数据库连接池 HikariCP



数据库连接池 HikariCP



在 JDBC4 的协议中，通过 Connection.isValid() 可以检测连接的有效性

结果缓存池

池（Pool）与缓存（Cache）之间的一个共同点

将对象加工后，存储在相对高速的区域

结果缓存池

- jsp 提供了网页的动态功能，可以在执行后，编译成 class 文件，加快执行速度
- 一些媒体平台，会将热门文章，定时转化成静态的 html 页面

  仅靠 nginx 的负载均衡即可应对高并发请求（动静分离）

结果缓存池（Result Cache Pool）

保存了某个执行步骤的结果，使得下次访问时不需要从头再来

小结

- 介绍了公用池化包 Commons Pool 2.0 的一些实现细节，并对一些重要参数的应用做了讲解

  Jedis 是在 Commons Pool 2.0 的基础上封装的

- 介绍了数据库连接池中速度速快的 **HikariCP**

小结

当你遇到下面的场景，可以考虑使用池化来增加系统性能：

- 对象的创建或者销毁，需要耗费较多的系统资源
- 对象的创建或者销毁，耗时长，需要繁杂的操作和较长时间的等待
- 对象创建后，通过一些状态重置，可以被反复使用

小结

小结

比如 Http 连接池，Okhttp 和 Httpclient 都提供了连接池的概念

在底层的中间件，比如 RPC，通常使用连接池技术加速资源获取

比如 Dubbo 连接池、 Feign 切换成 httppclient 的实现等技术

小结

线程池

通过队列对任务进行了二层缓冲

提供了多样的拒绝策略

思考&&探讨

线程池的特性可以借鉴到连接池技术中

用来缓解请求溢出，创建溢出策略

具体怎么做？有哪些做法呢？

Next：10 | 《案例分析：大对象复用的目标和注意点》

拉勾教育

—互联网人实战大学—

关注拉勾「教育公众号」
获取更多课程信息

L / A / G / O / U