vmware | SPRING & CLOUD FOUNDRY 开发者大会

springone CHINA

中国北京 - 2012年12月7日-8日

开发多语言持久性应用程序

Chris Richardson,
《POJOs in Action》作者，CloudFoundry.com 原创始人

@crichardson crichardson@vmware.com http://plainoldobjects.com

演讲目的

多语言持久性的优缺点

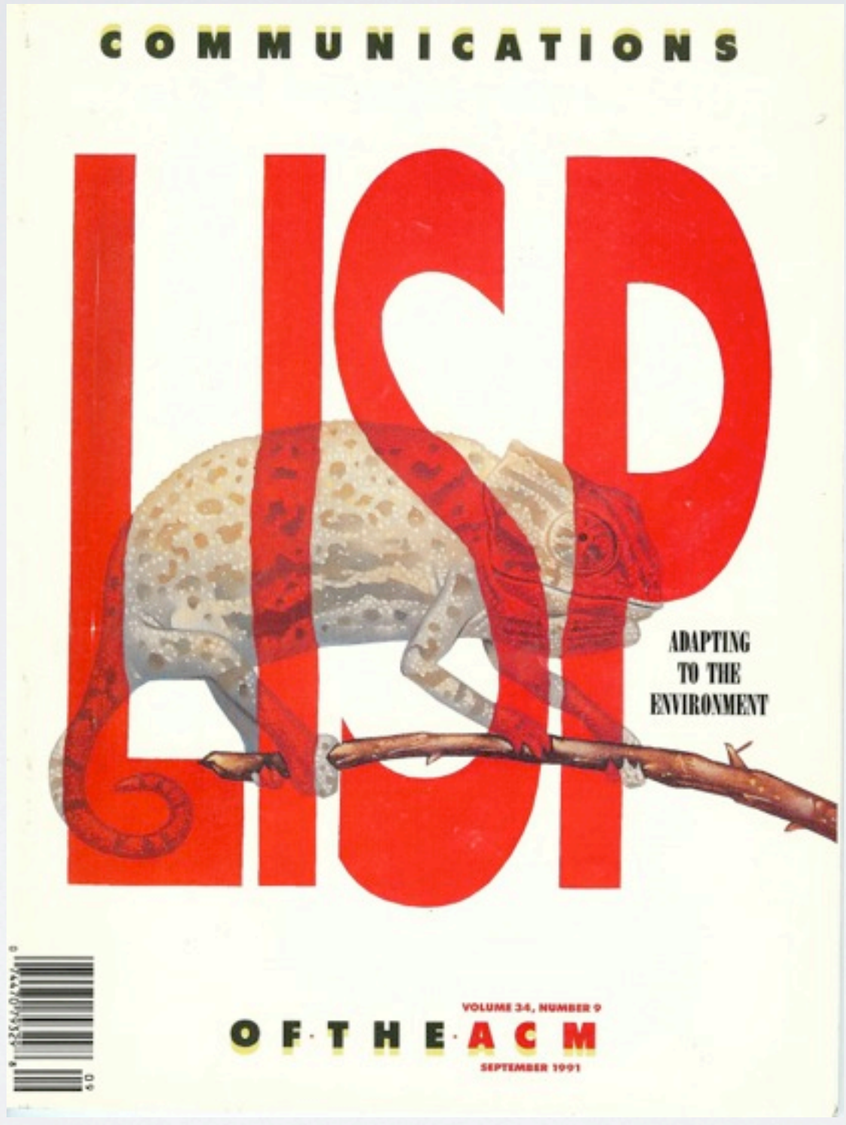
如何设计使用此方法的应用程序

Chris 介绍

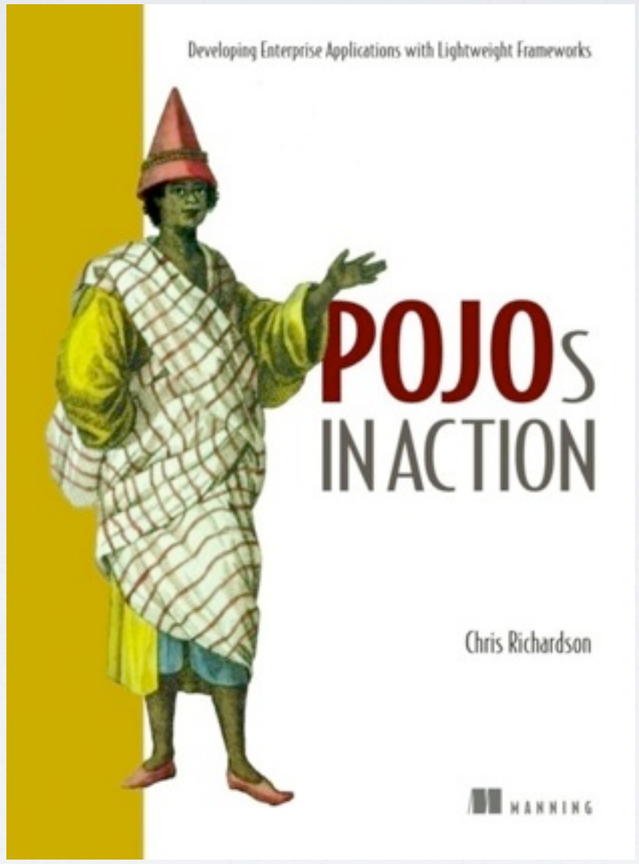

Aberdeen
Dundee
Glasgow
Londonderry / Derry
Belfast
United Kingdom
Hartlepool
Isle of Man
Leeds
Preston
Hull
Dublin
Ireland
Manchester
Leicester
Norwich
Birmingham
Ipswich
Northampton
Swansea
Barnet
Basildon
Bristol
A
Exeter
Portsmouth
Eastbourne
Plymouth


Medford
Klamath Falls
Yreka
Eureka
Redding
Elko
Chico
Reno
Nevada
Yuba City
Carson City
A
Sacramento
Santa Rosa
San Francisco
Stockton
San Jose
California
Fresno
Visalia
Las Vegas
Bakersfield
Santa Maria
Lompoc
Palmdale
Santa Clarita
Riverside
Hemet
Los Angeles
Oceanside
Yuma
Tijuana
Mexicali
Ensenada
Puerto Peñasco

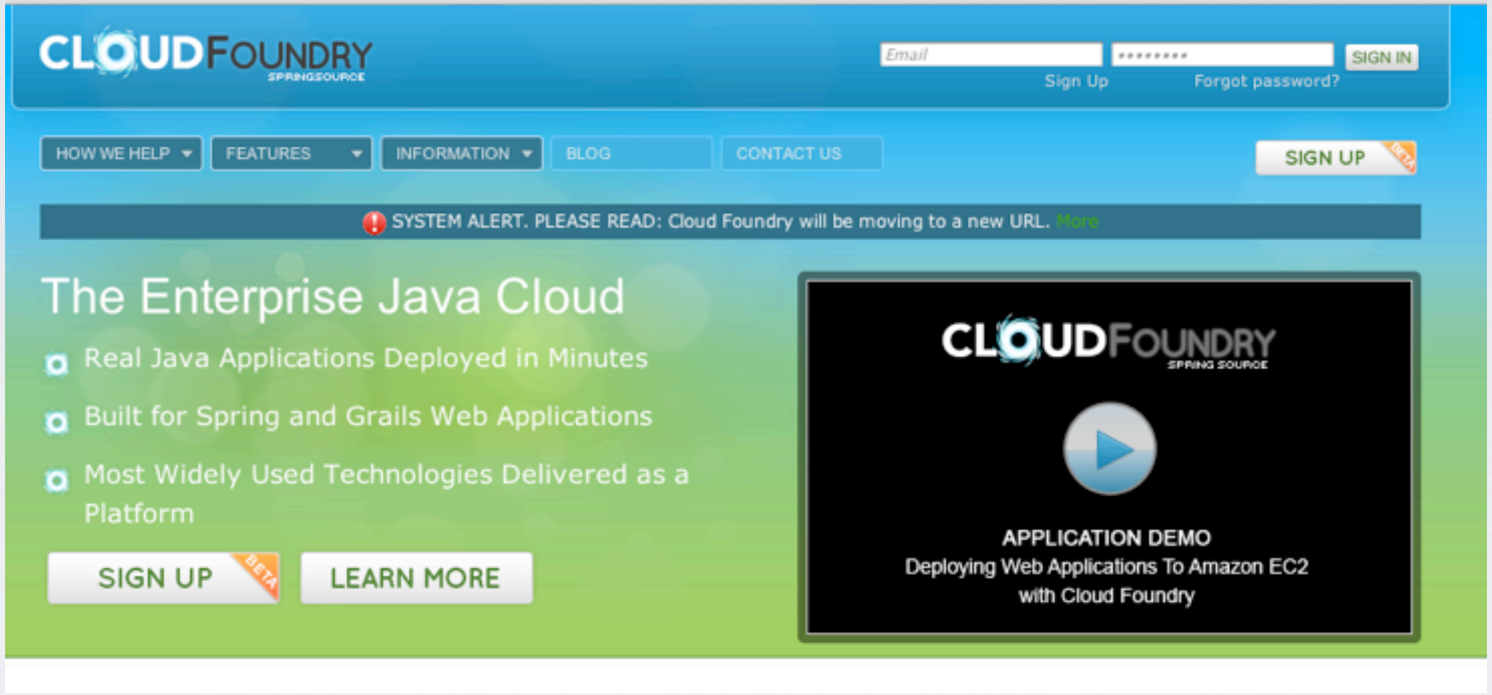
(Chris介绍)


COMMUNICATIONS
LISP
ADAPTING
TO THE
ENVIRONMENT
VOLUME 34, NUMBER 9
OF THE ACM
SEPTEMBER 1991

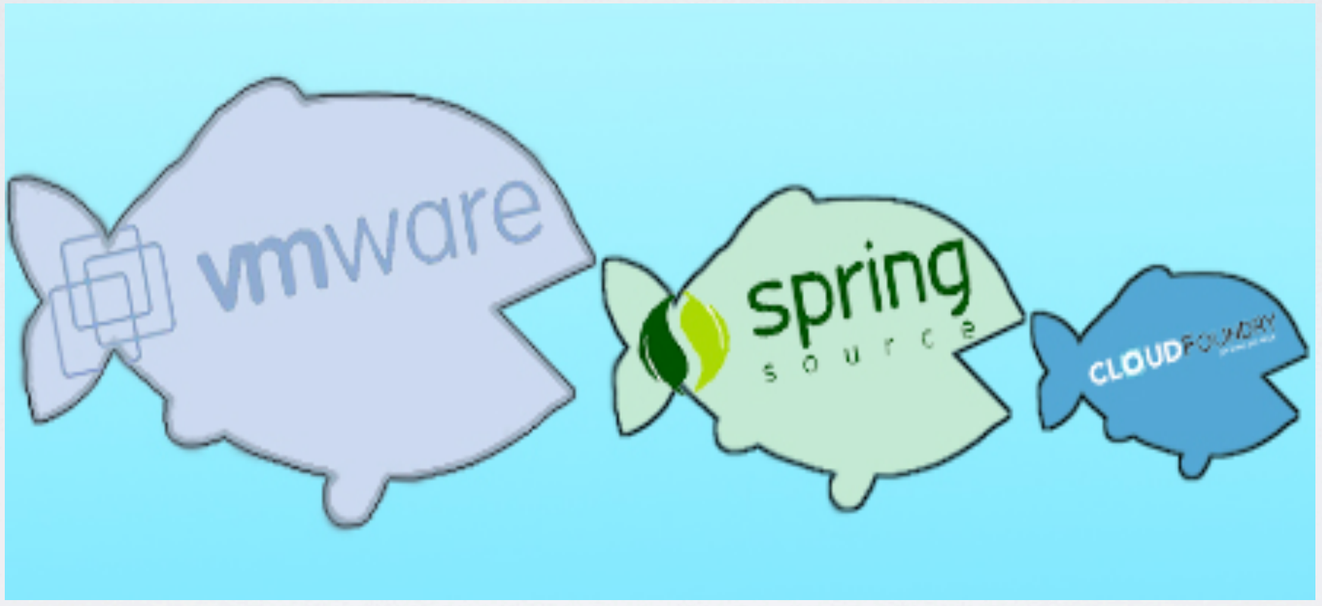
Chris介绍


Developing Enterprise Applications with Lightweight Frameworks
POJOs
IN ACTION
Chris Richardson
MANNING

Chris介绍


CLOUDFOUNDRY
SPRINGSOURCE
Email
SIGN IN
Sign Up
Forgot password?
HOW WE HELP
FEATURES
INFORMATION
BLOG
CONTACT US
SIGN UP
BETA
SYSTEM ALERT. PLEASE READ: Cloud Foundry will be moving to a new URL. More
The Enterprise Java Cloud
Real Java Applications Deployed in Minutes
Built for Spring and Grails Web Applications
Most Widely Used Technologies Delivered as a Platform
SIGN UP
BETA
LEARN MORE
CLOUDFOUNDRY
SPRING SOURCE
APPLICATION DEMO
Deploying Web Applications To Amazon EC2
with Cloud Foundry

Chris介绍



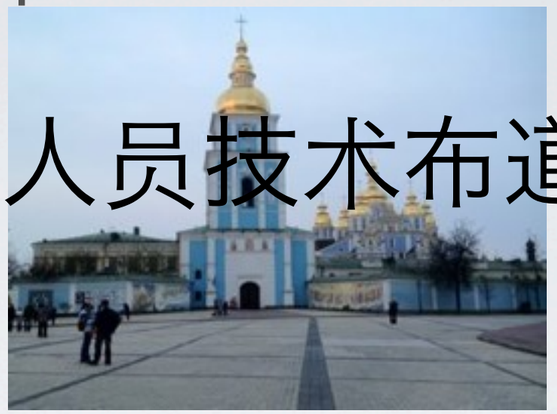
http://www.theregister.co.uk/2009/08/19/springsource_cloud_foundry/

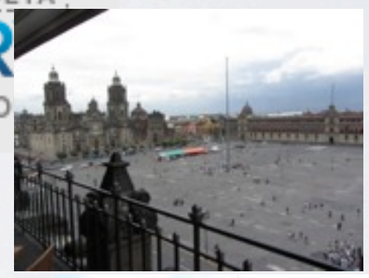
vmc push Chris介绍

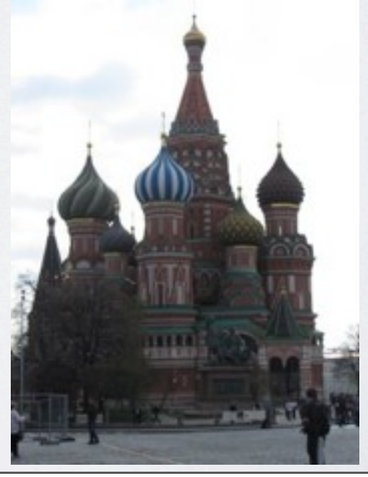
开发人员技术布道师

vmc push Chris介绍

开发人员技术布道师

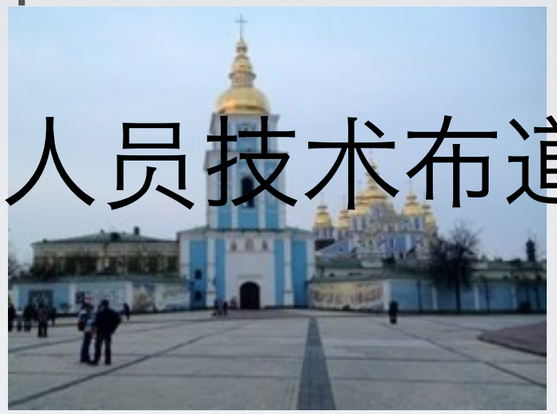


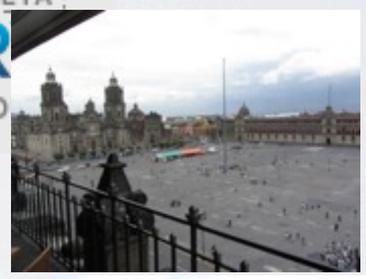




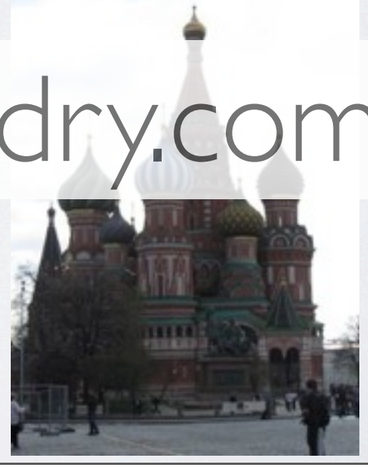
vmc push Chris介绍

开发人员技术布道师





注册网址 http://cloudfoundry.com

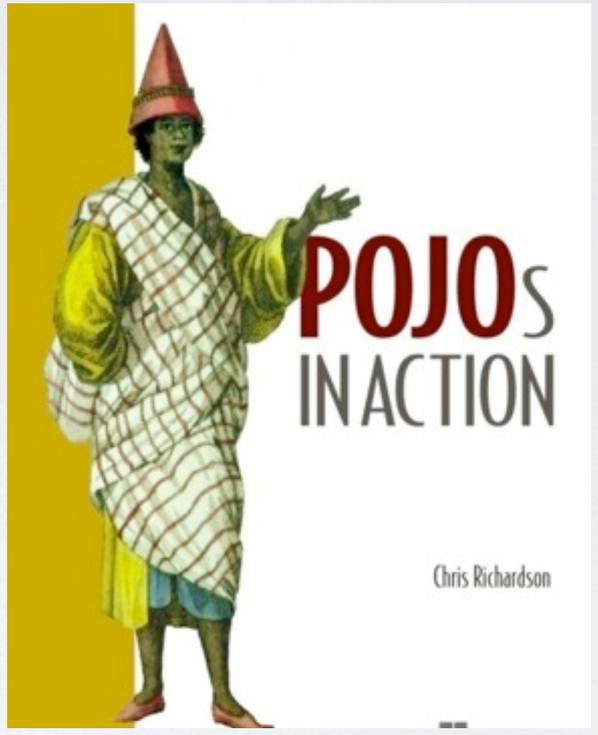


议题

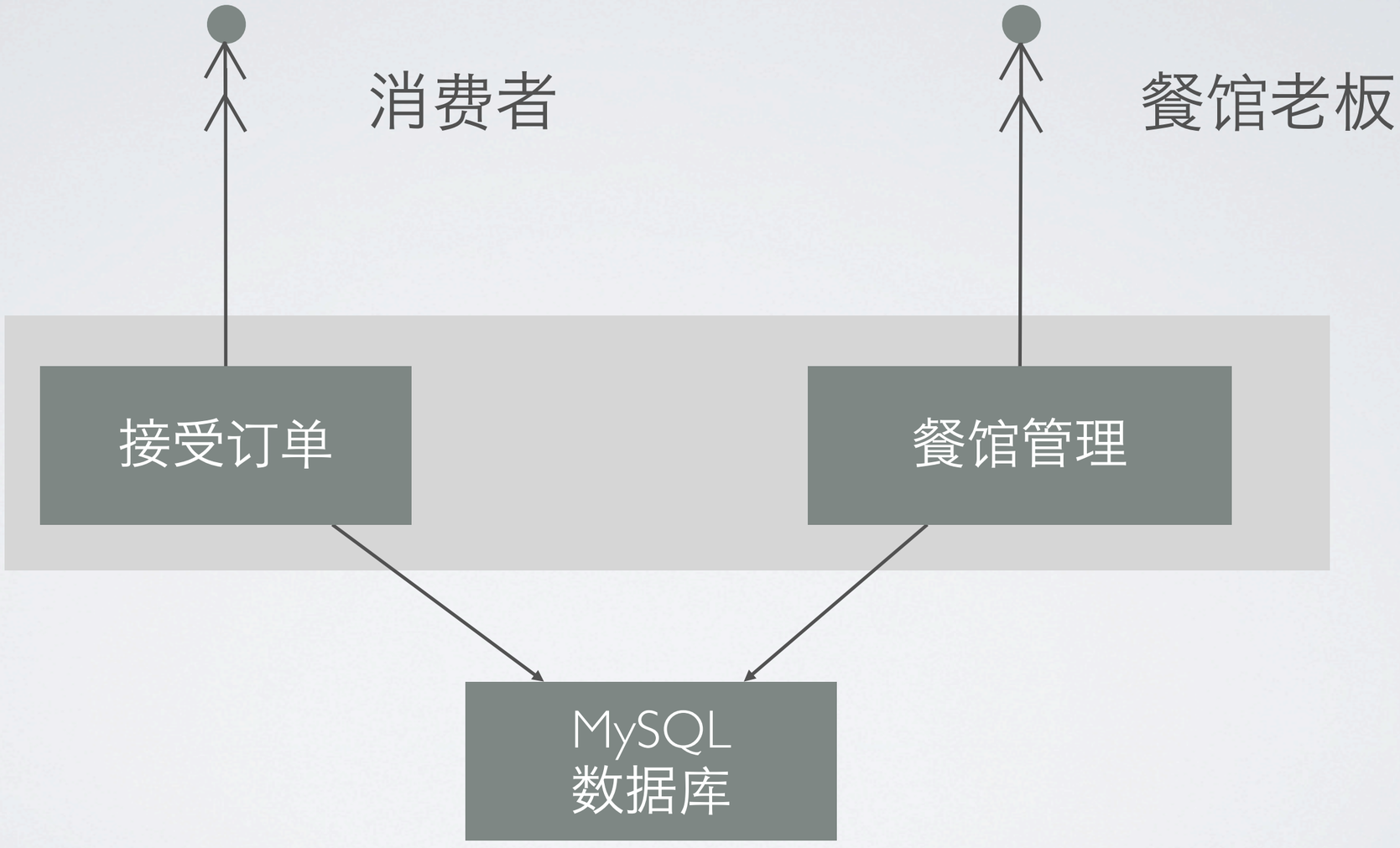
- 为何选择多语言持久性？
- 使用 Redis 作为缓存
- 使用 Redis 具体化视图优化查询
- 同步 MySQL 和 Redis
- 跟踪对实体的更改

“外卖餐馆”

- “外卖送餐服务”
- 2006 年“开张”



“外卖餐馆”的体系结构



成功 ⇒ 业务增长压力

- 不断增长的通信流量
- 不断增长的数据量
- 在多个数据中心之间分布
- 不断增加的域模型复杂性

关系数据库的局限性

关系数据库的局限性

- 可扩展性

关系数据库的局限性

- 可扩展性
- 分布

关系数据库的局限性

- 可扩展性
- 分布
- 架构更新

关系数据库的局限性

- 可扩展性
- 分布
- 架构更新
- 对象-关系阻抗不匹配

关系数据库的局限性

- 可扩展性
- 分布
- 架构更新
- 对象-关系阻抗不匹配
- 处理半结构化数据

解决方案：投入资金

http://upload.wikimedia.org/wikipedia/commons/e/e5/Rising_Sun_Yacht.JPG

还是

http://www.trekbikes.com/us/en/bikes/road/race_performance/madone_5_series/madone_5_2/#

解决方案：使用 NoSQL

- **优点**
- 性能更高
- 扩展性更强
- 数据模型更丰富
- 没有架构（schema less）

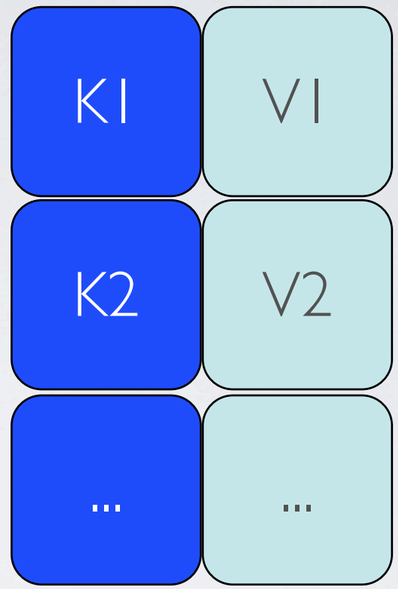
- **缺点**
- 事务受限
- 查询受限
- 一致性要求放松
- 数据无约束

NoSQL 数据库示例

数据库	主要特点
Cassandra	可扩展的列存储、可扩展性极强、分布式
Neo4j	图表数据库
MongoDB	面向文件、快速、可扩展
Redis	键值存储、速度极快

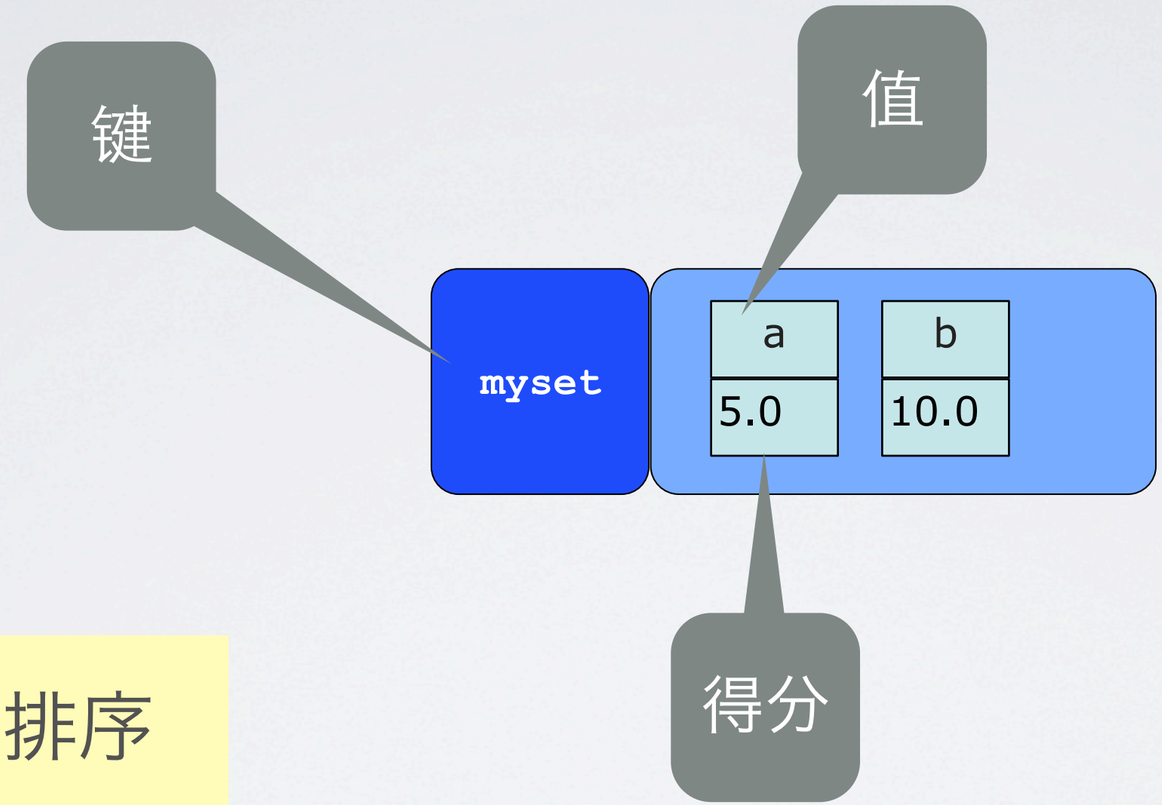
http://nosql-database.org/ 列出了超过 122 种 NoSQL 数据库

Redis



- 高级键值存储
- 速度极快，例如每秒请求数可达 100,000
- 可选持久性
- 乐观锁定事务
- 主从复制

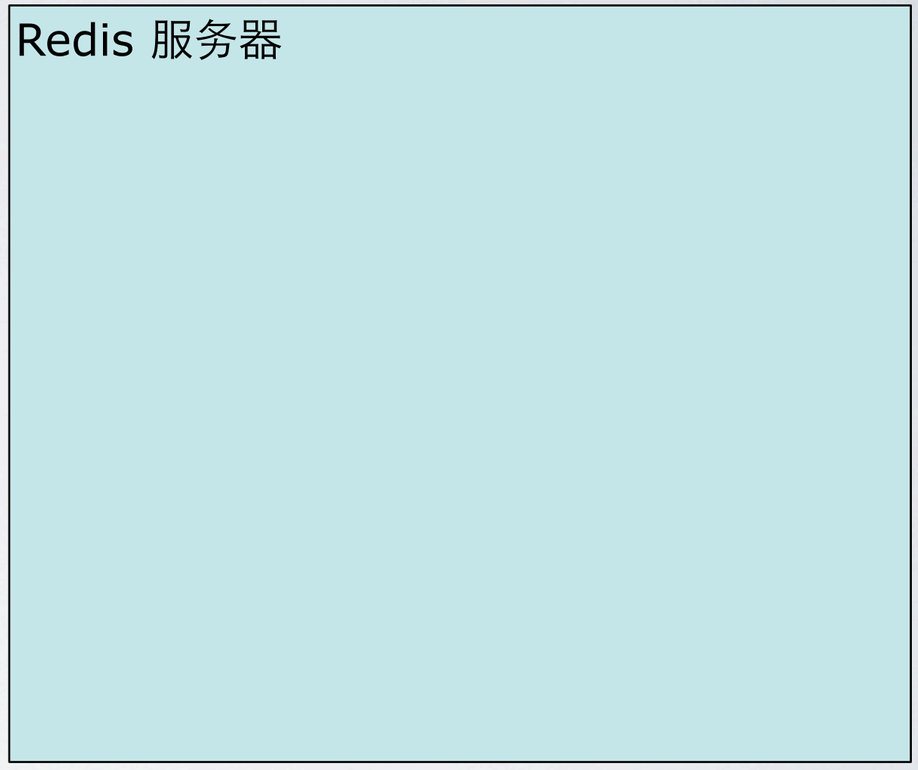
有序集



成员按得分排序

添加成员至有序集

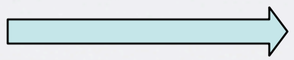

```
zadd myset 5.0 a
```



添加成员至有序集

键 得分 值

```
zadd myset 5.0 a
```

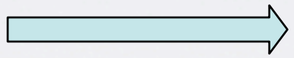


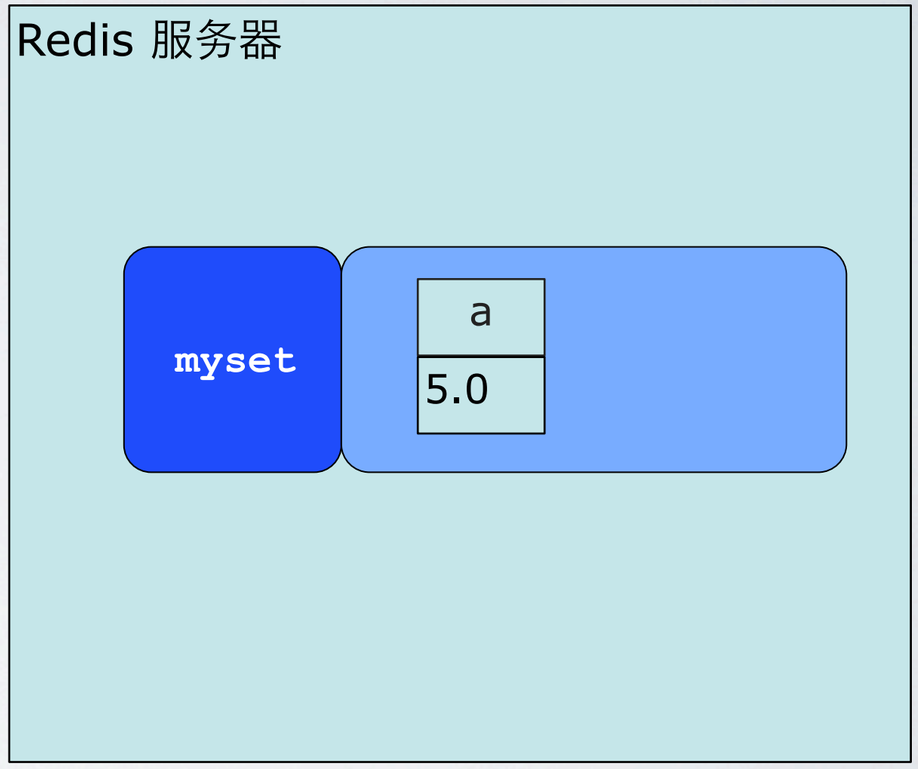
Redis 服务器

添加成员至有序集

键 得分 值

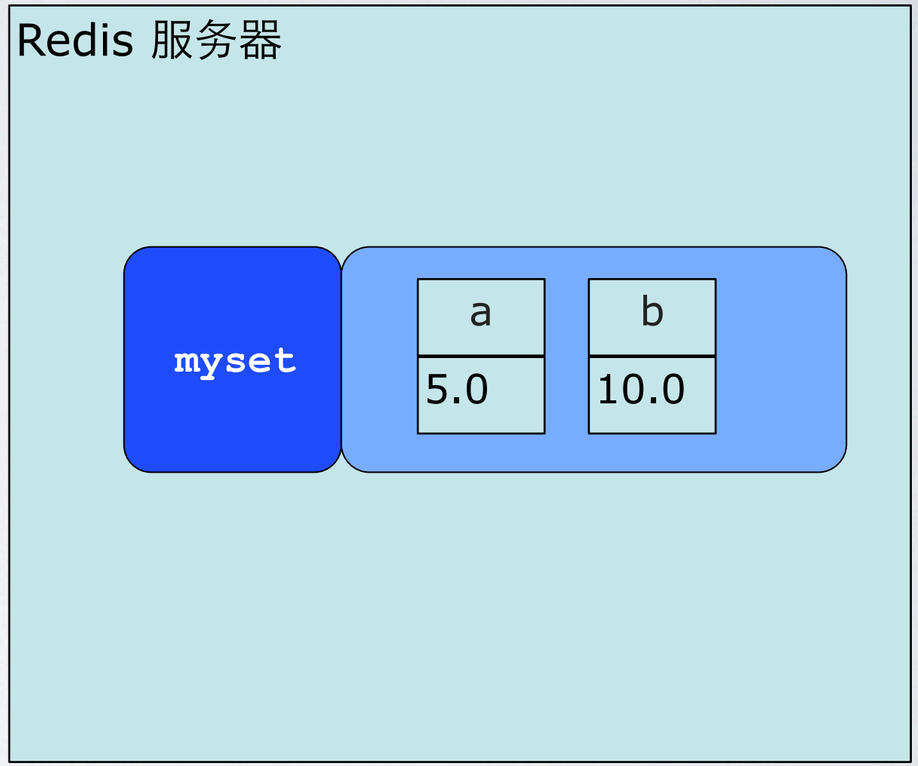
```
zadd myset 5.0 a
```





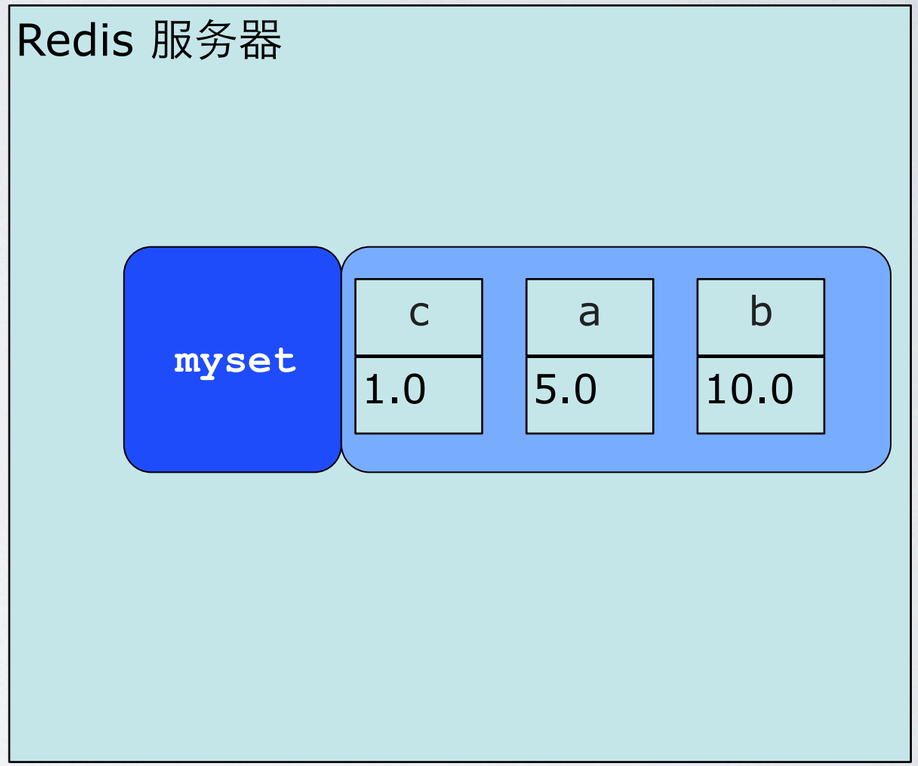
添加成员至有序集

```
zadd myset 10.0 b
```

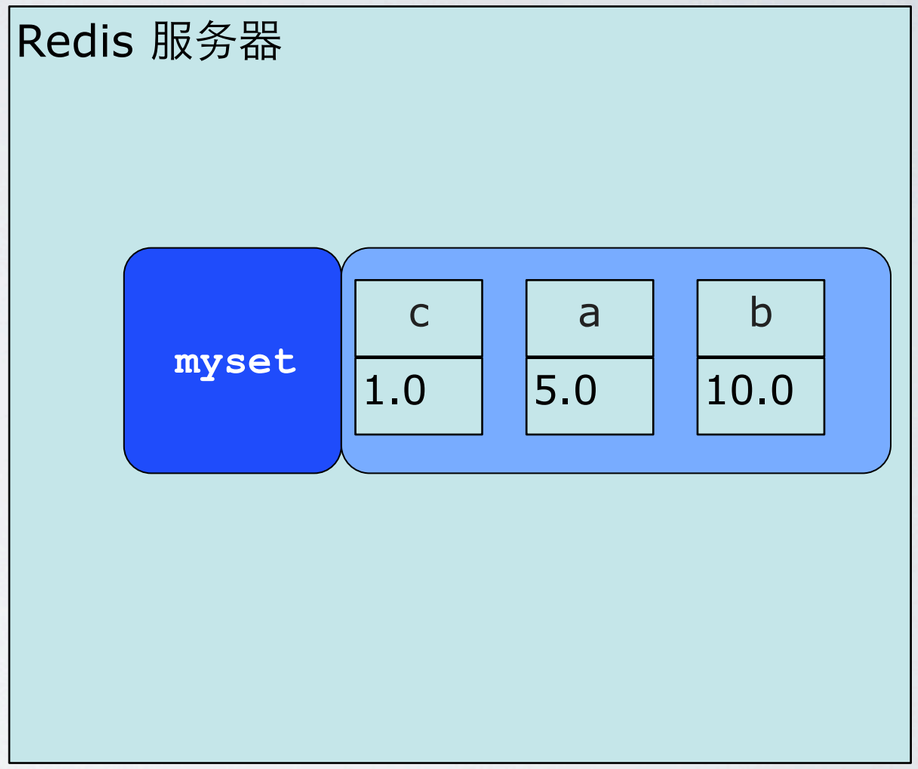


添加成员至有序集

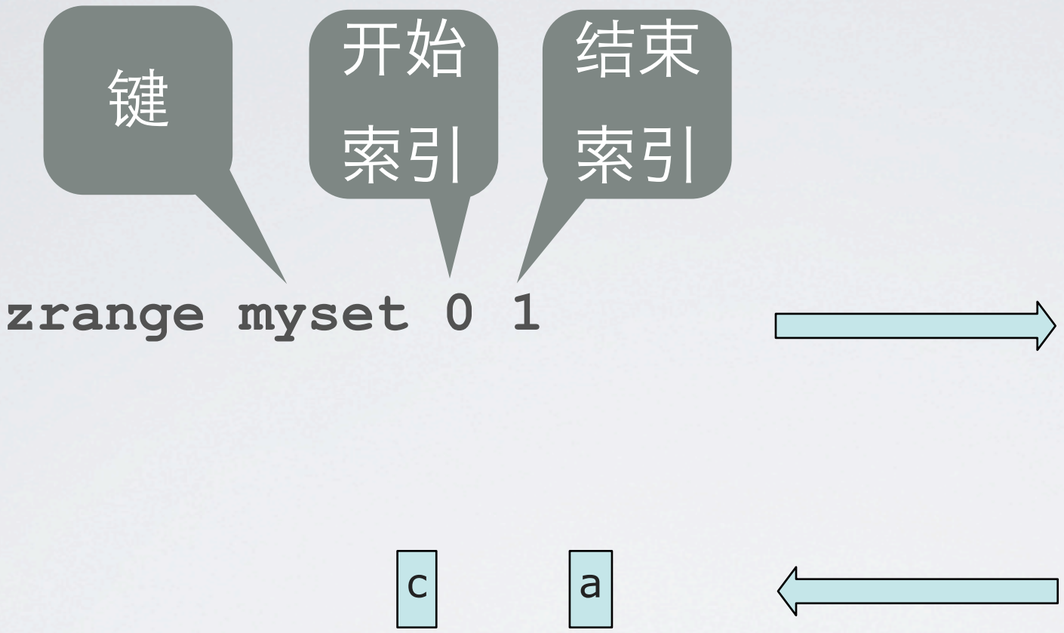
```
zadd myset 1.0 c
```



按索引范围检索成员



按索引范围检索成员



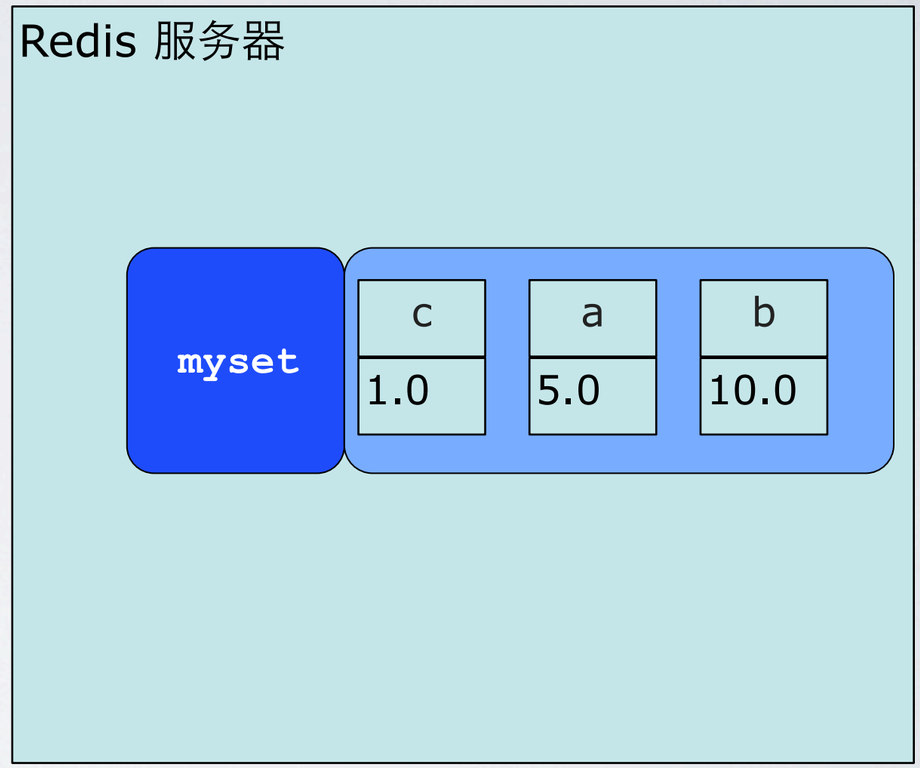
按得分检索成员

键

最小值

最大值

```
zrangebyscore myset 1 6
```



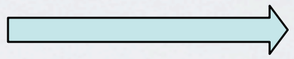
按得分检索成员

键

最小值

最大值

```
zrangebyscore myset 1 6
```



Redis 服务器

myset

c	a	b
1.0	5.0	10.0

c a

Redis 使用情形

- 替代 Memcached
  - 会话状态
  - **缓存从记录系统 (SOR) 检索的数据**
  - **复制记录系统 (SOR)，以用于性能要求高的查询**
- 处理使 RDBMS 超负荷的任务
  - 命中计数 – INCR
  - 最新的 N 项 – LPUSH 和 LTRIM
  - 随机选择一项 – SRANDMEMBER
  - 列队 – 按以下顺序列出：LPOP、RPUSH
  - ….

Redis 使用情形

- 替代 Memcached
  - 会话状态
  - **缓存从记录系统 (SOR) 检索的数据**
- **复制记录系统 (SOR)，以用于性能要求高的查询**

- 处理使 RDBMS 超负荷的任务
  - 命中计数 – INCR
  - 最新的 N 项 – LPUSH 和 LTRIM
  - 随机选择一项 – SRANDMEMBER
  - 列队 – 按以下顺序列出：LPOP、RPUSH

  ….

Redis 虽然很不错，但也有不足

Redis 虽然很不错，但也有不足

- 查询语言级别低：仅支持基于 PK 的访问

Redis 虽然很不错，但也有不足

- 查询语言级别低：仅支持基于 PK 的访问
- 事务模型有限：

Redis 虽然很不错，但也有不足

- 查询语言级别低：仅支持基于 PK 的访问
- 事务模型有限：
  - 首先读取，然后批量执行更新

Redis 虽然很不错，但也有不足

- 查询语言级别低：仅支持基于 PK 的访问
- 事务模型有限：
  - 首先读取，然后批量执行更新
  - 编写代码困难

Redis 虽然很不错，但也有不足

- 查询语言级别低：仅支持基于 PK 的访问
- 事务模型有限：
  - 首先读取，然后批量执行更新
  - 编写代码困难
- 数据必须适合内存

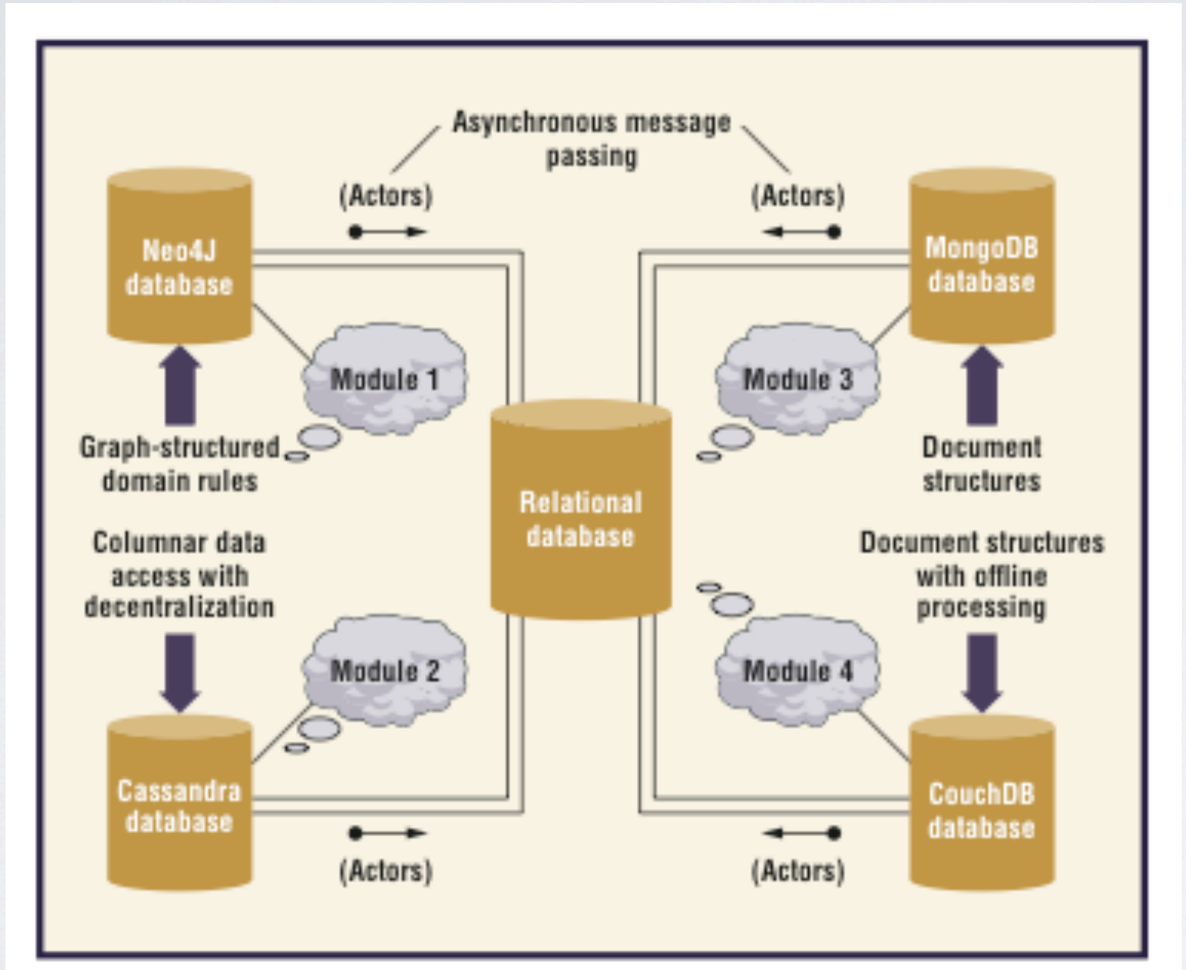
Redis 虽然很不错，但也有不足

- 查询语言级别低：仅支持基于 PK 的访问
- 事务模型有限：
  - 首先读取，然后批量执行更新
  - 编写代码困难
- 数据必须适合内存
- 单线程服务器：使用客户端分片运行多个任务

另外，别忘了：

RDBMS 适合很多应用程序

数据库的未来在于支持多语言



IEEE Software 2010 年 9 月/10 月 - Debasish Ghosh / Twitter @debasishg

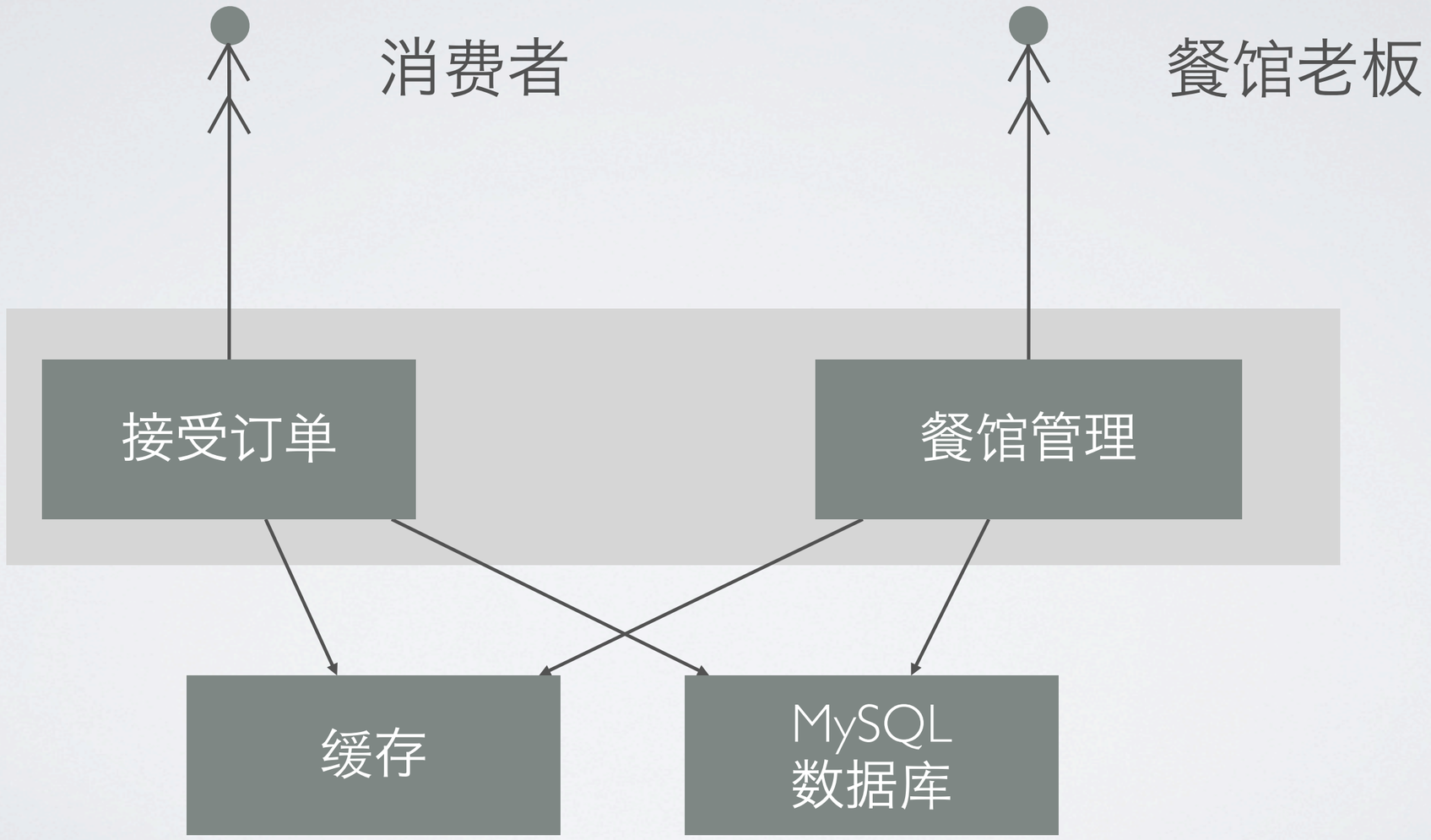
例如， Netflix

- RDBMS
- SimpleDB
- Cassandra
- Hadoop/Hbase

议题

- 为何选择多语言持久性？
- 使用 Redis 作为缓存
- 使用 Redis 具体化视图优化查询
- 同步 MySQL 和 Redis
- 跟踪对实体的更改

通过缓存增强可扩展性



选择缓存

- 情形:
  - Hibernate 二级缓存
  - 从应用程序代码显式调用
  - 缓存特性
- 缓存技术：Ehcache、 Memcached、 Infinispan ...

Redis 也是一种选择

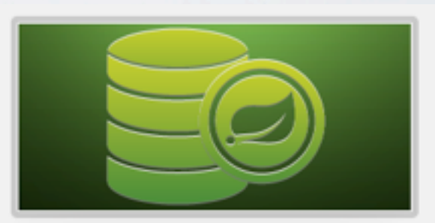
使用 Redis 作为缓存

使用 Redis 作为缓存

- Spring 3.1 缓存抽象
  - 用注释指定要缓存的方法
  - CacheManager — 可插拔后端缓存

使用 Redis 作为缓存

- Spring 3.1 缓存抽象
  - 用注释指定要缓存的方法
  - CacheManager — 可插拔后端缓存
- 适用于 Redis 的 Spring Data 技术
  - 简化 Redis 应用程序的开发
  - 提供 RedisTemplate（类似于 JdbcTemplate）
  - 提供 RedisCacheManager



使用 Spring 3.1 缓存

```
@Service
public class RestaurantManagementServiceImpl implements RestaurantManagementService {

  private final RestaurantRepository restaurantRepository;

  @Autowired
  public RestaurantManagementServiceImpl(RestaurantRepository restaurantRepository) {
    this.restaurantRepository = restaurantRepository;
  }

  @Override
  public void add(Restaurant restaurant) {
   restaurantRepository.add(restaurant);
  }

  @Override
  @Cacheable(value = "Restaurant")
  public Restaurant findById(int id) {
    return restaurantRepository.findRestaurant(id);
  }

  @Override
  @CacheEvict(value = "Restaurant", key="#restaurant.id")
  public void update(Restaurant restaurant) {
    restaurantRepository.update(restaurant);
  }
```

使用 Spring 3.1 缓存

```java
@Service
public class RestaurantManagementServiceImpl implements RestaurantManagementService {

  private final RestaurantRepository restaurantRepository;

  @Autowired
  public RestaurantManagementServiceImpl(RestaurantRepository restaurantRepository) {
    this.restaurantRepository = restaurantRepository;
  }

  @Override
  public void add(Restaurant restaurant) {
   restaurantRepository.add(restaurant);
  }

  @Override
  @Cacheable(value = "Restaurant")
  public Restaurant findById(int id) {
    return restaurantRepository.findRestaurant(id);
  }

  @Override
  @CacheEvict(value = "Restaurant", key="#restaurant.id")
  public void update(Restaurant restaurant) {
    restaurantRepository.update(restaurant);
  }
```

缓存结果

使用 Spring 3.1 缓存

```java
@Service
public class RestaurantManagementServiceImpl implements RestaurantManagementService {

  private final RestaurantRepository restaurantRepository;

  @Autowired
  public RestaurantManagementServiceImpl(RestaurantRepository restaurantRepository) {
    this.restaurantRepository = restaurantRepository;
  }

  @Override
  public void add(Restaurant restaurant) {
   restaurantRepository.add(restaurant);
  }

  @Override
  @Cacheable(value = "Restaurant")
  public Restaurant findById(int id) {
    return restaurantRepository.findRestaurant(id);
  }

  @Override
  @CacheEvict(value = "Restaurant", key="#restaurant.id")
  public void update(Restaurant restaurant) {
    restaurantRepository.update(restaurant);
  }
```

缓存结果

清除出缓存

配置 Redis 缓存管理器

```
<cache:annotation-driven />

<bean id="cacheManager"
      class="org.springframework.data.redis.cache.RedisCacheManager" >
   <constructor-arg ref="restaurantTemplate"/>
</bean>
```

配置 Redis 缓存管理器

启用缓存

```
<cache:annotation-driven />

<bean id="cacheManager"
      class="org.springframework.data.redis.cache.RedisCacheManager" >
   <constructor-arg ref="restaurantTemplate"/>
</bean>
```

配置 Redis 缓存管理器

启用缓存

```
<cache:annotation-driven />

<bean id="cacheManager"
      class="org.springframework.data.redis.cache.RedisCacheManager" >
   <constructor-arg ref="restaurantTemplate"/>
</bean>
```

指定 CacheManager 实现

配置 Redis 缓存管理器

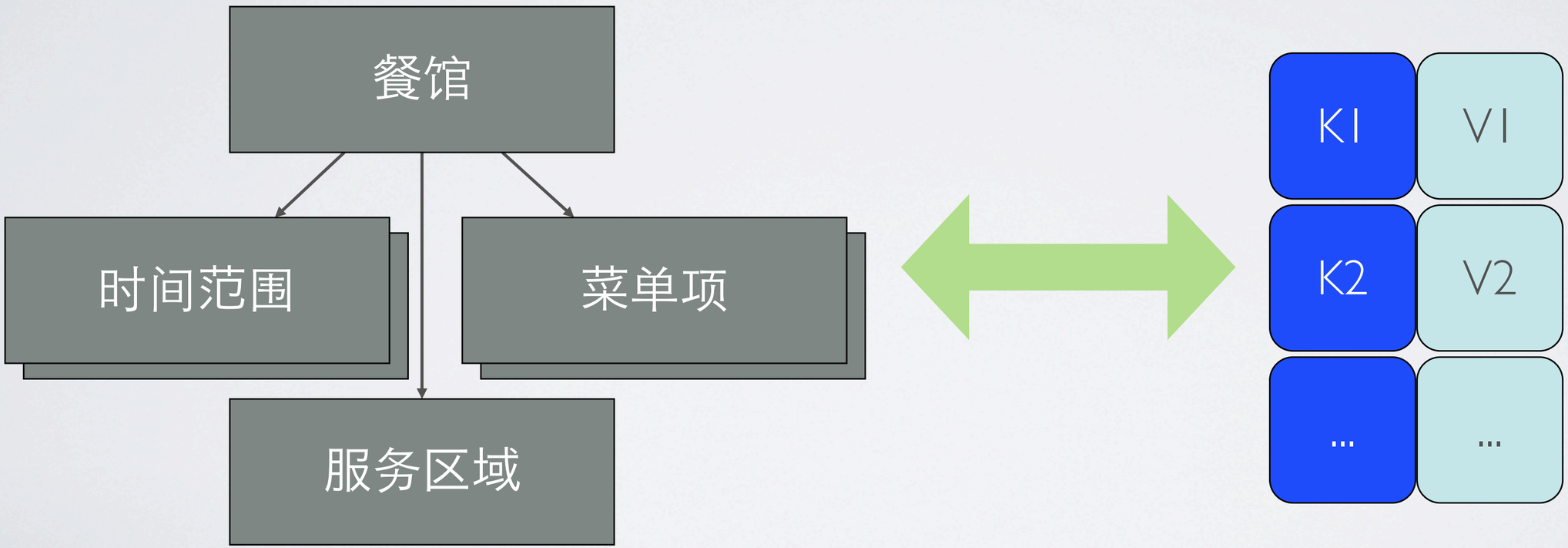
启用缓存

```
<cache:annotation-driven />

<bean id="cacheManager"
      class="org.springframework.data.redis.cache.RedisCacheManager" >
   <constructor-arg ref="restaurantTemplate"/>
</bean>
```

指定 CacheManager 实现

用来访问 Redis 的 RedisTemplate

域对象与键值映射？



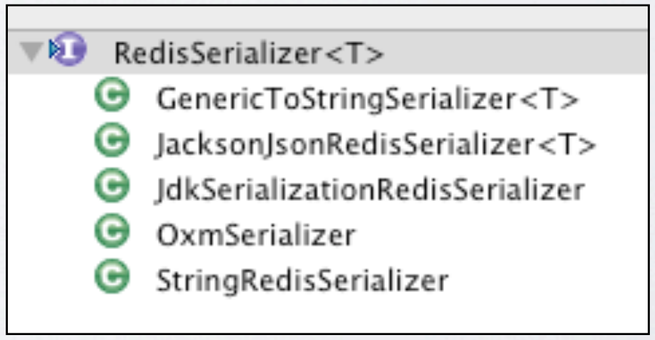
RedisTemplate

- 类似于 JdbcTemplate
- 封装样板代码，例如连接管理
- **映射 Java 对象 ⇔ Redis byte[]**

串行器：对象 ⇔ byte[]

- RedisTemplate 具有多个串行器
- 默认串行器 - 默认为 JdkSerializationRedisSerializer
- 键串行器 (KeySerializer)
- 值串行器 (ValueSerializer)
- 哈希键串行器 (HashKeySerializer)

```
public interface RedisSerializer<T> {

    * Serialize the given object to binary data.
  byte[] serialize(T t) throws SerializationException;

    * Deserialize an object from the given binary data.
  T deserialize(byte[] bytes) throws SerializationException;
}
```

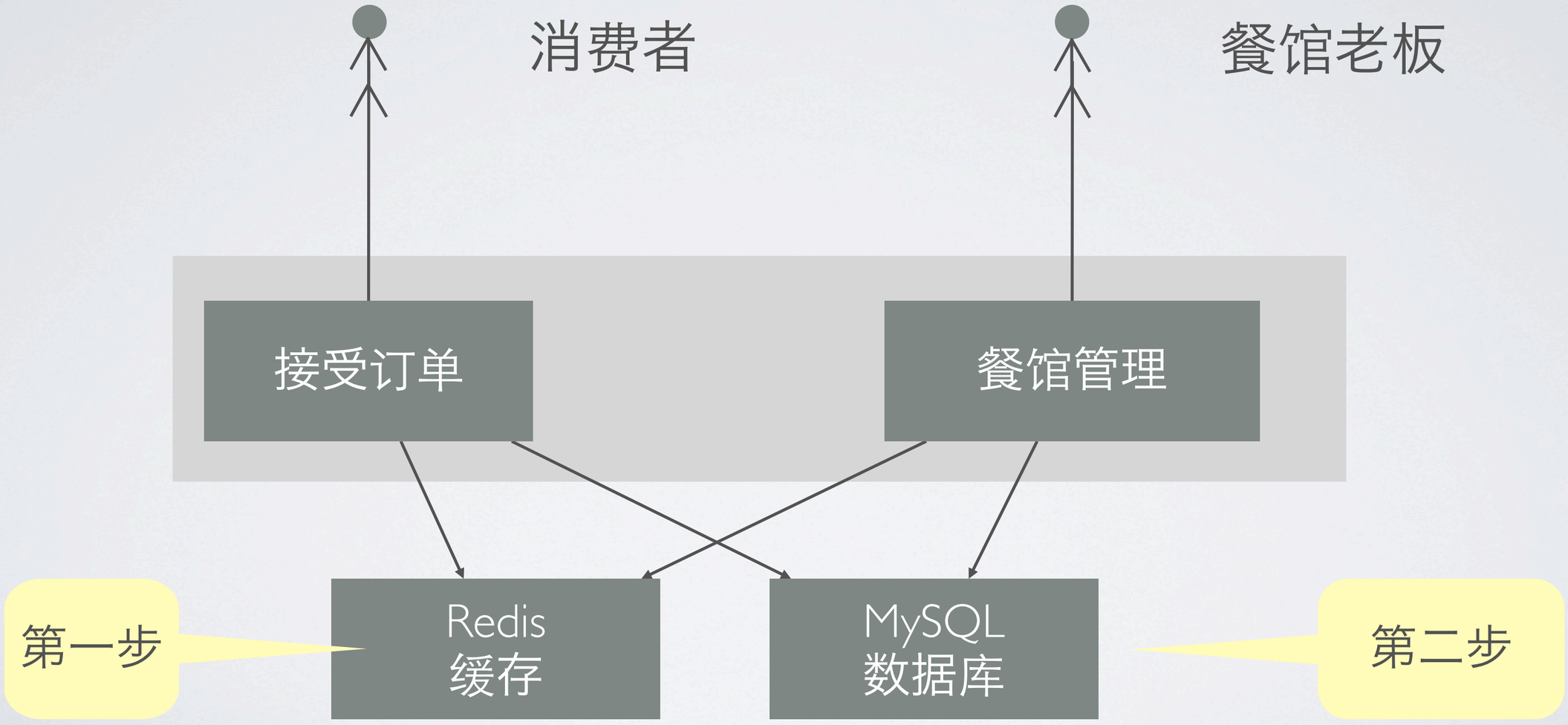


将餐馆串行化为 JSON

```
@Configuration
public class RestaurantManagementRedisConfiguration {

  @Autowired
  private RestaurantObjectMapperFactory restaurantObjectMapperFactory;

  private JacksonJsonRedisSerializer<Restaurant> makeRestaurantJsonSerializer() {
    JacksonJsonRedisSerializer<Restaurant> serializer =
       new JacksonJsonRedisSerializer<Restaurant>(Restaurant.class);
    ...
    return serializer;
  }

  @Bean
  @Qualifier("Restaurant")
  public RedisTemplate<String, Restaurant> restaurantTemplate(RedisConnectionFactory factory) {
    RedisTemplate<String, Restaurant> template = new RedisTemplate<String, Restaurant>();
    template.setConnectionFactory(factory);
    JacksonJsonRedisSerializer<Restaurant> jsonSerializer = makeRestaurantJsonSerializer();
    template.setValueSerializer(jsonSerializer);
    return template;
  }


}
```

使用 Jackson JSON 串行化餐馆

使用 Redis 进行缓存



议题

- 为何选择多语言持久性？
- 使用 Redis 作为缓存
- 使用 Redis 具体化视图优化查询
- 同步 MySQL 和 Redis
- 跟踪对实体的更改

查找可用餐馆

可用餐馆 =

提供交付地址的邮编

并且

在交付时间处于营业状态

```
public interface AvailableRestaurantRepository {

  List<AvailableRestaurant>
      findAvailableRestaurants(Address deliveryAddress,  Date deliveryTime);
  ...
}
```

外卖餐馆 – 域模型（部分）

```
class Restaurant {
  long id;
  String name;
  Set<String> serviceArea;
  Set<TimeRange> openingHours;
  List<MenuItem> menuItems;
}
```

```
class TimeRange {
  long id;
  int dayOfWeek;
  int openTime;
  int closeTime;
}
```

```
class MenuItem {
  String name;
  double price;
}
```

数据库架构

ID	Name	…
1	Ajanta	
2	Montclair Eggshop	

RESTAURANT 表

Restaurant_id	zipcode
1	94707
1	94619
2	94611
2	94619

RESTAURANT_ZIPCODE 表

RESTAURANT_TIME_RANGE 表

Restaurant_id	dayOfWeek	openTime	closeTime
1	Monday	1130	1430
1	Monday	1730	2130
2	Tuesday	1130	…

查找邮编为 94619 且在星期一早上 6 点 15 分营业的餐馆

直接三路联接

```
select r.*
from restaurant r
 inner join restaurant_time_range tr
   on r.id =tr.restaurant_id
 inner join restaurant_zipcode sa
   on r.id = sa.restaurant_id
where '94619' = sa.zip_code
 and tr.day_of_week='monday'
 and tr.openingtime <= 1815
 and 1815 <= tr.closingtime
```

如何进行规模查询？

选项一：查询缓存

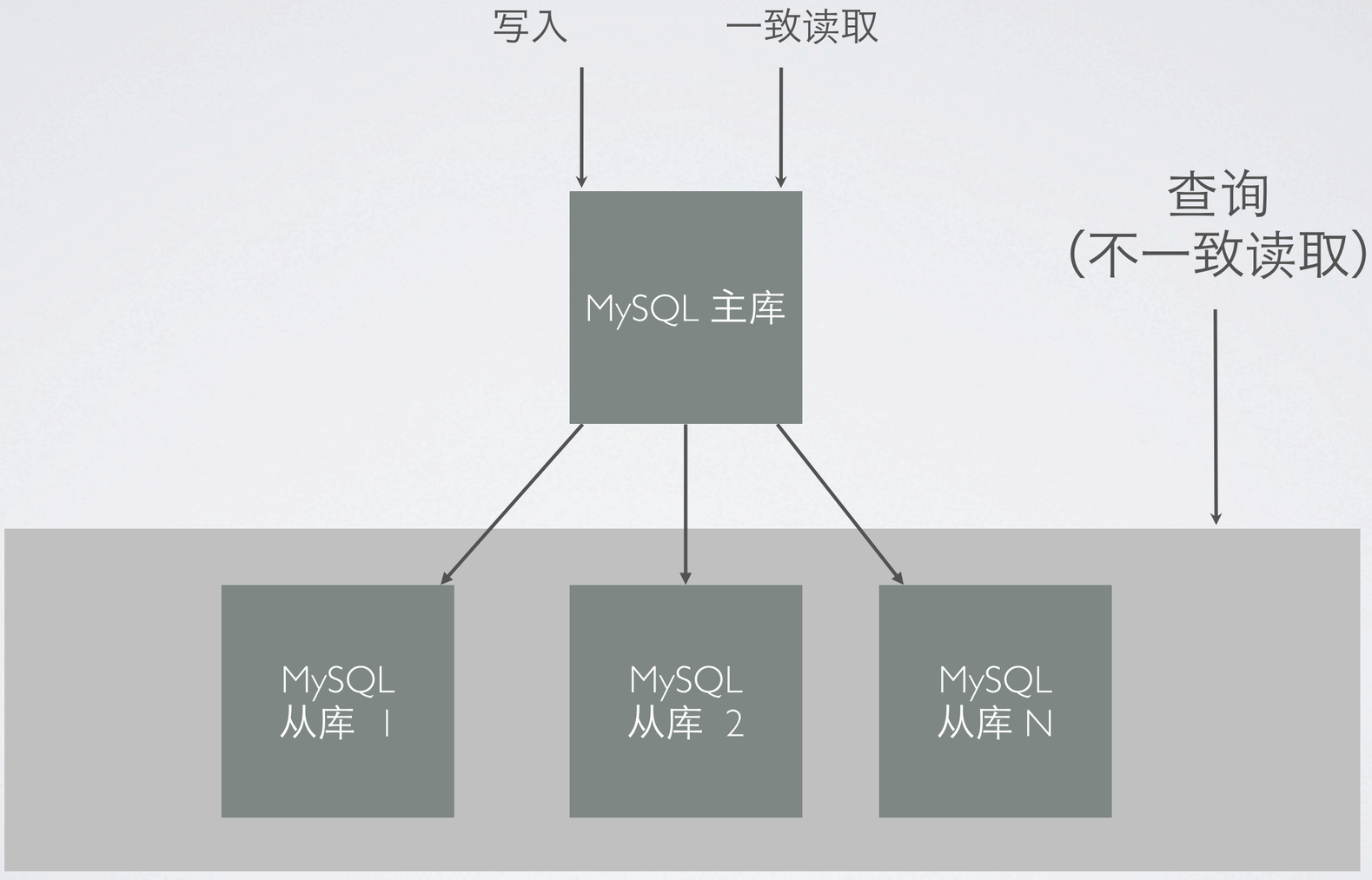
- [ZipCode, DeliveryTime] ⇒ 可用餐馆列表

但是

- 长尾查询
- 更新餐馆 ⇒ 刷新整个缓存

无效

选项二：主/从复制



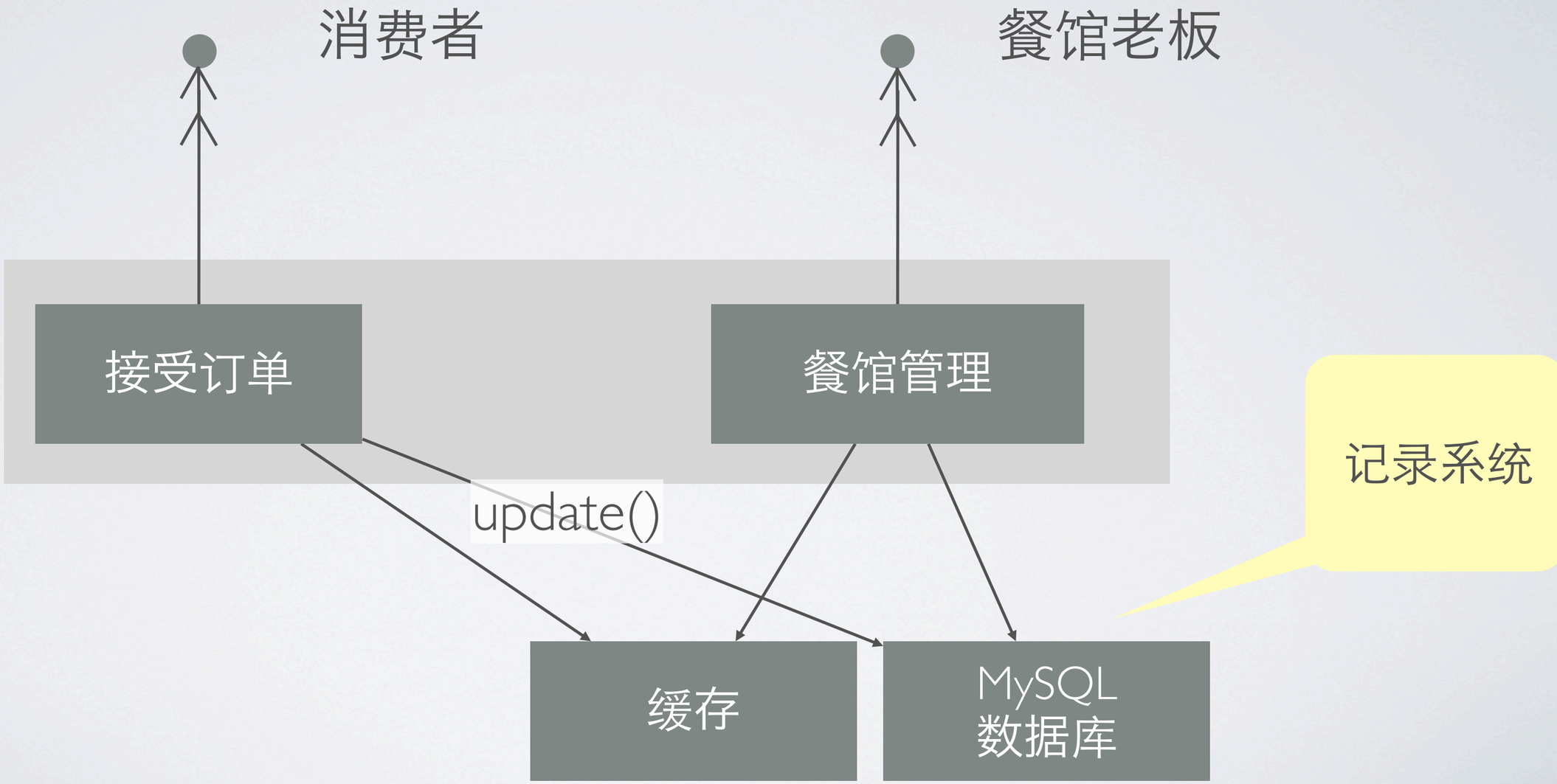
主/从复制

- 最直接

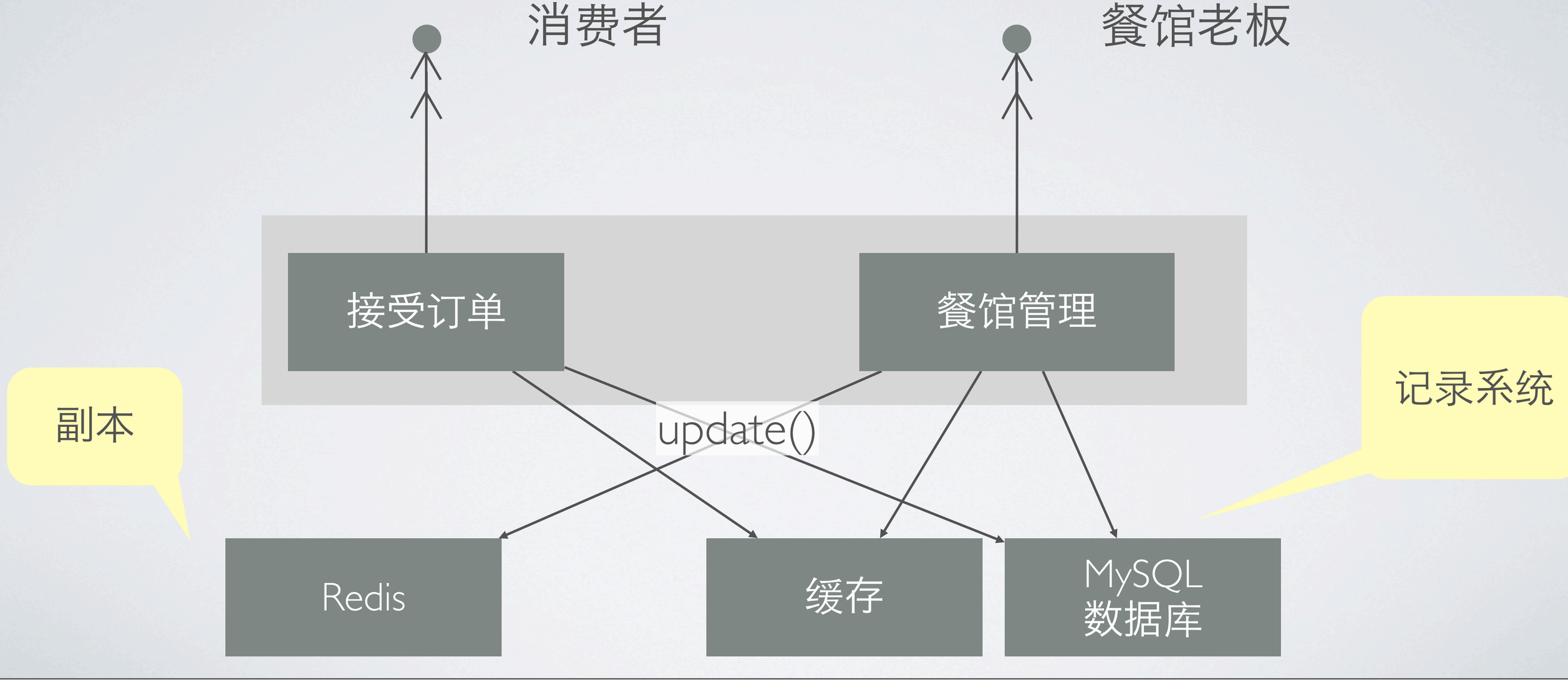
但是

- 假定 SQL 查询高效
- 从库的管理复杂
- 无法规模写入

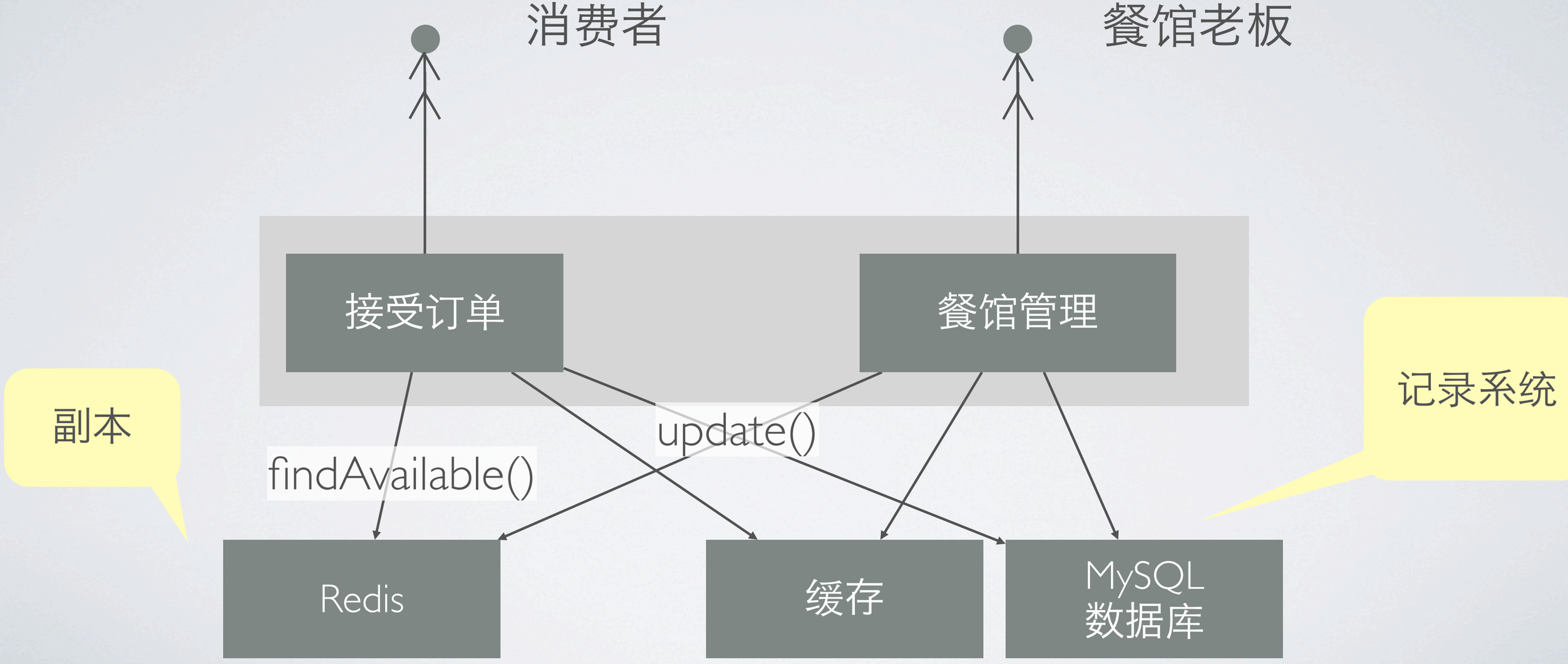
选项三：Redis 具体化视图



选项三：Redis 具体化视图

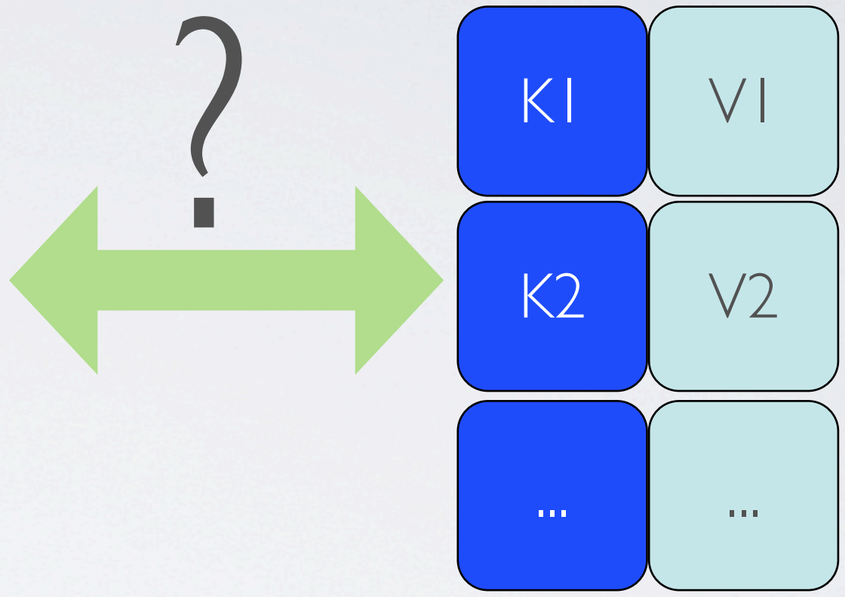


选项三：Redis 具体化视图



但如何用 Redis 实现 findAvailableRestaurants()？！

```
select r.*
from restaurant r
 inner join restaurant_time_range tr
   on r.id =tr.restaurant_id
 inner join restaurant_zipcode sa
   on r.id = sa.restaurant_id
where '94619' = sa.zip_code
 and tr.day_of_week='monday'
 and tr.openingtime <= 1815
 and 1815 <= tr.closingtime
```



我们需要让

```
ZRANGEBYSCORE myset 1 6
```

=

```
select value,score
from sorted_set
where key = 'myset'
  and score >= 1
  and score <= 6
```

sorted_set

键	值	得分

我们需要去规范化

考虑具体化视图

简化方式一：去规范化

Restaurant_id	Day_of_week	Open_time	Close_time	Zip_code
1	Monday	1130	1430	94707
1	Monday	1130	1430	94619
1	Monday	1730	2130	94707
1	Monday	1730	2130	94619
2	Monday	0700	1430	94619
…				

```
SELECT restaurant_id
 FROM time_range_zip_code
 WHERE day_of_week = ‘Monday’
     AND zip_code = 94619
     AND 1815 < close_time
     AND open_time < 1815
```

简化的查询：

- 无联接
- 2 个 = 和 2 个 <

简化方式二：应用程序筛选

```
SELECT restaurant_id, open_time
 FROM time_range_zip_code
 WHERE day_of_week = ‘Monday’
     AND zip_code = 94619
     AND 1815 < close_time
     AND open_time < 1815
```

进一步简化的查询

- 无联接
- 2 个 = 和 1 个 <

简化方式三：通过串联避免使用多个 =

Restaurant_id	Zip_dow	Open_time	Close_time
1	94707:Monday	1130	1430
1	94619:Monday	1130	1430
1	94707:Monday	1730	2130
1	94619:Monday	1730	2130
2	94619:Monday	0700	1430
…			

```
SELECT restaurant_id, open_time
 FROM time_range_zip_code
 WHERE zip_code_day_of_week = '94619:Monday'
    AND 1815 < close_time
```

键

范围

简化方式四：通过串联避免出现多个返回值

zip_dow	open_time_restaurant_id	close_time
94707:Monday	1130_1	1430
94619:Monday	1130_1	1430
94707:Monday	1730_1	2130
94619:Monday	1730_1	2130
94619:Monday	0700_2	1430
...		

```
SELECT open_time_restaurant_id,
 FROM time_range_zip_code
 WHERE zip_code_day_of_week = ‘94619:Monday’
    AND 1815 < close_time
```

简化方式四：通过串联避免出现多个返回值

zip_dow	open_time_restaurant_id	close_time
94707:Monday	1130_1	1430
94619:Monday	1130_1	1430
94707:Monday	1730_1	2130
94619:Monday	1730_1	2130
94619:Monday	0700_2	1430
...		

```
SELECT open_time_restaurant_id,
 FROM time_range_zip_code
 WHERE zip_code_day_of_week = ‘94619:Monday’
    AND 1815 < close_time
```

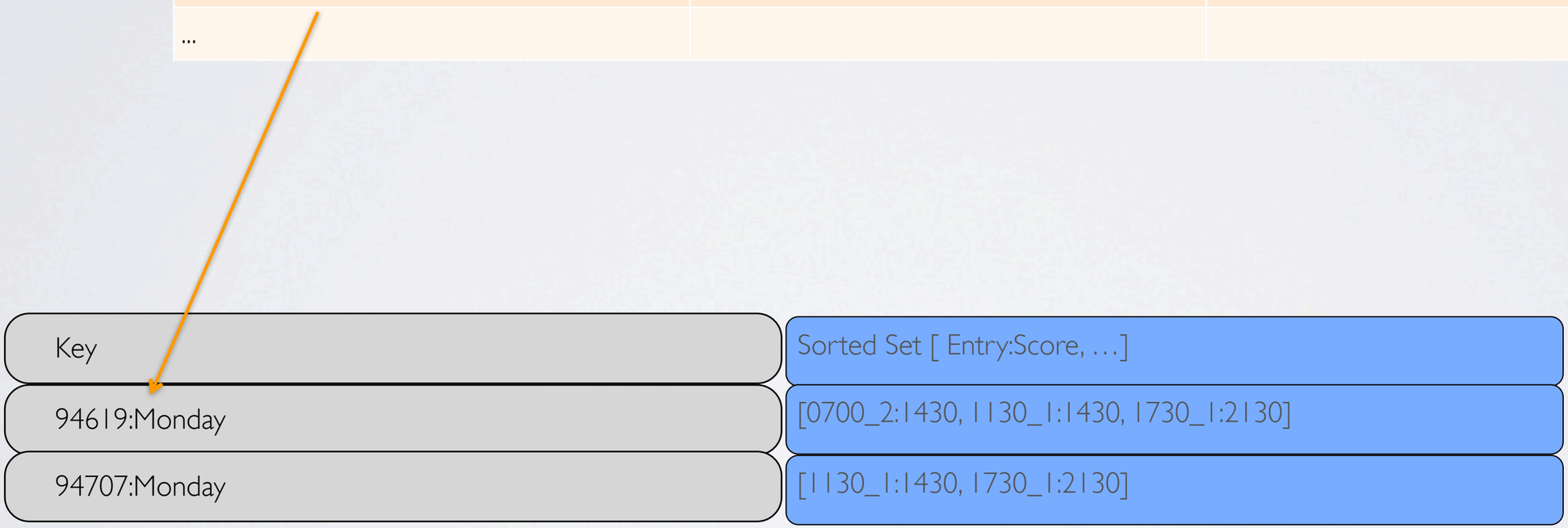
将 Redis 有序集作为索引

zip_dow	open_time_restaurant_id	close_time
94707:Monday	1130_1	1430
94619:Monday	1130_1	1430
94707:Monday	1730_1	2130
94619:Monday	1730_1	2130
94619:Monday	0700_2	1430
...		

Key	Sorted Set [Entry:Score, …]
94619:Monday	[0700_2:1430, 1130_1:1430, 1730_1:2130]
94707:Monday	[1130_1:1430, 1730_1:2130]

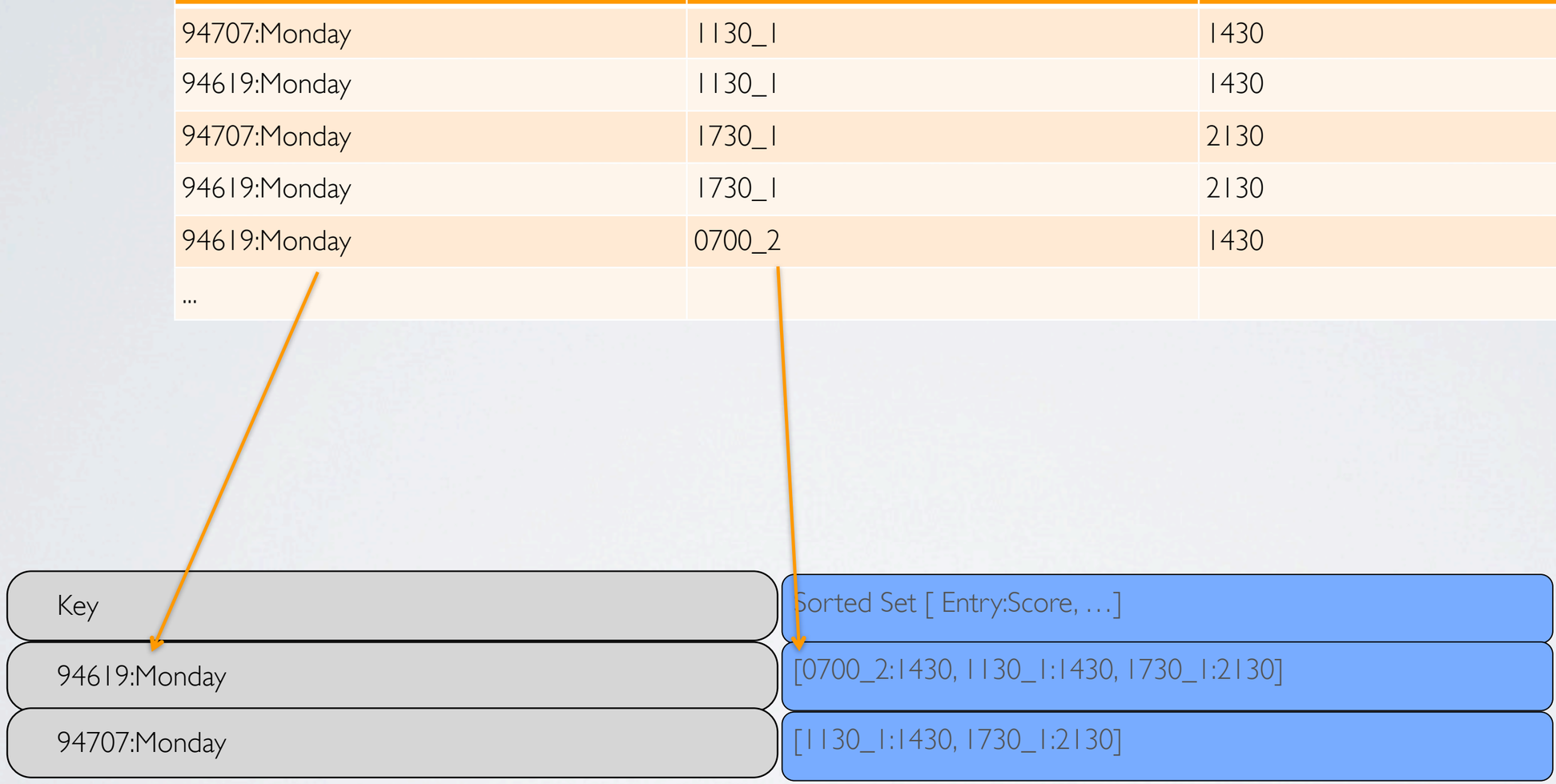
将 Redis 有序集作为索引

zip_dow	open_time_restaurant_id	close_time
94707:Monday	1130_1	1430
94619:Monday	1130_1	1430
94707:Monday	1730_1	2130
94619:Monday	1730_1	2130
94619:Monday	0700_2	1430
...		



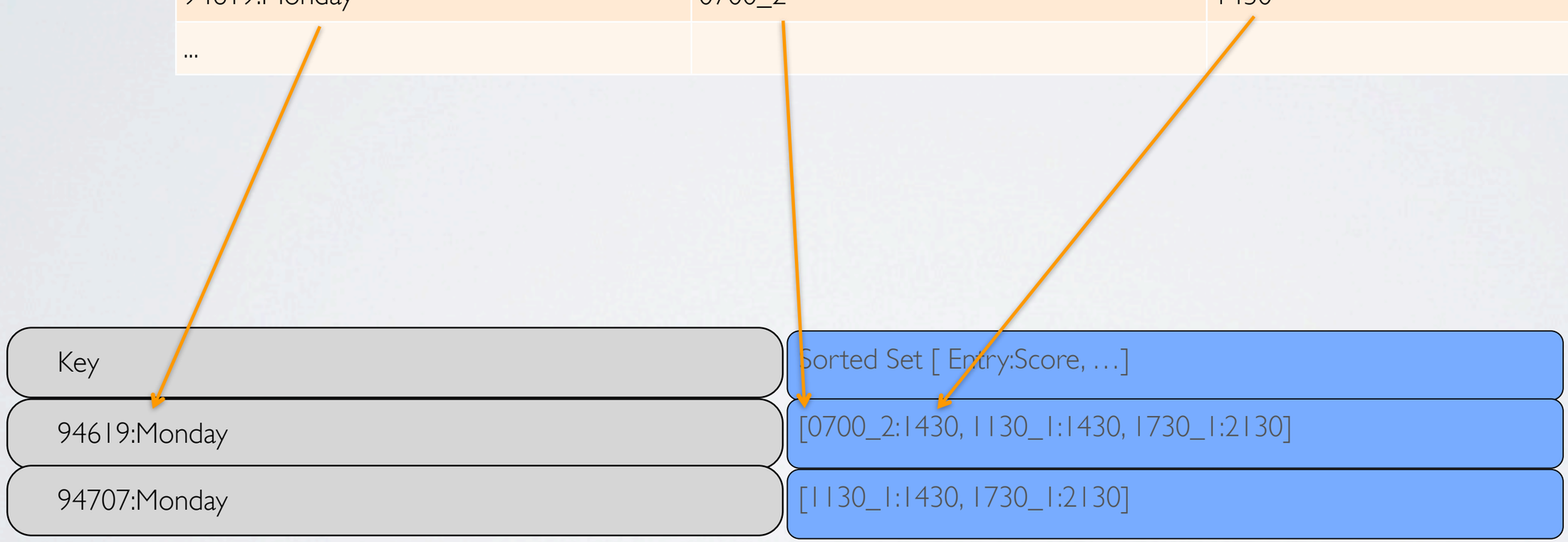
将 Redis 有序集作为索引

zip_dow	open_time_restaurant_id	close_time
94707:Monday	1130_1	1430
94619:Monday	1130_1	1430
94707:Monday	1730_1	2130
94619:Monday	1730_1	2130
94619:Monday	0700_2	1430
...		



将 Redis 有序集作为索引

zip_dow	open_time_restaurant_id	close_time
94707:Monday	1130_1	1430
94619:Monday	1130_1	1430
94707:Monday	1730_1	2130
94619:Monday	1730_1	2130
94619:Monday	0700_2	1430
...		



利用 ZRANGEBYSCORE 进行查询

Key	Sorted Set [Entry:Score, …]
94619:Monday	[0700_2:1430, 1130_1:1430, 1730_1:2130]
94707:Monday	[1130_1:1430, 1730_1:2130]

送餐地邮编和送餐日期

送餐时间

ZRANGEBYSCORE 94619:Monday 1815 2359

➔

{1730_1}

1730 在 1815 之前 ➔ Ajanta 正在营业

添加餐馆

```
@Component
public class AvailableRestaurantRepositoryImpl implements AvailableRestaurantRepository {

  @Override
  public void add(Restaurant restaurant) {
    addRestaurantDetails(restaurant);
    addAvailabilityIndexEntries(restaurant);
  }

  private void addRestaurantDetails(Restaurant restaurant) {
    restaurantTemplate.opsForValue().set(keyFormatter.key(restaurant.getId()), restaurant);
  }

  private void addAvailabilityIndexEntries(Restaurant restaurant) {
    for (TimeRange tr : restaurant.getOpeningHours()) {
      String indexValue = formatTrId(restaurant, tr);
      int dayOfWeek = tr.getDayOfWeek();
      int closingTime = tr.getClosingTime();
      for (String zipCode : restaurant.getServiceArea()) {
        redisTemplate.opsForZSet().add(closingTimesKey(zipCode, dayOfWeek), indexValue,
            closingTime);
      }
    }
  }
```

添加餐馆

```
@Component
public class AvailableRestaurantRepositoryImpl implements AvailableRestaurantRepository {

  @Override
  public void add(Restaurant restaurant) {
    addRestaurantDetails(restaurant);
    addAvailabilityIndexEntries(restaurant);
  }

  private void addRestaurantDetails(Restaurant restaurant) {
    restaurantTemplate.opsForValue().set(keyFormatter.key(restaurant.getId()), restaurant);
  }

  private void addAvailabilityIndexEntries(Restaurant restaurant) {
    for (TimeRange tr : restaurant.getOpeningHours()) {
      String indexValue = formatTrId(restaurant, tr);
      int dayOfWeek = tr.getDayOfWeek();
      int closingTime = tr.getClosingTime();
      for (String zipCode : restaurant.getServiceArea()) {
        redisTemplate.opsForZSet().add(closingTimesKey(zipCode, dayOfWeek), indexValue,
            closingTime);
      }
    }
  }
```

存储为 JSON

添加餐馆

```
@Component
public class AvailableRestaurantRepositoryImpl implements AvailableRestaurantRepository {

  @Override
  public void add(Restaurant restaurant) {
    addRestaurantDetails(restaurant);
    addAvailabilityIndexEntries(restaurant);
  }

  private void addRestaurantDetails(Restaurant restaurant) {
    restaurantTemplate.opsForValue().set(keyFormatter.key(restaurant.getId()), restaurant);
  }

  private void addAvailabilityIndexEntries(Restaurant restaurant) {
    for (TimeRange tr : restaurant.getOpeningHours()) {
      String indexValue = formatTrId(restaurant, tr);
      int dayOfWeek = tr.getDayOfWeek();
      int closingTime = tr.getClosingTime();
      for (String zipCode : restaurant.getServiceArea()) {
        redisTemplate.opsForZSet().add(closingTimesKey(zipCode, dayOfWeek), indexValue,
           closingTime);
      }
    }
  }
}
```

存储为 JSON

键

成员

得分

查找可用餐馆

```
@Component
public class AvailableRestaurantRepositoryImpl implements AvailableRestaurantRepository {
  @Override
  public List<AvailableRestaurant>
findAvailableRestaurants(Address deliveryAddress, Date deliveryTime) {
    String zipCode = deliveryAddress.getZip();
    int dayOfWeek = DateTimeUtil.dayOfWeek(deliveryTime);
    int timeOfDay = DateTimeUtil.timeOfDay(deliveryTime);
    String closingTimesKey = closingTimesKey(zipCode, dayOfWeek);

    Set<String> trsClosingAfter =
                redisTemplate.opsForZSet().rangeByScore(closingTimesKey, timeOfDay, 2359);

    Set<String> restaurantIds = new HashSet<String>();
    for (String tr : trsClosingAfter) {
      String[] values = tr.split("_");
      if (Integer.parseInt(values[0]) <= timeOfDay)
        restaurantIds.add(values[1]);
    }
    Collection<String> keys = keyFormatter.keys(restaurantIds);
    return availableRestaurantTemplate.opsForValue().multiGet(keys);
}
```

查找可用餐馆

```
@Component
public class AvailableRestaurantRepositoryImpl implements AvailableRestaurantRepository {
  @Override
  public List<AvailableRestaurant>
findAvailableRestaurants(Address deliveryAddress, Date deliveryTime) {
    String zipCode = deliveryAddress.getZip();
    int dayOfWeek = DateTimeUtil.dayOfWeek(deliveryTime);
    int timeOfDay = DateTimeUtil.timeOfDay(deliveryTime);
    String closingTimesKey = closingTimesKey(zipCode, dayOfWeek);

  Set<String> trsClosingAfter =
              redisTemplate.opsForZSet().rangeByScore(closingTimesKey, timeOfDay, 2359);

  Set<String> restaurantIds = new HashSet<String>();
  for (String tr : trsClosingAfter) {
    String[] values = tr.split("_");
    if (Integer.parseInt(values[0]) <= timeOfDay)
      restaurantIds.add(values[1]);
  }
  Collection<String> keys = keyFormatter.keys(restaurantIds);
  return availableRestaurantTemplate.opsForValue().multiGet(keys);
}
```

查找在以下时间后
停止营业的餐馆

查找可用餐馆

```
@Component
public class AvailableRestaurantRepositoryImpl implements AvailableRestaurantRepository {
  @Override
  public List<AvailableRestaurant>
findAvailableRestaurants(Address deliveryAddress, Date deliveryTime) {
    String zipCode = deliveryAddress.getZip();
    int dayOfWeek = DateTimeUtil.dayOfWeek(deliveryTime);
    int timeOfDay = DateTimeUtil.timeOfDay(deliveryTime);
    String closingTimesKey = closingTimesKey(zipCode, dayOfWeek);

  Set<String> trsClosingAfter =
              redisTemplate.opsForZSet().rangeByScore(closingTimesKey, timeOfDay, 2359);

  Set<String> restaurantIds = new HashSet<String>();
  for (String tr : trsClosingAfter) {
    String[] values = tr.split("_");
    if (Integer.parseInt(values[0]) <= timeOfDay)
      restaurantIds.add(values[1]);
  }
  Collection<String> keys = keyFormatter.keys(restaurantIds);
  return availableRestaurantTemplate.opsForValue().multiGet(keys);
}
```

查找在以下时间后停止营业的餐馆

筛选出在以下时间后开始营业的餐馆

查找可用餐馆

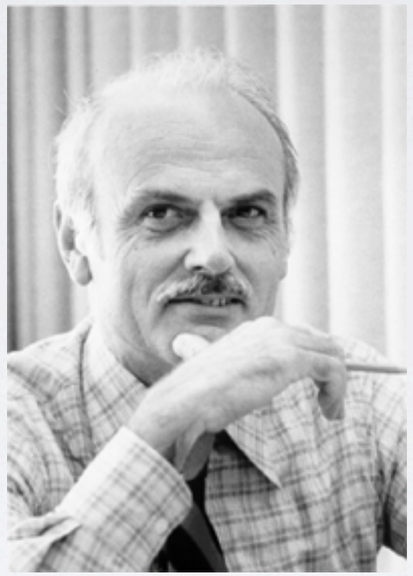
```
@Component
public class AvailableRestaurantRepositoryImpl implements AvailableRestaurantRepository {
  @Override
  public List<AvailableRestaurant>
findAvailableRestaurants(Address deliveryAddress, Date deliveryTime) {
    String zipCode = deliveryAddress.getZip();
    int dayOfWeek = DateTimeUtil.dayOfWeek(deliveryTime);
    int timeOfDay = DateTimeUtil.timeOfDay(deliveryTime);
    String closingTimesKey = closingTimesKey(zipCode, dayOfWeek);

  Set<String> trsClosingAfter =
              redisTemplate.opsForZSet().rangeByScore(closingTimesKey, timeOfDay, 2359);

  Set<String> restaurantIds = new HashSet<String>();
  for (String tr : trsClosingAfter) {
    String[] values = tr.split("_");
    if (Integer.parseInt(values[0]) <= timeOfDay)
      restaurantIds.add(values[1]);
  }
  Collection<String> keys = keyFormatter.keys(restaurantIds);
  return availableRestaurantTemplate.opsForValue().multiGet(keys);
}
```

查找在以下时间后停止营业的餐馆

筛选出在以下时间后开始营业的餐馆

检索正在营业的餐馆

抱歉，Ted！



http://en.wikipedia.org/wiki/Edgar_F._Codd

议题

- 为何选择多语言持久性？
- 使用 Redis 作为缓存
- 使用 Redis 具体化视图优化查询
- 同步 MySQL 和 Redis
- 跟踪对实体的更改

MySQL 和 Redis

需要保持一致

不能采用两阶段提交方式

- Redis 不支持这种方式
- 即使支持，也最好避免采用两阶段提交 (2PC)

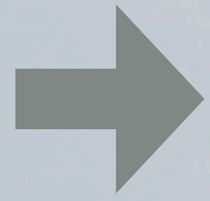
http://www.infoq.com/articles/ebay-scalability-best-practices

Atomic **原子性**

Consistent **一致性**

Isolated **隔离性**

Durable **持久性**



Basically **A**vailable **基本上可用**

Soft state **软状态**

Eventually consistent **最终一致**

BASE: An Acid Alternative http://queue.acm.org/detail.cfm?id=1394128

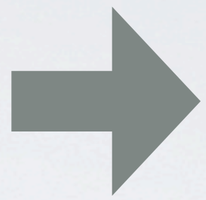
更新 Redis #FAIL

开始 MySQL 事务

更新 MySQL

更新 Redis

回滚 MySQL 事务



Redis 已更新

MySQL 未更新

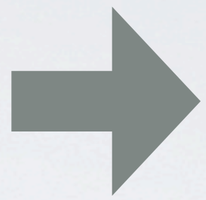
更新 Redis #FAIL

开始 MySQL 事务

更新 MySQL

更新 Redis

回滚 MySQL 事务



Redis 已更新

MySQL 未更新

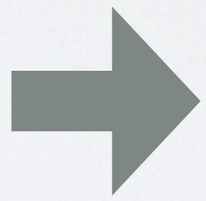
开始 MySQL 事务

更新 MySQL

提交 MySQL 事务

<<系统崩溃>>

更新 Redis



MySQL 已更新

Redis 未更新

可靠地更新 Redis

第 1 步（共 2 步）

开始 MySQL 事务

更新 MySQL

在 MySQL 中将 CRUD 事件列队

提交事务

ACID

可靠地更新 Redis

第 1 步（共 2 步）

开始 MySQL 事务

更新 MySQL

在 MySQL 中将 CRUD 事件列队

提交事务

ACID

事件 ID

操作：创建、更新、删除、新建实体状态， 例如 JSON

可靠地更新 Redis

第 2 步（共 2 步）

对于 MySQL 队列中的每个 CRUD 事件

可靠地更新 Redis

第 2 步（共 2 步）

对于 MySQL 队列中的每个 CRUD 事件

从 MySQL 队列中获取下一个 CRUD 事件

可靠地更新 Redis

第 2 步（共 2 步）

对于 MySQL 队列中的每个 CRUD 事件

从 MySQL 队列中获取下一个 CRUD 事件

如果 CRUD 事件不完全相同

则更新 Redis（包括事件 ID）

可靠地更新 Redis

第 2 步（共 2 步）

对于 MySQL 队列中的每个 CRUD 事件

从 MySQL 队列中获取下一个 CRUD 事件

如果 CRUD 事件不完全相同

则更新 Redis（包括事件 ID）

开始 MySQL 事务

将 CRUD 事件标记为已处理

提交事务

ENTITY_CRUD_EVENT

ID	JSON	已处理？

第 1 步

EntityCrudEvent
资源库

插入到 ...

ENTITY_CRUD_EVENT

ID	JSON	已处理？

第 1 步

第 2 步

计时器

EntityCrudEvent
资源库

EntityCrudEvent
处理器

插入到 ...

从 ... 中选择 ...

ENTITY_CRUD_EVENT

ID	JSON	已处理？

第 1 步

第 2 步

计时器

EntityCrudEvent
资源库

EntityCrudEvent
处理器

应用（事件）

Redis
更新程序

插入到 ...

从 ... 中选择 ...

ENTITY_CRUD_EVENT

ID	JSON	已处理？

第 1 步

第 2 步

计时器

EntityCrudEvent
资源库

EntityCrudEvent
处理器

应用（事件）

Redis
更新程序

插入到 ...

从 ... 中选择 ...

ENTITY_CRUD_EVENT

ID	JSON	已处理？

Redis

更新 Redis

乐观锁定

更新 Redis

```
WATCH restaurant:lastSeenEventId:«restaurantId»
```

更新 Redis

乐观锁定

```
WATCH restaurant:lastSeenEventId:«restaurantId»

lastSeenEventId = GET restaurant:lastSeenEventId:«restaurantId»

if (lastSeenEventId >= eventId) return;
```

重复检测

更新 Redis

乐观锁定

```
WATCH restaurant:lastSeenEventId:«restaurantId»

lastSeenEventId = GET restaurant:lastSeenEventId:«restaurantId»

if (lastSeenEventId >= eventId) return;

MULTI
   SET restaurant:lastSeenEventId:«restaurantId» eventId

   ... update the restaurant data...

EXEC
```

重复检测

事务

议题

- 为何选择多语言持久性？
- 使用 Redis 作为缓存
- 使用 Redis 具体化视图优化查询
- 同步 MySQL 和 Redis
- 跟踪对实体的更改

我们如何生成 CRUD 事件？

更改跟踪选项

更改跟踪选项

- 显式代码

更改跟踪选项

- 显式代码
- Hibernate 事件侦听器

更改跟踪选项

- 显式代码
- Hibernate 事件侦听器
- 服务层特性

更改跟踪选项

- 显式代码
- Hibernate 事件侦听器
- 服务层特性
- CQRS/事件源

HibernateEvent
侦听器

EntityCrudEvent
资源库

ENTITY_CRUD_EVENT

ID	JSON	已处理？

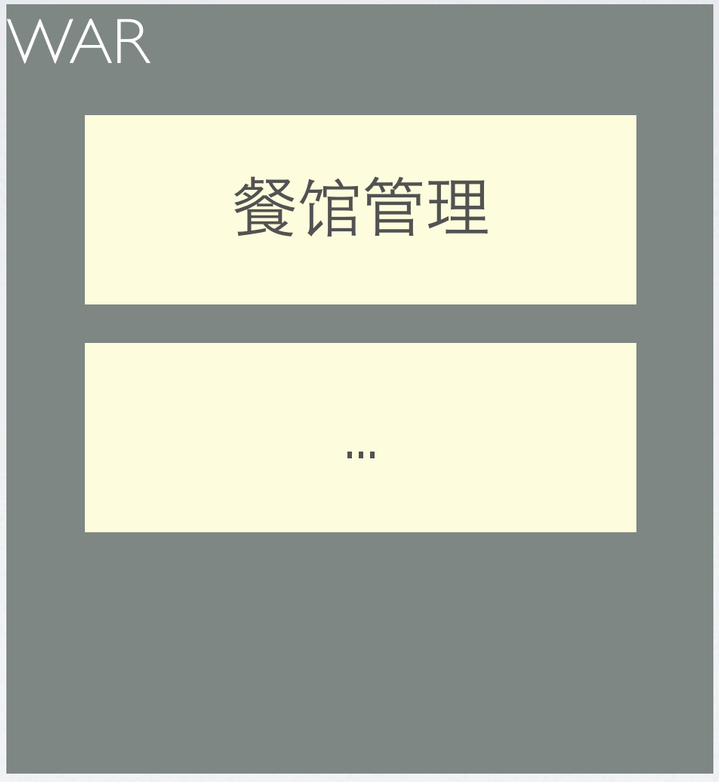
Hibernate 事件侦听器

```
public class ChangeTrackingListener
  implements PostInsertEventListener, PostDeleteEventListener, PostUpdateEventListener {

  @Autowired
  private EntityCrudEventRepository entityCrudEventRepository;

  private void maybeTrackChange(Object entity, EntityCrudEventType eventType) {
    if (isTrackedEntity(entity)) {
      entityCrudEventRepository.add(new EntityCrudEvent(eventType, entity));
    }
  }

  @Override
  public void onPostInsert(PostInsertEvent event) {
    Object entity = event.getEntity();
    maybeTrackChange(entity, EntityCrudEventType.CREATE);
  }

  @Override
  public void onPostUpdate(PostUpdateEvent event) {
    Object entity = event.getEntity();
    maybeTrackChange(entity, EntityCrudEventType.UPDATE);
  }

  @Override
  public void onPostDelete(PostDeleteEvent event) {
    Object entity = event.getEntity();
    maybeTrackChange(entity, EntityCrudEventType.DELETE);
  }
  }
```

议题

- 为何选择多语言持久性？
- 使用 Redis 作为缓存
- 使用 Redis 具体化视图优化查询
- 同步 MySQL 和 Redis
- 跟踪对实体的更改

原始体系结构



这种整体式体系结构的缺点

- 不利于高频次部署
- 超负荷 IDE 和 Web 容器
- 不利于扩展开发
- 技术锁定

需要模块化程度更高的体系结构

使用消息代理

使用消息代理

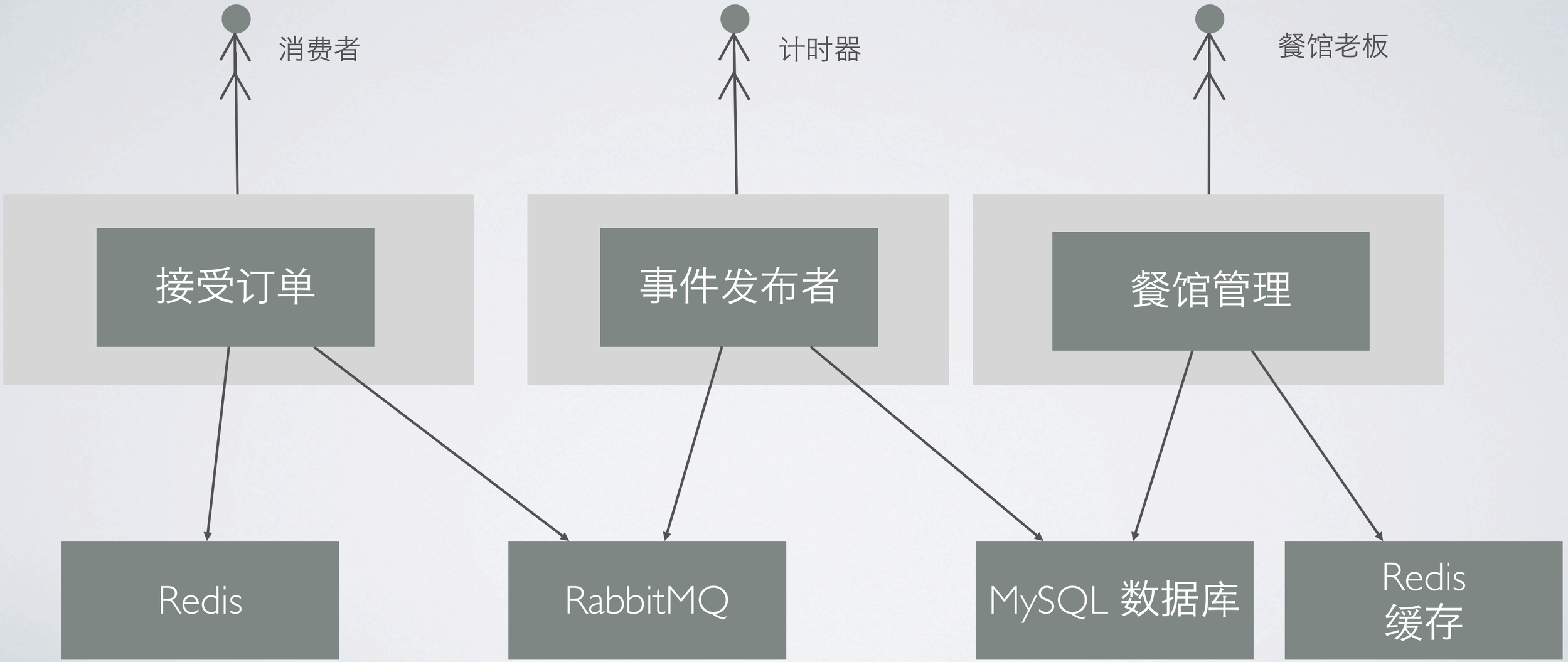
首选异步

使用消息代理

首选异步

虽然 JSON 很流行，但二进制格式更高效

模块化体系结构


消费者
计时器
餐馆老板
接受订单
事件发布者
餐馆管理
Redis
RabbitMQ
MySQL 数据库
Redis
缓存

模块化异步体系结构的优点

模块化异步体系结构的优点

- 扩展开发：独立地开发、部署和扩展每项服务

模块化异步体系结构的优点

- 扩展开发：独立地开发、部署和扩展每项服务
- 频繁/独立重新部署 UI

模块化异步体系结构的优点

- 扩展开发：独立地开发、部署和扩展每项服务
- 频繁/独立重新部署 UI
- 提高故障隔离能力

模块化异步体系结构的优点

- 扩展开发：独立地开发、部署和扩展每项服务
- 频繁/独立重新部署 UI
- 提高故障隔离能力
- 避免长期依赖单一技术

模块化异步体系结构的优点

- 扩展开发：独立地开发、部署和扩展每项服务
- 频繁/独立重新部署 UI
- 提高故障隔离能力
- 避免长期依赖单一技术
- 消息代理使生产者和使用者互不相干

第 2 步（共 2 步）

对于 MySQL 队列中的每个 CRUD 事件

从 MySQL 队列中获取下一个 CRUD 事件

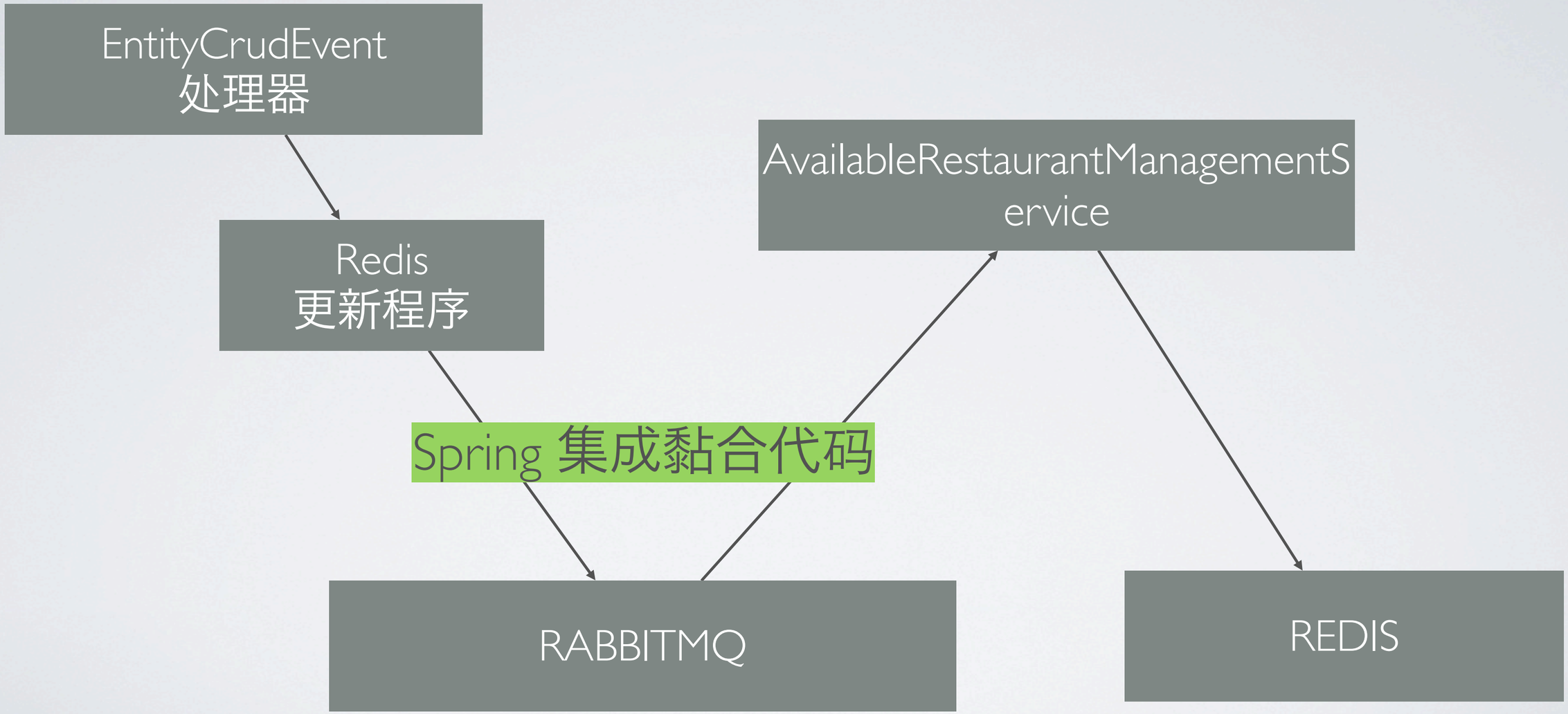
将持久性消息发布到 RabbitMQ

开始 MySQL 事务

将 CRUD 事件标记为已处理

提交事务

消息流


EntityCrudEvent
处理器
Redis
更新程序
AvailableRestaurantManagementS
ervice
Spring 集成黏合代码
RABBITMQ
REDIS

RedisUpdater ⇒ AMQP

```
<beans>

   <int:gateway id="redisUpdaterGateway"
      service-interface="net...RedisUpdater"
      default-request-channel="eventChannel"
      />

   <int:channel id="eventChannel"/>


   <int:object-to-json-transformer
input-channel="eventChannel" output-channel="amqpOut"/>

   <int:channel id="amqpOut"/>

   <amqp:outbound-channel-adapter
      channel="amqpOut"
      amqp-template="rabbitTemplate"
      routing-key="crudEvents"
      exchange-name="crudEvents"
    />

</beans>
```

创建代理

AMQP ⇒ 可用...服务

```
<beans>
   <amqp:inbound-channel-adapter
      channel="inboundJsonEventsChannel"
      connection-factory="rabbitConnectionFactory"
      queue-names="crudEvents"/>

   <int:channel id="inboundJsonEventsChannel"/>

   <int:json-to-object-transformer
      input-channel="inboundJsonEventsChannel"
      type="net.chrisrichardson.foodToGo.common.JsonEntityCrudEvent"
      output-channel="inboundEventsChannel"/>

   <int:channel id="inboundEventsChannel"/>

   <int:service-activator
      input-channel="inboundEventsChannel"
      ref="availableRestaurantManagementServiceImpl"
      method="processEvent"/>
</beans>
```

调用服务

总结

总结

- 每个 SQL/NoSQL 数据库都是一系列折衷的结果

总结

- 每个 SQL/NoSQL 数据库都是一系列折衷的结果
- 多语言持久性：利用 SQL 和 NoSQL 数据库的优势

总结

- 每个 SQL/NoSQL 数据库都是一系列折衷的结果
- 多语言持久性：利用 SQL 和 NoSQL 数据库的优势
- 使用 Redis 作为分布式缓存

总结

- 每个 SQL/NoSQL 数据库都是一系列折衷的结果
- 多语言持久性：利用 SQL 和 NoSQL 数据库的优势
- 使用 Redis 作为分布式缓存
- 将非规范化数据存储在 Redis 中以便快速查询

总结

- 每个 SQL/NoSQL 数据库都是一系列折衷的结果
- 多语言持久性：利用 SQL 和 NoSQL 数据库的优势
- 使用 Redis 作为分布式缓存
- 将非规范化数据存储在 Redis 中以便快速查询
- 需要可靠的数据库同步

@crichardson crichardson@vmware.com http://plainoldobjects.com

问题？

注册 CloudFoundry.com

Cloud Foundry 启动营

在www.cloudfoundry.com注册账号并成功上传应用程序,

即可于12月8日中午后凭账号ID和应用URL到签到处换取Cloud Foundry主题卫衣一件。

iPhone5 等你拿

第二天大会结束前， 请不要提前离开， 将填写完整的意见反馈表投到签到处的抽奖箱内，
即可参与“iPhone5”抽奖活动。

Birds of a Feather 专家面对面

所有讲师都会在课程结束后，到紫兰厅与来宾讨论课程上的问题